1 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：

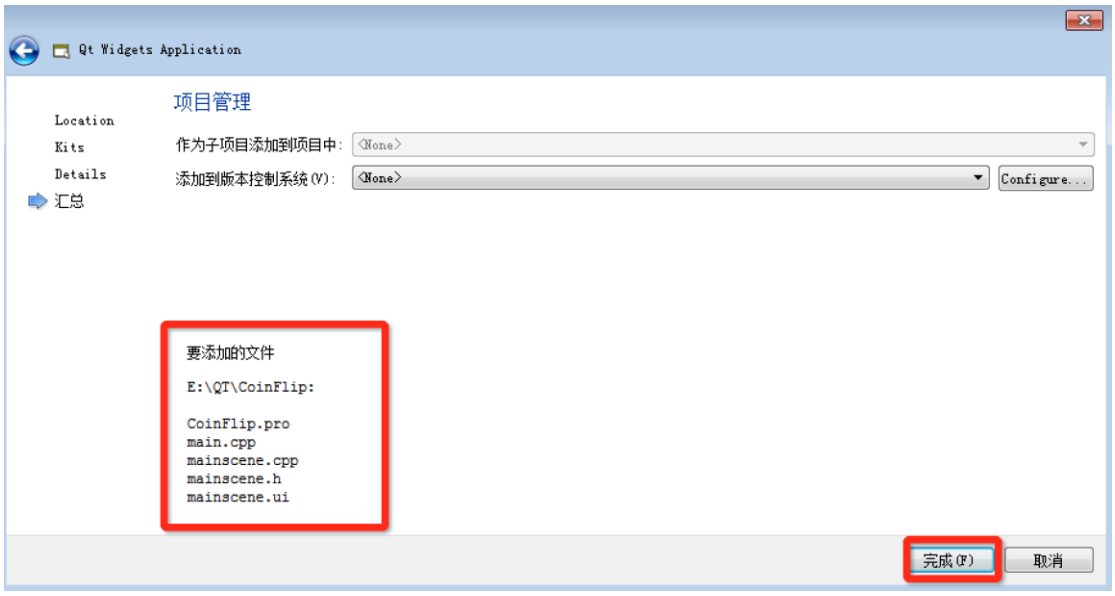
2 项目基本配置

2.1 创建项目

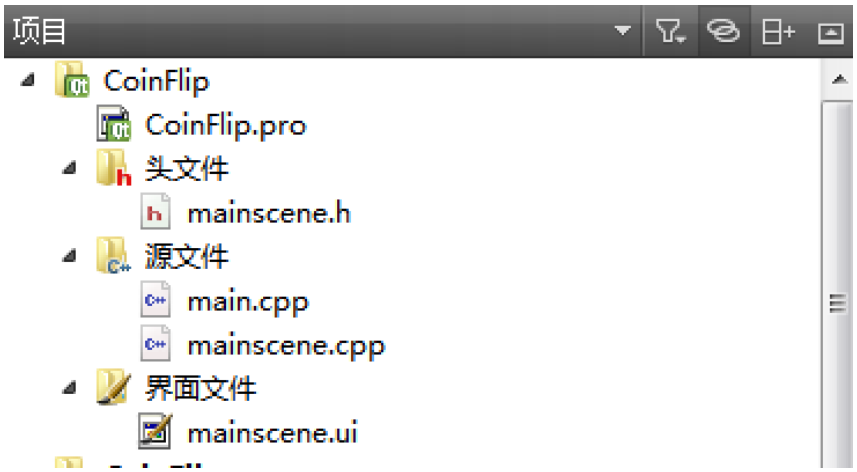
打开 Qt-Creator，创建项目：注意名称不要包含空格和回车，路径不要有中文

