# Sommaire

[Sommaire 1](#_Toc151501257)

[Introduction 3](#_Toc151501258)

[Recommandations techniques 3](#_Toc151501259)

[Création du programme principal 3](#_Toc151501260)

[Affichage de l’écran de démarrage 3](#_Toc151501261)

[Suppression de l’écran de démarrage 4](#_Toc151501262)

[Affichage de la fenêtre principale 4](#_Toc151501263)

[Création de la fenêtre principale 4](#_Toc151501264)

[Initialisation de la fenêtre principale 5](#_Toc151501265)

[Dimensionnement de la fenêtre principale 5](#_Toc151501266)

[Initialisation de la vue graphique 5](#_Toc151501267)

[Initialisation des menus 5](#_Toc151501268)

[Chargement des paramètres du jeu 8](#_Toc151501269)

[Sauvegarde des paramètres du jeu 9](#_Toc151501270)

[Alternance entre le mode normal et plein écran 9](#_Toc151501271)

[Création de la scène d’accueil 10](#_Toc151501272)

[Création de la vue graphique 10](#_Toc151501273)

[Initialisation de la vue graphique 10](#_Toc151501274)

[Suppression des scrollbars 10](#_Toc151501275)

[Création de la scène d’accueil 11](#_Toc151501276)

[Constructeur de la scène d’accueil 11](#_Toc151501277)

[Initialisation de la scène d’accueil 11](#_Toc151501278)

[Animation des nuages 12](#_Toc151501279)

[Initialisation de l’interface de connexion 13](#_Toc151501280)

[Gestion de la connexion invité 15](#_Toc151501281)

[Suppression de l’interface de connexion 15](#_Toc151501282)

[Chargement de la scène de jeu 15](#_Toc151501283)

[Création de la scène de jeu 16](#_Toc151501284)

[Constructeur de la scène de jeu 16](#_Toc151501285)

[Initialisation de la scène de jeu 16](#_Toc151501286)

[Animation de la scène de jeu 21](#_Toc151501287)

[Création du gestionnaire de Goomba 21](#_Toc151501288)

[Constructeur de Goomba 21](#_Toc151501289)

[Animation de Goomba 22](#_Toc151501290)

[Création du gestionnaire des pièces de monnaie 23](#_Toc151501291)

[Constructeur des pièces de monnaie 23](#_Toc151501292)

[Animation des pièces de monnaie 23](#_Toc151501293)

[Création du gestionnaire des questionnaires 24](#_Toc151501294)

[Constructeur des questionnaires 24](#_Toc151501295)

[Animation des questionnaires 24](#_Toc151501296)

[Création du gestionnaire des plantes carnivores 25](#_Toc151501297)

[Constructeur des plantes carnivores 25](#_Toc151501298)

[Animation des plantes carnivores 25](#_Toc151501299)

[Création du gestionnaire des briques 26](#_Toc151501300)

[Constructeur des briques 26](#_Toc151501301)

[Animation des briques 26](#_Toc151501302)

[Création du gestionnaire des paramètres du jeu 27](#_Toc151501303)

[Initialisation du gestionnaire 27](#_Toc151501304)

[Initialisation de l’interface graphique 27](#_Toc151501305)

[Enregistrement des paramètres 29](#_Toc151501306)

[Fermeture de l’interface 29](#_Toc151501307)

[Sauvegarde des paramètres du jeu 30](#_Toc151501308)

[Chargement des paramètres du jeu 30](#_Toc151501309)

[Affichage de l’emplacement des paramètres du jeu 30](#_Toc151501310)

[Alternance entre le mode normal et plein écran 31](#_Toc151501311)

[Création du gestionnaire de son 31](#_Toc151501312)

[Initialisation du gestionnaire 31](#_Toc151501313)

[Chargement des fichiers audio 31](#_Toc151501314)

[Lecture des fichiers audio 33](#_Toc151501315)

[Utilisation de QMediaPlayList 34](#_Toc151501316)

[Téléchargement de QMediaPlayList 34](#_Toc151501317)

# Introduction

Le but de ce tutoriel est de vous apprendre à développer un jeu de Super Mario Bros en C++ - Qt – Graphics - Multimédia.

# Recommandations techniques

Pour tester les extraits de code dans ce tutoriel, vous aurez besoin des éléments suivants:

* Une bibliothèque d’interface graphique Qt:

<https://www.qt.io/>

# Création du programme principal

## Affichage de l’écran de démarrage

Résultat:



Programme C++:

//===============================================

// main.cpp

//===============================================

int **main**(int **\_argc**, char\*\* **\_argv**) {

QApplication **lApp**(*\_argc*, *\_argv*);

lApp.setWindowIcon(QIcon(":/img/icon.ico"));

QSplashScreen\* **lSplash** = new QSplashScreen;

lSplash->setPixmap(QPixmap(":/img/mariosplash.png"));

lSplash->show();

return lApp.exec();

}

//===============================================

## Suppression de l’écran de démarrage

Programme C++:

//===============================================

// main.cpp

//===============================================

int **main**(int **\_argc**, char\*\* **\_argv**) {

QApplication **lApp**(*\_argc*, *\_argv*);

lApp.setWindowIcon(QIcon(":/img/icon.ico"));

QSplashScreen\* **lSplash** = new QSplashScreen;

lSplash->setPixmap(QPixmap(":/img/mariosplash.png"));

lSplash->show();

QTimer::singleShot(500, lSplash, &QSplashScreen::close);

return lApp.exec();

}

//===============================================

## Affichage de la fenêtre principale

Programme C++:

//===============================================

// main.cpp

//===============================================

int **main**(int **\_argc**, char\*\* **\_argv**) {

QApplication **lApp**(*\_argc*, *\_argv*);

lApp.setWindowIcon(QIcon(":/img/icon.ico"));

QSplashScreen\* **lSplash** = new QSplashScreen;

lSplash->setPixmap(QPixmap(":/img/mariosplash.png"));

lSplash->show();

GMainWindow **lWindow**;

lWindow.setSize();

QTimer::singleShot(500, lSplash, &QSplashScreen::close);

QTimer::singleShot(500, &lWindow, &GMainWindow::show);

return lApp.exec();

}

//===============================================

# Création de la fenêtre principale

## Initialisation de la fenêtre principale

Programme C++:

//===============================================

// GMainWindow.cpp

//===============================================

GMainWindow::**GMainWindow**(QWidget\* **\_parent**)

: QMainWindow(*\_parent*) {

}

//===============================================

## Dimensionnement de la fenêtre principale

Programme C++:

//===============================================

// GMainWindow.cpp

//===============================================

void GMainWindow::**setSize**() {

setFixedSize(1280, 740);

int **x** = ((QGuiApplication::primaryScreen()->geometry().width() - width()) / 2);

int **y** = ((QGuiApplication::primaryScreen()->geometry().height() - height()) / 2);

move(x, y);

}

//===============================================

## Initialisation de la vue graphique

Programme C++:

