Hao Huynh

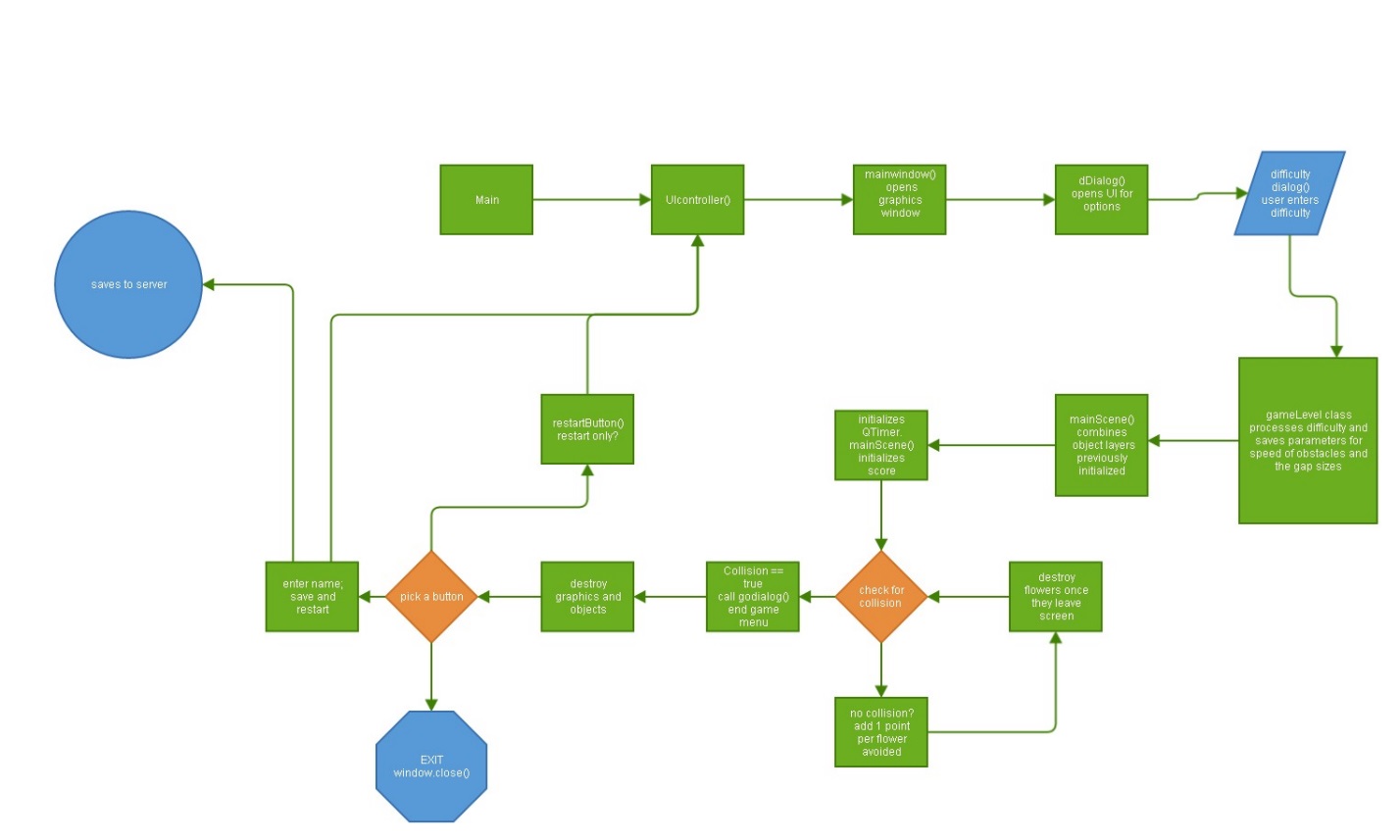
Abel Salazar

Tiffany Pan

Flappy Bird 2.0

When you first open the game a window pops up and to initiate the start menu you press the space bar. This will lead you to the main User Interface that has a combo box and several push buttons. The combo box will determine the speed in which the flower obstacles spawn and also the amount pull gravity has on the player.

This game is a 2D-scroller. The rules are simple and straight forward. By pressing the space-bar, you will get a certain amount of lift depending on the level of difficulty. Obstacles are spawned at the right side of the map in the form of flower stocks. Your objective is to navigate through as many flowers as possible without colliding with them. Your score is determined by the amount of flowers you clear. All player information is saved at the end of each round and is saved to the RCC server.

This is a simple representation of what is going on. The File is also provided as a jpeg with the file folder.

|  |  |
| --- | --- |
| **Q Libraries Used** | **#Where** |
| <QObject> | * Uicontroler.cpp * Mainscene.cpp * Godialog.cpp * Fbhelper.cpp * Difficultydialog.cpp |
| <QTimer> | * Uicontroller.cpp |
| <QMediaPlayer> | * Uicontroller.cpp * Mainwindow.cpp |
| <QMainWindow> | * mainwindow.cpp |
| <QGraphicsScene> | * Mainscene.cpp |
| <QtGui> | * Mainscene.cpp |
| <QGraphicsPixmapItem> | * Mainscene.cpp |
| <QLabel> | * Mainscene.cpp |
| <QGraphicsProxyWidget> | * Mainscene.cpp |
| <QDialog> | * Difficultydialog.cpp |
| <QDebug> | * Godialog.cpp |
| <QString> | * Gamelevel.cpp |
| <QApplication> | * Main.cpp |
| <QTcpSocket> | * Fbhelper.cpp |

|  |  |
| --- | --- |
| **C++ / STD Libraries** | **#Where?** |
| <ctime> | Mainscene.cpp - 13 |

|  |  |
| --- | --- |
| **Q-SQL Classes** | **#Where** |
| <QSql> |  |
| <QSqlQuery> |  |
| <QSqlDatabase> |  |

|  |  |  |
| --- | --- | --- |
| **Classes Created** | **Type** | **#namespace** |
| DifficultyDialog | Public QDialog | Ui |
| FBHelper |  |  |
| GameLevel |  |  |
| GODialog |  | Ui |
| MainScene | Public QGraphicsScene |  |
| MainWindow | Public MainWindow | Ui |
| UiController | Public QObject |  |

|  |  |  |
| --- | --- | --- |
| **Type** | **Public Slots** | **#Where** |
| Void | createFlowers() | Uicontroller.cpp - 73, 171 |
| Void | processSpaceKeyPress() | Uicontroller.cpp - 76, 178 |
| Void | processCollision() | Uicontroller.cpp - 89, 181 |
| Void | stateChanged(MediaPlayer::State) | Uicontroller.cpp - 82, 101, 114 |
| Void | startGame() | Uicontroller.cpp - 125, 187 |
| Void | restart() | Uicontroller.cpp - 146, 190 |
| Void | levelUp() | Uicontroller.cpp - 156, 184 |
| Void | moveFlowers() | mainWindow.cpp – 111 |
| Void | freeFallBird() | mainWindow.cpp – 99 |
| Void | sendData() | Fbhelper.cpp – 12, 38 |
| Void | processError() | Fbhelper.cpp – 14, 16, 53 |