类信息中，选择基类为 QMainWindow，类名称为 MainScene，代表着主场景。

点击完成，创建出项目：

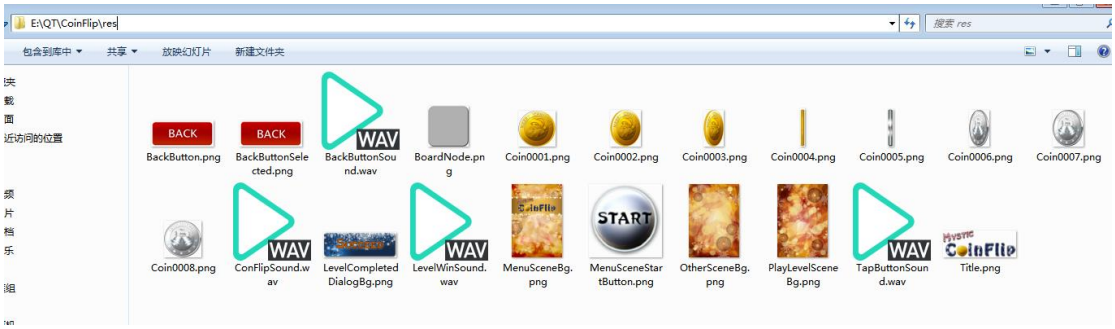


创建的项目结构如下：

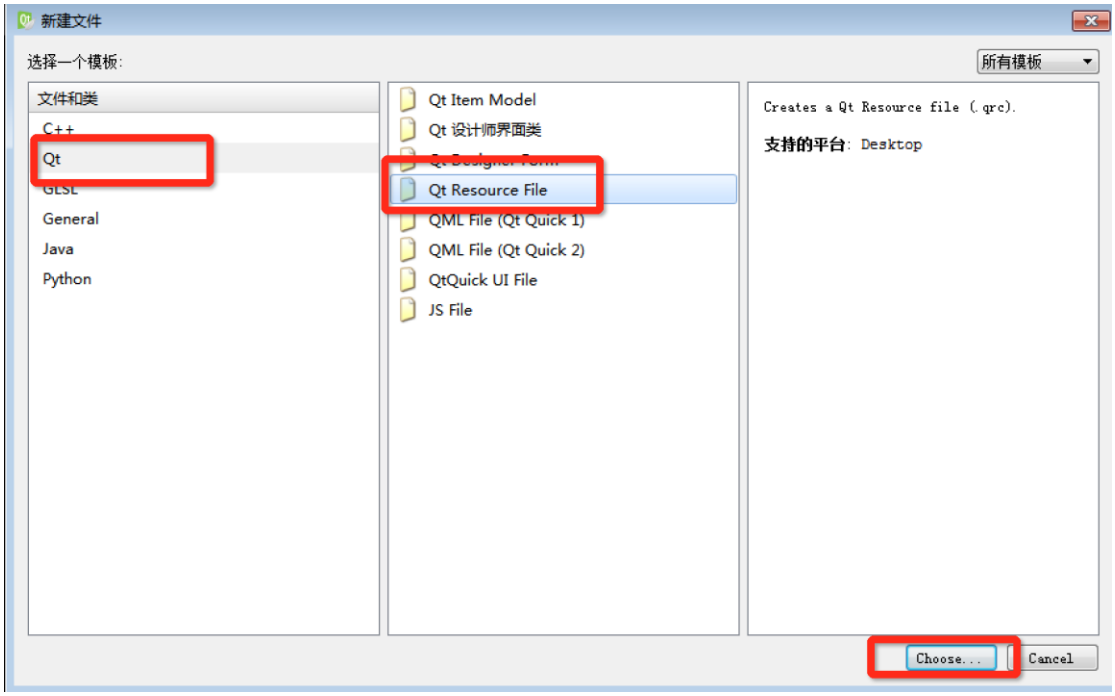


2.2 添加资源

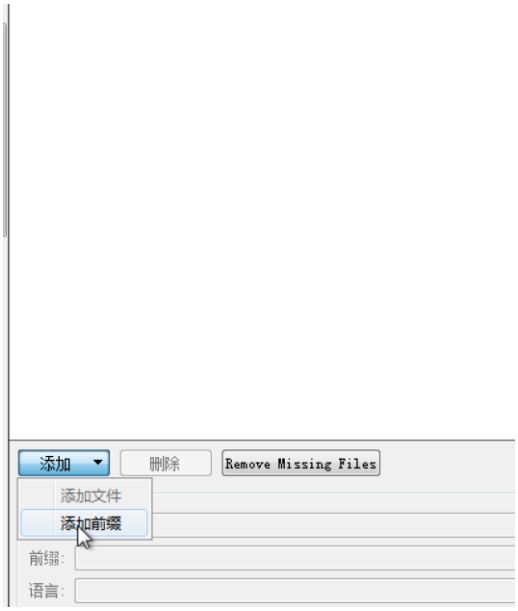
将资源添加到当前项目下

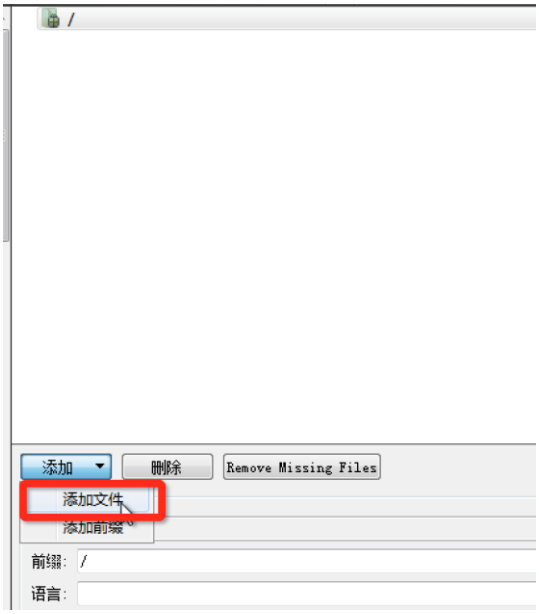


然后创建.qrc 文件

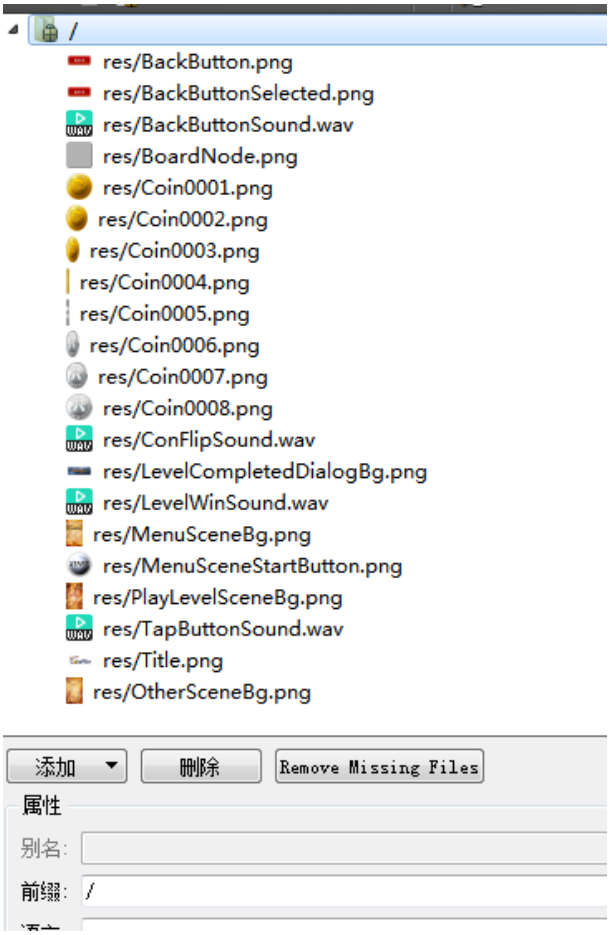


进入编辑模式，添加前缀“/”，添加文件





将所有资源文件进行添加

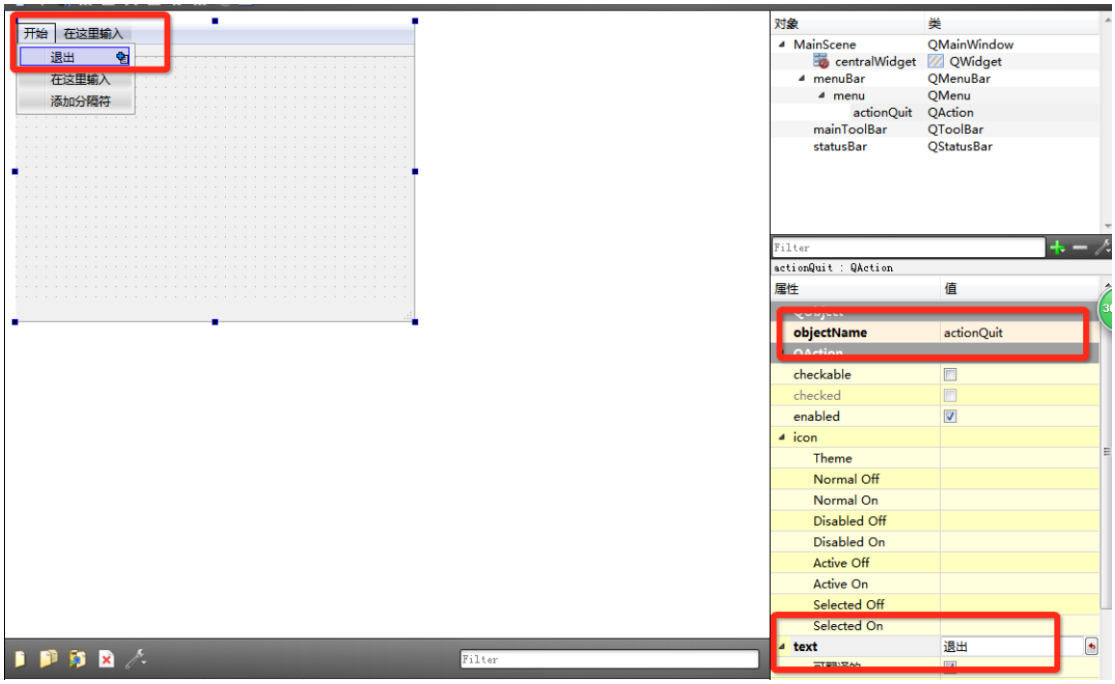


至此将所有需要的资源添加到了本项目中。

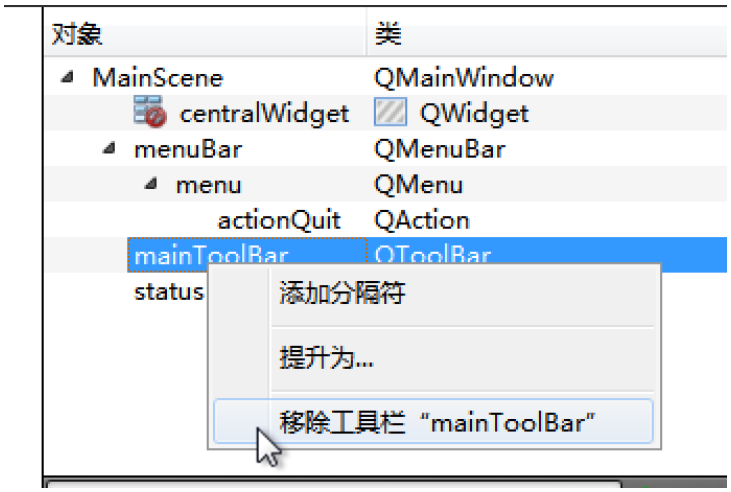
3 主场景

3.1 设置游戏主场景配置

点击 mainscene.ui 文件，设计其菜单栏如下：



设计“退出”菜单项，objectName 为 actionQuit， text 为 退出；
移除自带的工具栏与状态栏

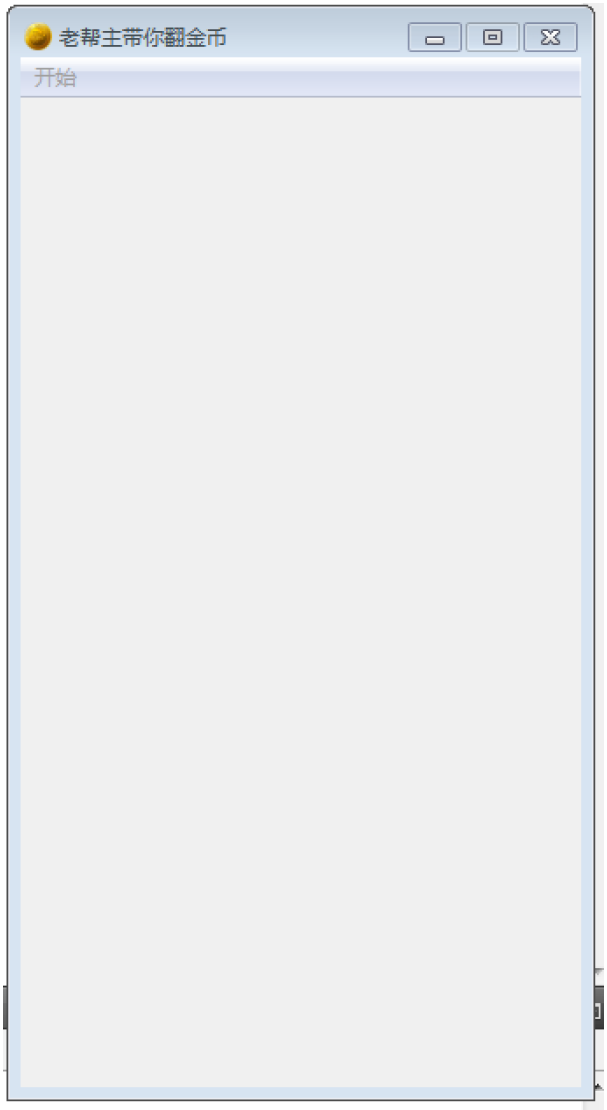


回到 MainScene.cpp 文件，进入构造函数中，进行场景的基本配置，代码如下：

```
    //设置固定大小
    this->setFixedSize(320,588);
    //设置应用图片
    this->setWindowIcon(QPixmap(":/res/Coin0001.png"));
```

```
    //设置窗口标题
    this->setWindowTitle("老帮主带你翻金币");
```

