Bibliothèques de Développement

Multimédia - Projet

Casse briques

DU Pingjie / YANG Di

Tutor : Christophe Ducottet

télécom saint-étienne
école d'ingénieurs / nouvelles technologies

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

Spécifications

Pour ce projet, nous avons, selon les demandes, développé une application permettant de jouer à Casse Briques en C++ sous l'environnement QtCreator avec les bibliothèques Qt, OpenGL et OpenCV. Nous avons réalisé une interface utilisateur conviviale et mis en place une interaction avec l'utilisateur à partir de l'acquisition et du traitement des images issues d'une WebCam.

Présentation de l'interface utilisateur :

Au commencement du fonctionnement, l'application affiche l'interface utilisateur qui se compose de **4 paties**. (*figure 1* ci-dessous)

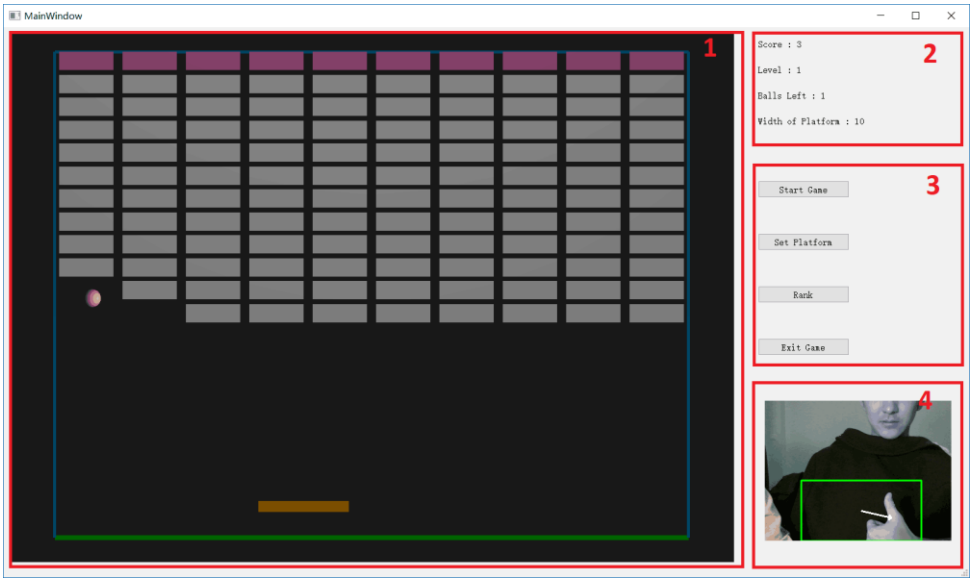


figure 1

- Un **GLWidget** visualisant d'une scène 3D dynamique pour afficher le jeu (Le jeu se déroule en 2 dimensions, même si les briques, le palet, la boule et les murs ont une représentation 3D.)
- 4 **labels** affichant les points obtenus par le joueur, le niveau actuel, le nombre des boules restant et la largeur du palet choisi par l'utilisateur.
- 4 **bouttons** pour réaliser le menu de ce jeu.
- **WebCam** pour déplacer le palet.

Principales interactions avec l'utilisateur :

- Si l'utilisateur appuie sur "Start Game", le système affichera automatiquement une boîte de dialogue pour initialiser la largeur du palet. Ensuite, l'utilisateur peut appuyer sur "H" du clavier pour servir. Si l'utilisateur veut modifier à nouveau la largeur du palet, il peut appuyer sur "Set Platform". Lorsque la dernière boule est détruite la partie est terminée en affichant une boîte de dialogue pour saisir le nom de joueur. Ensuite, le système enregistre le nom et les points et réinitialiser les informations affichées par les labels. L'utilisateur peut appuyer sur "H" afin de commencer une nouvelle partie ou voie le classement par le bouton "Rank".

- L'utilisateur contrôle le déplacement du palet par des gestes de balayage devant la zone active de la caméra. S'il fait un balayage horizontal (droite ou gauche) par la main avec le pouce élevé, le palet se déplace dans la bonne direction, à une vitesse proportionnelle à la vitesse du balayage et le palet s'arrête par les autres gestes de l'utilisateur.

