# 1 項目簡介

翻金幣項目是一款經典的益智類遊戲，我們需要將金幣都翻成同色，才視為勝利。首先，開始介面如下：



點擊start按鈕，進入下層介面，選擇關卡：



在這裡我們設立了20個關卡供玩家選擇，假設我們點擊了第1關，介面如下：



如果想要贏取勝利，我們需要點擊上圖中紅色方框選取的區域，翻動其上下左右的金幣，然後當所有金幣都變為金色，視為勝利，勝利介面如下：



# 2 專案基本配置

## 2.1 創建項目

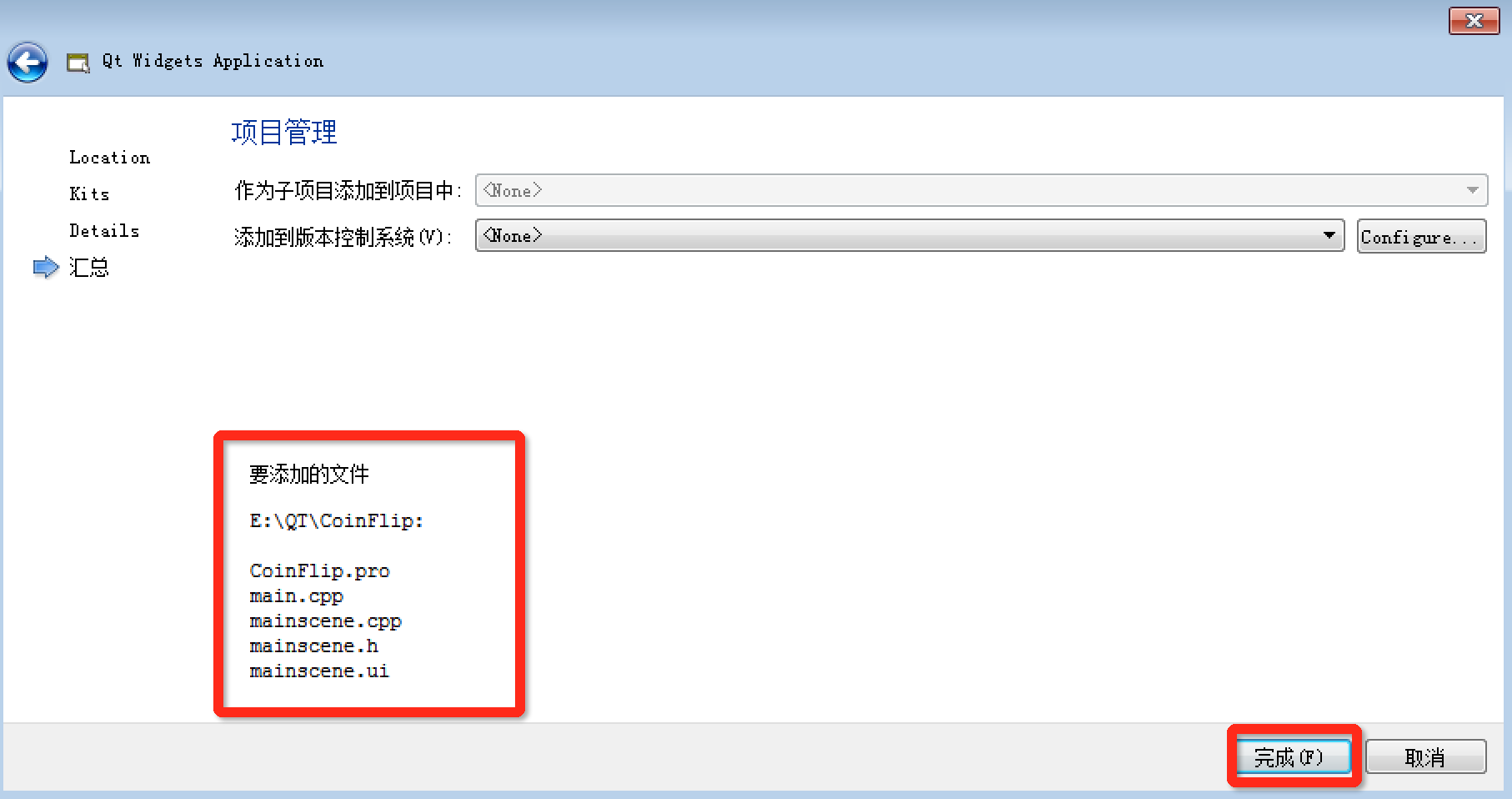
打開Qt-Creator，創建項目：注意名稱不要包含空格和回車，路徑不要有中文



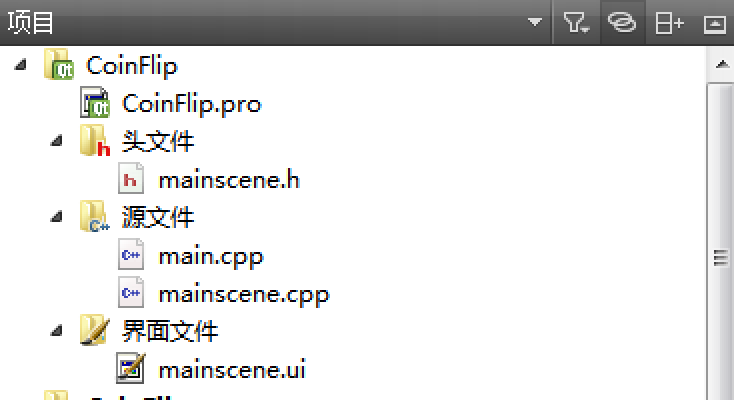
類資訊中，選擇基類為QMainWindow，類名稱為 MainScene，代表著主場景。



