实验五 飞机大战

1.程序介绍

该程序为名称为飞机大战。作为早期最经典游戏之一的飞机大战，它是一种传统的电脑游戏，经常出现在便携式终端、手机和计算机中。本游戏是一个基于面向对象编程思想,选用 Qt 框架和 c++语言来实现,PC 端的一款飞机大战游戏。该游戏操作灵活简单，趣味性较强，玩家可以通过鼠标进行游戏操作，通过移动躲避敌方飞机子弹，并发射子弹消灭敌方飞机，从而获取积分。游戏主要涉及了实现飞机移动、发射子弹、碰撞检测、敌方飞机、事件监听,刷新游戏画面及音乐,记录游戏分数和使用者的信息，使游戏界面更多样化，展现出游戏的整体的开发创新流程和设计想法。

2.操作说明

2.1 游戏开始界面

游戏以开始背景为起点，显示出游戏的背景和我方飞机状态，在点击开始游戏后即可进入游戏。有游戏规则提示窗口,选择是即可进入游戏。效果如图所示:

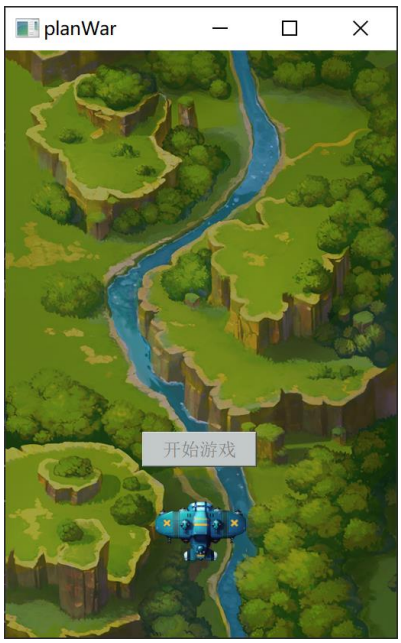

图 1 开始页面

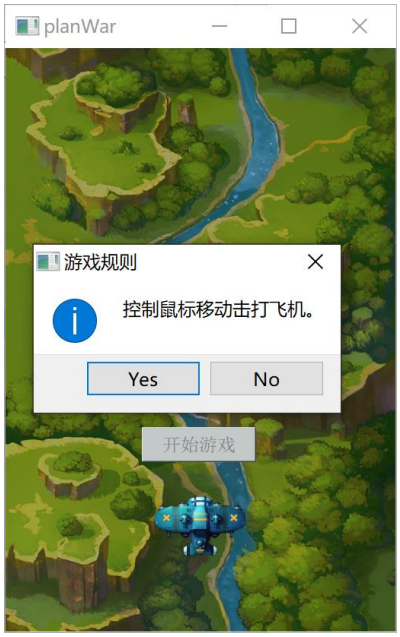

图 2 确认开始页面

2.2 游戏状态展示

这里以模拟游戏进行时为操作，展示游戏的操作性，效果如图所示：

图 3 游戏状态

2.3 游戏结束展示

如果触碰到敌机，游戏结束。显示得分情况。

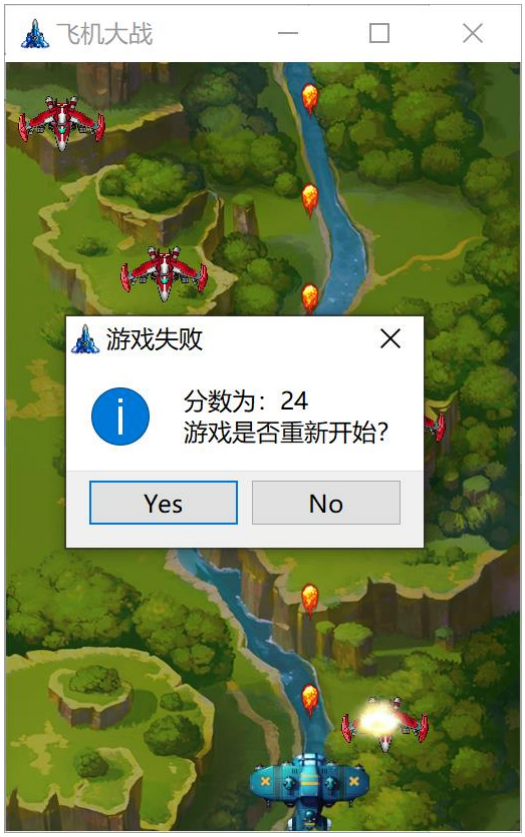

图 4 游戏结束

3.设计理念

3.1 设计目标

