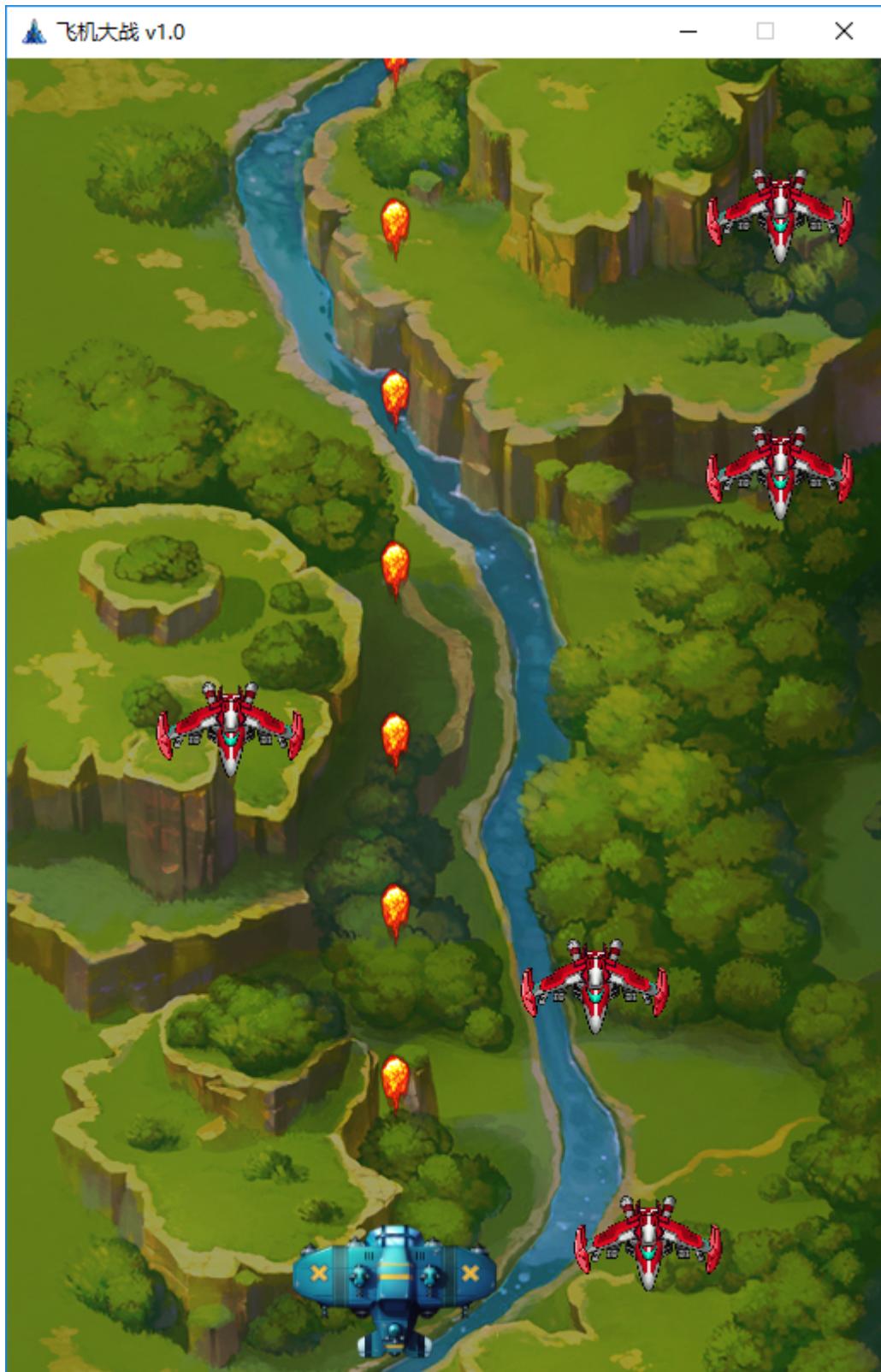


飞机大战_Qt制作

1 项目简介

飞机大战是我们大家所熟知的一款小游戏，本教程就是教大家如何制作一款自己的飞机大战

首先我们看一下效果图



玩家控制一架小飞机，然后自动发射子弹，如果子弹打到了飞下来的敌机，则射杀敌机，并且有爆炸的特效

接下来再说明一下案例的需求，也就是我们需要实现的内容

- 滚动的背景地图

- 飞机的制作和控制
- 子弹的制作和射击
- 敌机的制作
- 碰撞检测
- 爆炸效果
- 音效添加

2 创建项目

创建项目步骤如下：

- 打开Qt
- 按照向导创建项目
- 项目名称位置填写
- 类信息加入
- 完成创建

2.1 打开Qt

找到你安装的Qt Creator，打开它

如果安装时，没有选择在桌面上建立快捷方式，那么你的Qt软件位置如下

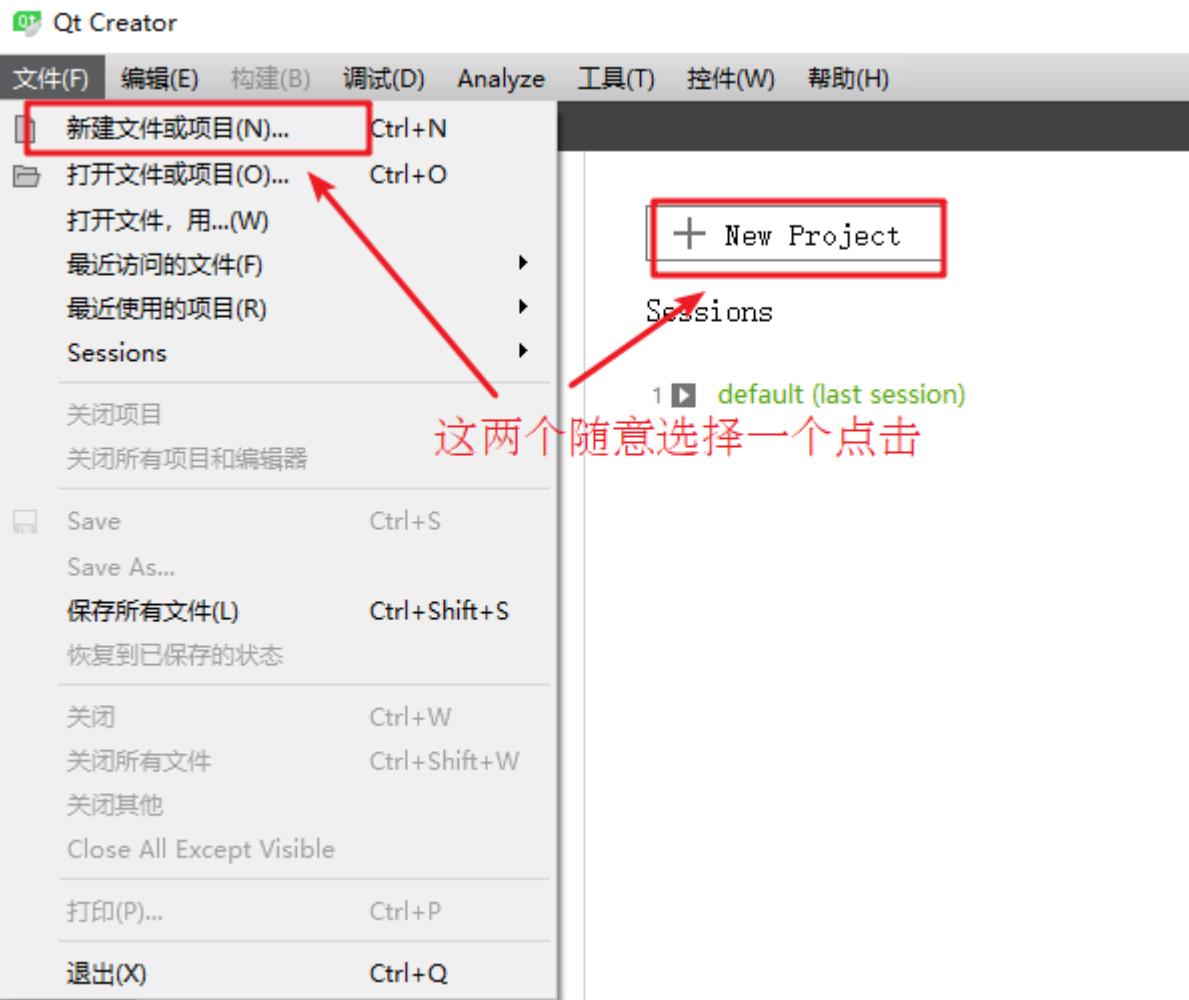
C:\qt\Qt5.x.x\Tools\QtCreator\bin

在这个路径下找到 `qtcreator.exe` 双击打开即可

2.2 按照向导创建项目

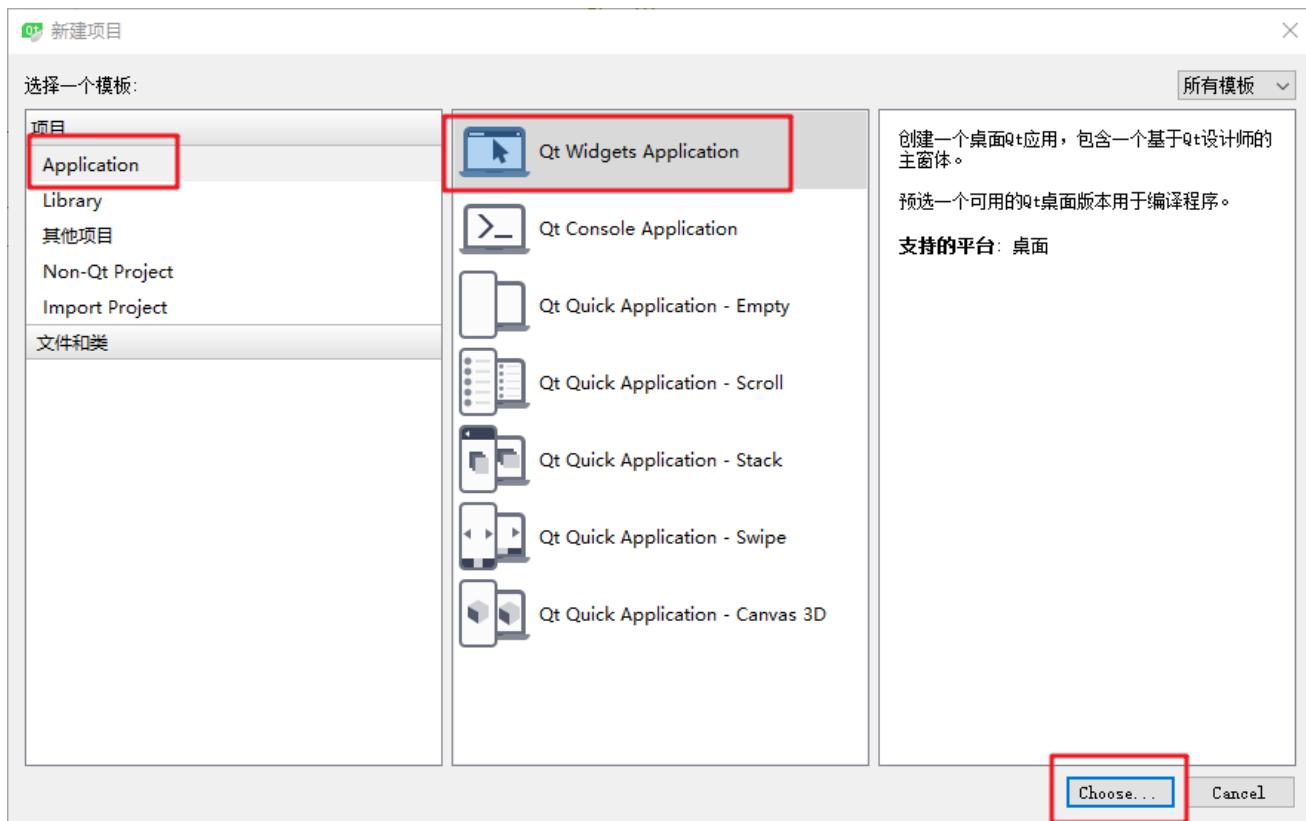
2.2.1 新建项目

点击菜单 中的文件 -> 新建文件或项目 或者 在首页面中点击New Project



2.2.2 选择模板

模板选择 Application -> Qt Widget Application



2.2.3 项目名称和位置

给项目起个名称以及选中项目要保存的地方



这一步选择后在Kits 构建套件中直接点击下一步即可

2.2.4 类信息

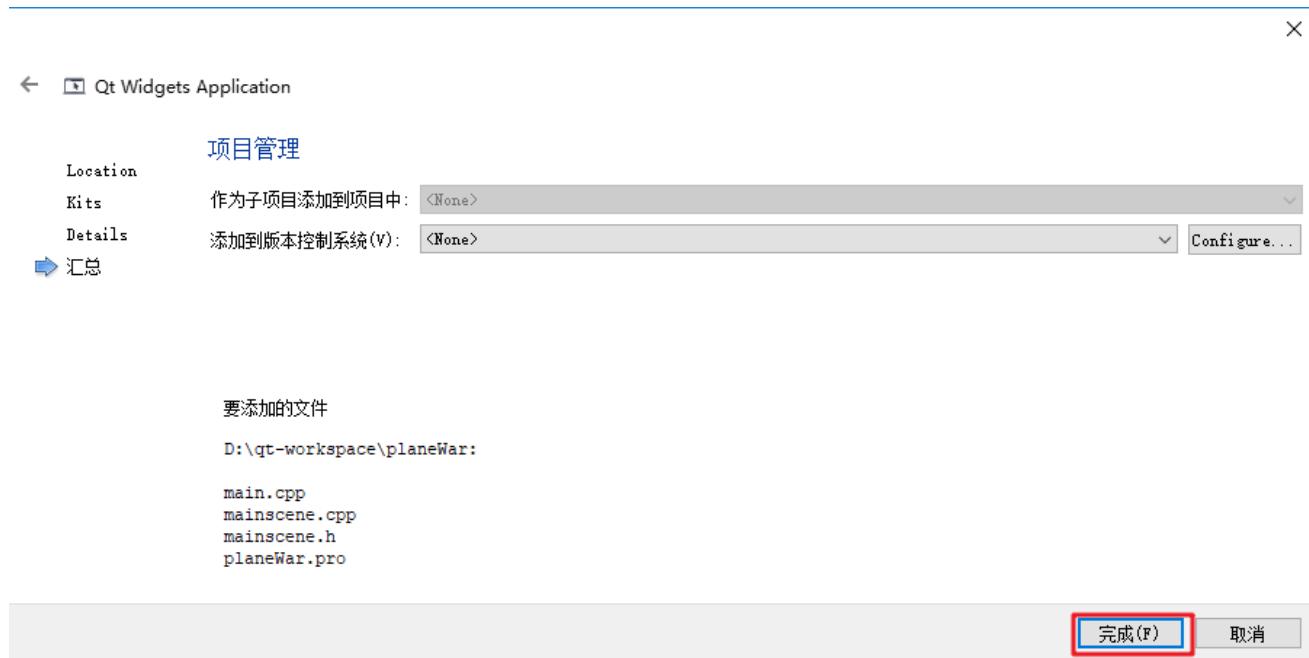
基类选择 QWidget

类名也就是我们第一个窗口场景的名称，这里我起名为 MainScene 代表游戏中的主场景

取消创建界面中的内容



2.2.5 完成创建



在汇总页面中点击完成，我们就迈开了项目的第一步！

3 设置主场景

主场景设置的步骤如下：

- 添加配置文件，保存游戏中所有配置数据
