飞机击中UFO

子弹命中UFO时，删除该子弹，而子弹攻击达到一定次数时，UFO也要删除并重生，首先要设立伤害系统，使得子弹有伤害，飞机有生命值。

首先修改子弹的结构体，为其添加伤害属性，并为飞机与UFO增加生命值属性，在子弹的初始化函数中，为伤害属性赋值。

```
//定义子弹的结构体，组成链表
typedef struct bullet
{
    float x, y;
    float vx, vy;
    int damage;//伤害
    int isExist;//判断子弹是否需要删除
    struct bullet* pnext;//指向下一个子弹节点的指针
}list;//给结构体起一个别名list,头结点可以表示列表的信息
```

飞行器增加生命

```
//定义飞机的结构体
struct aircraft
{
    int x ;
    int y;
    int width;
    int height;
    int speed;
    int life;
    int new_born_flg;
};
aircraft plane, ufoa, ufob, ufoc;
```

飞行器的初始化增加生命

```

//飞机数据初始化
void dataInit()
{
    plane = { 150,150,80,80,10,100,1 };
    ufoa = { 0,0 ,300,150,2,1000,1 };
    ufob = { 350,0,150,50,4,150,1 };
    ufoc = { 450,200 ,100,60,10,100,1 };

}
```

发射子弹的时候，同时赋予伤害值

```
//飞机发射的子弹，增加一个节点
list* creatPlaneBullet(float vx, float vy)
{
    p->damage = 50;
```

```
}

//ufoa发射的子弹，增加一个节点
list* creatUFOA_Bullet(float vx, float vy)
{
    p->damage = 10;
}

//ufoB发射的子弹，增加一个节点
list* creatUFOB_Bullet(float vx, float vy)
{

    p->damage = 20;

}
```

编写函数bulletHitJudge（子弹击中判断），判断子弹是否击中UFO，击中UFO后，UFO减少相应的生命值，并清除子弹存在标记。 此函数的参数为指向飞行器的指针，因此在函数内部就可以修改某个飞行器生命值属性。

```
//判断飞机的子弹是否击中UFO，执行相应的加分与减命的操作
void bulletHitUFO(aircraft* tmp)
{
    for (list* cur = plane_bullet_list; cur != NULL; cur = cur->pnext)
    {
        //子弹在UFO的矩形图片内时，认为击中
        if ((cur->x > tmp->x) && (cur->x < tmp->x + tmp->width))
        {
            if ((cur->y > tmp->y) && (cur->y < tmp->y + tmp->height))
            {
                tmp->life -= cur->damage;//飞行器生命值 - 子弹的伤害
                cur->isExist = 0;  //清除子弹的存在标记
            }
        }
    }
}
```

在绘制子弹函数“showBullet”中调用该函数，判断子弹是否发生碰撞：

```
//绘制发出来的子弹
void showBullet()
{
    ...
    bulletHitUFO(&ufoa);//判断子弹是否击中，更改生命和子弹状态
    ...
    bulletHitUFO(&ufob);//判断子弹是否击中，更改生命和子弹状态
    ...
    bulletHitUFO(&ufoc);//判断子弹是否击中，更改生命和子弹状态
    ...
}
```

运行代码，已经能够击中UFO了，子弹也会被消除掉.

“bulletHitUFO”函数已经将UFO的生命值进行了更改，编写2个函数，aircraftReborn用于标记飞行器的重生标记，飞行器重生；aircraftLifeJudge用于判断飞行器的生命值，如果生命值小于0，则调用aircraftReborn函数，使飞行器重生。

```
//飞行器重生  参数为飞行器的地址，以及重生的血量
void aircraftReborn(aircraft* tmp, int life)
{
    tmp->new_born_flg = 1;
    tmp->life = life;
}
//飞行器生存判断
void aircraftLifeJudge()
{
    if (ufoa.life <= 0)
    {
        //aircraftReborn(&ufoa, ufoa.life);//这是错误写法
        aircraftReborn(&ufoa, UFOA_LIFE);
    }
    if (ufob.life <= 0)
    {
        aircraftReborn(&ufob, UFOB_LIFE);
    }
    if (ufoc.life <= 0)
    {
        aircraftReborn(&ufoc, UFOC_LIFE);
    }
    if (plane.life <= 0)
    {
       // game_over = 1;
    }
}
```

定义初始血量

```
int PLANE_LIFE = 100;
int UFOA_LIFE = 1500;
int UFOB_LIFE = 150;
int UFOC_LIFE = 100;
```

主函数中进行生存判断

```
int _tmain(int argc, _TCHAR* argv[])
{
    dataInit();//初始化所有飞机，设置飞机坐标
    initgraph(WIDTH, HEIGHT);// 创建绘图窗口
    loadRes();
    clock_t start_time;
    BeginBatchDraw();
    srand(time(NULL));
    while(1)
    {
        ufoaMove();
        ufobMove();
        ufocMove();
        start_time = clock();
        getInput(); //获取输入
```

```
        dealInput();//调整飞机位置
        showAircraft();//显示飞机
        showBullet();
        putimage(0, 0, &temp_img);
        aircraftLifeJudge();//飞机生存判断
        ctrlFps(start_time);//控制Fps在60左右
        FlushBatchDraw();
    }
    EndBatchDraw();
    _getch();
    closegraph();

    return 0;
}
```

