Sommaire

Sommaire 1
Introduction 3
Recommandations techniques 3
Création du programme principal 3
Affichage de l'écran de démarrage 3
Suppression de l'écran de démarrage 4
Affichage de la fenêtre principale 4
Création de la fenêtre principale 4
Initialisation de la fenêtre principale 5
Dimensionnement de la fenêtre principale 5
Initialisation de la vue graphique 5
Initialisation des menus 5
Chargement des paramètres du jeu 8
Sauvegarde des paramètres du jeu 9
Alternance entre le mode normal et plein écran 9
Création de la scène d'accueil 10
Création de la vue graphique 10
Initialisation de la vue graphique 10
Suppression des scrollbars 10
Création de la scène d'accueil 11
Constructeur de la scène d'accueil 11
Initialisation de la scène d'accueil 11
Animation des nuages 12
Initialisation de l'interface de connexion 13
Gestion de la connexion invité 15
Suppression de l'interface de connexion 15
Chargement de la scène de jeu 15
Création de la scène de jeu 16
Constructeur de la scène de jeu 16
Initialisation de la scène de jeu 16
Animation de la scène de jeu 21

Création du gestionnaire de Goomba 21
Constructeur de Goomba 21
Animation de Goomba 22
Création du gestionnaire des pièces de monnaie 23
Constructeur des pièces de monnaie 23
Animation des pièces de monnaie 23
Création du gestionnaire des questionnaires 24
Constructeur des questionnaires 24
Animation des questionnaires 24
Création du gestionnaire des plantes carnivores 25
Constructeur des plantes carnivores 25
Animation des plantes carnivores 25
Création du gestionnaire des briques 26
Constructeur des briques 26
Animation des briques 26
Création du gestionnaire des paramètres du jeu 27
Initialisation du gestionnaire 27
Initialisation de l'interface graphique 27
Enregistrement des paramètres 29
Fermeture de l'interface 29
Sauvegarde des paramètres du jeu 30
Chargement des paramètres du jeu 30
Affichage de l'emplacement des paramètres du jeu 30
Alternance entre le mode normal et plein écran 31
Création du gestionnaire de son 31
Initialisation du gestionnaire 31
Chargement des fichiers audio 31
Lecture des fichiers audio 33
Utilisation de QMediaPlayList 34
Téléchargement de QMediaPlayList 34

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);
}
//=============================================
```

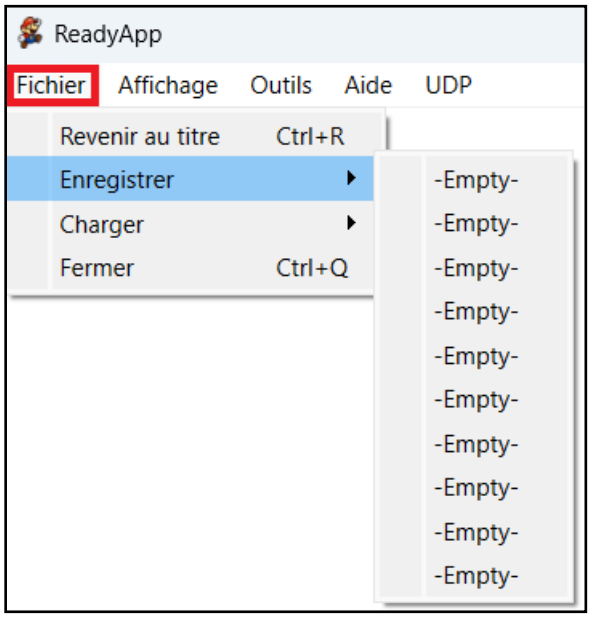
Initialisation de la vue graphique

Programme C++:

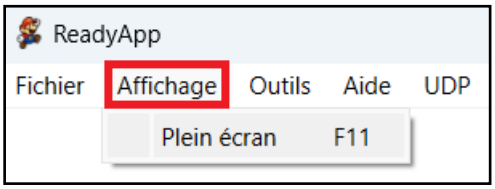
```
//=============================================
// GMainWindow.cpp
//=============================================
GMainWindow::GMainWindow(QWidget* _parent)
: QMainWindow(_parent)  {
    createActions();
    createMenus();
    createScene();
    setCentralWidget(m_view);
    setWindowTitle("ReadyApp");
}
//=============================================
void GMainWindow::createScene() {
    m_view = new GView(this);
}
//=============================================
```

Initialisation des menus

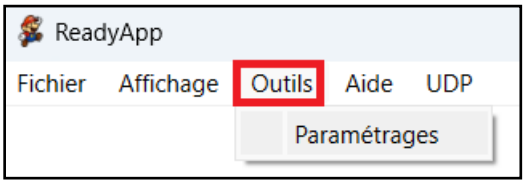
Résultat du menu Fichier:



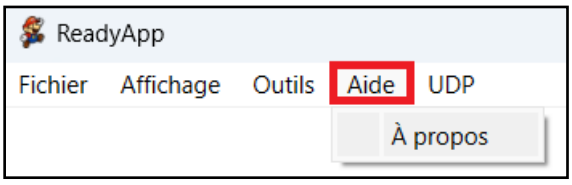
Résultat du menu Affichage:



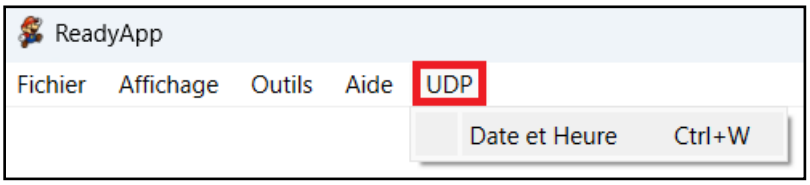
Résultat du menu Outils:



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:

```
GSettings::readSettings...
```

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:

```
GSettings::writeSettings...
```

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

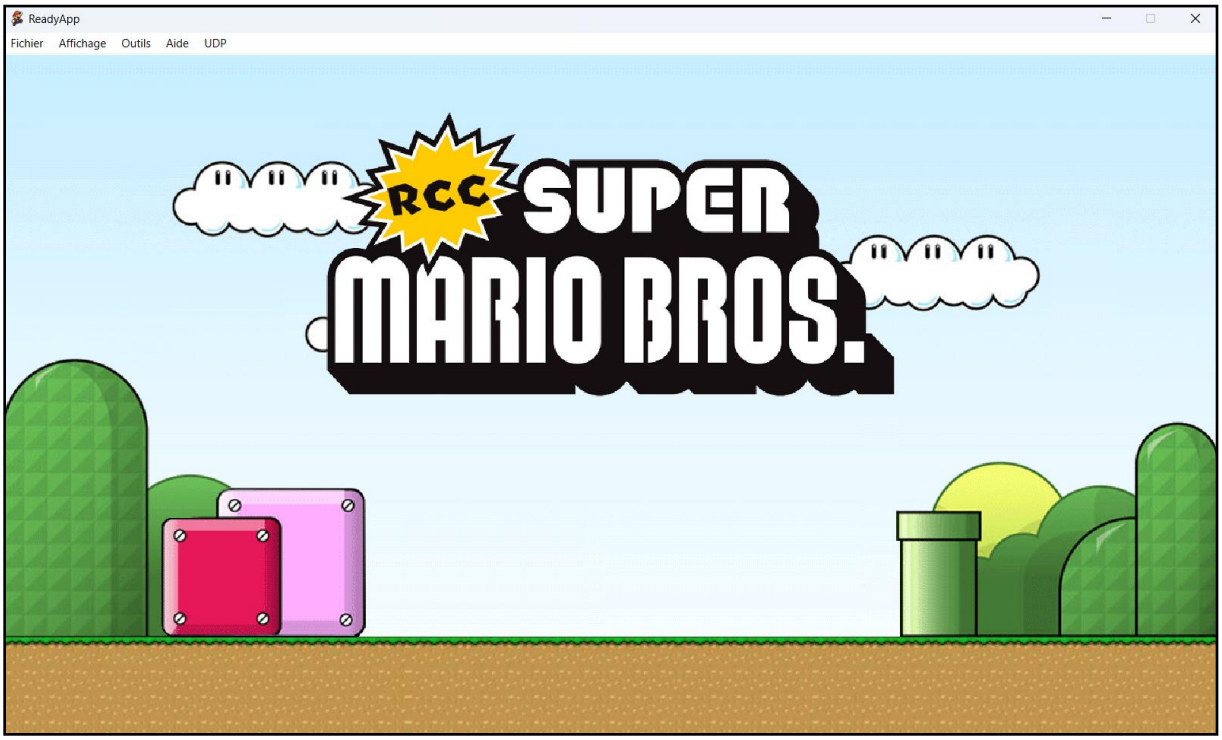
