# 项目简介

翻金币项目是一款经典的益智类游戏，我们需要将金币都翻成同色，才视为胜利。首先，开始界面如下：



点击start按钮，进入下层界面，选择关卡：



在这里我们设立了20个关卡供玩家选择，假设我们点击了第1关，界面如下：



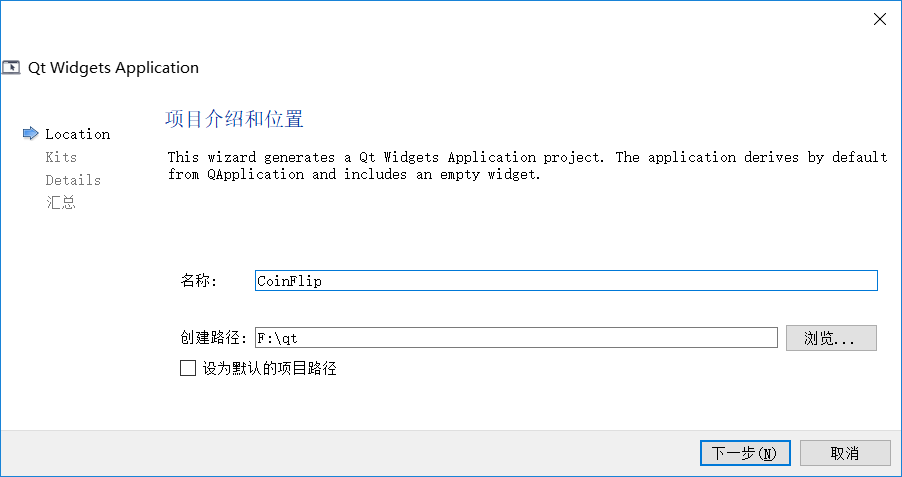
如果想要赢取胜利，我们需要点击上图中红色方框选取的区域，翻动其上下左右的金币，然后当所有金币都变为金色，视为胜利，胜利界面如下：



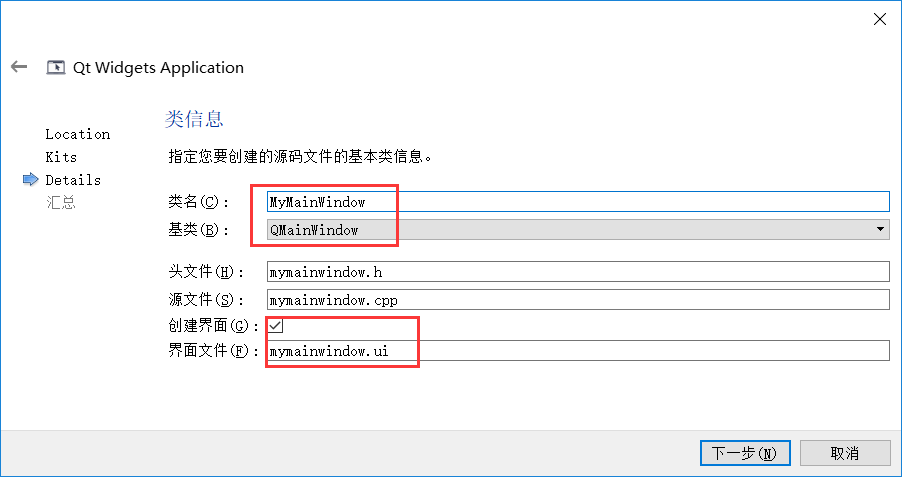
# 项目基本配置

## 创建项目

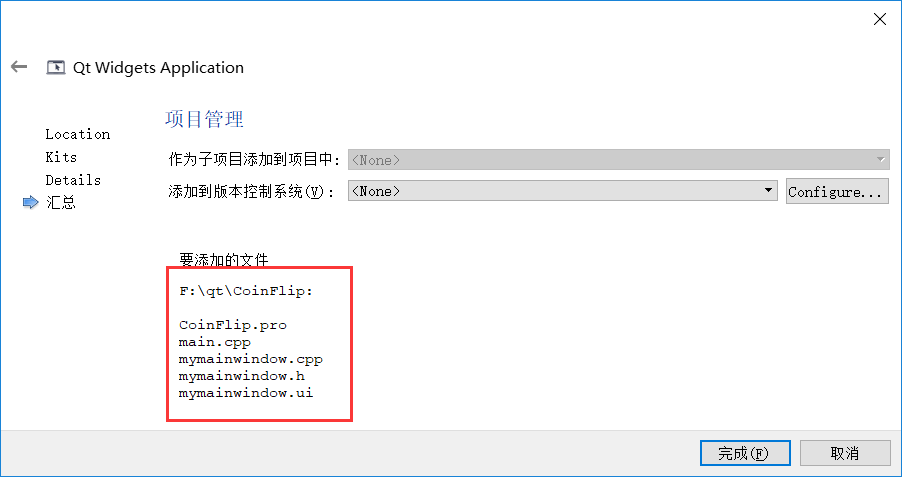
打开Qt-Creator，创建项目：注意名称不要包含空格和回车，路径不要有中文



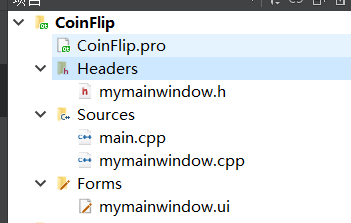
类信息中，选择基类为QMainWindow，类名称为 MyMainWindow，代表各个场景的一个基类。



