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：

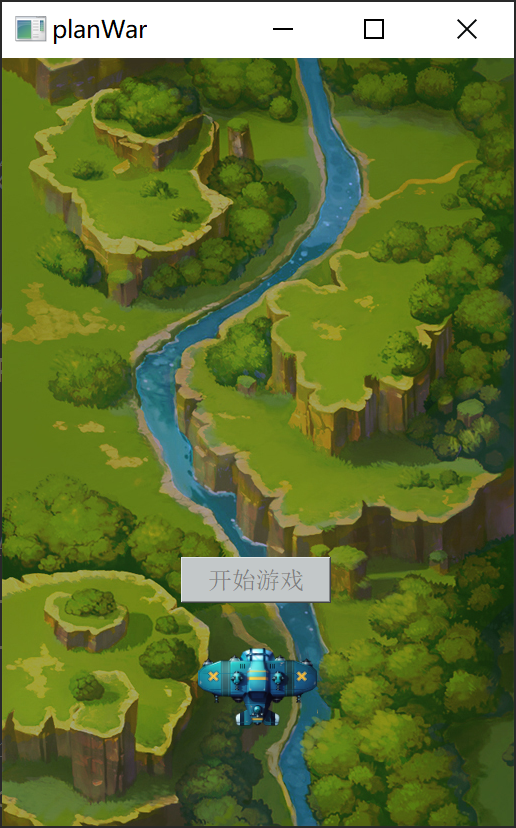


图 1 开始页面

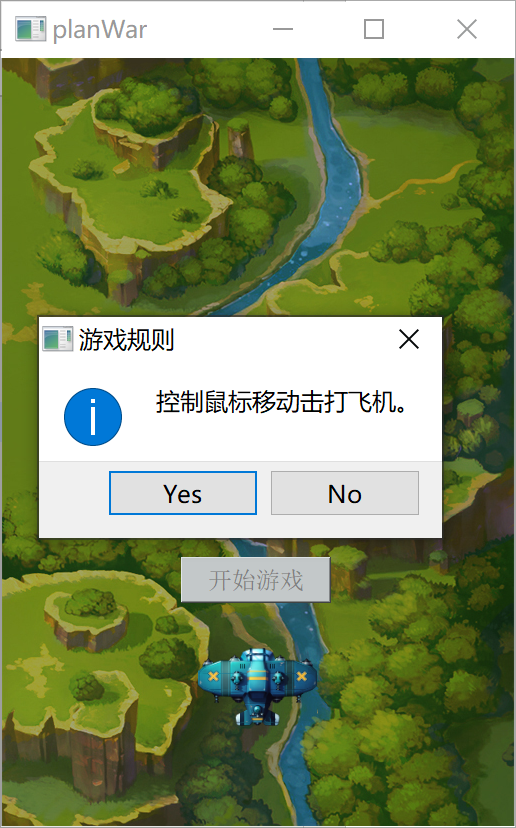


图 2 确认开始页面

## 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。

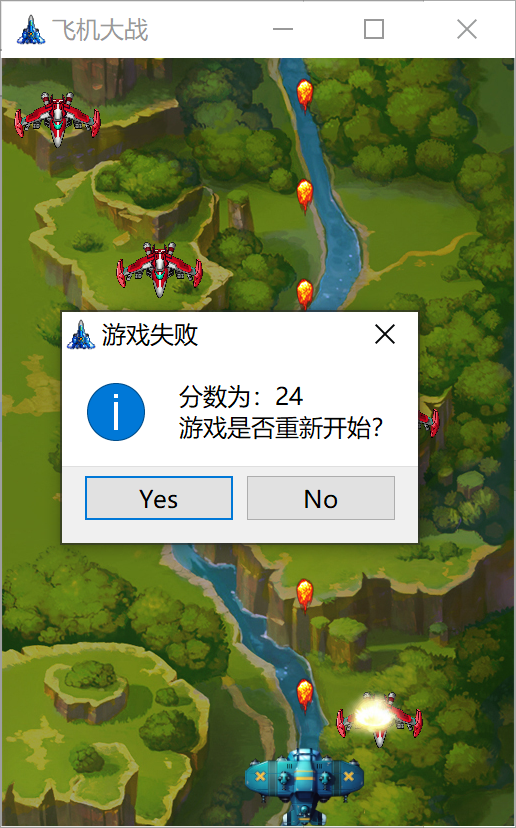


图 4 游戏结束

# 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。

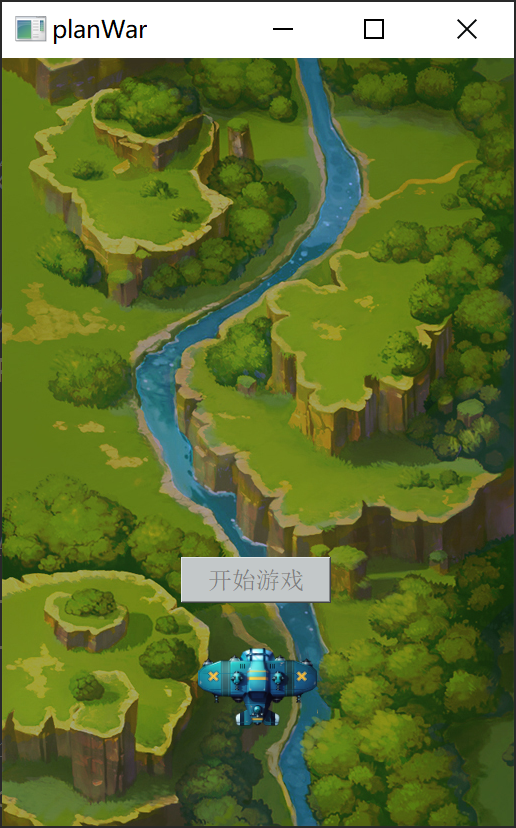
 

图 5游戏页面