该项目是想设计一个经典的游戏飞机大战,选用 Qt 框架和 c++语言来实现。期望游戏操作灵活简单，趣味性较强，玩家可以通过鼠标进行游戏操作，通过移动躲避敌方飞机子弹，并发射子弹消灭敌方飞机，从而获取积分。游戏主要涉及了实现飞机移动、发射子弹、碰撞检测、敌方飞机、事件监听,刷新游戏画面及音乐,记录游戏分数和使用者的信息。

在整个游戏中，我们看到的所有内容，我们都可以理解为游戏对象，每一个游戏对象,都由一个单独的类来创建;在游戏中主要游戏对象:我方飞机，子弹，背景，敌方飞机，爆炸效果。

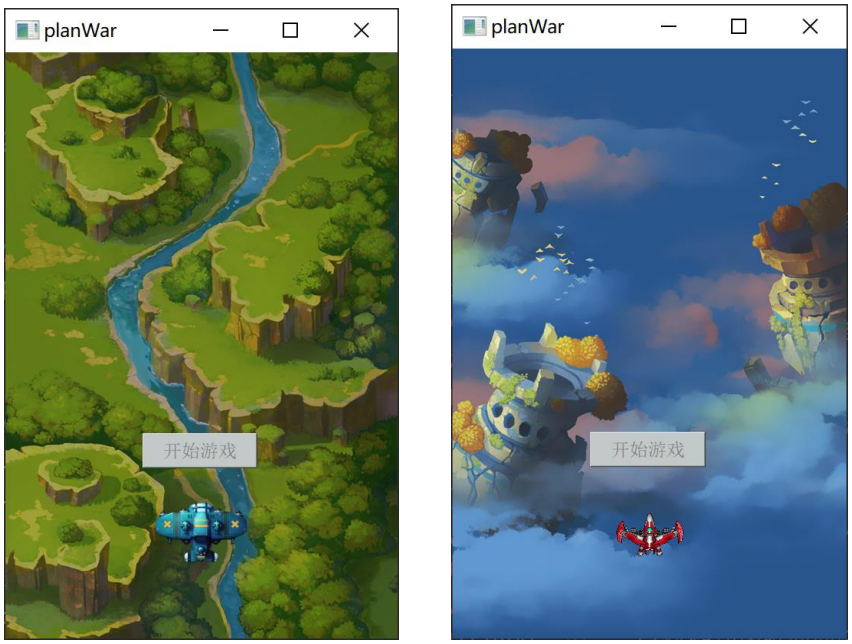

图 5 游戏页面

3.2 设计分析和算法分析

3.2.1 资源导入

游戏所需要的资源较多，这里我选用网上的素材包，进行一次性导入，因为资源内存较大，qrc 文件无法一次通过，这里 采用 rcc 注册二进制文件的方法导入资源。资源文件如图所示，步骤如下：

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

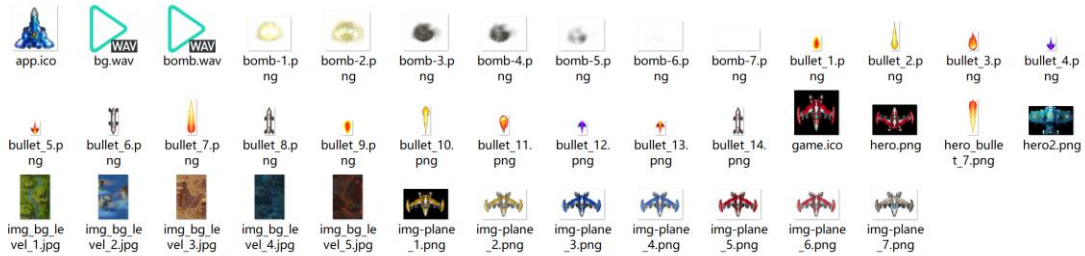

图 6 资源文件

在导入资源生成 rcc 文件后在 main 函数中用 QResource 注册外部的二进制资源文件。代码如下：

```
#include "mainscreen.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);

    MainScreen w;
    w.show();

    return a.exec();
}
```

