课程设计报告

问题定义

- **项目名称**

  微信版打飞机。

- **项目目标**

  制作一个性能高效、界面友好、操作简单的游戏软件。

- **游戏规则**

  1. 按下空格键游戏开始。
  2. 通过 WSAD 键控制玩家飞机移动上下左右移动，但不能飞出边境。
  3. 玩家飞机每隔一段时间自动发射一发炮弹。
  4. 玩家飞机被敌机碰到后，生命值会减少，生命值变为 0 的时候，游戏结束。
  5. 在游戏上方会一直出现电脑飞机（敌机），敌机分为大、中、小三种类型，越小的飞机出现的几率越大。
  6. 敌机被玩家飞机击中后，生命值会减少，生命值变为 0 时，该敌机被摧毁，玩家得分。越大的飞机越难击毁，击毁后玩家得分也越多。
  7. 敌机飞到界面下方时，删除敌机。
  8. 在界面外用一个文本显示玩家游戏得分。

可行性研究

- **技术可行性**

  本游戏采用 Fun Code & C++进行开发，运行于 Windows 操作系统。

Fun Code 提供了大量基础类库，可以快速方便地构造出游戏软件。

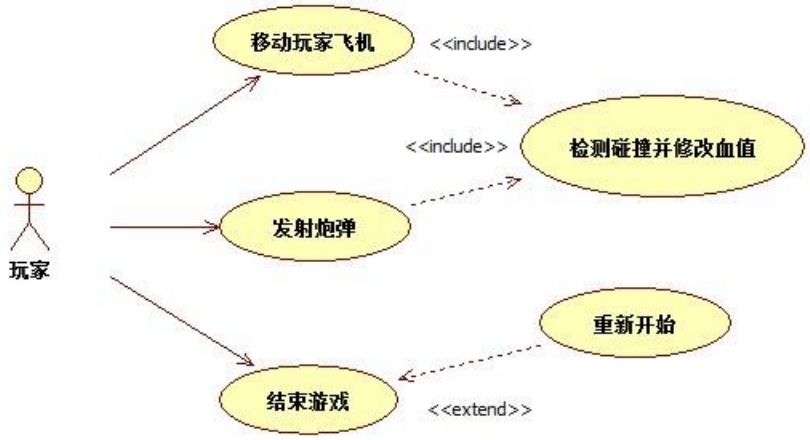
具有一定的 C++语言开发基础，对面向对象程序设计有一定了解。

- **运行可行性**

本游戏操作简单，界面友好，符合用户操作习惯。

需求分析

- **用例图**



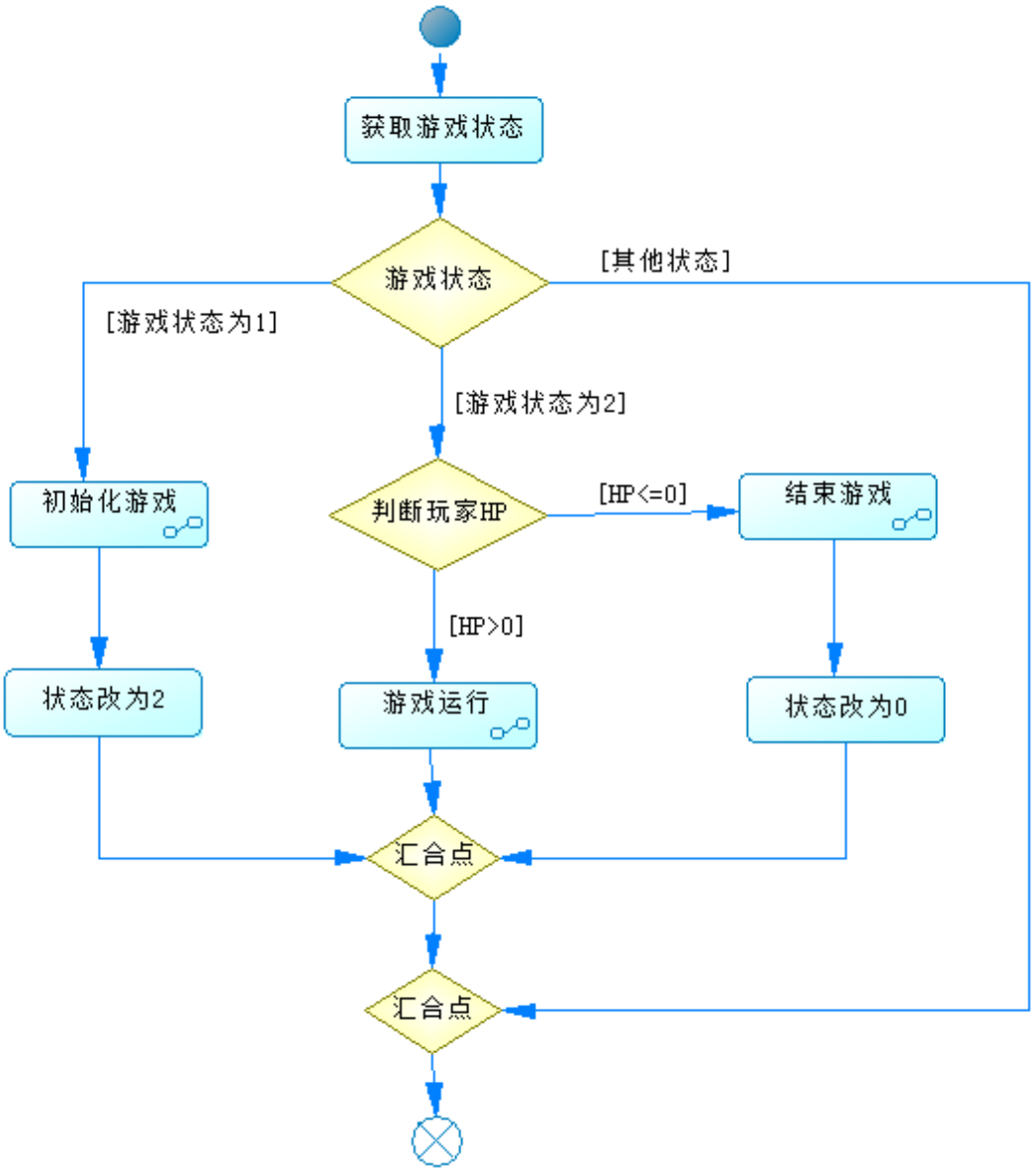
玩家可以移动飞机，并发射炮弹。移动飞机或发射炮弹后要进行碰撞检测，修改相应对象的血值、玩家得分，并销毁血值为 0 的对象。游戏结束后还可以重新开始。

- **游戏流程**

1. 主流程

游戏分四个状态：未开始、开始、运行和结束。CGameMain 类是控制游戏流程的类，游戏屏幕每刷新一次，调用一次该类的 GameMainLoop 方法。GameMainLoop 根据 m_iGameState 的值分别调用该类的 GameInit、GameRun 和 GameEnd 方法。这三个方法对应初始化游戏、游戏运行和游戏结束这三个状态。GameInit 只执行一次。执行后修改游戏状态。屏幕刷新后，进入状态 2，开始执行 GameRun 方法。当玩家

HP>0 即玩家飞机不死的时候，GameRun 一直被执行。玩家飞机被摧毁后，调用 GameEnd 执行一次，清除本局游戏数据，恢复游戏未开始状态。

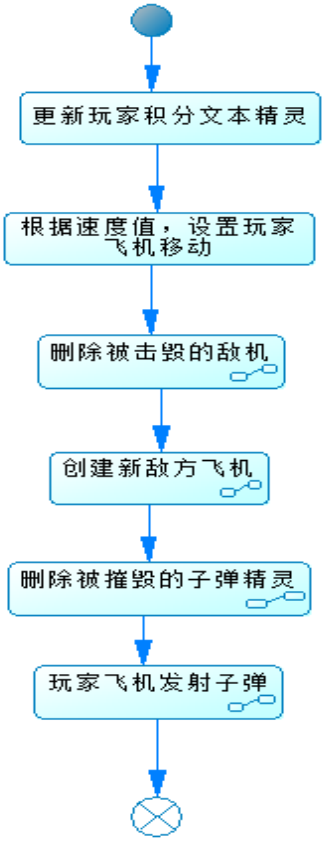


2. 游戏初始化

完成游戏初始工作，界面上的工作：玩家得分显示为 0，创建玩家飞机并放置在指定位置。此外，我们需要创建 PlaneGame 对象，该对象采用单例模式，所以不管游戏运行多少次，PlaneGame 对象始终只有一个，并且把玩家飞机对象加入到 PlaneGame 对象中。