Constructeur de la scène d'accueil

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

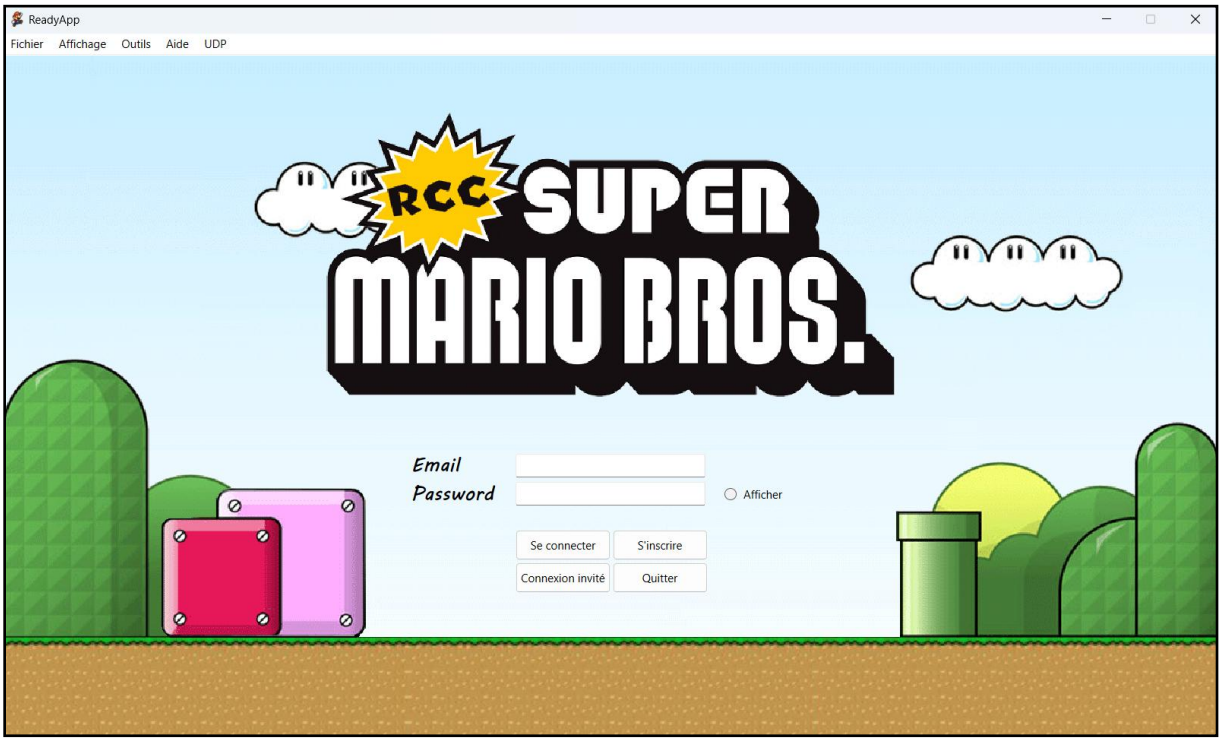
Programme C++:

```
//=============================================
// 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