|  |  |  |
| --- | --- | --- |
| **Type** | **Private Slots** | **#Where** |
| Void | on\_quitButton\_clicked() | Godialog.cpp - 46 |
| Void | on\_restartButton\_clicked() | Godialog.cpp - 54 |
| Void | on\_sRestartButton\_clicked() | Godialog.cpp - 63 |
| Void | on\_startButton\_clicked() | Difficultydialog.cpp - 39 |

|  |  |  |
| --- | --- | --- |
| **Type** | **Private Functions** | **#Where** |
| Void | connectSystems() | Uicontroller.cpp – 39, 168 |
| Short | getHeightScale() | mainScene.cpp – 53, 126, 134 |
| Void | addNewFlower(QGraphicsPixmapItem) | mainScene.cpp – 63, 131, 139 |
| Void | deletePFlower(QGraphicsPixmapItem) | mainScene.cpp – 74, 157 |
| Void | createABird(const QSize&) | mainScene.cpp – 28, 87, 170, 291 |
| Void | updateScore(QGrpahicsPixmapItem) | mainScene.cpp – 111, 154 |

|  |  |  |
| --- | --- | --- |
| **Type** | **Signals** | **#Where** |
| Void | pressSpaceKey() | mainWindow.cpp - 66 |
| Void | processCollision() | mainWindow.cpp – 203, 272 |
| Void | levelUp() | mainWindow.cpp - 78 |
| Void | Restart() | Godialog.cpp – 57, 69 |
| Void | startGame() | Difficultydialog.cpp - 42 |

|  |  |
| --- | --- |
| **Public Constructors** | **#Where** |
| explicit UIController(QObject \*) | * Main.cpp – 8 * Uicontroller.cpp – 8, 46, 58, 89, 114, 125, 146, 156, 168 |
| explicit MainWindow(QWidget \*) | * mainScene.cpp – 28, 29, 114, 203, 272, 291, 292 * mainWindow.cpp – 5, 13, 25, 56, 62, 71, 83, 89, 99, 105, 111 * uicontroller.cpp - 11 |
| MainScene(QObject) | * mainScene.cpp – 9, 43, 53, 63, 74, 87, 111, 122, 147, 166, 183, 216, 247, 264, 280 * mainWindow.cpp - 37 |
| Explicit GODialog(QWidget \*) | * godialog.cpp – 8, 20, 29, 38, 46, 54, 63 * uicontroller.cpp - 18 |
| populateUI(int) | * godialog.cpp – 28 * uicontroller.cpp - 106 |
| QString getLevelN() const | * difficultydialog.cpp – 14, 15, 16 * gamelevel.cpp - 38 |
| Static void setLevelN(const QString&) | * difficultydialog.cpp – 41 * gamelevel.cpp - 47 |
| int getId(int) | * gamelevel.cpp – 63, 101, 110 |
| Static void setId(int) | * gamelevel.cpp – 72 |
| Static GameLevel getInstance() | * gamelevel.cpp – 90 * mainScene.cpp – 170, 171 * uicontroller.cpp - 132 |
| static float getBirdPicScalar() | * gamelevel.cpp – 99 * mainScene.cpp – 170, 171 |
| static int getFlowerSpeed() | * gamelevel.cpp – 108 * uicontroller.cpp - 132 |
| FBHelper() | * FBHelper.cpp – 8, 26, 38, 53 * Godialog.cpp - 14 |
| Void save(QString ,int) | * FBHelper.cpp – 26 * Godialog.cpp - 67 |
| UI::DifficultyDialog(QWidget \*) | * difficultydialog.cpp – 8, 22, 30, 39 * mainWindow.cpp - 15 |

|  |  |  |
| --- | --- | --- |
| **Type** | **Protected** | **#Where** |
| Void | keyPressEvent(QKeyEvent\*) | mainWindow.cpp - 62 |
| Void | drawBackground(QPainter\*, consts QRectF) | mainScene.cpp - 43 |
| Virtual void | closeEvent(QCloseEvent\*) | Difficultydialog.cpp – 30  Godailog.cpp - 38 |

FB - Source Code (.h/.cpp)

//This is the Main.cpp file.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include "uicontroller.h"

#include <QApplication>

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

new UIController();

return a.exec();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef DIFFICULTYDIALOG\_H

#define DIFFICULTYDIALOG\_H

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <QDialog>

#include <gamelevel.h>

/\*\* Opens the difficulty dialog box and gives speed and separation options.

\*

\*/

namespace Ui {

class DifficultyDialog;

}

class DifficultyDialog : public QDialog

{

Q\_OBJECT

private:

Ui::DifficultyDialog \*ui;

public:

//The main constructor

explicit DifficultyDialog(QWidget \*parent = 0);

//The main destructor

~DifficultyDialog();

protected:

//Handling the default close button of this dialog

virtual void *closeEvent*(QCloseEvent\*);

signals:

//The start-game signal

void startGame();

private slots:

//Start button handler

void on\_startButton\_clicked();

};

#endif // DIFFICULTYDIALOG\_H

// difficultydialog.cpp file

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include "difficultydialog.h"

#include "ui\_difficultydialog.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*

\* @brief DifficultyDialog::DifficultyDialog

\* @param parent

\*/

DifficultyDialog::DifficultyDialog(QWidget \*parent) :

QDialog(parent),

ui(new Ui::DifficultyDialog)

{

ui->setupUi(this);

ui->dComboBox->addItem(GameLevel::EASY.getLevelN());

ui->dComboBox->addItem(GameLevel::MEDIUM.getLevelN());

ui->dComboBox->addItem(GameLevel::HARD.getLevelN());

}

/\*\*

\* @brief DifficultyDialog::~DifficultyDialog

\*/

DifficultyDialog::~DifficultyDialog()

{

delete ui;

}

/\*\*

\* @brief DifficultyDialog::closeEvent

\*/

void DifficultyDialog::*closeEvent*(QCloseEvent \*)

{

exit(EXIT\_SUCCESS);

}

/\*\*

\* @brief DifficultyDialog::on\_startButton\_clicked

\*/

void DifficultyDialog::on\_startButton\_clicked()

{

GameLevel::setLevelN(ui->dComboBox->currentText());

emit startGame();