- 初始化主场景窗口大小、标题

3.1 配置文件添加

创建新的头文件为 config.h 主要记录程序中所有的配置数据，方便后期修改

添加窗口宽度、高度的配置信息，依据背景图大小进行设置

```
***** 游戏配置数据 *****/
#define GAME_WIDTH 512 //宽度
#define GAME_HEIGHT 768 //高度
#define GAME_TITLE "飞机大战 v1.0" //标题
```

3.2 主场景基本设置

在mainScene.h中添加新的成员函数initScene 用来初始化游戏场景

```
void initScene();
```

在mainScene.cpp中实现如下代码

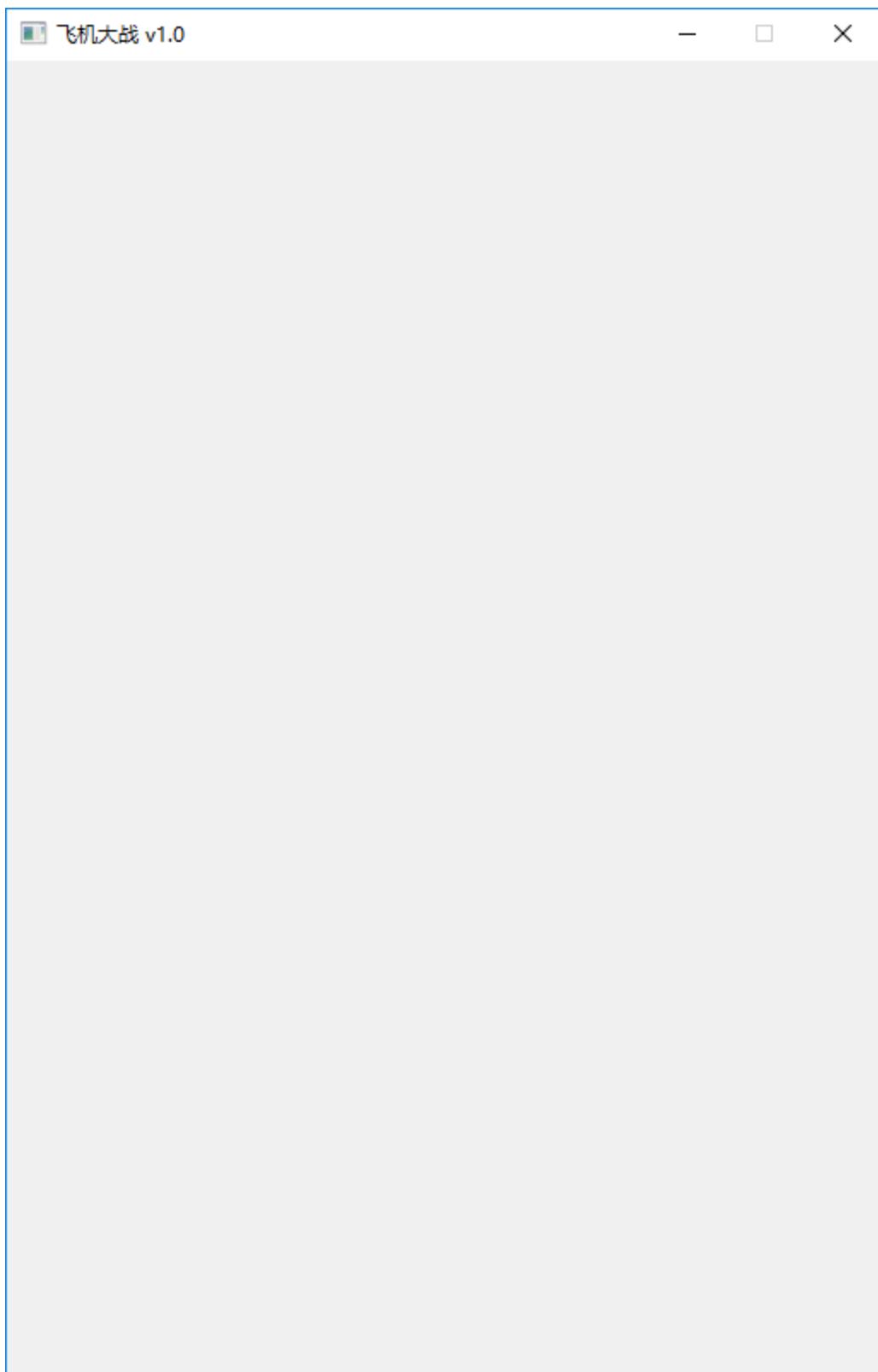
```
void MainScene::initScene()
{
    //初始化窗口大小
    setFixedSize(GAME_WIDTH, GAME_HEIGHT);

    //设置窗口标题
    setWindowTitle(GAME_TITLE);
}
```

在构造函数MainScene中调用该函数 initScene

```
MainScene::MainScene(QWidget *parent)
: QWidget(parent)
{
    //初始化场景
    initScene();
}
```

测试运行效果如图：



4 资源导入

在主场景中其实还有一个配置项没有实现，也就是窗口左上角的那个图标资源

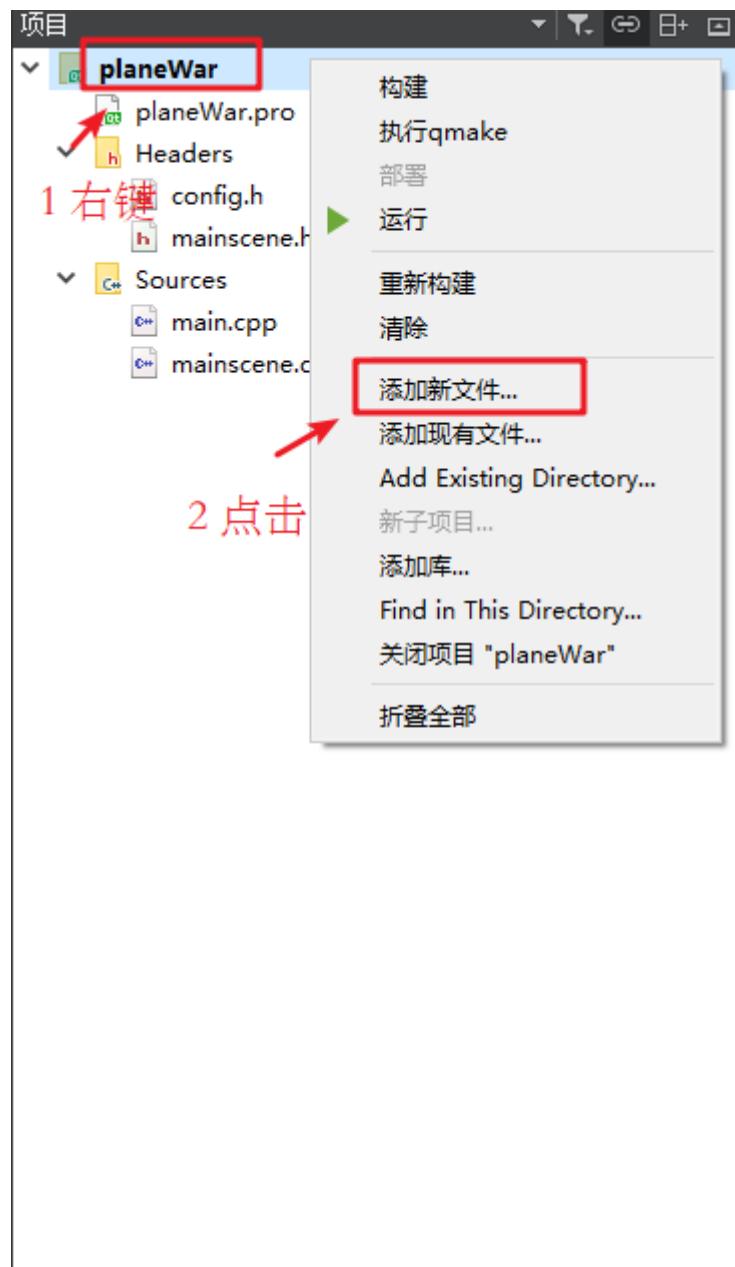
那么接下来我们将游戏中的资源进行导入并且设置游戏图标

资源导入步骤

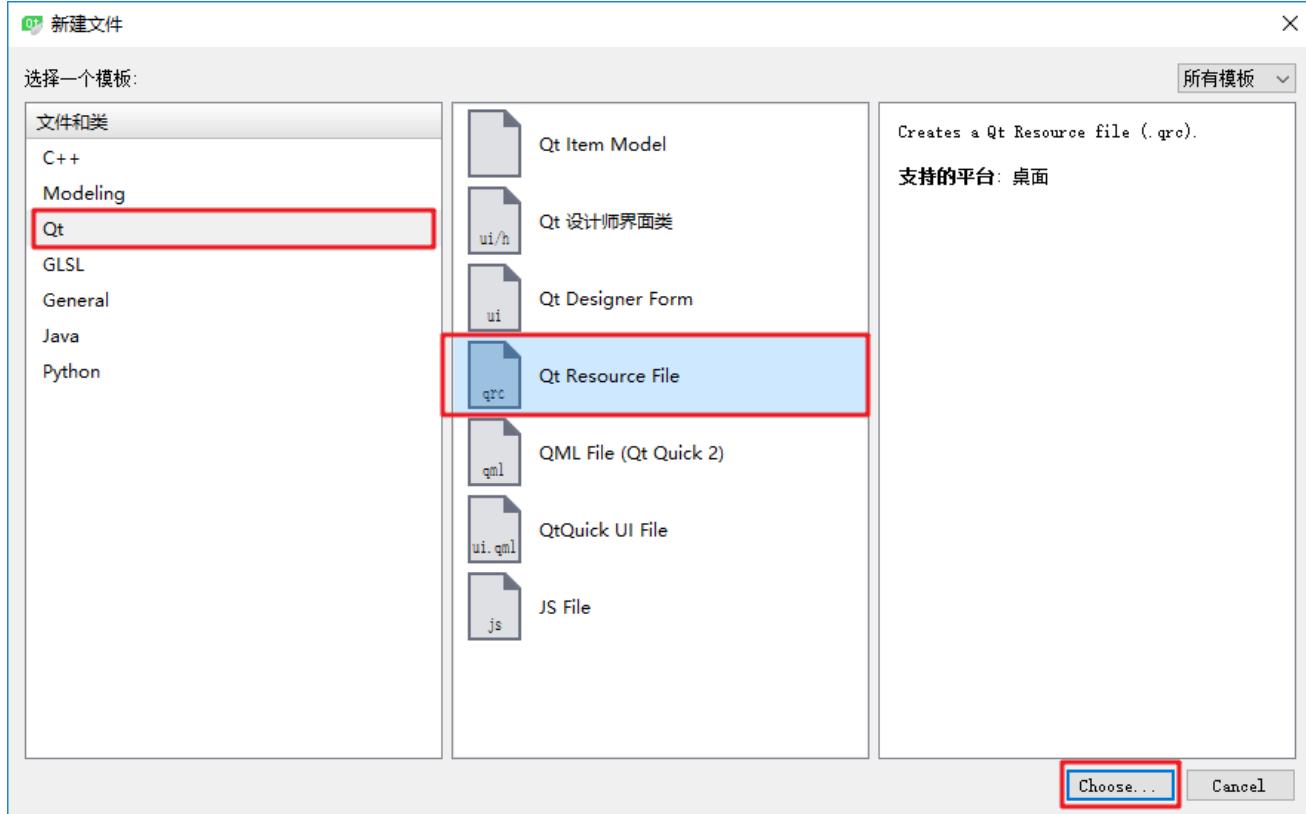
- 生成qrc文件
- 项目同级目录下创建res文件夹并将资源粘贴过来
- 编辑qrc，加入前缀和文件
- 利用qrc生成二进制文件 rcc
- rcc文件放入到debug同级目录下
- 注册二进制文件
- 添加图标资源