點擊完成，創建出項目：

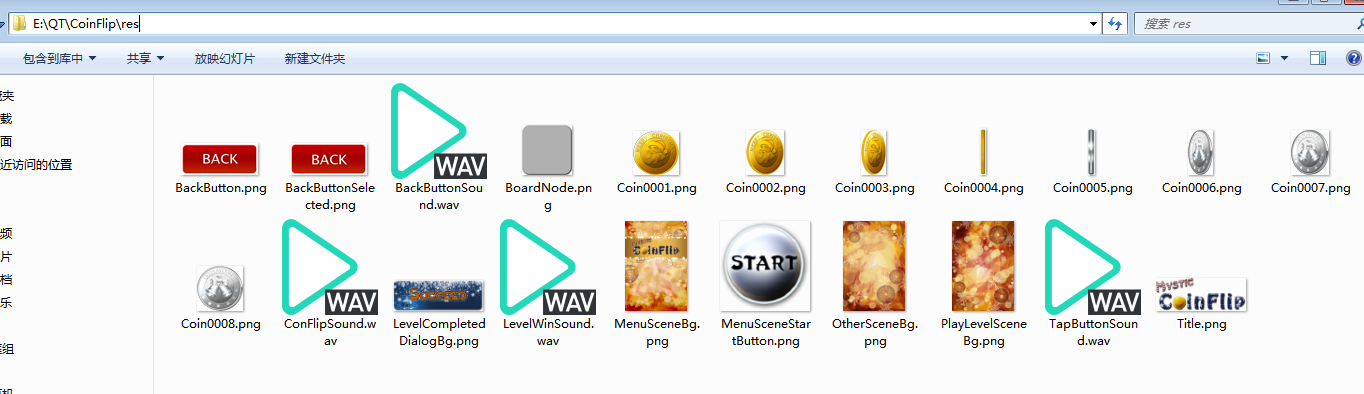


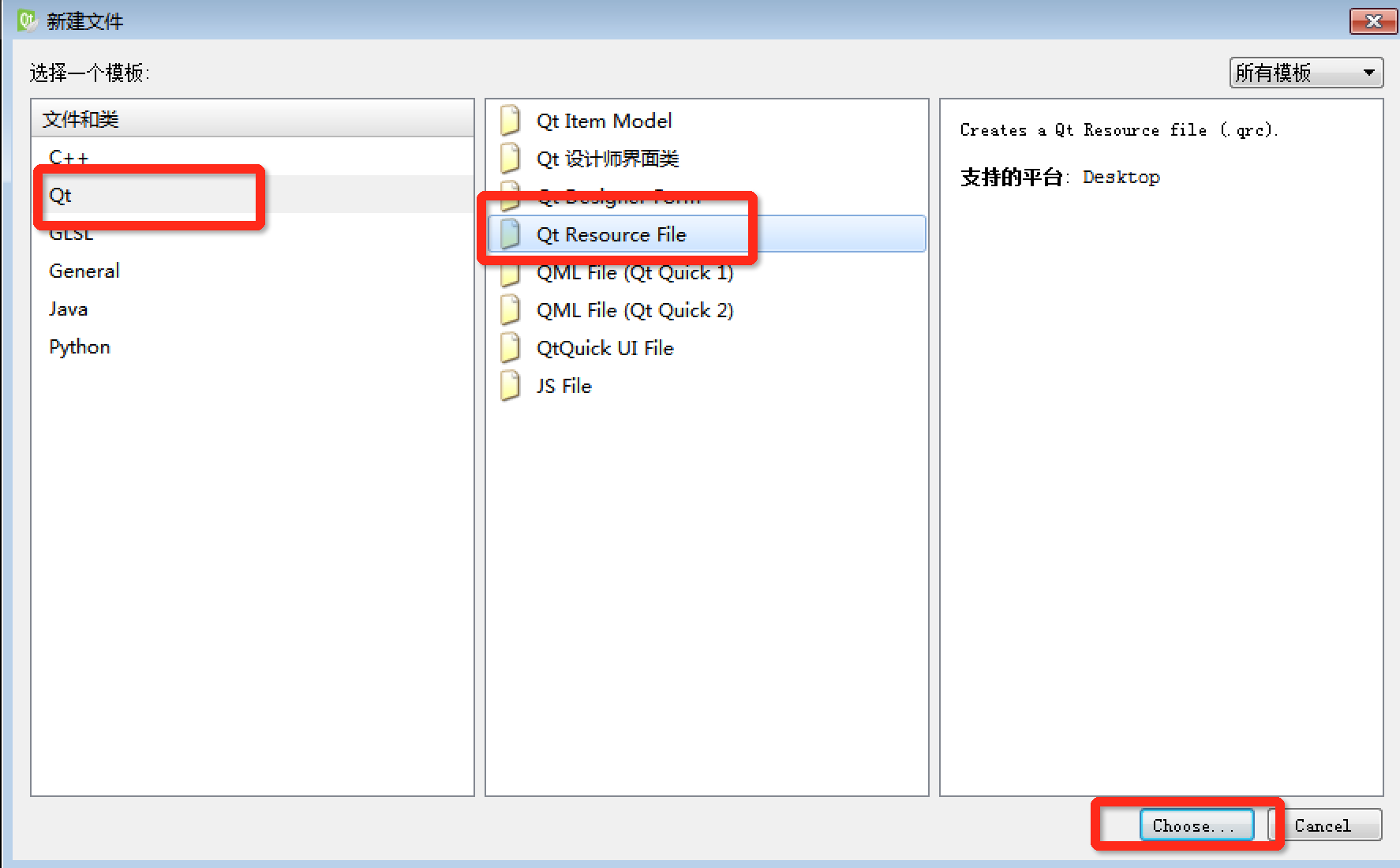
創建的專案結構如下：



## 2.2 添加資源

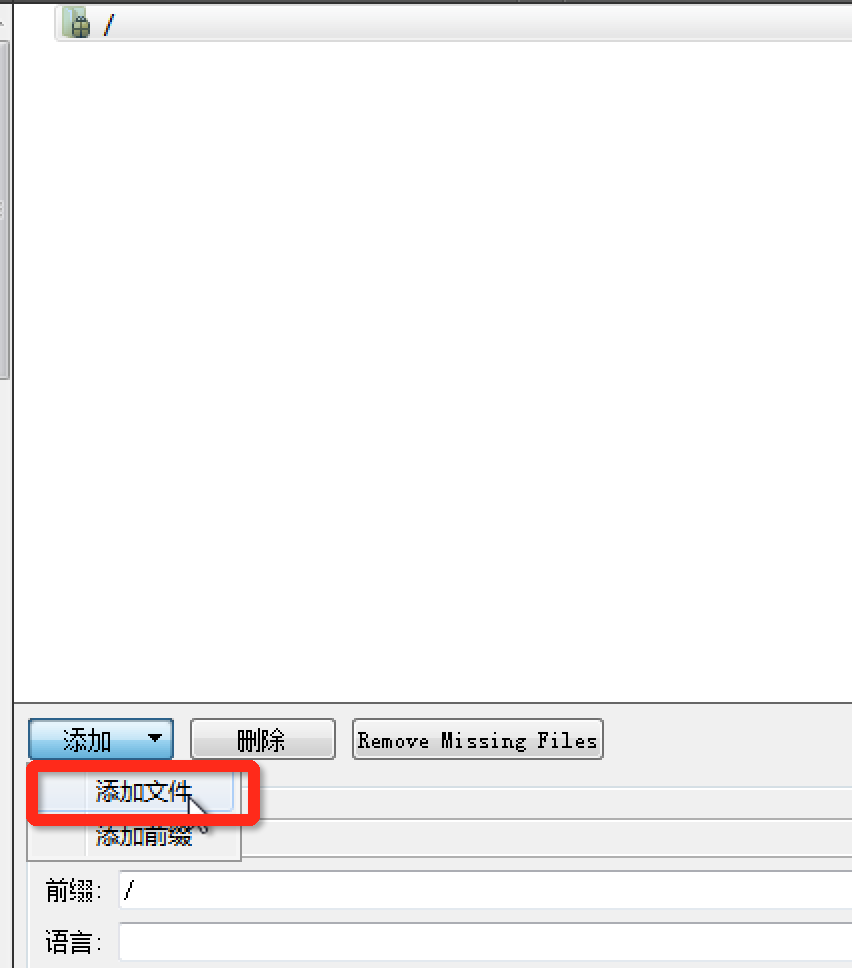
將資源添加到當前專案下

然後創建.qrc文件

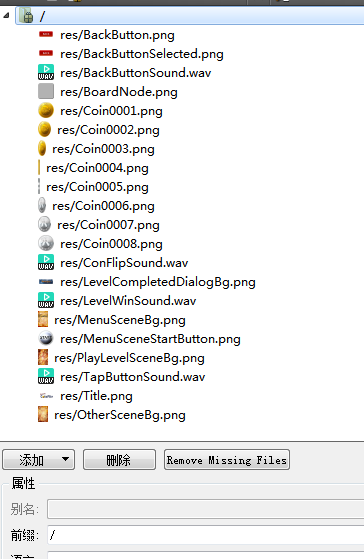


進入編輯模式，添加首碼 “/” ，添加文件





將所有資源檔進行添加

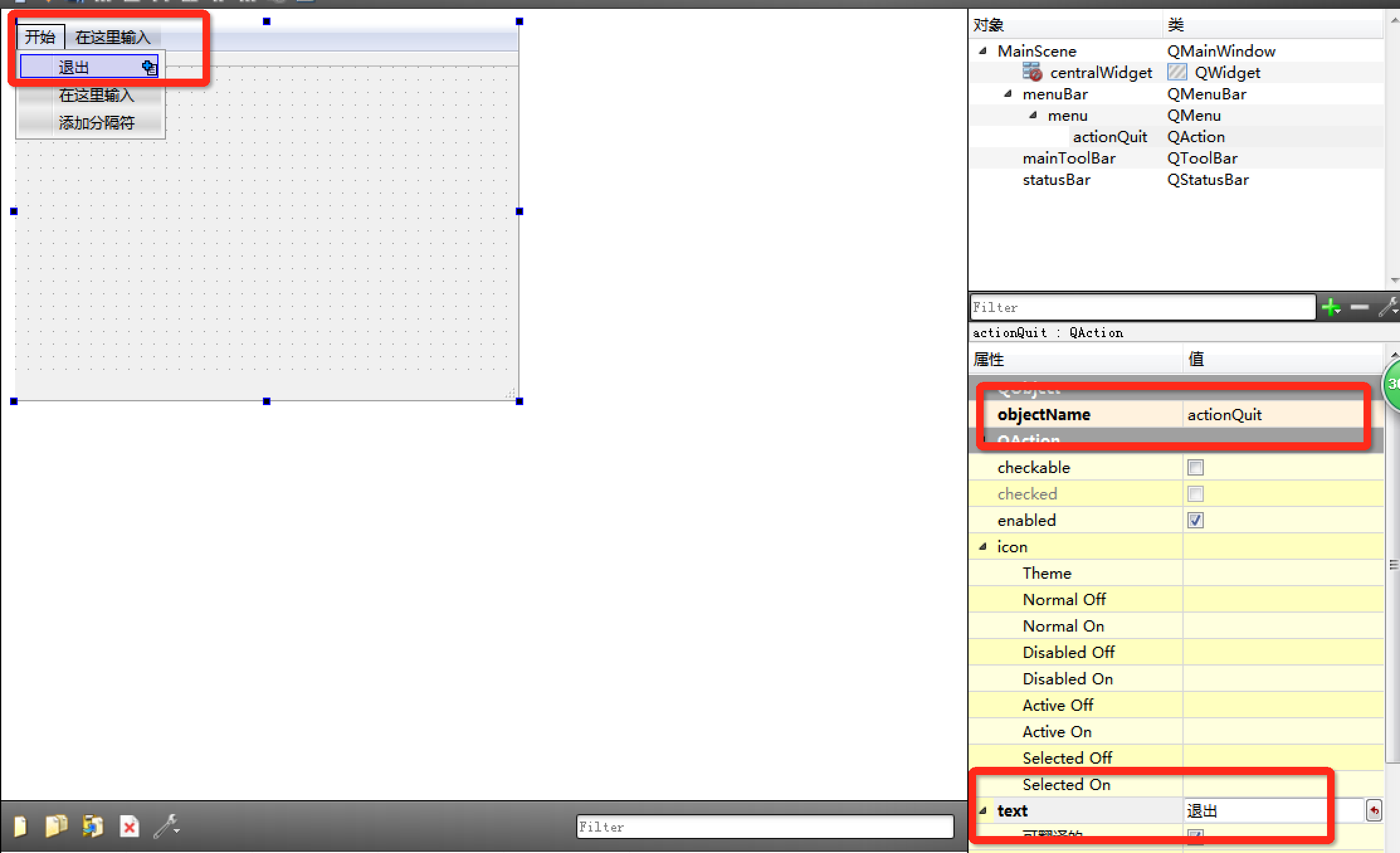


至此將所有需要的資源添加到了本專案中。

# 3 主場景

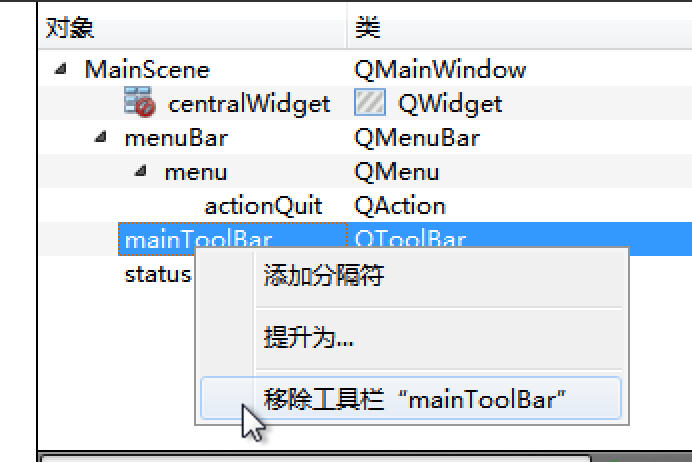
## 3.1 設置遊戲主場景配置

點擊mainscene.ui檔，設計其功能表列如下：



設計“退出”功能表項目，objectName為 actionQuit， text 為 退出；

移除自帶的工具列與狀態列



回到MainScene.cpp檔，進入構造函數中，進行場景的基本配置，代碼如下：

//設置固定大小

this->setFixedSize(320,588);