点击完成，创建出项目：

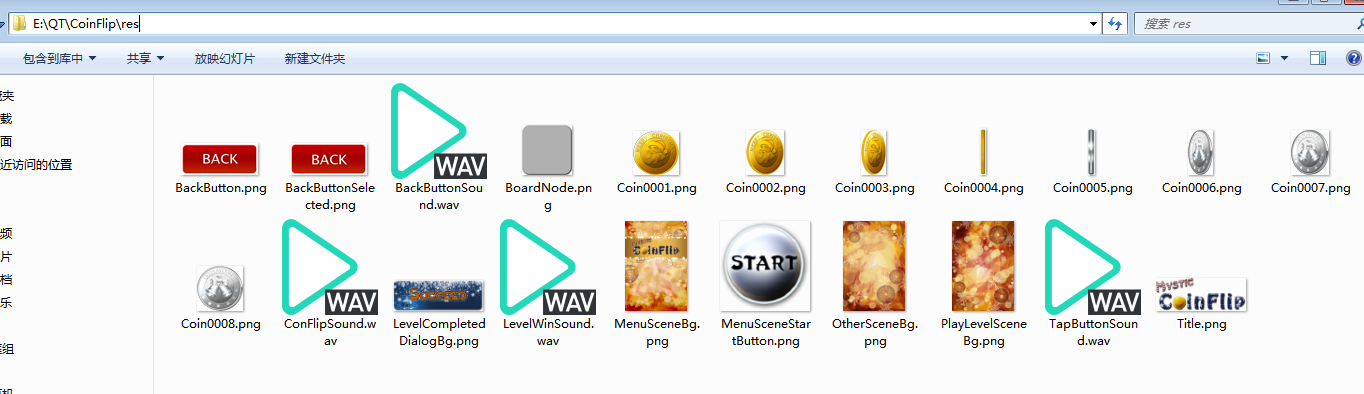


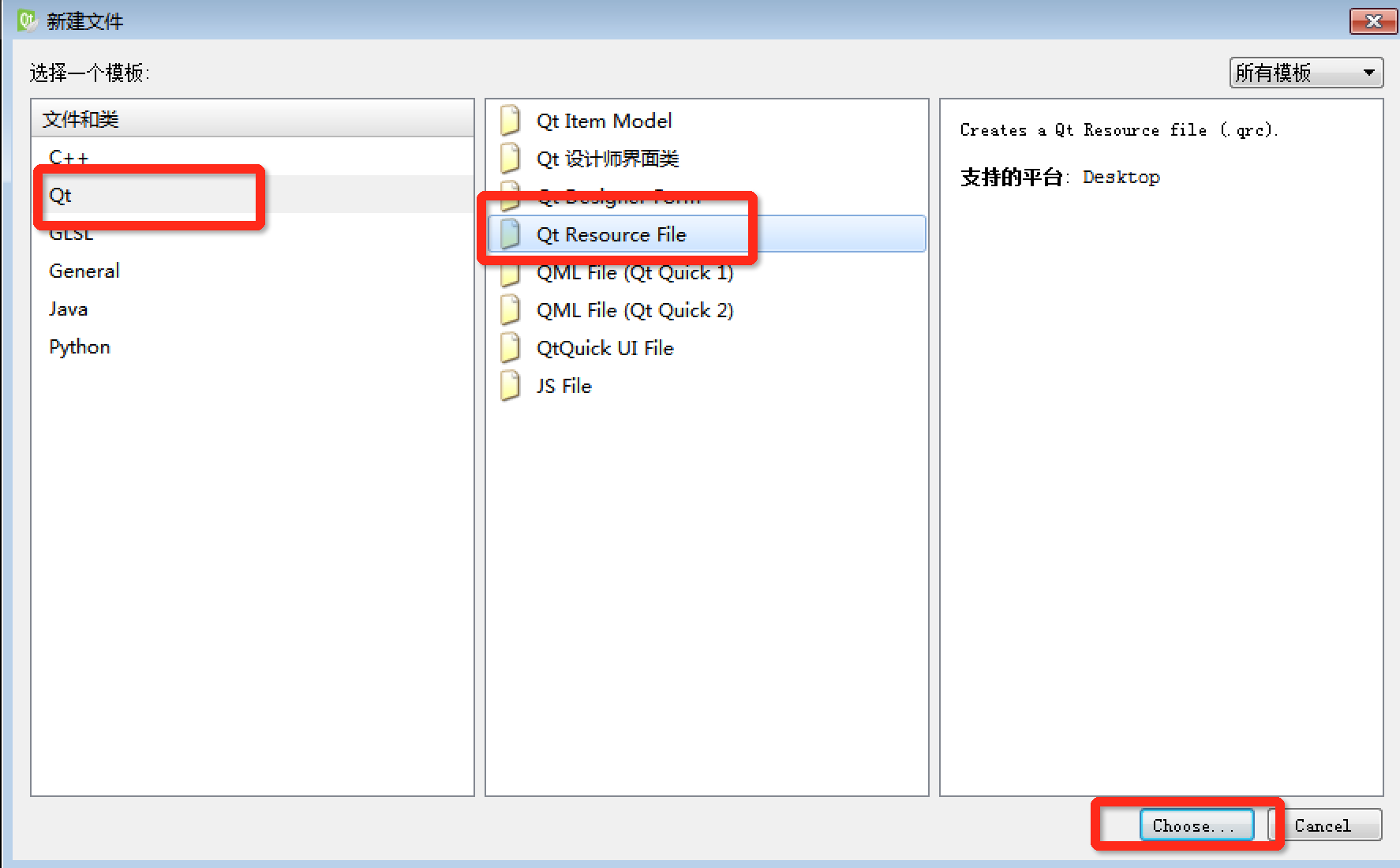
创建的项目结构如下：



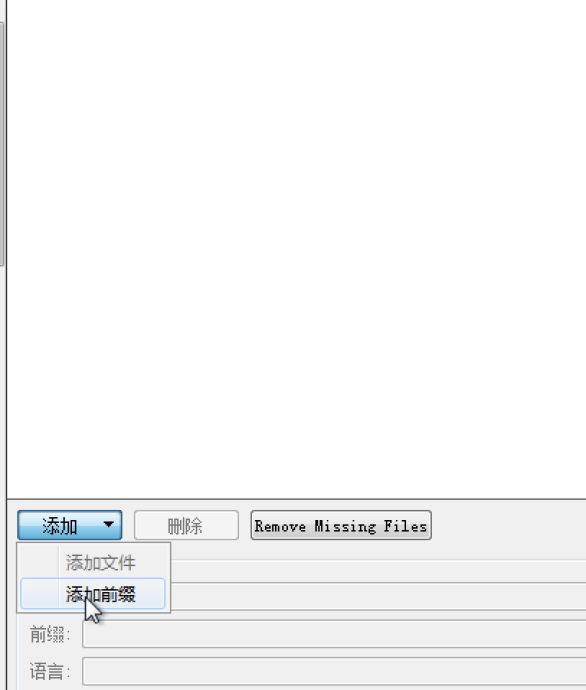
## 添加资源

将资源添加到当前项目下

然后创建.qrc文件

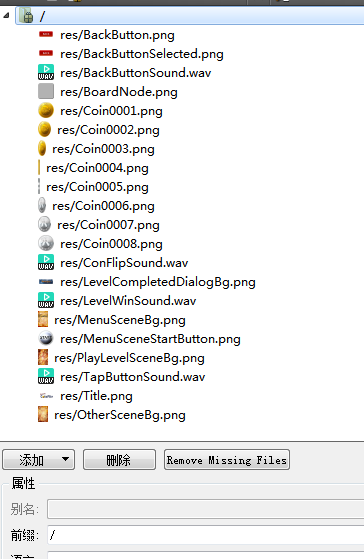


进入编辑模式，添加前缀 “/” ，添加文件





将所有资源文件进行添加



至此将所有需要的资源添加到了本项目中。

# 基类MyMainWindow的参数设置

MyMainWindow是作为一个基类使用，目的抽取游戏中三个场景窗口一些公共的代码出来，比如设置窗口大小，菜单栏等。

## 设定窗口基本参数

MyMainWindow构造函数中（mymainwindow.cpp）添加以下代码

//设置窗口大小、标题等

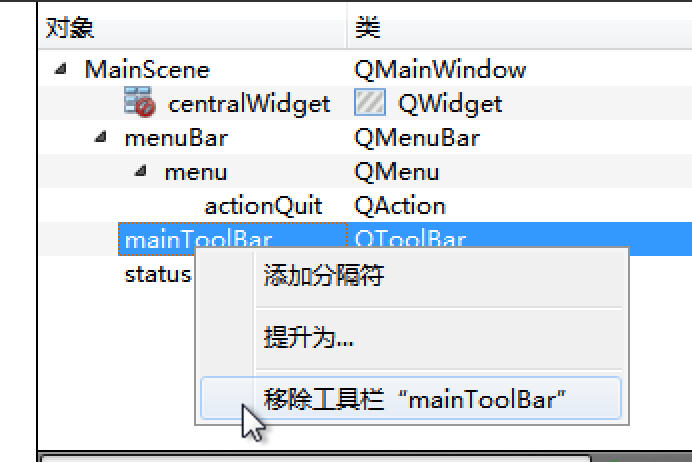
setFixedSize(320,588);

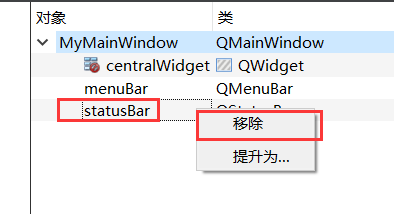
setWindowTitle("翻金币");

setWindowIcon(QIcon(":/res/Coin0001.png"));