运行代码

```
#include "stdafx.h"

#define HEIGHT 720  // 游戏画面尺寸
#define WIDTH 1280
#define PI 3.1415926

//定义飞机的结构体
struct aircraft
{
    int x ;
    int y;
    int width;
    int height;
    int speed;
    int life;
    int new_born_flg;
};
aircraft plane, ufoa, ufob, ufoc;

//定义子弹的结构体, 组成链表
typedef struct bullet
{
    float x, y;
    float vx, vy;
    int damage;//伤害
    int isExist;//判断子弹是否需要删除
    struct bullet* pnext;//指向下一个子弹节点的指针
}list;//给结构体起一个别名list,头结点可以表示列表的信息

list* plane_bullet_list = NULL; // 飞机子弹列表的头节点
list* ufob_bullet_list = NULL;  //UFOB的子弹列表的开头
list* ufoa_bullet_list = NULL;  //UFOA的子弹列表的开头

IMAGE img_bk, img_plane,temp_img,
    img_ufoa,img_ufob,img_ufoc,
    img_plane_bullet,img_ufoa_bullet,img_ufob_bullet;

int PLANE_LIFE = 100;
int UFOA_LIFE = 1500;
```

```
int UFOB_LIFE = 150;
int UFOC_LIFE = 100;


//按键输入的枚举列表
enum GAMEINPUT
{
    NOINPUT = 0X0,
    UPINPUT = 0X1,
    DOWNINPUT = 0X2,
    LEFTINPUT = 0X4,
    RIGHTINPUT = 0X8,
    FIREINPUT = 0X10
};
int input = NOINPUT;//判断输入变量
int speed = 10;

void dataInit();
void loadRes();
void drawAlpha(IMAGE* dstimg, int x, int y, IMAGE* srcimg);
void showAircraft();
void getInput();
void dealInput();
void ctrlFps(int start_time);
void ufoaMove();
void ufobMove();
void ufocMove();
void showBullet();
void listPushBack(list** pplist, list* newNode);
list* creatPlaneBullet(float vx, float vy);
list* creatUFOA_Bullet(float vx, float vy);
list* creatUFOB_Bullet(float vx, float vy);
void listChangeXY(list** pplist);
void listRemoveNode(list** pplist);
void bulletHitUFO(aircraft* tmp);
void aircraftReborn(aircraft* tmp, int life);
void aircraftLifeJudge();

int _tmain(int argc, _TCHAR* argv[])
{
    dataInit();//初始化所有飞机，设置飞机坐标
    initgraph(WIDTH, HEIGHT);// 创建绘图窗口
    loadRes();
    clock_t start_time;
    BeginBatchDraw();
    srand(time(NULL));
    while(1)
    {
        ufoaMove();
        ufobMove();
        ufocMove();
        start_time = clock();
        getInput(); //获取输入
        dealInput();//调整飞机位置
        showAircraft();//显示飞机
        showBullet();
        putimage(0, 0, &temp_img);
        aircraftLifeJudge();//飞机生存判断
```

```
        ctrlFps(start_time);//控制Fps在60左右
        FlushBatchDraw();
    }
    EndBatchDraw();
    _getch();
    closegraph();

    return 0;
}

//飞机数据初始化
void dataInit()
{
    plane = { 150,150,80,80,10,100,1 };
    ufoa = { 0,0 ,300,150,2,1000,1 };
    ufob = { 350,0,150,50,4,150,1 };
    ufoc = { 450,200 ,100,60,10,100,1 };
}

//以相对路径载入所有素材
void loadRes()
{
    loadimage(&img_bk, _T("res\\background.png"));
    loadimage(&temp_img, _T("res\\background.png"));
    loadimage(&img_plane, _T("res\\plane.png"));
    loadimage(&img_ufoa, _T("res\\ufoa.png"));
    loadimage(&img_ufob, _T("res\\ufob.png"));
    loadimage(&img_ufoc, _T("res\\ufoc.png"));
    loadimage(&img_plane_bullet, _T("res\\plane_bullet.png"));
    loadimage(&img_ufoa_bullet, _T("res\\ufoa_bullet.png"));
    loadimage(&img_ufob_bullet, _T("res\\ufob_bullet.png"));
}

// 根据透明度绘图
void drawAlpha(IMAGE *dstimg, int x, int y, IMAGE *srcimg)
{
    // 变量初始化
    DWORD *dst = GetImageBuffer(dstimg);
    DWORD *src = GetImageBuffer(srcimg);
    int src_width = srcimg->getwidth();
    int src_height = srcimg->getheight();
    int dst_width = (dstimg == NULL ? getwidth() : dstimg->getwidth());
    int dst_height = (dstimg == NULL ? getheight() : dstimg->getheight());

    // 计算贴图的实际长宽
    int iwidth = (x + src_width > dst_width) ? dst_width - x : src_width;
// 处理超出右边界
    int iheight = (y + src_height > dst_height) ? dst_height - y : src_height;
// 处理超出下边界
    if (x < 0) { src += -x;             iwidth -= -x;   x = 0; }
// 处理超出左边界
    if (y < 0) { src += src_width * -y; iheight -= -y;  y = 0; }
// 处理超出上边界

    // 修正贴图起始位置
    dst += dst_width * y + x;
    // 实现透明贴图
    for (int iy = 0; iy < iheight; ++iy)
```

```
    {
        for (int i = 0; i < iwidth; ++i)
        {
            int sa = ((src[i] & 0xff000000) >> 24);//获取阿尔法值
            if (sa != 0)//假如是完全透明就不处理
            if (sa == 255)//假如完全不透明则直接拷贝
            dst[i] = src[i];
            else//真正需要阿尔法混合计算的图像边界才进行混合
            dst[i] = ((((src[i] & 0xff0000) >> 16) + ((dst[i] & 0xff0000) >> 16)
* (255 - sa) / 255) << 16) |((((src[i] & 0xff00) >> 8) + ((dst[i] & 0xff00) >>
8) * (255 - sa) / 255) << 8) | ((src[i] & 0xff) + (dst[i] & 0xff) * (255 - sa) /
255);
        }
        dst += dst_width;
        src += src_width;
    }
}
//绘制所有的飞机
void showAircraft()
{
    drawAlpha(&temp_img,0, 0, &img_bk);
    drawAlpha(&temp_img,plane.x, plane.y, &img_plane);
    drawAlpha(&temp_img,ufoa.x, ufoa.y, &img_ufoa);
    drawAlpha(&temp_img,ufob.x, ufob.y, &img_ufob);
    drawAlpha(&temp_img,ufoc.x, ufoc.y, &img_ufoc);
}

//同时获取多个输入，操作飞机
void getInput()
{
    int reload_time = 100;//飞机发子弹的间隔时间，单位ms
    static int fire_start = 0;//静态变量，储存开火的时间
    int tmp = clock();
    if (GetAsyncKeyState('W') & 0x8000)
    {
        input |= UPINPUT;
    }
    if (GetAsyncKeyState('S') & 0x8000)
    {
        input |= DOWNINPUT;
    }
    if (GetAsyncKeyState('A') & 0x8000)
    {
        input |= LEFTINPUT;
    }
    if (GetAsyncKeyState('D') & 0x8000)
    {
        input |= RIGHTINPUT;
    }
    if (GetAsyncKeyState('K') & 0x8000)
    {
        if (tmp - fire_start >= reload_time)
        {
            input |= FIREINPUT;
            fire_start = tmp;
        }
    }
}
```

```
//同时处理多个输入，调整飞机的位置
void dealInput()
{
    if ((input & UPINPUT) && (plane.y >= 0))
    {
        plane.y -= speed;
    }