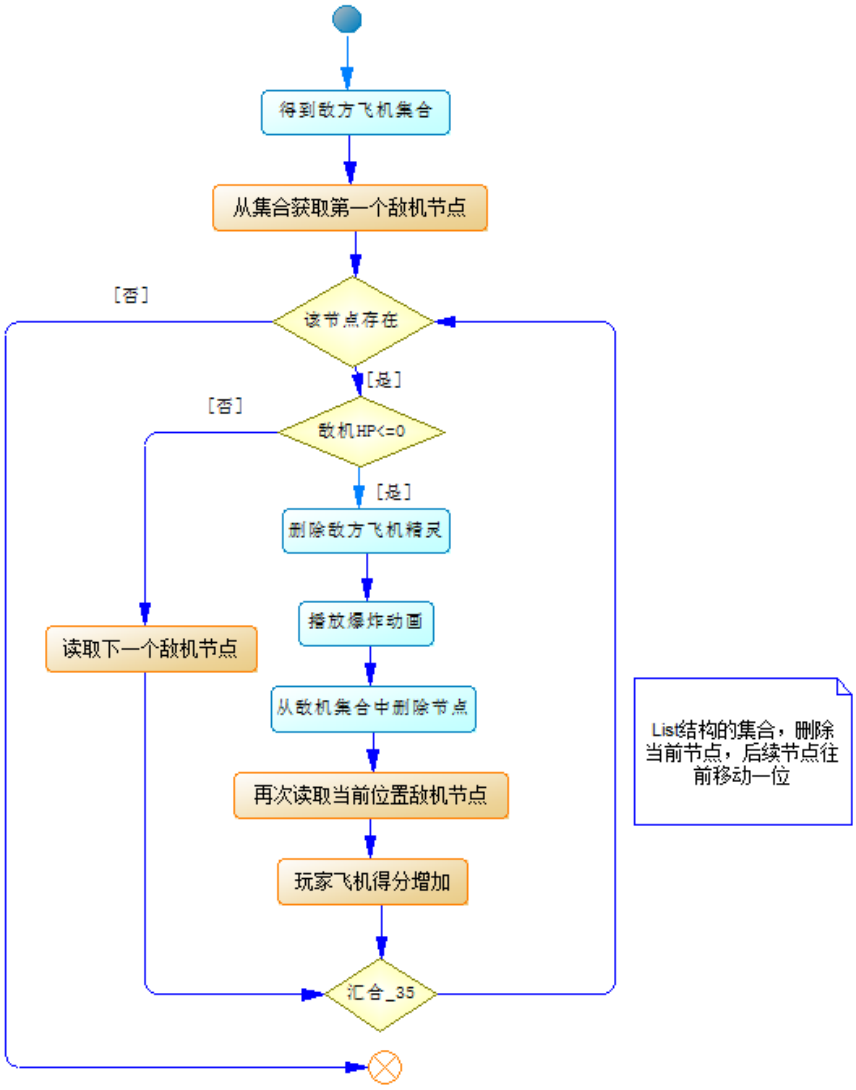
3. 游戏运行

游戏运行中，每刷新一次屏幕，需要根据最新数据，更新玩家得分，移动玩家飞机，删除被摧毁的敌机和子弹，当达到相应的时间间隔时，创建新的敌机精灵和子弹精灵（玩家发射新的子弹）。



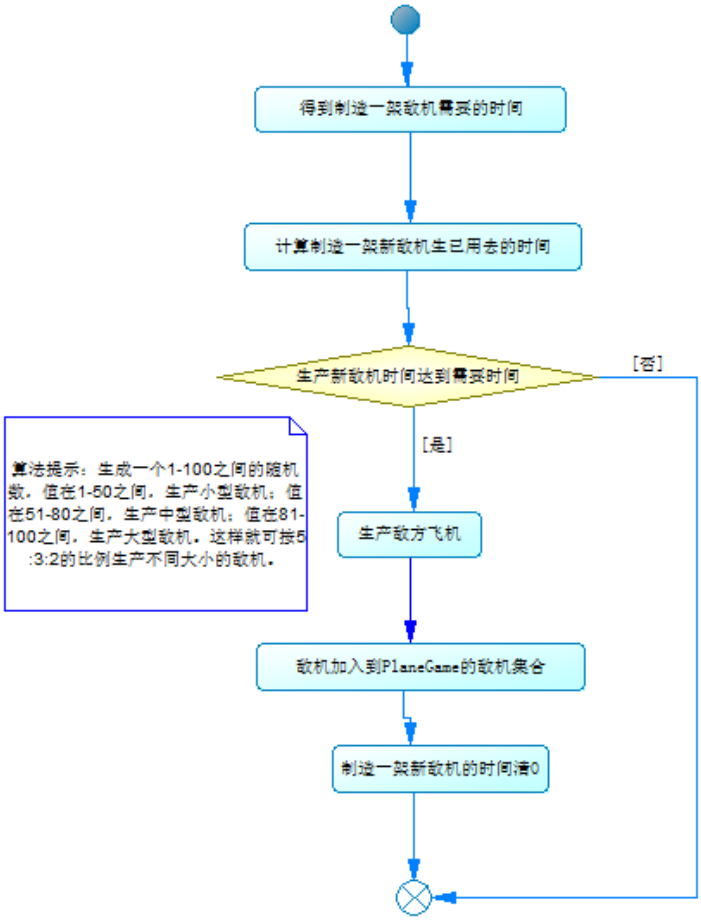
删除被击毁的敌机：

游戏中的敌机对象，都保存在飞机世界类的敌机集合中，从该集合中按顺序从中依次获得敌机，检查其 HP，如果 HP 小于等于 0，删除对应的精灵，然后从集合中删除该节点对象。

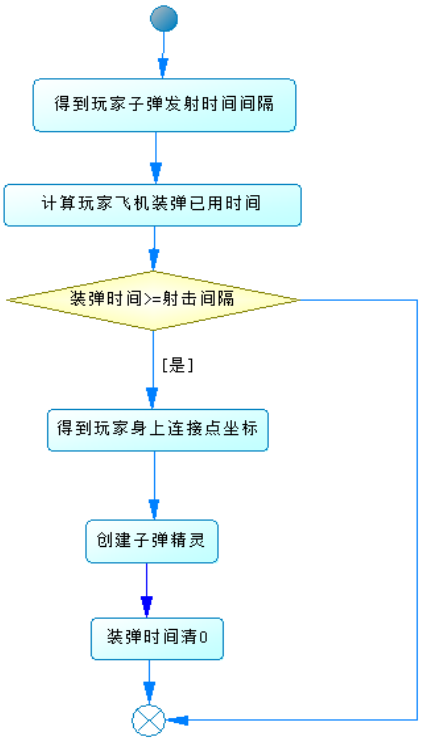


创建新的敌机：

GameRun 方法中有一个参数，用来保存程序主流程每运行一次（屏幕刷新一次）花费的时间。把这些时间累计起来，保存到飞机世界类的属性：制造敌机已用时间。当这个时间超过制造一架敌机需要花费的时间，表示一架新敌机已经制造好了，就在屏幕中生成一架新的敌机。

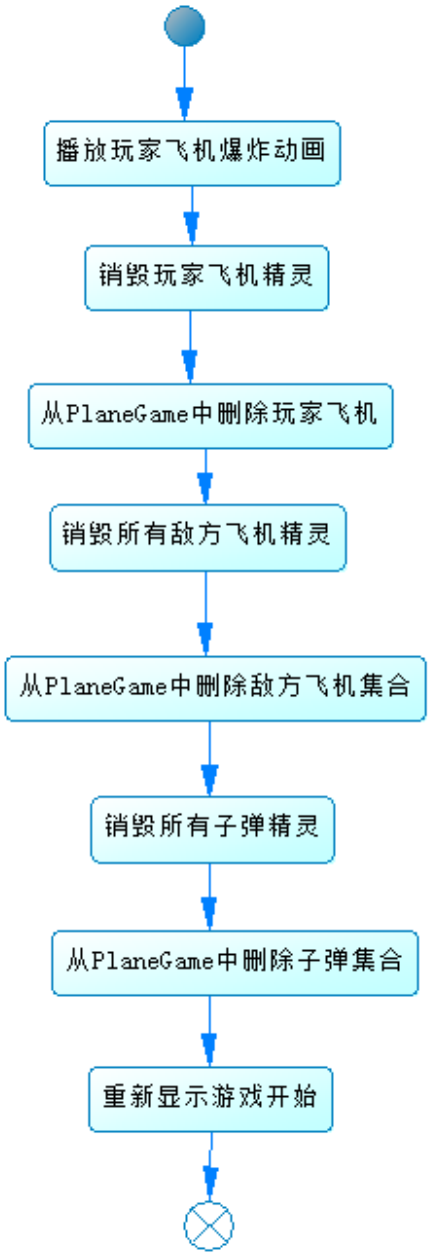

得到制造一架敌机需要的时间
计算制造一架新敌机生已用去的时间
生产新敌机时间达到需要时间
[否]
[是]
算法提示：生成一个1-100之间的随机数，值在1-50之间，生产小型敌机；值在51-80之间，生产中型敌机；值在81-100之间，生产大型敌机。这样就可按5:3:2的比例生产不同大小的敌机。
生产敌方飞机
敌机加入到PlaneGame的敌机集合
制造一架新敌机的时间清0