4.1 qrc文件生成

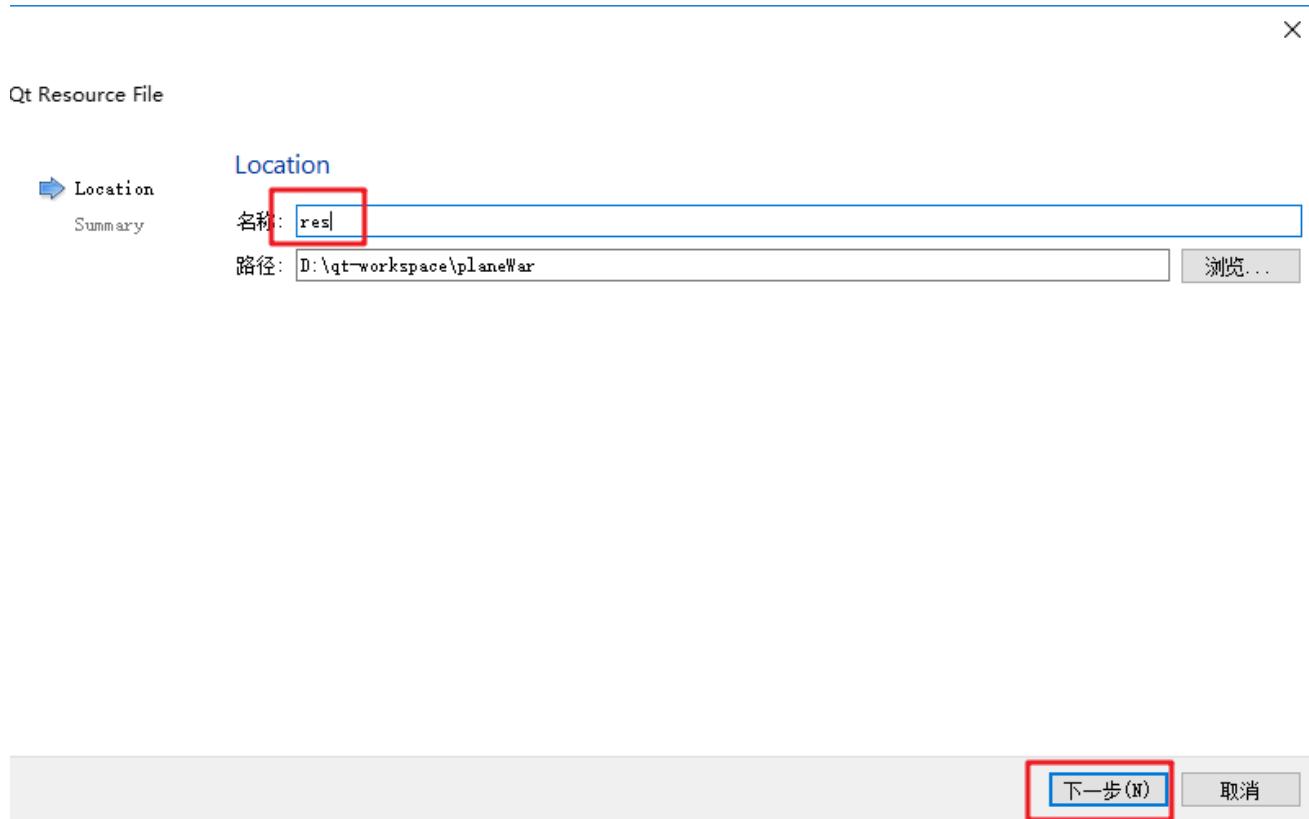
右键项目，点击添加新文件



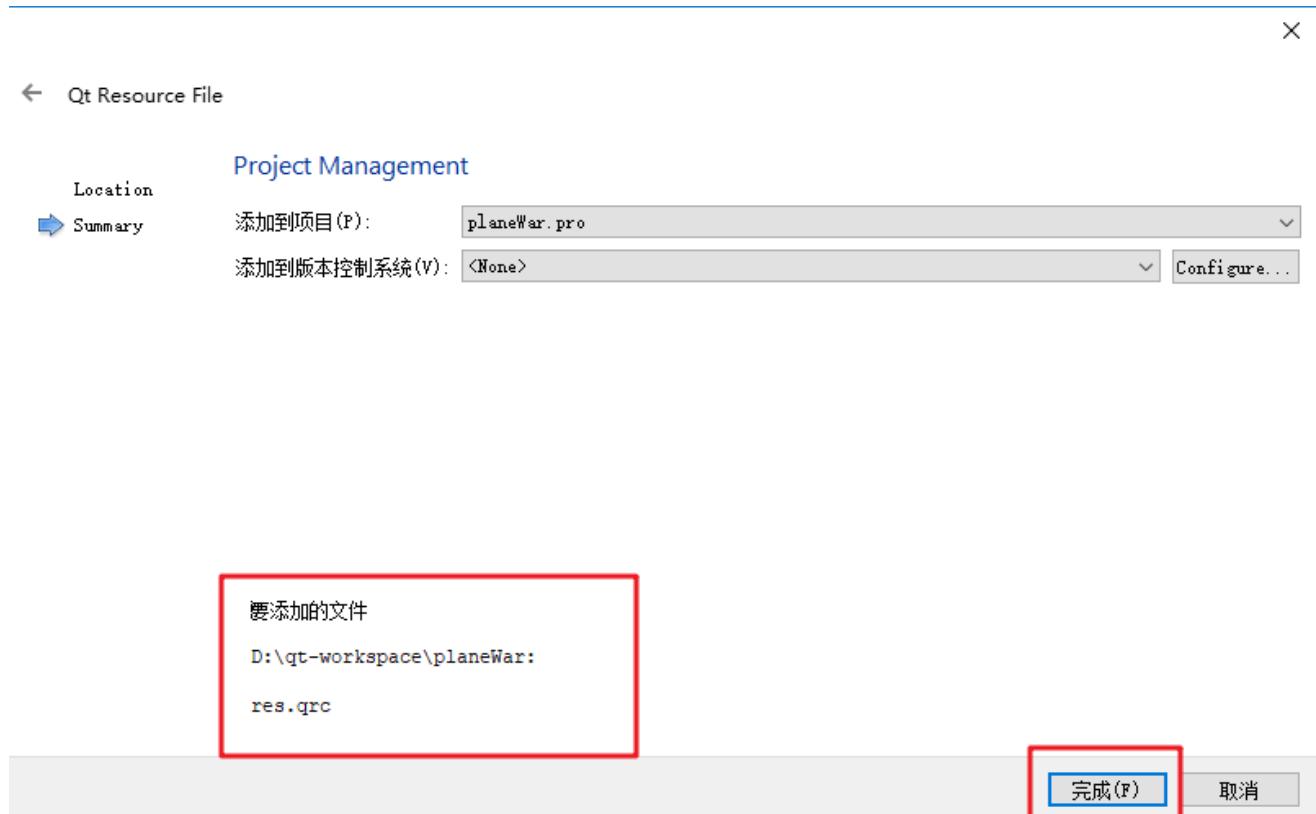
选择Qt -> Qt Resource File



资源文件起名 如 : res

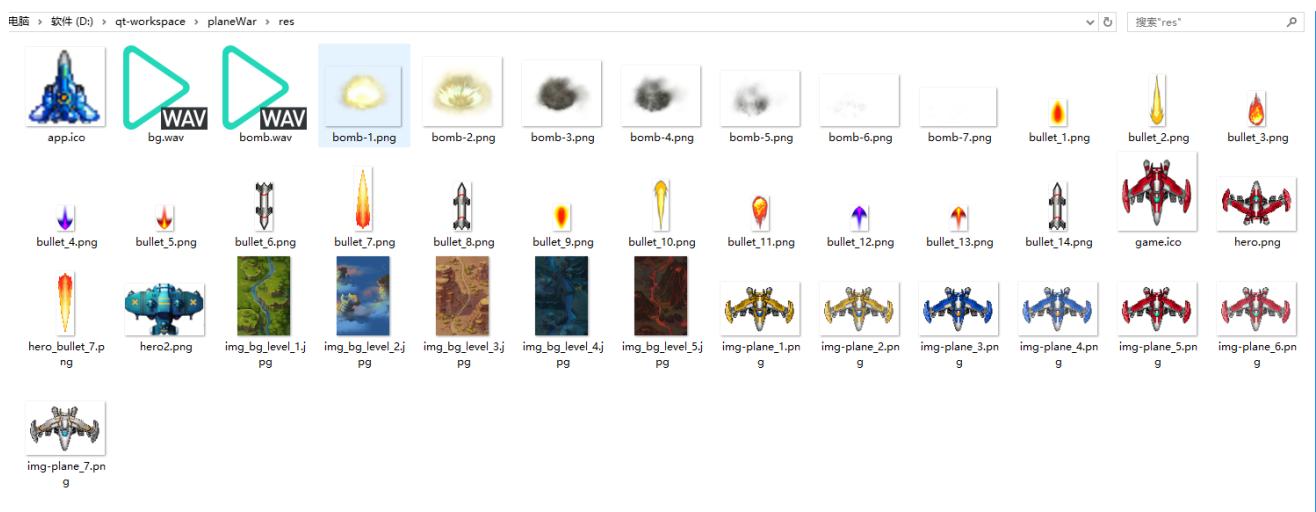


生成res.qrc文件



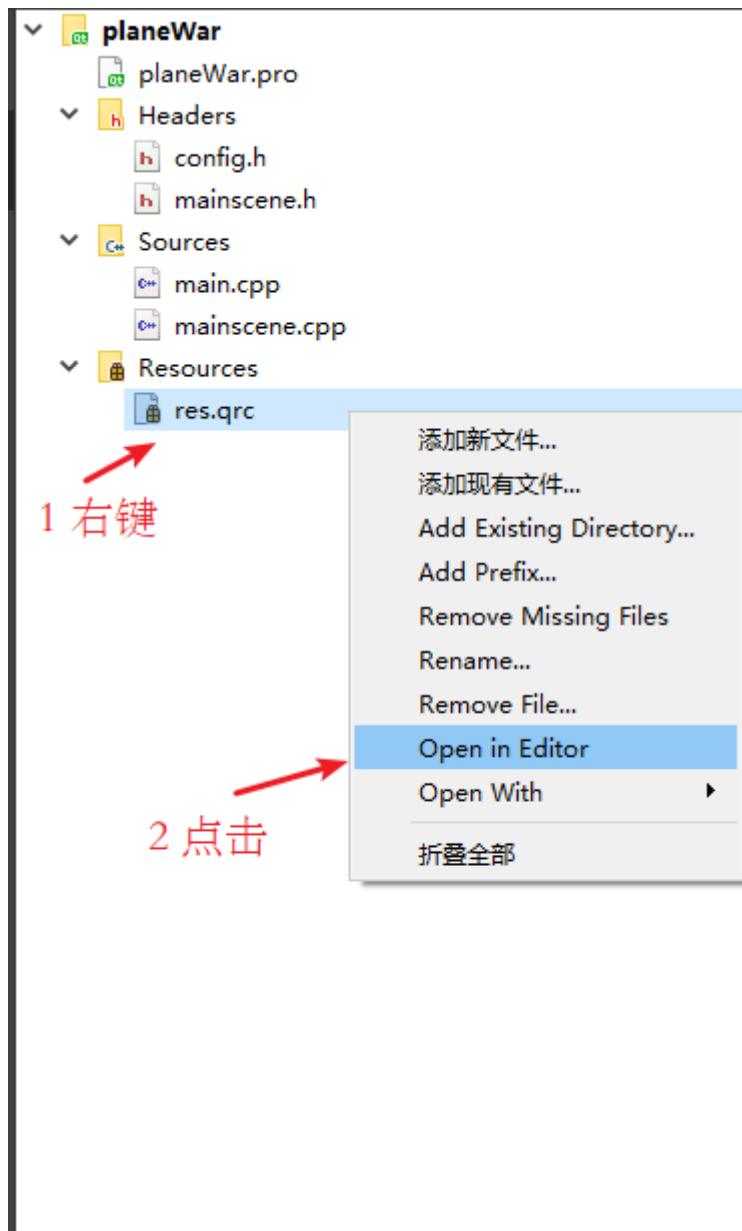
4.2 创建res文件夹

项目的同级目录下创建文件夹res，并将准备好的资源粘贴进去

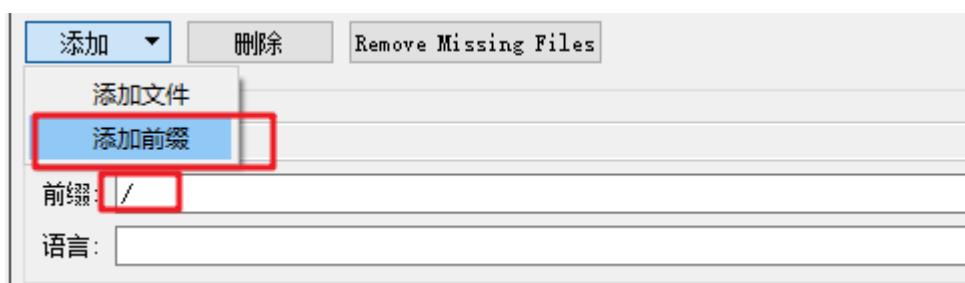


4.3 编辑qrc文件

右键qrc文件，选中Open in Editor



添加前缀为 "/"