    if ((input & DOWNINPUT) && (plane.y <= HEIGHT - 120))
    {
        plane.y += speed;
    }
    if ((input & LEFTINPUT) && (plane.x >= 0))
    {
        plane.x -= speed;
    }
    if ((input & RIGHTINPUT) && (plane.x <= WIDTH - 120))
    {
        plane.x += speed;
    }
    if (input & FIREINPUT)
    {
        listPushBack(&plane_bullet_list, creatPlaneBullet(0, -20));//水平方向无速
度，垂直向上速度20
        listPushBack(&plane_bullet_list, creatPlaneBullet(-10, -17.32));//30°散开
20*cos30约等于17.32
        listPushBack(&plane_bullet_list, creatPlaneBullet(10, -17.32));
    }

    input = NOINPUT;
}

//根据从开始到现在的时间，设置休眠的时间
void ctrlFps(int start_time)
{
    clock_t running_time = clock() - start_time;
    if((13 - running_time) >= 0)//防止睡眠函数使用负数
      Sleep(13 - running_time);//动态睡眠
    TCHAR time_text[50];
    int FPS = 1000 / (clock() - start_time);
    _stprintf_s(time_text, _T("FPS:%d"), FPS);
    settextstyle(60, 0, _T("黑体")); //为了演示，显示fps字体大小不宜太大
    outtextxy(0, 0, time_text);
}

//UFOA缓慢向前移动，到达一定的位置回去,转圈发射子弹
void ufoaMove()
{
    static int cnt = 0;
    static int dir = 1;//前进
    int ufoa_reload_cnt = 80;//发射子弹的计数器，数值越小发射子弹越快
    int ufoa_fire_num = 20;  //UFOA共发射多少路子弹
    if (1 == ufoa.new_born_flg)//新出生的ufoa ，重置x,y的坐标
    {
        ufoa.new_born_flg = 0;
        ufoa.x = rand() % (WIDTH - ufoa.width);
        ufoa.y = -50 ;
    }
```

```
    if(ufoa.y > 200)//纵坐标大于300，改为后退
    {
        dir = 0; //后退

    }
    else if(ufoa.y < -150)
    {
        dir = 1;
        ufoa.new_born_flg = 1;
    }

    if(dir == 1) //前进
        ufoa.y += ufoa.speed;
    else  //后退
        ufoa.y -= ufoa.speed;

    //cnt到达指定的间隔以后，转圈发射子弹
    if (++cnt % ufoa_reload_cnt == 0)
    {
        for (int i = 0; i < ufoa_fire_num; i++)
        {
            float angle = i * 2 * PI / ufoa_fire_num;
            float vx = 4 * sin(angle);
            float vy = 4 * cos(angle);
            listPushBack(&ufoa_bullet_list, creatUFOA_Bullet(vx, vy));
        }
    }
    if (cnt > 999999) cnt = 0;
}


//UFOB左右快速移动，慢速向下移动
void ufobMove()
{
    static int step = ufob.speed;//spep表示方向与速度
    static int cnt = 0;
    int ufob_reload_cnt = 60;//UFOb  发射子弹的计数器，数值越小发射子弹越快
    if (1 == ufob.new_born_flg)//新出生的ufob ，重置x,y的坐标
    {
        ufob.new_born_flg = 0;
        ufob.x = rand() % (WIDTH - ufob.width);
        ufob.y = -ufob.height;
    }
    //水平撞墙返回
    if ((ufob.x <= 0) || (ufob.x + ufob.width >= WIDTH))
        step = -step;
    ufob.x += step;
    ufob.y++;
    //超出下边界，重生，y坐标重置，x坐标随机
    if (ufob.y >= HEIGHT)
        ufob.new_born_flg = 1;
    if (++cnt % ufob_reload_cnt == 0)
    {
        listPushBack(&ufob_bullet_list, creatUFOB_Bullet(0, 5));
    }
    if (cnt > 999999) cnt = 0;//当计数器超过999999 进行清零
}
```

```
//ufoc撞向飞机
void ufocMove()
{
    static float dist_x = 0, dist_y = 0;//ufoc出生时，记录与飞机的横竖距离
    static float tmp_x = 0, tmp_y = 0; //储存x,y坐标的临时变量，浮点型方便计算
    static float vx = 0, vy = 0;
    float step = 1000 / ufoc.speed;       //调整UFO速度
    if (1 == ufoc.new_born_flg)
    {
        ufoc.new_born_flg = 0;
        tmp_x = rand() % (WIDTH - ufoc.width);
        tmp_y = -ufoc.height;
        dist_x = plane.x - tmp_x;
        dist_y = plane.y - tmp_y;
        vx = dist_x / step;
        vy = dist_y / step;
    }
    tmp_x += vx;
    tmp_y += vy;
    ufoc.x = (int)(tmp_x + 0.5);
    ufoc.y = (int)(tmp_y + 0.5);
    //边界判断，可以超出画面，但不超出太多
    if (ufoc.x < -ufoc.width)
        ufoc.new_born_flg = 1;
    else if (ufoc.x > WIDTH+ufoc.width)
        ufoc.new_born_flg = 1;
    //超出下边界，重生，y坐标重置，x坐标随机
    if (ufoc.y >= HEIGHT)
        ufoc.new_born_flg = 1;
}


//在某链表尾部插入一个数据
void listPushBack(list** pplist, list* newNode)
{
    if (*pplist == NULL)//如果链表为空，那么新增的节点就是第一个
    {
        *pplist = newNode;
        return;
    }
    list* cur = *pplist;
    while (cur->pnext != NULL)//找到最后一个节点
    {
        cur = cur->pnext;
    }
    cur->pnext = newNode;//插入新的节点
}

//飞机发射的子弹，增加一个节点
list* creatPlaneBullet(float vx, float vy)
{
    list* p = (list*)malloc(sizeof(list));
    p->x = plane.x + plane.width / 2+10;//飞机头部的位置
    p->y = plane.y;
    p->vx = vx;
    p->vy = vy;//速度
    p->damage = 50;
    p->isExist = 1;
```

```
    p->pnext = NULL;
    return p;
}

//ufoa发射的子弹，增加一个节点
list* creatUFOA_Bullet(float vx, float vy)
{
    list* p = (list*)malloc(sizeof(list));
    p->x = ufoa.x + ufoa.width / 2;//中间
    p->y = ufoa.y + ufoa.height;   //下方
    p->vx = vx;
    p->vy = vy;//速度
    p->damage = 10;
    p->isExist = 1;
    p->pnext = NULL;
    return p;
}

//ufoB发射的子弹，增加一个节点
list* creatUFOB_Bullet(float vx, float vy)
{
    list* p = (list*)malloc(sizeof(list));
    p->x = ufob.x + ufob.width / 2;//中间
    p->y = ufob.y + ufob.height;   //下方
    p->vx = vx;
    p->vy = vy;//速度
    p->damage = 20;
    p->isExist = 1;
    p->pnext = NULL;
    return p;
}
//修改某链中所有节点的坐标。
void listChangeXY(list** pplist)
{
    if (*pplist == NULL)//如果链表为空，那么新增的节点就是第一个
        return;
    list* cur = *pplist;//curret指向第一个节点
    while (cur != NULL)//遍历链表
    {
        cur->x += cur->vx;
        cur->y += cur->vy;
        //判断子弹是否离开视野
        if ((cur->y < -20)||(cur->y > HEIGHT)||(cur->x > WIDTH)||(cur->y < -20))
            cur->isExist = 0;
        cur = cur->pnext;//指向下一个节点
    }
}
//删除链表中isExist为0的节点
void listRemoveNode(list** pplist)
{
    if (*pplist == NULL)//如果链表为空，就没有可删除的节点了
        return;
    list* cur = *pplist;//curret先指向第一个节点
    list* prev = NULL;  //previous指向上一个节点的指针
    while (cur != NULL)//遍历链表
    {
        if (cur->isExist == 0)//判断节点是否需要删除
        {
```