代码块 1 main 代码

为了方便管理资源参数以及改变游戏背景，资源属性，这里参考网上游戏开发教程，单独生成资源管理头文件。Config 头文件，全部代码如下：

```
#ifndef CONFIG_H
#define CONFIG_H
//配置文件

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

```
/********** 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" //标题

#define GAME_RES_PATH "./plane.rcc" //二进制资源路径
#define GAME_ICON ":/res/app.ico"  //图标路径
#define GAME_RATE  10               //单位毫秒

/********** 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

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

/********** 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

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

/********** 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

/********** 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

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

#endif // CONFIG_H
```

代码块 2 Config 头文件

3.2.2 地图类的构建

这里用来控制背景图像，采用一个俩个图片拼接，每次刷新，向下移动实现滚动效果，这里难点是控制地图滚动幅度和游戏刷新逻辑保持一致。头文件及实现代码如下：

图 7 背景图资源

```
#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
```

代码块 3 Map 头文件

```
#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);

    //设置坐标
    m_map1_posY = -GAME_HEIGHT;
    m_map2_posY = 0;

    //设置滚动速度
    m_scroll_speed = MAP_SCROLL_SPEED;
}

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

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

```
    }

}
```

代码块 4 Map 实现代码

3.2.3 我方飞机构建

这里是创建一个我方飞机类，有坐标 x，y 及边界参数，顾应该有一个移动的操作函数,用来和鼠标移动所配合,因为这里要实现打飞机,后面引入子弹类,用于射击。相关 HeroPlane 类的头文件及实现代码如下:

图 8 我方飞机展示

```
#ifndef HEROPLANE_H
#define HEROPLANE_H
#include <QPixmap>
#include "bullet.h"

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;

    //弹匣
    Bullet m_bullets[BULLET_NUM];

    //发射间隔记录
    int m_recorder;
};
#endif // HEROPLANE_H
```

代码块 5 HeroPlane 头文件

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

HeroPlane::HeroPlane()
{
    //加载飞机图片资源
    m_Plane.load(HERO_PATH);

    //初始化坐标(通过计算飞机在屏幕坐标得)
    m_X = GAME_WIDTH * 0.5 - m_Plane.width() *0.5;
    m_Y = GAME_HEIGHT - m_Plane.height() - 100;

    //矩形边框 碰撞检测用
    m_Rect.setWidth(m_Plane.width());
    m_Rect.setHeight(m_Plane.height());
    m_Rect.moveTo(m_X,m_Y);

    //初始化间隔记录变量
    m_recorder = 0;
}

//设置飞机位置
void HeroPlane::setPosition(int x, int y)
{
    m_X = x;
    m_Y = y;
    m_Rect.moveTo(m_X,m_Y);
}

//发射子弹
void HeroPlane::shoot()
```

```
{
    //累加事件间隔记录的变量
    m_recorder++;


    //如果记录数字 未达到发射间隔，直接 return
    if(m_recorder < BULLET_INTERVAL)
    {
        return;
    }

    m_recorder = 0;

    //发射子弹
    for(int i = 0 ; i < BULLET_NUM;i++)
    {
        //如果是空闲状态的子弹，发射子弹
        if(m_bullets[i].m_Free)
        {
            m_bullets[i].m_Free = false;
            m_bullets[i].m_X = m_X+m_Rect.width()*0.5 - 10;
            m_bullets[i].m_Y = m_Y - 25 ;
            break;
        }

    }
}
```

代码块 6 HeroPlane 实现代码

3.2.4 子弹构建

这里为了实现射击函数，创建子弹类，子弹和我方飞机类似，拥有坐标 x，y 及边界，图片资源，为了进一步控制这里采用控制子弹移动速度和状态，状态来判断碰撞后是否存在，来实现游戏的基础逻辑。Bullet 类的头文件及实现代码如下：

图 9 子弹资源展示

```
#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
```

代码块 7 Bullet 头文件

```
#include "bullet.h"

Bullet::Bullet()
{
    //加载自动资源
    m_Bullet.load(BULLET_PATH);