运行效果如图：



实现点击开始，退出游戏功能，代码如下：

```
    //点击退出，退出程序
    connect(ui->actionQuit,&QAction::triggered,[=](){this->close();});
```

3.2 设置背景图片

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

```
void MainScene::paintEvent(QPaintEvent *)
{
```

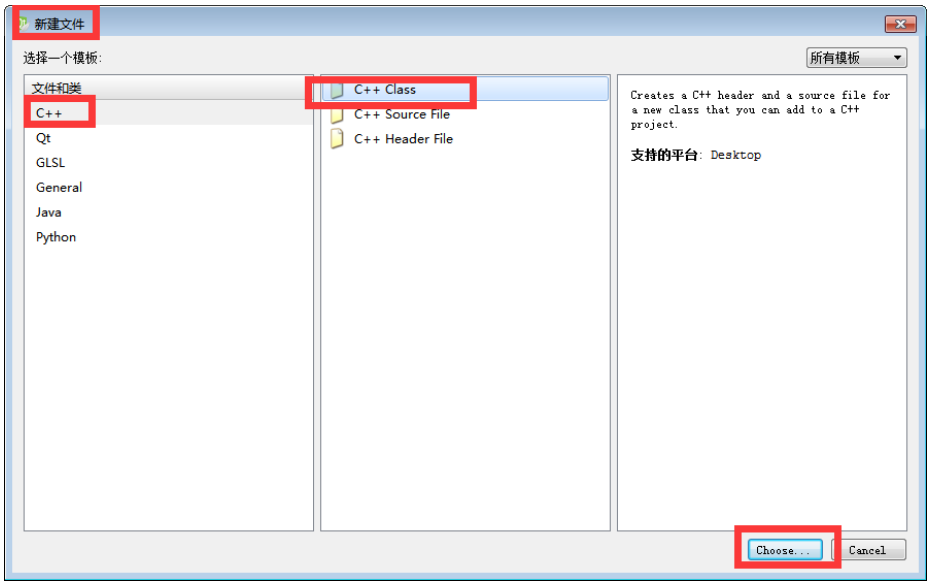
```
    //创建画家，指定绘图设备
    QPainter painter(this);
    //创建 QPixmap 对象
    QPixmap pix;
    //加载图片
    pix.load(":/res/PlayLevelSceneBg.png");
    //绘制背景图
    painter.drawPixmap(0,0,this->width(),this->height(),pix);


    //加载标题
    pix.load(":/res/Title.png");
    //缩放图片
    pix = pix.scaled(pix.width()*0.5,pix.height()*0.5);
    //绘制标题
    painter.drawPixmap( 10,30,pix.width(),pix.height(),pix);
}
```

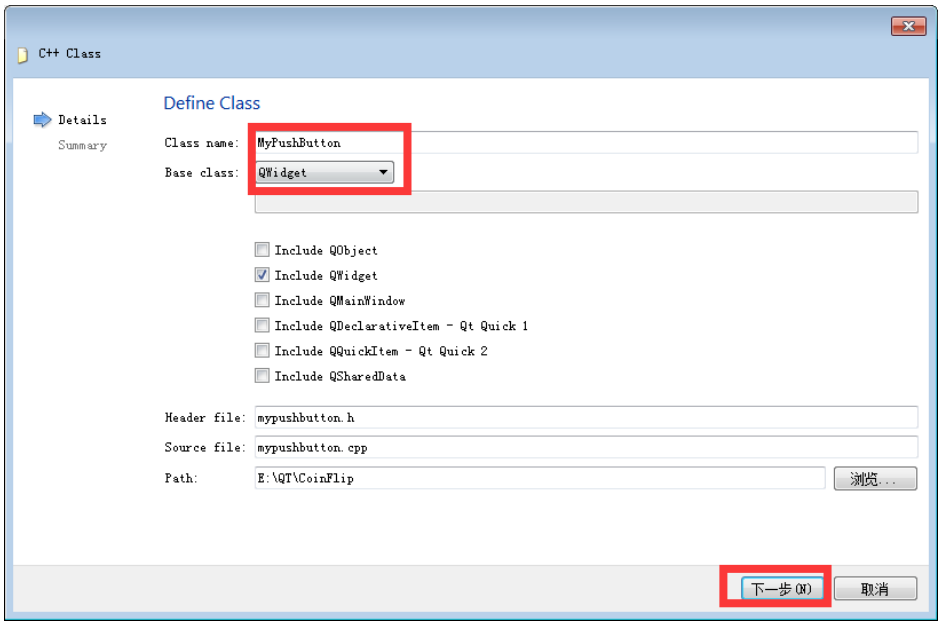
运行效果如图：

3.3 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

```
#ifndef MYPUSHBUTTON_H
#define MYPUSHBUTTON_H

#include <QPushButton>

class MyPushButton : public QPushButton
{
    Q_OBJECT
public:
    explicit MyPushButton(QWidget *parent = 0);

signals:

public slots:
};

#endif // MYPUSHBUTTON_H
```

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 normalImg,QString pressImg = "");

    QString normalImgPath;  //默认显示图片路径
    QString pressedImgPath; //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 normalImg,QString pressImg)
{
    //成员变量 normalImgPath 保存正常显示图片路径
    normalImgPath = normalImg;
    //成员变量 pressedImgPath 保存按下后显示的图片
    pressedImgPath = pressImg;
    //创建 QPixmap 对象
    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(normalImgPath);
    if(!ret)
    {
        qDebug() << normalImg << "加载图片失败!";
    }
    //设置图片的固定尺寸
```

```
    this->setFixedSize( pixmap.width(), pixmap.height() );
    //设置不规则图片的样式表
    this->setStyleSheet("QPushButton{border:0px;}");
    //设置图标
    this->setIcon(pixmap);
    //设置图标大小
    this->setIconSize(QSize(pixmap.width(),pixmap.height()));
}
```

回到 MainScene 的构造函数中，创建开始按钮

```
//创建开始按钮
    MyPushButton * startBtn = new MyPushButton(":/res/MenuSceneStartButton.png");
    startBtn->setParent(this);

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

运行效果如图：

不规则的开始按钮添加完成。

3.4 开始按钮跳跃特效实现

连接信号槽，监听开始按钮点击

```
    //监听点击事件，执行特效
    connect(startBtn,&MyPushButton::clicked,[=](){
        startBtn->zoom1(); //向下跳跃
        startBtn->zoom2(); //向上跳跃

    });
```