}

//FBHelper Header File

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef FBHELPER\_H

#define FBHELPER\_H

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <QTcpSocket>

/\*\* The Flapping Bird Helper class. \*/

class FBHelper : public QObject

{

/\*\*

\*

\* This class collects data for the server application that

\* runs conjointly with with class

\*/

Q\_OBJECT

private:

/\*\*

\* @brief tcpSocket

\*/

QTcpSocket\* tcpSocket;

/\*\*

\* @brief userName

\*/

QString userName;

/\*\*

\* @brief score

\*/

QString score;

public:

/\*The default constructor\*/

FBHelper();

/\* Save a record the database\*/

void save(QString userName, int score);

private slots:

/\*\*

\* @brief sendData

\*/

void sendData();

/\*\*

\* @brief processError

\*/

void processError();

};

#endif // FBHELPER\_H

//FBHelper.cpp file

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include "fbhelper.h"

#include <QtNetwork>

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*

\* @brief FBHelper::FBHelper

\*/

FBHelper::FBHelper()

{

tcpSocket = new QTcpSocket();

connect(tcpSocket, SIGNAL(connected()), this, SLOT(sendData()));

connect(tcpSocket, SIGNAL(disconnected()),

this, SLOT(processError()));

connect(tcpSocket, SIGNAL(error(QAbstractSocket::SocketError)),

this, SLOT(processError()));

}

/\*\*

\* @brief FBHelper::save

\* @param userName

\* @param score

\* @return

\*/

void FBHelper::save(QString userName, int score)

{

this->userName = userName;

this->score = QString::number(score);

tcpSocket->connectToHost(QHostAddress::LocalHost, 6178);

}

/\*\*

\* @brief FBHelper::sendData

\*/

void FBHelper::sendData()

{

QByteArray block;

QDataStream out(&block, QIODevice::WriteOnly);

out << userName << score;

tcpSocket->write(block);

tcpSocket->close();

}

/\*\*

\* @brief FBHelper::processError

\*/

void FBHelper::processError()

{

qDebug() << tcpSocket->errorString();

tcpSocket->close();

}

//gameLevel header file

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef GAMELEVEL\_H

#define GAMELEVEL\_H

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <QString>

/\*\* This class is implemented in form of combination of singleton and type-safe enum design patterns.

\* It is used for managing the difficuty of a game

\* , and the difficulty is defined base on the bird size and the flower speed.

\* The class is simultaneously accessed by all UI components.

\* @brief The GameLevel class

\*/

class GameLevel

{

public:

/\*The easy level object\*/

static const GameLevel EASY;

/\*The medium level object\*/

static const GameLevel MEDIUM;

/\*The hard level object\*/

static const GameLevel HARD;

private:

/\*The number of difficulties\*/

static const int NUMBER\_OF\_LEVEL = 3;

/\*The management array of all levels\*/

static const GameLevel levels[];

/\* The bird image scalars for easy, medium, and hard levels \*/

static const float BIRD\_PIC\_SCALARS[];

/\* The flower speed in miliseconds for easy, medium, and hard levels \*/

static const int FLOWER\_SPEEDS[];

/\*The singleton object\*/

static GameLevel gLevel;

/\*The name of a difficulty level\*/

QString levelN;

/\*The id of a difficulty level\*/

int id;

/\*The constructor of the singleton object\*/

GameLevel(QString levelN = "", int id = 0);

public:

/\*Auto generated getters and setters\*/

QString getLevelN() const;

static void setLevelN(const QString &value);

int getId() const;

static void setId(int value);

/\*The singleton design method\*/

static GameLevel getInstance();

/\*This function returns the bird picture scalar base on the current game difficulty\*/

static float getBirdPicScalar();

/\*This function returns the flower speed base on the current game difficulty\*/

static int getFlowerSpeed();

};

#endif // GAMELEVEL\_H

//Gamelevel.cpp file

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include "gamelevel.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*

\* Reference to the function declaration

\* @brief GameLevel::EASY

\*/

GameLevel const GameLevel::EASY("Easy", 0);

/\*\*

\* @brief GameLevel::MEDIUM

\*/

GameLevel const GameLevel::MEDIUM("Medium", 1);

/\*\*

\* @brief GameLevel::HARD

\*/

GameLevel const GameLevel::HARD("Hard", 2);

/\*\*

\* The singleton object

\* @brief GameLevel::gLevel

\*/

GameLevel GameLevel::gLevel;

/\*Reference to the constant declaration\*/

GameLevel const GameLevel::levels[] = { EASY, MEDIUM, HARD};

/\*Reference to the constant declaration\*/

const float GameLevel::BIRD\_PIC\_SCALARS[] = {0.15, 0.18, 0.2};

/\*Reference to the constant declaration\*/

const int GameLevel::FLOWER\_SPEEDS[] = {15, 10, 5};

/\*\*

\* @brief GameLevel::getLevelN

\* @return

\*/

QString GameLevel::getLevelN() const

{

return levelN;

}

/\*\*

\* @brief GameLevel::setLevelN

\* @param value

\*/

void GameLevel::setLevelN(const QString &value)

{

for (int i = 0; i < NUMBER\_OF\_LEVEL; ++i) {

if (value == levels[i].levelN){

gLevel.levelN = value;

gLevel.id = levels[i].id;

return;

}

}

}

/\*\*

\* @brief GameLevel::getId

\* @return

\*/

int GameLevel::getId() const

{

return id;

}

/\*\*

\* @brief GameLevel::setId

\* @param value

\*/

void GameLevel::setId(int value)

{

for (int i = 0; i < NUMBER\_OF\_LEVEL; ++i) {

if (value == levels[i].id){

gLevel.levelN = levels[i].levelN;

gLevel.id = value;

return;

}

}

}

/\*\*

\* The singleton method

\* @brief GameLevel::getGLevel

\* @return

\*/

GameLevel GameLevel::getInstance()

{

return gLevel;

}

/\*\*

\* @brief GameLevel::getBirdPicScalar

\* @return

\*/

float GameLevel::getBirdPicScalar()

{

return BIRD\_PIC\_SCALARS[gLevel.getId()];

}

/\*\*

\* @brief GameLevel::getFlowerSpeed

\* @return

\*/

int GameLevel::getFlowerSpeed()

{

return FLOWER\_SPEEDS[gLevel.getId()];

}

/\*\*

\* Reference to the function declaration

\* @brief GameLevel::GameLevel

\* @param levelN

\* @param id

\*/

GameLevel::GameLevel(QString levelN, int id)

{

this->levelN = levelN;

this->id = id;

}

//godialog header file

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef GODIALOG\_H

#define GODIALOG\_H

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <QDialog>

#include "fbhelper.h"

#include <QDebug>

namespace Ui {

class GODialog;

}

/\*\* The game-over dialog.