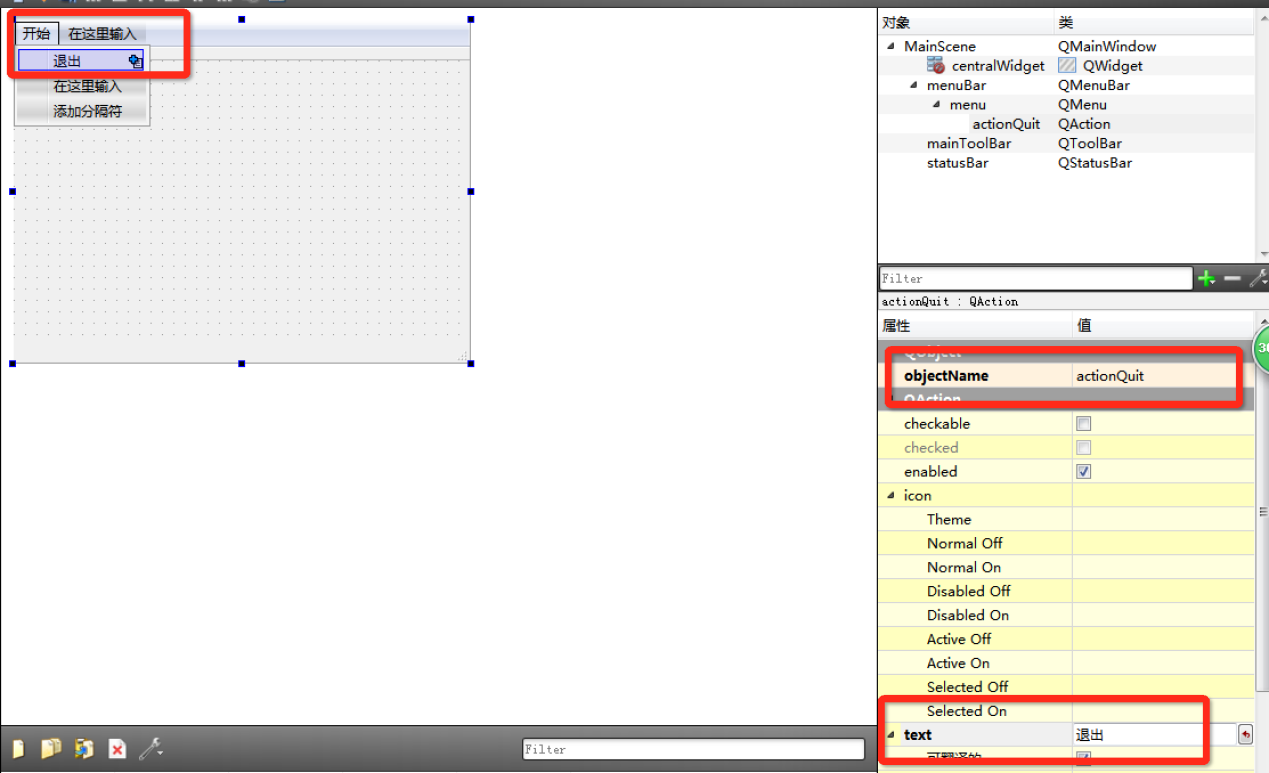
点击mymainwindow.ui文件移除工具栏和状态栏

## 移除自带的工具栏与状态栏





## 菜单栏设计



设计“退出”菜单项，objectName为 actionQuit， text 为 退出；

## 实现菜单栏退出游戏功能

可以通过写代码方式建立信号和槽的连接，不过我们直接用信号槽编辑器建立连接就可以。

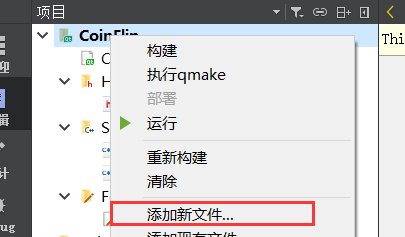


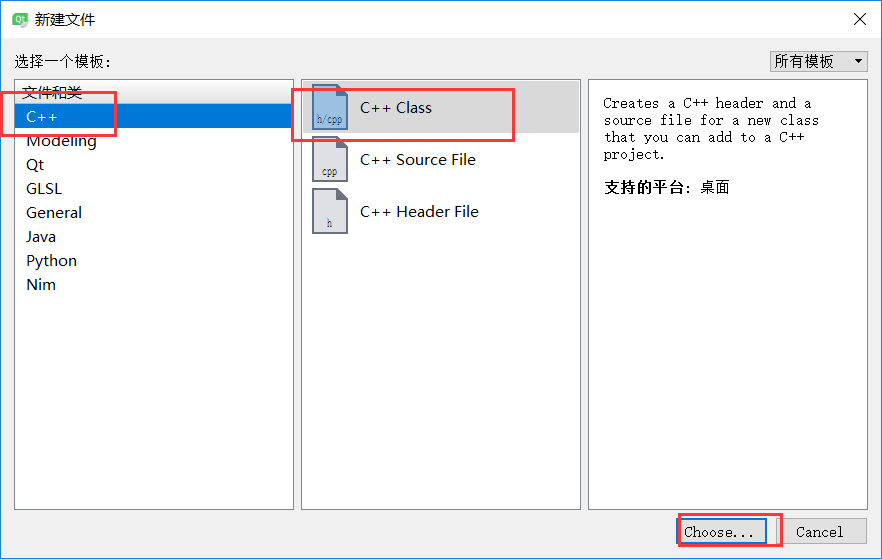
添加一个信号槽的连接，依次选择洗好发送者、信号、信号接收者、槽就可以实现关闭窗口的功能。

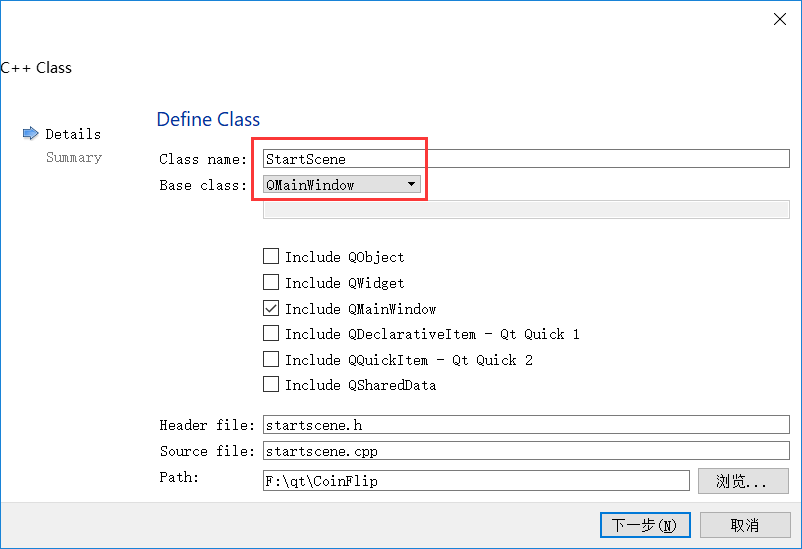


# 开始场景

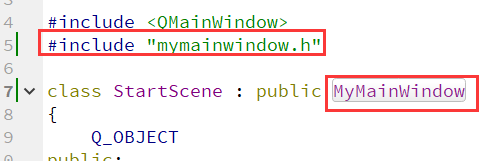
新建一个类StartScene，继承自QMainWindow，表示游戏的开始场景。



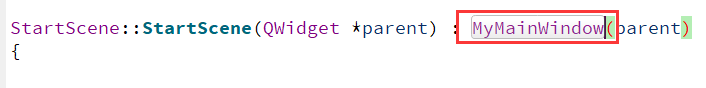




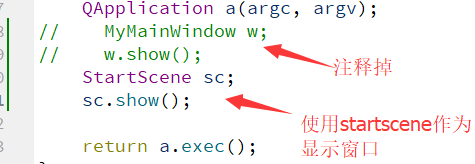
修改刚创建的StartScene类的父类为MyMainWindow，startscene.h修改父类：



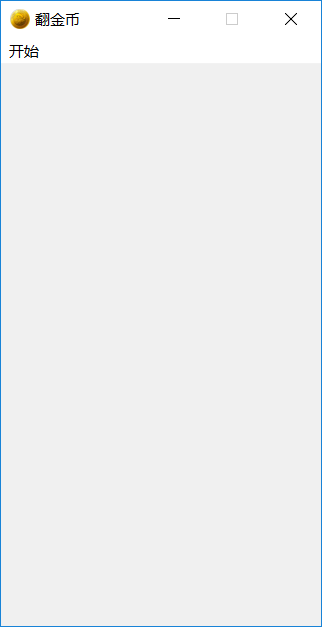
Startscene.cpp修改构造函数参数列表父类:



修改main.cpp中main函数，开始直接显示StartScene的一个实例：



运行效果如图：



此时窗口标题，图标，菜单栏功能都已经完成。

## 设置背景图片

重写StartScene的PaintEvent事件，并添加一下代码，绘制背景图片

void StartScene::***paintEvent***(QPaintEvent \*event)

{

//画家对象

QPainter painter(this);

//背景图片的pixmap

QPixmap pix(":/res/MenuSceneBg.png");

//在0,0坐标开始绘制图片

//这里第三、四个参数指定宽高可以让图片根据窗口大小进行缩放

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

}

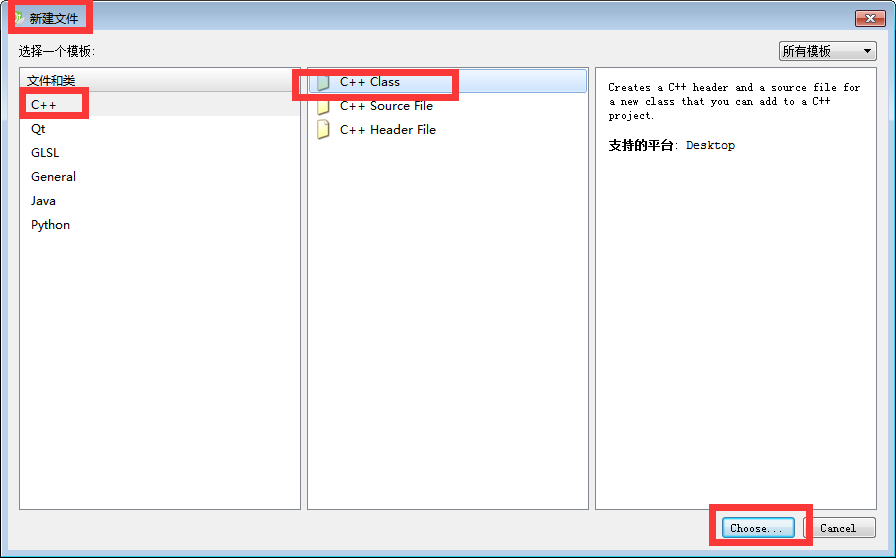
运行效果如图：

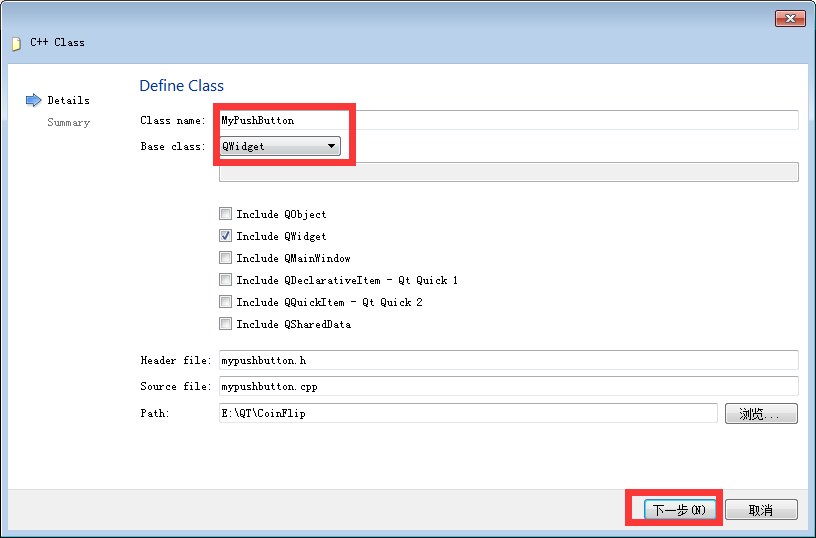


## 创建开始按钮

开始按钮点击后有弹跳效果，这个效果是我们利用自定义控件实现的（QPushButton不会自带这类特效），我们可以自己封装出一个按钮控件，来实现这些效果。

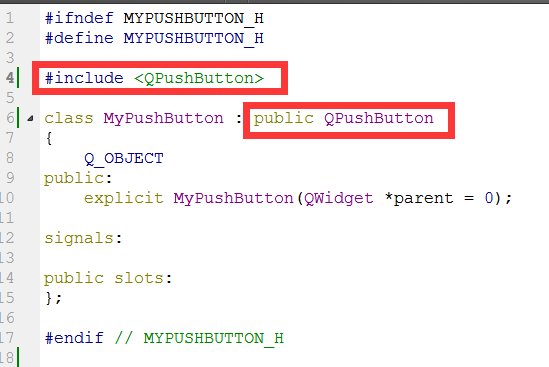
创建MyPushButton，继承与QPushButton





点击完成。

修改MyPushButton的父类



提供MyPushButton的构造的重载版本，可以让MyPushButton提供正常显示的图片以及按下后显示的图片

代码如下：

//normalImg 代表正常显示的图片

//pressImg 代表按下后显示的图片，默认为空

MyPushButton(QString normalImgPath,QString pressImgPath = "",QWidget \*parent = nullptr);