Etat de finalisation

- **Affichage de ce jeu :**

Nous avons affiché une boule qui rebondit sur les murs et sur le palet et qui détruit des briques placées les unes à côté des autres, un palet qui permet de diriger la boule, et 3 murs sur lesquels rebondit la boule et 1 mur, en dessous du palet, qui détruit la boule. Le jeu a une dimension de 10 briques de largeur et 12 briques de profondeur et les briques ont un rapport d'un pour trois.

- **Interaction avec l'utilisateur :**

Nous avons mis en place une interaction avec l'utilisateur à partir de l'acquisition et du traitement des images issues d'une WebCam en réalisant le déplacement du palet contrôlé par gestes de balayage (la main avec le pouce élevé) et une gestion (poing) d'un effet de freinage ou d'un geste d'arrêt.

- **Fonctions réalisées :**

1. Rebond de la boule sur les murs, sur le palet et sur les briques.

2. Destruction des briques (disparition) lorsqu'elles sont touchées par la boule.

3. Rebondissement sur le palet et contrôle de la direction de la boule en fonction du point d'impact sur le palet.

4. Décompte des boules utilisées et contrôle à la fin d'une partie. Au début d'une partie, trois boules sont disponibles. Si les boules frappent le mur derrière le palet, elles seront détruites et le nombre de boules disponibles diminue. Lorsque la dernière boule est détruite la partie est terminée.

5. Génération aléatoire d'un nouveau niveau avec une vitesse de la boule supérieure par rapport au niveau précédent lorsque l'ensemble des briques d'un écran est détruit.

6. Calculation des points. Les points sont incrémentés chaque fois qu'une brique est détruite. (Les briques rouges donneront 2 points)

7. Choix de la taille du palet par le joueur.

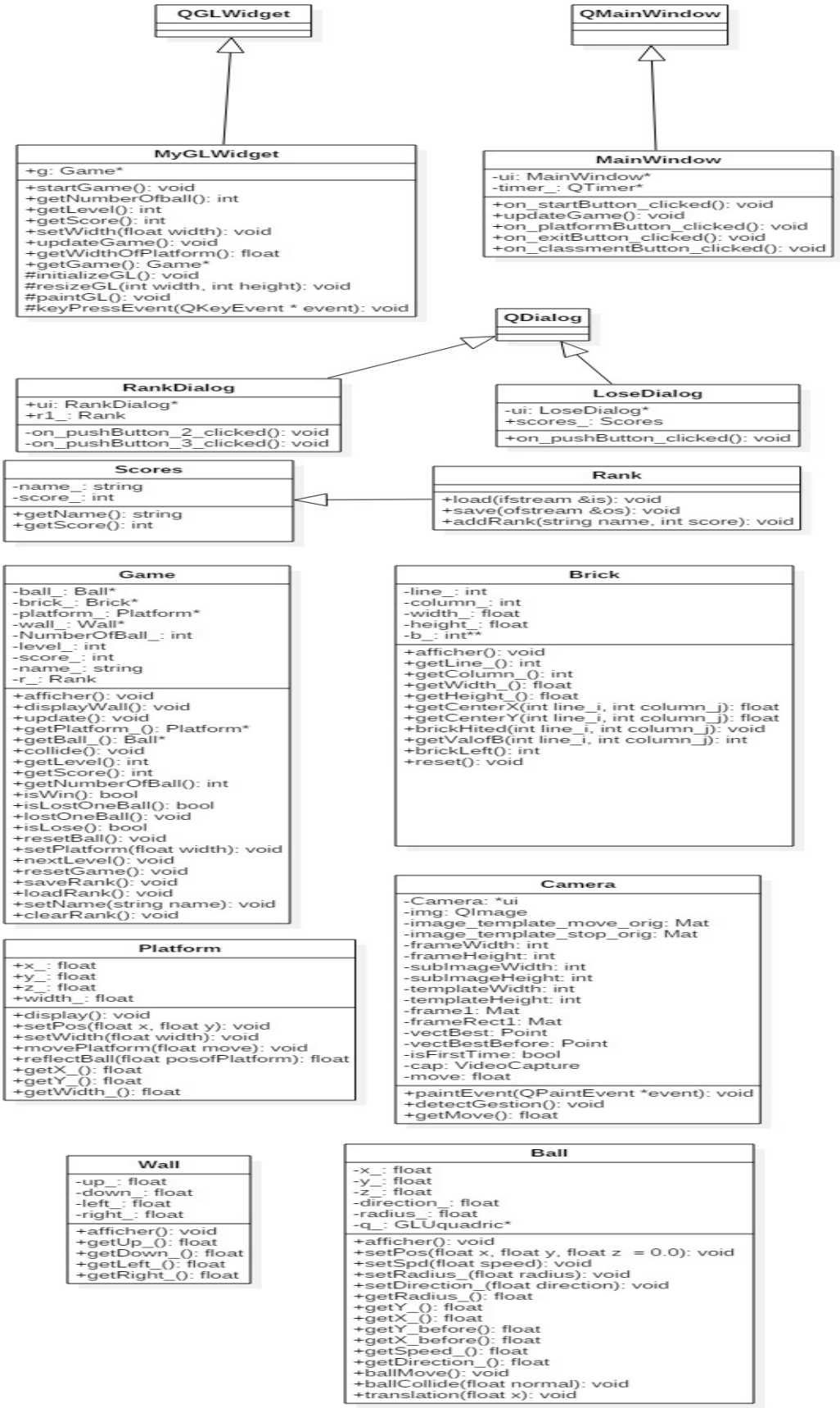
8. Sauvegarde du score/nom de joueur et affichage des premiers 10 joueurs dans le classement.