//===============================================

// GMainWindow.cpp

//===============================================

GMainWindow::**GMainWindow**(QWidget\* **\_parent**)

: QMainWindow(*\_parent*) {

createActions();

createMenus();

createScene();

setCentralWidget(*m\_view*);

setWindowTitle("ReadyApp");

}

//===============================================

void GMainWindow::**createScene**() {

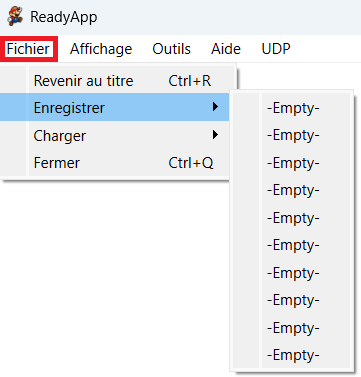
m\_view = new GView(this);

}

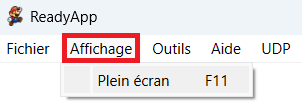
//===============================================

## Initialisation des menus

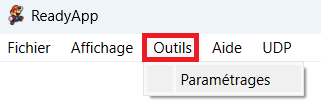
Résultat du menu Fichier:



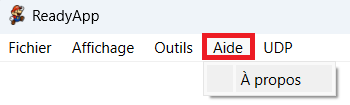
Résultat du menu Affichage:



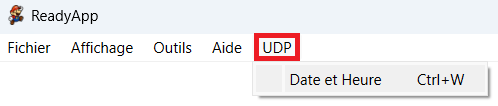
Résultat du menu Outils:



Résultat du menu Aide:



Résultat du menu UDP:



Programme C++:

//===============================================

// GMainWindow.cpp

//===============================================

GMainWindow::**GMainWindow**(QWidget\* **\_parent**)

: QMainWindow(*\_parent*) {

createActions();

createMenus();

createScene();

setCentralWidget(*m\_view*);

setWindowTitle("ReadyApp");

}

//===============================================

void GMainWindow::**createActions**() {

returnAction = new QAction(tr("&Revenir au titre"), this);

returnAction->setShortcut(tr("Ctrl+R"));

returnAction->setStatusTip(tr("Revenir à l'écran titre"));

for(int **i**=0; i< MAX\_STATE; ++i){

saveStateAction[i] = new QAction("-Empty-", this);

loadStateAction[i] = new QAction("-Empty-", this);

}

quitAction = new QAction(tr("&Fermer"), this);

quitAction->setShortcut(tr("Ctrl+Q"));

quitAction->setStatusTip(tr("Fermer"));

connect(quitAction, &QAction::triggered, this, &GMainWindow::close);

weatherStationAction = new QAction(tr("&Date et Heure"), this);

weatherStationAction->setShortcut(tr("Ctrl+w"));

weatherStationAction->setStatusTip(tr("Date et Heure"));

connect(weatherStationAction , &QAction::triggered, this, &GMainWindow::showWeather);

fullScreenAction = new QAction(tr("Plein écran"), this);

fullScreenAction->setShortcut(tr("F11"));

fullScreenAction->setStatusTip(tr("Affichage plein écran"));

connect(fullScreenAction, &QAction::triggered, this, &GMainWindow::alterScreen);

settingsAction = new QAction(tr("&Paramétrages"), this);

settingsAction->setStatusTip(tr("Modifier les paramètres du jeu"));

connect(settingsAction, &QAction::triggered, this, &GMainWindow::settings);

aboutAction = new QAction(tr("&À propos"), this);

connect(aboutAction, &QAction::triggered, this, &GMainWindow::showAbout);

}

//===============================================

void GMainWindow::**createMenus**() {

fileMenu = menuBar()->addMenu(tr("&Fichier"));

fileMenu->addAction(*returnAction*);

saveSubMenu = fileMenu->addMenu(tr("&Enregistrer"));

loadSubMenu = fileMenu->addMenu(tr("&Charger"));

for(int **i**=0; i < MAX\_STATE; i++){

saveSubMenu->addAction(saveStateAction[i]);

loadSubMenu->addAction(loadStateAction[i]);

}

fileMenu->addAction(*quitAction*);

viewMenu = menuBar()->addMenu(tr("&Affichage"));

viewMenu->addAction(*fullScreenAction*);

toolMenu = menuBar()->addMenu(tr("&Outils"));

toolMenu->setToolTip("Paramètres du jeu");

toolMenu->addAction(*settingsAction*);

helpMenu = menuBar()->addMenu(tr("&Aide"));

helpMenu->setToolTip("Info du développeur");

helpMenu->addAction(*aboutAction*);

weatherMenu = menuBar()->addMenu(tr("&UDP"));

weatherMenu->setToolTip("UDP");

weatherMenu->addAction(*weatherStationAction*);

}

//===============================================

## Chargement des paramètres du jeu

Résultat:



Programme C++:

//===============================================

// GMainWindow.cpp

//===============================================

void GMainWindow::**createActions**() {

returnAction = new QAction(tr("&Revenir au titre"), this);

returnAction->setShortcut(tr("Ctrl+R"));

returnAction->setStatusTip(tr("Revenir à l'écran titre"));

for(int **i**=0; i< MAX\_STATE; ++i){

saveStateAction[i] = new QAction("-Empty-", this);

loadStateAction[i] = new QAction("-Empty-", this);

}

quitAction = new QAction(tr("&Fermer"), this);

quitAction->setShortcut(tr("Ctrl+Q"));

quitAction->setStatusTip(tr("Fermer"));

connect(quitAction, &QAction::triggered, this, &GMainWindow::close);

weatherStationAction = new QAction(tr("&Date et Heure"), this);

weatherStationAction->setShortcut(tr("Ctrl+w"));

weatherStationAction->setStatusTip(tr("Date et Heure"));

connect(weatherStationAction , &QAction::triggered, this, &GMainWindow::showWeather);

fullScreenAction = new QAction(tr("Plein écran"), this);

fullScreenAction->setShortcut(tr("F11"));

fullScreenAction->setStatusTip(tr("Affichage plein écran"));

connect(fullScreenAction, &QAction::triggered, this, &GMainWindow::alterScreen);

settingsAction = new QAction(tr("&Paramétrages"), this);

settingsAction->setStatusTip(tr("Modifier les paramètres du jeu"));

connect(settingsAction, &QAction::triggered, this, &GMainWindow::settings);

aboutAction = new QAction(tr("&À propos"), this);

connect(aboutAction, &QAction::triggered, this, &GMainWindow::showAbout);

m\_setting = new GSettings(this);

m\_setting->readSettings();

}

//===============================================

// GSettings.cpp

//===============================================

void GSettings::**readSettings**() {

qDebug() << "GSettings::readSettings...";

}

//===============================================

## Sauvegarde des paramètres du jeu

Résultat:



Programme C++:

//===============================================

// GMainWindow.cpp

//===============================================

void GMainWindow::***closeEvent***(QCloseEvent\* **\_event**){

m\_setting->writeSettings();

\_event->accept();

}

//===============================================

// GSettings.cpp

//===============================================

void GSettings::**writeSettings**() {

qDebug() << "GSettings::writeSettings...";

}

//===============================================

## Alternance entre le mode normal et plein écran

Programme C++:

//===============================================

// GMainWindow.cpp

//===============================================

void GMainWindow::**alterScreen**() {

m\_setting->alterState();

}

//===============================================

## Création de la scène d’accueil

Programme C++:

//===============================================

// GMainWindow.cpp

//===============================================

void GMainWindow::**createScene**() {

m\_view = new GView(this);

m\_home = new GHome(*m\_view*);

}

//===============================================

# Création de la vue graphique

## Initialisation de la vue graphique

Programme C++:

//===============================================

// GView.cpp

//===============================================

GView::**GView**(QWidget\* **\_parent**)

: QGraphicsView(*\_parent*) {

}

//===============================================

## Suppression des scrollbars

Programme C++:

//===============================================

// GView.cpp

//===============================================

GView::**GView**(QWidget\* **\_parent**)

: QGraphicsView(*\_parent*) {

m\_bgm = new QMediaPlaylist;

setFixedSize(QSize(1280, 720));

setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

}

//===============================================

# Création de la scène d’accueil

## Constructeur de la scène d’accueil

Programme C++:

//===============================================

// GHome.cpp

//===============================================

GHome::**GHome**(GView\* **\_view**, QWidget\* **\_parent**)

: QGraphicsScene(*\_parent*)

, m\_view(*\_view*) {

}

//===============================================

## Initialisation de la scène d’accueil

Résultat:



Programme C++:

//===============================================

// GHome.cpp

//===============================================

GHome::**GHome**(GView\* **\_view**, QWidget\* **\_parent**)

: QGraphicsScene(*\_parent*)

, m\_view(*\_view*) {

m\_view->setViewportUpdateMode(QGraphicsView::SmartViewportUpdate);

m\_scroll = m\_view->horizontalScrollBar();

background = new GAnimation;

background->setPixmap(QPixmap(":/img/background.png"));

foreground = new QGraphicsPixmapItem(QPixmap(":/img/title.png"));

logo = new QGraphicsPixmapItem(QPixmap(":/img/logo.png"));

logo->setPos((WIDTH - logo->*boundingRect*().width()) / 2, HEIGHT / 12);

addItem(*background*);

addItem(*foreground*);

addItem(*logo*);

setFocus();

setSceneRect(0, 0, WIDTH, HEIGHT);

m\_view->setScene(this);

}

//===============================================

## Animation des nuages

Programme C++:

//===============================================

// GHome.cpp

//===============================================

GHome::**GHome**(GView\* **\_view**, QWidget\* **\_parent**)

: QGraphicsScene(*\_parent*)

, m\_view(*\_view*) {

m\_view->setViewportUpdateMode(QGraphicsView::SmartViewportUpdate);

m\_scroll = m\_view->horizontalScrollBar();

background = new GAnimation;

background->setPixmap(QPixmap(":/img/background.png"));

foreground = new QGraphicsPixmapItem(QPixmap(":/img/title.png"));

logo = new QGraphicsPixmapItem(QPixmap(":/img/logo.png"));

animation = new QPropertyAnimation(*background*, "pos");

animation->setLoopCount(-1);

animation->setDuration(150000);

animation->setStartValue(QPoint(-WIDTH, 0));

animation->setEndValue(QPoint(0,0));

animation->start();

logo->setPos((WIDTH - logo->*boundingRect*().width()) / 2, HEIGHT / 12);

addItem(*background*);

addItem(*foreground*);

addItem(*logo*);

setFocus();

setSceneRect(0, 0, WIDTH, HEIGHT);

m\_view->setScene(this);

}

//===============================================

// GAnimation.h

//===============================================

class **GAnimation** : public QObject, public QGraphicsPixmapItem {

Q\_OBJECT

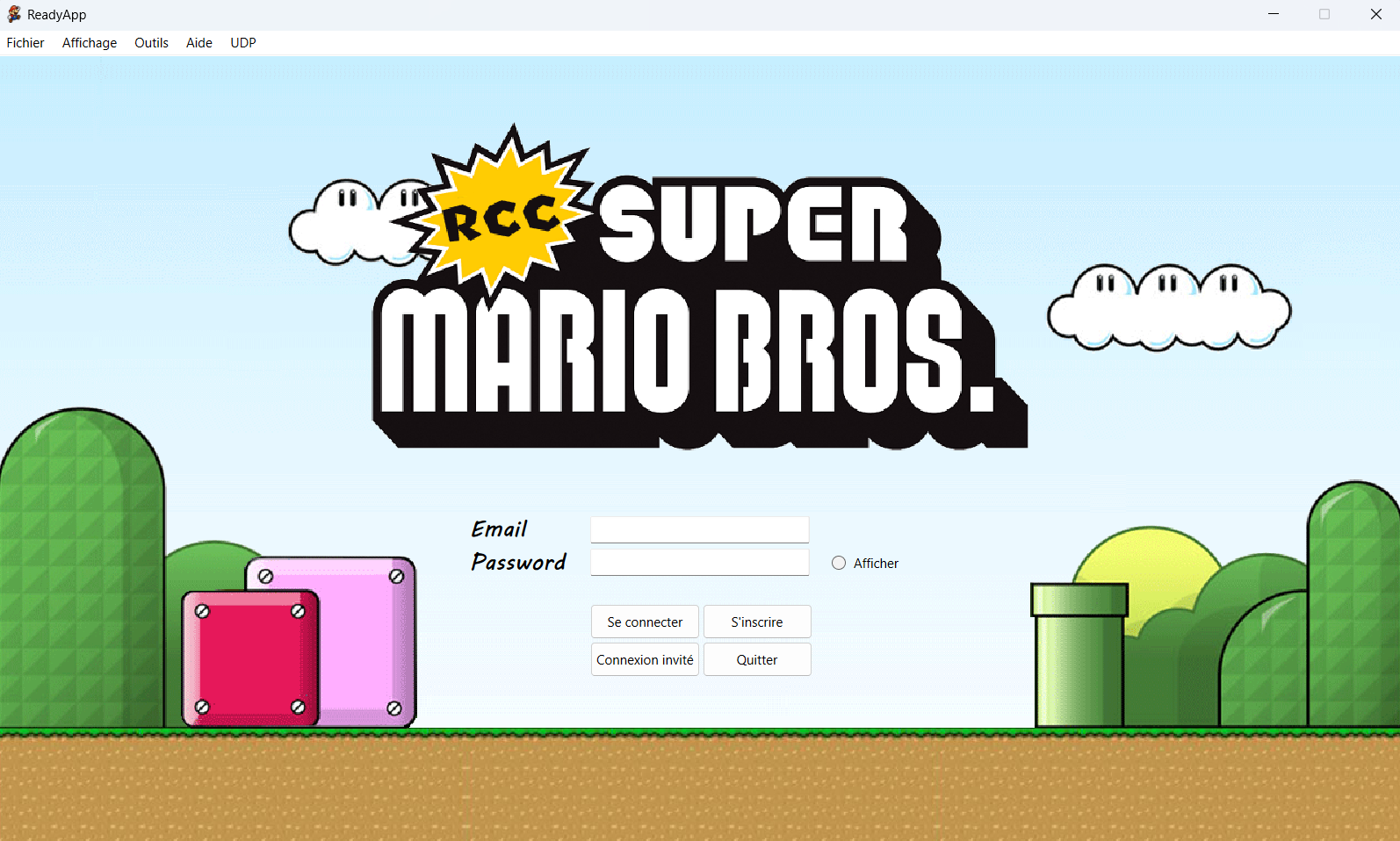
Q\_PROPERTY(QPointF pos READ pos WRITE setPos)