QString mNormalImgPath; //默认显示图片路径

QString mPressedImgPath; //按下后显示图片路径

实现的重载版本MyPushButton构造函数代码如下：

MyPushButton::**MyPushButton**(QString normalImgPath, QString pressImgPath, QWidget \*parent)

//父类构造函数传参

:QPushButton(parent)

//成员变量赋值

,mNormalImgPath(normalImgPath)

,mPressedImgPath(pressImgPath)

{

QPixmap pixmap;

//判断是否能够加载正常显示的图片，若不能提示加载失败

bool ret = pixmap.load(mNormalImgPath);

if(!ret)

{

qDebug() << mNormalImgPath << "加载图片失败!";

}

//设置不规则图片的样式表

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

//设置图标

this->setIcon(pix map);

}

重写MyPushButton的绘图事件处理函数paintEvent

void MyPushButton::***paintEvent***(QPaintEvent \*event)

{

//这里不需要调用父类QPushButton的paintEvent

//因为我们要重新实现勾画按钮的方式

//每当setIcon函数被调用的时候都会触发paintEvent

//所以就从this->icon()函数获取设置的图片就行

QPainter painter(this);

QPixmap pix = this->icon().pixmap(QSize(this->width(),this->height()));

painter.drawPixmap(0,0,

this->width(),

this->height(),

pix);

}

回到MainScene的构造函数中，创建开始按钮

StartScene::**StartScene**(QWidget \*parent) : MyMainWindow(parent)

{

//开始按钮创建，传入默认显示的图片

//按下的图片不用设置

MyPushButton \*btnStart =

new MyPushButton(

":/res/MenuSceneStartButton.png",

"",

this);

//设置按钮大小为图片大小

btnStart->setFixedSize(QSize(114,114));

//将按钮移动到窗口水平中央

//垂直3/4的位置

btnStart->move(

(this->width()-btnStart->width())/2,

this->height()\*3/4-btnStart->height()/2

);

}

运行效果如图：



不规则的开始按钮添加完成。

## 开始按钮跳跃特效实现

在StartScene的构造函数中连接信号槽，监听开始按钮点击，然后播放动画

connect(btnStart,&MyPushBton::clicked,[=](){

//播放往下走的动画

btnStart->moveDown();

//播放网上走的动画

btnStart->moveUp();

});

moveDown与moveUp 为MyPushButton中扩展的特效代码，具体如下：

void MyPushButton::**moveDown**()

{

//属性动画对象，设置要做动画的对象和属性

//属性动画就会根据时间生成目标对象某个属性过渡的动画

//这里对象就是按钮this，属性是geometry（包含x,y坐标和宽w高h的一个属性）

QPropertyAnimation \*animation =

new QPropertyAnimation(this,"geometry");

//设置动画时长100毫秒

animation->setDuration(100);

//设置动画开始时按钮的位置，就用当前位置

animation->setStartValue(this->geometry());

//设置动画结束时按钮的位置，y坐标增加10

animation->setEndValue(

QRect(this->x(),

this->y()+10,

this->width(),

this->height()));

//开始播放动画，参数表示当动画结束后自动删除动画对象

animation->start(QPropertyAnimation::DeleteWhenStopped);

}

void MyPushButton::**moveUp**()

{

QPropertyAnimation \*animation =

new QPropertyAnimation(this,"geometry");

animation->setDuration(100);

animation->setStartValue(this->geometry());

animation->setEndValue(

QRect(this->x(),

this->y()-10,

this->width(),

this->height()));

animation->start(QPropertyAnimation::DeleteWhenStopped);

}

运行代码，点击按钮，测试弹跳效果。发现问题是按钮只顾着往上走，没有先往下再网上。这时因为动画播放start函数并不会阻塞，所以moveDown函数调用后又马上调用了moveUp，此时往下的动画还没播放，就被往上的动画给覆盖了，这时候按钮就只往上边走。解决方法就是在往下的动作播放完成后再播放往上走的动画（使用QTimer定时器）：

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

//播放往下走的动画

btnStart->moveDown();

//当往下走的动画播放完后（100毫秒过后），播放往上走的动画

QTimer::singleShot(100,[=](){

btnStart->moveUp();

});

});

重新运行程序，看效果可以看到动画顺序进行，先往下走再往上走。但又有个问题，频繁点击按钮，按钮会一直往下走，不能回到最初的位置。这是因为当第一个动画往下走了一段距离，第二个动画还没播放完成，这时候按钮的位置就还没有回到原来最初的位置，此时按钮又被点击的话，moveDown函数又被触发，覆盖了moveUp的动画，按钮继续往下走。解决方法就是，从第一个动画播放开始到第二个动画播放完成前的这段时间禁止按钮被点击：

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

//设置按钮不可再点击

//防止在动画播放的过程中又被点击，动画被覆盖按钮位置不能恢复

btnStart->setEnabled(false);

//播放往下走的动画

btnStart->moveDown();

//当往下走的动画播放完后（100毫秒过后），播放往上走的动画

QTimer::singleShot(100,[=](){

btnStart->moveUp();

});

//当两个动画都播放完（200毫秒后），按钮可点击

QTimer::singleShot(200,[=](){

btnStart->setEnabled(true);

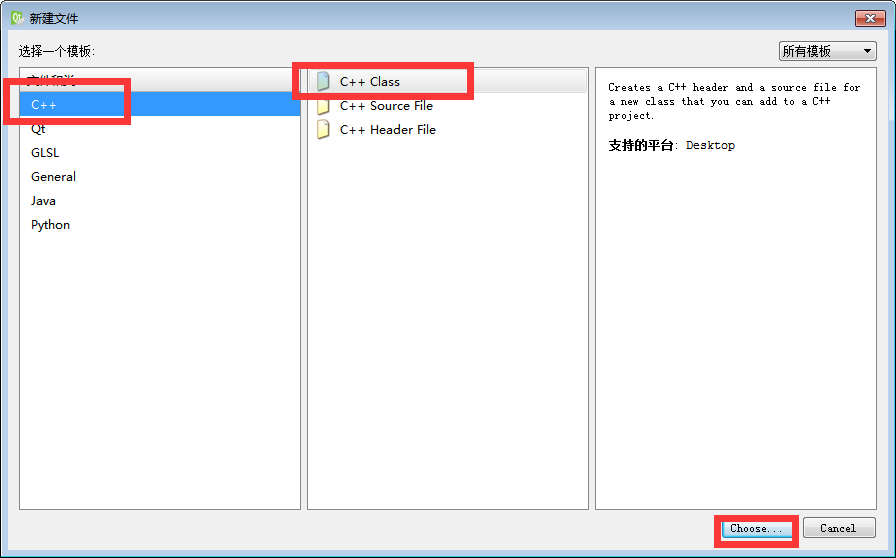
});

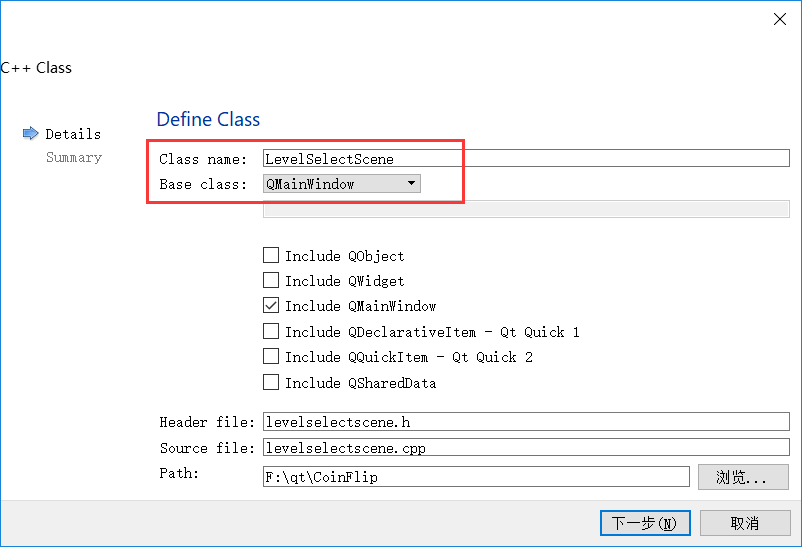
});