- **Amélioration :**

1. Il existe une limite de la vitesse et du rayon de la boule.

2. Nous avons éssayé d'optimiser des paramètres pour augmenter la fiabilité et la fluidité (utilisation des fonctions de multi-threading) mais nous l'avons pas bien réalisé. Nous étudierons dans le futur son amélioration.

Diagramme de classes

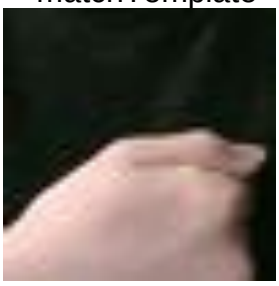


Les principales classes

Class MainWidget : le principal widget d'interface utilisateur graphique
Class Scores : management de scores
Class RankDialog : boite de dialogue pour afficher le Rank
Class LoseDialog : boite de dialogue d'afficher quand le jeu est perdu
Class MyGLWidget : Le widget d'afficher l'interface graphique en utilisant OpenGL

Class Camera : Le widget d'obtenir la gestion de balayage horizontal et arrêt et le traiter par OpenCV.

Fonction detectGestion() : identifier le gestion de balayage horizontal et arrêt par « matchTemplate » en utilisant deux images

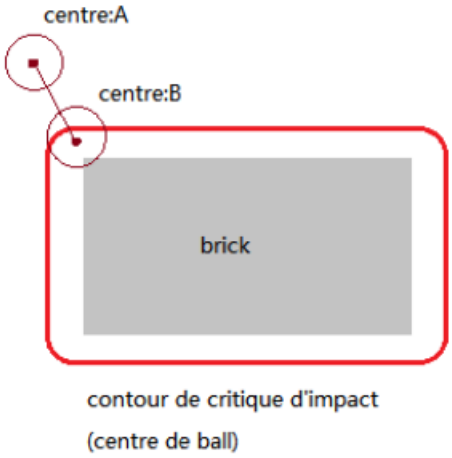


2:gestion de balayage

1:gestion d'arrêt

Class Game : management de mécanisme de jeu

Fonction collide() : identifier si la balle peut bosser contre le brick par le centre A de balle du dernier moment et le centre B de balle du ce moment. Si le segment AB et le contour de critique d'impact intersecte, on peut dire que la balle bosse contre le brick. En plus, cette fonction peut distinguer si la balle bosse en les bords de brick ou les corners de brick. Ça ira influence l'angle de réflexion.