    //子弹坐标初始化
    m_X = GAME_WIDTH *0.5 - m_Bullet.width() * 0.5;
    m_Y = GAME_HEIGHT;
```

```
    //子弹空闲状态
    m_Free = true;

    //子弹速度
    m_Speed = BULLET_SPEED;

    //子弹矩形边框 （碰撞检测）
    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，不需要计算坐标
    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;
    }

}
```

代码块 8 Bullet 实现函数

3.2.5 敌方飞机构建

这里类似子弹类，拥有坐标 x，y 及边界，图片资源，为了进一步控制这里采用控制敌方飞机移动速度和状态，状态来判断碰撞后是否存在，来实现游戏的基础逻辑。EnemyPlane 类头文件及代码如下：

图 10 敌方飞机资源展示

```
#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
```

代码块 9 EnemyPlane 头文件

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

```
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;
    }
}
```

代码块 10 EnemyPlane 实现代码

3.2.6 爆炸类构建

爆炸类为当飞机与敌机碰撞（边界碰撞），或子弹与敌机碰撞（边界碰撞）时，在这个位置产生爆炸效果及爆炸图片展示。其和我方飞机构建原理类似。Bomb 类头文件及实现代码如下：

图 11 爆炸资源展示

```
#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_Recoder;
```

```
    //爆炸时加载的图片下标
    int m_index;
};

#endif // BOMB_H
```

代码块 11 Bomb 头文件

```
#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 ":/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_Recoder = 0;
}

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

    m_Recoder++;
```

```
    if(m_Recoder < BOMB_INTERVAL)
    {
        //记录爆炸间隔未到，直接 return，不需要切图
        return;
    }
    //重置记录
    m_Recoder = 0;

    //切换爆炸播放图片
    m_index++;

    //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;
    }
}
```

代码块 12 Bomb 实现代码

3.2.7 游戏整体构建

这里 MainScreen 继承所有构建类，在 MainScreen 类中实现相关算法以及游戏逻辑，这里首先是音乐的添加，游戏的初始化，控制游戏进程采用定时器对象 QTimer 类来实现，这里应该包含敌机随机产生，子弹的发射，paintEvent 的事件刷新时，各单位的移动操作，以及碰撞检测（及边界检测）和游戏结束函数和重新开始函数。下面展示游戏主体逻辑 MainScreen 的头文件：

```
#ifndef MAINSCREEN_H
#define MAINSCREEN_H

#include <QWidget>
#include "config.h"
#include "map.h"
#include "heroplane.h"
#include "bullet.h"
#include "enemyplane.h"
#include "bomb.h"
#include "start.h"
#include <QIcon>
```

```
#include <QTimer>
#include <QPainter>
#include <QEvent>
#include <QMouseEvent>
#include <QSound>
#include <ctime>
#include <QMessageBox>
#include <QPushButton>

class MainScreen : public QWidget
{
    Q_OBJECT

public:
    MainScreen(QWidget *parent = 0);
    ~MainScreen();

    //初始化场景
    void initScene();

    //游戏重新开始
    void slotStart();

    //初始化显示
    void initialDisplay();
    //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);

    //重写鼠标移动事件
    void mouseMoveEvent(QMouseEvent *);

    //敌机出场
    void enemyToScene();

    //碰撞检测
    void collisionDetection();
```

```
    //地图对象
    Map m_map;

    //创建飞机对象
    HeroPlane m_hero;


    //定时器对象
    QTimer m_Timer;


    //敌机数组
    EnemyPlane m_enemys[ENEMY_NUM];

    //敌机出场间隔记录
    int m_recorder;

    //爆炸数组
    Bomb m_bombs[BOMB_NUM];


    //得分计算
    int score;

    int Inumber;

    Start ui;
    //测试子弹
    //Bullet temp_Bullet;
};

#endif // MAINSCREEN_H
```

代码块 13 MainScreen 头文件

3.2.8 背景音乐

这里背景音乐采用 Qt 框架的 QSound 类，实现游戏背景音乐的循环播放。代码如下：

```
    //启动背景音乐(循环播放)
    //QSound::play(SOUND_BACKGROUND);
    QSound *sound = new QSound(SOUND_BACKGROUND, this);
```

