UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

MINES TELECOM INSTITUT Mines-Télécom

Telecom Saint-Etienne

télécom saint-étienne école d'ingénieurs / nouvelles technologies

RAPPORT de PROJET Tetris 3D

Bibliothèques de Développement Multimédia

Qi LI

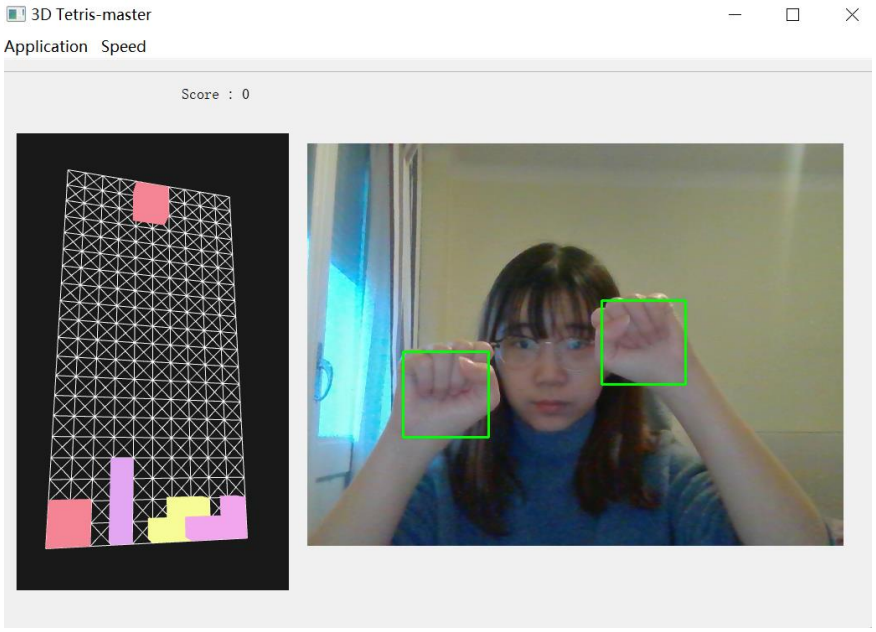
Yueming YANG

I. Specification

Quand on lance le programme, l'interface utilisateur est affichée à l'écran. L'interface de jeu est à gauche et l'interface de contrôle est à droite. Il y a deux barres de menu en haut à gauche : 'Application' et 'Speed'. Dans la barre de menu de 'Application', il y a deux options : 'Quit' et 'New game' pour quitter le jeu et commencer un nouveau jeu. Ils correspondent aux touches de raccourci 'Q' et 'N'. Dans la barre de menu de 'Speed', il y a trois options : 'Slow', 'Medium' et 'Fast' pour ajuster la vitesse de chute des tétriminos. Ils correspondent aux touches de raccourci '1', '2' et '3'.

D'abord, L'interface du jeu n'affiche que le fond noir initial et le grille 10*20 avec le croix, l'interface de contrôle n'a pas non plus d'affichage. Nous devons commencer le jeu via le bouton dans le menu ou par le raccourci 'N'. Nous pouvons aussi commencer le jeu par cliquer sur le plateau incliné. Nous pouvons ouvrir la caméra avec le raccourci 'B' et voir l'interface de contrôle.

Nous pouvons non seulement contrôler le déplacement et la rotation du tétrimino par des gestes, mais également par le clavier. Lorsque la position de la main gauche est plus haute que la main droite, le tétrimino se déplace vers la droite. La geste de déplacement du tétrimino à gauche est l'inverse. Lorsque les poings du joueur sont frappés l'un contre l'autre, le tétrimino est en rotation. Lorseque les 2 poings déplace vers le bas ensemble,les tétriminos descente rapide vers le bas. Le clavier contrôle le mouvement gauche et droit à l'aide de touche fléchée gauche et droite. Lorsque nous appuyons sur la touche fléchée haute, le tétrimino tourne dans le sens des aiguilles d'une montre et lorsque nous appuyons sur la touche fléchée bas, le tétrimino tourne contre le sens des aiguilles d'une montre. Lorsque nous appuyons sur la barre d'espace, le tétrimino atteindra directement le bas du plateau. Lorsque nous éliminons une ligne entière, nous obtenons un point. Lorsque la grille est plein, la partie est terminée et nous pouvons commencer une nouvelle partie par le souris, le raccourci 'N' ou le bouton ' New game' dans le menu.