\* It has options to save to the RCC server

\*/

class GODialog : public QDialog

{

Q\_OBJECT

private:

Ui::GODialog \*ui;

FBHelper\* fbHelper;

public:

//The main constructor

explicit GODialog(QWidget \*parent = 0);

//The main destructor

~GODialog();

//This function pupolate the UI with the final score

void populateUI(int score);

protected:

//Handling the default close button of this dialog

virtual void *closeEvent*(QCloseEvent\*);

signals:

//Start a new game

void restart();

private slots:

//Quit button handler

void on\_quitButton\_clicked();

//Restart button handler

void on\_restartButton\_clicked();

//Save and Restart button handler

void on\_sRestartButton\_clicked();

};

#endif // GODIALOG\_H

//godialog.cpp file

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include "godialog.h"

#include "ui\_godialog.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*

\* @brief GODialog::GODialog

\* @param parent

\*/

GODialog::GODialog(QWidget \*parent) :

QDialog(parent),

ui(new Ui::GODialog)

{

ui->setupUi(this);

fbHelper = new FBHelper();

}

/\*\*

\* @brief GODialog::~GODialog

\*/

GODialog::~GODialog()

{

delete ui;

}

/\*\*

\* @brief GODialog::show

\* @param score

\*/

void GODialog::populateUI(int score)

{

ui->scoreDisplay->display(score);

show();

}

/\*\*

\* @brief GODialog::closeEvent

\*/

void GODialog::*closeEvent*(QCloseEvent \*)

{

exit(EXIT\_SUCCESS);

}

/\*\*

\* @brief GODialog::on\_quitButton\_clicked

\*/

void GODialog::on\_quitButton\_clicked()

{

exit(0);

}

/\*\*

\* @brief GODialog::on\_restartButton\_clicked

\*/

void GODialog::on\_restartButton\_clicked()

{

hide();

emit restart();

}

/\*\*

\* @brief GODialog::on\_sRestartButton\_clicked

\*/

void GODialog::on\_sRestartButton\_clicked()

{

hide();

fbHelper->save(ui->nameLineEdit->text(),ui->scoreDisplay->value());

emit restart();

}

//MainScene header file

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef MAINSCENE\_H

#define MAINSCENE\_H

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <QGraphicsScene>

#include <QtGui>

#include <QGraphicsPixmapItem>

#include <ctime>

#include <QLabel>

#include <QGraphicsProxyWidget>

#include <gamelevel.h>

#include <cmath>

/\*\* Authors: Hoa Huynh, Abel Salazar, Tiffany Pan

\*

\* @brief The MainScene class

\* This is the main graphics scene of the game

\* It is used to manage all the moving items (the birds and flowers) as well as the background

\*/

class MainScene : public QGraphicsScene

{

private:

/\*Path to the back ground image\*/

const QString BG\_FILE\_NAME = ":/Images/BackGround.png";

/\*Path to the upward flower image\*/

const QString UF\_FILE\_NAME = ":/Images/UpwardFlower.png";

/\*Path to the downward flower image\*/

const QString DF\_FILE\_NAME = ":/Images/DownwardFlower.png";

/\*Path to the flappig bird image\*/

const QString FB\_FILE\_NAME = ":/Images/FlappingBird.gif";

/\*The minimum height of a flower in the scene\*/

const short MIN\_FLOWER\_HEIGHT = 150;

/\*The maximum height of a flower in the scene\*/

const short MAX\_FLOWER\_HEIGHT = 200;

/\*The angle of the falling bird \*/

const qreal FREE\_FALL\_ANGLE = 90;

/\*The height that the bird will fall down after a certain period of time\*/

const qreal FREE\_FALL\_DIST= 30;

/\*The height that the bird will fly up after a key-press\*/

const qreal FLY\_UP\_DIST= 60;

/\*The Background Image Object\*/

QImage bgImage;

/\*The Upward Flower Image Object\*/

QImage upFlowerIm;

/\*The Downward Flower Image Object\*/

QImage downFlowerIm;

/\*This object helps to create flowers in the scene\*/

QGraphicsPixmapItem \*flower;

/\*A vetors of all available flowers in the scene\*/

QVector<QGraphicsPixmapItem \*> flowers;

/\*This object is used for controlling the flap of the bird\*/

QMovie \*birdMovie;

/\*This graphics proxy object is used for managing the bird in the main scene\*/

QGraphicsProxyWidget \*bird = NULL;

/\*Check if the bird is in the free-fall mode\*/

bool isFreeFall;

/\*Check if the bird is in the fly-up mode\*/

bool isFlyUp;

/\*The current vertical position of the bird\*/

qreal bXPos;

public:

/\*\*

\* The main constructor is uesed for initating all components in this scene

\*/

MainScene(QObject \*parent = 0);

protected:

/\*\*

\* Overriding function from QGraphicsScene

\* This function will draw the background being one of three layers of the scene

\* @brief drawBackground

\* @param painter

\* @param rect

\*/

void drawBackground(QPainter \*painter, const QRectF &rect);

private:

/\*Generate a scale for the height of a flower in a reasonable range\*/

short getHeightScale();

/\*Update lists of flowers for processing theirs movements\*/

void addNewFlower(QGraphicsPixmapItem \*flower);

/\*Deleting the flower that passed the scence\*/

void deletePFlower(QGraphicsPixmapItem \*flower);

/\*Creating a bird and putting it on the main scene\*/

void createABird(const QSize& birdSize);

/\*Check if the bird has just passed a column of flowers\*/

void updateScore(QGraphicsPixmapItem \*flower);

signals:

public slots:

public:

/\*Add moving flowers to the scene\*/

void createFlowers();

/\*Move flowers to the scene\*/

void moveFlowers();

/\*Locate the bird to start the game\*/

void play();

/\*Apply the gravity on the bird\*/

void freeFallBird();

/\*User controls the bird to fly up\*/

void flyUpBird();

/\*Check for the collision between the bird and a flower\*/

bool hasCollision();

/\*This function will notify the main window in case of a collision happens\*/

void checkForCollision();

/\*Clean-up all components to prepare for a new game\*/

void restartScene();

};

#endif // MAINSCENE\_H

//MainScene.cpp

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include "mainscene.h"

#include "mainwindow.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*

\* Reference to the declaration of this constructor

\* @brief MainScene::MainScene

\* @param parent

\*/

MainScene::MainScene(QObject \*parent) : QGraphicsScene(parent)

{

/\*Seeding the value for a random\*/

qsrand(time(NULL));

/\*Loading a background picture into the equivalent image object\*/

bgImage.load(BG\_FILE\_NAME);

/\*Loading a upper flower picture into the equivalent image object\*/

upFlowerIm.load(UF\_FILE\_NAME);