玩家飞机发射子弹：


得到玩家子弹发射时间间隔
计算玩家飞机装弹已用时间
装弹时间>=射击间隔
[是]
得到玩家身上连接点坐标
创建子弹精灵
装弹时间清0

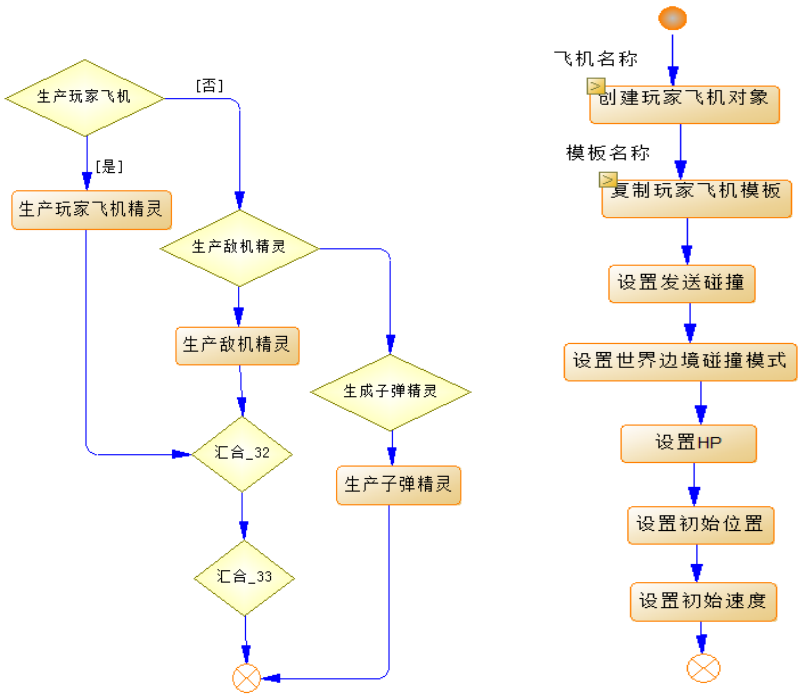
4. 游戏结束

将游戏中剩余的精灵删除，包括删除精灵本身和集合中对应的对象。重新显示游戏开始的提示图片。



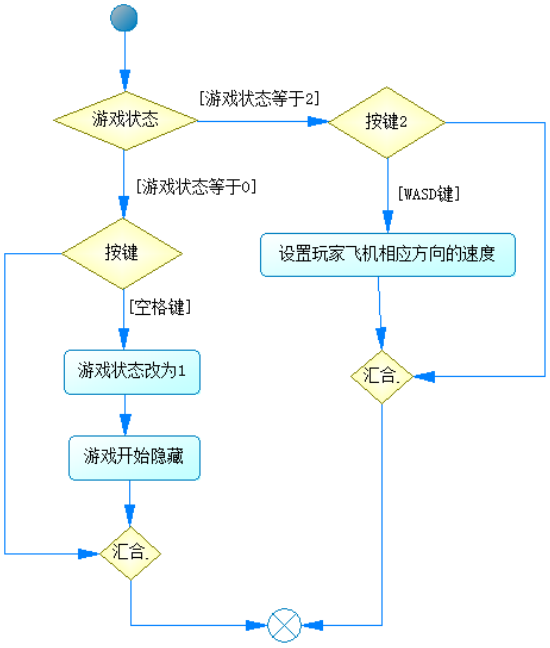
5. 生产精灵对象

本游戏中的玩家飞机、敌方飞机和子弹这三种精灵类的对象都在精灵工厂类中产生。根据参数确定生产何种精灵。右下图是玩家飞机的生产过程，主要是创建对象，复制模板，并设置几个基本属性（碰撞、世界边界、初始位置、初始速度）。



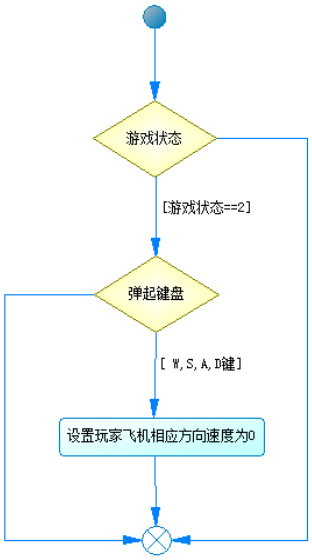
6. 键盘按下事件

本游戏中，游戏状态为 1 时，按下空格键，游戏状态转为 2。游戏状态为 2 时，玩家按下了 WSAD 键，分别设置玩家飞机的上下左右速度。



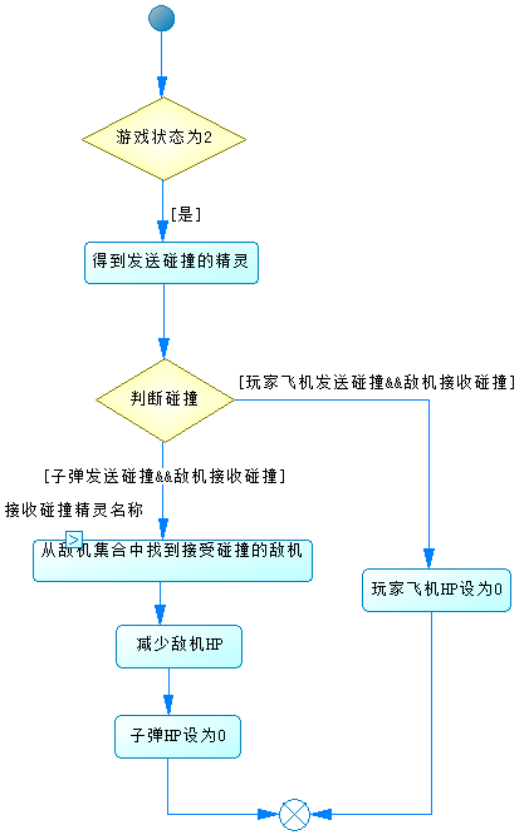
7. 键盘弹起事件

本游戏中，游戏状态为 2 时，玩家松开 WSAD 键，相应设置玩家飞机上下左右速度为 0。



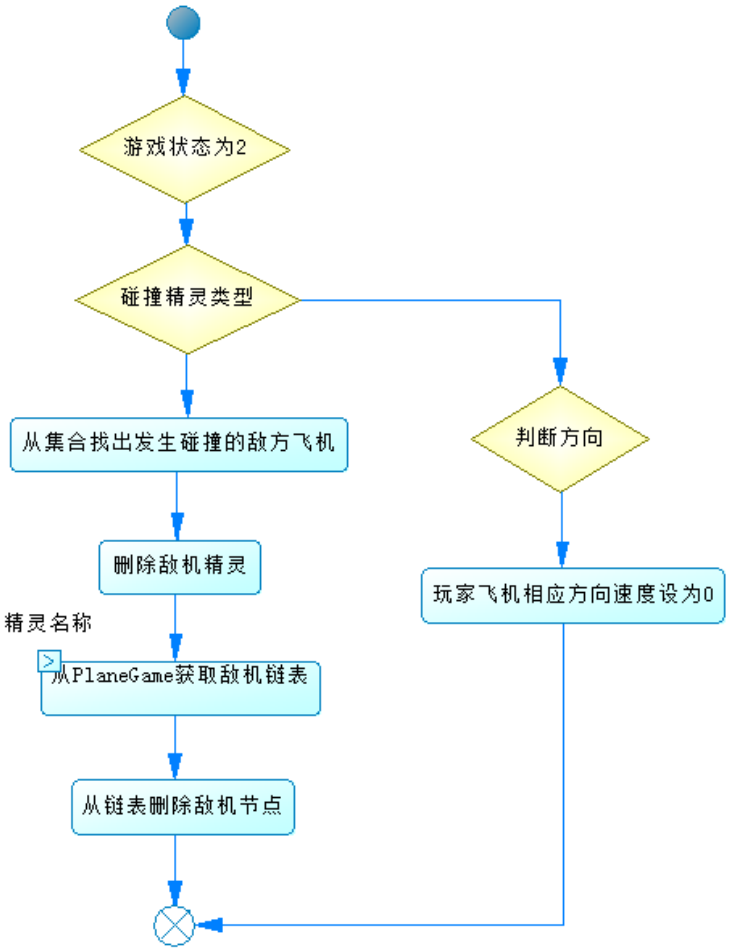
8. 精灵与精灵碰撞

本游戏中 ,玩家飞机和子弹精灵设置为发送碰撞 ,敌方飞机设置为接收碰撞。因此，会发生两种精灵与精灵的碰撞：子弹与敌方飞机的碰撞；玩家飞机与敌方飞机的碰撞。精灵碰撞时，相应减少精灵的 HP。这样，在游戏运行时，就可以得到精灵最新的 HP，并根据 HP 对精灵做相应处理。



