1 项目简介

需要实现的内容：

1. 滚动的背景地图
2. 飞机的制作和控制
3. 子弹的制作和射击
4. 敌机的制作
5. 碰撞检测
6. 爆炸效果
7. 音效添加

2 创建项目

基类选择QWidget空窗口

第一个场景为主场景MainScene

不带UI界面

3 设置主场景

步骤：

- 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

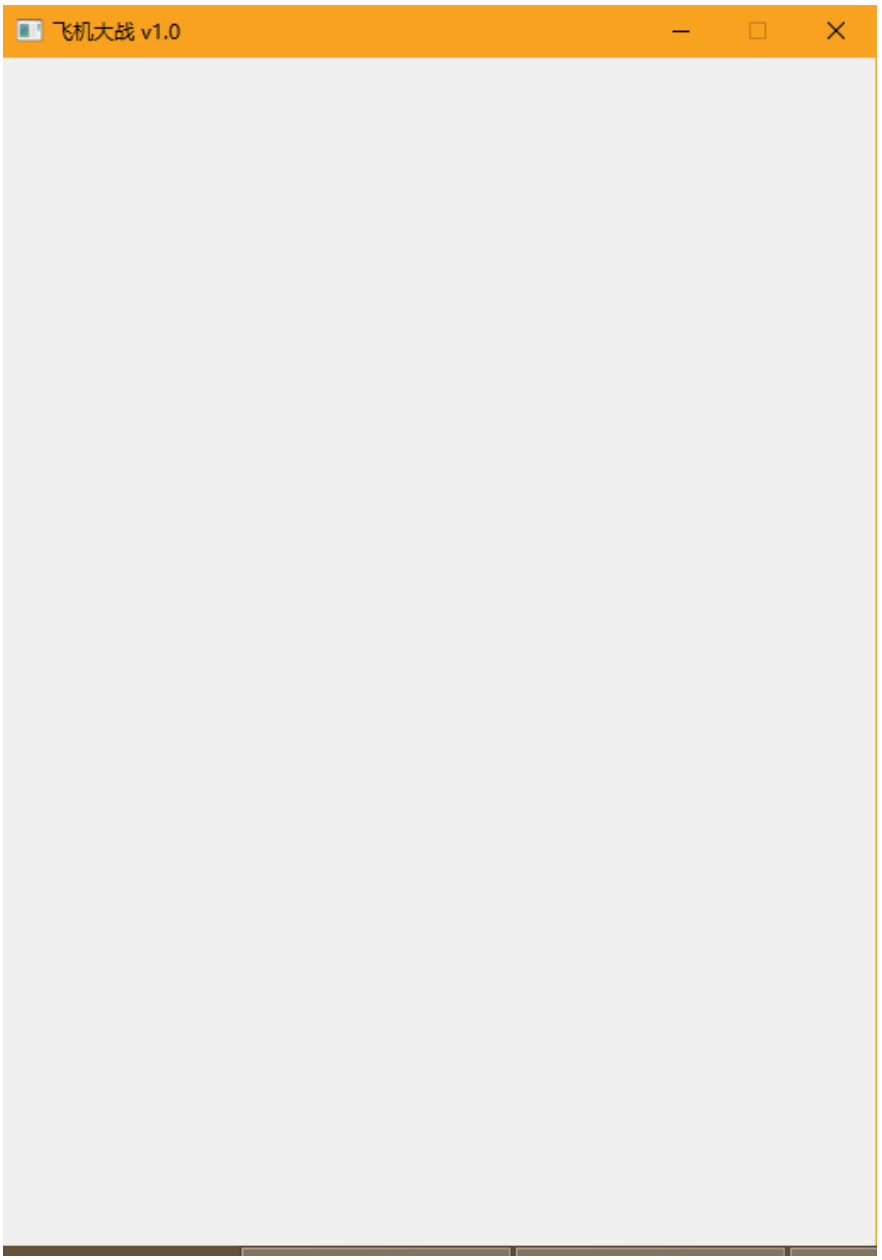
添加窗口宽度、高度的配置信息，依据背景图大小进行设置。

3.2 主场景基本设置

```
//config.h//

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