## 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：

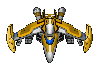
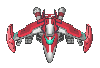
 

图 10 敌方飞机资源展示

|  |
| --- |
| #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：

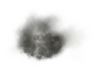
  

图 11 爆炸资源展示

|  |
| --- |
| #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

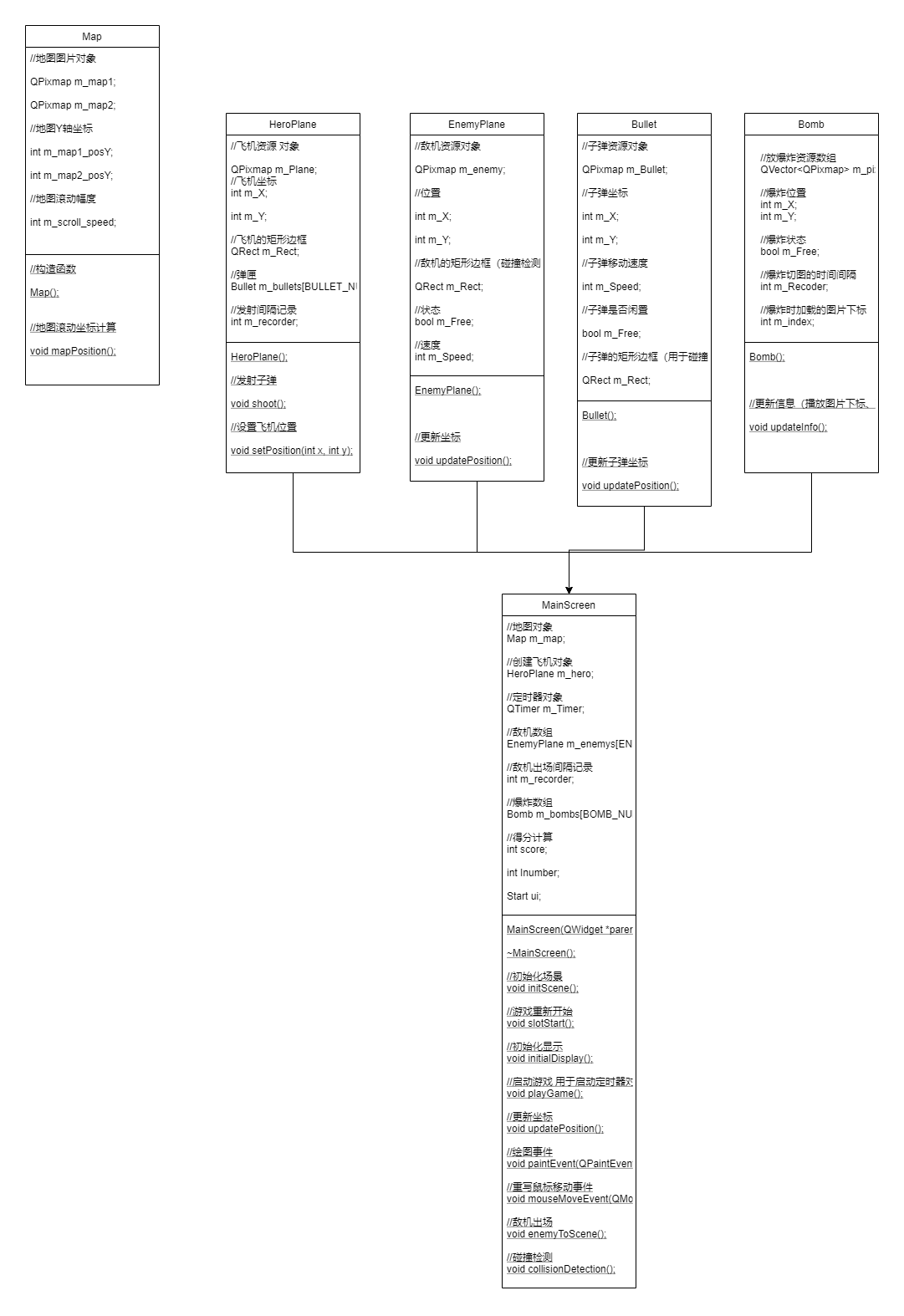


图 12类关系图

# 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。