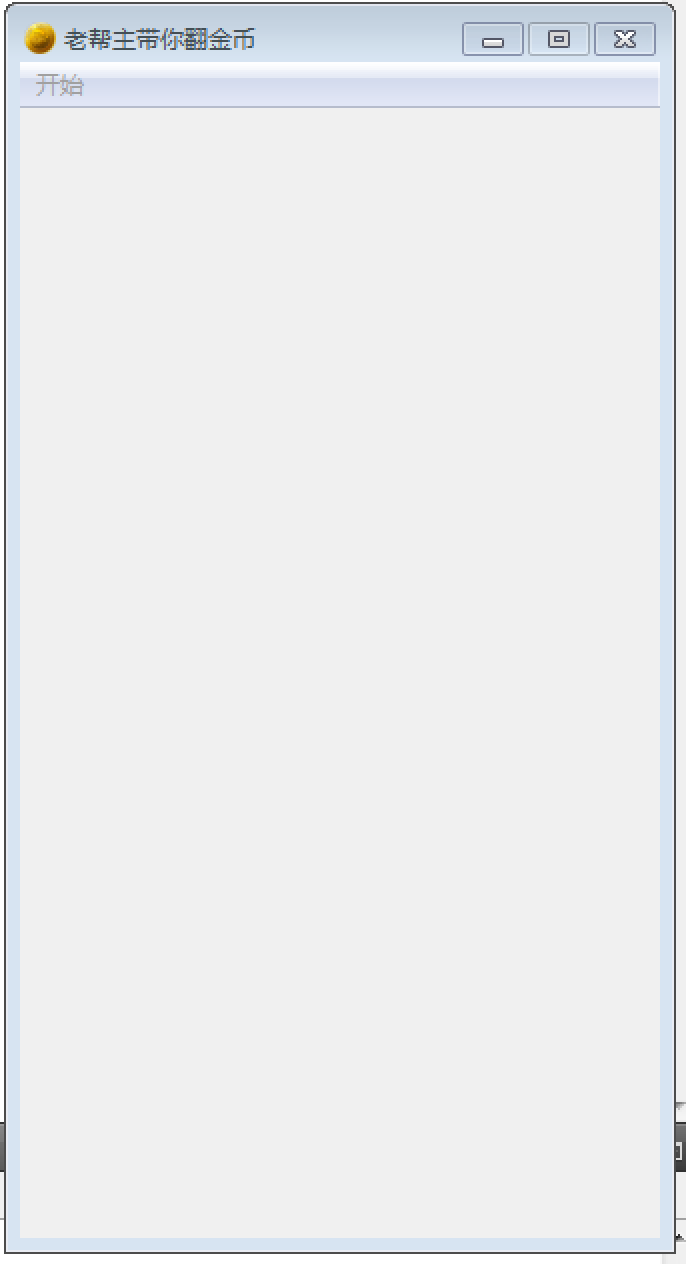
//設置應用圖片

this->setWindowIcon(QPixmap(":/res/Coin0001.png"));

//設置視窗標題

this->setWindowTitle("老幫主帶你翻金幣");

運行效果如圖：



實現點擊開始，退出遊戲功能，代碼如下：

//點擊退出，退出程式

connect(ui->actionQuit,&QAction::triggered,[=](){this->close();});

## 3.2 設置背景圖片

重寫MainScene的PaintEvent事件，並添加一下代碼，繪製背景圖片

void MainScene::*paintEvent*(QPaintEvent \*)

{

//創建畫家，指定繪圖設備

QPainter painter(this);

//創建QPixmap對象

QPixmap pix;

//載入圖片

pix.load(":/res/PlayLevelSceneBg.png");

//繪製背景圖

painter.drawPixmap(0,0,this->width(),this->height(),pix);

//載入標題

pix.load(":/res/Title.png");

//縮放圖片

pix = pix.scaled(pix.width()\*0.5,pix.height()\*0.5);

//繪製標題

painter.drawPixmap( 10,30,pix.width(),pix.height(),pix);

}

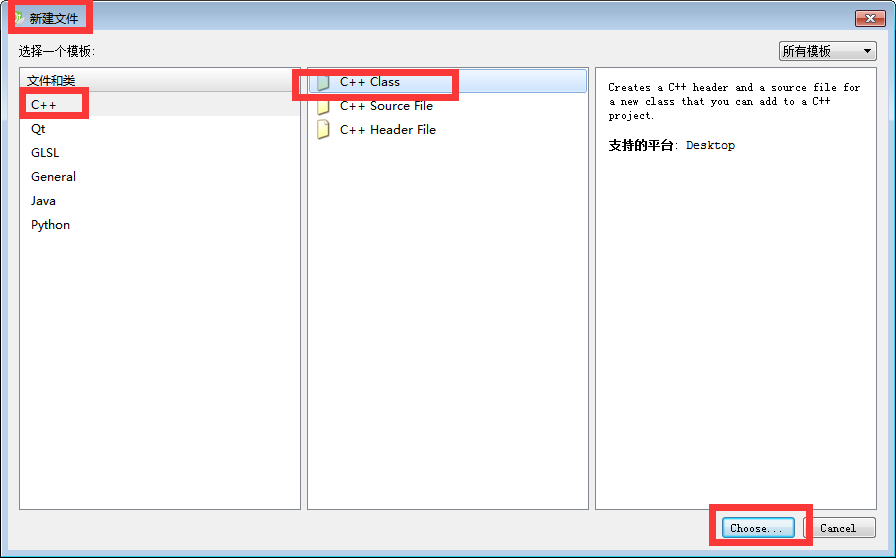
運行效果如圖：

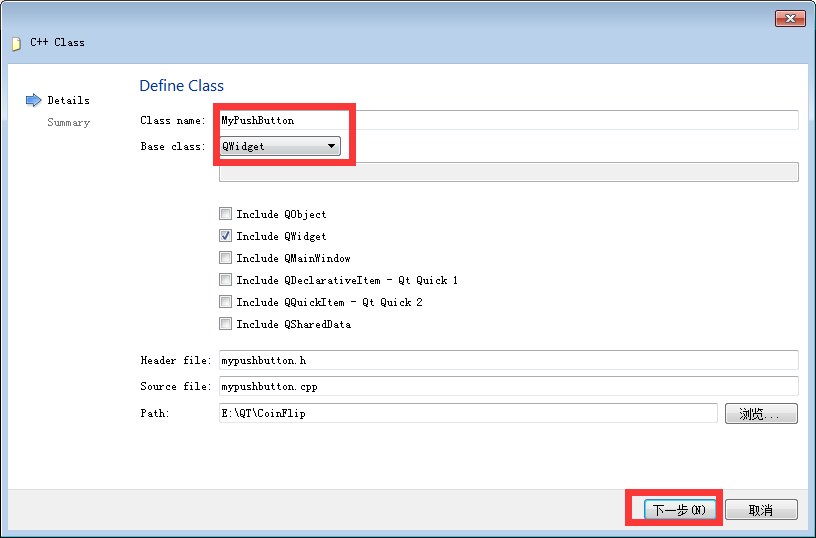


## 3.3 創建開始按鈕

開始按鈕點擊後有彈跳效果，這個效果是我們利用自訂控制項實現的（QPushButton不會自帶這類特效），我們可以自己封裝出一個按鈕控制項，來實現這些效果。

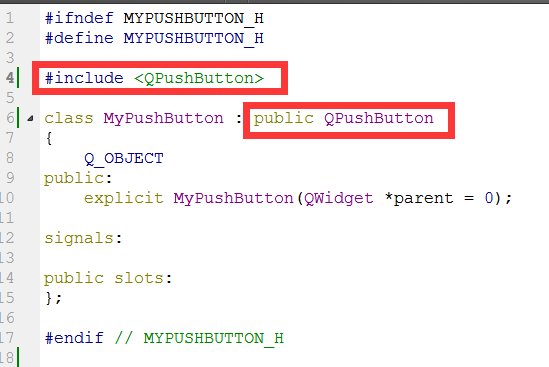
創建MyPushButton，繼承與QPushButton





點擊完成。

修改MyPushButton的父類



提供MyPushButton的構造的重載版本，可以讓MyPushButton提供正常顯示的圖片以及按下後顯示的圖片

代碼如下：

//normalImg 代表正常顯示的圖片

//pressImg 代表按下後顯示的圖片，預設為空

MyPushButton(QString normalImg,QString pressImg = "");

QString normalImgPath; //預設顯示圖片路徑

QString pressedImgPath; //按下後顯示圖片路徑

實現的重載版本MyPushButton構造函數代碼如下：

MyPushButton::MyPushButton(QString normalImg,QString pressImg)

{

//成員變數normalImgPath保存正常顯示圖片路徑

normalImgPath = normalImg;

//成員變數pressedImgPath保存按下後顯示的圖片

pressedImgPath = pressImg;

//創建QPixmap對象

QPixmap pixmap;

//判斷是否能夠載入正常顯示的圖片，若不能提示載入失敗

bool ret = pixmap.load(normalImgPath);

if(!ret)

{

qDebug() << normalImg << "載入圖片失敗!";

}

//設置圖片的固定尺寸

this->setFixedSize( pixmap.width(), pixmap.height() );

//設置不規則圖片的樣式表

this->setStyleSheet("QPushButton{border:0px;}");

//設置圖示

this->setIcon(pixmap);

//設置圖示大小

this->setIconSize(QSize(pixmap.width(),pixmap.height()));

}

回到MainScene的構造函數中，創建開始按鈕

//創建開始按鈕

MyPushButton \* startBtn = new MyPushButton(":/res/MenuSceneStartButton.png");

startBtn->setParent(this);

startBtn->move(this->width()\*0.5-startBtn->width()\*0.5,this->height()\*0.7);

運行效果如圖：



不規則的開始按鈕添加完成。

## 3.4 開始按鈕跳躍特效實現

連接信號槽，監聽開始按鈕點擊

//監聽點擊事件，執行特效

connect(startBtn,&MyPushButton::clicked,[=](){

startBtn->zoom1(); //向下跳躍

startBtn->zoom2(); //向上跳躍

});

zoom1與zoom2 為MyPushButton中擴展的特效代碼，具體如下：

void MyPushButton::zoom1()

{

//創建動畫對象

QPropertyAnimation \* animation1 = new QPropertyAnimation(this,"geometry");

//設置時間間隔，單位毫秒

animation1->setDuration(200);

//創建起始位置

animation1->setStartValue(QRect(this->x(),this->y(),this->width(),this->height()));

//創建結束位置

animation1->setEndValue(QRect(this->x(),this->y()+10,this->width(),this->height()));

//設置緩和曲線，QEasingCurve::OutBounce 為彈跳效果 animation1->setEasingCurve(QEasingCurve::OutBounce);

//開始執行動畫

animation1->start();

}

void MyPushButton::zoom2()