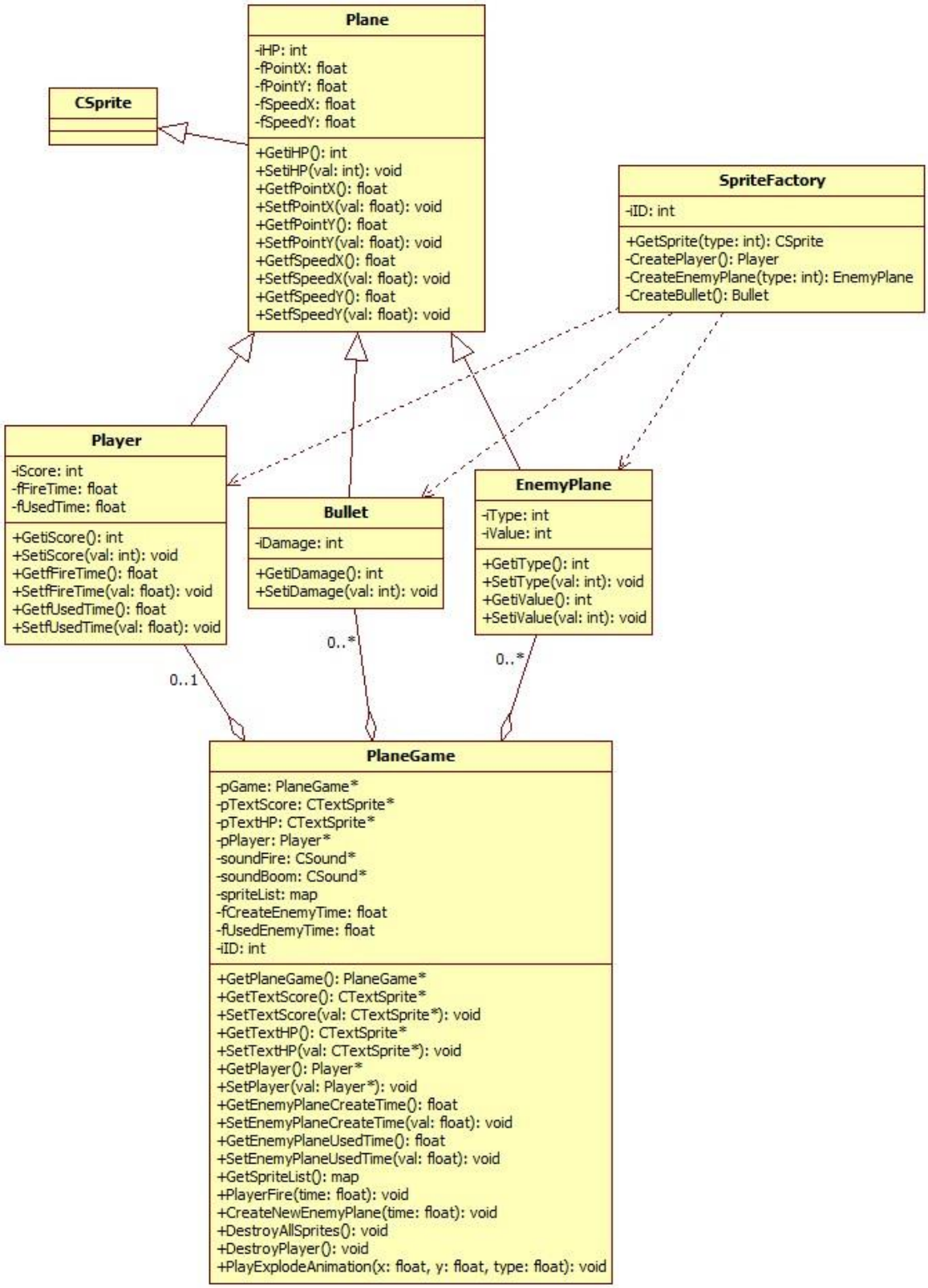
9. 世界边界碰撞事件

本游戏中，游戏状态为 2 时，玩家飞机碰到边界停止继续向前飞行，可通过 Fun Code 设置。敌方飞机碰到边界，删除精灵，并从敌机集合中删除节点。敌方飞机的世界边界应该设置的比游戏屏幕大，这样，敌方飞机碰到屏幕的左右边界不受影响，敌方飞机落到屏幕下方之后才会被删除。



概要/详细设计

- **类设计**


Plane
-iHP: int
-fPointX: float
-fPointY: float
-fSpeedX: float
-fSpeedY: float
+GetiHP(): int
+SetiHP(val: int): void
+GetfPointX(): float
+SetfPointX(val: float): void
+GetfPointY(): float
+SetfPointY(val: float): void
+GetfSpeedX(): float
+SetfSpeedX(val: float): void
+GetfSpeedY(): float
+SetfSpeedY(val: float): void
CSprite
SpriteFactory
-iID: int
+GetSprite(type: int): CSprite
-CreatePlayer(): Player
-CreateEnemyPlane(type: int): EnemyPlane
-CreateBullet(): Bullet
Player
-iScore: int
-fFireTime: float
-fUsedTime: float
+GetiScore(): int
+SetiScore(val: int): void
+GetfFireTime(): float
+SetfFireTime(val: float): void
+GetfUsedTime(): float
+SetfUsedTime(val: float): void
Bullet
-iDamage: int
+GetiDamage(): int
+SetiDamage(val: int): void
EnemyPlane
-iType: int
-iValue: int
+GetiType(): int
+SetiType(val: int): void
+GetiValue(): int
+SetiValue(val: int): void
0..1
0..*
0..*
PlaneGame
-pGame: PlaneGame*
-pTextScore: CTextSprite*
-pTextHP: CTextSprite*
-pPlayer: Player*
-soundFire: CSound*
-soundBoom: CSound*
-spriteList: map
-fCreateEnemyTime: float
-fUsedEnemyTime: float
-iID: int
+GetPlaneGame(): PlaneGame*
+GetTextScore(): CTextSprite*
+SetTextScore(val: CTextSprite*): void
+GetTextHP(): CTextSprite*
+SetTextHP(val: CTextSprite*): void
+GetPlayer(): Player*
+SetPlayer(val: Player*): void
+GetEnemyPlaneCreateTime(): float
+SetEnemyPlaneCreateTime(val: float): void
+GetEnemyPlaneUsedTime(): float
+SetEnemyPlaneUsedTime(val: float): void
+GetSpriteList(): map
+PlayerFire(time: float): void
+CreateNewEnemyPlane(time: float): void
+DestroyAllSprites(): void
+DestroyPlayer(): void
+PlayExplodeAnimation(x: float, y: float, type: float): void

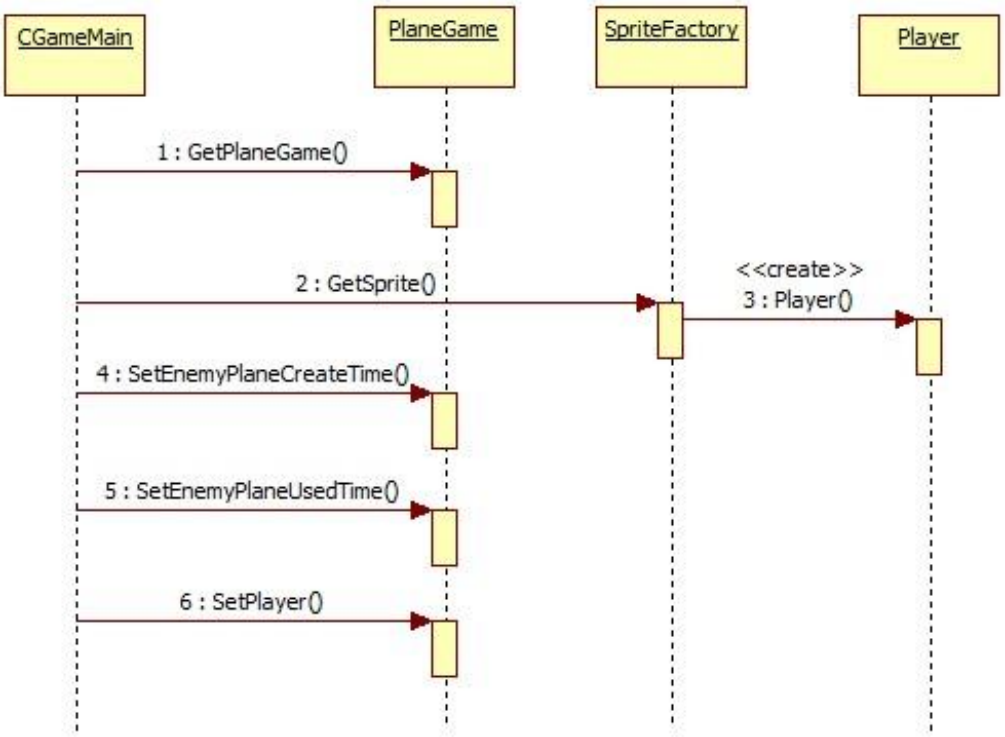
Plane 类继承自 CSprite 类，它是 Player 类、Bullet 类和 EnemyPlane 类的基类。Player 类、Bullet 类和 EnemyPlane 类是实体类，它们维护着对象相应的属性值，并提供对相应对象进行操作的方法。

PlaneGame 类用以管理 Player 类、Bullet 类和 EnemyPlane 类的对象，相当于控制类，控制游戏的逻辑。PlaneGame 类采用单例模式，只能存在一个对象。

SpriteFactory 类提供统一方法，用以创建 Player 类、Bullet 类和 EnemyPlane 类的对象，采用工厂模式。

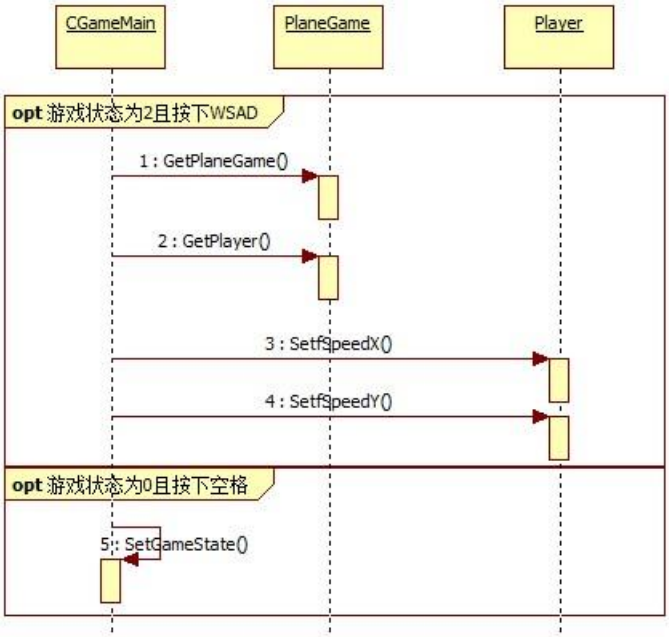
- **对象动态行为分析**

1. 游戏初始化



获取 PlaneGame 对象，设置敌机创建所需时间和已用时间。创建 Player 对象，并加入到 PlaneGame 对象中。

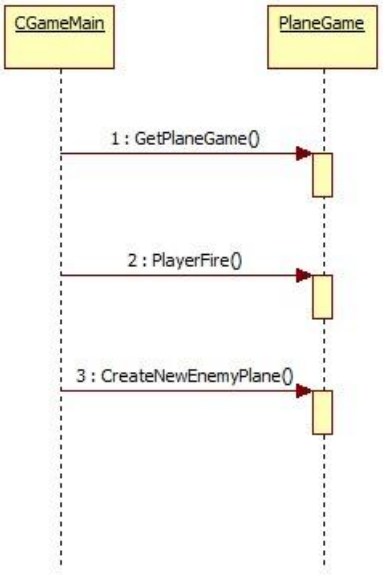
2. 键盘按下



如果游戏状态为 2 ,且按下了 WSAD 键 ,那么得到 PlaneGame 对象 ,通过 PlaneGame 对象得到 Player 对象，然后设置 Player 对象的速度。