Class Brick：Management d'information de tous les bricks
Class Platform : management d'information de platform
Class Wall : management de border de jeu.
Class Ball : management d'information de ball

Code .h

```
#ifndef BALL_H
#define BALL_H
#include <GL/glu.h>

/*
*This class is used to regist the informations of the ball.    // YANG et DU
*
*/
class Ball
{
private:
    float x_,y_,z_,direction_, speed_,radius_;//The position, direction the radius and the speed of the ball.
    GLUquadric* q_;//The quadric (ball) in QGLWidget

public:
    Ball(float x, float y, float z, float radius, float direction, float speed);// Constructor
    void afficher() const;//Display the ball in GLWidget
    virtual ~Ball();    //Destructor
     //setters
    void setPos(float x,float y,float z=0.0);
    void setSpd(float speed);
    void setRadius_(float radius);
    void setDirection_(float direction);
     //getters
    float    getRadius_();
    float getY_();
    float getX_();
    float getY_before();
    float getX_before();
    float getSpeed_();
    float getDirection_();

    void ballMove();//move the ball by change its position x et y
    void ballCollide(float normal);//change the ball's direction by the normal direction of the reflective interface
    void translation(float x);//the ball move x horizontally
     };

#endif // BALL_H

#ifndef BRICK_H
```

```
#define BRICK_H
#include <GL/glu.h>

/*
*This class is used to regist the informations of the bricks.    // YANG et DU
*
*/
class Brick
{
private:
    int line_, column_;// the number of the bricks.
    float width_,height_;// size of one brick
    int **b_;// a table 2D for represent the stats of the brick (1 exist, 2 special point,or 0 destucted)

public:
    Brick(int line, int column);//constructor
    void afficher() const;//display in the GLWidget
    virtual ~Brick();//destructor
     //getters
    int getLine_();
    int getColumn_();
    float getWidth_();
    float getHeight_();
    float getCenterX(int line_i,int column_j) const;
    float getCenterY(int line_i, int column_j) const;

    void   brickHited(int line_i,int column_j);//change the brick located in i,j to destroyed
    int    getValofB(int line_i,int column_j);// get the value of the brick located in i,j
    int    brickLeft();// the number of the actuel bricks exist
    void   reset();// reset the values of bricks
};

#endif // BRICK_H

#ifndef CAMERA_H
#define CAMERA_H
#include "opencv2/video/tracking.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <QtConcurrentrun>

#include <cstdio>
#include <iostream>
```

```
using namespace cv;
using namespace std;

#include <QWidget>

namespace Ui {
class Camera;
}

class Camera : public QWidget
{
    Q_OBJECT

public:
    explicit Camera(QWidget *parent = 0);
    ~Camera();
    void paintEvent(QPaintEvent *event);
    void detectGestion();
    float getMove();

private:
    Ui::Camera *ui;
    QImage img;
    Mat image_template_move_orig;
    Mat image_template_stop_orig;

    int frameWidth=320;
    int frameHeight=240;
    int subImageWidth=200;
    int subImageHeight=100;
    int templateWidth=50;
    int templateHeight=50;
    Mat frame1,frameRect1;
//    Mat frame1,frame2,frameRect1,frameRect2,image_template_move,image_template_stop;
//    Mat resultImageMove,resultImageStop;    // to store the matchTemplate result
    Point vectBest;
    Point vectBestBefore;
    bool isFirstTime;
    VideoCapture cap;
    float move;
};

#endif // CAMERA_H
```

```
#ifndef GAME_H
#define GAME_H

#include "ball.h"
#include "brick.h"
#include "platform.h"
#include "wall.h"
#include "rank.h"
/*
*This class is used to control the process of the game.    // YANG et DU
*
*/

using namespace std;

typedef struct point// definition of a struct Point
{
	float x;
	float y;
}Point;

class Game
{
private:
	Ball* ball_;
	Brick* brick_;
	Platform* platform_;
	Wall* wall_;

	int NumberOfBall_;// number of the ball left. (0,1,2,3)
	int level_;// the level of the game
	int score_;// the point got by player
	string name_;// the name of the player
