Tetris

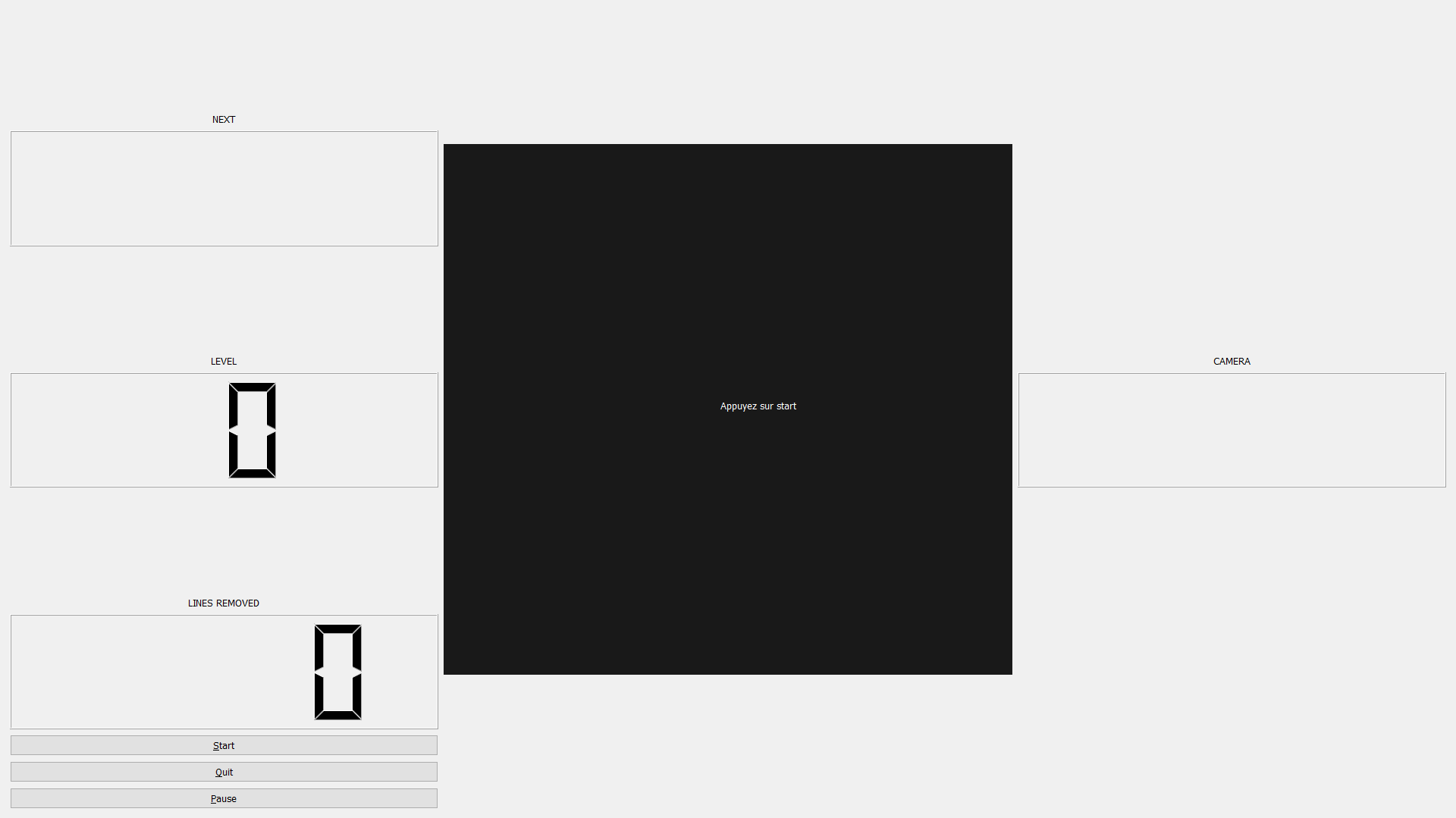
SOMMAIRE :

1. [Spécification de l’application p.2](#spécification_texte)
2. [Conception de l’application p.3](#conception_texte)
3. [Finalisation de l’application p.4](#finalisation_texte)
4. [Annexe (En-tête) p.5](#annexe_texte)
5. [CameraWidget p.5](#camera_texte)
6. [FistDetection p.6](#FistDetection_texte)
7. [GLWidget p.7](#GLWidget_texte)
8. [TetrixPiece p.8](#TextrixPiece_texte)
9. [TetrixWindow p.9](#TetrixWindow_texte)
10. Spécification de l’application :