zoom1 与 zoom2 为 MyPushButton 中扩展的特效代码，具体如下：

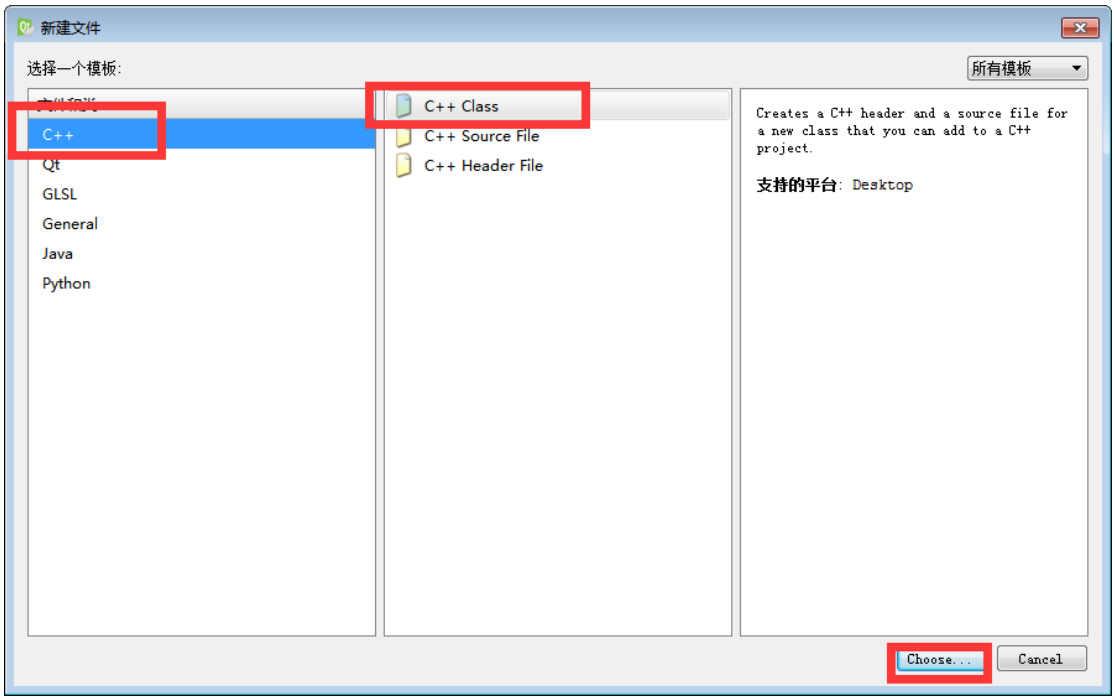
```
void MyPushButton::zoom1()
{
    //创建动画对象
    QPropertyAnimation * animation1 = new QPropertyAnimation(this,"geometry");
    //设置时间间隔，单位毫秒
    animation1->setDuration(200);
    //创建起始位置

animation1->setStartValue(QRect(this->x(),this->y(),this->width(),this->height(
)));
    //创建结束位置

animation1->setEndValue(QRect(this->x(),this->y()+10,this->width(),this->height
()));
    //设置缓和曲线，QEasingCurve::OutBounce 为弹跳效果
    animation1->setEasingCurve(QEasingCurve::OutBounce);
    //开始执行动画
    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->heig
ht()));
```

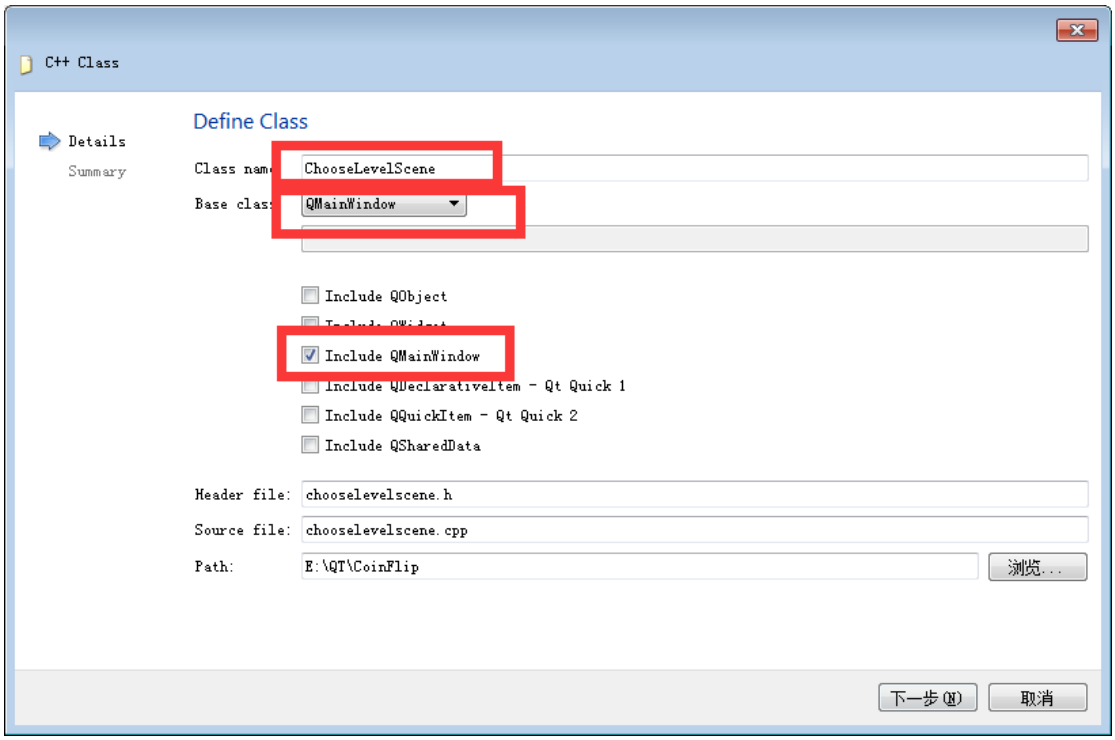
```
animation1->setEndValue(QRect(this->x(),this->y(),this->width(),this->height())
);
    animation1->setEasingCurve(QEasingCurve::OutBounce);
    animation1->start();
}
```

运行代码，点击按钮，测试弹跳效果。

3.5 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 ChooseLevelScene 选择基类为 QMainWindow，点击下一步，然后点击完成。

3.6 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 MainScene.h 中 保存 ChooseScene 选择关卡场景对象

```
//选择关卡场景
ChooseLevelScene *chooseScene = new ChooseLevelScene;
```

我们在 zoom1 和 zoom2 特效后，延时 0.5 秒，进入选择关卡场景，代码如下：

```
//延时 0.5 秒后 进入选择场景
        QTimer::singleShot(500, this,[=](){
            this->hide();
            chooseScene->show();
        });
```

```
//监听点击事件，执行特效
connect(startBtn,&MyPushButton::clicked,[=](){
    startBtn->zoom1(); //向下跳跃
    startBtn->zoom2(); //向上跳跃

    //延时0.5秒后 进入选择场景
    QTimer::singleShot(500, this,[=](){
        this->hide();
        chooseScene.show();
    });

});
```

测试点击开始，执行特效后延时 0.5 秒进入选择关卡场景

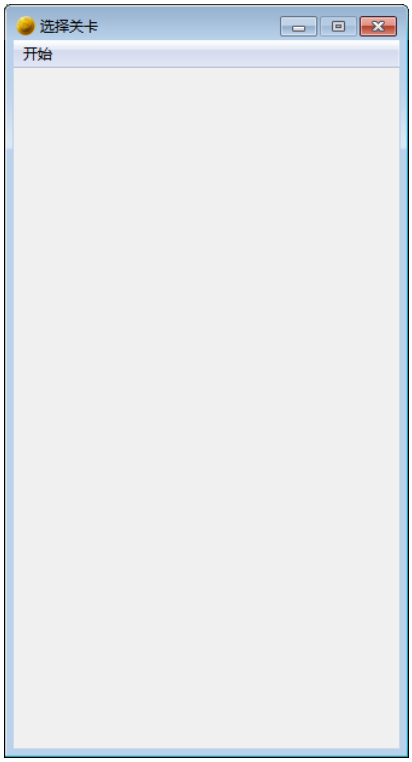
4 选择关卡场景

4.1 场景基本设置

选择关卡构造函数如下：

```
    //设置窗口固定大小
    this->setFixedSize(320,588);
    //设置图标
    this->setWindowIcon(QPixmap(":/res/Coin0001.png"));
    //设置标题
    this->setWindowTitle("选择关卡");

    //创建菜单栏
    QMenuBar * bar = this->menuBar();
    this->setMenuBar(bar);
    //创建开始菜单
    QMenu * startMenu = bar->addMenu("开始");
    //创建按钮菜单项
    QAction * quitAction = startMenu->addAction("退出");
    //点击退出 退出游戏
    connect(quitAction,&QAction::triggered,[=](){this->close();});
```

运行效果如图：



4.2 背景设置

```
void ChooseLevelScene::paintEvent(QPaintEvent *)
{
    QPainter painter(this);
    QPixmap pix;
    pix.load(":/res/OtherSceneBg.png");
    painter.drawPixmap(0,0,this->width(),this->height(),pix);

     //加载标题
    pix.load(":/res/Title.png");
    painter.drawPixmap( (this->width() -
    pix.width())*0.5,30,pix.width(),pix.height(),pix);
}
```

4.3 创建返回按钮

```
    //返回按钮
    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