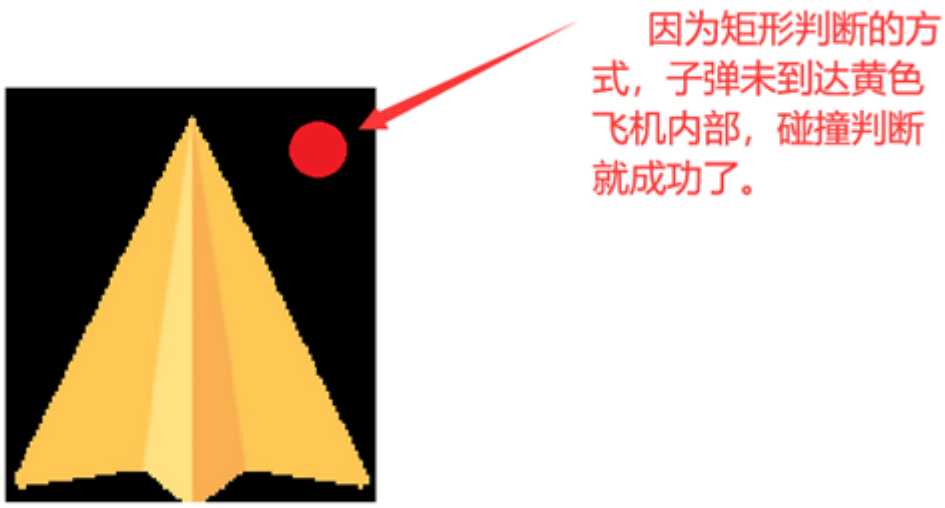
```
            if (*pplist == cur)//如果删除的是第一个节点
            {
                *pplist = cur->pnext;  //更改链表的地址，让下一个节点作为头结点 ，如果没有节点，则链表为空
                free(cur);             //释放当前节点（第一个节点的）空间
                cur = *pplist;         //让cur指向下一个节点
            }
            else
            {
                prev->pnext = cur->pnext;  //记录下一个节点的地址
                free(cur);                 //释放当前节点空间
                cur = prev;                //当前节点变成前一个节点
            }
        }
        else //如果不需要删除节点，储存当前节点为前一个节点，然后指向下一个节点
        {
            prev = cur;
            cur = cur->pnext;
        }
    }
}

//绘制发出来的子弹
void showBullet()
{
        //判断飞机子弹是否击中，更改敌机生命和子弹状态
    bulletHitUFO(&ufoa);
    bulletHitUFO(&ufob);
    bulletHitUFO(&ufoc);

    //飞机发射的子弹
    listChangeXY(&plane_bullet_list);//计算子弹新的位置
    listRemoveNode(&plane_bullet_list);//超出视野或者击中飞行器的子弹删除掉
    for (list* cur = plane_bullet_list; cur != NULL; cur = cur->pnext)
    {
        drawAlpha(&temp_img,cur->x, cur->y, &img_plane_bullet);
    }

    //ufoa发出的子弹
    listChangeXY(&ufoa_bullet_list);//计算子弹新的位置
    listRemoveNode(&ufoa_bullet_list);//超出视野或者击中飞行器的子弹删除掉
    for (list* cur = ufoa_bullet_list; cur != NULL; cur = cur->pnext)
    {
        drawAlpha(&temp_img,cur->x-10, cur->y-10, &img_ufoa_bullet);
    }

    //ufob发出的子弹
    listChangeXY(&ufob_bullet_list);//计算子弹新的位置
    listRemoveNode(&ufob_bullet_list);//超出视野或者击中飞行器的子弹删除掉
    for (list* cur = ufob_bullet_list; cur != NULL; cur = cur->pnext)
    {
        drawAlpha(&temp_img,cur->x-15, cur->y-30, &img_ufob_bullet);
    }

}

//判断飞机的子弹是否击中UFO，执行相应的加分与减命的操作
void bulletHitUFO(aircraft* tmp)
```

```
{
    for (list* cur = plane_bullet_list; cur != NULL; cur = cur->pnext)
    {
        //子弹在UFO的矩形图片内时，认为击中
        if ((cur->x > tmp->x) && (cur->x < tmp->x + tmp->width))
        {
            if ((cur->y > tmp->y) && (cur->y < tmp->y + tmp->height))
            {
                tmp->life -= cur->damage;//飞行器生命值 - 子弹的伤害
                cur->isExist = 0;  //清除子弹的存在标记
            }
        }
    }
}

//飞行器重生  参数为飞行器的地址，以及重生的血量
void aircraftReborn(aircraft* tmp, int life)
{
    tmp->new_born_flg = 1;
    tmp->life = life;
}
//飞行器生存判断
void aircraftLifeJudge()
{
    if (ufoa.life <= 0)
    {
        //aircraftReborn(&ufoa, ufoa.life);//这是错误写法
        aircraftReborn(&ufoa, UFOA_LIFE);
    }
    if (ufob.life <= 0)
    {
        aircraftReborn(&ufob, UFOB_LIFE);
    }
    if (ufoc.life <= 0)
    {
        aircraftReborn(&ufoc, UFOC_LIFE);
    }
    if (plane.life <= 0)
    {
       // game_over = 1;
    }
}

/*
//判断UFO的子弹是否击中飞机，执行相应的加分与减命的操作,参数是UFO的子弹链表
void bulletHitPlane(list* bullet_list)
{
    for (list* cur = bullet_list; cur != NULL; cur = cur->pnext)
    {
        //子弹在飞机的矩形图片内时，再判断是否在飞机的三角形内，减少计算量
        if ((cur->x > plane.x) && (cur->x < plane.x + plane.width))
        {
            if ((cur->y > plane.y) && (cur->y < plane.y + plane.height))
            {
                triangle tri = getPlaneTriangle();//获取飞机的三角形参数
                if (isPointInTriangle(tri, cur->x, cur->y))//子弹与飞机相撞
                {
                    plane.life -= cur->damage;
```

```
                    cur->isExist = 0;
                }
            }
        }
    }
}
*/
```