{

QPropertyAnimation \* animation1 = new QPropertyAnimation(this,"geometry");

animation1->setDuration(200);

animation1->setStartValue(QRect(this->x(),this->y()+10,this->width(),this->height()));

animation1->setEndValue(QRect(this->x(),this->y(),this->width(),this->height()));

animation1->setEasingCurve(QEasingCurve::OutBounce);

animation1->start();

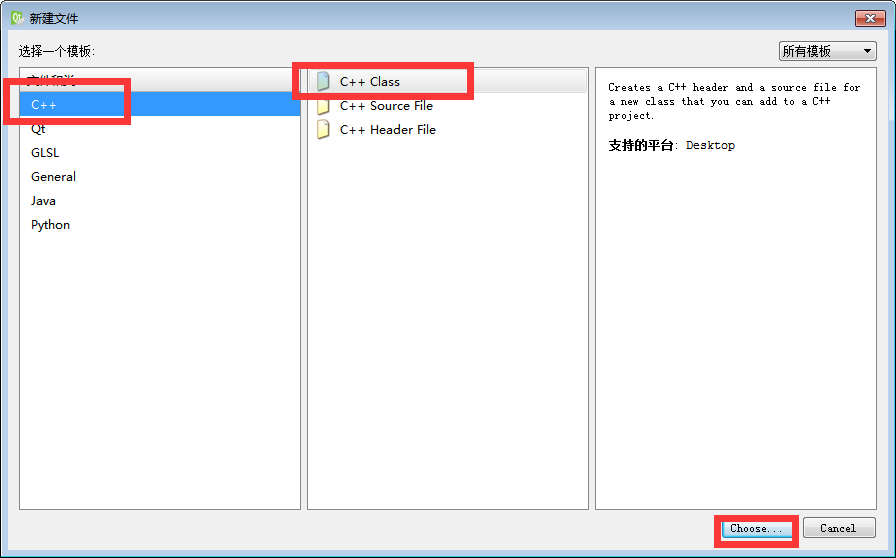
}

運行代碼，點擊按鈕，測試彈跳效果。

## 3.5 創建選擇關卡場景

點擊開始按鈕後，進入選擇關卡場景。

首先我們先創建選擇關卡場景，添加新的C++檔





類名為ChooseLevelScene 選擇基類為QMainWindow，點擊下一步，然後點擊完成。

## 3.6 點擊開始按鈕進入選擇關卡場景

目前點擊主場景的開始按鈕，只有彈跳特效，但是我們還需要有功能上的實現，特效結束後，我們應該進入選擇關卡場景

在MainScene.h中 保存ChooseScene選擇關卡場景物件

//選擇關卡場景

ChooseLevelScene \*chooseScene = new ChooseLevelScene;

我們在zoom1和zoom2特效後，延時0.5秒，進入選擇關卡場景，代碼如下：

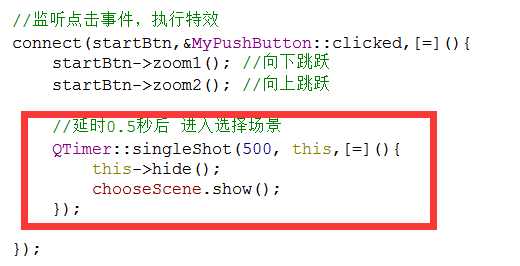
//延時0.5秒後 進入選擇場景

QTimer::singleShot(500, this,[=](){

this->hide();

chooseScene->show();

});



測試點擊開始，執行特效後延時0.5秒進入選擇關卡場景

# 4 選擇關卡場景

## 4.1 場景基本設置

選擇關卡構造函數如下：

//設置視窗固定大小

this->setFixedSize(320,588);

//設置圖示

this->setWindowIcon(QPixmap(":/res/Coin0001.png"));

//設置標題

this->setWindowTitle("選擇關卡");

//創建功能表列

QMenuBar \* bar = this->menuBar();

this->setMenuBar(bar);

//創建開始菜單

QMenu \* startMenu = bar->addMenu("開始");

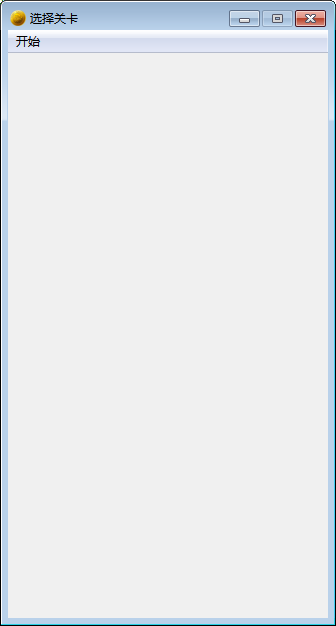
//創建按鈕功能表項目

QAction \* quitAction = startMenu->addAction("退出");

//點擊退出 退出遊戲

connect(quitAction,&QAction::triggered,[=](){this->close();});

運行效果如圖：



## 4.2 背景設置

void ChooseLevelScene::*paintEvent*(QPaintEvent \*)

{

QPainter painter(this);

QPixmap pix;

pix.load(":/res/OtherSceneBg.png");

painter.drawPixmap(0,0,this->width(),this->height(),pix);

//載入標題

pix.load(":/res/Title.png");

painter.drawPixmap( (this->width() - pix.width())\*0.5,30,pix.width(),pix.height(),pix);

}

4.3 創建返回按鈕

//返回按鈕

MyPushButton \* closeBtn = new MyPushButton(":/res/BackButton.png",":/res/BackButtonSelected.png");

closeBtn->setParent(this);

closeBtn->move(this->width()-closeBtn->width(),this->height()-closeBtn->height());

返回按鈕是有正常顯示圖片和點擊後顯示圖片的兩種模式，所以我們需要重寫MyPushButton中的 MousePressEvent和MouseReleaseEvent

//滑鼠事件

void MyPushButton::*mousePressEvent*(QMouseEvent \*e)

{

if(pressedImgPath != "") //選中路徑不為空，顯示選中圖片

{

QPixmap pixmap;

bool ret = pixmap.load(pressedImgPath);

if(!ret)

{

qDebug() << pressedImgPath << "載入圖片失敗!";

}

this->setFixedSize( pixmap.width(), pixmap.height() );

this->setStyleSheet("QPushButton{border:0px;}");

this->setIcon(pixmap);

this->setIconSize(QSize(pixmap.width(),pixmap.height()));

}

//交給父類執行按下事件

return QPushButton::*mousePressEvent*(e);

}

void MyPushButton::*mouseReleaseEvent*(QMouseEvent \*e)

{

if(normalImgPath != "") //選中路徑不為空，顯示選中圖片

{

QPixmap pixmap;

bool ret = pixmap.load(normalImgPath);

if(!ret)

{

qDebug() << normalImgPath << "載入圖片失敗!";

}

this->setFixedSize( pixmap.width(), pixmap.height() );

this->setStyleSheet("QPushButton{border:0px;}");

this->setIcon(pixmap);

this->setIconSize(QSize(pixmap.width(),pixmap.height()));

}

//交給父類執行 釋放事件

return QPushButton::*mouseReleaseEvent*(e);

}

## 4.3 返回按鈕

在這裡我們點擊返回後，延時0.5後隱藏自身，並且發送自訂信號，告訴外界自身已經選擇了返回按鈕。

//返回按鈕功能實現

connect(closeBtn,&MyPushButton::clicked,[=](){

QTimer::singleShot(500, this,[=](){

this->hide();

//觸發自訂信號，關閉自身，該信號寫到 signals下做聲明

emit this->chooseSceneBack();

}

);

});

在主場景MainScene中 點擊開始按鈕顯示選擇關卡的同時，監聽選擇關卡的返回按鈕消息

//監聽選擇場景的返回按鈕 connect(chooseScene,&ChooseLevelScene::chooseSceneBack,[=](){

this->show();

});

測試主場景與選擇關卡場景的切換功能。

## 4.4 創建選擇關卡按鈕

//創建關卡按鈕

for(int i = 0 ; i < 20;i++)

{

MyPushButton \* menuBtn = new MyPushButton(":/res/LevelIcon.png");

menuBtn->setParent(this);

menuBtn->move(25 + (i%4)\*70 , 130+ (i/4)\*70);

//按鈕上顯示的文字

QLabel \* label = new QLabel;

label->setParent(this);

label->setFixedSize(menuBtn->width(),menuBtn->height());

label->setText(QString::number(i+1));

label->setAlignment(Qt::AlignHCenter | Qt::AlignVCenter); //設置居中

label->move(25 + (i%4)\*70 , 130+ (i/4)\*70);

label->setAttribute(Qt::WA\_TransparentForMouseEvents,true); //滑鼠事件穿透

}

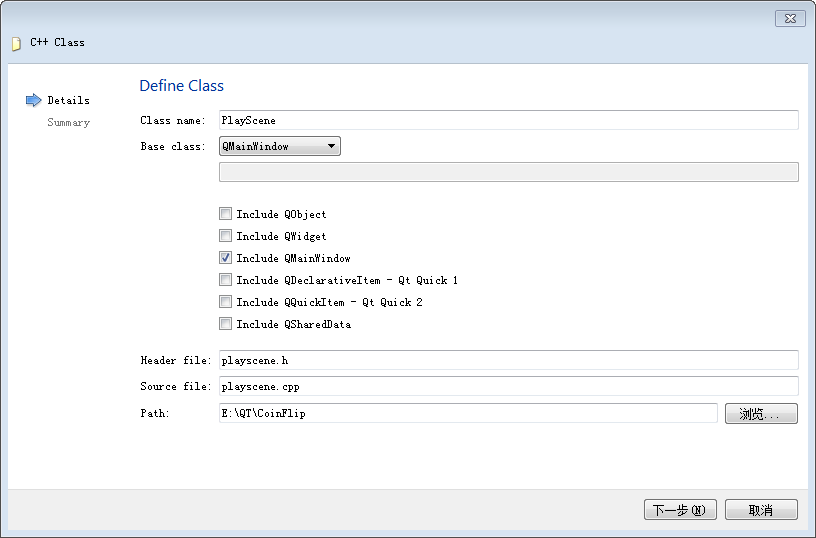
運行效果如果：



## 4.5 創建翻金幣場景

點擊關卡按鈕後，會進入遊戲的核心場景，也就是翻金幣的場景，首先先創建出該場景的.h和.cpp文件

創建PlayScene



點擊選擇關卡按鈕後會跳入到該場景

建立點擊按鈕，跳轉場景的信號槽連接

在ChooseLevelScene.h 中聲明

PlayScene \*pScene = NULL;