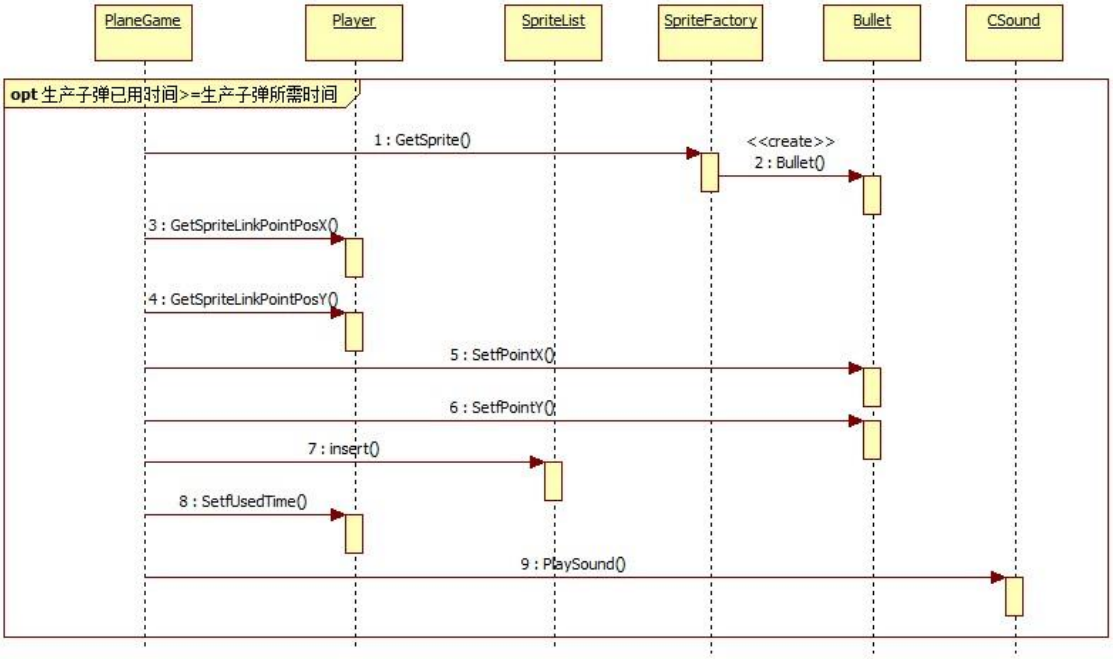
如果游戏状态为 0，且按下了空格键，那么设置游戏状态为开始。

3. 游戏运行



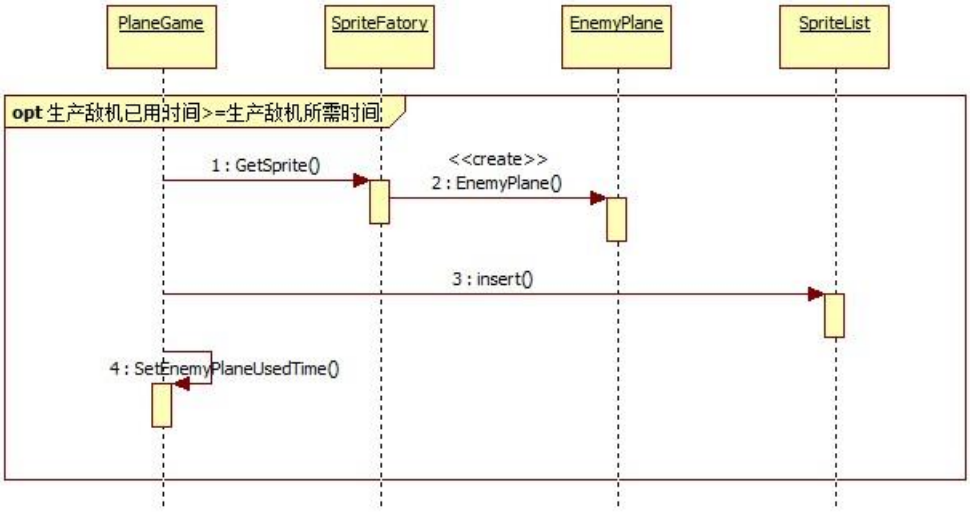
得到 PlaneGame 对象，然后让玩家开火并创建新的敌机。

4. 玩家发射子弹



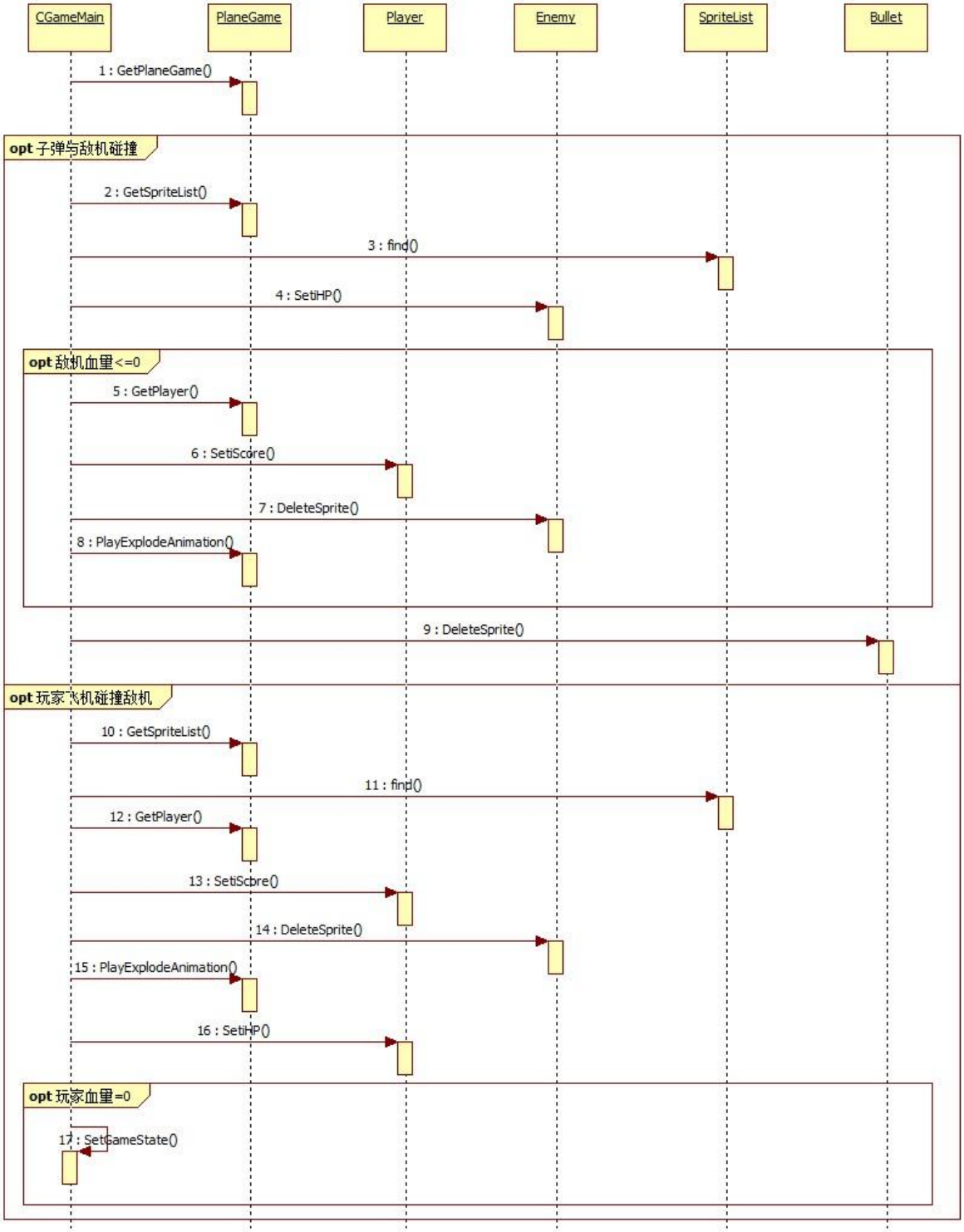
如果生成子弹已用时间大于等于所需时间，那么创建子弹对象，获取玩家对象的连接点位置，并将这个位置赋值给子弹对象，再将子弹对象插入精灵列表，然后将生成子弹已用时间归零，最后播放发射子弹的声音。