飞机中弹

使用矩形碰撞判定飞机是否中弹就不够准确。飞机实际体积是三角形，判断子弹是否击中飞机，也就是判断子弹是否在飞机这个三角形之内。



新建向量与三角形的结构体

```
//向量的结构体
struct vector
{
    float x, y;
};
//三角形的结构体
struct triangle
{
    float ax, ay, bx, by, cx, cy;
};
```

三角形相关的辅助函数

```
//2个向量的叉乘，结果仍然是向量，正负可以表示方向
float crossProduct(vector a, vector b)
{
    float tmp = a.x * b.y - a.y * b.x;
    return tmp;
}
//判断点(x,y)是否在三角形内
int isPointInTriangle(triangle tri, float x, float y)
{
    vector pa = getVector(tri.ax, tri.ay, x, y);//向量pa，是a-p
    vector pb = getVector(tri.bx, tri.by, x, y);//向量pb，是b-p
```

```
    vector pc = getVector(tri.cx, tri.cy, x, y);//向量pc，是c-p
    float t1 = crossProduct(pa, pb);
    float t2 = crossProduct(pb, pc);
    float t3 = crossProduct(pc, pa);
    return t1 * t2 >= 0 && t1 * t3 >= 0 && t2 * t3 >= 0;
}
//传入2组坐标，生成向量
vector getVector(float x1, float y1, float x2, float y2)
{
    vector tmp;
    tmp.x = x2 - x1;
    tmp.y = y2 - y1;
    return tmp;
}
//根据飞机图片的x 与 y坐标 来构建一个三角形，用于碰撞判断
triangle getPlaneTriangle()
{
    triangle tmp;
    //a 是最上边的点，b 是右下， c是左下。
    tmp.ax = plane.x + plane.width / 2;
    tmp.ay = plane.y;
    tmp.bx = plane.x + plane.width;
    tmp.by = plane.y + plane.height;
    tmp.cx = plane.x;
    tmp.cy = plane.y + plane.height;
    return tmp;
}
```

判断UFO的子弹是否击中飞机

```
//判断UFO的子弹是否击中飞机，执行相应的加分与减命的操作,参数是UFO的子弹链表
void bulletHitPlane(list* bullet_list)
{
    for (list* cur = bullet_list; cur != NULL; cur = cur->pnext)
    {
        //子弹在飞机的矩形图片内时，再判断是否在飞机的三角形内，减少计算量
        if ((cur->x > plane.x) && (cur->x < plane.x + plane.width))
        {
            if ((cur->y > plane.y) && (cur->y < plane.y + plane.height))
            {
                triangle tri = getPlaneTriangle();//获取飞机的三角形参数
                if (isPointInTriangle(tri, cur->x, cur->y))//子弹与飞机相撞
                {
                    plane.life -= cur->damage;
                    cur->isExist = 0;
                }
            }
        }
    }
}
```

在显示子弹中调用

```
//绘制发出来的子弹
void showBullet()
{
        //判断飞机子弹是否击中，更改敌机生命和子弹状态
    bulletHitUFO(&ufoa);

    //判断敌机子弹是否击中，更改飞机生命和子弹状态
    bulletHitPlane(ufoa_bullet_list);
    bulletHitPlane(ufob_bullet_list);
```