/\*Loading a bottom flower picture into the equivalent image object\*/

downFlowerIm.load(DF\_FILE\_NAME);

/\*Loading a bird movie into the equivalent movie object\*/

birdMovie = new QMovie(FB\_FILE\_NAME);

/\*Creating a huge bird before the game is started\*/

createABird(QSize(static\_cast<MainWindow\*>(this->parent())->geometry().width()

,static\_cast<MainWindow\*>(this->parent())->geometry().height()));

/\*At first the bird is not in either free-fall or fly-up mode\*/

isFreeFall = false;

isFlyUp = false;

}

/\*\*

\* Reference to the declaration of this function

\* @brief MainScene::drawBackground

\* @param painter

\* @param rect

\*/

void MainScene::drawBackground(QPainter \*painter, const QRectF &rect)

{

painter->drawImage(sceneRect(),bgImage);

}

/\*\*

\* Reference to the declaration of this function

\* @brief MainScene::getHeightScale

\* @return

\*/

short MainScene::getHeightScale()

{

return MIN\_FLOWER\_HEIGHT + qrand()%(MAX\_FLOWER\_HEIGHT - MIN\_FLOWER\_HEIGHT + 1);

}

/\*\*

\* Reference to the declaration of this function

\* @brief MainScene::addNewFlower

\* @param flower

\*/

void MainScene::addNewFlower(QGraphicsPixmapItem \*flower)

{

this->addItem(flower);

flowers.append(flower);

}

/\*\*

\* Reference to the declaration of this function

\* @brief MainScene::deletePFlower

\* @param flower

\*/

void MainScene::deletePFlower(QGraphicsPixmapItem \*flower)

{

if (flower->pos().x() < this->sceneRect().bottomLeft().x()){

this->removeItem(flower);

flowers.removeOne(flower);

}

}

/\*\*

\* Reference to the declaration of this function

\* @brief MainScene::createABird

\*/

void MainScene::createABird(const QSize& birdSize)

{

//This label contains the animated bird gif file

QLabel \*birdLabel = new QLabel();

//Let the bird image occupy the area being proportional to birdSize

birdMovie->stop();

birdMovie->setScaledSize(birdSize);

birdLabel->setMovie(birdMovie);

//Make the gif look transparent on the main scene

birdLabel->setAttribute(Qt::WA\_TranslucentBackground,true);

birdMovie->start();

//Using a graphics proxy object to manage the bird in the main scene

bird = addWidget(birdLabel);

}

/\*\*

\* Reference to the declaration of this function

\* @brief MainScene::updateScore

\* @param flower

\*/

void MainScene::updateScore(QGraphicsPixmapItem \*flower)

{

if (flower->pos().x() == bXPos){

static\_cast<MainWindow\*>(this->parent())->updateScore();

}

}

/\*\*

\* Reference to the declaration of this function

\* @brief MainScene::createFlowers

\*/

void MainScene::createFlowers()

{

//Creating a top flower

flower = new QGraphicsPixmapItem(QPixmap::fromImage(upFlowerIm.scaledToHeight(getHeightScale())));

flower->setPos(sceneRect().bottomRight().x() - fmod(qrand(),flower->sceneBoundingRect().width()\*2)

, sceneRect().bottom() - flower->sceneBoundingRect().height());

addNewFlower(flower);

//Creating a bottom flower

flower = new QGraphicsPixmapItem(QPixmap::fromImage(downFlowerIm.scaledToHeight(getHeightScale())));

flower->setPos(sceneRect().topRight().x() - fmod(qrand(),flower->sceneBoundingRect().width()\*2)

, sceneRect().top());

addNewFlower(flower);

}

/\*\*

\* Reference to the declaration of this function

\* @brief MainScene::moveFlowers

\*/

void MainScene::moveFlowers()

{

for (int i = 0; i < flowers.size(); i++) {

//Moving flowers from right to left

flowers[i]->setPos(flowers[i]->pos().x() - 1,flowers[i]->pos().y());

//Update the score when the bird pass a column of flowers

updateScore(flowers[i]);

//Remove the one that was out of sight

deletePFlower(flowers[i]);

}

}

/\*\*

\* Reference to the declaration of this function

\* @brief MainScene::play

\*/

void MainScene::play()

{

//Creating a small bird after deleting the old huge one

removeItem(bird);

createABird(QSize(bird->geometry().width()\*GameLevel::getInstance().getBirdPicScalar()

,bird->geometry().height()\*GameLevel::getInstance().getBirdPicScalar()));

/\*Locating the vertical position of the bird for scoring\*/

bXPos = sceneRect().bottomLeft().x() + bird->geometry().width();

bird->setPos(bXPos, sceneRect().center().y());

}

/\*\*

\* Reference to the declaration of this function

\* @brief MainScene::freeFallBird

\*/

void MainScene::freeFallBird()

{

isFlyUp = false;

if (!isFreeFall){ //Let the bird start falling down

birdMovie->stop();

bird->setRotation(FREE\_FALL\_ANGLE);

bird->setPos(bird->pos().x() + bird->geometry().width()

,bird->pos().y());

isFreeFall = true;

}else{// In the free-fall mode

//The bird should be always in the main scene

if (bird->pos().y() < (sceneRect().bottom() - bird->geometry().height())){

bird->setPos(bird->pos().x(),bird->pos().y() + FREE\_FALL\_DIST);

}else{// Game over when the bird hit the ground

emit static\_cast<MainWindow\*>(this->parent())->processCollision();

}

}

//Check if the bird hits a flower

checkForCollision();

}

/\*\*

\* Reference to the declaration of this function

\* @brief MainScene::flyUpBird

\*/

void MainScene::flyUpBird()

{

isFreeFall = false;

if (!isFlyUp){//Let the bird start flying up

birdMovie->start();

bird->setRotation(0);

bird->setPos(bird->pos().x() - bird->geometry().width()

,bird->pos().y());

isFlyUp = true;

}else{// In the fly-up mode

//Keep the bird in the main scene

if (bird->pos().y() > sceneRect().top() ){

bird->setPos(bird->pos().x(),bird->pos().y() - FREE\_FALL\_DIST);

}

}

//Check if the bird hits a flower

checkForCollision();

}

/\*\*

\* Reference to the declaration of this function

\* @brief MainScene::hasCollision

\* @return

\*/

bool MainScene::hasCollision()

{

if (!flowers.isEmpty() && (bird != NULL)){

for (int i = 0; i < flowers.size(); i++) {

if (bird->collidesWithItem(flowers[i])){

return true;

}

}

}

return false;

}

/\*\*

\* Reference to the declaration of this function

\* @brief MainScene::checkForCollision

\*/

void MainScene::checkForCollision()