5. 产生新敌机



如果生成敌机已用时间大于等于所需时间，那么创建敌机对象，并加入到精灵列表中，最后将生产敌机已用时间归零。

6. 精灵与精灵碰撞



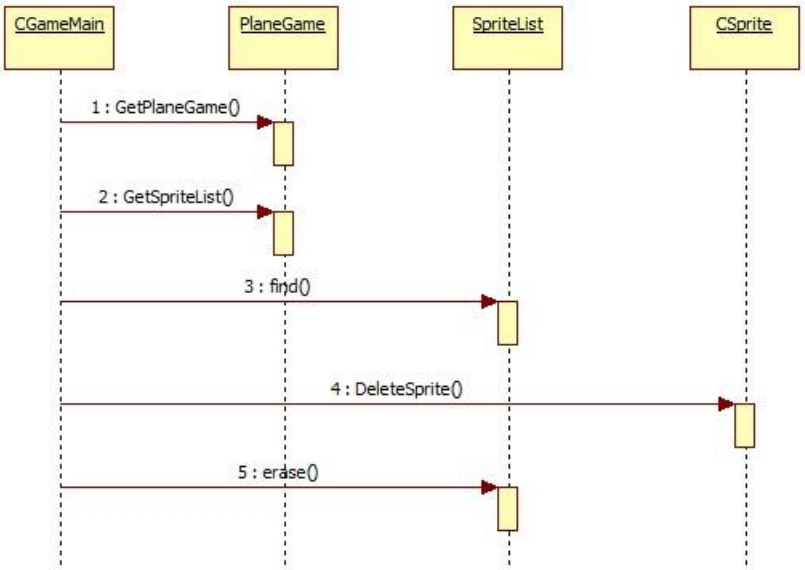
首先得到 PlaneGame 对象。

如果是子弹与敌机碰撞，那么从精灵列表中找到发生碰撞的子弹与敌机，修改敌机的血量。如果敌机血量小于等于 0，那么得到玩家对象，并增加其得分，然后删除敌机对象并播

放爆炸动画。最后删除子弹对象。

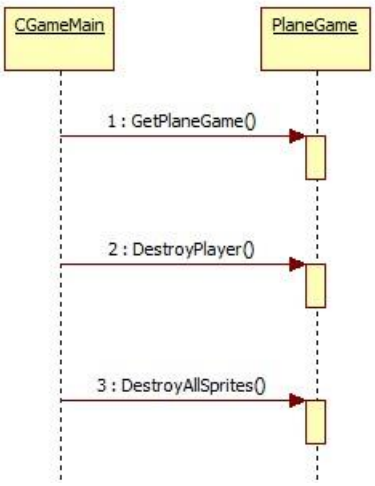
如果是玩家飞机与敌机碰撞，那么从精灵列表中找到发生碰撞的敌机，删除敌机并播放爆炸动画，然后得到玩家对象，增加其得分，并减少血量。如果玩家血量为 0，则设置游戏状态为即将结束。

7. 精灵与世界边界碰撞



得到 PlaneGame 对象，然后从精灵列表中找到发生碰撞的对象（由于玩家对象不在精灵列表中，所以不会被找到），然后删除碰撞的对象，并将其从列表中移除。

8. 游戏结束



得到 PlaneGame 对象，然后销毁玩家对象以及精灵列表中所有的对象。

编码

- **EnumDefine 类**

该类中只定义一些程序中需要的静态常量。

```
#ifndef ENUMDEFINE_H_INCLUDED
#define ENUMDEFINE_H_INCLUDED

class EnumDefine
{
public:
    static const int BIG_ENEMY = 0;
    static const int MEDIUM_ENEMY = 1;
    static const int SMALL_ENEMY = 2;
    static const int PLAYER_PLANE = 3;
    static const int BULLET = 4;
    static const int BIG_ENEMY_VALUE = 10;
    static const int MEDIUM_ENEMY_VALUE = 5;
    static const int SMALL_ENEMY_VALUE = 2;
};

#endif // ENUMDEFINE_H_INCLUDED
```

- **Plane 类**

头文件

```
#ifndef PLANE_H
#define PLANE_H

#include "CommonClass.h"

class Plane : public CSprite
{
public:
    Plane(const char *name);
```

```
    virtual ~Plane();
    // 血量
    int GetiHP() { return iHP; }
    void SetiHP(int val) { iHP = val; }
    // 坐标 X
    float GetfPointX() { return fPointX; }
    void SetfPointX(float val) { fPointX = val; SetSpritePositionX(val); }
    // 坐标 Y
    float GetfPointY() { return fPointY; }
    void SetfPointY(float val) { fPointY = val; SetSpritePositionY(val); }
    // 速度 X
    float GetfSpeedX() { return fSpeedX; }
    void SetfSpeedX(float val) { fSpeedX = val; SetSpriteLinearVelocityX(val); }
    // 速度 Y
    float GetfSpeedY() { return fSpeedY; }
    void SetfSpeedY(float val) { fSpeedY = val; SetSpriteLinearVelocityY(val); }
protected:
private:
    int iHP = 0;
    float fPointX = 0.0f;
    float fPointY = 0.0f;
    float fSpeedX = 0.0f;
    float fSpeedY = 0.0f;
};

#endif // PLANE_H
```

源文件

```
#include "Plane.h"

Plane::Plane(const char *name) : CSprite(name)
{
    //ctor
}

Plane::~Plane()
{
    //dtor
}
```

- **Player 类**

头文件

```
#ifndef PLAYER_H
#define PLAYER_H

#include "Plane.h"

class Player : public Plane
{
public:
    Player(const char *name);
    virtual ~Player();
    // 分数
    int GetiScore() { return iScore; }
    void SetiScore(int val) { iScore = val; }
    // 开火时间间隔
    float GetfFireTime() { return fFireTime; }
    void SetfFireTime(float val) { fFireTime = val; }
    // 开火已用时间
    float GetfUsedTime() { return fUsedTime; }
    void SetfUsedTime(float val) { fUsedTime = val; }
protected:
private:
    int iScore = 0;
    float fFireTime = 0.0f;
    float fUsedTime = 0.0f;
};

#endif // PLAYER_H
```

源文件

```
#include "Player.h"
Player::Player(const char *name) : Plane(name)
{
    //ctor
    CloneSprite("playerPlane");
}

Player::~Player()
{
    //dtor
}
```

- **Bullet 类**

头文件

```
#ifndef BULLET_H
#define BULLET_H

#include "Plane.h"

class Bullet : public Plane
{
public:
    Bullet(const char *name);
    virtual ~Bullet();
    int GetiDamage() { return iDamage; }
    void SetiDamage(int val) { iDamage = val; }
protected:
private:
    int iDamage = 0;
};

#endif // BULLET_H
```

源文件

```
#include "Bullet.h"

Bullet::Bullet(const char *name) : Plane(name)
{
    //ctor
    CloneSprite("shellPlane");
}

Bullet::~Bullet()
{
    //dtor
}
```

- **EnemyPlane 类**

头文件

```
#ifndef ENEMYPLANE_H
#define ENEMYPLANE_H
```

```
#include "Plane.h"

class EnemyPlane : public Plane
{
    public:
        EnemyPlane(const char *name);
        virtual ~EnemyPlane();
        int GetiValue() { return iValue; }
        void SetiValue(int val) { iValue = val; }
        int GetiType() { return iType; }
        void SetiType(int val) { iType = val; }
    protected:
    private:
        int iValue = 0;
        int iType = 0;
};

#endif // ENEMYPLANE_H
```

源文件

```
#include "EnemyPlane.h"

EnemyPlane::EnemyPlane(const char *name) : Plane(name)
{
    //ctor
}

EnemyPlane::~EnemyPlane()
{
    //dtor
}
```

- **SpriteFactory 类**

头文件

```
#ifndef SPRITEFACTORY_H
#define SPRITEFACTORY_H

#include "EnumDefine.h"
#include "Bullet.h"
#include "Player.h"
```

```
#include "EnemyPlane.h"
#include <cstdio>

class SpriteFactory
{
public:
    SpriteFactory();
    virtual ~SpriteFactory();
    static CSprite *GetSprite(int type);
protected:
private:
    static Player *CreatePlayer();
    static EnemyPlane *CreateEnemyPlane(const int type);
    static Bullet *CreateBullet();
    static int iID;
};

#endif // SPRITEFACTORY_H
```