演示，飞机能够被击中

```
#include "stdafx.h"

#define HEIGHT 720  // 游戏画面尺寸
#define WIDTH 1280
#define PI 3.1415926

//定义飞机的结构体
struct aircraft
{
    int x ;
    int y;
    int width;
    int height;
    int speed;
    int life;
    int new_born_flg;
};
aircraft plane, ufoa, ufob, ufoc;

//向量的结构体
struct vector
{
    float x, y;
};
//三角形的结构体
struct triangle
{
    float ax, ay, bx, by, cx, cy;
};

//定义子弹的结构体，组成链表
typedef struct bullet
{
    float x, y;
    float vx, vy;
    int damage;//伤害
    int isExist;//判断子弹是否需要删除
    struct bullet* pnext;//指向下一个子弹节点的指针
}list;//给结构体起一个别名list,头结点可以表示列表的信息

list* plane_bullet_list = NULL; // 飞机子弹列表的头节点
list* ufob_bullet_list = NULL;  //UFOB的子弹列表的开头
```

```
list* ufoa_bullet_list = NULL;  //UFOA的子弹列表的开头

IMAGE img_bk, img_plane,temp_img,
    img_ufoa,img_ufob,img_ufoc,
    img_plane_bullet,img_ufoa_bullet,img_ufob_bullet;

int PLANE_LIFE = 100;
int UFOA_LIFE = 1500;
int UFOB_LIFE = 150;
int UFOC_LIFE = 100;


//按键输入的枚举列表
enum GAMEINPUT
{
    NOINPUT = 0X0,
    UPINPUT = 0X1,
    DOWNINPUT = 0X2,
    LEFTINPUT = 0X4,
    RIGHTINPUT = 0X8,
    FIREINPUT = 0X10
};
int input = NOINPUT;//判断输入变量
int speed = 10;

void dataInit();
void loadRes();
void drawAlpha(IMAGE* dstimg, int x, int y, IMAGE* srcimg);
void showAircraft();
void getInput();
void dealInput();
void ctrlFps(int start_time);
void ufoaMove();
void ufobMove();
void ufocMove();
void showBullet();
void listPushBack(list** pplist, list* newNode);
list* creatPlaneBullet(float vx, float vy);
list* creatUFOA_Bullet(float vx, float vy);
list* creatUFOB_Bullet(float vx, float vy);
void listChangeXY(list** pplist);
void listRemoveNode(list** pplist);

void aircraftReborn(aircraft* tmp, int life);
void aircraftLifeJudge();

//处理三角形相关函数
vector getVector(float x1, float y1, float x2, float y2);//生成向量
float crossProduct(vector a, vector b);//2个向量的叉乘
int isPointInTriangle(triangle tri, float x, float y);//判断点(x,y)是否在三角形内
triangle getPlaneTriangle();//构建一个三角形，用于碰撞判断

//碰撞相关函数
void bulletHitPlane(list* bullet_list);//判断UFO的子弹是否击中飞机
void bulletHitUFO(aircraft* tmp);//判断飞机的子弹是否击中UFO
void ufoCrash(aircraft* tmp);//UFO是否与飞机碰撞
void ufosCrashJudge();//整个UFO是否与飞机碰撞
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    dataInit();//初始化所有飞机，设置飞机坐标
    initgraph(WIDTH, HEIGHT);// 创建绘图窗口
    loadRes();
    clock_t start_time;
    BeginBatchDraw();
    srand(time(NULL));
    while(1)
    {
        ufoaMove();
        ufobMove();
        ufocMove();
        start_time = clock();
        getInput(); //获取输入
        dealInput();//调整飞机位置
        showAircraft();//显示飞机
        showBullet();
        putimage(0, 0, &temp_img);
        aircraftLifeJudge();//飞机生存判断
        ctrlFps(start_time);//控制Fps在60左右
        FlushBatchDraw();
    }
    EndBatchDraw();
    _getch();
    closegraph();

    return 0;
}

//飞机数据初始化
void dataInit()
{
    plane = { 150,150,80,80,10,100,1 };
    ufoa = { 0,0 ,300,150,2,1000,1 };
    ufob = { 350,0,150,50,4,150,1 };
    ufoc = { 450,200 ,100,60,10,100,1 };
}

//以相对路径载入所有素材
void loadRes()
{
    loadimage(&img_bk, _T("res\\background.png"));
    loadimage(&temp_img, _T("res\\background.png"));
    loadimage(&img_plane, _T("res\\plane.png"));
    loadimage(&img_ufoa, _T("res\\ufoa.png"));
    loadimage(&img_ufob, _T("res\\ufob.png"));
    loadimage(&img_ufoc, _T("res\\ufoc.png"));
    loadimage(&img_plane_bullet, _T("res\\plane_bullet.png"));
    loadimage(&img_ufoa_bullet, _T("res\\ufoa_bullet.png"));
    loadimage(&img_ufob_bullet, _T("res\\ufob_bullet.png"));
}

// 根据透明度绘图
void drawAlpha(IMAGE *dstimg, int x, int y, IMAGE *srcimg)
{
    // 变量初始化
    DWORD *dst = GetImageBuffer(dstimg);
```

```
    DWORD *src = GetImageBuffer(srcimg);
    int src_width = srcimg->getwidth();
    int src_height = srcimg->getheight();
    int dst_width = (dstimg == NULL ? getwidth() : dstimg->getwidth());
    int dst_height = (dstimg == NULL ? getheight() : dstimg->getheight());

    // 计算贴图的实际长宽
    int iwidth = (x + src_width > dst_width) ? dst_width - x : src_width;
// 处理超出右边界
    int iheight = (y + src_height > dst_height) ? dst_height - y : src_height;
// 处理超出下边界
    if (x < 0) { src += -x;             iwidth -= -x;   x = 0; }
// 处理超出左边界
    if (y < 0) { src += src_width * -y; iheight -= -y;  y = 0; }
// 处理超出上边界

    // 修正贴图起始位置
    dst += dst_width * y + x;
    // 实现透明贴图
    for (int iy = 0; iy < iheight; ++iy)
    {
        for (int i = 0; i < iwidth; ++i)
        {
            int sa = ((src[i] & 0xff000000) >> 24);//获取阿尔法值
            if (sa != 0)//假如是完全透明就不处理
            if (sa == 255)//假如完全不透明则直接拷贝
            dst[i] = src[i];
            else//真正需要阿尔法混合计算的图像边界才进行混合
            dst[i] = ((((src[i] & 0xff0000) >> 16) + ((dst[i] & 0xff0000) >> 16)
* (255 - sa) / 255) << 16) |((((src[i] & 0xff00) >> 8) + ((dst[i] & 0xff00) >>
8) * (255 - sa) / 255) << 8) | ((src[i] & 0xff) + (dst[i] & 0xff) * (255 - sa) /
255);
        }
        dst += dst_width;
        src += src_width;
    }
}
//绘制所有的飞机