II. Conception

Classe Piece :

Représente le tétrimino actuel, inclut les informations nécessaires pour effectuer l'opération.

Classe Game :

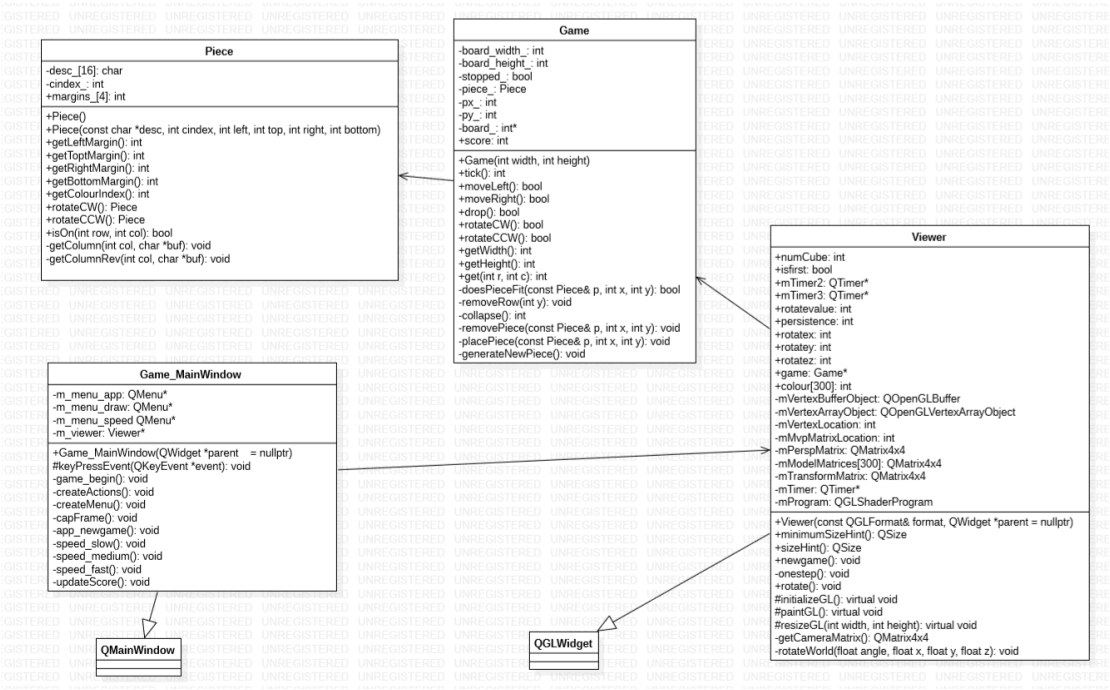
Représente un jeu, y compris les différentes fonctions du fonctionnement du jeu.

Class Game_MainWindow :

L'interface principale du jeu, y compris le menu, l'interface de jeu et l'interface de contrôle.

Class Viewer :

Dédié à l'affichage de la scène du jeu OpenGL



III. L'état de finalisation de l'application

Les fonctions qui ont été validées :

1. Affichage en perspective du plateau constitué d'une grille de 10 × 20 cellules représentant le plateau.
2. Apparition d'un tétriminos et descente de ce tétriminos en cliquant sur l'interface du jeu ou en appouyant sur la touche 'N' ou en cliquant 'New game' dans le menu.
3. Déplacement latéral du tétriminos commandé par un geste de rotation des mains du joueur vers la droite ou vers la gauche.
4. Rotation du tétriminos commandé lorsque les poings du joueur sont frappés l'un contre l'autre.
5. Arrêt du tétriminos lorsqu'il arrive en bas et apparition du tétriminos suivant en haut.
6. Détection d'une ligne remplie et incrémentation des points.
7. Disparition des blocs associés à la ligne remplie et descente des blocs situés au dessus.
8. Détection de la fin de la partie et ré-initialisation pour la partie suivante.
9. Ajout de niveaux de jeux correspondant à une accélération de la descente des tétriminos.
10. Descente rapide du tétriminos commandée par un geste de déplacement des 2 poings vers le bas

Les fonctions qui ne sont pas finalisées :

1. Apparition d'un tétriminos et descente de ce tétriminos en glissant sur le plateau incliné.
2. Affichage du tétrimino suivant.