添加文件 将res下所有文件选中即可

1、选中点击文件，将res下所有资源选中点击确定

2、成功后所有资源会显示到项目中

- res/bomb-1.png
- res/bomb-2.png
- res/bomb-3.png
- res/bomb-4.png
- res/bomb-5.png
- res/bomb-6.png
- res/bomb-7.png
- res/bullet_1.png
- res/bullet_2.png
- res/bullet_3.png
- res/bullet_4.png
- res/bullet_5.png
- res/bullet_6.png
- res/bullet_7.png
- res/bullet_8.png
- res/bullet_9.png
- res/bullet_10.png
- res/bullet_11.png
- res/bullet_12.png
- res/bullet_13.png
- res/bullet_14.png
- res/game.ico
- res/hero.png
- res/hero_bullet_7.png
- res/hero2.png
- res/img_bg_level_1.jpg
- res/img_bg_level_2.jpg
- res/img_bg_level_3.jpg
- res/img_bg_level_4.jpg
- res/img_bg_level_5.jpg
- res/img-plane_1.png
- res/img-plane_2.png
- res/img-plane_3.png
- res/img-plane_4.png
- res/img-plane_5.png
- res/img-plane_6.png
- res/img-plane_7.png

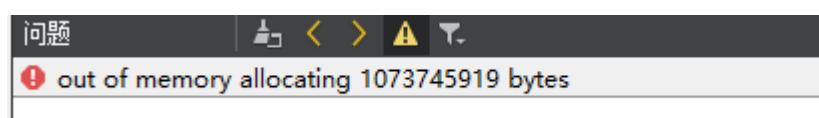
添加 删除 Remove Missing Files

添加文件

添加前缀

4.4 qrc生成 rcc二进制文件

由于资源过大，会提示错误：



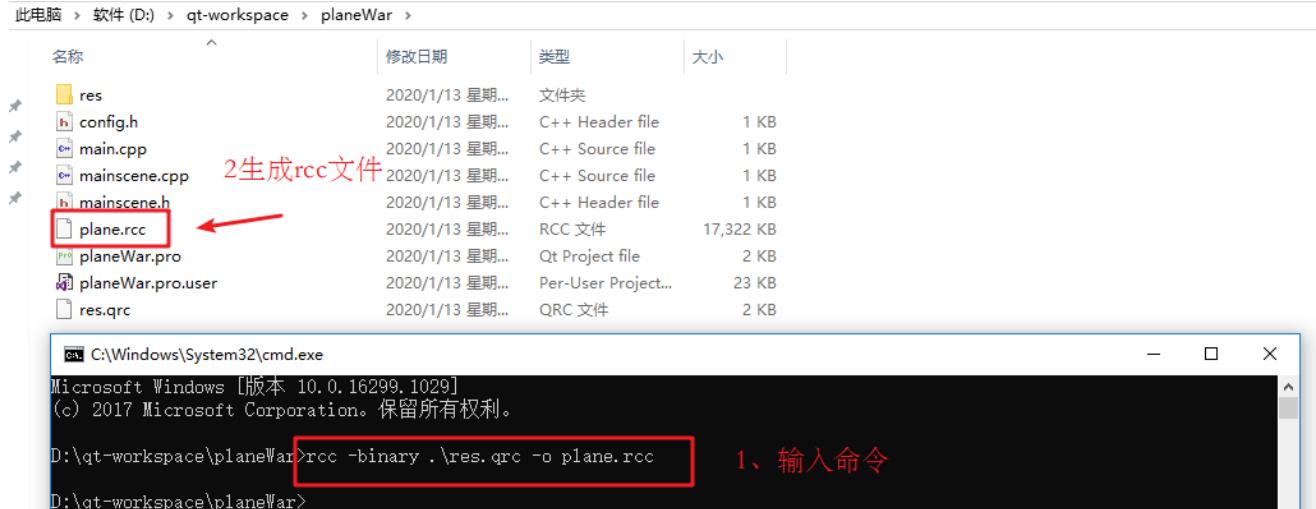
这个错误也就是“编译器的堆空间不足”。

由于资源文件qrc过大，超出分配的内存范围

因此我们需要利用二进制资源，而生成二进制资源就需要我们刚刚的qrc文件

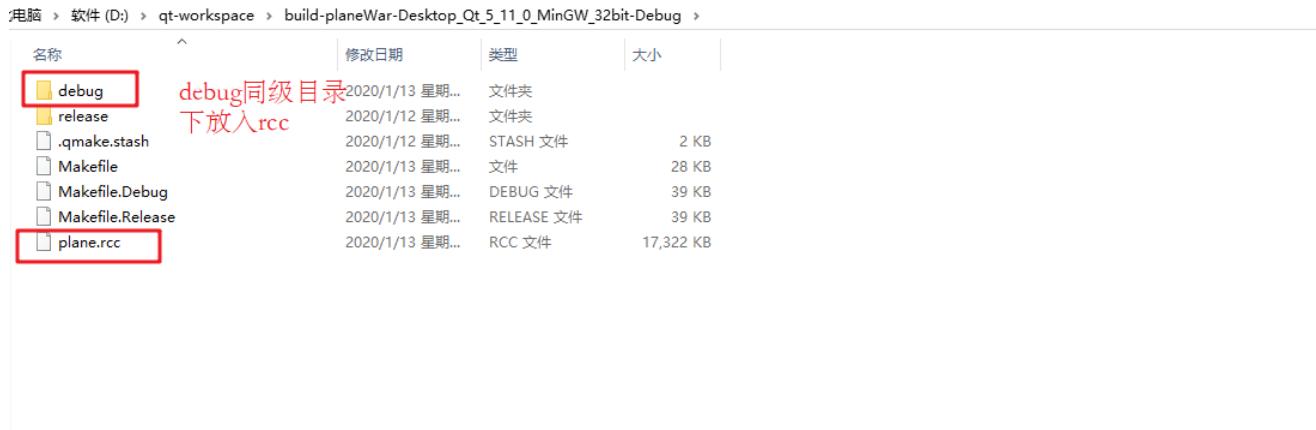
利用cmd打开终端，定位到res.qrc的目录下，输入命令

```
rcc -binary .\res.qrc -o plane.rcc
```



4.5 复制rcc文件

将生成好的rcc文件，放入到debug同级目录中一份



4.6 注册二进制文件

在config.h中追加配置数据

```
#define GAME_RES_PATH "./plane.rcc" //rcc文件路径
```

在main.cpp中修改代码

```
#include "mainscene.h"
#include <QApplication>
#include <QResource>
#include "config.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    //注册外部的二进制资源文件
    QResource::registerResource(GAME_RES_PATH);

    MainScene w;
    w.show();

    return a.exec();
}
```

此时，qrc文件已经没用了，删除即可！

最简单的删除方式就是 .pro工程文件中删除代码,与工程无瓜葛

删除以下代码：

```
RESOURCES += \
    res.qrc
```

4.7 添加图标资源

配置文件config.h中追加代码

虚拟资源路径语法如下：

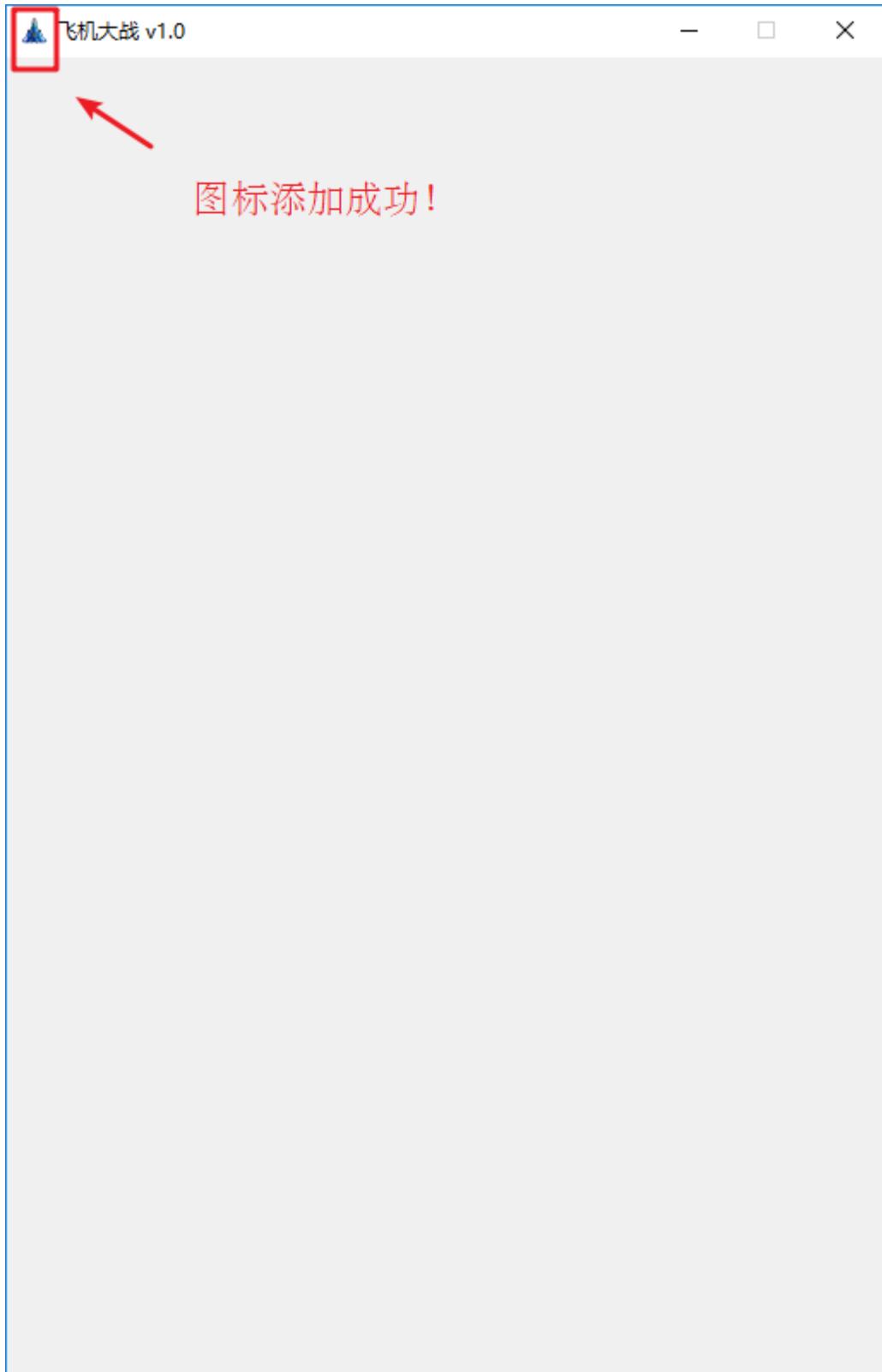
" :+ 前缀名 + 文件路径 "

```
#define GAME_ICON (":/res/app.ico")
```

在mainScene.cpp的 initScene函数中追加代码：

```
//设置图标资源
setWindowIcon(QIcon( GAME_ICON)); //加头文件 #include <QIcon>
```