源文件

```
#include "SpriteFactory.h"

int SpriteFactory::iID = 0;

SpriteFactory::SpriteFactory()
{
    //ctor
}

SpriteFactory::~SpriteFactory()
{
    //dtor
}

CSprite *SpriteFactory::GetSprite(int type)
{
    CSprite *rtn = NULL;
    switch (type)
    {
        case EnumDefine::SMALL_ENEMY:
        case EnumDefine::MEDIUM_ENEMY:
        case EnumDefine::BIG_ENEMY:
            rtn = CreateEnemyPlane(type);
            break;
```

```
        case EnumDefine::PLAYER_PLANE:
            rtn = CreatePlayer();
            break;
        case EnumDefine::BULLET:
            rtn = CreateBullet();
            break;
    }
    return rtn;
}

Player *SpriteFactory::CreatePlayer()
{
    ++iID;
    char name[16];
    sprintf(name, "player%d", iID);
    Player *player = new Player(name);
    player->SetfPointX(0.0f);
    player->SetfPointY(CSystem::GetScreenBottom() - 5.0f);
    player->SetiHP(5);
    player->SetiScore(0);
    player->SetfFireTime(0.3f);
    player->SetfUsedTime(0.0f);
    player->SetSpriteCollisionSend(true);
    player->SetSpriteWorldLimit(WORLD_LIMIT_STICKY, CSystem::GetScreenLeft(),
CSystem::GetScreenTop(),
                                CSystem::GetScreenRight(),
CSystem::GetScreenBottom());
    return player;
}

EnemyPlane *SpriteFactory::CreateEnemyPlane(const int type)
{
    ++iID;
    char name[16];
    EnemyPlane *enemy = NULL;
    switch (type)
    {
    case EnumDefine::SMALL_ENEMY:
        sprintf(name, "enemy%d", iID);
        enemy = new EnemyPlane(name);
        enemy->CloneSprite("smallPlane");
        enemy->SetiValue(EnumDefine::SMALL_ENEMY_VALUE);
        enemy->SetiHP(1);
        enemy->SetfSpeedY(30);
```

```
        break;
    case EnumDefine::MEDIUM_ENEMY:
        sprintf(name, "enemy%d", iID);
        enemy = new EnemyPlane(name);
        enemy->CloneSprite("mediumPlane");
        enemy->SetiValue(EnumDefine::MEDIUM_ENEMY_VALUE);
        enemy->SetiHP(3);
        enemy->SetfSpeedY(20);
        break;
    case EnumDefine::BIG_ENEMY:
        sprintf(name, "enemy%d", iID);
        enemy = new EnemyPlane(name);
        enemy->CloneSprite("bigPlane");
        enemy->SetiValue(EnumDefine::BIG_ENEMY_VALUE);
        enemy->SetiHP(5);
        enemy->SetfSpeedY(10);
        break;
    }
    enemy->SetiType(type);
    enemy->SetfPointY(CSystem::GetScreenTop() - 5.0f);
    enemy->SetfPointX(CSystem::GetScreenLeft() + 5.0f + rand() %
((int)CSystem::GetScreenRight() - (int)CSystem::GetScreenLeft() - 10));
    enemy->SetSpriteCollisionReceive(true);
    enemy->SetSpriteWorldLimit(WORLD_LIMIT_NULL, CSystem::GetScreenLeft(),
CSystem::GetScreenTop() - 13.0f,
                               CSystem::GetScreenRight(),
CSystem::GetScreenBottom() + 13.0f);
    return enemy;
}

Bullet *SpriteFactory::CreateBullet()
{
    ++iID;
    char name[16];
    sprintf(name, "bullet%d", iID);
    Bullet *bullet = new Bullet(name);
    bullet->SetiHP(1);
    bullet->SetiDamage(1);
    bullet->SetfSpeedY(-50.0f);
    bullet->SetSpriteCollisionSend(true);
    bullet->SetSpriteWorldLimit(WORLD_LIMIT_NULL, CSystem::GetScreenLeft(),
CSystem::GetScreenTop() - 5.0f,
                                CSystem::GetScreenRight(),
CSystem::GetScreenBottom());
```

```
    return bullet;
}
```

- **PlaneGame 类**

头文件

```
#ifndef PLANEGAME_H
#define PLANEGAME_H

#include "SpriteFactory.h"
#include <map>
using namespace std;

class Compare
{
public:
    bool operator() (const char *a, const char *b)
    {
        return strcmp(a, b) < 0;
    }
};

class PlaneGame
{
public:
    virtual ~PlaneGame();
    static PlaneGame *GetPlaneGame();
    // 分数显示
    CTextSprite *GetTextScore() { return pTextScore; }
    void SetTextScore(CTextSprite *val) { pTextScore = val; }
    // 血量显示
    CTextSprite *GetTextHP() { return pTextHP; }
    void setTextHP(CTextSprite *val) { pTextHP = val; }
    // 玩家
    Player *GetPlayer() { return pPlayer; }
    void SetPlayer(Player *val) { pPlayer = val; }
    // 敌机创建时间间隔
    float GetEnemyPlaneCreateTime() { return fCreateEnemyTime; }
    void SetEnemyPlaneCreateTime(float val) { fCreateEnemyTime = val; }
    // 敌机创建已用时间
    float GetEnemyPlaneUsedTime() { return fUsedEnemyTime; }
    void SetEnemyPlaneUsedTime(float val) { fUsedEnemyTime = val; }
```

```
    // 精灵列表
    map<const char *, CSprite *, Compare>& GetSpriteList() { return spriteList; }
    // 玩家开火
    void PlayerFire(float time);
    // 创建新的敌机
    void CreateNewEnemyPlane(float time);
    // 销毁所有精灵
    void DestroyAllSprites();
    // 销毁玩家飞机
    void DestroyPlayer();
    // 显示爆炸效果
    void PlayExplodeAnimation(float x, float y, int type);
protected:
private:
    PlaneGame();
    static PlaneGame *pGame;
    CTextSprite *pTextScore = NULL;
    CTextSprite *pTextHP = NULL;
    Player *pPlayer = NULL;
    CSound *soundFire = new CSound("paizi", false, 0.5f);
    CSound *soundBoom = new CSound("baozha", false, 1.0f);
    map<const char *, CSprite *, Compare> spriteList;
    float fCreateEnemyTime = 0.0f;
    float fUsedEnemyTime = 0.0f;
    static int iID;
};

#endif // PLANEGAME_H
```

源文件

```
#include "PlaneGame.h"

PlaneGame *PlaneGame::pGame = NULL;
int PlaneGame::iID = 0;

PlaneGame::PlaneGame()
{
    //ctor
}

PlaneGame::~PlaneGame()
{
    //dtor
    delete soundBoom;
```

```
    delete soundFire;
}

PlaneGame *PlaneGame::GetPlaneGame()
{
    if (pGame == NULL)
        pGame = new PlaneGame();
    return pGame;
}

void PlaneGame::PlayerFire(float time)
{
    pPlayer->SetfUsedTime(pPlayer->GetfUsedTime() + time);
    // 到了开火的时间
    if (pPlayer->GetfUsedTime() >= pPlayer->GetfFireTime())
    {
        // 产生两颗子弹
        for (int i = 1; i <= 2; i++)
        {
            Bullet *bullet = (Bullet
*)SpriteFactory::GetSprite(EnumDefine::BULLET);
            bullet->SetfPointX(pPlayer->GetSpriteLinkPointPosX(i));
            bullet->SetfPointY(pPlayer->GetSpriteLinkPointPosY(i));
            spriteList.insert(make_pair(bullet->GetName(), bullet));
        }
        pPlayer->SetfUsedTime(0.0f);
        // 播放开火声音
        soundFire->PlaySound();
    }
}

void PlaneGame::CreateNewEnemyPlane(float time)
{
    SetEnemyPlaneUsedTime(GetEnemyPlaneUsedTime() + time);
    // 到了创建敌机的时间
    if (GetEnemyPlaneUsedTime() >= GetEnemyPlaneCreateTime())
    {
        // 产生 0-99 的随机数 0-49 小 50-79 中 80-99 大
        int rnd = rand() % 100;
        int type;
        if (rnd >= 0 && rnd < 50)
            type = EnumDefine::SMALL_ENEMY;
        else if (rnd >= 50 && rnd < 80)
            type = EnumDefine::MEDIUM_ENEMY;
```

```
        else
            type = EnumDefine::BIG_ENEMY;
        // 产生敌机并插入列表
        EnemyPlane *enemy = (EnemyPlane *)SpriteFactory::GetSprite(type);
        spriteList.insert(make_pair(enemy->GetName(), enemy));
        SetEnemyPlaneUsedTime(0.0f);
    }
}

void PlaneGame::DestroyAllSprites()
{
    // 遍历精灵列表
    for (auto it : spriteList)
    {
        // 如果是敌机 则播放爆炸效果
        if (it.second->GetName()[0] == 'e')
            PlayExplodeAnimation(it.second->GetSpritePositionX(),
it.second->GetSpritePositionY(),
                                 ((EnemyPlane *)it.second)->GetiType());
        // 删除精灵
        it.second->DeleteSprite();
        delete it.second;
    }
    spriteList.clear();
}

void PlaneGame::DestroyPlayer()
{
    PlayExplodeAnimation(pPlayer->GetSpritePositionX(),
pPlayer->GetSpritePositionY(), EnumDefine::PLAYER_PLANE);
    pPlayer->DeleteSprite();
    delete pPlayer;
}

void PlaneGame::PlayExplodeAnimation(float x, float y, int type)
{
    ++iID;
    char name[16];
    sprintf(name, "explode%d", iID);
    // 产生指定类型的动画精灵
    CAnimateSprite *explode = new CAnimateSprite(name);
    switch (type)
    {
    case EnumDefine::SMALL_ENEMY:
```

```
            explode->CloneSprite("smallExplode");
            break;
        case EnumDefine::MEDIUM_ENEMY:
            explode->CloneSprite("mediumExplode");
            break;
        case EnumDefine::BIG_ENEMY:
            explode->CloneSprite("bigExplode");
            break;
        case EnumDefine::PLAYER_PLANE:
            explode->CloneSprite("playerExplode");
            break;
        }
        // 播放爆炸动画
        explode->SetSpritePosition(x, y);

explode->AnimateSpritePlayAnimation(explode->GetAnimateSpriteAnimationName(
), false);
        explode->SetSpriteLifeTime(0.2f);
        // 播放爆炸声音
        soundBoom->PlaySound();
}
```