.

IV. Les fichiers d'entête des classes

Class Piece

```
#ifndef GAME_HPP
#define GAME_HPP

// @author Yueming YANG
class Piece {
public:
	// Constructor
	Piece();
	// @param desc: an array for represent a piece of tetris
	// @param cindex: the index for a color
	// @param left: the margin on the left of the piece
	// @param top: the margin on the top of the piece
	// @param right: the margin on the right of the piece
	// @param bottom: the margin on the bottom of the piece
	Piece(const char *desc, int cindex,
		int left, int top, int right, int bottom);

	Piece& operator =(const Piece& other);

	// Get the corresponding margin value
	int getLeftMargin() const;
	int getTopMargin() const;
	int getRightMargin() const;
	int getBottomMargin() const;

	// Get the corresponding index of colour
	int getColourIndex() const;

	// Set clockwise rotation
	Piece rotateCW() const;
	// Set counterclockwise rotation
	Piece rotateCCW() const;

	// Determine if it is part of a piece
	bool isOn(int row, int col) const;

private:
	// Get information for each column
```

```
    void getColumn(int col, char *buf) const;
    // Get the information of each column in reverse order
    void getColumnRev(int col, char *buf) const;

    // Declare an array of size 16
    char desc_[16];
    // Declare the index for a color
    int cindex_;
    // Declqre an array of size 4
    int margins_[4];
};
```

Class Game

```
// @author Qi LI
class Game
{
public:
    // Create a new game instance with a well of the given dimensions.
    // Note that internally, the board has four extra rows, to hold a
    // piece that has just begun to fall.
    // Constructor
    Game(int width, int height);

    // Destructor
    ~Game();

    // Advance the game by one tick.    This usually just pushes the
    // currently falling piece down by one row.    It can sometimes cause
    // one or more rows to be filled and removed.    This method returns
    // three kinds of values:
    //                                  <0: game over!
    //                    0: business as usual.
    //        {1,2,3,4}: the most recent piece removed 1,2,3 or 4 rows, and
    //                                          a new piece has started to fall.
    int tick();

    // Move the currently falling piece left or right by one unit.
    // Returns whether the move was successful.
    bool moveLeft();
    bool moveRight();

    // Drop the current piece to the lowest position it can legally
    // occupy.    Returns whether anything happened.
```

```
    bool drop();

    // Rotate the piece clockwise or counter-clockwise.  Returns whether
    // the rotation was successful.
    bool rotateCW();
    bool rotateCCW();

    // Declare and initilize the score
    int score = 0;

    // Get the width of the board
    int getWidth() const
    {
        return board_width_;
    }
    // Get the height of the board
    int getHeight() const
    {
        return board_height_;
    }

    // Get the contents of the cell at row r and column c.  Returns
    // the following values:
    //                                          -1: Cell is empty.
    //     {0,1,2,3,4,5,6}: Cell contains a piece with the given ID.  Use
    //                                              this ID to choose a colour when drawing this
cell.
    // NOTE!  You can (and should) actually call this method with values
    // for r in [0,board_height_+4), not [0,board_height_].  The top four
    // rows are added on to accommodate new pieces that are falling into
    // the well.

    // Get the location of a unit
    int get(int r, int c) const;
    int& get(int r, int c);

private:
    // Determine if the current position can place this piece
    bool doesPieceFit(const Piece& p, int x, int y) const;

    // Delete this row when a row is filled and move all of the above pieces down one row
    void removeRow(int y);

    // Repeat scanning all rows from bottom to top, removing the first full
```

```
        // row, stopping when there are no more full rows.
        int collapse();

        // Remove one piece
        void removePiece(const Piece& p, int x, int y);
        // Place a piece
        void placePiece(const Piece& p, int x, int y);

        // Generate a new piece
        void generateNewPiece();

private:
        // The size of the board
        int board_width_;
        int board_height_;

        // Game ending sign
        bool stopped_;

        // Declare a piece of tetrimino of type Piece
        Piece piece_;
        // Coordinates of current piece
        int px_;
        int py_;
        // The total number of pieces of a board
        int* board_;
};

#endif
```