## 创建选择关卡场景

点击开始按钮后，进入选择关卡场景。

首先我们先创建选择关卡场景，添加新的C++文件

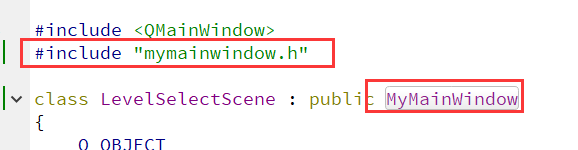




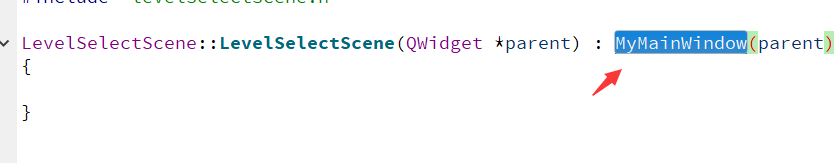
类名为LevelSelectScene 选择基类为QMainWindow，点击下一步，然后点击完成。

修改LevelSelectScene父类为MyMainWindow，复用窗口基本配置的一些代码（菜单栏、标题等）:

Levelselectscene.h头文件修改父类



Levelselectscene.cpp文件修改构造函数父类：



## 点击开始按钮进入选择关卡场景

目前点击主场景的开始按钮，只有弹跳特效，但是我们还需要有功能上的实现，特效结束后，我们应该进入选择关卡场景

在StartScene.h中当按钮动画播放完成后创建选关卡的场景

//当两个动画都播放完（200毫秒后），按钮可点击

QTimer::singleShot(200,[=](){

btnStart->setEnabled(true);

//进入第二个场景

//先创建第二个场景的窗口

LevelSelectScene \*levelSelectScene =

new LevelSelectScene;

//设置场景关闭的时候自动释放内存

levelSelectScene->setAttribute(Qt::WA\_DeleteOnClose);

//显示第二个场景的窗口

levelSelectScene->show();

//隐藏开始场景的窗口

this->hide();

});

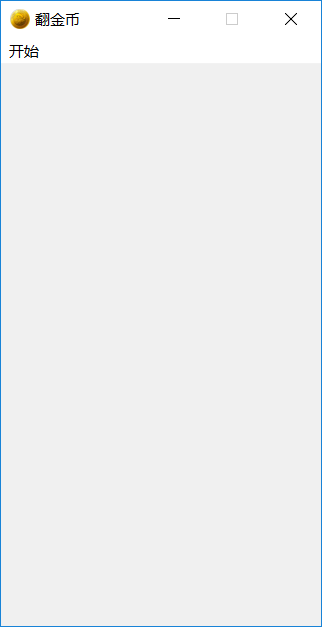
# 4 选择关卡场景

## 场景基本设置

由于关卡场景继承了MyMainWindow，所以窗口大小，标题等配置都已经设定好，这里为了做个小小的区分，在构造函数重新设定我们的窗口标题：

this->setWindowTitle("选择关卡");

运行效果如图：



## 背景设置

同样使用绘图事件处理函数来画背景图

void LevelSelectScene::***paintEvent***(QPaintEvent \*event)

{

QPainter painter(this);

QPixmap pix;

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

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

//加载标题Logo

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

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

}

## 创建返回按钮

在LevelSelectScene构造函数中添加返回按钮，这时候用我们的MyPushButton，按钮位置为窗口右下角

//返回按钮

MyPushButton \* closeBtn =

new MyPushButton(

":/res/BackButton.png",

":/res/BackButtonSelected.png",

this);

closeBtn->setFixedSize(72,32);

//移动到窗口右下角

closeBtn->move(

this->width()-closeBtn->width(),

this->height()-closeBtn->height());

返回按钮是有正常显示图片和点击后显示图片的两种模式，所以我们需要重写MyPushButton中的 MousePressEvent和MouseReleaseEvent

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

{

//别忘了一开始先调用父类mousePressEvent

//否则会导致按钮一些信号不能触发

QPushButton::*mousePressEvent*(e);

//如果有按钮按下的图片(mPressImgPath不是空字符串)，就更换图片

if(!mPressedImgPath.isEmpty())

{

QPixmap pixmap;

//判断是否能够加载正常显示的图片，若不能提示加载失败

bool ret = pixmap.load(mPressedImgPath);

if(!ret)

{

qDebug() << mPressedImgPath << "加载图片失败!";

return;

}

this->setIcon(pixmap);

}

}

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

{

//别忘了一开始先调用父类mousePressEvent

//否则会导致按钮一些信号不能触发

QPushButton::*mouseReleaseEvent*(e);

//如果有按钮按下的图片(mNormalImgPath不是空字符串)，就更换图片

if(!mNormalImgPath.isEmpty())

{

QPixmap pixmap;

//判断是否能够加载正常显示的图片，若不能提示加载失败

bool ret = pixmap.load(mNormalImgPath);

if(!ret)

{

qDebug() << mNormalImgPath << "加载图片失败!";

return;

}

this->setIcon(pixmap);

}

}

## 返回按钮

在这里我们点击返回后，发送自定义信号，告诉外界点击了返回按钮。

//返回按钮功能实现

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

emit this->backBtnClicked();

});

在主场景StartScene中 点击开始按钮显示选择关卡场景的同时，监听选择关卡的返回按钮消息

//监听选择场景的返回按钮

connect(levelSelectScene,&LevelSelectScene::backBtnClicked,[=](){

this->show();

levelSelectScene->close();

});

测试主场景与选择关卡场景的切换功能。

## 创建选择关卡按钮

在LevelSelectScene的构造函数中添加以下代码创建20个自定义按钮

//xy 偏移量

const int xOffset = 25;

const int yOffset = 130;

//行高和列宽

const int columnWidth = 70;

const int rowHeight = 70;

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

{

//先计算行坐标列坐标

//列坐标乘以列宽得到按钮x坐标

//然后行坐标乘以行高得到按钮y坐标

//每个按钮xy坐标再加上偏移量得到最后居显示的效果

int row = i/4; //行坐标

int column = i%4; //列坐标

int x = xOffset + column\*columnWidth;

int y = yOffset + row\*rowHeight;

MyPushButton \*btn = new MyPushButton(":/res/LevelIcon.png","",this);

//设定按钮文字i+1

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

//setGeometry其实是move和resize的结合体

btn->setGeometry(

x, //x坐标

y, //y坐标

57, //按钮宽

57 //按钮高

);

}

运行效果如下，发现没有文字显示，因为我们重写了paintEvent只是显示图片，没有显示文字：



重新调整MyPushButton的paintEvent，让文字居中显示：

void MyPushButton::***paintEvent***(QPaintEvent \*event)

{

//这里不需要调用父类QPushButton的paintEvent

//因为我们要重新实现勾画按钮的方式

//每当setIcon函数被调用的时候都会触发paintEvent

//所以就从this->icon()函数获取设置的图片就行

QPainter painter(this);

QPixmap pix = this->icon().pixmap(QSize(this->width(),this->height()));

painter.drawPixmap(0,0,

this->width(),

this->height(),

pix);

//绘制按钮文字，水平和垂直都居中

painter.drawText(

0,

0,

this->width(),

this->height(),

Qt::AlignVCenter|Qt::AlignHCenter,

this->text());

}

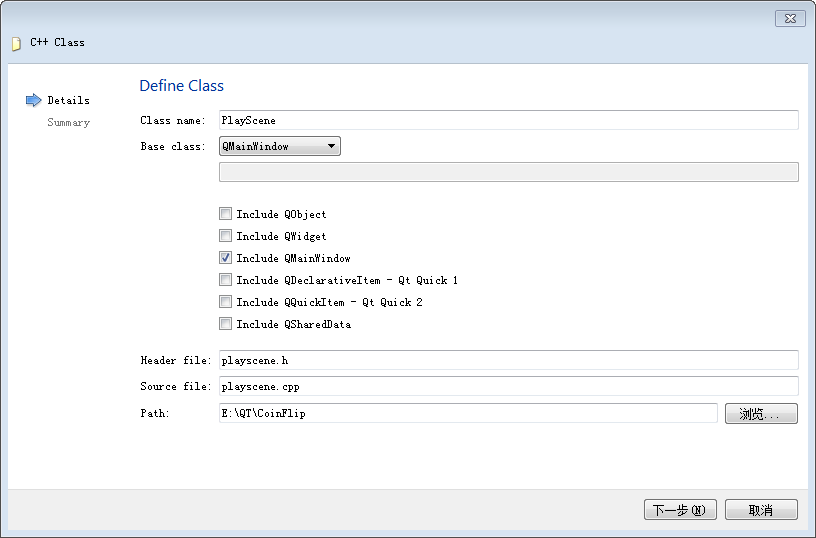
最终效果：



## 创建翻金币场景

点击关卡按钮后，会进入游戏的核心场景，也就是翻金币的场景，首先先创建出该场景的.h和.cpp文件

创建PlayScene



点击选择关卡按钮后会跳入到该场景

建立点击按钮，跳转场景的信号槽连接，在创建关卡按钮的for循环内：

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

//点击进入游戏场景，同时传递关卡数给游戏场景

this->hide();

//构造函数中传递关卡数

PlayScene \*playScene = new PlayScene(i+1);

playScene->show();

playScene->setAttribute(Qt::WA\_DeleteOnClose);

});

这里PlayScene \*playScene = new PlayScene(i+1); 将用户所选的关卡号发送给playScene，也就是翻金币场景，当然PlayScene 要提供重载的有参构造版本，来接受这个参数，（看下一节）。

# 翻金币场景

## 场景基本设置

PlayScene.h中 声明成员变量，用于记录当前用户选择的关卡

private:

//成员变量 记录关卡索引

int mLevel ;

同时默认的构造函数为

public:

PlayScene(int level , QWidget \*parent = nullptr);

以和关卡选择场景同样的方式添加一个返回按钮和返回按钮被点击的信号

playscene.h文件中添加信号：

signals:

//返回按钮被点击的信号

void **backBtnClicked**();

playscene构造函数中添加：

PlayScene::**PlayScene**(int level, QWidget \*parent) : MyMainWindow(parent)

{

//返回按钮

MyPushButton \* closeBtn =

new MyPushButton(

":/res/BackButton.png",

":/res/BackButtonSelected.png",

this);

closeBtn->setFixedSize(72,32);

//移动到窗口右下角

closeBtn->move(

this->width()-closeBtn->width(),

this->height()-closeBtn->height());

//点击返回按钮发送backBtnClicked信号

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

emit this->backBtnClicked();

});

}

LevelSelectScene中创建playscene之后也要监听返回按钮被点击的信号



## 背景设置

重写PlayScene的paintEvent函数

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

{

QPainter painter(this);

//搬动画家到菜单栏以下位置画画，否则画出来的

//东西有可能会被菜单栏盖掉一部分

painter.translate(0,this->menuBar()->height());

QPixmap pix;

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

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

//加载标题Logo

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

painter.drawPixmap(0,0,pix.width()\*0.5,pix.height()\*0.5,pix);

}

## 显示当前关卡

//当前关卡标题

QLabel \* label = new QLabel;

label->setParent(this);

QFont font;

font.setFamily("华文新魏");

font.setPointSize(20);

label->setFont(font);

QString str = QString("Level: %1").arg(this->mLevel);

label->setText(str);

label->setGeometry(QRect(30, this->height() - 50,150, 50)); //设置大小和位置

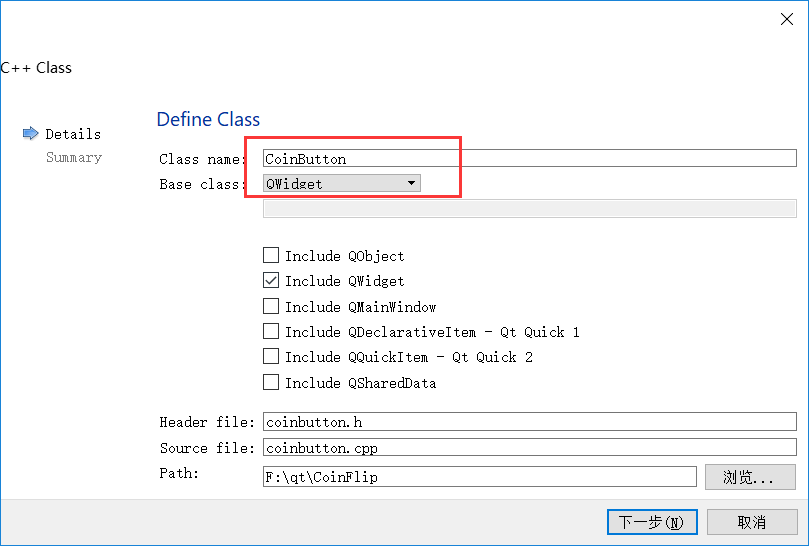
假设我们选择了第15关卡，运行效果如果：



## 创建金币按钮类

我们知道，金币是本游戏的核心对象，并且在游戏中可以利用二维数组进行维护，可以点击，拥有支持点击，翻转特效等特殊性，因此不妨将金币单独封装到一个类中继承自QPushButton，完成金币所需的所有功能。

右键项目创建新C++类继承自QWidget，然后再修改父类为QPushButton



CoinButton可以添加一个表示硬币状态的变量mStat，使用0代表银币，1代表金币，同时创建他对的get、set方法，以及设置0或者1的时候设置他的icon为银币或者金币。

CoinButton.h添加：

//表示硬币状态的变量

//0是银币，1是金币

int mStat;

int **stat**() const;

void **setStat**(int stat);

CoinButton.cpp添加：

int CoinButton::**stat**() const

{

return mStat;

}

void CoinButton::**setStat**(int stat)

{

mStat = stat;

if(mStat)

{

//设置金币icon

setIcon(":/res/Coin0001.png");

}

else

{

//设置银币icon

setIcon(":/res/Coin0008.png");

}

//设置icon大小为按钮大小

setIconSize(this->size());

}

## 创建金币按钮矩阵

为了便于验证代码效果，还是先将CoinButton显示出来

创建4x4的按钮矩阵，显示到PlayScene中，在PlayScene的构造函数过程中添加以下代码：

//行数、列数

const int rowCount = 4;

const int columnCount = 4;

//行高、列宽

const int rowHeight = 50;

const int columnWidth = 50;

//x、y偏移量

const int xOffset = 57;

const int yOffset = 200;

//每个按钮宽、高

const int coinWidth = 50;

const int coinHeight = 50;

for(int row = 0 ; row < rowCount ; ++row)

{

for(int column = 0 ; column < columnCount ; ++column)

{

//按钮坐标计算

//x = 列坐标column \* 列宽columnWidth + x偏移

//y = 行坐标row \* 行高rowHeight + y偏移

int x = column \* columnWidth + xOffset;

int y = row \* rowHeight + yOffset;

CoinButton \*btn = new CoinButton(this);

//设置按钮的坐标以及按钮大小

btn->setGeometry(

x,

y,

coinWidth,

coinHeight

);

//设定按钮金币银币状态

btn->setStat(1);

}

}

效果如下：



### 构造函数

为了让完全定义金币按钮的绘图方式，首先我们先去掉按钮边框，在CoinButton的构造函数中加入：

//去掉按钮边框

setStyleSheet("QPushButton{border:None;}");



### 勾画硬币按钮背景

重写CoinButton的paintEvent

void CoinButton::***paintEvent***(QPaintEvent \*event)

{

QPainter painter(this);

QPixmap pix(":/res/BoardNode.png");

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

//勾画完背景后，这里需要调用父类的绘图函数

//因为我们要使用按钮的Icon属性，方便我们开发

//而绘制图标Icon的代码就在QPushButton的绘图函数中

QPushButton::*paintEvent*(event);

}



### 点击硬币按钮图片直接反转

在CoinButton中添加函数flip

void CoinButton::**flip**()

{

//置反按钮状态，0会变成1,1会变成0

//同时会导致图标变化

this->setStat(!this->stat());

}

在PlayScene创建金币按钮的循环中添加代码，连接点击信号和槽，置反按钮，实现按钮图片从金币变成银币或者从银币变成金币的功能。

//点击硬币按钮就反转

connect(btn,&CoinButton::clicked,[=](){

btn->flip();

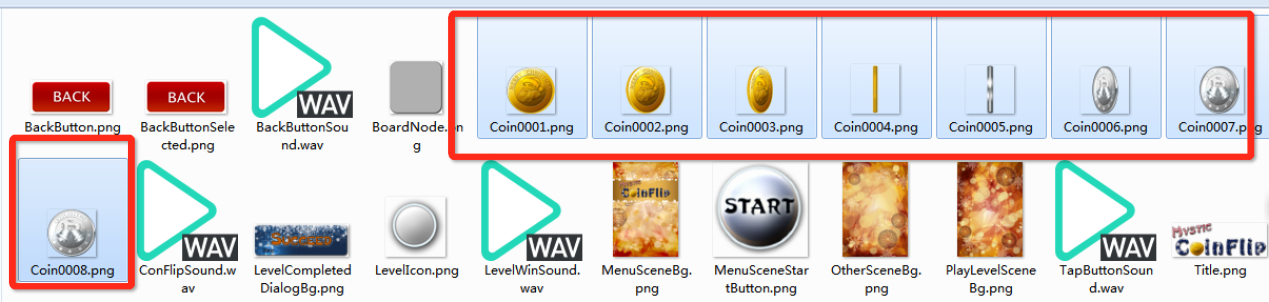
});

此时点击金币按钮后能够马上变成银币，但是还没有动画效果



## 硬币翻转动画

在资源图片中，我们可以看到，金币翻转的效果原理是多张图片切换而形成的，而以下八张图片中，第一张与最后一张比较特殊，因此我们在给用户看的时候，无非是金币Coin0001或者是银币 Coin0008这两种图。所以要制作动画，我们就要想办法实现在一段时间内，每间隔n毫秒就切换按钮一张图片的效果。



此时我们应该想到用定时器，每隔一段时间就触发一次，将按钮的图片换成下一帧要显示的图片，因此这里我们还要记录当前播放的帧序号，同时当帧序号到达金币帧或者银币帧的时候就停止定时器。

在CoinButton中添加以下声明：

//设置状态，同时播放动画

void **setStatWithAnimation**(int stat);

//存储当前动画帧序号

int mFrameNo;

//定时器切换图片的定时器

QTimer \*mTimer;

在CoinButton.cpp中添加代码：

void CoinButton::**setStatWithAnimation**(int stat)

{

//首先设置状态值

mStat = stat;

if(mStat)

{

//如果刚才设定的状态是1

//表示要反转到金币

//所以应该是银币反转到金币的过程开始

//帧序号从8开始递减到1

this->mFrameNo = 8;

}

else

{

//如果刚才设定的状态是0

//表示要反转到银币

//所以应该是金币反转到银币的过程开始

//帧序号从1开始递增到8

this->mFrameNo = 1;