运行测试：



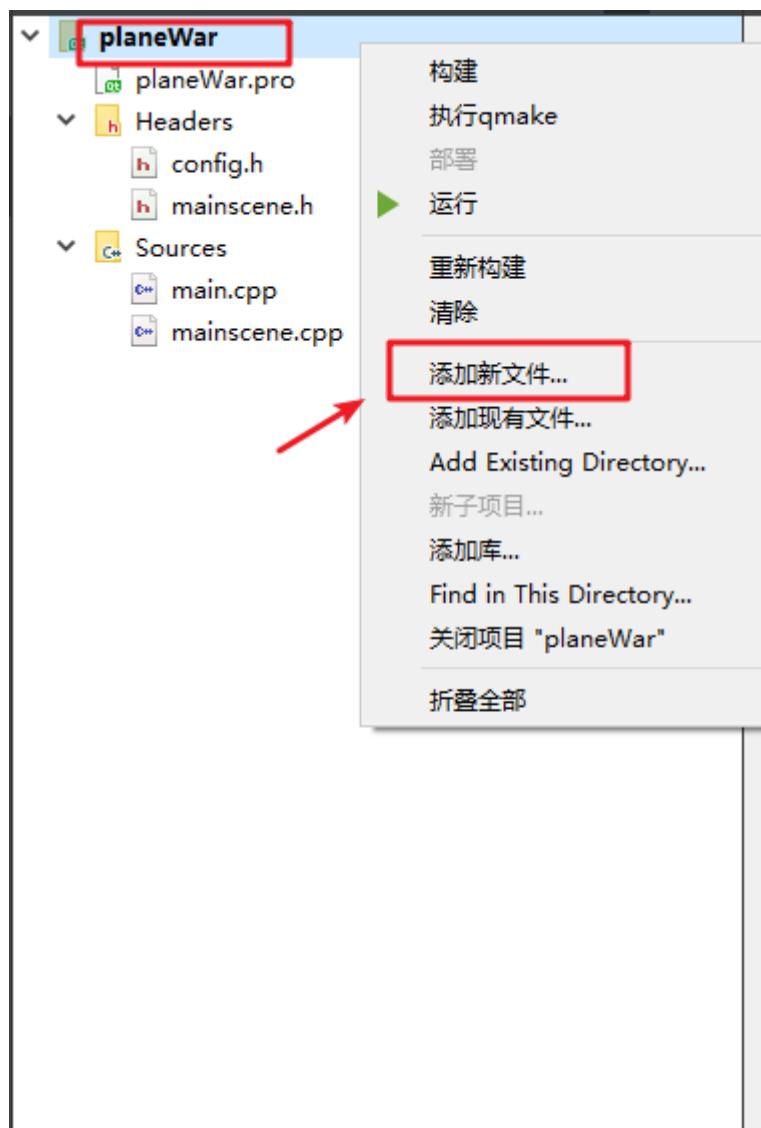
5 地图滚动

步骤：

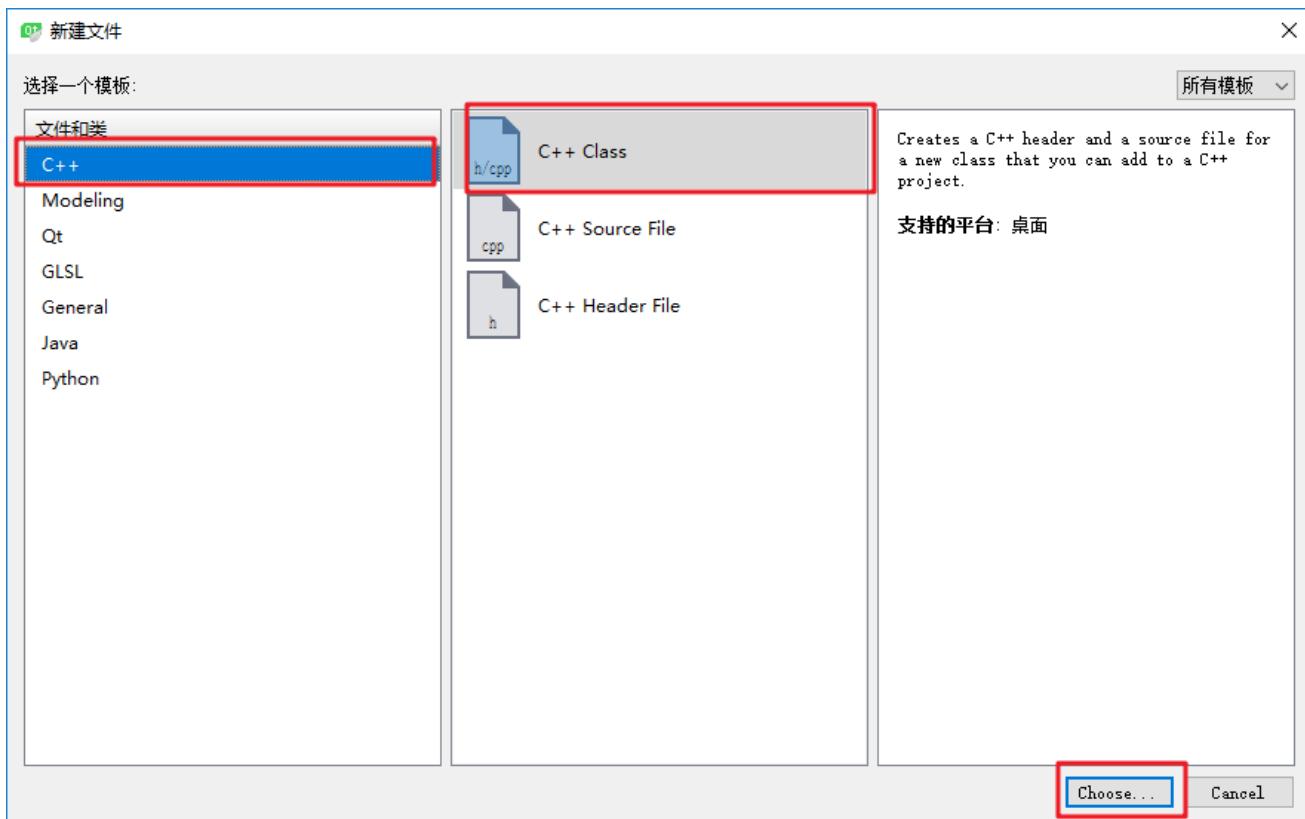
- 创建地图文件和类
- 添加成员函数和成员属性 实现成员函数
- 游戏运行调用定时器
- 启动定时器，监听定时器信号实现游戏循环
 - 计算游戏内元素坐标
 - 绘制到屏幕中

5.1 创建地图文件和类

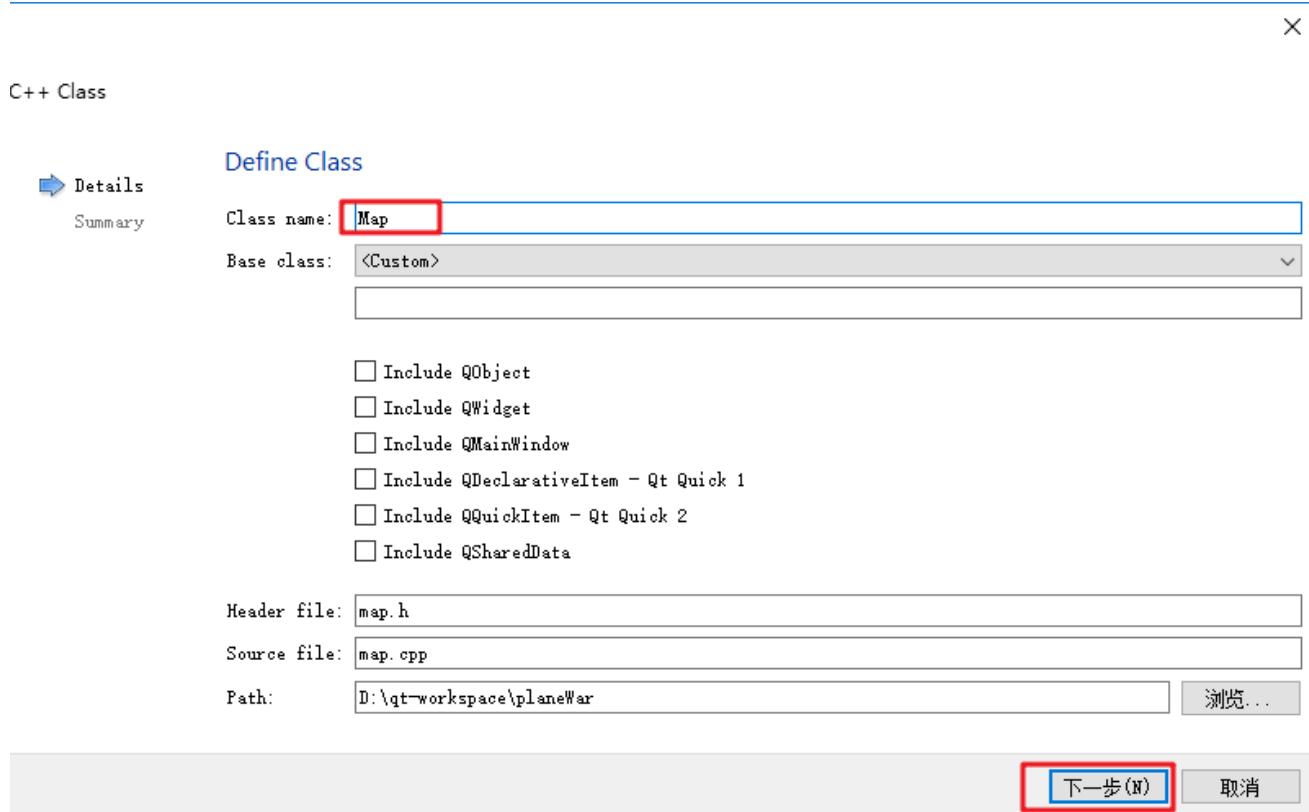
右键项目，添加新文件



选择C++ -> C++ Class



修改类名为map，点击下一步，直到创建完毕



至此，地图Map的文件和类创建完毕

5.2 地图的成员函数和成员属性

在map.h中添加如下代码

```
#ifndef MAP_H
#define MAP_H
#include <QPixmap>

class Map
{
public:
    //构造函数
    Map();

    //地图滚动坐标计算
    void mapPosition();

public:
    //地图图片对象
    QPixmap m_map1;
    QPixmap m_map2;

    //地图Y轴坐标
    int m_map1_posY;
    int m_map2_posY;

    //地图滚动幅度
    int m_scroll_speed;
};

#endif // MAP_H
```

5.3 实现成员函数

在config.h中添加新的配置数据

```
***** 地图配置数据 *****/
#define MAP_PATH ":/res/img_bg_level_1.jpg" //地图图片路径
#define MAP_SCROLL_SPEED 2 //地图滚动速度
```

在map.cpp中实现成员函数

```
#include "map.h"
#include "config.h"

Map::Map()
{
    //初始化加载地图对象
```

```

    m_map1.load(MAP_PATH);
    m_map2.load(MAP_PATH);

    //设置地图其实y轴坐标
    m_map1_posY = -GAME_HEIGHT;
    m_map2_posY = 0;

    //设置地图滚动速度
    m_scroll_speed = MAP_SCROLL_SPEED;
}

void Map::mapPosition()
{
    //处理第一张图片滚动
    m_map1_posY += MAP_SCROLL_SPEED;
    if(m_map1_posY >= 0)
    {
        m_map1_posY -= GAME_HEIGHT;
    }

    //处理第二张图片滚动
    m_map2_posY += MAP_SCROLL_SPEED;
    if(m_map2_posY >= GAME_HEIGHT )
    {
        m_map2_posY =0;
    }
}

```

5.4 定时器添加

在mainScene.h中添加新的定时器对象

```
QTimer m_Timer;
```

在 config.h中添加 屏幕刷新间隔

```
#define GAME_RATE 10 //刷新间隔，帧率 单位毫秒
```

在MainScene.cpp的initScene中追加代码

```
//定时器设置
m_Timer.setInterval(GAME_RATE);
```

5.5 启动定时器实现地图滚动

在MainScene.h中添加新的成员函数以及成员对象

```
//启动游戏 用于启动定时器对象
void playGame();
//更新坐标
void updatePosition();
//绘图事件
void paintEvent(QPaintEvent *event);

//地图对象
Map m_map;
```

在MainScene.cpp中实现成员函数

```
void MainScene::playGame()
{
    //启动定时器
    m_Timer.start();

    //监听定时器
    connect(&m_Timer,&QTimer::timeout,[=](){
        //更新游戏中元素的坐标
        updatePosition();
        //重新绘制图片
        update();
    });
}

void MainScene::updatePosition()
{
    //更新地图坐标
    m_map.mapPosition();
}

void MainScene::paintEvent(QPaintEvent *event)
{
    QPainter painter(this);

    //绘制地图
    painter.drawPixmap(0,m_map.m_map1_posY , m_map.m_map1);
    painter.drawPixmap(0,m_map.m_map2_posY , m_map.m_map2);
}
```

测试运行游戏，实现地图滚动

6 英雄飞机

步骤如下：

- 创建英雄文件和类
- 添加成员函数和成员属性
- 实现成员函数
- 创建飞机对象并显示
- 拖拽飞机

6.1 创建英雄文件和类

创建HeroPlane类以及生成对应的文件

和创建地图的步骤一样，这里就不在详细截图了

创建好后生成HeroPlane.h 和 HeroPlane.cpp两个文件

The screenshot shows the Qt Creator interface. On the left, the project tree for 'planeWar' is displayed, containing 'planeWar.pro', 'Headers' (with 'config.h' and 'heroplane.h'), and 'Sources' (with 'heroplane.cpp', 'main.cpp', 'mainscene.cpp', and 'map.cpp'). The 'heroplane.h' file is selected and highlighted with a red box. On the right, the code editor shows the generated code for 'heroplane.h':

```
1 #ifndef HEROPLANE_H
2 #define HEROPLANE_H
3
4
5 class HeroPlane
6 {
7 public:
8     HeroPlane();
9 };
10
11 #endif // HEROPLANE_H
```