};

//===============================================

## Initialisation de l’interface de connexion

Résultat:



Programme C++:

//===============================================

// GHome.cpp

//===============================================

GHome::**GHome**(GView\* **\_view**, QWidget\* **\_parent**)

: QGraphicsScene(*\_parent*)

, m\_view(*\_view*) {

m\_view->setViewportUpdateMode(QGraphicsView::SmartViewportUpdate);

m\_scroll = m\_view->horizontalScrollBar();

background = new GAnimation;

background->setPixmap(QPixmap(":/img/background.png"));

foreground = new QGraphicsPixmapItem(QPixmap(":/img/title.png"));

logo = new QGraphicsPixmapItem(QPixmap(":/img/logo.png"));

animation = new QPropertyAnimation(*background*, "pos");

animation->setLoopCount(-1);

animation->setDuration(150000);

animation->setStartValue(QPoint(-WIDTH, 0));

animation->setEndValue(QPoint(0,0));

animation->start();

logo->setPos((WIDTH - logo->*boundingRect*().width()) / 2, HEIGHT / 12);

addItem(*background*);

addItem(*foreground*);

addItem(*logo*);

setFocus();

setSceneRect(0, 0, WIDTH, HEIGHT);

m\_view->setScene(this);

loginButton = new QPushButton(*m\_view*);

loginButton->setText("Se connecter");

loginButton->setToolTip("Cliquez pour vous connecter");

loginButton->setGeometry(QRect(540, 500, 100, 32));

connect(loginButton, SIGNAL(clicked()), this, SLOT(login()));

developerButton = new QPushButton(*m\_view*);

developerButton->setText("Connexion invité");

developerButton->setToolTip("Connectez-vous en tant qu'invité");

developerButton->setGeometry(QRect(540, 535, 100, 32));

connect(developerButton, SIGNAL(clicked()), this, SLOT(developerLogin()));

quitButton = new QPushButton(*m\_view*);

quitButton->setText("Quitter");

quitButton->setToolTip("Quitter le programme");

quitButton->setGeometry(QRect(642, 535, 100, 32));

connect(quitButton, SIGNAL(clicked()), this, SLOT(quitProgram()));

newUserButton = new QPushButton(*m\_view*);

newUserButton->setText("S'inscrire");

newUserButton->setToolTip("Cliquez pour créer un identifiant");

newUserButton->setGeometry(QRect(642, 500, 100, 32));

connect(newUserButton, SIGNAL(clicked()), this, SLOT(newUser()));

userLine = new QLineEdit(*m\_view*);

userLine->setToolTip("Entrez une adresse email");

userLine->setGeometry(QRect(540, 420, 200, 25));

QFont **font**("MV Boli", 15, QFont::Bold);

userName = new QLabel(*m\_view*);

userName->setFont(font);

userName->setText("Email");

userName->setGeometry(QRect(430, 420, 100, 25));

passLine = new QLineEdit(*m\_view*);

passLine->setEchoMode(QLineEdit::Password);

passLine->setToolTip("Entrez le mot de passe");

passLine->setGeometry(QRect(540, 450, 200, 25));

password = new QLabel(*m\_view*);

password->setFont(font);

password->setText("Password");

password->setGeometry(QRect(430, 450, 100, 25));

radioButton = new QRadioButton(*m\_view*);

radioButton->setToolTip("Cliquez pour afficher le texte du mot de passe");

radioButton->setGeometry(QRect(760, 450, 100, 25));

connect(radioButton, SIGNAL(toggled(bool)), this, SLOT(on\_radioButton\_toggled(bool)));

radioText = new QLabel(*m\_view*);

radioText->setText("Afficher");

radioText->setToolTip("Cliquez pour afficher le texte du mot de passe");

radioText->setGeometry(QRect(780, 450, 100, 25));

radioText->setBuddy(*radioButton*);

}

//===============================================

## Gestion de la connexion invité

### Suppression de l’interface de connexion

Programme C++:

//===============================================

// GHome.cpp

//===============================================

void GHome::**developerLogin**() {

loginButton->close();

newUserButton->close();

passLine->close();

userLine->close();

userName->close();

password->close();

radioButton->close();

radioText->close();

developerButton->close();

quitButton->close();

}

//===============================================

### Chargement de la scène de jeu

Programme C++:

//===============================================

// GHome.cpp

//===============================================

void GHome::**developerLogin**() {

loginButton->close();

newUserButton->close();

passLine->close();

userLine->close();

userName->close();

password->close();

radioButton->close();

radioText->close();

developerButton->close();

quitButton->close();

scene = new GScene(*m\_scroll*, this);

m\_view->setScene(*scene*);

emit playSound("stopMusic");

}

//===============================================

# Création de la scène de jeu

## Constructeur de la scène de jeu

Programme C++:

//===============================================

// GScene.cpp

//===============================================

GScene::**GScene**(QScrollBar\* **\_scroll**, QObject\* **\_parent**)