{

if (hasCollision()){

//Let the bird fall down completely

bird->setPos(bird->pos().x(),sceneRect().bottom() - bird->geometry().height());

//Notify the UIController through the Main Window

emit static\_cast<MainWindow\*>(this->parent())->processCollision();

}

}

/\*\*

\* @brief MainScene::restartScene

\*/

void MainScene::restartScene()

{

//Clean up all the flowers and the bird

QVector<QGraphicsPixmapItem \*>::Iterator iFlower;

for (iFlower = flowers.begin(); iFlower != flowers.end(); iFlower++){

removeItem(\*iFlower);

}

flowers.clear();

removeItem(bird);

/\*Creating a huge bird before the game is started\*/

createABird(QSize(static\_cast<MainWindow\*>(this->parent())->geometry().width()

,static\_cast<MainWindow\*>(this->parent())->geometry().height()));

/\*At first the bird is not in either free-fall or fly-up mode\*/

isFreeFall = false;

isFlyUp = false;

}

//mainwindow header file

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <QMainWindow>

#include <QMediaPlayer>

#include "mainscene.h"

namespace Ui {

class MainWindow;

}

/\*\*

\*

\*@brief This class handles the creating of the objects that are later

\* controlled by the MainScene class

\*

\*/

class MainWindow : public QMainWindow

{

Q\_OBJECT

private:

/\*Path to the score sound track\*/

const QString SCORE\_S\_FILE\_NAME = "qrc:/SoundTracks/Score.mp3";

/\*The number of flowers the player has to pass to be leveled up\*/

const int NUMBER\_OF\_FLOWERS\_PER\_LEVEL = 10;

/\*Main UI\*/

Ui::MainWindow \*ui;

/\*The fundamental graphics scene of the game\*/

MainScene \*mainScene;

/\*The current score of a game\*/

int crScore = 0;

/\*The media player of the scoring sound\*/

QMediaPlayer \*scoreMs;

public:

/\*The primary constructor of this window\*/

explicit MainWindow(QWidget \*parent = 0);

/\*Main window destructor\*/

~MainWindow();

protected:

/\*This function waits for a key press event that belongs to the main window\*/

void keyPressEvent(QKeyEvent\* event);

signals:

/\*This function is called when the Space Bar key was pressed\*/

void pressSpaceKey();

/\*This function is called when the bird collides with a flower\*/

void processCollision();

/\*This function is called when the bird passed NUMBER\_OF\_FLOWERS\_PER\_LEVEL flowers\*/

void levelUp();

public slots:

/\*This function asks main scene to move flowers when the GUI thread is running\*/

void moveFlowers();

/\*This function asks main scene to apply the gravity on the bird when the GUI thread is running\*/

void freeFallBird();

public:

/\*This function asks main scene to create new flowers when the GUI thread is running\*/

void createFlowers();

/\*This function asks main scene to start the game when the GUI thread is running\*/

void play();

/\*This function asks main scene to let the bird fly-up when the GUI thread is running\*/

void flyUpBird();

/\*Increase the socre when the bird passes a column of flower\*/

void updateScore();

/\*The auto-generated getter of the current score\*/

int getCrScore() const;

/\*Clean up all the components to prepare for a new game\*/

void restartUI();

};

#endif // MAINWINDOW\_H

//MainWindow.cpp

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include "mainwindow.h"

#include "ui\_mainwindow.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*Reference to the function declaration\*/

int MainWindow::getCrScore() const

{

return crScore;

}

/\*\*

\* @brief MainWindow::restartUI

\*/

void MainWindow::restartUI()

{

//Ask main scene to reset its components

mainScene->restartScene();

//Reset the score

crScore = 0;

ui->lcdScore->display(crScore);

ui->lcdScore->hide();

}

/\*Reference to the function declaration\*/

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

//Set a permanent size for this main window and its graphics view

setFixedSize(geometry().width(),geometry().height());

ui->graphicsView->setFixedSize(ui->graphicsView->geometry().width()

,ui->graphicsView->geometry().height());

//Setting up the main scene in this window

mainScene = new MainScene(this);

mainScene->setSceneRect(ui->graphicsView->geometry().topLeft().x()

,ui->graphicsView->geometry().topLeft().y()

,ui->graphicsView->geometry().width()

,ui->graphicsView->geometry().height());

ui->graphicsView->setScene(mainScene);

//Setup the score panel before getting the game started

ui->lcdScore->setStyleSheet("QLCDNumber {color: blue;}");

ui->lcdScore->hide();

//Setup the scoring sound

scoreMs = new QMediaPlayer();

scoreMs->setMedia(QUrl(SCORE\_S\_FILE\_NAME));

}

/\*Reference to the function declaration\*/

MainWindow::~MainWindow()

{

delete ui;

}

/\*Reference to the function declaration\*/

void MainWindow::keyPressEvent(QKeyEvent \*event)

{

//Check if the space bar was pressed

if (event->key() == Qt::Key\_Space){

emit pressSpaceKey();

}

}

/\*Reference to the function declaration\*/

void MainWindow::updateScore()

{

ui->lcdScore->display(++crScore);

scoreMs->play();

if ((crScore%NUMBER\_OF\_FLOWERS\_PER\_LEVEL) == 0){

emit levelUp();

}

}

/\*Reference to the function declaration\*/

void MainWindow::createFlowers()

{

mainScene->createFlowers();

}

/\*Reference to the function declaration\*/

void MainWindow::play()

{

//Start scoring the game

ui->lcdScore->show();

//Ask the main scene to animate the its components

mainScene->play();

}

/\*Reference to the function declaration\*/

void MainWindow::freeFallBird()

{

mainScene->freeFallBird();

}

/\*Reference to the function declaration\*/

void MainWindow::flyUpBird()

{

mainScene->flyUpBird();

}

/\*Reference to the function declaration\*/

void MainWindow::moveFlowers()

{

mainScene->moveFlowers();

}

//Uicontroller header file

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef UICONTROLLER\_H

#define UICONTROLLER\_H

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <QObject>

#include <QTimer>

#include "mainwindow.h"

#include <QMediaPlayer>

#include "difficultydialog.h"

#include "godialog.h"

/\*\*

\* @brief The SceneController class

\* This is the primary controller of the GUI.