void showAircraft()
{
    drawAlpha(&temp_img,0, 0, &img_bk);
    drawAlpha(&temp_img,plane.x, plane.y, &img_plane);
    drawAlpha(&temp_img,ufoa.x, ufoa.y, &img_ufoa);
    drawAlpha(&temp_img,ufob.x, ufob.y, &img_ufob);
    drawAlpha(&temp_img,ufoc.x, ufoc.y, &img_ufoc);
}

//同时获取多个输入，操作飞机
void getInput()
{
    int reload_time = 100;//飞机发子弹的间隔时间，单位ms
    static int fire_start = 0;//静态变量，储存开火的时间
    int tmp = clock();
    if (GetAsyncKeyState('W') & 0x8000)
    {
        input |= UPINPUT;
    }
    if (GetAsyncKeyState('S') & 0x8000)
```

```
    {
        input |= DOWNINPUT;
    }
    if (GetAsyncKeyState('A') & 0x8000)
    {
        input |= LEFTINPUT;
    }
    if (GetAsyncKeyState('D') & 0x8000)
    {
        input |= RIGHTINPUT;
    }
    if (GetAsyncKeyState('K') & 0x8000)
    {
        if (tmp - fire_start >= reload_time)
        {
            input |= FIREINPUT;
            fire_start = tmp;
        }
    }
}

//同时处理多个输入，调整飞机的位置
void dealInput()
{
    if ((input & UPINPUT) && (plane.y >= 0))
    {
        plane.y -= speed;
    }
    if ((input & DOWNINPUT) && (plane.y <= HEIGHT - 120))
    {
        plane.y += speed;
    }
    if ((input & LEFTINPUT) && (plane.x >= 0))
    {
        plane.x -= speed;
    }
    if ((input & RIGHTINPUT) && (plane.x <= WIDTH - 120))
    {
        plane.x += speed;
    }
    if (input & FIREINPUT)
    {
        listPushBack(&plane_bullet_list, creatPlaneBullet(0, -20));//水平方向无速
度，垂直向上速度20
        listPushBack(&plane_bullet_list, creatPlaneBullet(-10, -17.32));//30°散开
20*cos30约等于17.32
        listPushBack(&plane_bullet_list, creatPlaneBullet(10, -17.32));
    }

    input = NOINPUT;
}

//根据从开始到现在的时间，设置休眠的时间
void ctrlFps(int start_time)
{
    clock_t running_time = clock() - start_time;
    if((13 - running_time) >= 0)//防止睡眠函数使用负数
      Sleep(13 - running_time);//动态睡眠
```

```
    TCHAR time_text[50];
    int FPS = 1000 / (clock() - start_time);
    _stprintf_s(time_text, _T("FPS:%d"), FPS);
    settextstyle(60, 0, _T("黑体")); //为了演示，显示fps字体大小不宜太大
    outtextxy(0, 0, time_text);
}

//UFOA缓慢向前移动，到达一定的位置回去,转圈发射子弹
void ufoaMove()
{
    static int cnt = 0;
    static int dir = 1;//前进
    int ufoa_reload_cnt = 80;//发射子弹的计数器，数值越小发射子弹越快
    int ufoa_fire_num = 20;  //UFOA共发射多少路子弹
    if (1 == ufoa.new_born_flg)//新出生的ufoa ，重置x,y的坐标
    {
        ufoa.new_born_flg = 0;
        ufoa.x = rand() % (WIDTH - ufoa.width);
        ufoa.y = -50 ;
    }
    if(ufoa.y > 200)//纵坐标大于300，改为后退
    {
        dir = 0; //后退

    }
    else if(ufoa.y < -150)
    {
        dir = 1;
        ufoa.new_born_flg = 1;
    }

    if(dir == 1) //前进
        ufoa.y += ufoa.speed;
    else  //后退
        ufoa.y -= ufoa.speed;

    //cnt到达指定的间隔以后，转圈发射子弹
    if (++cnt % ufoa_reload_cnt == 0)
    {
        for (int i = 0; i < ufoa_fire_num; i++)
        {
            float angle = i * 2 * PI / ufoa_fire_num;
            float vx = 4 * sin(angle);
            float vy = 4 * cos(angle);
            listPushBack(&ufoa_bullet_list, creatUFOA_Bullet(vx, vy));
        }
    }
    if (cnt > 999999) cnt = 0;
}


//UFOB左右快速移动，慢速向下移动
void ufobMove()
{
    static int step = ufob.speed;//spep表示方向与速度
    static int cnt = 0;
    int ufob_reload_cnt = 60;//UFOb  发射子弹的计数器，数值越小发射子弹越快
    if (1 == ufob.new_born_flg)//新出生的ufob ，重置x,y的坐标
```

```
    {
        ufob.new_born_flg = 0;
        ufob.x = rand() % (WIDTH - ufob.width);
        ufob.y = -ufob.height;
    }
    //水平撞墙返回
    if ((ufob.x <= 0) || (ufob.x + ufob.width >= WIDTH))
        step = -step;
    ufob.x += step;
    ufob.y++;
    //超出下边界，重生，y坐标重置，x坐标随机
    if (ufob.y >= HEIGHT)
        ufob.new_born_flg = 1;
    if (++cnt % ufob_reload_cnt == 0)
    {
        listPushBack(&ufob_bullet_list, creatUFOB_Bullet(0, 5));
    }
    if (cnt > 999999) cnt = 0;//当计数器超过999999 进行清零
}

//ufoc撞向飞机
void ufocMove()
{
    static float dist_x = 0, dist_y = 0;//ufoc出生时，记录与飞机的横竖距离
    static float tmp_x = 0, tmp_y = 0; //储存x,y坐标的临时变量，浮点型方便计算
    static float vx = 0, vy = 0;
    float step = 1000 / ufoc.speed;       //调整UFO速度
    if (1 == ufoc.new_born_flg)
    {
        ufoc.new_born_flg = 0;
        tmp_x = rand() % (WIDTH - ufoc.width);
        tmp_y = -ufoc.height;
        dist_x = plane.x - tmp_x;
        dist_y = plane.y - tmp_y;
        vx = dist_x / step;
        vy = dist_y / step;
    }
    tmp_x += vx;
    tmp_y += vy;
    ufoc.x = (int)(tmp_x + 0.5);
    ufoc.y = (int)(tmp_y + 0.5);
    //边界判断，可以超出画面，但不超出太多
    if (ufoc.x < -ufoc.width)
        ufoc.new_born_flg = 1;
    else if (ufoc.x > WIDTH+ufoc.width)
        ufoc.new_born_flg = 1;
    //超出下边界，重生，y坐标重置，x坐标随机
    if (ufoc.y >= HEIGHT)
        ufoc.new_born_flg = 1;
}


//在某链表尾部插入一个数据
void listPushBack(list** pplist, list* newNode)
{
    if (*pplist == NULL)//如果链表为空，那么新增的节点就是第一个
    {
        *pplist = newNode;
```

```
        return;
    }
    list* cur = *pplist;
    while (cur->pnext != NULL)//找到最后一个节点
    {
        cur = cur->pnext;
    }
    cur->pnext = newNode;//插入新的节点
}