```
    sound->setLoops(-1);
    sound->play();
```

代码块 14 背景音乐代码

3.2.9 开始按钮构建

这里在原背景暂停，构建一个开始游戏按钮，在构造函数时调用，设置槽函数搭建初始化连接，代码如下：

```
MainScreen::MainScreen(QWidget *parent)
    : QWidget(parent)
{

    //启动背景音乐(循环播放)
    //QSound::play(SOUND_BACKGROUND);
    QSound *sound = new QSound(SOUND_BACKGROUND, this);
    sound->setLoops(-1);
    sound->play();



    QPushButton *button=new QPushButton(this);
    button->setText("开始游戏");
    button->setStyleSheet("background:#C3C8C9;color:gray");
    resize(GAME_WIDTH,GAME_HEIGHT);
    button->move(width()*0.35,height()*0.65);

    Inumber = 0;
    score = 0;

    connect(button,&QPushButton::clicked,[=](){

        QMessageBox::StandardButton rb = QMessageBox::information(this,"游戏规则","控制鼠标移动击打飞机。",QMessageBox::Yes | QMessageBox::No, QMessageBox::Yes);
        if(rb == QMessageBox::Yes)
        {
            initScene();
        }
        else if (rb == QMessageBox::No)
        {
            exit(1);
        }
```

```
        button->hide();
        //this->hide();
        });
}
```

代码块 15 开始按钮

3.2.10 初始化函数

设置页面大小（与背景图片大小保持一致），调用游戏开始接口，实现游戏逻辑基本框架，随机数的刷新,，为了方便后面死亡后二次调用初始化，做出微量调整，代码如下：

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

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

    //设置窗口标题
    setWindowIcon(QIcon(GAME_ICON));

    //设置定时器间隔
    //m_Timer.setSingleShot(true);
    m_Timer.setInterval(GAME_RATE);

    //QMessageBox::about(this,"游戏规则","鼠标左键移动");


    //清屏
    //system ("clear");
    //initialDisplay();
    //调用启动游戏接口
    playGame();

    //敌机出场纪录变量 初始化
    m_recorder = 0;

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

```
}

void MainScreen::slotStart()
{
    //QMessageBox::about(this,"游戏规则","鼠标左键移动");


    //设置定时器间隔
    m_Timer.setInterval(GAME_RATE);
    //initialDisplay();

    //system ("clear");
    initialDisplay();

    playGame();

    //敌机出场纪录变量 初始化
    m_recorder = 0;

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

代码块 16 初始化函数

3.2.11 刷新展示

这里实现主见面的刷新主体，绘制飞机，绘制子弹，绘制敌机，绘制爆炸图片，生成随机数种子，敌机出场，更新游戏中所有元素的坐标，把游戏中的元素绘制到屏幕中。代码如下：

```
void MainScreen::initialDisplay()
{

    //绘制飞机
    m_hero.m_X = GAME_WIDTH * 0.5 - m_hero.m_Plane.width() *0.5;
    m_hero.m_Y = GAME_HEIGHT - m_hero.m_Plane.height() - 100;


    //绘制子弹

    for(int i = 0 ;i < BULLET_NUM;i++)
```

```
    {
        m_hero.m_bullets[i].m_X = GAME_WIDTH *0.5 - m_hero.m_bullets[i].m_Bullet.width()
* 0.5;
        m_hero.m_bullets[i].m_Y = GAME_HEIGHT;
        m_hero.m_bullets[i].m_Free == true;
    }



    //绘制敌机
    for(int i = 0 ; i< ENEMY_NUM;i++)
    {
        m_enemys[i].m_X = 0;
        m_enemys[i].m_Y = 0;

        m_enemys[i].m_Free == true;
    }


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


    m_recorder = 0;

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

    m_Timer.setTimerType(Qt::CoarseTimer);
    m_Timer.setInterval(GAME_RATE + Inumber);

    //敌机出场
    enemyToScene();
    //更新游戏中所有元素的坐标
    updatePosition();
    //游戏中的元素 绘制到屏幕中
    update();

}
```

代码块 17 刷新函数

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

    //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 == false)
        {
            m_hero.m_bullets[i].updatePosition();
        }