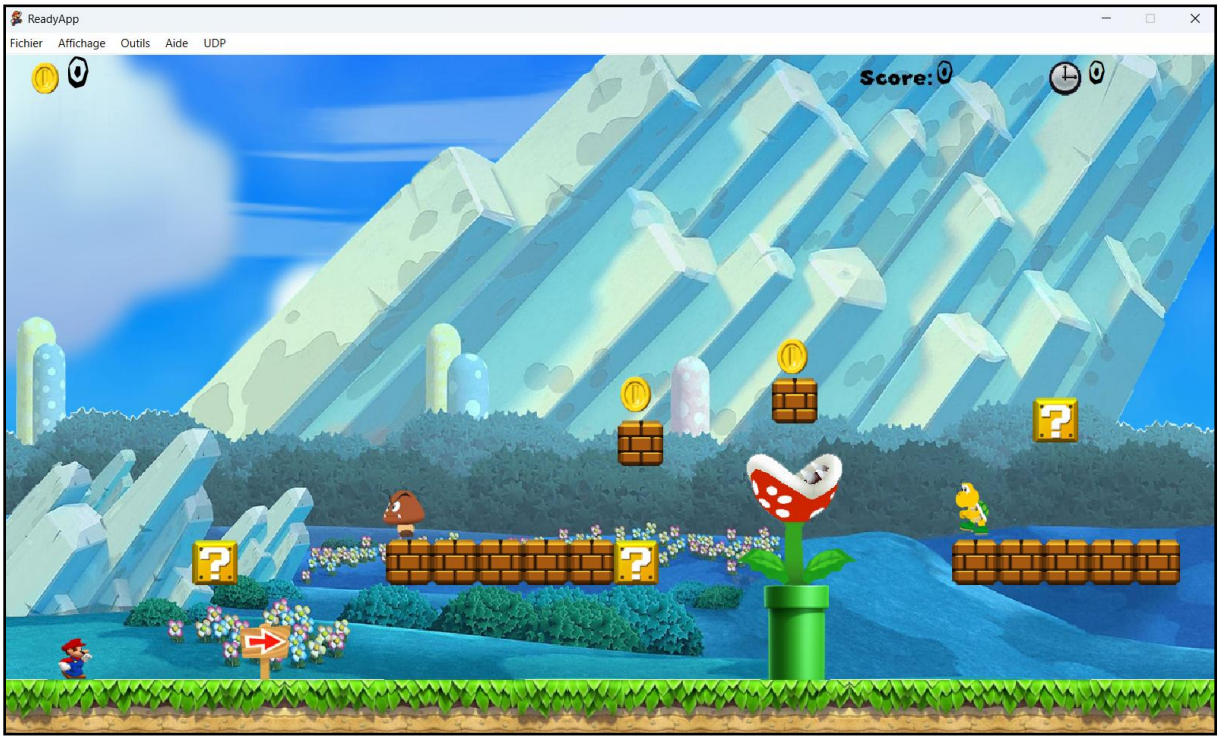
Constructeur de la scène de jeu

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:

```
GScene::timerEvent...
GScene::timerEvent...
GScene::timerEvent...
```

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:

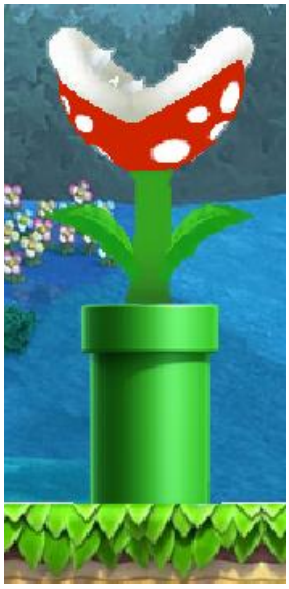
Programme C++:

```
//=============================================
// 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

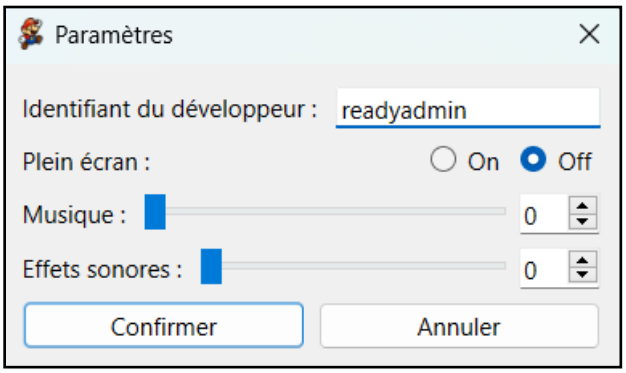
Initialisation du gestionnaire

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:

```
"\\HKEY_CURRENT_USER\\Software\\ReadyTeam\\ReadyApp"
```

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