//監聽選擇關卡按鈕的信號槽

connect(menuBtn,&MyPushButton::clicked,[=](){

// qDebug() << "select: " << i;

if(pScene == NULL) //遊戲場景最好不用複用，直接移除掉創建新的場景

{

this->hide();

pScene = new PlayScene(i+1); //將選擇的關卡號 傳入給PlayerScene

pScene->show();

}

});

這裡pScene = new PlayScene(i+1); 將用戶所選的關卡號發送給pScene，也就是翻金幣場景，當然PlayScene 要提供重載的有參構造版本，來接受這個參數

# 5 翻金幣場景

## 5.1 場景基本設置

PlayScene.h中 聲明成員變數，用於記錄當前使用者選擇的關卡

//成員變數 記錄關卡索引

int levalIndex;

PlayScene.cpp中 初始化該場景配置

PlayScene::PlayScene(int index)

{

//qDebug() << "當前關卡為"<< index;

this->levalIndex = index;

//設置視窗固定大小

this->setFixedSize(320,588);

//設置圖示

this->setWindowIcon(QPixmap(":/res/Coin0001.png"));

//設置標題

this->setWindowTitle("翻金幣");

//創建功能表列

QMenuBar \* bar = this->menuBar();

this->setMenuBar(bar);

//創建開始菜單

QMenu \* startMenu = bar->addMenu("開始");

//創建按鈕功能表項目

QAction \* quitAction = startMenu->addAction("退出");

//點擊退出 退出遊戲

connect(quitAction,&QAction::triggered,[=](){this->close();});

}

## 5.2 背景設置

void PlayScene::*paintEvent*(QPaintEvent \*)

{

//載入背景

QPainter painter(this);

QPixmap pix;

pix.load(":/res/PlayLevelSceneBg.png");

painter.drawPixmap(0,0,this->width(),this->height(),pix);

//載入標題

pix.load(":/res/Title.png");

pix = pix.scaled(pix.width()\*0.5,pix.height()\*0.5);

painter.drawPixmap( 10,30,pix.width(),pix.height(),pix);

}

## 5.3 返回按鈕

//返回按鈕

MyPushButton \* closeBtn = new MyPushButton(":/res/BackButton.png",":/res/BackButtonSelected.png");

closeBtn->setParent(this);

closeBtn->move(this->width()-closeBtn->width(),this->height()-closeBtn->height());

//返回按鈕功能實現

connect(closeBtn,&MyPushButton::clicked,[=](){

QTimer::singleShot(500, this,[=](){

this->hide();

//觸發自訂信號，關閉自身，該信號寫到 signals下做聲明

emit this->chooseSceneBack();

}

);

});

5.4 在ChooseScene選擇關卡場景中，監聽PlayScene的返回信號

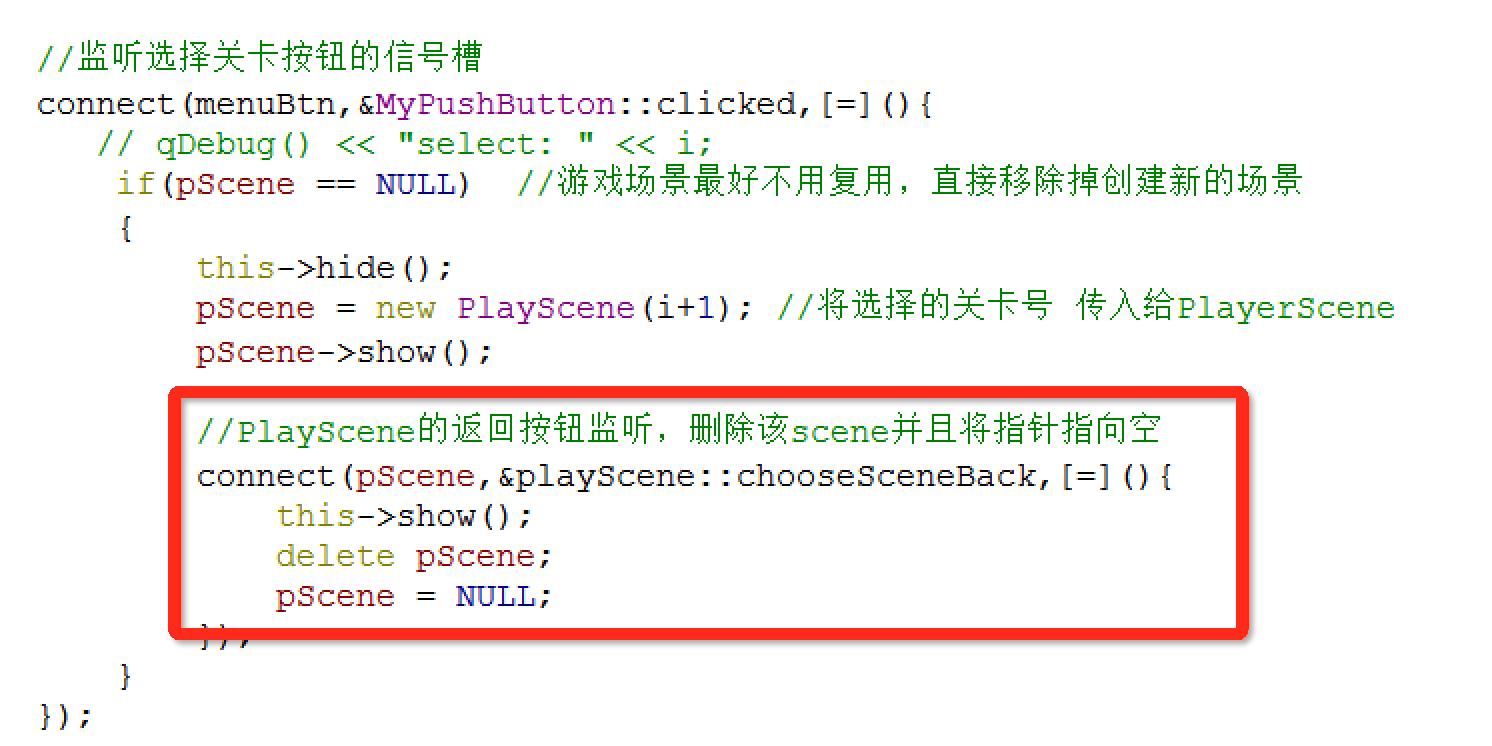
connect(pScene,&PlayScene::chooseSceneBack,[=](){

this->show();

delete pScene;

pScene = NULL;

});



## 5.4 顯示當前關卡

//當前關卡標題

QLabel \* label = new QLabel;

label->setParent(this);

QFont font;

font.setFamily("華文新魏");

font.setPointSize(20);

label->setFont(font);

QString str = QString("Leavel: %1").arg(this->levalIndex);

label->setText(str);

label->setGeometry(QRect(30, this->height() - 50,120, 50)); //設置大小和位置

假設我們選擇了第15關卡，運行效果如果：



## 5.5 創建金幣背景圖片

//創建金幣的背景圖片

for(int i = 0 ; i < 4;i++)

{

for(int j = 0 ; j < 4; j++)

{

//繪製背景圖片

QLabel\* label = new QLabel;

label->setGeometry(0,0,50,50);

label->setPixmap(QPixmap(":/res/BoardNode.png"));

label->setParent(this);

label->move(57 + i\*50,200+j\*50);

}

}

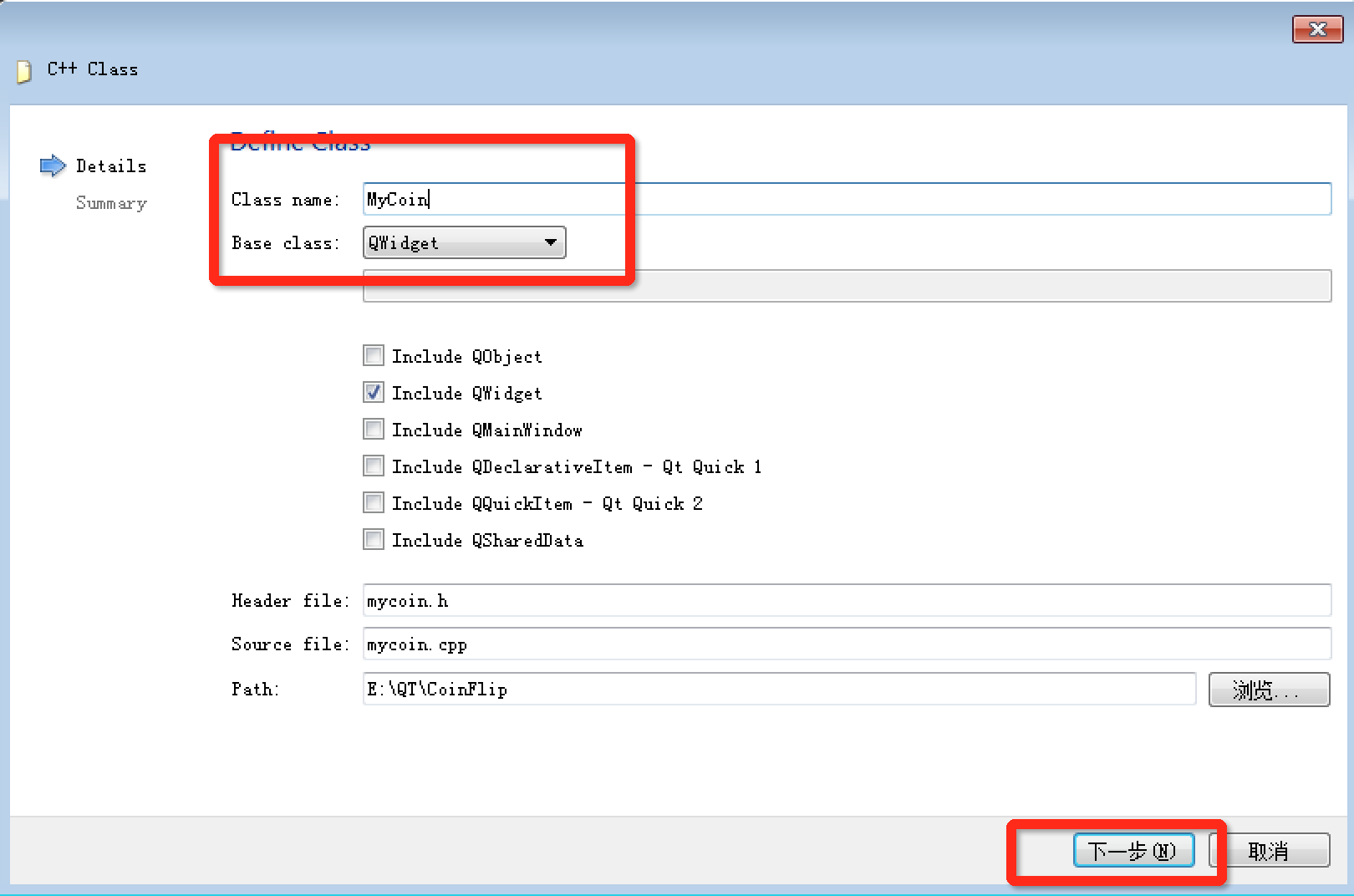
運行效果如圖：



## 5.6 創建金幣類

我們知道，金幣是本遊戲的核心物件，並且在遊戲中可以利用二維陣列進行維護，擁有支援點擊，翻轉特效等特殊性，因此不妨將金幣單獨封裝到一個類中，完成金幣所需的所有功能。