\* It controls all the behaviors on the main window and its scene

\*/

class UIController : public QObject

{

Q\_OBJECT

private:

/\*The minimum miliseconds for creating flowers\*/

const int MIN\_TIME\_IN\_MIL = 1500;

/\*The maximum miliseconds for creating flowers\*/

const int MAX\_TIME\_IN\_MIL = 5000;

/\*The default miliseconds for the bird fall down\*/

const int BIRD\_FALLING\_SPEED = 200;

/\*Path to the back ground sound track\*/

const QString BG\_S\_FILE\_NAME = "qrc:/SoundTracks/BGround.mp3";

/\*Path to the game-over sound track\*/

const QString GO\_S\_FILE\_NAME = "qrc:/SoundTracks/EndGame.mp3";

/\*The main window of Flappy Bird\*/

MainWindow \*mainWindow;

/\*The start menu of a game\*/

DifficultyDialog \*dDialog;

/\*The game-over dialog\*/

GODialog \*gODialog;

/\*This timer is used for controlling the creation of flowers\*/

QTimer \*cFlowerTimer;

/\*This timer is used for controlling the movement of flowers\*/

QTimer \*mFlowerTimer;

/\*This timer is used for controlling the gravity on the bird\*/

QTimer \*gBirdTimer;

/\*The media player of the back ground music\*/

QMediaPlayer \*bgMusic;

/\*The media player of the game-over sound\*/

QMediaPlayer \*endMs;

/\*Identifying the current game mode\*/

bool isGameStarted;

public:

/\*The constructor of this main UI controller\*/

explicit UIController(QObject \*parent = 0);

signals:

public slots:

/\*Randomly ask for new flowers to add to the main scene\*/

void createFlowers();

/\*Processing the items in the scene according to the key press\*/

void processSpaceKeyPress();

/\*This function is called when the bird collides with a flower\*/

void processCollision();

/\*This function is called when the back ground music was stoped\*/

void stateChanged(QMediaPlayer::State newState);

/\*This function is called after the user has chosen the difficulty \*/

void startGame();

/\*This function is called after the user has chosen to restart the game \*/

void restart();

/\*This function handles the level-up event from the main window\*/

void levelUp();

private:

/\*Connect all the subsystems to this UI controller\*/

void connectSystems();

};

#endif // UICONTROLLER\_H

//uicontroller.cpp

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include "uicontroller.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*

\* Reference to the function declaration

\* @brief UIController::UIController

\* @param parent

\*/

UIController::UIController(QObject \*parent) : QObject(parent)

{

//Initialize the main window with its components

mainWindow = new MainWindow();

mainWindow->show();

//Intialized the start game menu

dDialog = new DifficultyDialog(mainWindow);

//Intialized the end game menu

gODialog = new GODialog(mainWindow);

//Seeding the value for a random

qsrand(time(NULL));

//Initializing all the timers

cFlowerTimer = new QTimer(this);

mFlowerTimer = new QTimer(this);

gBirdTimer = new QTimer(this);

//Waiting for a user to start the game

isGameStarted = false;

//Setup the multimedia players

bgMusic = new QMediaPlayer();

endMs = new QMediaPlayer();

bgMusic->setMedia(QUrl(BG\_S\_FILE\_NAME));

endMs->setMedia(QUrl(GO\_S\_FILE\_NAME));

//Connect all the components together

connectSystems();

}

/\*\*

\* Reference to the function declaration

\* @brief UIController::createNewFLowers

\*/

void UIController::createFlowers()

{

//Randomize the time for creating new flowers

cFlowerTimer->stop();

mainWindow->createFlowers();

cFlowerTimer->start(MIN\_TIME\_IN\_MIL + (qrand()%(MAX\_TIME\_IN\_MIL - MIN\_TIME\_IN\_MIL + 1)));

}

/\*\*

\* Reference to the function declaration

\* @brief UIController::processKeyPress

\*/

void UIController::processSpaceKeyPress()

{

if (!isGameStarted){ //Starting the game at the first time

isGameStarted = true;

//Populate the difficulty dialog UI

dDialog->show();

//Using SIGNAL-SLOT to replay the back ground music

connect(bgMusic,SIGNAL(stateChanged(QMediaPlayer::State)),this,SLOT(stateChanged(QMediaPlayer::State)));

//Play the back ground music

bgMusic->play();

}else { //Processing the bird movements

//Stop free-fall for fly-up

gBirdTimer->stop();

mainWindow->flyUpBird();

//Start free-fall again

gBirdTimer->start(BIRD\_FALLING\_SPEED);

}

}

/\*\*

\* Reference to the function declaration

\* @brief UIController::processCollision

\*/

void UIController::processCollision()

{

//Stop creating and moving flowers

cFlowerTimer->stop();

mFlowerTimer->stop();

//Stop letting the bird fall down

gBirdTimer->stop();

/\*Process media affects\*/

//Stop using SIGNAL-SLOT to replay the back ground music

disconnect(bgMusic,SIGNAL(stateChanged(QMediaPlayer::State)),this,SLOT(stateChanged(QMediaPlayer::State)));

bgMusic->stop();

endMs->play();

//Populate the game-over UI

gODialog->populateUI(mainWindow->getCrScore());

}

/\*\*

\* Reference to the function declaration

\* @brief UIController::stateChanged

\* @param newState

\*/

void UIController::stateChanged(QMediaPlayer::State newState)

{

if (newState == QMediaPlayer::StoppedState){

bgMusic->play(); // Replay the music

}

}

/\*\*

\* Reference to the function declaration

\* @brief UIController::startGame

\*/

void UIController::startGame()

{

//Hide the difficulty dialog

dDialog->hide();

//Let flowers appear and move in the scene

cFlowerTimer->start(MIN\_TIME\_IN\_MIL);

mFlowerTimer->start(GameLevel::getInstance().getFlowerSpeed());

//Let the bird free fall

gBirdTimer->start(BIRD\_FALLING\_SPEED);

//Put the bird in the right position to start the game

mainWindow->play();

}

/\*\*

\* Reference to the function declaration

\* @brief UIController::restart

\*/

void UIController::restart()

{

mainWindow->restartUI();

isGameStarted = false;

}

/\*\*

\* Reference to the function declaration

\* @brief UIController::levelUp

\*/

void UIController::levelUp()

{

int crTimer = mFlowerTimer->interval();

if ((crTimer--) > 1){

mFlowerTimer->setInterval(crTimer);

}

}

/\*\*

\* @brief UIController::connectSystems

\*/

void UIController::connectSystems()

{

//Connecting timer to the behaviors of flowers in the main scene

connect(cFlowerTimer,SIGNAL(timeout()),this,SLOT(createFlowers()));

connect(mFlowerTimer,SIGNAL(timeout()),mainWindow,SLOT(moveFlowers()));

//Connecting timer to the behaviors of the bird

connect(gBirdTimer,SIGNAL(timeout()),mainWindow,SLOT(freeFallBird()));

//Getting connect to the key event of the main window

connect(mainWindow,SIGNAL(pressSpaceKey()),this,SLOT(processSpaceKeyPress()));