- **CGameMain 类**

头文件

```
////////////////////////////////////////////////////////////////////////////
//////
//
//
//
//
////////////////////////////////////////////////////////////////////////////
//////
#ifndef _LESSON_X_H_
#define _LESSON_X_H_
//
#include <Windows.h>

////////////////////////////////////////////////////////////////////////////
//////
//
// 游戏总管类。负责处理游戏主循环、游戏初始化、结束等工作
```

```
class CGameMain
{
private:
    int m_iGameState = 0;              // 游戏状态，0：结束或者等待开始；1：初始
化；2：游戏进行中
    CSprite *gameBegin = new CSprite("GameBegin");
    const char *strHP[6] = {"", "Y", "YY", "YYY", "YYYY", "YYYYY"};
public:
    CGameMain();           //构造函数
    ~CGameMain();          //析构函数

    // Get 方法
    int GetGameState() { return m_iGameState; }

    // Set 方法
    void SetGameState(const int iState) { m_iGameState = iState; }

    // 游戏主循环等
    void GameMainLoop(float fDeltaTime);
    void GameInit();
    void GameRun(float fDeltaTime);
    void GameEnd();

    // 事件响应
    void OnKeyDown(const int iKey, const bool bAltPress, const bool bShiftPress,
const bool bCtrlPress);
    void OnKeyUp(const int iKey);
    void OnSpriteColSprite(const char *szSrcName, const char *szTarName);
    void OnSpriteColWorldLimit(const char *szName, const int iColSide);
};

//////////////////////////////////////////////////////////////////////////
//////
//
extern CGameMain    g_GameMain;

#endif // _LESSON_X_H_
```

源文件

```
//////////////////////////////////////////////////////////////////////////
//////
//
//
//
```

```
//
////////////////////////////////////////////////////////////////////////////
//////
#include <Stdio.h>
#include "CommonClass.h"
#include "LessonX.h"
#include "SpriteFactory.h"
#include "PlaneGame.h"
////////////////////////////////////////////////////////////////////////////
/////
//
//
CGameMain       g_GameMain;

//=========================================================================
=====
//
// 大体的程序流程为：GameMainLoop 函数为主循环函数，在引擎每帧刷新屏幕图像之后，都
会被调用一次。

//=========================================================================
=====
//
// 构造函数
CGameMain::CGameMain()
{
}
//=========================================================================
=====
//
// 析构函数
CGameMain::~CGameMain()
{
    delete gameBegin;
}

//=========================================================================
=====
//
// 游戏主循环，此函数将被不停的调用，引擎每刷新一次屏幕，此函数即被调用一次
// 用以处理游戏的开始、进行中、结束等各种状态.
// 函数参数 fDeltaTime : 上次调用本函数到此次调用本函数的时间间隔，单位：秒
void CGameMain::GameMainLoop( float fDeltaTime )
{
```

```
    switch( GetGameState() )
    {
        // 初始化游戏，清空上一局相关数据
    case 1:
        GameInit();
        SetGameState(2); // 初始化之后，将游戏状态设置为进行中
        break;

        // 游戏进行中，处理各种游戏逻辑
    case 2:
        GameRun(fDeltaTime);
        break;

        // 游戏即将结束
    case 3:
        SetGameState(0);
        GameEnd();
        break;
        // 游戏结束/等待按空格键开始
    case 0:
    default:
        break;
    };
}
//=============================================================================
====
//
// 每局开始前进行初始化，清空上一局相关数据
void CGameMain::GameInit()
{
    gameBegin->SetSpriteVisible(false);
    PlaneGame *game = PlaneGame::GetPlaneGame();
    game->SetEnemyPlaneCreateTime(0.5f);
    game->SetEnemyPlaneUsedTime(0.0f);
    Player *player = (Player
*)SpriteFactory::GetSprite(EnumDefine::PLAYER_PLANE);
    game->SetPlayer(player);
    if (game->GetTextScore() == NULL)
        game->SetTextScore(new CTextSprite("txtScore"));
    if (game->GetTextHP() == NULL)
        game->setTextHP(new CTextSprite("txtHP"));
    game->GetTextScore()->SetTextValue(0);
    game->GetTextHP()->SetTextString(strHP[5]);
}
```

```
//=============================================================================
====
//
// 每局游戏进行中
void CGameMain::GameRun(float fDeltaTime)
{
    PlaneGame *game = PlaneGame::GetPlaneGame();
    game->PlayerFire(fDeltaTime);
    game->CreateNewEnemyPlane(fDeltaTime);
}
//=============================================================================
====
//
// 本局游戏结束
void CGameMain::GameEnd()
{
    gameBegin->SetSpriteVisible(true);
    PlaneGame *game = PlaneGame::GetPlaneGame();
    game->DestroyPlayer();
    game->DestroyAllSprites();
}

// 精灵碰撞精灵
void CGameMain::OnSpriteColSprite(const char *szSrcName, const char
*szTarName)
{
    if (m_iGameState == 2)
    {
        PlaneGame *game = PlaneGame::GetPlaneGame();
        // 子弹碰撞敌机
        if (strstr(szSrcName, "bullet") != NULL && strstr(szTarName, "enemy") !=
NULL)
        {
            auto &spriteList = game->GetSpriteList();
            // 找到发生碰撞的子弹和敌机
            auto itBullet = spriteList.find(szSrcName);
            auto itEnemy = spriteList.find(szTarName);
            if (itBullet != spriteList.end() && itEnemy != spriteList.end())
            {
                Bullet *bullet = (Bullet *)itBullet->second;
                EnemyPlane *enemy = (EnemyPlane *)itEnemy->second;
                enemy->SetiHP(enemy->GetiHP() - bullet->GetiDamage());
                // 如果敌机血量为 0 则销毁敌机
                if (enemy->GetiHP() <= 0)
```

```
                {
                    // 增加分数

game->GetPlayer()->SetiScore(game->GetPlayer()->GetiScore() +
enemy->GetiValue());

game->GetTextScore()->SetTextValue(game->GetPlayer()->GetiScore());
                    // 播放爆炸动画
                    game->PlayExplodeAnimation(enemy->GetSpritePositionX(),
enemy->GetSpritePositionY(), enemy->GetiType());
                    // 销毁敌机
                    enemy->DeleteSprite();
                    delete enemy;
                    spriteList.erase(itEnemy);
                }
                // 销毁子弹
                bullet->DeleteSprite();
                delete bullet;
                spriteList.erase(itBullet);
            }
        }
        // 玩家飞机碰撞敌机
        else if (strstr(szSrcName, "player") != NULL && strstr(szTarName,
"enemy") != NULL)
        {
            // 找到发生碰撞的敌机
            auto &spriteList = game->GetSpriteList();
            auto itEnemy = spriteList.find(szTarName);
            if (itEnemy != spriteList.end())
            {
                EnemyPlane *enemy = (EnemyPlane *)itEnemy->second;
                // 增加分数
                game->GetPlayer()->SetiScore(game->GetPlayer()->GetiScore() +
enemy->GetiValue());

game->GetTextScore()->SetTextValue(game->GetPlayer()->GetiScore());
                // 播放爆炸动画
                game->PlayExplodeAnimation(enemy->GetSpritePositionX(),
enemy->GetSpritePositionY(), enemy->GetiType());
                // 销毁敌机
                enemy->DeleteSprite();
                delete enemy;
                spriteList.erase(itEnemy);
                game->GetPlayer()->SetiHP(game->GetPlayer()->GetiHP() - 1);
```

```
            game->GetTextHP()->SetTextString(strHP[game->GetPlayer()->GetiHP()]);
                    // 如果玩家飞机血量为 0 则进入游戏即将结束的状态
                    if (game->GetPlayer()->GetiHP() == 0)
                        m_iGameState = 3;
                }
            }
        }
    }

    // 精灵碰撞世界边界
    void CGameMain::OnSpriteColWorldLimit(const char *szName, const int iColSide)
    {
        if (m_iGameState == 2)
        {
            // 找到碰撞的精灵
            auto &spriteList = PlaneGame::GetPlaneGame()->GetSpriteList();
            auto it = spriteList.find(szName);
            if (it != spriteList.end())
            {
                // 销毁精灵
                it->second->DeleteSprite();
                delete it->second;
                spriteList.erase(it);
            }
        }
    }

    void CGameMain::OnKeyDown(const int iKey, const bool bAltPress, const bool
    bShiftPress, const bool bCtrlPress)
    {
        // 空格键开始游戏
        if (m_iGameState == 0 && iKey == KeyCodes::KEY_SPACE)
            m_iGameState = 1;
        else if (m_iGameState == 2)
        {
            Player *player = PlaneGame::GetPlaneGame()->GetPlayer();
            // WSAD 控制玩家飞机移动方向
            switch (iKey)
            {
            case KEY_W:
                player->SetfSpeedY(-40.0f);
                player->SetfSpeedX(0.0f);
                break;
```

```
        case KEY_S:
            player->SetfSpeedY(40.0f);
            player->SetfSpeedX(0.0f);
            break;
        case KEY_A:
            player->SetfSpeedX(-40.0f);
            player->SetfSpeedY(0.0f);
            break;
        case KEY_D:
            player->SetfSpeedX(40.0f);
            player->SetfSpeedY(0.0f);
            break;
        }
    }
}

void CGameMain::OnKeyUp(const int iKey)
{
    if (m_iGameState == 2)
    {
        Player *player = PlaneGame::GetPlaneGame()->GetPlayer();
        // 停止移动玩家飞机
        switch (iKey)
        {
        case KEY_W:
        case KEY_S:
            player->SetfSpeedY(0.0f);
            break;
        case KEY_A:
        case KEY_D:
            player->SetfSpeedX(0.0f);
            break;
        }
    }
}
```