### 5.6.1 創建金幣類 MyCoin

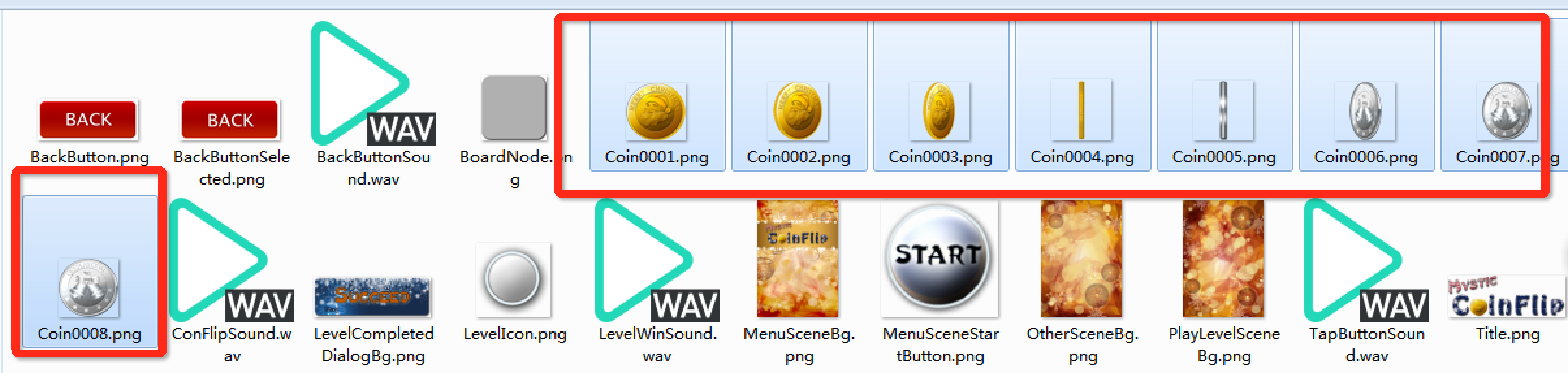


並修改MyCoin的基類為QPushButton

### 5.6.2 構造函數

在資源圖片中，我們可以看到，金幣翻轉的效果原理是多張圖片切換而形成的，而以下八張圖片中，第一張與最後一張比較特殊，因此我們在給用戶看的時候，無非是金幣Coin0001或者是銀幣 Coin0008這兩種圖。

因此我們在創建一個金幣物件時候，應該提供一個參數，代表著傳入的是金幣資源路徑還是銀幣資源路徑，根據路徑我們創建不同樣式的圖案。



在MyCoin.h中聲明：

MyCoin(QString butImg); //代表圖片路徑

在MyCoin.cpp中進行實現

MyCoin::MyCoin(QString butImg)

{

QPixmap pixmap;

bool ret = pixmap.load(butImg);

if(!ret)

{

qDebug() << butImg << "載入圖片失敗!";

}

this->setFixedSize( pixmap.width(), pixmap.height() );

this->setStyleSheet("QPushButton{border:0px;}");

this->setIcon(pixmap);

this->setIconSize(QSize(pixmap.width(),pixmap.height()));

}

### 5.6.3 測試

在翻金幣場景 PlayScene中，我們測試下封裝的金幣類是否可用，可以在創建好的金幣背景代碼後，添加如下代碼：

//金幣對象

MyCoin \* coin = new MyCoin(":/res/Coin0001.png");

coin->setParent(this);

coin->move(59 + i\*50,204+j\*50);

運行效果如圖

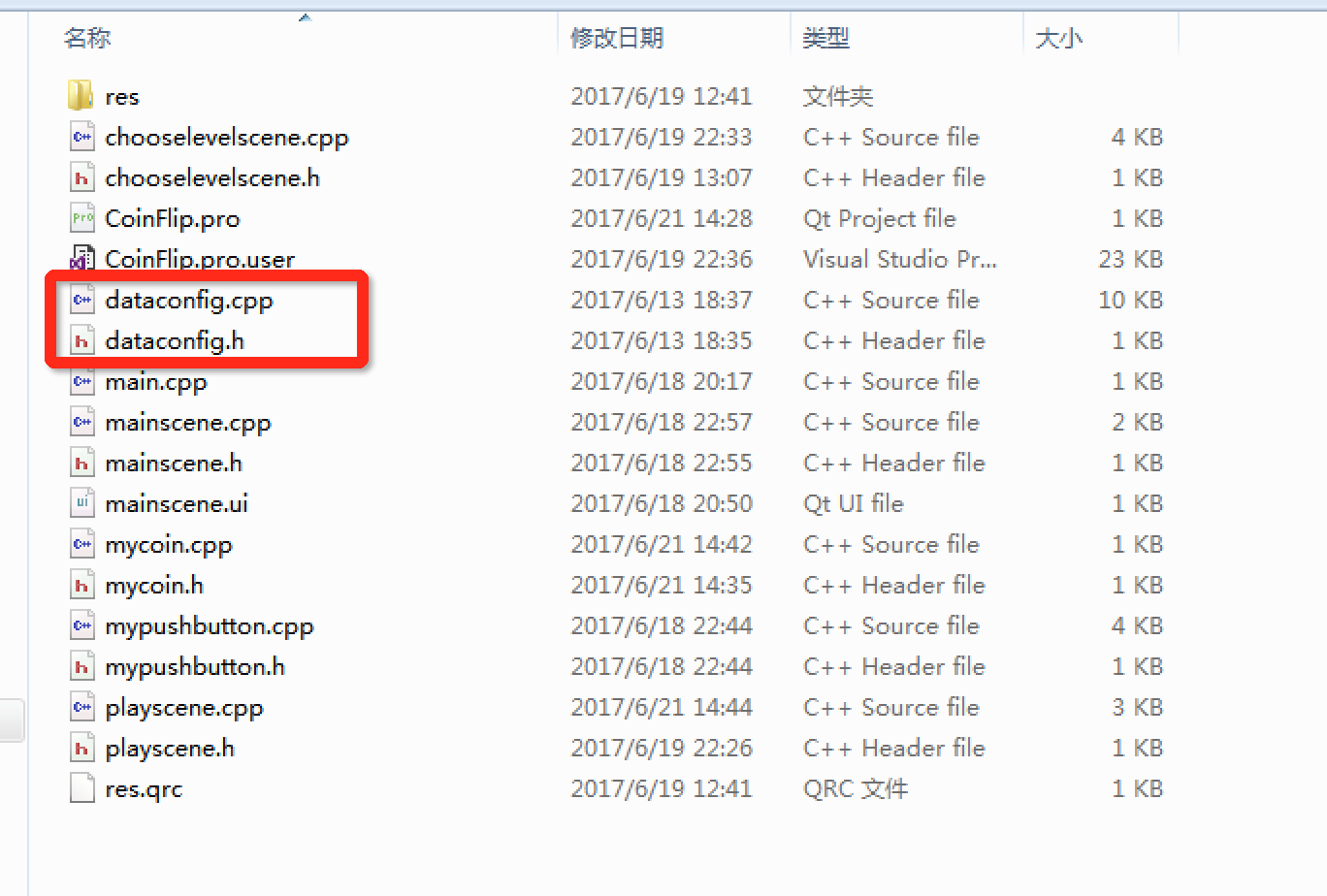


## 5.7 引入關卡資料

當然上述的測試只是為了讓我們知道提供的對外介面可行，但是每個關卡的初始化介面並非如此，因此需要我們引用一個現有的關卡檔，檔中記錄了各個關卡的金幣排列清空，也就是二維陣列的數值。