6.2 飞机的成员函数和成员属性

在HeroPlane.h中添加代码

```
class HeroPlane
{
public:
    HeroPlane();

    //发射子弹
    void shoot();
    //设置飞机位置
    void setPosition(int x, int y);
```

```
public:  
    //飞机资源 对象  
    QPixmap m_Plane;  
  
    //飞机坐标  
    int m_X;  
    int m_Y;  
  
    //飞机的矩形边框  
    QRect m_Rect;  
};
```

6.3 成员函数实现

这里飞机有个发射子弹的成员函数，由于我们还没有做子弹

因此这个成员函数先写成空实现即可

在config.h中追加飞机配置参数

```
***** 飞机配置数据 *****/  
#define HERO_PATH ":/res/hero2.png"
```

heroPlane.cpp中实现成员函数代码：

```
#include "heroplane.h"  
#include "config.h"  
  
HeroPlane::HeroPlane()  
{  
    //初始化加载飞机图片资源  
    m_Plane.load(HERO_PATH);  
  
    //初始化坐标  
    m_X = GAME_WIDTH * 0.5 - m_Plane.width()*0.5;  
    m_Y = GAME_HEIGHT - m_Plane.height();  
  
    //初始化矩形框  
    m_Rect.setWidth(m_Plane.width());  
    m_Rect.setHeight(m_Plane.height());  
    m_Rect.moveTo(m_X,m_Y);  
  
}  
  
void HeroPlane::setPosition(int x, int y)  
{
```

```
m_X = x;  
m_Y = y;  
m_Rect.moveTo(m_X,m_Y);  
}  
  
void HeroPlane::shoot()  
{  
  
}
```

6.4 创建飞机对象并显示

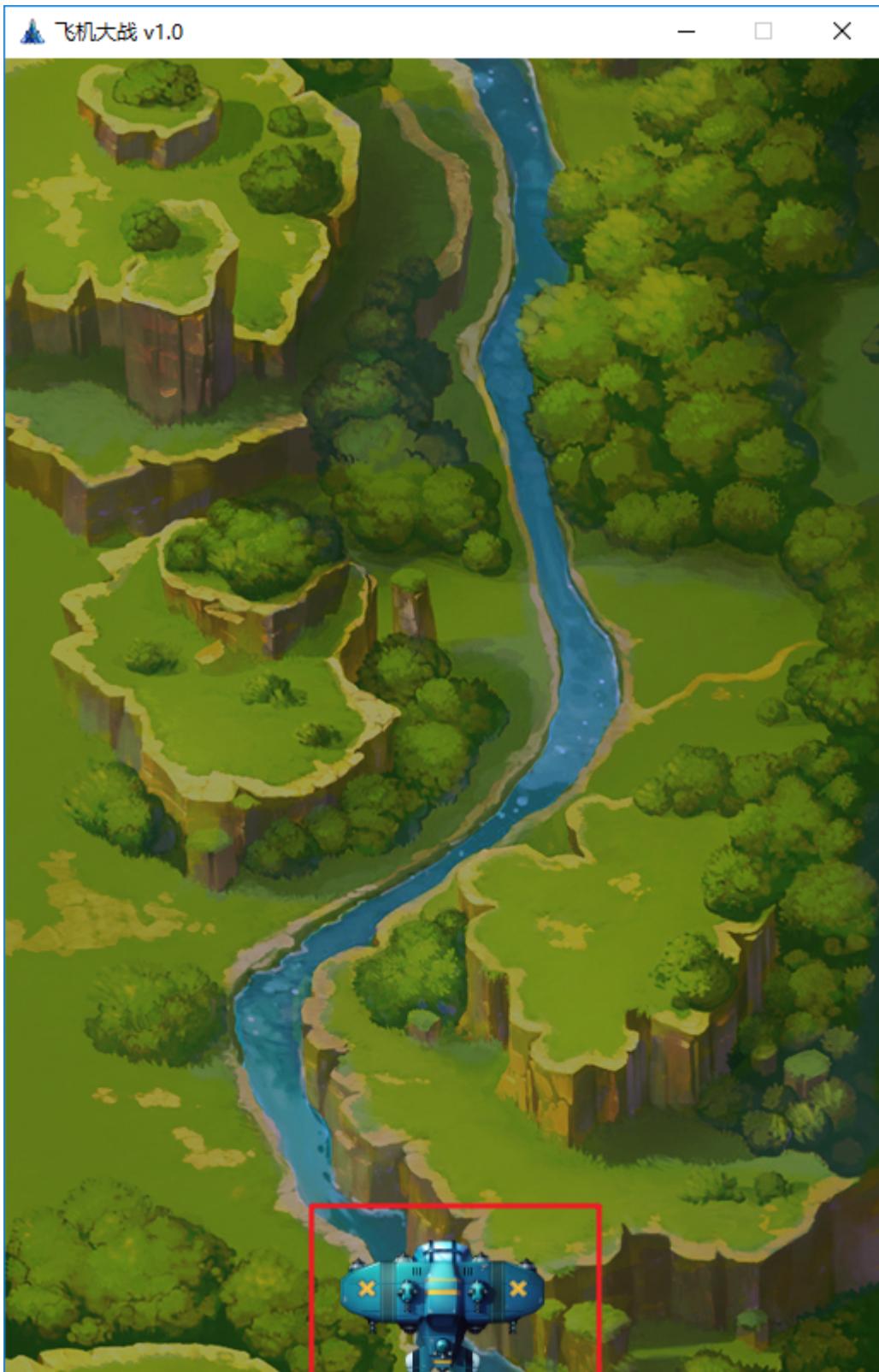
在MainScene.h中追加新的成员属性

```
//飞机对象  
HeroPlane m_hero;
```

在MainScene.cpp的paintEvent中追加代码

```
//绘制英雄  
painter.drawPixmap(m_hero.m_X,m_hero.m_Y,m_hero.m_Plane);
```

测试飞机显示到屏幕中



6.5 拖拽飞机

在MainScene.h中添加鼠标移动事件

```
//鼠标移动事件  
void mouseMoveEvent(QMouseEvent *event);
```

重写鼠标移动事件

```
void MainScene::mouseMoveEvent(QMouseEvent *event)  
{  
    int x = event->x() - m_hero.m_Rect.width()*0.5; //鼠标位置 - 飞机矩形的一半  
    int y = event->y() - m_hero.m_Rect.height()*0.5;  
  
    //边界检测  
    if(x <= 0)  
    {  
        x = 0;  
    }  
    if(x >= GAME_WIDTH - m_hero.m_Rect.width())  
    {  
        x = GAME_WIDTH - m_hero.m_Rect.width();  
    }  
    if(y <= 0)  
    {  
        y = 0;  
    }  
    if(y >= GAME_HEIGHT - m_hero.m_Rect.height())  
    {  
        y = GAME_HEIGHT - m_hero.m_Rect.height();  
    }  
    m_hero.setPosition(x,y);  
}
```

测试飞机可以拖拽



7 子弹制作

制作步骤如下：

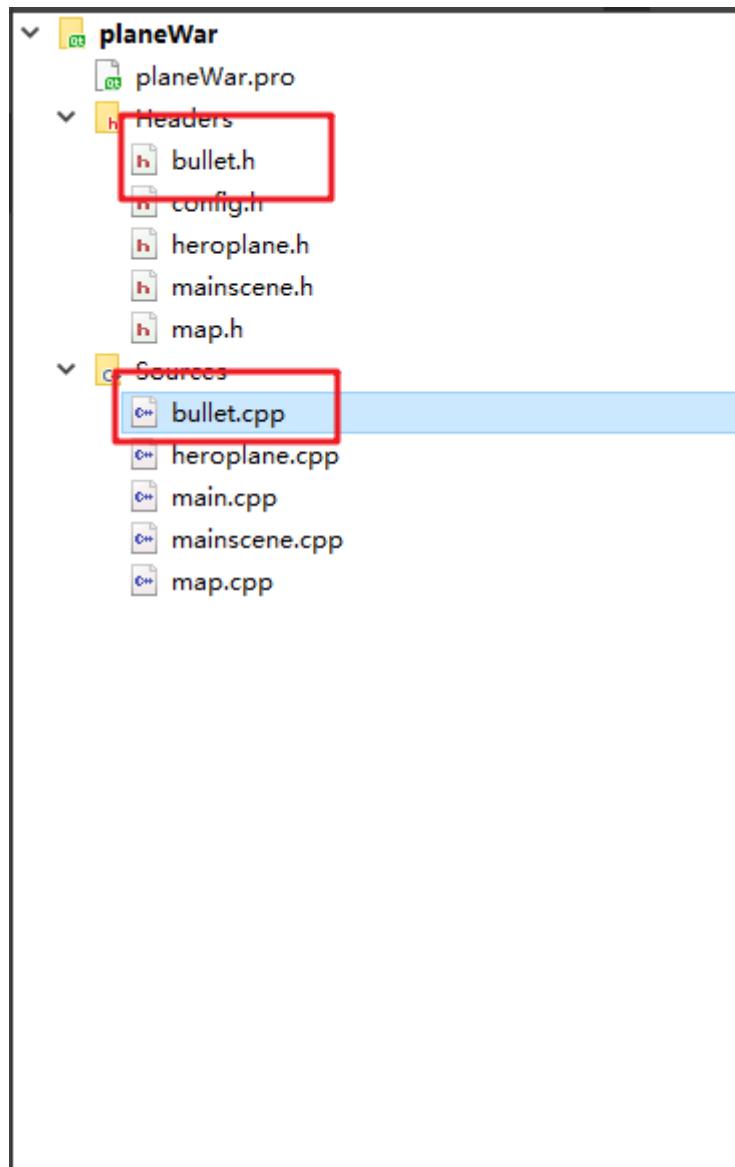
- 创建子弹文件和类
- 添加子弹类中的成员函数和成员属性

- 实现成员函数
- 测试子弹

7.1 创建子弹文件和类

创建Bullet类以及生成对应的文件

创建好后生成bullet.h 和 bullet.cpp两个文件



7.2 子弹的成员函数和成员属性

在Bullet.h中添加代码

```
#ifndef BULLET_H
#define BULLET_H
#include "config.h"
#include <QPixmap>
```

```

class Bullet
{
public:
    Bullet();

    //更新子弹坐标
    void updatePosition();

public:
    //子弹资源对象
    QPixmap m_Bullet;
    //子弹坐标
    int m_X;
    int m_Y;
    //子弹移动速度
    int m_Speed;
    //子弹是否闲置
    bool m_Free;
    //子弹的矩形边框（用于碰撞检测）
    QRect m_Rect;
};

#endif // BULLET_H

```

7.3 子弹类成员函数实现

在config.h中追加子弹配置信息

```

***** 子弹配置数据 *****/
#define BULLET_PATH ":/res/bullet_11.png"      //子弹图片路径
#define BULLET_SPEED 5  //子弹移动速度

```

在bullet.cpp中实现成员函数，代码如下：

```

#include "bullet.h"

Bullet::Bullet()
{
    //加载子弹资源
    m_Bullet.load(BULLET_PATH);

    //子弹坐标 初始坐标可随意设置，后期会重置
    m_X = GAME_WIDTH*0.5 - m_Bullet.width()*0.5;
    m_Y = GAME_HEIGHT;

    //子弹状态
    m_Free = true;
}

```

```

//子弹速度
m_Speed = BULLET_SPEED;

//子弹矩形框
m_Rect.setWidth(m_Bullet.width());
m_Rect.setHeight(m_Bullet.height());
m_Rect.moveTo(m_X,m_Y);
}

void Bullet::updatePosition()
{
    //如果子弹是空闲状态，不需要坐标计算
    //玩家飞机可以控制子弹的空闲状态为false
    if(m_Free)
    {
        return;
    }

    //子弹向上移动
    m_Y -= m_Speed;
    m_Rect.moveTo(m_X,m_Y);

    if(m_Y <= -m_Rect.height())
    {
        m_Free = true;
    }
}

```

7.4 测试子弹

子弹本身应该由飞机发射，测试阶段我们写一段辅助代码，看看效果即可

测试过后，这些代码可以删除掉

在MainScene.h中添加测试代码

```

//测试子弹代码
Bullet temp_bullet;

```

在MainScene.cpp中的updatePosition里添加测试代码

```

//测试子弹代码
temp_bullet.m_Free = false;
temp_bullet.updatePosition();