Zone d’affichage de la prochaine pièce

Zone de jeu

Zone d’affichage de la caméra



Zone d’affichage du niveau

Boutons

Zone du nombre de lignes effacées

Figure : Capture d’écran de l’interface lors du lancement

L’utilisateur peut depuis cette interface :

* Appuyez sur le bouton start pour démarrer le jeu
* Appuyez sur le bouton pause pour mettre en pause le jeu
* Appuyez sur le bouton exit pour quitter le jeu

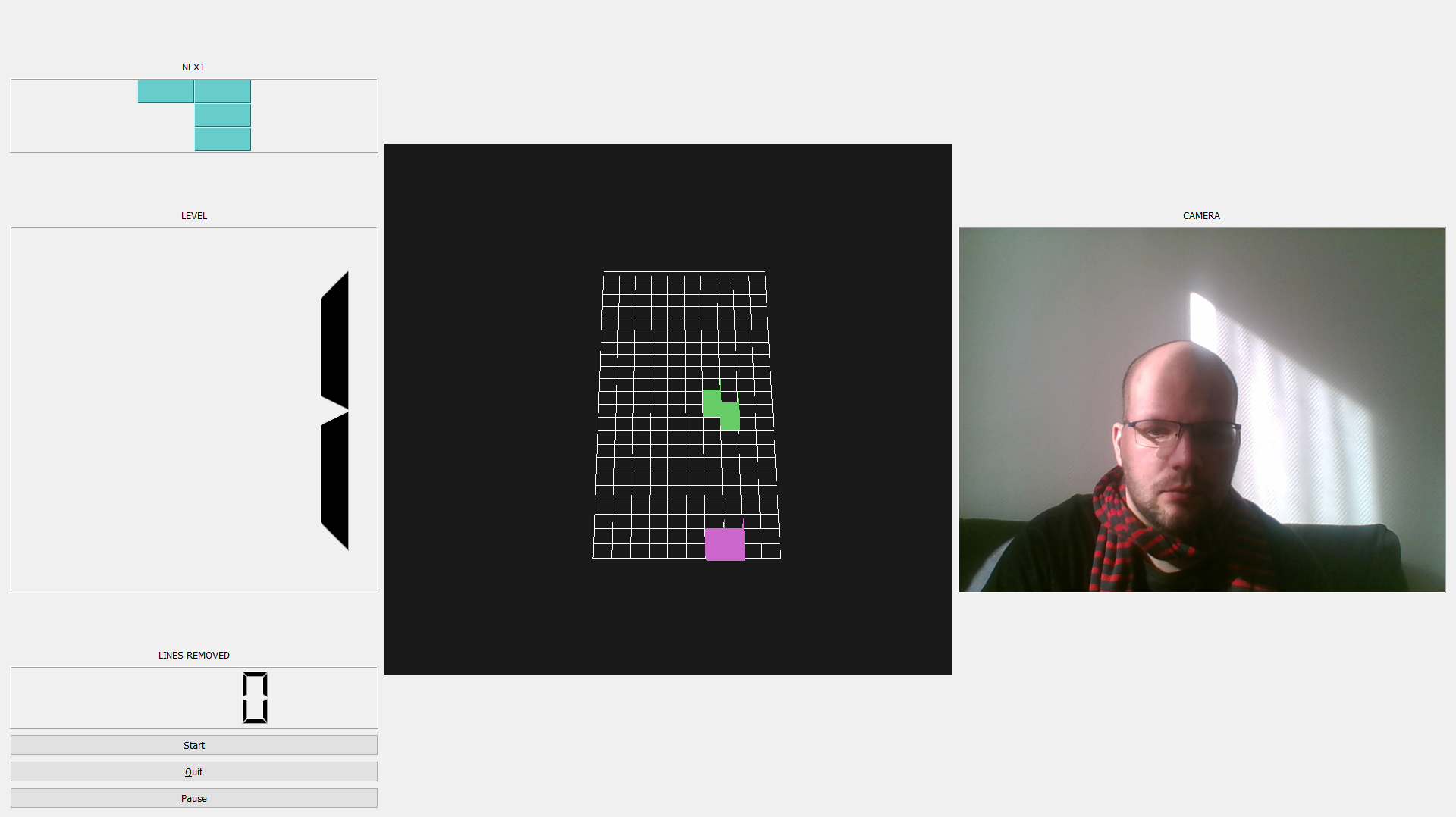


Figure : Capture d’écran de l’interface lors du démarrage d’une partie

Une fois que l’utilisateur à appuyer sur le bouton start, la caméra s’affiche.