```
//鼠标事件
void MyPushButton::mousePressEvent(QMouseEvent *e)
{
    if(pressedImgPath != "") //选中路径不为空，显示选中图片
    {
        QPixmap pixmap;
        bool ret = pixmap.load(pressedImgPath);
        if(!ret)
        {
            qDebug() << pressedImgPath << "加载图片失败!";
        }

        this->setFixedSize( pixmap.width(), pixmap.height() );
        this->setStyleSheet("QPushButton{border:0px;}");
        this->setIcon(pixmap);
        this->setIconSize(QSize(pixmap.width(),pixmap.height()));
    }
    //交给父类执行按下事件
    return QPushButton::mousePressEvent(e);
}
void MyPushButton::mouseReleaseEvent(QMouseEvent *e)
{
    if(normalImgPath != "") //选中路径不为空，显示选中图片
    {
        QPixmap pixmap;
        bool ret = pixmap.load(normalImgPath);
        if(!ret)
        {
            qDebug() << normalImgPath << "加载图片失败!";
        }
        this->setFixedSize( pixmap.width(), pixmap.height() );
        this->setStyleSheet("QPushButton{border:0px;}");
        this->setIcon(pixmap);
        this->setIconSize(QSize(pixmap.width(),pixmap.height()));
    }
    //交给父类执行 释放事件
```

```
    return QPushButton::mouseReleaseEvent(e);
}
```

4.3 返回按钮

在这里我们点击返回后，延时 0.5 后隐藏自身，并且发送自定义信号，告诉外界自身已经选择了返回按钮。

```
//返回按钮功能实现
    connect(closeBtn,&MyPushButton::clicked,[=](){
        QTimer::singleShot(500, this,[=](){
            this->hide();
            //触发自定义信号，关闭自身，该信号写到 signals 下做声明
            emit this->chooseSceneBack();
             }
        );
    });
```

在主场景 MainScene 中 点击开始按钮显示选择关卡的同时，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(chooseScene,&ChooseLevelScene::chooseSceneBack,[=](){
                chooseScene ->hide();
                 this->show();
            });
```

测试主场景与选择关卡场景的切换功能。

4.4 创建选择关卡按钮

```
//创建关卡按钮
    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);

        //按钮上显示的文字
        QLabel * label = new QLabel;
        label->setParent(this);
        label->setFixedSize(menuBtn->width(),menuBtn->height());
```

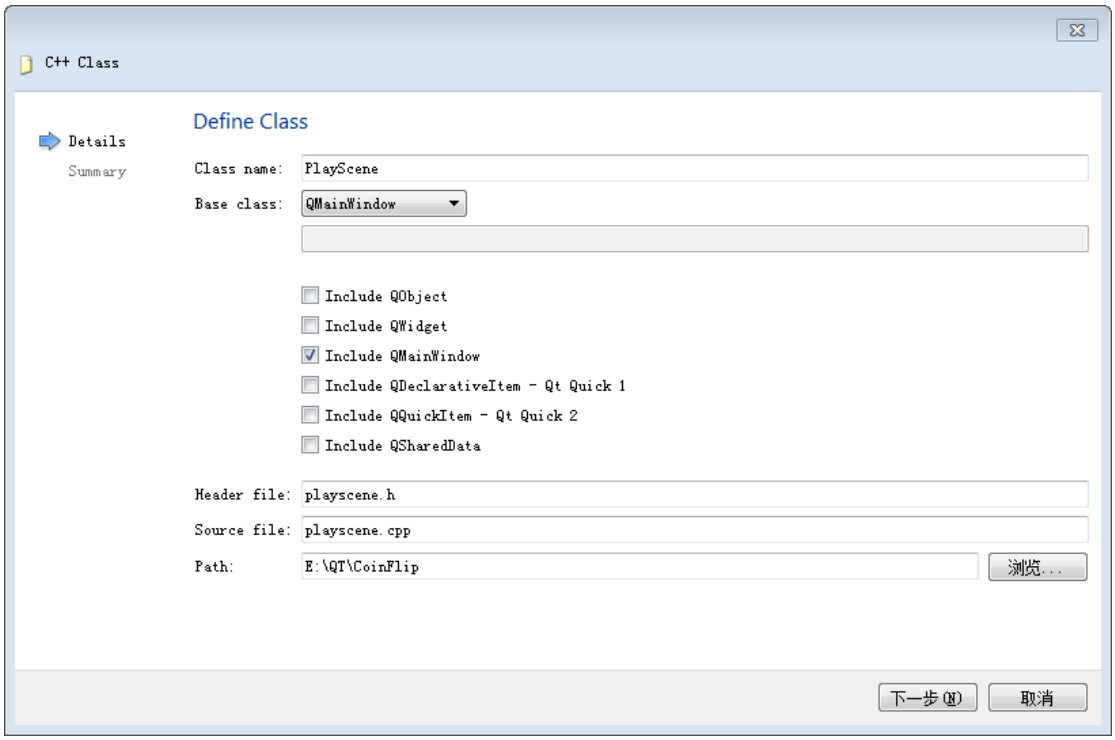
```
        label->setText(QString::number(i+1));
        label->setAlignment(Qt::AlignHCenter | Qt::AlignVCenter); //设置居中
        label->move(25 + (i%4)*70 , 130+ (i/4)*70);
        label->setAttribute(Qt::WA_TransparentForMouseEvents,true);  //鼠标事件穿透
    }
```

运行效果如果：

4.5 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 ChooseLevelScene.h 中声明
PlayScene *pScene = NULL;

```
//监听选择关卡按钮的信号槽
        connect(menuBtn,&MyPushButton::clicked,[=](){
           // qDebug() << "select: " << i;
            if(pScene == NULL)  //游戏场景最好不用复用，直接移除掉创建新的场景
            {
                this->hide();
                pScene = new PlayScene(i+1); //将选择的关卡号 传入给PlayerScene
                pScene->show();
            }
        });
```

这里 pScene = new PlayScene(i+1); 将用户所选的关卡号发送给 pScene，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

5 翻金币场景

5.1 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

```
//成员变量 记录关卡索引
    int levalIndex;
```

PlayScene.cpp 中 初始化该场景配置

```
PlayScene::PlayScene(int index)
{
    //qDebug() << "当前关卡为"<< index;
    this->levalIndex = index;
    //设置窗口固定大小
    this->setFixedSize(320,588);
    //设置图标
    this->setWindowIcon(QPixmap(":/res/Coin0001.png"));
    //设置标题
    this->setWindowTitle("翻金币");

    //创建菜单栏
    QMenuBar * bar = this->menuBar();
    this->setMenuBar(bar);
    //创建开始菜单
    QMenu * startMenu = bar->addMenu("开始");
    //创建按钮菜单项
    QAction * quitAction = startMenu->addAction("退出");
    //点击退出 退出游戏
    connect(quitAction,&QAction::triggered,[=](){this->close();});
}
```

5.2 背景设置

```
void PlayScene::paintEvent(QPaintEvent *)
{
    //加载背景
```

```
    QPainter painter(this);
    QPixmap pix;
    pix.load(":/res/PlayLevelSceneBg.png");
    painter.drawPixmap(0,0,this->width(),this->height(),pix);

    //加载标题
    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 返回按钮

```
    //返回按钮
    MyPushButton * closeBtn = new
MyPushButton(":/res/BackButton.png",":/res/BackButtonSelected.png");
    closeBtn->setParent(this);

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

    //返回按钮功能实现
    connect(closeBtn,&MyPushButton::clicked,[=](){
        QTimer::singleShot(500, this,[=](){
            this->hide();
            //触发自定义信号，关闭自身，该信号写到 signals 下做声明
            emit this->chooseSceneBack();
             }
        );
    });
```

5.4 在 ChooseScene 选择关卡场景中，监听 PlayScene 的返回信号

```
connect(pScene,&PlayScene::chooseSceneBack,[=](){
                    this->show();
                    delete pScene;
                    pScene = NULL;
                });
```

```
//监听选择关卡按钮的信号槽
connect(menuBtn,&MyPushButton::clicked,[=](){
    // qDebug() << "select: " << i;
     if(pScene == NULL)  //游戏场景最好不用复用，直接移除掉创建新的场景
     {
         this->hide();
         pScene = new PlayScene(i+1); //将选择的关卡号 传入给PlayerScene
         pScene->show();