    }

    //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();
```

```
        }
    }

    //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();
        }
    }

    //测试子弹
//    temp_Bullet.m_Free = false;
//    temp_Bullet.updatePosition();
}

void MainScreen::paintEvent(QPaintEvent *event)
{
    //利用画家画图图片
    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);

    //绘制飞机
    painter.drawPixmap(m_hero.m_X,m_hero.m_Y,m_hero.m_Plane);

    //绘制子弹
    for(int i = 0 ;i < BULLET_NUM;i++)
    {
```

```
        //如果子弹状态为非空闲 ， 绘制图片
        if(m_hero.m_bullets[i].m_Free == false)
        {

painter.drawPixmap(m_hero.m_bullets[i].m_X,m_hero.m_bullets[i].m_Y,m_hero.m_bullets[i].m_
Bullet);
        }

    }

    //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);
        }
    }

    //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])
;
        }
    }

    //测试子弹
    //painter.drawPixmap(temp_Bullet.m_X,temp_Bullet.m_Y,temp_Bullet.m_Bullet);
}
```

代码块 18 位置更新函数

3.2.12 游戏主体

这里游戏主体采用 QTimer 来实现，QTimer 在多线程程序中，可以在一个有事件循环的任何线程中使用 QTimer。使用 QThread::exec()，从非 GUI 线程启动一个事件循环。Qt 使用定时器的线程关联，以确定哪个线程会发出 timeout()信

号。正因为如此，你必须在它的线程中启动和停止定时器，不可能从另一个线程启动定时器。在这里来控制游戏进度时间触发推进。代码如下：

```
void MainScreen::playGame()
{



    //玩游戏 启动定时器
    m_Timer.start();

    //监听定时器的信号
    connect(&m_Timer , &QTimer::timeout,[=](){


        //敌机出场
        enemyToScene();
        //更新游戏中所有元素的坐标
        updatePosition();
        //游戏中的元素 绘制到屏幕中
        update();
        //再调用 paintEvent 函数

        //碰撞检测
        collisionDetection();
    });
}


void MainScreen::enemyToScene()
{
    //累加出场间隔
    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;
        }
    }
}
```

代码块 19 开始游戏函数

3.2.13 鼠标控制绑定

通过 mouseMoveEvent 函数来实现定位鼠标坐标与我方飞机的绑定操作，因为飞机资源为图像，为了使鼠标与飞机中心完美匹配，对 x，y 设计一定的偏移量，代码如下：

```
void MainScreen::mouseMoveEvent(QMouseEvent * event)
{
    int x =  event->x() -m_hero.m_Rect.width() * 0.5;
    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);
}
```

代码块 20 鼠标绑定函数

3.2.14 碰撞检测

每个资源类都可以看成一个长方形，具有边界范围，利用循环遍历是否发生碰撞，如果发生碰撞，产生爆炸效果，并调用相关函数和音效，如果我方飞机与敌方飞机碰撞，则游戏结束。代码如下：

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

```
                m_enemys[i].m_Free = true;
                m_hero.m_bullets[j].m_Free = true;

                //调用爆炸特效
                //播放爆炸效果
                score++;
            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;
                }
            }

        }
    }
}

//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;
    }
    if(m_enemys[i].m_Rect.intersects(m_hero.m_Rect))
    {
        QSound::play(SOUND_BOMB);

        m_Timer.stop();

        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;
                }
            }


            QMessageBox::StandardButton rb = QMessageBox::information(this,"游戏失败","分数为：" + QString::number(score)+"\n"+"游戏是否重新开始？",QMessageBox::Yes | QMessageBox::No, QMessageBox::Yes);
            if(rb == QMessageBox::Yes)
            {
                score = 0;
                Inumber += 10;
                initialDisplay();
                //initScene();
                playGame();
                //需要一个游戏开始的页面
                //slotStart();
            }
            else if (rb == QMessageBox::No)
            {
                exit(1);
            }
            //QMessageBox::about(this,"游戏失败","分数为：" + QString::number(score)+"");
            //exit(1);
        }