```

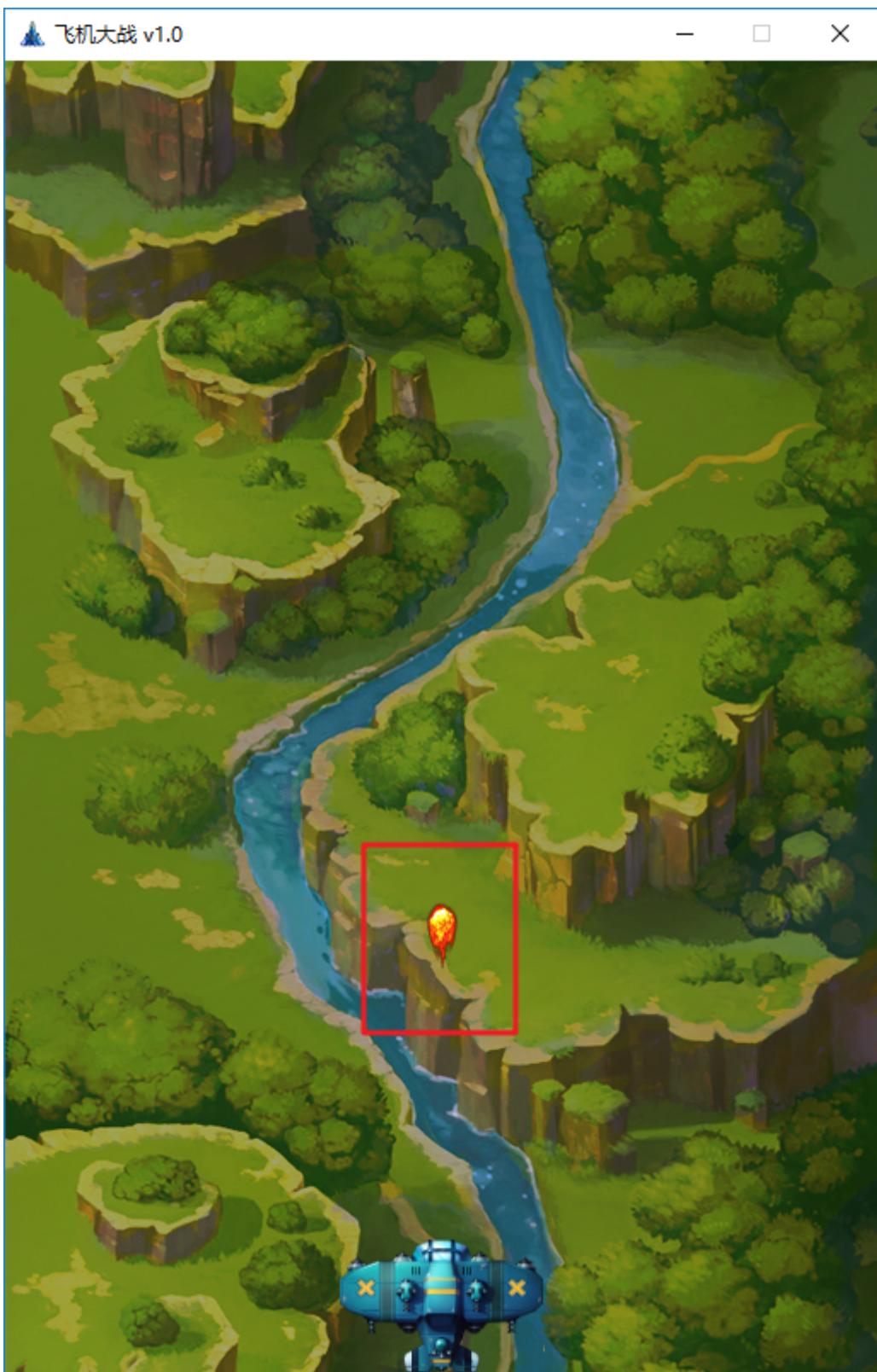
在MainScene.cpp中的paintEvent里添加测试代码

```

//测试子弹代码
painter.drawPixmap(temp_bullet.m_X,temp_bullet.m_Y,temp_bullet.m_Bullet);

```

运行程序，此时会有一发子弹从屏幕中射出



测试完毕后，测试代码删除或注释即可

8 玩家发射子弹

玩家发射子弹制作步骤如下：

- 英雄飞机添加新的成员属性
- 实现发射成员函数
- 主场景控制子弹发射

8.1 飞机添加新成员属性

在config.h中添加新的配置数据

```
#define BULLET_NUM 30 //弹匣中子弹总数  
#define BULLET_INTERVAL 20 //发射子弹时间间隔
```

在HeroPlane.h中新增成员属性如下：

```
//弹匣  
Bullet m_bullets[BULLET_NUM];  
  
//发射间隔记录  
int m_recorder;
```

8.2 成员函数补充

在构造函数 HeroPlane 中初始化发生间隔记录

```
//初始化发射间隔记录  
m_recorder = 0;
```

之前在英雄飞机类中预留的一个shoot函数我们进行实现，代码如下：

```
void HeroPlane::shoot()  
{  
    //累加时间间隔记录变量  
    m_recorder++;  
    //判断如果记录数字 未达到发射间隔，直接return  
    if(m_recorder < BULLET_INTERVAL)  
    {  
        return;  
    }  
    //到达发射时间处理  
    //重置发射时间间隔记录  
    m_recorder = 0;
```

```
//发射子弹
for(int i = 0 ; i < BULLET_NUM;i++)
{
    //如果是空闲的子弹进行发射
    if(m_bullets[i].m_Free)
    {
        //将改子弹空闲状态改为假
        m_bullets[i].m_Free = false;
        //设置发射的子弹坐标
        m_bullets[i].m_X = m_X + m_Rect.width()*0.5 - 10;
        m_bullets[i].m_Y = m_Y - 25 ;
        break;
    }
}
```

8.3 主场景中实现发射子弹

在MainScene.cpp的updatePosition成员函数中追加如下代码

```
//发射子弹
m_hero.shoot();
//计算子弹坐标
for(int i = 0 ;i < BULLET_NUM;i++)
{
    //如果子弹状态为非空闲，计算发射位置
    if(!m_hero.m_bullets[i].m_Free)
    {
        m_hero.m_bullets[i].updatePosition();
    }
}
```

在MainScene.cpp的paintEvent成员函数中追加如下代码：

```
//绘制子弹
for(int i = 0 ;i < BULLET_NUM;i++)
{
    //如果子弹状态为非空闲，计算发射位置
    if(!m_hero.m_bullets[i].m_Free)
    {

painter.drawPixmap(m_hero.m_bullets[i].m_X,m_hero.m_bullets[i].m_Y,m_hero.m_bullets[i].m_Bullet
);
    }
}
```

测试运行，玩家可以发射子弹



9 敌机制作

敌机制作与子弹制作原理类似，也是每隔一定的时间让敌机出场

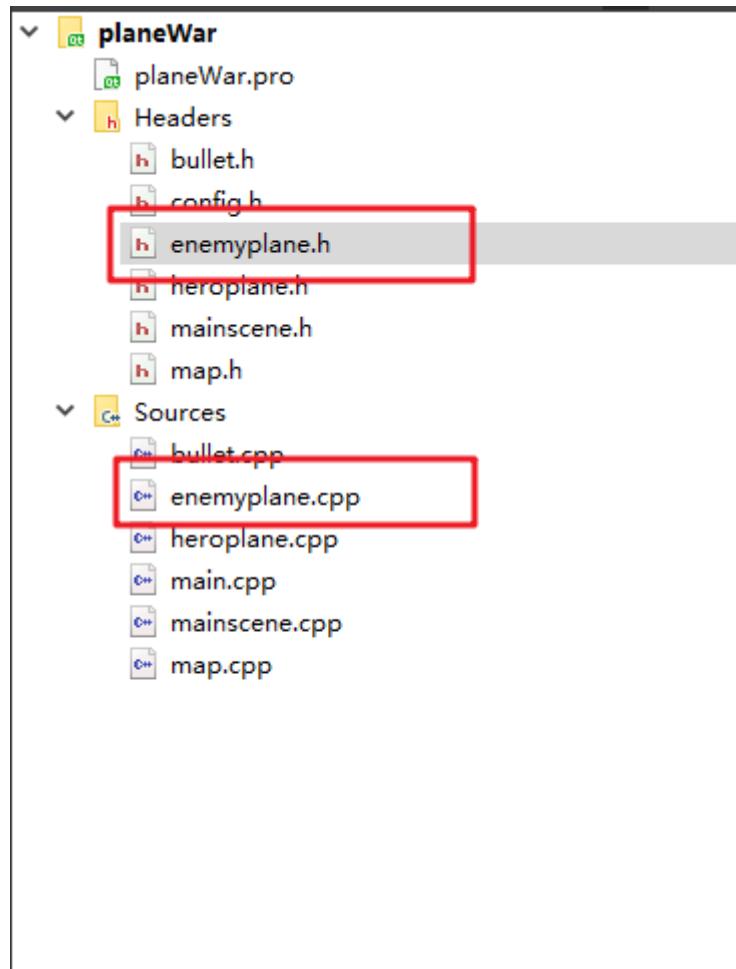
制作步骤如下：

- 创建敌机文件和类
- 添加敌机类中的成员函数和成员属性
- 实现成员函数
- 敌机出场
- 测试敌机

9.1 创建敌机文件和类

创建EnemyPlane类以及生成对应的文件

创建好后生成enemyPlane.h 和 enemyPlane.cpp两个文件



9.2 敌机成员函数和成员属性

在enemyPlane.h中添加如下代码：

```
#ifndef ENEMYPLANE_H
#define ENEMYPLANE_H
#include <QPixmap>

class EnemyPlane
{
public:
```

```

EnemyPlane();

//更新坐标
void updatePosition();
public:
//敌机资源对象
QPixmap m_enemy;

//位置
int m_X;
int m_Y;

//敌机的矩形边框（碰撞检测）
QRect m_Rect;

//状态
bool m_Free;

//速度
int m_Speed;
};

#endif // ENEMYPLANE_H

```

9.3 敌机成员函数实现

在config.h中追加敌机配置信息

```

***** 敌机配置数据 *****/
#define ENEMY_PATH ":/res/img-plane_5.png" //敌机资源图片
#define ENEMY_SPEED 5 //敌机移动速度
#define ENEMY_NUM 20 //敌机总数量
#define ENEMY_INTERVAL 30 //敌机出场时间间隔

```

在enemyPlane.cpp中实现成员函数，代码如下：

```

EnemyPlane::EnemyPlane()
{
    //敌机资源加载
    m_enemy.load(ENEMY_PATH);

    //敌机位置
    m_X = 0;

    m_Y = 0;
}

```

```

//敌机状态
m_Free = true;

//敌机速度
m_Speed = ENEMY_SPEED;

//敌机矩形
m_Rect.setWidth(m_enemy.width());
m_Rect.setHeight(m_enemy.height());
m_Rect.moveTo(m_X,m_Y);
}

void EnemyPlane::updatePosition()
{
    //空闲状态，不计算坐标
    if(m_Free)
    {
        return;
    }

    m_Y += m_Speed;
    m_Rect.moveTo(m_X,m_Y);

    if(m_Y >= GAME_HEIGHT)
    {
        m_Free = true;
    }
}

```

9.4 敌机出场

在MainScene.h中追加敌机出场的成员函数

在MainScene.h中追加敌机数组 和 敌机出场间隔记录 的成员属性

```

//敌机出场
void enemyToScene();

//敌机数组
EnemyPlane m_enemys[ENEMY_NUM];

//敌机出场间隔记录
int m_recorder;

```

初始化间隔记录属性,在MainScene.cpp的 initScene 成员函数中追加

```
m_recorder = 0;
```

实现成员函数 enemyToScene

```
void MainScene::enemyToScene()
{
    m_recorder++;
    if(m_recorder < ENEMY_INTERVAL)
    {
        return;
    }

    m_recorder = 0;

    for(int i = 0 ; i< ENEMY_NUM;i++)
    {
        if(m_enemys[i].m_Free)
        {
            //敌机空闲状态改为false
            m_enemys[i].m_Free = false;
            //设置坐标
            m_enemys[i].m_X = rand() % (GAME_WIDTH - m_enemys[i].m_Rect.width());
            m_enemys[i].m_Y = -m_enemys[i].m_Rect.height();
            break;
        }
    }
}
```

在PlayGame成员函数的timeout信号发送时候，槽函数中首先追加 enemyToScene

```
//敌机出场
enemyToScene();
```

```

void MainScene::playGame()
{
    //启动定时器
    m_Timer.start();

    //监听定时器
    connect(&m_Timer,&QTimer::timeout,[=](){
        //敌机出场
        enemyToScene();
        //更新游戏中元素的坐标
        updatePosition();
        //重新绘制图片
        update();
    });
}

```

更新敌机坐标，在updatePosition成员函数中追加代码

```

//敌机坐标计算
for(int i = 0 ; i< ENEMY_NUM;i++)
{
    //非空闲敌机 更新坐标
    if(m_enemys[i].m_Free == false)
    {
        m_enemys[i].updatePosition();
    }
}

```

绘制敌机，在paintEvent成员函数中追加绘制敌机代码

```

//绘制敌机
for(int i = 0 ; i< ENEMY_NUM;i++)
{
    if(m_enemys[i].m_Free == false)
    {
        painter.drawPixmap(m_enemys[i].m_X,m_enemys[i].m_Y,m_enemys[i].m_enemy);
    }
}

```

添加随机数种子

在MainScene.cpp中 initScene 成员函数里添加随机数种子

```
//随机数种子  
srand((unsigned int)time(NULL)); //头文件 #include <ctime>
```