L’utilisateur peut depuis cet écran de jeu :

* Appuyez sur les flèches pour déplacer et faire la rotation du tétriminos.
* Appuyez sur la barre espace pour déplacer vers le bas le tétriminos.
* Utilisez l’interaction avec la caméra pour déplacer et faire la rotation du tétriminos.

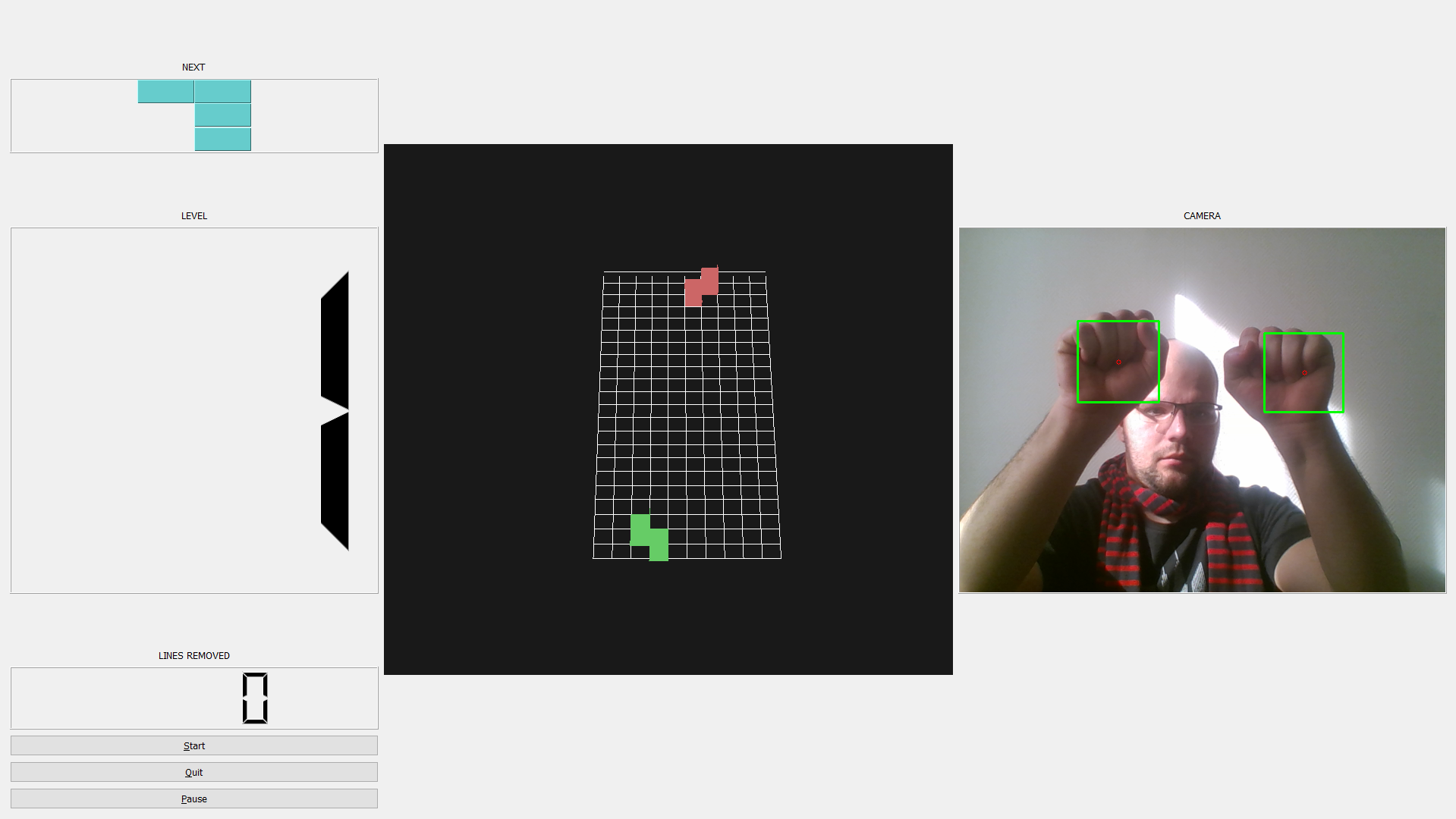


Figure : Capture d’écran d’une interaction avec la caméra

1. Conception de l’application :
2. Finalisation de l’application

Nous avons implémenté la totalité des fonctions obligatoires et nous avons implémenter l’utilisation des niveaux dans notre jeu.