```
//mainsource.h//

#ifndef MAINSCENCE_H
#define MAINSCENCE_H

#include <QWidget>

class MainScence : public QWidget
{
    Q_OBJECT

public:
    MainScence(QWidget *parent = nullptr);
    ~MainScence();
    void initScene(); //添加新的成员函数来初始化游戏场景
};
#endif // MAINSCENCE_H

//mainscence.cpp//

#include "mainscence.h"
#include "config.h"

void MainScence::initScene()
{
    setFixedSize(GAME_WIDTH,GAME_HEIGH); //初始化窗口大小
    setWindowTitle(GAME_TITLE); //设置窗口标题
}

// 构造函数
MainScence::MainScence(QWidget *parent)
    : QWidget(parent)
{
    initScene(); //调用initScene()初始化场景
}

MainScence::~MainSource()
{
}
```

运行效果



4 资源导入

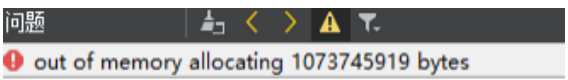
将游戏中的资源进行导入并设置游戏图标

资源导入步骤：

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

4.4 qrc生成rcc二进制文件（注：未配置好，rcc文件生成不了）

如果资源过大，则会提示错误：“编译器的堆空间不足”。



解决方法：利用二进制资源

步骤：

利用cmd打开终端，定位到res.qrc目录下，输入以下命令：

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

4.7 添加图标资源

配置文件config.h中追加代码

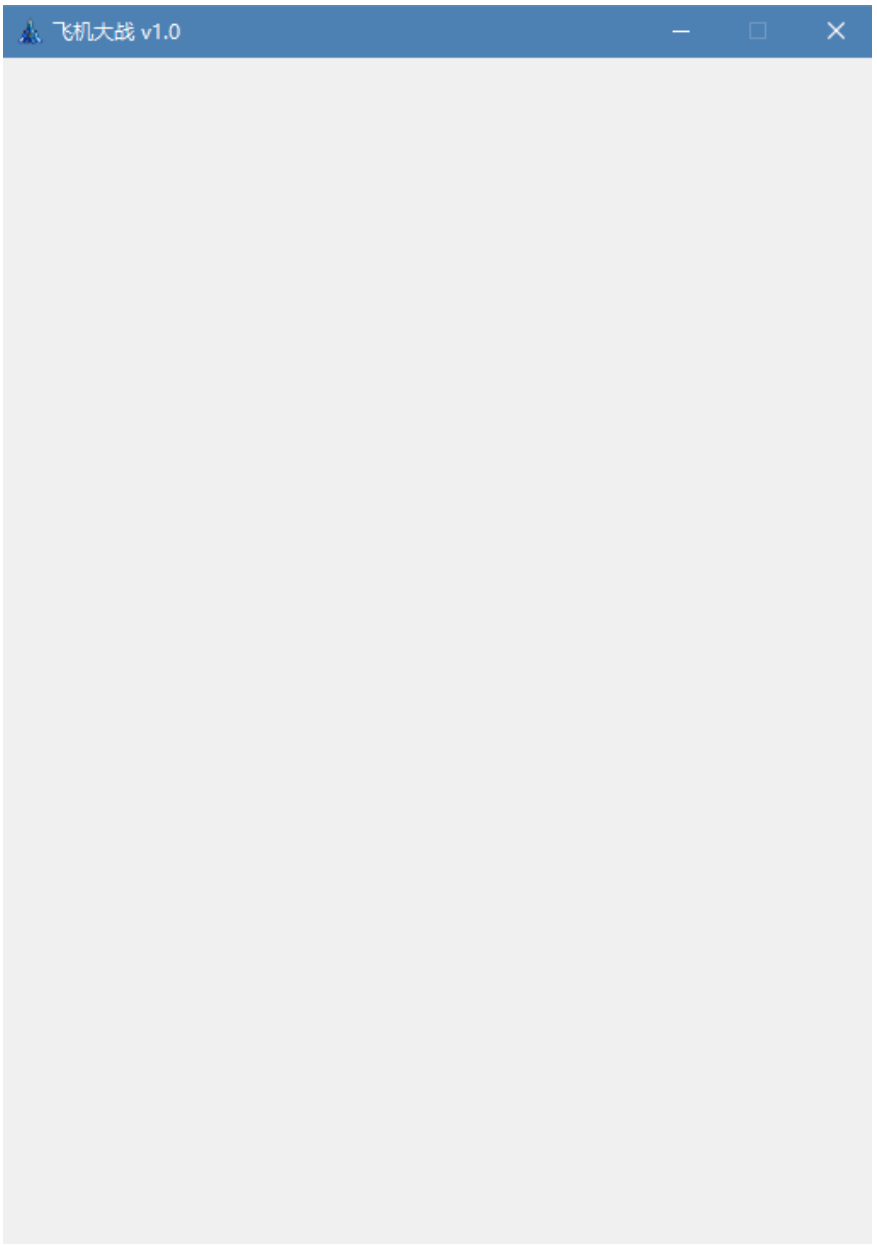
虚拟资源路径语法：":+前缀名+文件路径"