运行测试敌机出场效果



10 碰撞检测

实现碰撞检测步骤如下：

- 添加并实现碰撞检测成员函数
- 调用并测试函数

10.1 添加并实现碰撞检测函数

在MainScene.h中添加新的成员函数

```
void collisionDetection();
```

在MainScene.cpp中实现该成员函数

```
void MainScene::collisionDetection()
{
    //遍历所有非空闲的敌机
    for(int i = 0 ; i < ENEMY_NUM;i++)
    {
        if(m_enemys[i].m_Free)
        {
            //空闲飞机 跳转下一次循环
            continue;
        }

        //遍历所有 非空闲的子弹
        for(int j = 0 ; j < BULLET_NUM;j++)
        {
            if(m_hero.m_bullets[j].m_Free)
            {
                //空闲子弹 跳转下一次循环
                continue;
            }

            //如果子弹矩形框和敌机矩形框相交，发生碰撞，同时变为空闲状态即可
            if(m_enemys[i].m_Rect.intersects(m_hero.m_bullets[j].m_Rect))
            {
                m_enemys[i].m_Free = true;
                m_hero.m_bullets[j].m_Free = true;
            }
        }
    }
}
```

10.2 调用并测试函数

在MainScene.cpp中 playGame成员函数里，追加碰撞检测代码

```
void MainScene::playGame()
{
    //启动定时器
    m_Timer.start();

    //监听定时器
    connect(&m_Timer,&QTimer::timeout,[=](){
        //敌机出场
        enemyToScene();
        //更新游戏中元素的坐标
        updatePosition();
        //重新绘制图片
        update();
        //碰撞检测
        collisionDetection();
    });
}
```

运行查看效果，子弹和敌机碰撞后会同时消失

11 爆炸效果

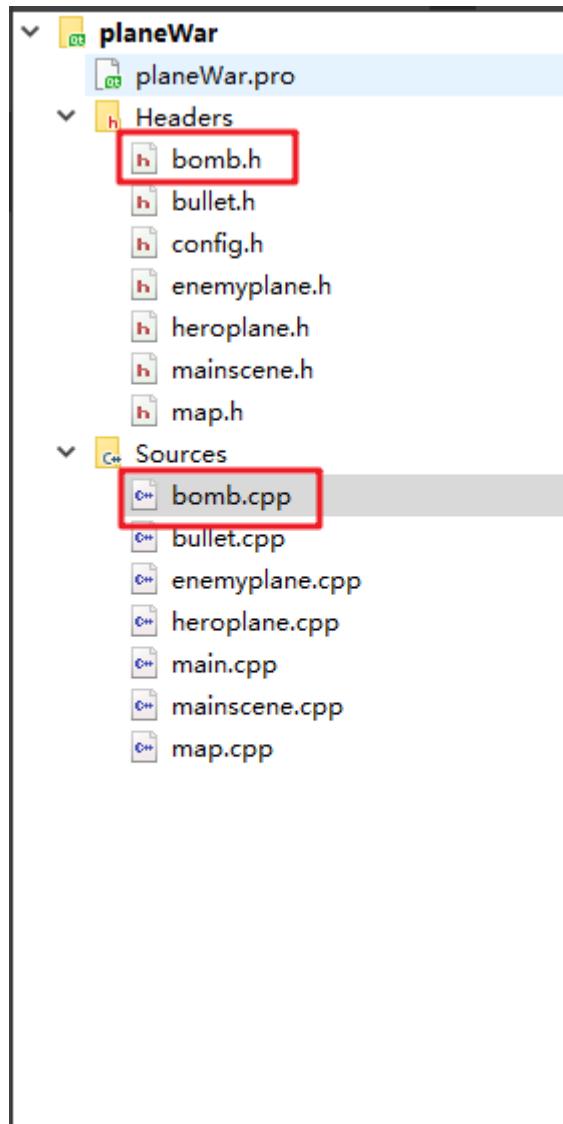
爆炸效果功能实现步骤如下：

- 创建爆炸文件和类
- 添加爆炸类中的成员函数和成员属性
- 实现成员函数
- 调用并测试效果

11.1 创建爆炸文件和类

创建Bomb类以及生成对应的文件

创建好后生成bomb.h 和 bomb.cpp两个文件



11.2 爆炸成员函数和成员属性

在config.h中加入爆炸配置数据

```
#define BOMB_PATH    ":/res/bomb-%1.png"      //爆炸资源图片
#define BOMB_NUM     20              //爆炸数量
#define BOMB_MAX      7               //爆炸图片最大索引
#define BOMB_INTERVAL 20             //爆炸切图时间间隔
```

在bomb.h中添加如下代码：

```
#ifndef BOMB_H
#define BOMB_H
#include "config.h"
#include <QPixmap>
#include <QVector>

class Bomb
```

```

{
public:
Bomb();

//更新信息（播放图片下标、播放间隔）
void updateInfo();

public:

//放爆炸资源数组
QVector<QPixmap> m_pixArr;

//爆炸位置
int m_X;
int m_Y;

//爆炸状态
bool m_Free;

//爆炸切图的时间间隔
int m_Recoedr;

//爆炸时加载的图片下标
int m_index;
};

#endif // BOMB_H

```

11.3 实现成员函数

```

Bomb::Bomb()
{
    //初始化爆炸图片数组
    for(int i = 1 ;i <= BOMB_MAX ;i++)
    {
        //字符串拼接，类似 ":/res/bomb-1.png"
        QString str = QString(BOMB_PATH).arg(i);
        m_pixArr.push_back(QPixmap(str));
    }

    //初始化坐标
    m_X = 0;
    m_Y = 0;

    //初始化空闲状态
    m_Free = true;

    //当前播放图片下标
    m_index = 0;
}

```

```

//爆炸间隔记录
m_Recober = 0;
}

void Bomb::updateInfo()
{
    //空闲状态
    if(m_Free)
    {
        return;
    }

    m_Recober++;
    if(m_Recober < BOMB_INTERVAL)
    {
        //记录爆炸间隔未到，直接return，不需要切图
        return;
    }

    //重置记录
    m_Recober = 0;

    //切换爆炸播放图片
    m_index++;
    //注：数组中的下标从0开始，最大是6
    //如果计算的下标大于6，重置为0
    if(m_index > BOMB_MAX-1)
    {
        m_index = 0;
        m_Free = true;
    }
}

```

11.4 加入爆炸数组

在MainScene.h中加入爆炸数组 成员属性

```

//爆炸数组
Bomb m_bombs[BOMB_NUM];

```

在碰撞检测成员函数中，当发生碰撞时，设置爆炸对象的信息

```

//播放爆炸效果
for(int k = 0 ; k < BOMB_NUM;k++)
{
    if(m_bombs[k].m_Free)
    {
        //爆炸状态设置为非空闲
        m_bombs[k].m_Free = false;
    }
}

```

```
//更新坐标

m_bombs[k].m_X = m_enemys[i].m_X;
m_bombs[k].m_Y = m_enemys[i].m_Y;
break;
}
}
```

在 MainScene.cpp 的 **updatePosition** 中追加代码

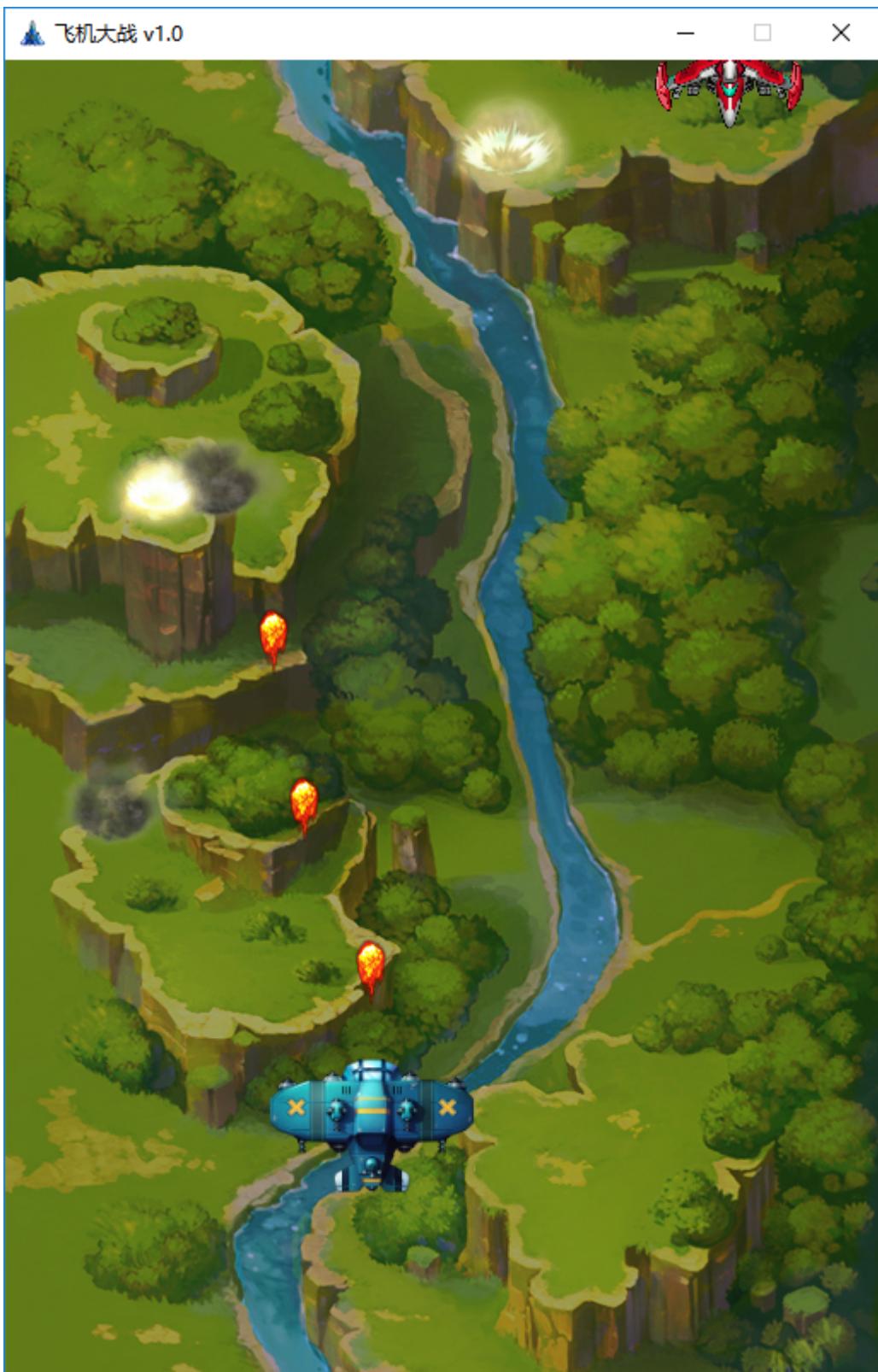
```
//计算爆炸播放的图片
for(int i = 0 ; i < BOMB_NUM;i++)
{
    if(m_bombs[i].m_Free == false)
    {
        m_bombs[i].updateInfo();
    }
}
```

在 MainScene.cpp 的 **paintEvent** 中追加绘制爆炸代码

```
//绘制爆炸图片
for(int i = 0 ; i < BOMB_NUM;i++)
{
    if(m_bombs[i].m_Free == false)
    {

painter.drawPixmap(m_bombs[i].m_X,m_bombs[i].m_Y,m_bombs[i].m_pixArr[m_bombs[i].m_index]);
    }
}
```

测试，实现爆炸效果



12 音效添加

音效添加步骤如下：

- 添加多媒体模块
- 播放音效

12.1 添加多媒体模块

在工程文件planeWar.pro 中修改代码

```
QT += core gui multimedia
```

```
QT += core gui multimedia
```

12.2 播放音效

在config.h中 添加音效的配置路径

```
#define SOUND_BACKGROUND ":/res/bg.wav"  
#define SOUND_BOMB ":/res/bomb.wav"
```

注：QSound使用时候要包含头文件 #include <QSound>

在PlayGame中添加背景音乐

```
//启动背景音乐  
QSound::play(SOUND_BACKGROUND);
```

在爆炸时候添加爆炸音效

```
//播放音效  
QSound::play(SOUND_BOMB);
```

测试音效

13 打包发布

1. 确定环境变量配置好 PATH: `C:\Qt\Qt5.x.x\5.x.x\mingw53_32\bin`
2. 在QT中把运行模式切换成 release 模式，编译。在外层目录中会有 release 版本的目录。
3. 将目录中的 rcc 二进制资源文件、可执行程序文件(`.exe`)拷贝到另外一个单独的文件夹中。
4. 进入 cmd 命令模式，切换到可执行程序所在的目录。执行以下命令，将可执行程序所需的库文件拷贝到当前目录：

```
windeployqt PlaneWar.exe
```

5. 额外可以将 ico 图标也拷贝到当前可执行程序所在的目录。
6. 启动 HM NIS EDIT 软件，在软件中选择：文件->新建脚本向导，接下来跟着向导操作。
7. 为了让安装包安装软件也有快捷方式图标，在生成的脚本里。进行修改：

```
CreateShortCut "$DESKTOP\飞机大战.lnk" "$INstdir\PlaneWar.exe"  
CreateShortCut "$DESKTOP\飞机大战.lnk" "$INstdir\PlaneWar.exe" "" "$INstdir\app.ico"
```

8. 点击菜单栏的 `NSIS`，然后选择编译，在桌面生成安装包。

至此本案例实现完毕！

Qt以及打包软件的 安装包

链接：<https://pan.baidu.com/s/19Vcv1oAIDk2Ool-9bWfDg> 提取码：6kmf