//Getting a notify from the main window for a collision

connect(mainWindow,SIGNAL(processCollision()),this,SLOT(processCollision()));

//Register with the main window to receive a level-up signal

connect(mainWindow,SIGNAL(levelUp()),this,SLOT(levelUp()));

//Register with the difficulty dialog to receive a start-game signal

connect(dDialog,SIGNAL(startGame()),this,SLOT(startGame()));

//Register with the game-over dialog to receive a restart-game signal

connect(gODialog,SIGNAL(restart()),this,SLOT(restart()));

}

FB - Source Code for UI’s

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//difficultydialog.ui

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>DifficultyDialog</class>

<widget class="QDialog" name="DifficultyDialog">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>400</width>

<height>225</height>

</rect>

</property>

<property name="windowTitle">

<string>StartMenu</string>

</property>

<widget class="QLabel" name="titleLabel">

<property name="geometry">

<rect>

<x>100</x>

<y>20</y>

<width>191</width>

<height>31</height>

</rect>

</property>

<property name="font">

<font>

<family>Arial</family>

<pointsize>16</pointsize>

</font>

</property>

<property name="text">

<string>Flappy Bird</string>

</property>

<property name="alignment">

<set>Qt::AlignCenter</set>

</property>

</widget>

<widget class="QWidget" name="horizontalLayoutWidget">

<property name="geometry">

<rect>

<x>70</x>

<y>80</y>

<width>241</width>

<height>51</height>

</rect>

</property>

<layout class="QHBoxLayout" name="horizontalLayout">

<item>

<widget class="QLabel" name="difficultyLabel">

<property name="text">

<string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p align=&quot;center&quot;&gt;Difficulty:&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>

</property>

</widget>

</item>

<item>

<widget class="QComboBox" name="dComboBox">

<property name="currentText">

<string/>

</property>

</widget>

</item>

</layout>

</widget>

<widget class="QPushButton" name="startButton">

<property name="geometry">

<rect>

<x>150</x>

<y>170</y>

<width>75</width>

<height>23</height>

</rect>

</property>

<property name="text">

<string>Start Game</string>

</property>

</widget>

</widget>

<resources/>

<connections/>

</ui>

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//godialog.ui

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>GODialog</class>

<widget class="QDialog" name="GODialog">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>400</width>

<height>242</height>

</rect>

</property>

<property name="windowTitle">

<string>EndMenu</string>

</property>

<widget class="QLabel" name="gameOverLabel">

<property name="geometry">

<rect>

<x>160</x>

<y>40</y>

<width>91</width>

<height>16</height>

</rect>

</property>

<property name="text">

<string>Game Over</string>

</property>

</widget>

<widget class="QLineEdit" name="nameLineEdit">

<property name="geometry">

<rect>

<x>170</x>

<y>140</y>

<width>113</width>

<height>20</height>

</rect>

</property>

</widget>

<widget class="QLabel" name="nameLabel">

<property name="geometry">

<rect>

<x>80</x>

<y>140</y>

<width>47</width>

<height>13</height>

</rect>

</property>

<property name="text">

<string>Name</string>

</property>

</widget>

<widget class="QLabel" name="scoreLabel">

<property name="geometry">

<rect>

<x>80</x>

<y>80</y>

<width>47</width>

<height>13</height>

</rect>

</property>

<property name="text">

<string>Score</string>

</property>

</widget>

<widget class="QLCDNumber" name="scoreDisplay">

<property name="geometry">

<rect>

<x>190</x>

<y>70</y>

<width>64</width>

<height>23</height>

</rect>

</property>

</widget>

<widget class="QPushButton" name="quitButton">

<property name="geometry">

<rect>

<x>274</x>

<y>200</y>

<width>71</width>

<height>23</height>

</rect>

</property>

<property name="text">

<string>Quit</string>

</property>

</widget>

<widget class="QPushButton" name="sRestartButton">

<property name="geometry">

<rect>

<x>90</x>

<y>200</y>

<width>81</width>

<height>23</height>

</rect>

</property>

<property name="text">

<string>Save &amp;&amp; Restart</string>

</property>

</widget>

<widget class="QPushButton" name="restartButton">

<property name="geometry">

<rect>

<x>190</x>

<y>200</y>

<width>71</width>

<height>23</height>

</rect>

</property>

<property name="text">

<string>Restart</string>

</property>

</widget>

</widget>

<resources/>

<connections/>

</ui>

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//mainwindow.ui

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>MainWindow</class>

<widget class="QMainWindow" name="MainWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>520</width>

<height>520</height>

</rect>

</property>

<property name="maximumSize">

<size>

<width>745</width>

<height>700</height>

</size>

</property>

<property name="windowTitle">

<string>FlappyBird</string>

</property>

<property name="windowOpacity">

<double>1.000000000000000</double>

</property>

<widget class="QWidget" name="centralWidget">

<widget class="QGraphicsView" name="graphicsView">

<property name="geometry">

<rect>

<x>9</x>

<y>9</y>

<width>500</width>

<height>500</height>

</rect>

</property>

<property name="sizePolicy">

<sizepolicy hsizetype="Maximum" vsizetype="Maximum">

<horstretch>0</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

<property name="maximumSize">

<size>

<width>727</width>

<height>690</height>

</size>

</property>

<property name="cursor" stdset="0">

<cursorShape>CrossCursor</cursorShape>

</property>

<property name="mouseTracking">

<bool>false</bool>

</property>

<property name="acceptDrops">

<bool>false</bool>

</property>

<property name="verticalScrollBarPolicy">

<enum>Qt::ScrollBarAlwaysOff</enum>

</property>

<property name="horizontalScrollBarPolicy">

<enum>Qt::ScrollBarAlwaysOff</enum>

</property>

<property name="sizeAdjustPolicy">

<enum>QAbstractScrollArea::AdjustToContentsOnFirstShow</enum>

</property>

</widget>

<widget class="QLCDNumber" name="lcdScore">

<property name="geometry">

<rect>

<x>250</x>

<y>220</y>

<width>121</width>

<height>81</height>

</rect>

</property>

<property name="font">

<font>

<family>Times New Roman</family>

<pointsize>14</pointsize>

<weight>75</weight>

<bold>true</bold>

</font>

</property>

<property name="autoFillBackground">

<bool>false</bool>

</property>

<property name="frameShape">

<enum>QFrame::NoFrame</enum>

</property>

<property name="lineWidth">

<number>1</number>

</property>

<property name="midLineWidth">

<number>0</number>

</property>

<property name="digitCount">

<number>3</number>

</property>

<property name="value" stdset="0">

<double>0.000000000000000</double>

</property>

</widget>

</widget>

</widget>

<layoutdefault spacing="6" margin="11"/>

<resources/>

<connections/>

</ui>