Class

```
#ifndef GAME_MAINWINDOW_H
#define GAME_MAINWINDOW_H

#include <QMainWindow>
#include <QMenuBar>
#include <QMenu>
#include <QAction>
#include <vector>
#include "Viewer.hpp"
#include <QKeyEvent>
```

Class Game_MainWindow

```
// @author Qi LI

// Separate UI layout control from other control code
namespace Ui {
class Game_MainWindow;
}

// L'interface principale du jeu
class Game_MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    // Constructor
    explicit Game_MainWindow(QWidget *parent = nullptr);
    // Destructor
    ~Game_MainWindow();

protected:
    // Function for keyboard interaction management
    void keyPressEvent(QKeyEvent *event);

private:
    Ui::Game_MainWindow *ui;

    // Begin the game
    void game_begin();

    // Create a new action for quiting and pushes it onto the menu actions vector
    void createActions();
    // Create the menu 'Application' and 'Speed'
    void createMenu();

    // Each menu itself
    QMenu* m_menu_app;
    QMenu* m_menu_draw;
    QMenu* m_menu_speed;

    // Create a menu corresponding action
    std::vector<QAction*> m_menu_actions;
    std::vector<QAction*> m_menu_actions1;
```

```
    // Add a viewer to show the game
    Viewer* m_viewer;

private slots:
    // Screen capture, get two punch position information
    void capFrame();
    // Start a new game
    void app_newgame();
    void updateScore();

    //speed of game
    void speed_slow();
    void speed_medium();
    void speed_fast();

};

#endif // GAME_MAINWINDOW_H
```

Class Viewer

```
#ifndef VIEWER_HPP
#define VIEWER_HPP

#include <QGLWidget>
#include <QGLShaderProgram>
#include <QMatrix4x4>
#include <QtGlobal>
#include "game.hpp"
#include <QOpenGLBuffer>
#include <QOpenGLVertexArrayObject>
#include <QGLBuffer>


// @author Qi LI

// L'interface du jeu
class Viewer : public QGLWidget {

    Q_OBJECT

public:
    // Constructor
```

```
    Viewer(const QGLFormat& format,QWidget *parent = nullptr);
    // Destructor
    virtual ~Viewer();

    // windows' size
    QSize minimumSizeHint() const;
    QSize sizeHint() const;

    // The position of the cube in the array
    int numCube;

    // Initialize all cube's information
    void newgame();
    // Flag for start the timer
    bool isfirst;

    // The timer for a piece move one step
    QTimer* mTimer2;

    //implementing gravity
    QTimer* mTimer3;// for gravity, each tick rotates once
    int rotatevalue;

    //implementing holding multiple mouse buttons, rotating;
    int persistence;

    //implementing reset
    int rotatex;
    int rotatey;
    int rotatez;

    // Declare a game
    Game* game;

    // implement colour
    int colour[300];

private slots:
    // Advance the game by one tick.
    void onestep();
    // Rotate the tetrismino(In fact, here we only do translation transformation)
    void rotate();

protected:
```

```
    // Called when GL is first initialized
    virtual void initializeGL();
    // Called when our window needs to be redrawn
    virtual void paintGL();
    // Called when the window is resized (formerly on_configure_event)
    virtual void resizeGL(int width, int height);
    // Function for mouse interaction management to start the game after we click on the screen
of game
    virtual void mousePressEvent(QMouseEvent *event);

private:

    // Create a matrix for camera that supports transformation functions
    QMatrix4x4 getCameraMatrix();
    // Function for make rotation
    void rotateWorld(float angle, float x, float y, float z);

    // Create a VertexBufferObject
    QOpenGLBuffer mVertexBufferObject;
    // Create a VertexArrayObject
    QOpenGLVertexArrayObject mVertexArrayObject;

    // The location of vertex
    int mVertexLocation;
    // The final transformation matrix
    int mMvpMatrixLocation;

    // A matrix representing the visual range
    QMatrix4x4 mPerspMatrix;
    // A matrix that holds cube information
    QMatrix4x4 mModelMatrices[300];
    // A matrix for transformation
    QMatrix4x4 mTransformMatrix;

    // The timer to update the interface
    QTimer* mTimer;
    // Create a object for shader program
    QGLShaderProgram mProgram;
};

#endif
```