         //PlayScene的返回按钮监听，删除该scene并且将指针指向空
         connect(pScene,&playScene::chooseSceneBack,[=](){
             this->show();
             delete pScene;
             pScene = NULL;
         });
     }
});
```

5.4 显示当前关卡

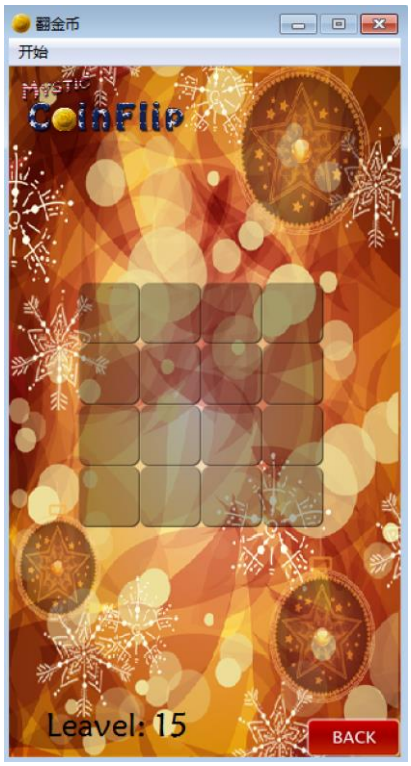
```
//当前关卡标题
    QLabel * label = new QLabel;
    label->setParent(this);
    QFont font;
    font.setFamily("华文新魏");
    font.setPointSize(20);
    label->setFont(font);
    QString str = QString("Leavel: %1").arg(this->levalIndex);
    label->setText(str);
    label->setGeometry(QRect(30, this->height() - 50,120, 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：

5.5 创建金币背景图片

```
//创建金币的背景图片
  for(int i = 0 ; i < 4;i++)
  {
      for(int j = 0 ; j < 4; j++)
      {
         //绘制背景图片
          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);
      }
  }
```

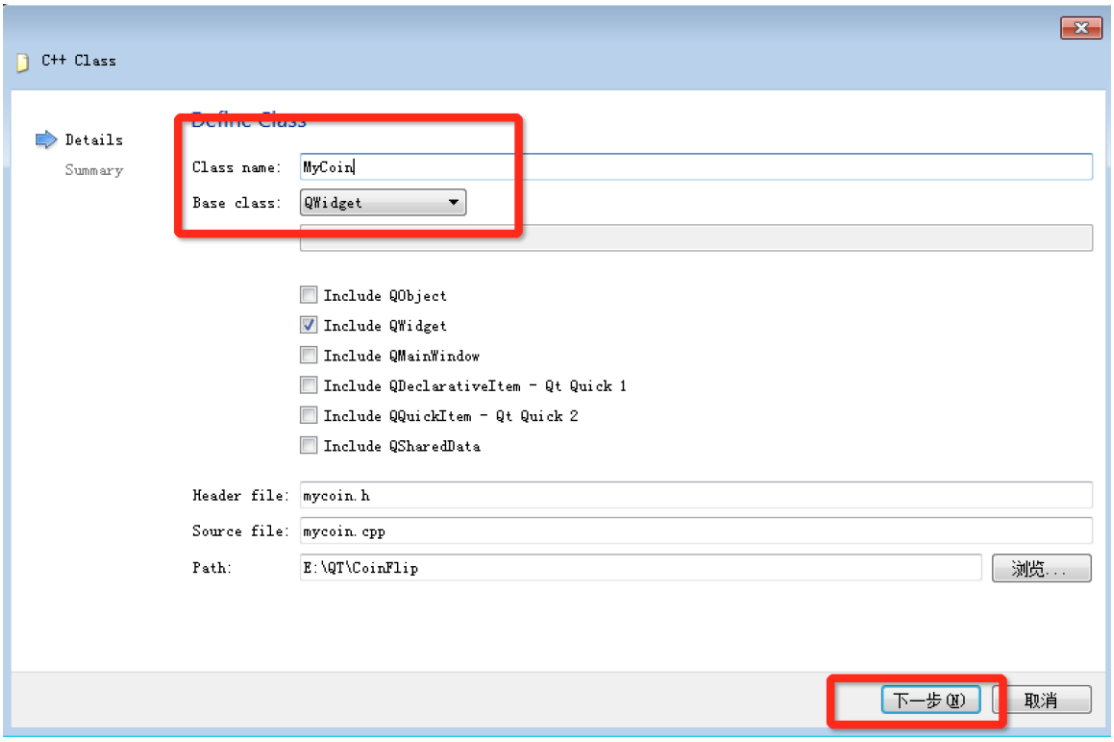
运行效果如图：



5.6 创建金币类

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，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，完成金币所需的所有功能。

5.6.1 创建金币类 MyCoin

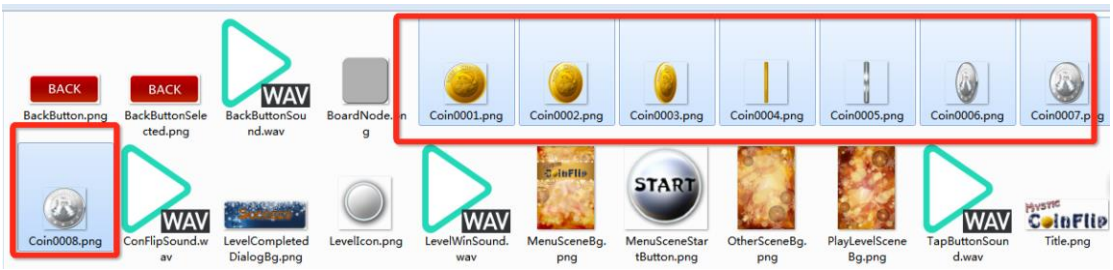


并修改 MyCoin 的基类为 QPushButton

5.6.2 构造函数

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。

因此我们在创建一个金币对象时候，应该提供一个参数，代表着传入的是金币资源路径还是银币资源路径，根据路径我们创建不同样式的图案。



在 MyCoin.h 中声明：

```
MyCoin(QString butImg); //代表图片路径
```

在 MyCoin.cpp 中进行实现

```
MyCoin::MyCoin(QString butImg)
{

    QPixmap pixmap;
    bool ret = pixmap.load(butImg);
    if(!ret)
    {
        qDebug() << butImg << "加载图片失败!";
    }

    this->setFixedSize( pixmap.width(), pixmap.height() );
    this->setStyleSheet("QPushButton{border:0px;}");
    this->setIcon(pixmap);
    this->setIconSize(QSize(pixmap.width(),pixmap.height()));

}
```

5.6.3 测试

在翻金币场景 PlayScene 中，我们测试下封装的金币类是否可用，可以在创建好的金币背景代码后，添加如下代码：

```
            //金币对象
            MyCoin * coin = new MyCoin(":/res/Coin0001.png");
            coin->setParent(this);
            coin->move(59 + i*50,204+j*50);
```