```
#define GAME_ICON ":/image/res/app.ico" //添加图标资源
```

在mainScene.cpp的 initScene函数中追加代码：

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

运行结果



5 地图滚动

步骤：

1. 创建地图文件和类

2. 添加成员函数和成员属性，实现成员函数
3. 游戏运行启动定时器
4. 启动定时器，监听定时器信号，实现游戏循环
5. 计算游戏内元素坐标
6. 绘制到屏幕中

5.1 创建地图文件和类（即map.cpp和map.h）

5.2 地图的成员函数和成员属性

在map.h中添加代码

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

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

    void mapPosition(); //地图滚动坐标计算

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 ":/image/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);

    //设置地图对象Y轴坐标
    m_map1_posY = -GAME_HEIGH;
    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_HEIGH;
    }
    //处理第二张图片滚动
    m_map2_posY += MAP_SCROLL_SPEED;
    if(m_map2_posY >=GAME_HEIGH)
    {
        m_map2_posY = 0;
    }
}
```

5.4 定时器添加

在mainScene.h中添加新的定时器对象

```
#include <QTimer>
QTimer m_Timer; //添加新的定时器对象
```

在 config.h中添加 屏幕刷新间隔

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

在MainScene.cpp的initScene中追加代码

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

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

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

Map m_map; //地图对象
```

在MainScene.cpp中实现成员函数

```
#include <QTimer>
#include <QPainter>
void MainScence::playGame()
{
    m_Timer.start();//启动定时器

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

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

void MainScence::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);
}
```

实现地图滚动需要在构造函数MinScence中调用playGame()函数，启动定时器

6 英雄飞机

步骤：

1. 创建英雄文件和类
2. 添加成员函数和成员属性
3. 实现成员函数
4. 创建飞机对象并显示
5. 拖曳飞机

6.1 创建英雄文件和类（HeroPlane.cpp和HeroPlane.h）

6.2 飞机的成员函数和成员属性

在HeroPlane.h中添加代码

```
#ifndef HEROPLANE_H
#define HEROPLANE_H
#include <QPixmap>
#include <QRect>


class HeroPlane
{
public:
    HeroPlane();

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

public:
    QPixmap m_Plane;//飞机资源 对象

    //飞机坐标
    int m_X;
    int m_Y;

    QRect m_Rect;//飞机的矩形边框
};

#endif // HEROPLANE_H
```

6.3 成员函数实现

由于飞机有一个发射子弹的成员函数，但子弹对象还没有创建，因此现将子弹成员函数写为空。

在config.h中追加飞机配置参数

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

heroPlane.cpp中实现成员函数代码：

```
#include "heroplane.h"
#include "config.h"

HeroPlane::HeroPlane()
{
    m_Plane.load(HERO_PATH);//初始化加载飞机图片信息
```

```
    //初始化坐标
    m_X = GAME_WIDTH*0.5-m_Plane.width()*0.5;
    m_Y = GAME_HEIGH-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

```
#include "heroplane.h"
HeroPlane m_hero;//飞机对象
```