1. Annexe (En-tête)
   1. CameraWidget

#ifndef CAMERAWIDGET\_H

#define CAMERAWIDGET\_H

#include <opencv2/core.hpp>

#include <opencv2/imgproc.hpp>

#include <opencv2/highgui.hpp>

#include <opencv2/objdetect.hpp>

#include "opencv2/core/version.hpp"

#if CV\_MAJOR\_VERSION == 2

#include "opencv2/opencv.hpp"

#elif CV\_MAJOR\_VERSION == 4

#include <opencv2/videoio.hpp>

#endif

#include <QWidget>

#include <cstdio>

#include <iostream>

#include <QLabel>

#include "fistdetection.h"

/\* Autor : Thomas Mion \*/

using namespace std;

class CameraWidget : public QWidget

{ Q\_OBJECT

public:

// Constructor

explicit CameraWidget(QLabel\* label, QWidget \*parent = nullptr);

private:

// cam object

cv::VideoCapture cap;

// frame captured by cap

cv::Mat frame\_;

// image transformed

QImage img\_;

// label where display img

QLabel\* labelCam;

// fist detector

FistDetection detector;

bool play\_ = false;

// Transform Mat to QImage to display it

QImage Mat2QImage(cv::Mat const& src,bool flipe =true);

signals:

void tryMoveCam(Movment mvm);

public slots:

// 2 methods to display or not the camera.

void changePlay();

void play();};

#endif // CAMERAWIDGET\_H

* 1. FistDetection

#ifndef FISTDETECTION\_H

#define FISTDETECTION\_H

#include "opencv2/video/tracking.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include "opencv2/highgui/highgui.hpp"

#include <opencv2/objdetect.hpp>

#include "opencv2/core/version.hpp"

#if CV\_MAJOR\_VERSION == 2

#include "opencv2/opencv.hpp"

#elif CV\_MAJOR\_VERSION == 4

#include <opencv2/videoio.hpp>

#endif

#include <cstdio>

#include <iostream>

/\* Autor : Thomas Mion \*/

using namespace std;

// Enum type that stock all movment that can be detected

enum Movment {kNone, rRight, rLeft, mRight, mLeft};

// Class that permit to detect fist

class FistDetection

{

private:

// Frame to display

cv::Mat displayFrame;

// Fist centers detected

vector<cv::Point> currentFistCenters;

// Cascade used

cv::CascadeClassifier face\_cascade;

// number of time that width frame it divide for create the width of the rectangle for double fists

int divisionDetect = 4;

// number of pixel of heigth of the rectangle for double fists

int heigthDetect = 15;

public:

// Constructor

FistDetection();

// Load default cascade

bool loadCascade();

// Load personal cascade

bool loadCascade(cv::String path);

// detect movment from a frame

Movment detection(cv::Mat frame);

void setDivisionDetect(int value){divisionDetect=value;}

cv::Mat getDisplayFrame(){return displayFrame;}

};

#endif // FISTDETECTION\_H

* 1. GLWidget

#ifndef GLWIDGET\_H

#define GLWIDGET\_H

#include <QBasicTimer>

#include <QFrame>

#include <QPointer>

#include <QTime>

#include "tetrixpiece.h"

#include "fistdetection.h"

#include <QDebug>

#include <QGLWidget>

#include <QMouseEvent>

#include <QTimer>

#include <GL/glu.h>

/\* Autor : Hugo Nicolle \*/

class QLabel;

// Classe dediee a l'affichage d'une scene OpenGL et au déroulement du jeu

class GLWidget : public QGLWidget

{

Q\_OBJEC

public:

// Constructeur

GLWidget(QWidget \* parent = nullptr);

//Draw a cube in the coordinate system of the GL scene

void createCube(double x, double z, TetrixShape shape);

void setNextPieceLabel(QLabel \*label);

public slots:

//Initialize a game

void start();

//Put the game

void pause();

//Move a piece according to the argument movement

void tryMoveCam(Movment mvm);

//Interactions with the keyboard

void kPressEvent(QKeyEvent \*event);

signals:

//Indicates the score changed

void scoreChanged(int score);

//Indicates a level change

void levelChanged(int level);

//Indicates lines were removed from the board

void linesRemovedChanged(int numLines);

protected:

// Fonction d'initialisation

void initializeGL();

// Fonction de redimensionnement

void resizeGL(int width, int height);

// Fonction d'affichage

void paintGL();

// Timer gestion

void timerEvent(QTimerEvent \*event) ;

private:

enum { BoardWidth = 10, BoardHeight = 22 };