}

//启动定时器，每间隔30毫秒触发一次

this->mTimer->start(30);

}

构造函数中：

//初始化mTimer

mTimer = new QTimer(this);

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

//每间隔一段时间切换一张动画

if(mStat)

{

//如果刚才设定的状态是1

//表示要反转到金币

//所以应该是银币反转到金币的过程

//帧序号从8开始递减到1

--mFrameNo;

}

else

{

//如果刚才设定的状态是0

//表示要反转到银币

//所以应该是金币反转到银币的过程

//帧序号从1开始递增到8

++mFrameNo;

}

//如果序号是1或者8，表示帧动画走到了最后一帧，停止定时器

if(mFrameNo==1||mFrameNo==8)

mTimer->stop();

QString iconName = QString(":/res/Coin000%1.png").arg(mFrameNo);

setIcon(QIcon(iconName));

});

调整CoinButton的flip函数

void CoinButton::**flip**()

{

//置反按钮状态，0会变成1,1会变成0

//同时会导致图标变化

// this->setStat(!this->stat());

this->setStatWithAnimation(!this->stat());

}

此时点击单个硬币按钮反转动画完成

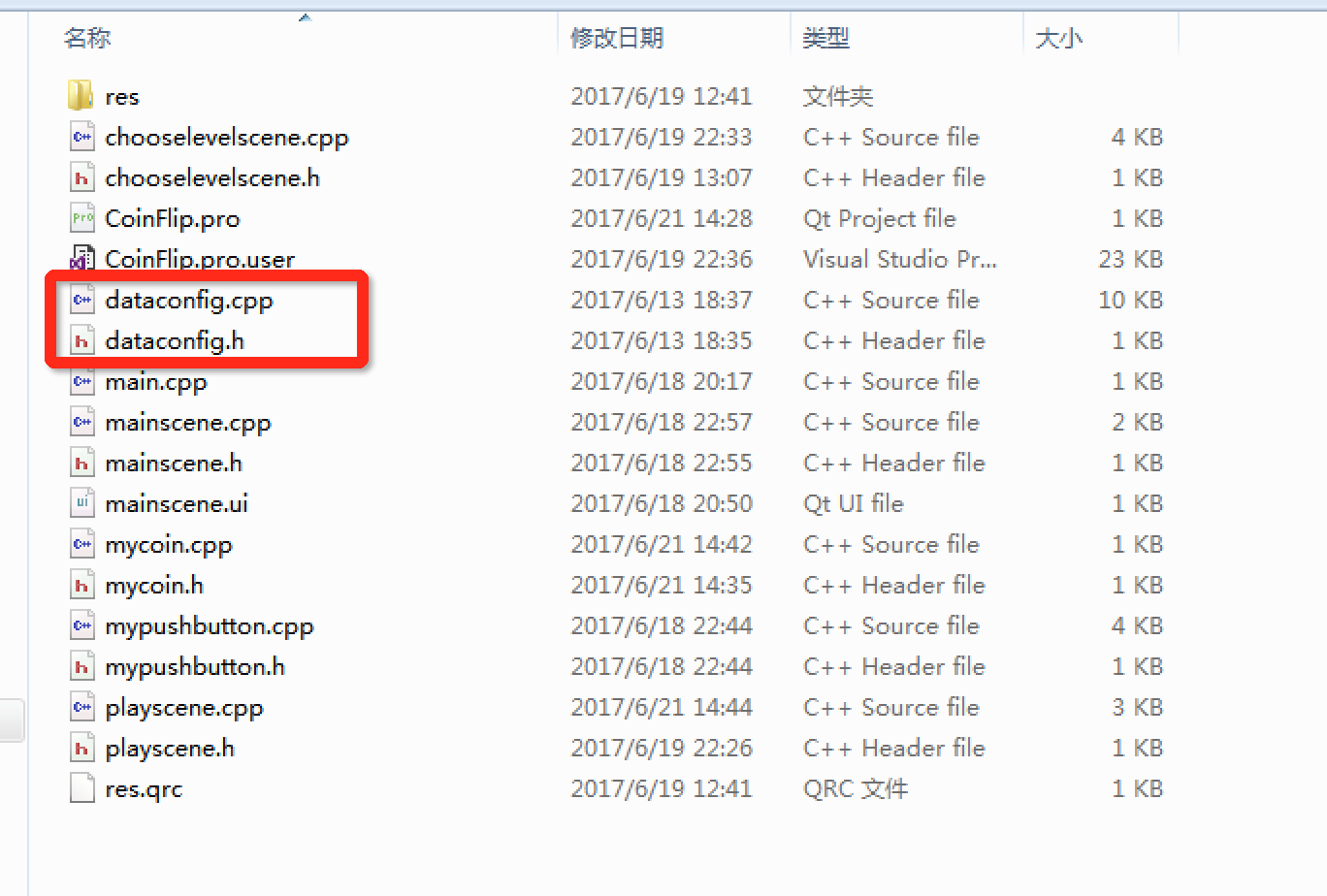


## 引入关卡数据

当然上述的测试只是为了让我们知道提供的对外接口可行，但是每个关卡的初始化界面并非如此，因此需要我们引用一个现有的关卡文件，文件中记录了各个关卡的金币排列清空，也就是二维数组的数值。