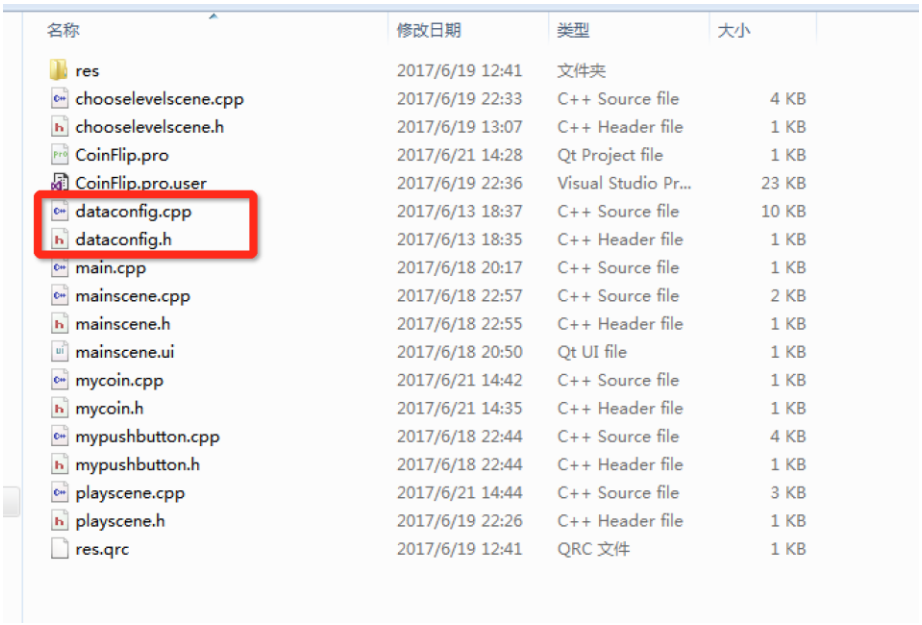
运行效果如图

5.7 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。

5.7.1 添加现有文件 dataConfig

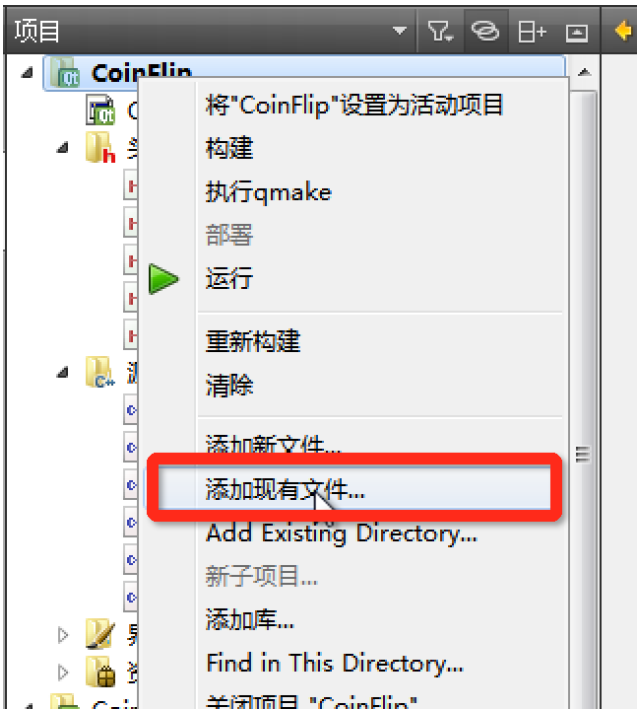
首先先将 dataConfig.h 和 dataConfig.cpp 文件放入到当前项目下：



名称	修改日期	类型	大小
res	2017/6/19 12:41	文件夹	
chooselevelscene.cpp	2017/6/19 22:33	C++ Source file	4 KB
chooselevelscene.h	2017/6/19 13:07	C++ Header file	1 KB
CoinFlip.pro	2017/6/21 14:28	Qt Project file	1 KB
CoinFlip.pro.user	2017/6/19 22:36	Visual Studio Pr...	23 KB
dataconfig.cpp	2017/6/13 18:37	C++ Source file	10 KB
dataconfig.h	2017/6/13 18:35	C++ Header file	1 KB
main.cpp	2017/6/18 20:17	C++ Source file	1 KB
mainscene.cpp	2017/6/18 22:57	C++ Source file	2 KB
mainscene.h	2017/6/18 22:55	C++ Header file	1 KB
mainscene.ui	2017/6/18 20:50	Qt UI file	1 KB
mycoin.cpp	2017/6/21 14:42	C++ Source file	1 KB
mycoin.h	2017/6/21 14:35	C++ Header file	1 KB
mypushbutton.cpp	2017/6/18 22:44	C++ Source file	4 KB
mypushbutton.h	2017/6/18 22:44	C++ Header file	1 KB
playscene.cpp	2017/6/21 14:44	C++ Source file	3 KB
playscene.h	2017/6/19 22:26	C++ Header file	1 KB
res.qrc	2017/6/19 12:41	QRC 文件	1 KB

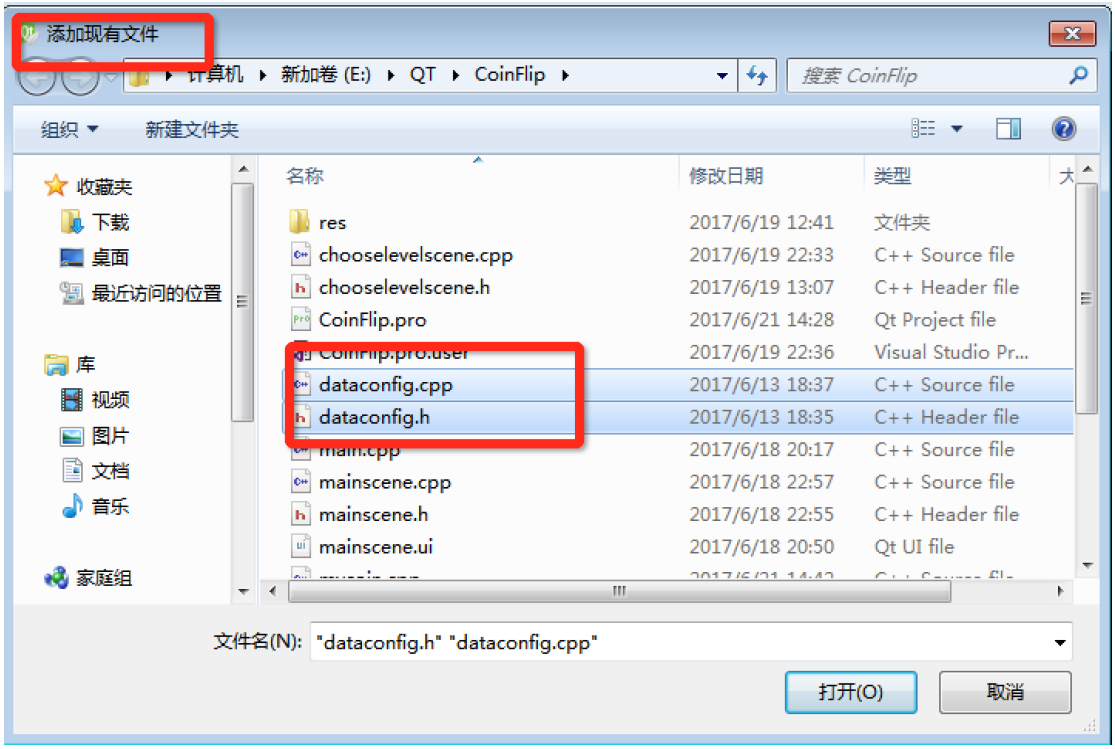
5.7.2 添加现有文件

其次在 Qt_Creator 项目右键，点击添加现有文件



5.7.3 完成添加

选择当前项目下的文件，并进行添加



5.7.4 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

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

在上图中，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。

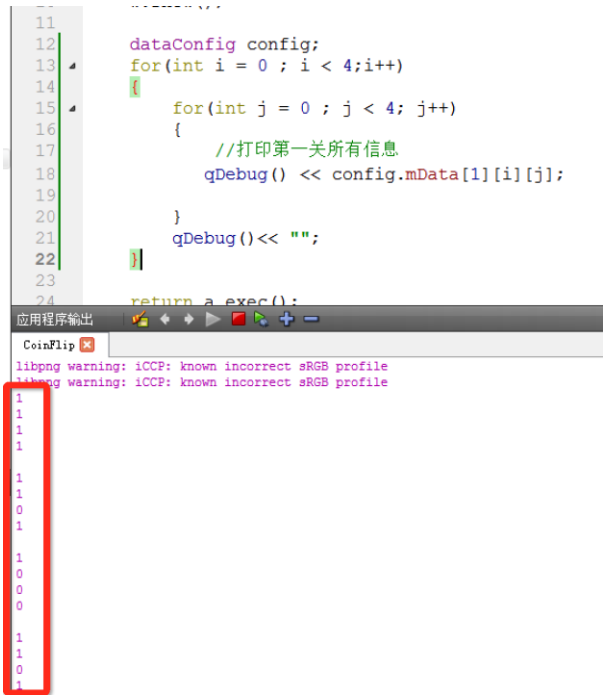
5.7.5 测试关卡数据

在 Main 函数可以测试第一关的数据，添加如下代码：

```
dataConfig config;
    for(int i = 0 ; i < 4;i++)
    {
        for(int j = 0 ; j < 4; j++)
        {
```

```
            //打印第一关所有信息
            qDebug() << config.mData[1][i][j];

        }
        qDebug()<< "";
    }
```

输出结果如下图：



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;
```

5.8 初始化各个关卡

首先，可以在 playScene 中声明一个成员变量，用户记录当前关卡的二维数组

```
int gameArray[4][4]; //二维数组数据
```

之后，在.cpp 文件中，初始化这个二维数组

```
//初始化二维数组
    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];
        }
    }
```

初始化成功后，在金币类 也就是 MyCoin 类中，扩展属性 posX，posY，以及 flag
这三个属性分别代表了，该金币在二维数组中 x 的坐标，y 的坐标，以及当前的正反标志。

```
    int posX; //x 坐标
    int posY; //y 坐标
    bool flag; //正反标志
```

然后完成金币初始化，代码如下：

```
        //金币对象
             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; //记录 x 坐标
            coin->posY = j; //记录 y 坐标
            coin->flag =gameArray[i][j]; //记录正反标志
```

运行测试各个关卡初始化，例如第一关效果如图：

5.9 翻金币特效

5.9.1 MyCoin 类扩展属性和行为

关卡的初始化完成后，下面就应该点击金币，进行翻转的效果了，那么首先我们先在 MyCoin 类中创建出该方法。

在 MyCoin.h 中声明：