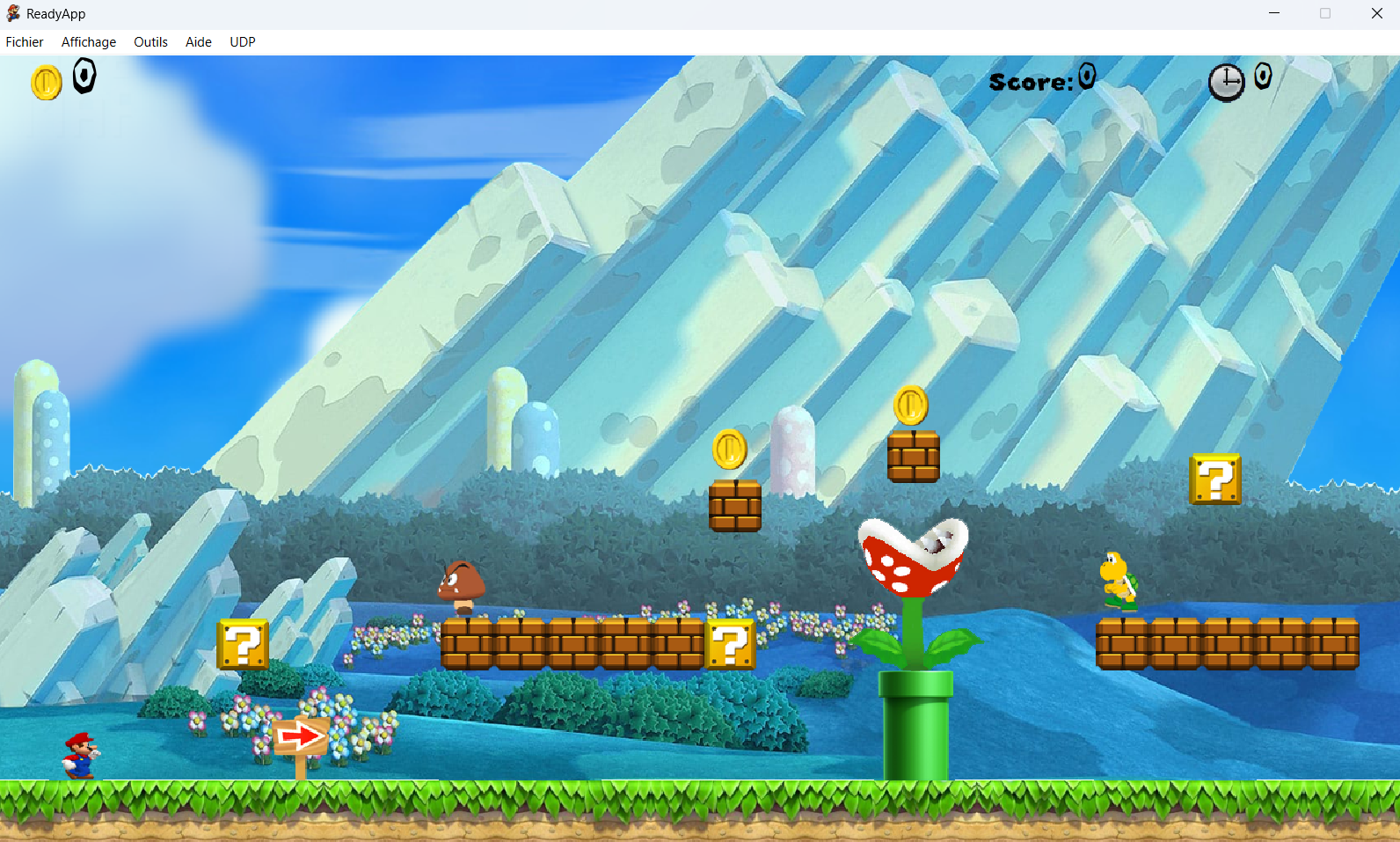
: QGraphicsScene(*\_parent*) {

}

//===============================================

## Initialisation de la scène de jeu

Résultat:



Programme C++:

//===============================================

// GScene.cpp

//===============================================

GScene::**GScene**(QScrollBar\* **\_scroll**, QObject\* **\_parent**)

: QGraphicsScene(*\_parent*)

, m\_jumpAnimation(new QPropertyAnimation(this)) {

initPlayField();

}

//===============================================

void GScene::**initPlayField**(){

int **lWidth** = 8000;

int **lHeight** = 720;

m\_groundLevel = 660;

setSceneRect(0, 0, lWidth, lHeight);

//===============================================

// background

//===============================================

sky = new GBackground(QPixmap(":/img/sky.png"));

sky->setPos(0, 0);

addItem(*sky*);

//...

//===============================================

// coins

//===============================================

m\_coins = new GCoin;

m\_coins->setPos(1300, m\_groundLevel - m\_coins->*boundingRect*().height() - 200);

addItem(*m\_coins*);

//...

//===============================================

// question

//===============================================

m\_questbox = new GQuestion(1);

m\_questbox->setPos(643, m\_groundLevel - m\_questbox->*boundingRect*().height()-100);

addItem(*m\_questbox*);

//...

//===============================================

// mushroom

//===============================================

mushroomQuestBox = new GMushroom(1);

mushroomQuestBox->setPos(200, m\_groundLevel - mushroomQuestBox->*boundingRect*().height()-100);

mushroomQuestBox->setZValue(1);

addItem(*mushroomQuestBox*);

//===============================================

// flower

//===============================================

flowerQuestBox = new GFlower(1);

flowerQuestBox->setPos(1085, m\_groundLevel - flowerQuestBox->*boundingRect*().height()-250);

flowerQuestBox->setZValue(3);

addItem(*flowerQuestBox*);

//===============================================

// bricks

//===============================================

mBrickPlatform = new GBrick(5);

mBrickPlatform->setPos(404, m\_groundLevel - mBrickPlatform->*boundingRect*().height()-100);

addItem(*mBrickPlatform*);

//...

//===============================================

// walls

//===============================================

m\_wallPlatform = new GWall(6);

m\_wallPlatform->setPos(2910, m\_groundLevel - m\_wallPlatform->*boundingRect*().height()-240);

addItem(*m\_wallPlatform*);

//...

//===============================================

// notes

//===============================================

m\_NoteBox = new GNote(5);

m\_NoteBox->setPos(2100, m\_groundLevel - m\_NoteBox->*boundingRect*().height()-130);

addItem(*m\_NoteBox*);

//...

//===============================================

// turtles

//===============================================

m\_turtle = new GTurtle(QRectF(m\_wallPlatform7->pos(), m\_wallPlatform7->*boundingRect*().size()), -1);

m\_turtle->setPos(995, m\_groundLevel - m\_turtle->*boundingRect*().height()-150);

addItem(*m\_turtle*);

connect(this->m\_turtle, SIGNAL(marioSizeStatus(int)),this, SLOT(setMarioSize(int)));

//...

//===============================================

// piranha

//===============================================

m\_piranha = new GPiranha;

m\_piranha->setPos(773, m\_groundLevel - m\_piranha->*boundingRect*().height()-95);

addItem(*m\_piranha*);

//...