	Rank r_;// the rank of all players(a vector of Socre(name, score))

public:
	Game();//contructor
	void afficher() const;//display the objects in the GLWidget
	void displayWall() const;// display the wall
	void update();//update to the next stats of the process of the game
	 //getters
	Platform* getPlatform_();
	Ball* getBall_();
```

```
        void collide();
        int getLevel();
        int getScore();
        int getNumberOfBall();

         //some methodes for distinguish or control the process of the game
        bool isWin();
        bool isLostOneBall();
        void lostOneBall();
        bool isLose();
        void resetBall();//put the ball on the platform
        void setPlatform(float width);
        void nextLevel();
        void resetGame();
        void saveRank();//save the rank
        void loadRank();
        void setName(string name);
        void clearRank();
        ~Game();//destructor
};

#endif // BRICK_H

#ifndef LOSEDIALOG_H
#define LOSEDIALOG_H
#include <QDialog>
#include "scores.h"

namespace Ui {
class LoseDialog;
}

class LoseDialog : public QDialog
{
        Q_OBJECT

public:
        explicit LoseDialog(QWidget *parent = 0);
        ~LoseDialog();

private slots:

        void on_pushButton_clicked();
```

```
private:
    Ui::LoseDialog *ui;
    Scores scores_;
};

#endif // LOSEDIALOG_H

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTimer>
#include "rankdialog.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_startButton_clicked();
    void updateGame();

    void on_platformButton_clicked();

    void on_exitButton_clicked();

    void on_classmentButton_clicked();

private:
    Ui::MainWindow *ui;
    QTimer *timer_;
};

#endif // MAINWINDOW_H

#ifndef MYGLWIDGET_H
```

```
#define MYGLWIDGET_H

#include <QGLWidget>
#include <QKeyEvent>
#include <QColor>
#include <QVector2D>
#include <GL/glu.h>
#include "game.h"

// Classe dediee a l'affichage d'une scene OpenGL
class MyGLWidget : public QGLWidget
{
    Q_OBJECT

public:

    // Constructeur
    MyGLWidget(QWidget * parent = nullptr);
    void startGame();



    int getNumberOfball();
    int getLevel();
    int getScore();
    void setWidth(float width);
    void updateGame();
    float getWidthOfPlatform();
    Game* getGame();
    ~MyGLWidget();

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);
```

```
private:
    Game* g;
    // Quelques variables a definir
public slots:

};

#endif // MYGLWIDGET_H


#ifndef PLATFORM_H
#define PLATFORM_H
#include <GL/glu.h>

class Platform
{
private:
    float x_,y_,z_;
    float width_;

public:
    Platform(float x, float y, float width);
    void display() const;
    void setPos(float x,float y);
    void setWidth(float width);
    void movePlatform(float move);
    float reflectBall(float posofPlatform);
    float getX_();
    float getY_();
    float getWidth_();
};

#endif // PLATFORM_H

#ifndef RANK_H
#define RANK_H
#include<vector>
#include"scores.h"

class Rank: public vector<Scores>
{
public:
    Rank();
```

```
    void load(ifstream &is);
    void save(ofstream &os);
    void addRank(string name, int score);
};

#endif // RANK_H

#ifndef RANKDIALOG_H
#define RANKDIALOG_H

#include <QDialog>
#include "rank.h"

namespace Ui {
class RankDialog;
}

class RankDialog : public QDialog
{
    Q_OBJECT

public:
    explicit RankDialog(QWidget *parent = 0);
    ~RankDialog();

private slots:
    void on_pushButton_2_clicked();

    void on_pushButton_3_clicked();

private:
    Ui::RankDialog *ui;
    Rank r1_;
};

#endif // RANKDIALOG_H

#ifndef SCORES_H
#define SCORES_H

#include <fstream>
#include <string>
using namespace std;
```

```
class Scores
{
	string name_;
	int score_;

public:
	Scores(string, int);
	Scores();
	string getName();
	int getScore();
};

#endif // SCORES_H

#ifndef WALL_H
#define WALL_H
#include <GL/glu.h>

class Wall
{
private:
	float up_,down_,left_,right_;
	public:

public:
	Wall(float up,float down,float left,float right);
	void afficher() const;
	float getUp_();
	float getDown_();
	float getLeft_();
	float getRight_();
};

#endif // WALL_H
```