// Find a shape in the board

TetrixShape &shapeAt(int x, int y) { return board[(y \* BoardWidth) + x]; }

// Returns a timeout time for the timer

int timeoutTime() { return 1000 / (1 + level); }

// Completely clears the board for a new game

void clearBoard();

// Drops the current piece at the bottom

void dropDown();

// Moves a piece one line down

void oneLineDown();

// Changes game variables when a piece reaches the bottom of the board

void pieceDropped(int dropHeight);

// Removes lines that are full

void removeFullLines();

// Draws a square in a painter

void drawSquare(QPainter &painter, int x, int y, TetrixShape shape);

// Changes the current piece

void newPiece();

// Shows the next piece

void showNextPiece();

// Tells if a piece can move to a location and move it if it's possible

bool tryMove(const TetrixPiece &newPiece, int newX, int newY);

// Last time where the fist are detected close to one another

QTime lastMvmTime;

QBasicTimer timer;

QPointer<QLabel> nextPieceLabel;

bool isStarted;

bool isPaused;

bool isWaitingAfterLine;

// The piece that the player is moving

TetrixPiece curPiece;

TetrixPiece nextPiece;

// The X position of the current piece

int curX;

// The X position of the current piece

int curY;

// The number of lines removed

int numLinesRemoved;

// The number of pieces dropped

int numPiecesDropped;

int score;

int level;

// A 1D table representing the board

TetrixShape board[BoardWidth \* BoardHeight];

// Draws the grid

void paintLines();

};

#endif // GLWIDGET\_H

* 1. TetrixPiece

#ifndef TETRIXPIECE\_H

#define TETRIXPIECE\_H

/\* Inspired by QtCreator, modified by Thomas Mion and Hugo Nicolle \*/

// Enum type containing all the shapes a piece can have

enum TetrixShape { NoShape, ZShape, SShape, LineShape, TShape, SquareShape,

LShape, MirroredLShape };

//Represents a piece

class TetrixPiece

{

public:

TetrixPiece() { setShape(NoShape); }

//Set the shape of a piece at random

void setRandomShape();

// Set the shape of piece to a defined one

void setShape(TetrixShape shape);

// Returns the shape of a piece

TetrixShape shape() const { return pieceShape; }

// Returns the x coordinates of the squares

int x(int index) const { return coords[index][0]; }

// Returns the y coordinates of the squares

int y(int index) const { return coords[index][1]; }

// Returns the min of the x coordinates of the squares

int minX() const;

// Returns the max of the x coordinates of the squares

int maxX() const;

// Returns the min of the y coordinates of the squares

int minY() const;

// Returns the max of the y coordinates of the squares

int maxY() const;

// Rotates a piece counter clockwise

TetrixPiece rotatedLeft() const;

// Rotates a piece clockwise

TetrixPiece rotatedRight() const;

private:

// Set the x coordinates of the squares

void setX(int index, int x) { coords[index][0] = x; }

// Set the y coordinates of the squares

void setY(int index, int y) { coords[index][1] = y; }

// The shape of the piece

TetrixShape pieceShape;

// Coordinates of the squares composing a piece

int coords[4][2];

};

#endif

* 1. TetrixWindow

#ifndef TETRIXWINDOW\_H

#define TETRIXWINDOW\_H

#include <QFrame>

#include <QWidget>

#include "camerawidget.h"

#include "glwidget.h"

QT\_BEGIN\_NAMESPACE

class QLCDNumber;

class QLabel;

class QPushButton;

QT\_END\_NAMESPACE

class TetrixBoard;

/\* Inspired by QtCreator, modified by Thomas Mion and Hugo Nicolle \*/

class TetrixWindow : public QWidget

{

Q\_OBJECT

public:

// Constructor

TetrixWindow();

private:

// Create a label from a text

QLabel \*createLabel(const QString &text);

// Widget for gl object

GLWidget \*glWidget;

TetrixBoard \*board;

// Widget for method camera

CameraWidget \*Camera;

// Qlabel for camera display

QLabel \*labelCamera;

// QLabel for next Piece display

QLabel \*nextPieceLabel;

// LCD for score

QLCDNumber \*scoreLcd;

// LCD for level

QLCDNumber \*levelLcd;

// LCD for lines destroyed

QLCDNumber \*linesLcd;

// Button to start game

QPushButton \*startButton;

// Button to quit application

QPushButton \*quitButton;

// Button to pause game

QPushButton \*pauseButton;

void keyPressEvent(QKeyEvent \*event);

private slots :

// Slots that permit to destroy the cam when we quit application

void destroyCam();

signals:

void kPressEvent(QKeyEvent \*event);

};

#endif