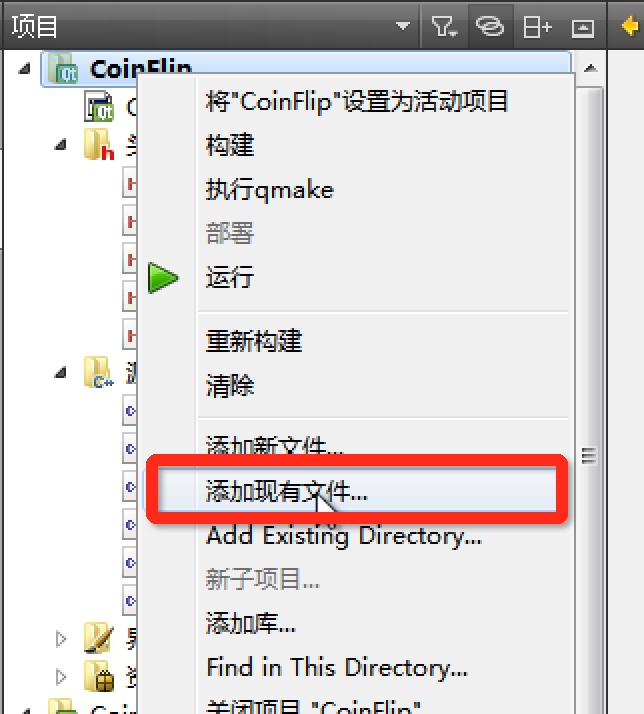
### 添加现有文件dataConfig

首先先将dataConfig.h 和 dataConfig.cpp文件放入到当前项目下：



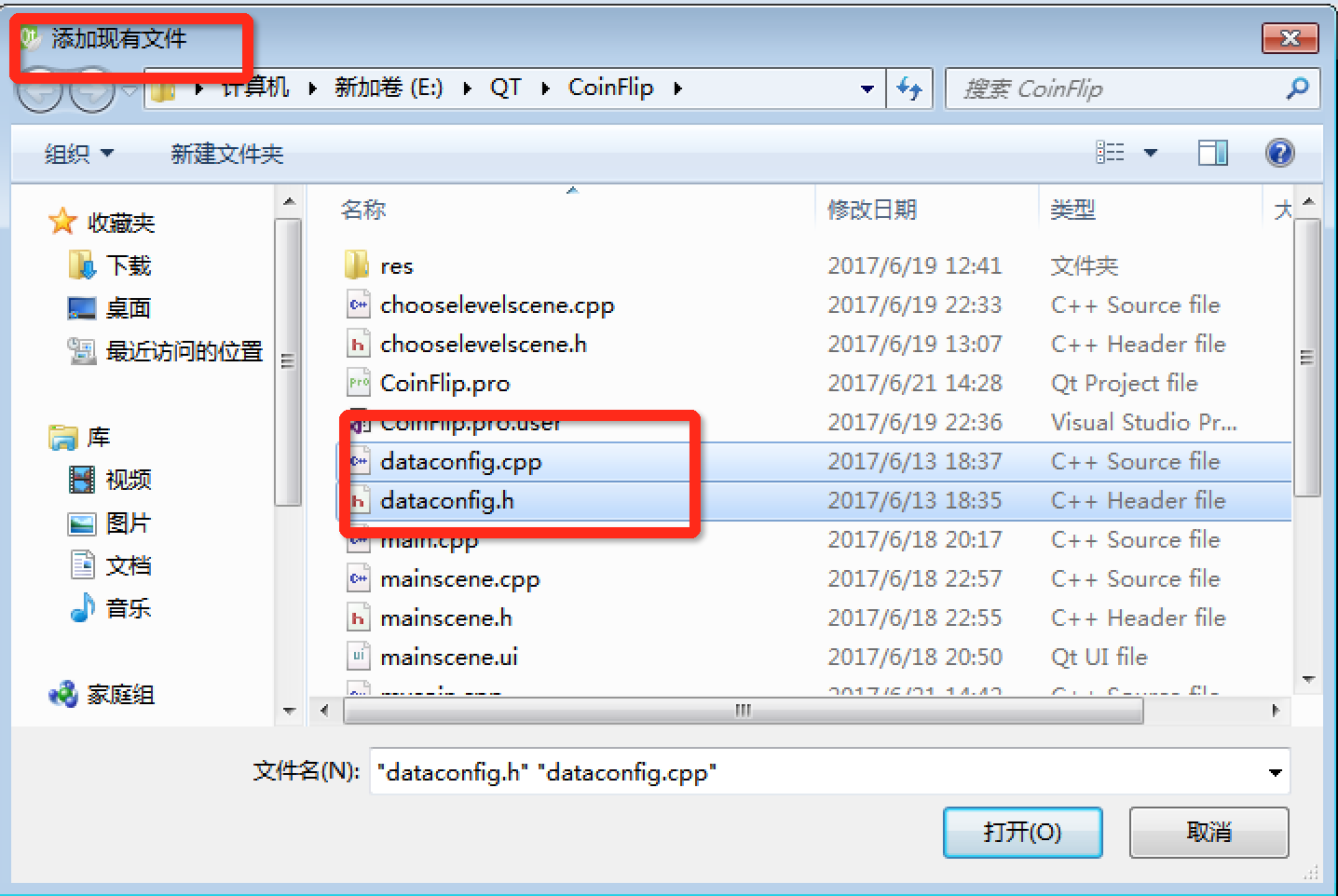
### 添加现有文件

其次在Qt\_Creator项目右键，点击添加现有文件



### 完成添加

选择当前项目下的文件，并进行添加



### 数据分析

我们可以看到，其实dataConfig.h中只有一个数据是对外提供的，如下图

//硬币数据映射表

//key为关卡数（1-20）

//value为一个二维整形数组，1表示金币，0表示银币

QMap<int, QVector< QVector<int> > >mData;

在上面代码中，QMap<int,QVector<QVector<int>>>mData;都记录着每个关卡中的数据。

其中，int代表对应的关卡 ，也就是QMap中的key值，而value值就是对应的二维数组，我们利用的是 QVector<QVector<int>>来记录着其中的二维数组。

对应着dataConfig.cpp中第一关数据来看，与之匹配成功，以后我们就可以用dataConfig中的数据来对关卡进行初始化了

DataConfig::**DataConfig**(QObject \*parent) : QObject(parent)

{

int array1[4][4] = {{1, 1, 1, 1},

{1, 1, 0, 1},

{1, 0, 0, 0},

{1, 1, 0, 1} } ;

QVector< QVector<int>> v;

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

{

QVector<int>v1;

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

{

v1.push\_back(array1[i][j]);

}

v.push\_back(v1);

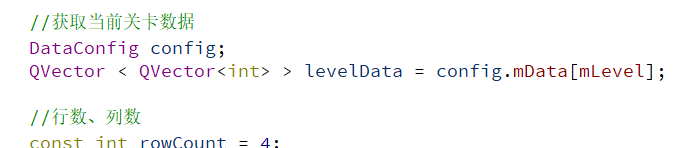
}

mData.insert(1,v);

…

## 初始化各个关卡

在PlayScene的初始化金币按钮之前，创建一个DataConfig对象



在初始化按钮的过程中，设置按钮状态改为：



运行测试各个关卡初始化，例如第一关效果如图：



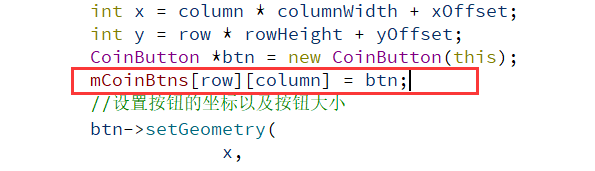
## 翻动周围金币

将用户点击的周围 上下左右4个金币也进行延时翻转，代码写到监听点击金币下。

此时我们发现还需要记录住每个按钮的内容，所以我们将所有金币按钮也放到一个二维数组中，PlayScene.h中声明

CoinButton \*mCoinBtns[4][4]; //金币按钮数组

在初始化16个按钮的过程中同时初始化mCoinBtns



PlayScene添加反转某个硬币的函数：

头文件声明：

void **flipCoin**(int row , int column);

金币按钮的点击信号槽lambda表达式改为：

//点击硬币按钮就反转

connect(btn,&CoinButton::clicked,[=](){

this->flipCoin(row,column);

});

flipCoin代码实现：

void PlayScene::**flipCoin**(int row, int col)

{

//翻动row行col列的按钮

this->mCoinBtns[row][col]->flip();

//延时翻动上下左右4个硬币

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

//翻动上边的硬币，row-1行，col列

if(row-1>=0)

{

this->mCoinBtns[row-1][col]->flip();

}

//翻动下边的硬币，row+1行，col列

if(row+1<4)

{

this->mCoinBtns[row+1][col]->flip();

}

//翻动左边的硬币，row行，col-1列

if(col-1>=0)

{

this->mCoinBtns[row][col-1]->flip();

}

//翻动右边的硬币，row行，col+1列

if(col+1<4)

{

this->mCoinBtns[row][col+1]->flip();

}

});

}

## 判断是否胜利

胜利条件是盘面全是金币的时候，所以我们只要判断所有金币按钮的stat状态都是1就行，如果有一个是0，表示还没有胜利，游戏还得继续。

PlayScene添加judgeWin函数：

void **judgeWin**();

PlayScene.cpp文件中添加：

void PlayScene::**judgeWin**()

{

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

{

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

{

//如果有一个硬币是银币，游戏就继续，函数返回

if(mCoinBtns[row][col]->stat()==0)

return;

}

}

QMessageBox::information(this,"恭喜","你已经胜利了");

}

在flipCoin函数中最后翻动上下左右四个硬币后添加 judgeWin函数调用



最终效果



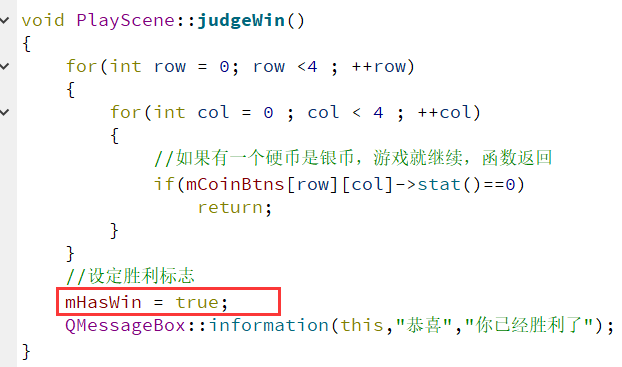
## 胜利后禁止金币再次反转

判断胜利后使用一个成员变量mHasWin记住当前场景已经胜利的状态，然后再flipCoin函数中判断该变量的值，如果已经胜利，那么就不翻动任何金币，直接终止函数

PlayScene.h中添加成员变量：

bool mHasWin;

PlayScene.cpp中修改：



flipCoin添加判断是否胜利条件的return语句

void PlayScene::**flipCoin**(int row, int col)

{

if(mHasWin)

return;

…

}

## 胜利图片显示

使用QPropertyAnimation将图片弹下来即可

//开始时，图片在窗口上方隐藏

QLabel\* winLabel = new QLabel(this);

QPixmap tmpPix;

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

winLabel->resize(tmpPix.width(),tmpPix.height());

winLabel->setPixmap(tmpPix);

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

winLabel->show();

//然后播放动画，图片下降

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

animation->setDuration(1000);

animation->setStartValue(winLabel->geometry());

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

animation->setEasingCurve(QEasingCurve::OutBounce);

animation->start(QPropertyAnimation::DeleteWhenStopped);

# 音效添加

音效添加使用QSound::play(“音效资源”);的方式播放音效

Incude<QSound>前别忘了在pro文件中添加 QT += multimedia 引入多媒体模块

## 开始音效

点击开始按钮，播放音效

QSound::play(":/res/TapButtonSound.wav");

## 选择关卡音效

在选择关卡场景中，点击关卡按钮的时候添加音效

QSound::play(":/res/TapButtonSound.wav");

## 返回按钮音效

在选择关卡场景与翻金币游戏场景中，分别添加返回按钮音效如下：

QSound::play(":/res/BackButtonSound.wav");

## 翻金币与胜利音效

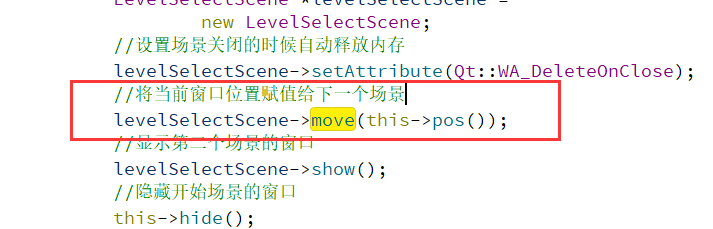
在PlayScene中添加，翻金币的音效以及 胜利的音效

QSound::play(":/res/ConFlipSound.wav");

QSound::play(":/res/LevelWinSound.wav");

# 优化项目

当我们移动开始场景到屏幕左边某个位置，点击开始按钮，选择场景窗口弹出的时候，就会出现在屏幕的正中央，顶层窗口默认出现位置是屏幕中央 。解决问题就是当我们show一个新的窗口对之前将该窗口移动到当前窗口的位置，比如开始场景切换到关卡选择场景时：



其他场景切换的时候也是同理。

至此，本案例全部制作完成。