    }
}
```

代码块 21 碰撞检测函数

3.3 类图关系

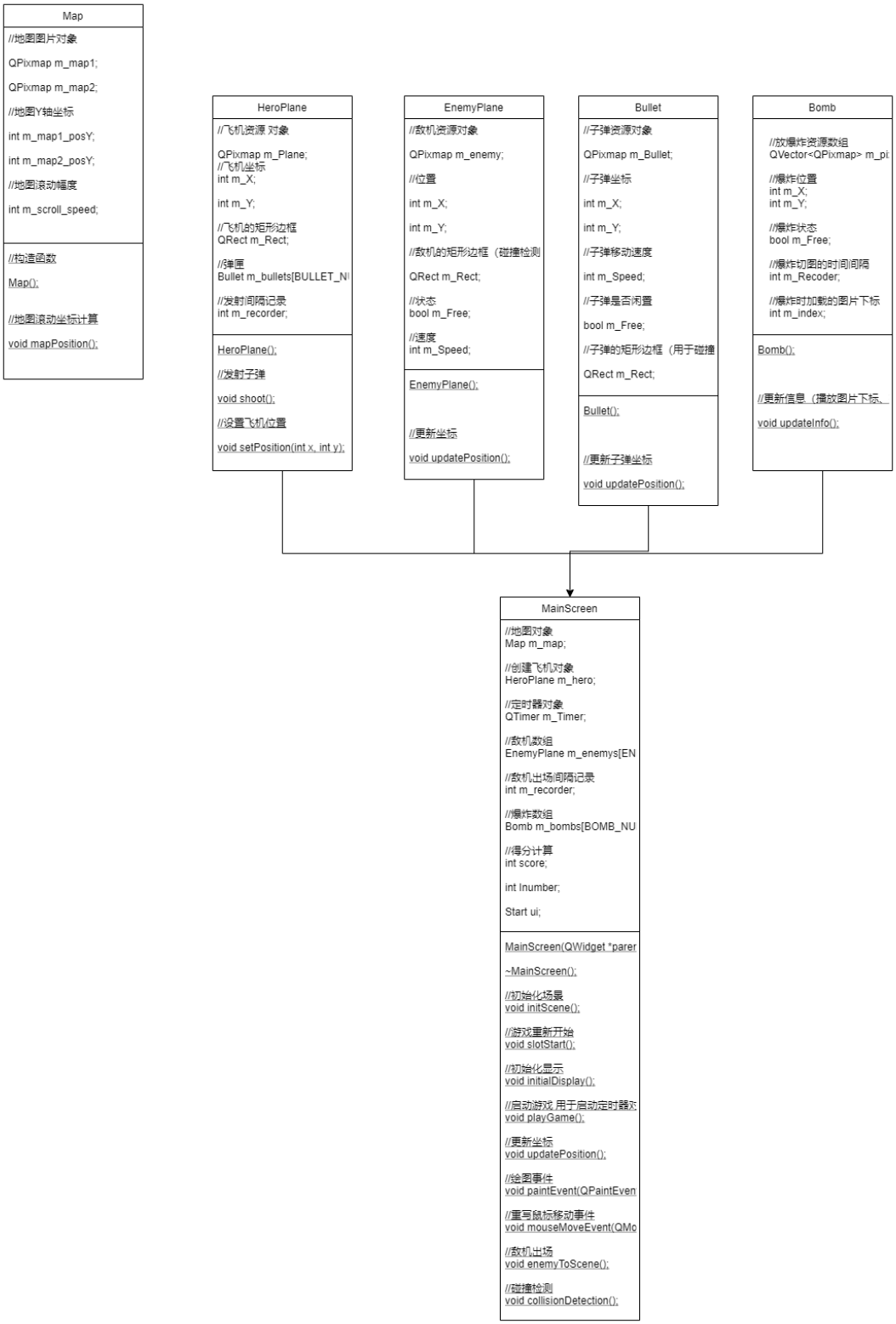

图 12 类关系图

4.程序展示

一些操作实际展示，展示效果如下图：

图 13 丛林版

图 14 熔岩版

图 15 云顶版

5.总结思考

通过使用 Qt 应用框架实现了飞机大战游戏,并通过 QTimer 控制游戏进程,采用面向对象的思想，将每个资源费封装，根据资源不同可以换不同的版本，可玩性较高，基本上实现了游戏目标。该程序是一个集继承、图形界面、事件处理等面向对象编程知识的综合应用的实例程序。