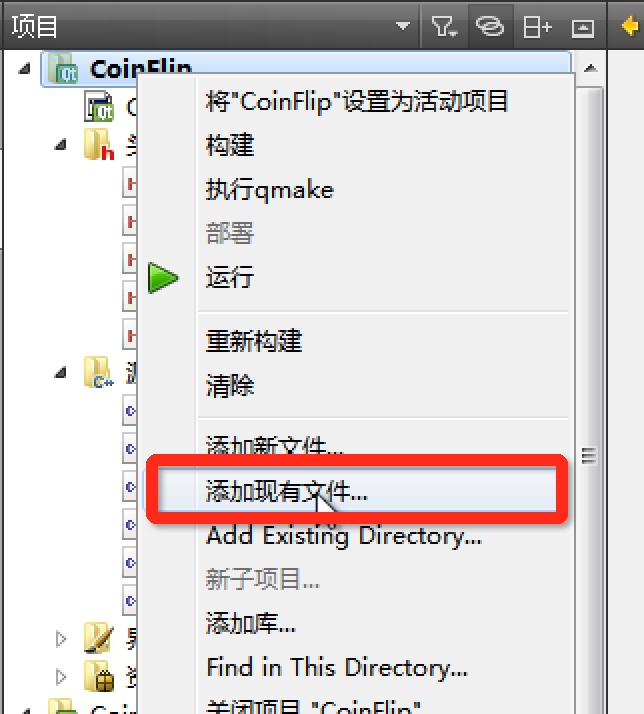
### 5.7.1 添加現有文件dataConfig

首先先將dataConfig.h 和 dataConfig.cpp檔放入到當前專案下：



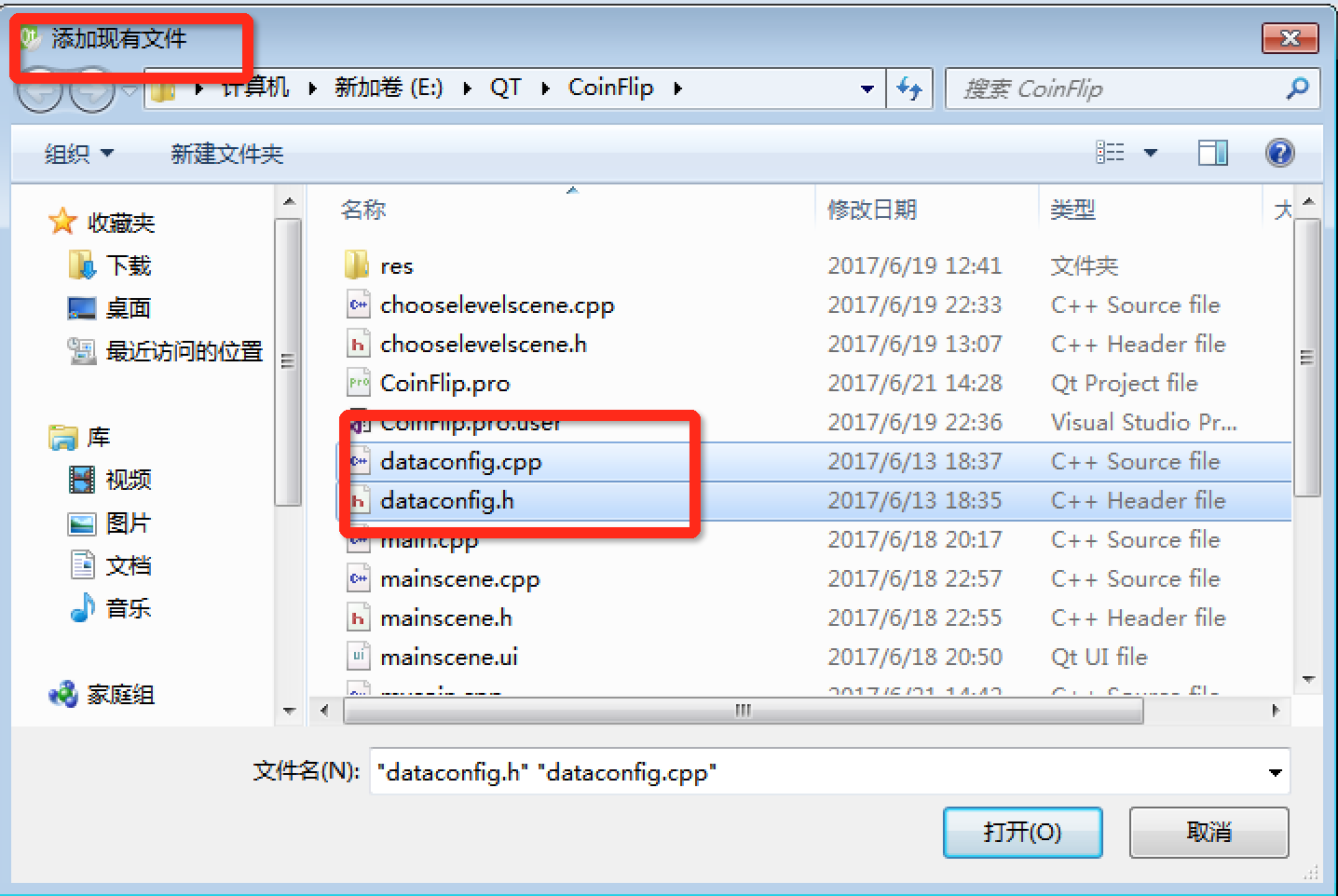
### 5.7.2 添加現有文件

其次在Qt\_Creator專案右鍵，點擊添加現有檔



### 5.7.3 完成添加

選擇當前項目下的檔，並進行添加



### 5.7.4 資料分析

我們可以看到，其實dataConfig.h中只有一個資料是對外提供的，如下圖



在上圖中，QMap<int,QVector<QVector<int>>>mData;都記錄著每個關卡中的資料。

其中，int代表對應的關卡 ，也就是QMap中的key值，而value值就是對應的二維陣列，我們利用的是 QVector<QVector<int>>來記錄著其中的二維陣列。

### 5.7.5 測試關卡資料

在Main函數可以測試第一關的資料，添加如下代碼：

dataConfig config;

for(int i = 0 ; i < 4;i++)

{

for(int j = 0 ; j < 4; j++)

{

//列印第一關所有資訊

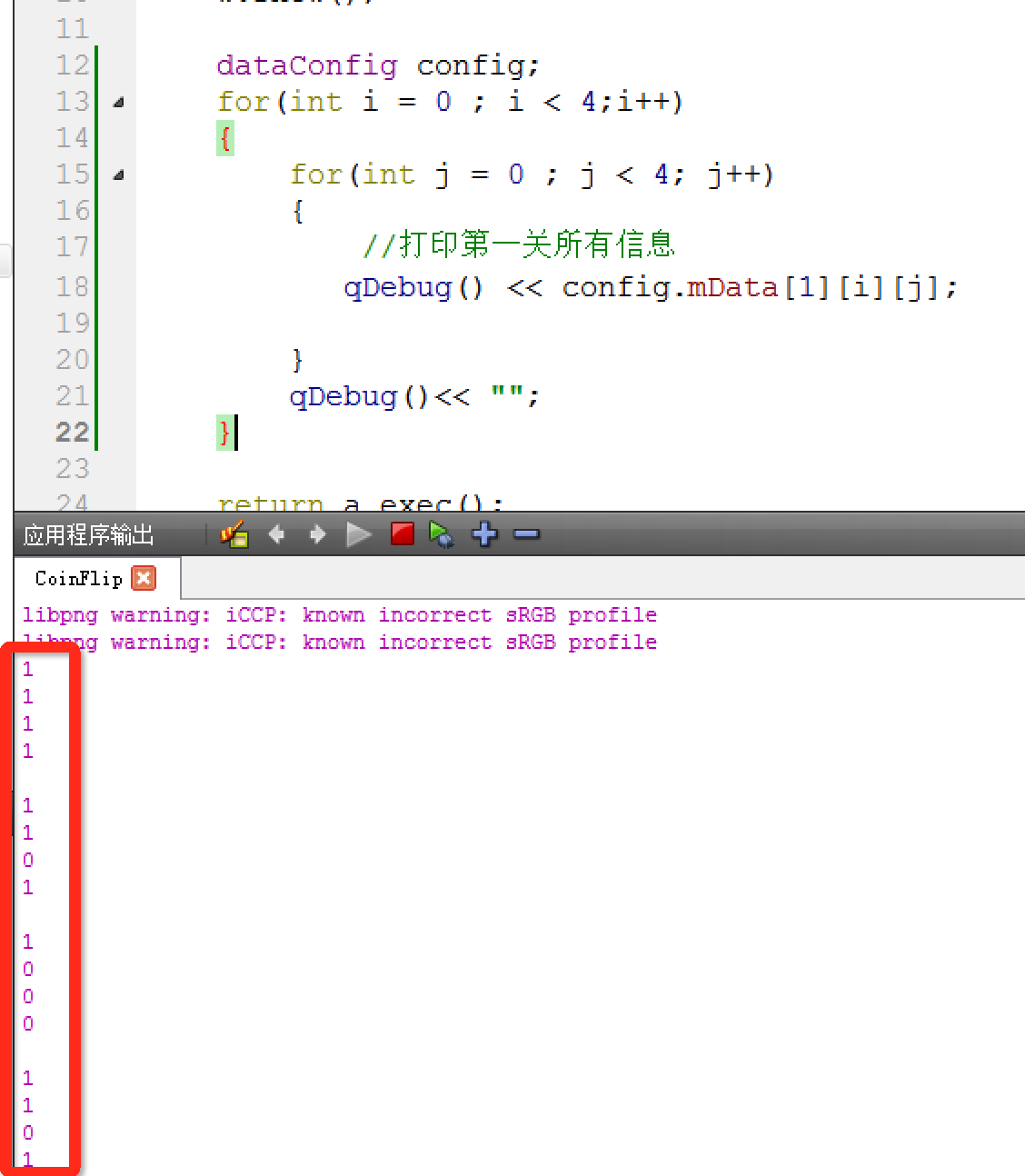
qDebug() << config.mData[1][i][j];

}

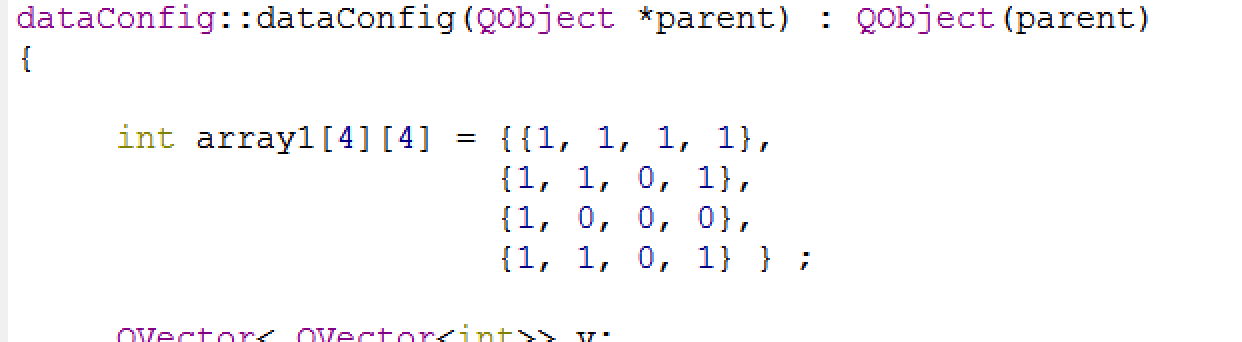
qDebug()<< "";

}

輸出結果如下圖：



對應著dataConfig.cpp中第一關資料來看，與之匹配成功，以後我們就可以用dataConfig中的資料來對關卡進行初始化了



## 5.8 初始化各個關卡

首先，可以在playScene中聲明一個成員變數，使用者記錄當前關卡的二維陣列

int gameArray[4][4]; //二維陣列資料

之後，在.cpp檔中，初始化這個二維陣列

//初始化二維陣列

dataConfig config;

for(int i = 0 ; i < 4;i++)