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);
```

运行测试

6.5 拖曳飞机

在MainScene.h中添加鼠标移动事件

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

在MainScence.cpp中重写鼠标移动事件

```
void MainScence::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_HEIGH-m_hero.m_Rect.height()){
        x=GAME_HEIGH-m_hero.m_Rect.height();
    }
    m_hero.setPosition(x,y);
}
```

运行测试飞机可拖曳

7 子弹制作

步骤：

1. 创建子弹文件和类
2. 添加子弹类中的成员函数和成员属性
3. 实现成员函数

4. 测试子弹

7.1 创建子弹文件和类（bullet.h和bullet.cpp）

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

public:
    QPixmap m_Bullet;//子弹资源对象

    //子弹坐标
    int m_X;
    int m_Y;

    int m_Speed;//子弹移动速度
    bool m_Free;//子弹是否闲置
    QRect m_Rect;//子弹的矩形边距（用于碰撞检测）
};

#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 ":/image/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

```
#include "bullet.h"

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

    m_Free = true;//子弹状态
    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，则不需要坐标计算
    //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

子弹由飞机发射，测试阶段添加一段辅助代码，测试后，删掉代码即可

在MainScene.h中添加测试代码

```
#include "bullet.h"
//测试子弹，添加子弹对象
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

```
//绘制子弹
    painter.drawPixmap(temp_bullet.m_X,temp_bullet.m_Y,temp_bullet.m_Bullet);
```

开始游戏

```
MainScence::MainScence(QWidget *parent)
```

```
    : QWidget(parent)
{
    initScene(); //调用initScene()初始化场景
    playGame();
}
```

测试完毕后，删除程序

8 玩家发射子弹

步骤：

1. 英雄飞机添加新的成员属性
2. 实现发射成员函数
3. 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：

```
#include "bullet.h"
Bullet m_bullets[BULLET_NUM]; //弹匣
int m_recorder; //发射时间记录
```

8.2 成员函数补充

在HeroPlane.cpp的构造函数 HeroPlane() 中初始化发生间隔记录

```
m_recorder = 0; //初始化发射间隔记录
```

补充之前在HeroPlane.cpp中预留的一个shoot函数：

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

    //发射子弹
    for(int i=0;i<BULLET_NUM;i++){
        //如果是空闲子弹则进行发射
        if(m_bullets[i].m_Free){
            m_bullets->m_Free = false; //将子弹的空闲状态改为假
            //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

```
m_hero.shoot(); //发射子弹
    //计算子弹坐标
    for(int i = 0;i < BULLET_NUM;i++){
        //如果子弹的状态为非空闲，则计算发射位置
        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);
        }
    }
```

添加playGame()函数测试运行，可以发射子弹

9 敌机制作

与子弹制作原理类似，即每隔一定时间让敌机出现

步骤

1. 添加敌机文件和类
2. 添加敌机类中的成员函数和成员属性
3. 实现成员函数
4. 敌机出场
5. 测试敌机

9.1 创建敌机文件和类（enemyPlane.h和enemyPlane.cpp）

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>
#include <QRect>
class EnemyPlane
```

```
{
public:
    EnemyPlane();

    void updatePosition(); //更新坐标

public:
    QPixmap m_enemy; //敌机资源对象
    //位置
    int m_X;
    int m_Y;

    QRect m_Rect; //敌机的矩形边框（碰撞检测）
    bool m_Free; //状态
    int m_Speed; //速度
};

#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 ":/image/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

```
#include "enemyplane.h"
#include "config.h"

EnemyPlane::EnemyPlane()
{
    m_enemy.load(ENEMY_PATH); //敌机资源加载
    // 敌机位置
    m_X = 0;
    m_Y = 0;

    m_Free = true; //敌机状态
    m_Speed = ENEMY_SPEED; //敌机速度
    //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()
{
    //空闲状态则不计算坐标
    if(m_Free){
        return;
    }

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

    if(m_Y >= GAME_HEIGH + m_Rect.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

```
    void enemyToScene();//敌机出场
    EnemyPlane m_enemys[ENEMY_NUM];//敌机数量
    int m_recorder;//敌机出场间隔记录
```