```
void changeFlag();//改变标志,执行翻转效果
    QTimer *timer1; //正面翻反面 定时器
    QTimer *timer2; //反面翻正面 定时器
    int min = 1; //最小图片
    int max = 8; //最大图片
```

MyCoin.cpp 中做实现

```
void MyCoin::changeFlag()
{
    if(this->flag) //如果是正面，执行下列代码
```

```
    {
        timer1->start(30);
        this->flag = false;
    }
    else //反面执行下列代码
    {
        timer2->start(30);
        this->flag = true;
    }
}
```

当然在构造函数中，记得创建出两个定时器

```
//初始化定时器
    timer1 = new QTimer(this);
    timer2 = new QTimer(this);
```

5.9.2 创建特效

当我们分别启动两个定时器时，需要在构造函数中做监听操作，并且做出响应，翻转金币，然后再结束定时器。

构造函数中 进行下列监听代码：

```
    //监听正面翻转的信号槽
    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) //如果大于最大值，重置最小值，并停止定时器
        {
            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) //如果小于最小值，重置最大值，并停止
定时器
        {
            this->max = 8;
            timer2->stop();
        }
    });
```

5.9.3 测试

监听每个按钮的点击效果，并翻转金币

```
connect(coin,&MyCoin::clicked,[=](){
                //qDebug() << "点击的位置：  x = " <<  coin->posX << " y
= " << coin->posY ;
                coin->changeFlag();
                gameArray[i][j] = gameArray[i][j] == 0 ? 1 : 0; //数组
内部记录的标志同步修改
            });
```

```
//绘制背景图片
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);

//金币对象
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; //记录x坐标
coin->posY = j; //记录y坐标
coin->flag =gameArray[i][j]; //记录正反标志

connect(coin,&MyCoin::clicked,[=](){
    //qDebug() << "点击的位置：  x = " <<  coin->posX << " y = " << coin->posY ;
    coin->changeFlag();
    gameArray[i][j] = gameArray[i][j] == 0 ? 1 : 0; //数组内部记录的标志同步修改
});
```

5.9.3 禁用按钮

此时，确实已经可以执行翻转金币代码了，但是如果快速点击，会在金币还没有执行一个完整动作之后 ，又继续开始新的动画，我们应该在金币做动画期间，禁止再次点击，并在完成动画后，开启点击。

在 MyCoin 类中加入一个标志 isAnimation 代表是否正在做翻转动画，默认 isAnimation 值为 false。

```
bool isAnimation    = false; //做翻转动画的标志
```

在 MyCoin 做动画期间加入

```
this->isAnimation    = true;
```

也就是 changeFlag 函数中将标志设为 true

加入位置如下：

```
void MyCoin::changeFlag()
{
    if(this->flag) //如果是正面，执行下列代码
    {
        this->isAnimation    = true;
    }
    else //反面执行下列代码
    {
        this->isAnimation    = true;
    }
}
```

并且在做完动画时，将标志改为 false

```
//监听正面翻转的信号槽
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) //如果大于最大值，重置最小值，并停止定时器
    {
        this->min = 1;
        this->isAnimation  = false;
        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) //如果小于最小值，重置最大值，并停止定时器
    {
        this->max = 8;
        this->isAnimation  = false;
        timer2->stop();
    }
});
```

int MyCoin::max

重写按钮的按下事件，判断如果正在执行动画，那么直接 return 掉，不要执行后续代码。

代码如下：

```
void MyCoin::mousePressEvent(QMouseEvent *e)
{
    if(this->isAnimation )
    {
        return;
    }
    else
    {
        return QPushButton::mousePressEvent(e);
    }
}
```

5.10 翻周围金币

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，在.h 中声明

```
MyCoin * coinBtn[4][4]; //金币按钮数组
```

并且记录每个按钮的位置

```
coinBtn[i][j] = coin;
```

```
        {
            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; //记录x坐标
        coin->posY = j; //记录y坐标
        coin->flag =gameArray[i][j]; //记录正反标志

        coinBtn[i][j] = coin;
        connect(coin,&MyCoin::clicked,[=](){
            //qDebug() << "点击的位置： x = " <<  coin->posX << " y = " << coin->posY ;
            coin->changeFlag();
            gameArray[i][j] = gameArray[i][j] == 0 ? 1 : 0; //数组内部记录的标志同步修改



        });
    }
```

延时翻动其他周围金币

```
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 判断是否胜利

在 MyCoin.h 中加入 isWin 标志，代表是否胜利。

```
    bool isWin = true; //是否胜利
```

默认设置为 true，只要有一个反面的金币，就将该值改为 false，视为未成功。
代码写到延时翻金币后 进行判断

```
//判断是否胜利
            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，代表胜利了！

```
    if(this->isWin)
     {
            qDebug() << "胜利";
```

```
        }
```

5.12 胜利图片显示

将胜利的图片提前创建好，如果胜利触发了，将图片弹下来即可

```
  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());
```

如果胜利了，将上面的图片移动下来

```
  if(this->isWin)
    {
       qDebug() << "胜利";
        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,winLabe
l->width(),winLabel->height()));

animation1->setEasingCurve(QEasingCurve::OutBounce);
                        animation1->start();
                    }
```

5.13 胜利后禁用按钮

当胜利后，应该禁用所有按钮的点击状态，可以在每个按钮中加入标志位 isWin，如果 isWin 为 true， MousePressEvent 直接 return 掉即可

MyCoin 中.h 里添加：

```
bool isWin = false;//胜利标志
```

在鼠标按下事件中修改为

```
void MyCoin::mousePressEvent(QMouseEvent *e)
{
    if(this->isAnimation|| isWin == true )
    {
        return;
    }
    else
    {
        return QPushButton::mousePressEvent(e);
    }
}
```

```
//禁用所有按钮点击事件
    for(int i = 0 ; i < 4;i++)
      {
        for(int j = 0 ; j < 4; j++)
        {
          coinBtn[i][j]->isWin = true;
        }
}
```

测试，胜利后不可以点击任何的金币。

6 音效添加

6.1 开始音效

```

```

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

点击开始按钮，播放音效

```
startSound->play(); //开始音效
```

6.2 选择关卡音效

在选择关卡场景中，添加音效

```
//选择关卡按钮音效
    QSound *chooseSound = new QSound(":/res/TapButtonSound.wav",this);
```

选中关卡后，播放音效

```
chooseSound->play();
```

6.3 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：

```
//返回按钮音效
    QSound *backSound = new QSound(":/res/BackButtonSound.wav",this);
```

分别在点击返回按钮后，播放该音效

```
backSound->play();
```

6.4 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

```
  //翻金币音效
    QSound *flipSound = new QSound(":/res/ConFlipSound.wav",this);
    //胜利按钮音效
    QSound *winSound = new QSound(":/res/LevelWinSound.wav",this);
```

在翻金币时播放 翻金币音效

```
flipSound->play();
```

胜利时，播放胜利音效

```
winSound->play();
```

测试音效，使音效正常播放。

7 优化项目

当我们移动场景后，如果进入下一个场景，发现场景还在中心位置，如果想设置场景的位置，需要添加如下下图中的代码：

MainScene 中添加：

```
//监听点击事件，执行特效
connect(startBtn,&MyPushButton::clicked,[=](){

    startSound->play(); //开始音效
    startBtn->zoom1(); //向下跳跃
    startBtn->zoom2(); //向上跳跃

    //延时0.5秒后 进入选择场景
    QTimer::singleShot(500, this,[=](){
        this->hide();
        chooseScene->setGeometry(this->geometry());
        chooseScene->show();


        //监听选择场景的返回按钮
        connect(chooseScene,&ChooseLevelScene::chooseSceneBack,[=](){
            this->setGeometry(chooseScene->geometry());
            this->show();
        });

    });
});
```

ChooseScene 中添加：

```
//监听选择关卡按钮的信号槽
connect(menuBtn,&MyPushButton::clicked,[=](){
    // qDebug() << "select: " << i;

     chooseSound->play();
     if(pScene == NULL)  //游戏场景最好不用复用，直接移除掉创建新的场景
     {
         this->hide();
         pScene = new PlayScene(i+1); //将选择的关卡号 传入给PlayerScene
         pScene->setGeometry(this->geometry());
         pScene->show();

         //PlayScene的返回按钮监听，删除该scene并且将指针指向空
         connect(pScene,&PlayScene::chooseSceneBack,[=](){
             this->setGeometry(pScene->geometry());
             this->show();
             delete pScene;
             pScene = NULL;
         });
     }
```

测试切换三个场景的进入与返回都在同一个位置下，优化成功。

至此，本案例全部制作完成。