//===============================================

// flag

//===============================================

m\_flag2 = new GFlag;

m\_flag2->setPos(7320, m\_groundLevel - m\_flag2->*boundingRect*().height()-285);

addItem(*m\_flag2*);

//===============================================

// counter

//===============================================

m\_count = new GCounter;

m\_count->setPos(65, m\_groundLevel - m\_count->*boundingRect*().height()-602);

addItem(*m\_count*);

//===============================================

// score

//===============================================

m\_score = new GScore;

m\_score->setPos(980, m\_groundLevel - m\_score->*boundingRect*().height()-610);

addItem(*m\_score*);

//===============================================

// timer

//===============================================

m\_gameTimer = new GTimer;

m\_gameTimer->setPos(1140, m\_groundLevel - m\_gameTimer->*boundingRect*().height()-610);

addItem(*m\_gameTimer*);

//===============================================

// stretch

//===============================================

m\_stretch = new GStretch;

m\_stretch->setPos(2100, m\_groundLevel - m\_stretch->*boundingRect*().height()-20);

addItem(*m\_stretch*);

//===============================================

// sign

//===============================================

m\_sign = new GBackground(QPixmap(":/img/sign.png"));

m\_sign->setPos(250, m\_groundLevel - m\_sign->*boundingRect*().height());

addItem(*m\_sign*);

//===============================================

// conveyor

//===============================================

m\_conveyor = new GConveyor;

m\_conveyor->setPos(4160, m\_groundLevel - m\_conveyor->*boundingRect*().height()-100);

addItem(*m\_conveyor*);

//...

//===============================================

// giant\_goomba

//===============================================

m\_giantgoomba = new GGiantGoomba;

m\_giantgoomba->setPos(4210, m\_groundLevel - m\_giantgoomba->*boundingRect*().height()-150);

addItem(*m\_giantgoomba*);

//...

//===============================================

// warp\_tube

//===============================================

m\_warpTube1 = new GWarpTube;

m\_warpTube1->setPos(800, m\_groundLevel - m\_warpTube1->*boundingRect*().height());

addItem(*m\_warpTube1*);

//...

//===============================================

// stairblock

//===============================================

m\_stairBlock = new GStairBlock(9);

m\_stairBlock->setPos(6750, m\_groundLevel - m\_stairBlock->*boundingRect*().height());

addItem(*m\_stairBlock*);

//...

//===============================================

// goombas

//===============================================

m\_goomba = new GGoomba(QRectF(m\_wallPlatform6 ->pos(), m\_wallPlatform6 ->*boundingRect*().size()), -1);

m\_goomba->setPos(400, m\_groundLevel - m\_goomba->*boundingRect*().height()-150);

addItem(*m\_goomba*);

connect(m\_goomba, SIGNAL(marioSizeStatus(int)),this, SLOT(setMarioSize(int)));

//===============================================

// bombs

//===============================================

m\_bomb2 = new GBomb(QRectF(m\_wallPlatform10->pos(), m\_wallPlatform10->*boundingRect*().size()), -1);

m\_bomb2->setPos(6100, m\_groundLevel - m\_bomb2->*boundingRect*().height()-190);

addItem(*m\_bomb2*);

//===============================================

// red\_turtles

//===============================================

m\_redTurtle = new GRedTurtle(QRectF(m\_wallPlatform4->pos(), m\_wallPlatform4->*boundingRect*().size()), -1);

m\_redTurtle->setPos(5100, m\_groundLevel - m\_redTurtle->*boundingRect*().height()+4);

addItem(*m\_redTurtle*);

//===============================================

// spiny

//===============================================

m\_spiny = new GSpiny(QRectF(m\_wallPlatform5->pos(), m\_wallPlatform5->*boundingRect*().size()), -1);

m\_spiny->setPos(1345, m\_groundLevel - m\_spiny->*boundingRect*().height()+4);

addItem(*m\_spiny*);

//===============================================

// small\_mario

//===============================================

m\_smallPlayer = new GSmallMario;

m\_smallPlayer->setPos(50, m\_groundLevel - m\_smallPlayer->*boundingRect*().height() );

addItem(*m\_smallPlayer*);

//===============================================

// castle

//===============================================

h\_castle = new GCastle;

h\_castle->setPos(7637, m\_groundLevel - h\_castle->*boundingRect*().height());

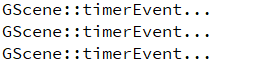
addItem(*h\_castle*);

}

//===============================================

## Animation de la scène de jeu

Résultat:



Programme C++:

//===============================================

// GScene.cpp

//===============================================

void GScene::**initPlayField**(){

//...

startTimer(100);

}

//===============================================

void GScene::***timerEvent***(QTimerEvent\* **\_event**) {

qDebug() << "GScene::timerEvent...";

}

//===============================================

# Création du gestionnaire de Goomba

## Constructeur de Goomba

Résultat:



Programme C++:

//===============================================

// GGoomba.cpp

//===============================================

GGoomba::**GGoomba**(QRectF **\_platformRect**, int **\_direction**, QGraphicsItem\* **\_parent**)

: QGraphicsItem(*\_parent*)

, m\_platformRect(\_platformRect)

, m\_direction(-1)

, m\_currentFrame(0)

, m\_pixmap(":/img/goombas.png") {

setFlag(ItemClipsToShape);

}

//===============================================

QRectF GGoomba::***boundingRect***() const {

return QRectF(0, 0, 52, 50);

}

//===============================================

void GGoomba::***paint***(QPainter\* **\_painter**, const QStyleOptionGraphicsItem\* **\_option**, QWidget\* **\_widget**) {

\_painter->drawPixmap(0, 0, m\_pixmap, m\_currentFrame, 0, 52, 50);

setTransformOriginPoint(*boundingRect*().center());

}

//===============================================

## Animation de Goomba

Résultat:



Programme C++:

//===============================================

// GGoomba.cpp

//===============================================

GGoomba::**GGoomba**(QRectF **\_platformRect**, int **\_direction**, QGraphicsItem\* **\_parent**)

: QGraphicsItem(*\_parent*)

, m\_platformRect(\_platformRect)

, m\_direction(-1)

, m\_currentFrame(0)

, m\_pixmap(":/img/goombas.png") {

setFlag(ItemClipsToShape);

QTimer \***lTimer** = new QTimer(this);

connect(lTimer, &QTimer::timeout, this, &GGoomba::nextFrame);

lTimer->start(100);

}

//===============================================

void GGoomba::**nextFrame**() {

m\_currentFrame += 54;

if (m\_currentFrame >= 862) {

m\_currentFrame = 0;

}

if(pos().x() < m\_platformRect.x() || pos().x() > m\_platformRect.x() + m\_platformRect.width()) {

m\_direction = -m\_direction;

setTransform(QTransform(-m\_direction, 0, 0, 1, *boundingRect*().width(), 0));

}

setPos(pos().x() + (m\_direction\*7), pos().y());

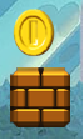
}

//===============================================

# Création du gestionnaire des pièces de monnaie

## Constructeur des pièces de monnaie

Résultat:



Programme C++:

//===============================================

// GCoin.cpp

//===============================================

GCoin::**GCoin**(QGraphicsItem\* **\_parent**)

: QGraphicsItem(*\_parent*)

, mCurrentFrame(0)

, mPixmap(":/img/coin.png") {

setFlag(ItemClipsToShape);

}

//===============================================

QRectF GCoin::***boundingRect***() const {

return QRectF(0, 0, 39, 41);