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

```
m_recorder = 0;//初始化敌机出场时间间隔
```

实现成员函数 enemyToScene

```
void MainScence::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){
            m_enemys[i].m_Free = false;//敌机空闲状态改为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

```
enemyToScene();//敌机出场
```

```
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

```
//计算敌机坐标
    for(int i = 0;i<ENEMY_NUM;i++){
        //非空闲敌机，则更新坐标
        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);
        }
    }
```

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

```
srand((unsigned int)time(NULL));//随机数种子
```

10 碰撞检测

步骤：

1. 添加并实现碰撞检测成员函数
2. 调用并测试成员函数

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();//碰撞事件
```

在MainScene.cpp中实现该成员函数

```
void MainScence::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){
            continue;//将飞机空闲，跳转至下一循环
        }
        //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){
                continue;//将子弹空闲，跳转至下一次循环
            }
            //如果子弹的矩形框与敌机的矩形框相交，则发生碰撞，将敌机与子弹同时变为空闲状态
            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();
    });
}
```

运行检测，子弹与敌机接触后会自动消失。

11 爆炸效果

【存在问题：运行提示快速异常检测失败，未查找出原因，将与爆炸有关的程序删除后正常】

步骤：

1. 创建爆炸文件和类

2. 添加爆炸类中的成员函数和成员属性
3. 实现成员函数
4. 调用并测试结果

11.1 创建爆炸文件和类（Bomb.h和Bomb.cpp）

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

```
/*********爆炸配置数据**********/
#define BOMB_PATH ":/image/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();

    void updateInfo(); //更新信息*播放图片下标、播放间隔
public:
    QVector<QPixmap> m_pixArr; //放爆炸资源数组

    //爆炸位置
    int m_X;
    int m_Y;

    bool m_Free; //爆炸状态
    int m_Recoder; //爆炸切图的时间间隔
    int m_index; //爆炸时加载的图片下标
};

#endif // BOMB_H
```

11.3 实现成员函数

在bomb.cpp中写入如下代码：

```
#include "bomb.h"
```

```
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":/image/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;

    m_Free=true; //初始化空闲状态
    m_index=0; //当前播放图片下标
    m_Recoder=0; //爆炸间隔记录
}

void Bomb::updateInfo()
{
    //空闲状态
    if(m_Free){
        return;
    }
    m_Recoder++;
    if(m_Recoder<BOMB_INTERVAL){
        return; //记录爆炸间隔，若未到，直接return，不需要切图
    }
    m_Recoder=0; //重置记录

    m_index++; //切换爆炸播放图片
    //若计算的下标大于6，重置为0（注：数组中的下标从0开始，最大是6）
    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

```
Bomb m_bombs[BOMB_NUM];//爆炸数组
```

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（mainscence.cpp中）

```
//播放爆炸效果
```

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

```
//绘制操作图片
    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]);
        }
    }
```

运行测试实现爆炸效果

12 音效添加

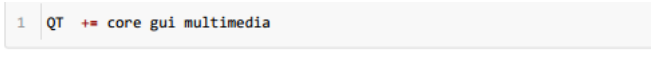
步骤：

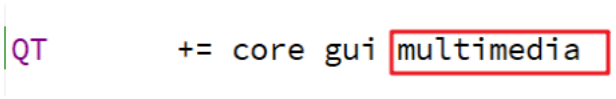
- 添加多媒体模块。
- 播放音效

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
```





12.2 播放音效

在config.h中 添加音效的配置路径

```
/*********音效配置路径**********/
#define SOUND_BACKGROUND ":/image/res/bg.wav"
#define SOUND_BOMB ":/image/res/bomb.wav"
```

在MainScence.cpp的PlayGame中添加背景音乐。

注：QSound使用时需要添加头文件#include <QtMultimedia/QSound>

```
QSound::play(SOUND_BACKGROUND); //启动背景音乐
QSound::play(SOUND_BOMB); //播放音效
```

测试音效

13 打包发布 【也有问题！！！cmd】

1. 配置好环境变量PATH：E:\Qt\Qt_install\5.14.2\mingw73_64\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：`c 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：`c 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.