{

for(int j = 0 ; j < 4; j++)

{

gameArray[i][j] = config.mData[this->levalIndex][i][j];

}

}

初始化成功後，在金幣類 也就是MyCoin類中，擴展屬性 posX，posY，以及flag

這三個屬性分別代表了，該金幣在二維陣列中 x的座標，y的座標，以及當前的正反標誌。

int posX; //x座標

int posY; //y座標

bool flag; //正反標誌

然後完成金幣初始化，代碼如下：

//金幣對象

QString img;

if(gameArray[i][j] == 1)

{

img = ":/res/Coin0001.png";

}

else

{

img = ":/res/Coin0008.png";

}

MyCoin \* coin = new MyCoin(img);

coin->setParent(this);

coin->move(59 + i\*50,204+j\*50);

coin->posX = i; //記錄x座標

coin->posY = j; //記錄y座標

coin->flag =gameArray[i][j]; //記錄正反標誌

運行測試各個關卡初始化，例如第一關效果如圖：



## 5.9 翻金幣特效

### 5.9.1 MyCoin類擴展屬性和行為

關卡的初始化完成後，下面就應該點擊金幣，進行翻轉的效果了，那麼首先我們先在MyCoin類中創建出該方法。

在MyCoin.h中聲明：

void changeFlag();//改變標誌,執行翻轉效果

QTimer \*timer1; //正面翻反面 計時器

QTimer \*timer2; //反面翻正面 計時器

int min = 1; //最小圖片

int max = 8; //最大圖片

MyCoin.cpp中做實現

void MyCoin::changeFlag()

{

if(this->flag) //如果是正面，執行下列代碼

{

timer1->start(30);

this->flag = false;

}

else //反面執行下列代碼

{

timer2->start(30);

this->flag = true;

}

}

當然在構造函數中，記得創建出兩個計時器

//初始化計時器

timer1 = new QTimer(this);

timer2 = new QTimer(this);

### 5.9.2 創建特效

當我們分別啟動兩個計時器時，需要在構造函數中做監聽操作，並且做出回應，翻轉金幣，然後再結束計時器。

構造函數中 進行下列監聽代碼：

//監聽正面翻轉的信號槽

connect(timer1,&QTimer::timeout,[=](){

QPixmap pixmap;

QString str = QString(":/res/Coin000%1.png").arg(this->min++);

pixmap.load(str);

this->setFixedSize(pixmap.width(),pixmap.height() );

this->setStyleSheet("QPushButton{border:0px;}");

this->setIcon(pixmap);

this->setIconSize(QSize(pixmap.width(),pixmap.height()));

if(this->min > this->max) //如果大於最大值，重置最小值，並停止計時器

{

this->min = 1;

timer1->stop();

}

});

connect(timer2,&QTimer::timeout,[=](){

QPixmap pixmap;

QString str = QString(":/res/Coin000%1.png").arg((this->max)-- );

pixmap.load(str);

this->setFixedSize(pixmap.width(),pixmap.height() );

this->setStyleSheet("QPushButton{border:0px;}");

this->setIcon(pixmap);

this->setIconSize(QSize(pixmap.width(),pixmap.height()));

if(this->max < this->min) //如果小於最小值，重置最大值，並停止計時器

{

this->max = 8;

timer2->stop();

}

});

5.9.3 測試

監聽每個按鈕的點擊效果，並翻轉金幣

connect(coin,&MyCoin::clicked,[=](){

//qDebug() << "點擊的位置： x = " << coin->posX << " y = " << coin->posY ;

coin->changeFlag();

gameArray[i][j] = gameArray[i][j] == 0 ? 1 : 0; //陣列內部記錄的標誌同步修改

});



### 5.9.3 禁用按鈕

此時，確實已經可以執行翻轉金幣代碼了，但是如果快速點擊，會在金幣還沒有執行一個完整動作之後 ，又繼續開始新的動畫，我們應該在金幣做動畫期間，禁止再次點擊，並在完成動畫後，開啟點擊。

在MyCoin類中加入一個標誌 isAnimation 代表是否正在做翻轉動畫。

bool isAnimation = false; //做翻轉動畫的標誌

在MyCoin做動畫期間加入

this->isAnimation = true;

也就是changeFlag函數中將標誌設為true

加入位置如下：



並且在做完動畫時，將標誌改為false



重寫按鈕的按下事件，判斷如果正在執行動畫，那麼直接return掉，不要執行後續代碼。

代碼如下：

void MyCoin::*mousePressEvent*(QMouseEvent \*e)

{

if(this->isAnimation )

{

return;

}

else

{

return QPushButton::*mousePressEvent*(e);

}

}

## 5.10 翻周圍金幣

將用戶點擊的周圍 上下左右4個金幣也進行延時翻轉，代碼寫到監聽點擊金幣下。

此時我們發現還需要記錄住每個按鈕的內容，所以我們將所有金幣按鈕也放到一個二維陣列中，在.h中聲明

MyCoin \* coinBtn[4][4]; //金幣按鈕陣列

並且記錄每個按鈕的位置

coinBtn[i][j] = coin;



延時翻動其他周圍金幣

QTimer::singleShot(300, this,[=](){

if(coin->posX+1 <=3)

{

coinBtn[coin->posX+1][coin->posY]->changeFlag();

gameArray[coin->posX+1][coin->posY] = gameArray[coin->posX+1][coin->posY]== 0 ? 1 : 0;

}

if(coin->posX-1>=0)

{

coinBtn[coin->posX-1][coin->posY]->changeFlag();

gameArray[coin->posX-1][coin->posY] = gameArray[coin->posX-1][coin->posY]== 0 ? 1 : 0;

}

if(coin->posY+1<=3)

{

coinBtn[coin->posX][coin->posY+1]->changeFlag();

gameArray[coin->posX][coin->posY+1] = gameArray[coin->posX+1][coin->posY]== 0 ? 1 : 0;

}

if(coin->posY-1>=0)

{

coinBtn[coin->posX][coin->posY-1]->changeFlag();

gameArray[coin->posX][coin->posY-1] = gameArray[coin->posX+1][coin->posY]== 0 ? 1 : 0;

}

});

## 5.11 判斷是否勝利

在MyCoin.h中加入 isWin標誌，代表是否勝利。

bool isWin = true; //是否勝利

默認設置為true，只要有一個反面的金幣，就將該值改為false，視為未成功。

代碼寫到延時翻金幣後 進行判斷

//判斷是否勝利

this->isWin = true;

for(int i = 0 ; i < 4;i++)

{

for(int j = 0 ; j < 4; j++)

{

//qDebug() << coinBtn[i][j]->flag ;

if( coinBtn[i][j]->flag == false)

{

this->isWin = false;

break;

}

}

}

如果isWin依然是true，代表勝利了！

if(this->isWin)

{

qDebug() << "勝利";

}

## 5.12 勝利圖片顯示

將勝利的圖片提前創建好，如果勝利觸發了，將圖片彈下來即可

QLabel\* winLabel = new QLabel;

QPixmap tmpPix;

tmpPix.load(":/res/LevelCompletedDialogBg.png");

winLabel->setGeometry(0,0,tmpPix.width(),tmpPix.height());

winLabel->setPixmap(tmpPix);

winLabel->setParent(this);

winLabel->move( (this->width() - tmpPix.width())\*0.5 , -tmpPix.height());

如果勝利了，將上面的圖片移動下來

if(this->isWin)

{

qDebug() << "勝利";

QPropertyAnimation \* animation1 = new QPropertyAnimation(winLabel,"geometry");

animation1->setDuration(1000);

animation1->setStartValue(QRect(winLabel->x(),winLabel->y(),winLabel->width(),winLabel->height()));

animation1->setEndValue(QRect(winLabel->x(),winLabel->y()+114,winLabel->width(),winLabel->height()));

animation1->setEasingCurve(QEasingCurve::OutBounce);

animation1->start();

}

## 5.13 勝利後禁用按鈕

當勝利後，應該禁用所有按鈕的點擊狀態，可以在每個按鈕中加入標誌位元 isWin，如果isWin為true，MousePressEvent直接return掉即可

MyCoin中.h裡添加：

bool isWin = false;//勝利標誌

在滑鼠按下事件中修改為

void MyCoin::*mousePressEvent*(QMouseEvent \*e)

{

if(this->isAnimation|| isWin == true )

{

return;

}

else

{

return QPushButton::*mousePressEvent*(e);

}

}

//禁用所有按鈕點擊事件

for(int i = 0 ; i < 4;i++)

{

for(int j = 0 ; j < 4; j++)

{

coinBtn[i][j]->isWin = true;

}

}

測試，勝利後不可以點擊任何的金幣。

# 6 音效添加

## 6.1 開始音效

QSound \*startSound = new QSound(":/res/TapButtonSound.wav",this);

點擊開始按鈕，播放音效

startSound->play(); //開始音效

## 6.2 選擇關卡音效

在選擇關卡場景中，添加音效

//選擇關卡按鈕音效

QSound \*chooseSound = new QSound(":/res/TapButtonSound.wav",this);

選中關卡後，播放音效

chooseSound->play();

## 6.3 返回按鈕音效

在選擇關卡場景與翻金幣遊戲場景中，分別添加返回按鈕音效如下：

//返回按鈕音效

QSound \*backSound = new QSound(":/res/BackButtonSound.wav",this);

分別在點擊返回按鈕後，播放該音效

backSound->play();

6.4 翻金幣與勝利音效

在PlayScene中添加，翻金幣的音效以及 勝利的音效

//翻金幣音效

QSound \*flipSound = new QSound(":/res/ConFlipSound.wav",this);

//勝利按鈕音效

QSound \*winSound = new QSound(":/res/LevelWinSound.wav",this);

在翻金幣時播放 翻金幣音效

flipSound->play();

勝利時，播放勝利音效

winSound->play();

測試音效，使音效正常播放。

# 7 優化項目

當我們移動場景後，如果進入下一個場景，發現場景還在中心位置，如果想設置場景的位置，需要添加如下下圖中的代碼：

MainScene中添加：



ChooseScene中添加：



測試切換三個場景的進入與返回都在同一個位置下，優化成功。

至此，本案例全部製作完成。