}

//===============================================

void GCoin::***paint***(QPainter \***painter**, const QStyleOptionGraphicsItem \***option**, QWidget \***widget**) {

painter->drawPixmap(0,0, mPixmap, mCurrentFrame, 0,39, 41);

setTransformOriginPoint(*boundingRect*().center());

}

//===============================================

## Animation des pièces de monnaie

Résultat:



Programme C++:

//===============================================

// GCoin.cpp

//===============================================

void GCoin::**nextFrame**() {

mCurrentFrame += 38;

if(mCurrentFrame >= 300) {

mCurrentFrame = 0;

}

}

//===============================================

# Création du gestionnaire des questionnaires

## Constructeur des questionnaires

Résultat:



Programme C++:

//===============================================

// GQuestion.cpp

//===============================================

## Animation des questionnaires

Résultat:



Programme C++:

//===============================================

// GQuestion.cpp

//===============================================

# Création du gestionnaire des plantes carnivores

## Constructeur des plantes carnivores

Résultat:



Programme C++:

//===============================================

// GPiranha.cpp

//===============================================

## Animation des plantes carnivores

Résultat:



Programme C++:

//===============================================

// GPiranha.cpp

//===============================================

# Création du gestionnaire des briques

## Constructeur des briques

Résultat:



Programme C++:

//===============================================

// GPiranha.cpp

//===============================================

## Animation des briques

Résultat:



Programme C++:

//===============================================

// GPiranha.cpp

//===============================================

# Création du gestionnaire des paramètres du jeu

## Initialisation du gestionnaire

Programme C++:

//===============================================

// GSettings.cpp

//===============================================

GSettings::**GSettings**(QWidget\* **\_parent**)

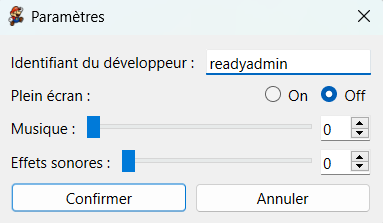
: QDialog(*\_parent*) {

}

//===============================================

## Initialisation de l’interface graphique

Résultat:



Programme C++:

//===============================================

// GSettings.cpp

//===============================================

GSettings::**GSettings**(QWidget\* **\_parent**)

: QDialog(*\_parent*) {

label = new QLabel(tr("Identifiant du développeur : "));

lineEdit = new QLineEdit;

label->setBuddy(*lineEdit*);

bgmLabel = new QLabel(tr("Musique : "));

bgmSlider = new QSlider(Qt::Horizontal);

bgmSpinBox = new QSpinBox;

bgmSlider->setRange(0, 100);

bgmSpinBox->setRange(0, 100);

sfxLabel = new QLabel(tr("Effets sonores : "));

sfxSlider = new QSlider(Qt::Horizontal);

sfxSpinBox = new QSpinBox;

sfxSlider->setRange(0, 100);

sfxSpinBox->setRange(0, 100);

screenSize = new QLabel(tr("Plein écran : "));

fullScreenView = new QRadioButton("On");

windowedView = new QRadioButton("Off");

windowedView->setChecked(true);

confirmButton = new QPushButton(tr("Confirmer"));

cancelButton = new QPushButton(tr("Annuler"));

connect(bgmSpinBox, SIGNAL(valueChanged(int)), bgmSlider, SLOT(setValue(int)));

connect(bgmSlider, SIGNAL(valueChanged(int)), bgmSpinBox, SLOT(setValue(int)));

connect(bgmSlider, SIGNAL(valueChanged(int)), this, SLOT(bgmChanged()));

connect(sfxSpinBox, SIGNAL(valueChanged(int)), sfxSlider, SLOT(setValue(int)));

connect(sfxSlider, SIGNAL(valueChanged(int)), sfxSpinBox, SLOT(setValue(int)));

connect(sfxSlider, SIGNAL(valueChanged(int)), this, SLOT(sfxChanged()));

connect(confirmButton, SIGNAL(clicked(bool)), this, SLOT(confirm()));

connect(cancelButton, SIGNAL(clicked(bool)), this, SLOT(reject()));

QHBoxLayout \***firstLayout** = new QHBoxLayout;

firstLayout->addWidget(*label*);

firstLayout->addWidget(*lineEdit*);

QHBoxLayout \***secondLayout** = new QHBoxLayout;

secondLayout->addWidget(*screenSize*);

secondLayout->addStretch();

secondLayout->addWidget(*fullScreenView*);

secondLayout->addWidget(*windowedView*);

QHBoxLayout \***thirdLayout** = new QHBoxLayout;

thirdLayout->addWidget(*bgmLabel*);

thirdLayout->addWidget((bgmSlider));

thirdLayout->addWidget(*bgmSpinBox*);

QHBoxLayout \***fourthLayout** = new QHBoxLayout;

fourthLayout->addWidget(*sfxLabel*);

fourthLayout->addWidget((sfxSlider));

fourthLayout->addWidget(*sfxSpinBox*);

QHBoxLayout \***fifthLayout** = new QHBoxLayout;

fifthLayout->addWidget(*confirmButton*);

fifthLayout->addWidget(*cancelButton*);

QVBoxLayout \***mainLayout** = new QVBoxLayout;

mainLayout->addLayout(*firstLayout*);

mainLayout->addLayout(*secondLayout*);

mainLayout->addLayout(*thirdLayout*);

mainLayout->addLayout(*fourthLayout*);

mainLayout->addLayout(*fifthLayout*);

setLayout(*mainLayout*);

setWindowModality(Qt::WindowModal);

setWindowTitle("Paramètres");

}

//===============================================

## Enregistrement des paramètres

Programme C++:

//===============================================

// GSettings.cpp

//===============================================

void GSettings::**confirm**() {

setState();

close();

emit fullScreen(fullScreenView->isChecked());

emit bgmAdjust(bgmSlider->value());

emit sfxAdjust(sfxSlider->value());

}

//===============================================

void GSettings::**setState**() {

m\_isFullscreen = fullScreenView->isChecked();

m\_isWindow = windowedView->isChecked();

m\_bgm = bgmSlider->value();

m\_sfx = sfxSlider->value();

}

//===============================================

## Fermeture de l’interface

Programme C++:

//===============================================

// GSettings.cpp

//===============================================

void GSettings::***reject***() {

revertState();

QDialog::*reject*();

}

//===============================================

void GSettings::**revertState**() {

fullScreenView->setChecked(m\_isFullscreen);

windowedView->setChecked(m\_isWindow);

bgmSlider->setValue(m\_bgm);

sfxSlider->setValue(m\_sfx);

}

//===============================================

## Sauvegarde des paramètres du jeu

Programme C++:

//===============================================

// GSettings.cpp

//===============================================

void GSettings::**writeSettings**() {

QSettings **settings**("ReadyTeam", "ReadyApp");

settings.setValue("fullscreen", fullScreenView->isChecked());

settings.setValue("bgm", bgmSlider->value());

settings.setValue("sfx", sfxSlider->value());

}

//===============================================

## Chargement des paramètres du jeu

Programme C++:

//===============================================

// GSettings.cpp

//===============================================

void GSettings::**readSettings**() {

QSettings **settings**("ReadyTeam", "ReadyApp");

fullScreenView->setChecked(settings.value("fullscreen", false).toBool());

bgmSlider->setValue(settings.value("bgm", 50).toInt());

sfxSlider->setValue(settings.value("sfx", 50).toInt());

confirm();

}

//===============================================

## Affichage de l’emplacement des paramètres du jeu

Résultat:



Programme C++:

//===============================================

// GSettings.cpp

//===============================================

void GSettings::**readSettings**() {

QSettings **settings**("ReadyTeam", "ReadyApp");

fullScreenView->setChecked(settings.value("fullscreen", false).toBool());

bgmSlider->setValue(settings.value("bgm", 50).toInt());

sfxSlider->setValue(settings.value("sfx", 50).toInt());

confirm();

qDebug() << settings.fileName();

}

//===============================================

## Alternance entre le mode normal et plein écran

Programme C++:

//===============================================

// GMainWindow.cpp

//===============================================

void GSettings::**alterState**() {

if(m\_isFullscreen){

windowedView->setChecked(true);

}

else {

fullScreenView->setChecked(true);

}

confirm();

}

//===============================================

# Création du gestionnaire de son

## Initialisation du gestionnaire

Programme C++:

//===============================================

// GSound.cpp

//===============================================

GSound::**GSound**(QObject\* **\_parent**)

: QObject(*\_parent*) {

}

//===============================================

## Chargement des fichiers audio

Programme C++:

//===============================================

// GSound.cpp

//===============================================

GSound::**GSound**(QObject\* **\_parent**)

: QObject(*\_parent*) {

audioOutput = new QAudioOutput;

select = new QSoundEffect;

select->setSource(QUrl("qrc:/audio/Select.wav"));

select->setLoopCount(0);

select->setVolume(.25f);

coin = new QSoundEffect;

coin->setSource(QUrl("qrc:/audio/coin.wav"));

coin->setLoopCount(0);

coin->setVolume(.25f);

mario\_jump = new QSoundEffect;

mario\_jump->setSource(QUrl("qrc:/audio/jump.wav"));

mario\_jump->setLoopCount(0);

mario\_jump->setVolume(.25f);

mario\_death = new QSoundEffect;

mario\_death->setSource(QUrl("qrc:/audio/death.wav"));

mario\_death->setLoopCount(0);

mario\_death->setVolume(.25f);

levelClear = new QSoundEffect;

levelClear->setSource(QUrl("qrc:/audio/levelclear.wav"));

levelClear->setLoopCount(0);

levelClear->setVolume(.25f);

ghost = new QSoundEffect;

ghost->setSource(QUrl("qrc:/audio/ghost.wav"));

ghost->setLoopCount(0);

ghost->setVolume(.25f);

shrink = new QSoundEffect;

shrink->setSource(QUrl("qrc:/audio/shrink.wav"));

shrink->setLoopCount(0);

shrink->setVolume(.25f);

powerup = new QSoundEffect;

powerup->setSource(QUrl("qrc:/audio/powerup.wav"));

powerup->setLoopCount(0);

powerup->setVolume(.25f);

sprout = new QSoundEffect;

sprout->setSource(QUrl("qrc:/audio/sprout.wav"));

sprout->setLoopCount(0);

sprout->setVolume(.25f);

fsprout = new QSoundEffect;

fsprout->setSource(QUrl("qrc:/audio/sprout.wav"));

fsprout->setLoopCount(0);

fsprout->setVolume(.25f);

kick = new QSoundEffect;

kick ->setSource(QUrl("qrc:/audio/kick.wav"));

kick ->setLoopCount(0);

kick ->setVolume(.25f);

fireball = new QSoundEffect;

fireball->setSource(QUrl("qrc:/audio/fireball.wav"));

fireball->setLoopCount(0);

fireball->setVolume(.25f);

hitWarptube = new QSoundEffect;

hitWarptube->setSource(QUrl("qrc:/audio/hitwarptube.wav"));

hitWarptube->setLoopCount(0);

hitWarptube->setVolume(.25f);

level1 = new QMediaPlayer;

level1->setAudioOutput(*audioOutput*);

level1->setSource(QUrl("qrc:/audio/level1.mp3"));

soundPlayer = new QMediaPlayer;

soundPlayer->setAudioOutput(*audioOutput*);

soundPlayer->setSource(QUrl("qrc:/audio/ThemeSong.mp3"));

}

//===============================================

## Lecture des fichiers audio

Programme C++:

//===============================================

// GSound.cpp

//===============================================

void GSound::**onPlaySound**(const QString& **\_name**) {

if(\_name == "mario\_jump") {

mario\_jump->play();

}

else if(\_name == "mario\_death") {

level1->stop();

mario\_death->play();

}

else if(\_name == "coin") {

coin->play();

}

else if(\_name == "select") {

select->play();

}

else if(\_name == "theme") {

soundPlayer->play();

}

else if(\_name == "level1") {

level1->play();

}

else if(\_name == "stopMusic") {

soundPlayer->stop();

}

else if(\_name == "stopLevelMusic") {

level1->stop();

}

else if(\_name == "levelClear") {

levelClear->play();

}

else if(\_name == "ghost") {

ghost->play();

}

else if(\_name == "shrink") {

shrink->play();

}

else if(\_name == "powerup") {

powerup->play();

}

else if(\_name == "sprout") {

sprout->play();

}

else if(\_name == "fsprout") {

fsprout->play();

}

else if(\_name == "kick") {

kick->play();

}

else if(\_name == "fireball") {

fireball->play();

}

else if(\_name == "hitWarptube") {

hitWarptube->play();

}

else {

qDebug() << "Le fichier audio n'est pas chargé: " << \_name;

}

}

//===============================================

# Utilisation de QMediaPlayList

## Téléchargement de QMediaPlayList

qmediaplaylist.cpp

<https://code.qt.io/cgit/qt/qtmultimedia.git/plain/examples/multimedia/player/qmediaplaylist.cpp>

qmediaplaylist.h

<https://code.qt.io/cgit/qt/qtmultimedia.git/plain/examples/multimedia/player/qmediaplaylist.h>

qmediaplaylist\_p.cpp

<https://code.qt.io/cgit/qt/qtmultimedia.git/plain/examples/multimedia/player/qmediaplaylist_p.cpp>

qmediaplaylist\_p.h

<https://code.qt.io/cgit/qt/qtmultimedia.git/plain/examples/multimedia/player/qmediaplaylist_p.h>

qplaylistfileparser.cpp

<https://code.qt.io/cgit/qt/qtmultimedia.git/plain/examples/multimedia/player/qplaylistfileparser.cpp>

qplaylistfileparser.h

<https://code.qt.io/cgit/qt/qtmultimedia.git/plain/examples/multimedia/player/qplaylistfileparser.h>