//飞机发射的子弹，增加一个节点
list* creatPlaneBullet(float vx, float vy)
{
    list* p = (list*)malloc(sizeof(list));
    p->x = plane.x + plane.width / 2+10;//飞机头部的位置
    p->y = plane.y;
    p->vx = vx;
    p->vy = vy;//速度
    p->damage = 50;
    p->isExist = 1;
    p->pnext = NULL;
    return p;
}

//ufoa发射的子弹，增加一个节点
list* creatUFOA_Bullet(float vx, float vy)
{
    list* p = (list*)malloc(sizeof(list));
    p->x = ufoa.x + ufoa.width / 2;//中间
    p->y = ufoa.y + ufoa.height;   //下方
    p->vx = vx;
    p->vy = vy;//速度
    p->damage = 10;
    p->isExist = 1;
    p->pnext = NULL;
    return p;
}

//ufoB发射的子弹，增加一个节点
list* creatUFOB_Bullet(float vx, float vy)
{
    list* p = (list*)malloc(sizeof(list));
    p->x = ufob.x + ufob.width / 2;//中间
    p->y = ufob.y + ufob.height;   //下方
    p->vx = vx;
    p->vy = vy;//速度
    p->damage = 20;
    p->isExist = 1;
    p->pnext = NULL;
    return p;
}
//修改某链中所有节点的坐标。
void listChangeXY(list** pplist)
{
    if (*pplist == NULL)//如果链表为空，那么新增的节点就是第一个
        return;
    list* cur = *pplist;//curret指向第一个节点
    while (cur != NULL)//遍历链表
```

```
    {
        cur->x += cur->vx;
        cur->y += cur->vy;
        //判断子弹是否离开视野
        if ((cur->y < -20)||(cur->y > HEIGHT)||(cur->x > WIDTH)||(cur->y < -20))
            cur->isExist = 0;
        cur = cur->pnext;//指向下一个节点
    }
}
//删除链表中isExist为0的节点
void listRemoveNode(list** pplist)
{
    if (*pplist == NULL)//如果链表为空，就没有可删除的节点了
        return;
    list* cur = *pplist;//curret先指向第一个节点
    list* prev = NULL;  //previous指向上一个节点的指针
    while (cur != NULL)//遍历链表
    {
        if (cur->isExist == 0)//判断节点是否需要删除
        {
            if (*pplist == cur)//如果删除的是第一个节点
            {
                *pplist = cur->pnext;  //更改链表的地址，让下一个节点作为头结点 ，如果没
有节点，则链表为空
                free(cur);             //释放当前节点（第一个节点的）空间
                cur = *pplist;         //让cur指向下一个节点
            }
            else
            {
                prev->pnext = cur->pnext;  //记录下一个节点的地址
                free(cur);                 //释放当前节点空间
                cur = prev;                //当前节点变成前一个节点
            }
        }
        else //如果不需要删除节点，储存当前节点为前一个节点，然后指向下一个节点
        {
            prev = cur;
            cur = cur->pnext;
        }
    }
}

//绘制发出来的子弹
void showBullet()
{
        //判断飞机子弹是否击中，更改敌机生命和子弹状态
    bulletHitUFO(&ufoa);
    bulletHitUFO(&ufob);
    bulletHitUFO(&ufoc);

    //判断敌机子弹是否击中，更改飞机生命和子弹状态
    bulletHitPlane(ufoa_bullet_list);
    bulletHitPlane(ufob_bullet_list);

    //飞机发射的子弹
    listChangeXY(&plane_bullet_list);//计算子弹新的位置
    listRemoveNode(&plane_bullet_list);//超出视野或者击中飞行器的子弹删除掉
    for (list* cur = plane_bullet_list; cur != NULL; cur = cur->pnext)
```

```
    {
        drawAlpha(&temp_img,cur->x, cur->y, &img_plane_bullet);
    }

    //ufoa发出的子弹
    listChangeXY(&ufoa_bullet_list);//计算子弹新的位置
    listRemoveNode(&ufoa_bullet_list);//超出视野或者击中飞行器的子弹删除掉
    for (list* cur = ufoa_bullet_list; cur != NULL; cur = cur->pnext)
    {
        drawAlpha(&temp_img,cur->x-10, cur->y-10, &img_ufoa_bullet);
    }

    //ufob发出的子弹
    listChangeXY(&ufob_bullet_list);//计算子弹新的位置
    listRemoveNode(&ufob_bullet_list);//超出视野或者击中飞行器的子弹删除掉
    for (list* cur = ufob_bullet_list; cur != NULL; cur = cur->pnext)
    {
        drawAlpha(&temp_img,cur->x-15, cur->y-30, &img_ufob_bullet);
    }

}

//判断飞机的子弹是否击中UFO，执行相应的加分与减命的操作
void bulletHitUFO(aircraft* tmp)
{
    for (list* cur = plane_bullet_list; cur != NULL; cur = cur->pnext)
    {
        //子弹在UFO的矩形图片内时，认为击中
        if ((cur->x > tmp->x) && (cur->x < tmp->x + tmp->width))
        {
            if ((cur->y > tmp->y) && (cur->y < tmp->y + tmp->height))
            {
                tmp->life -= cur->damage;//飞行器生命值 - 子弹的伤害
                cur->isExist = 0;  //清除子弹的存在标记
            }
        }
    }
}

//飞行器重生  参数为飞行器的地址，以及重生的血量
void aircraftReborn(aircraft* tmp, int life)
{
    tmp->new_born_flg = 1;
    tmp->life = life;
}
//飞行器生存判断
void aircraftLifeJudge()
{
    if (ufoa.life <= 0)
    {
        //aircraftReborn(&ufoa, ufoa.life);//这是错误写法
        aircraftReborn(&ufoa, UFOA_LIFE);
    }
    if (ufob.life <= 0)
    {
        aircraftReborn(&ufob, UFOB_LIFE);
    }
    if (ufoc.life <= 0)
```

```
    {
        aircraftReborn(&ufoc, UFOC_LIFE);
    }
    if (plane.life <= 0)
    {
       // game_over = 1;
    }
}
//UFO是否与飞机碰撞  判断UFO下方中间的点,中间偏左与中间偏右的点，是否处于飞行器的范围内
void ufoCrash(aircraft* tmp)
{
    triangle tri = getPlaneTriangle();//获取飞机的三角形参数
    if (isPointInTriangle(tri, tmp->x + tmp->width / 2, tmp->y + tmp->height)
        || isPointInTriangle(tri, tmp->x + tmp->width / 4, tmp->y + tmp->height
/ 2)
        || isPointInTriangle(tri, tmp->x + tmp->width - tmp->width / 4, tmp->y +
tmp->height / 2))
    {
        plane.life -= PLANE_LIFE / 2;//飞船掉一半的最大生命值
        tmp->life = 0;//产生撞击的UFO死掉
    }
}
//飞机的撞击判断
void ufosCrashJudge()
{
    ufoCrash(&ufoa);
    ufoCrash(&ufob);
    ufoCrash(&ufoc);
}
//2个向量的叉乘，结果仍然是向量，正负可以表示方向
float crossProduct(vector a, vector b)
{
    float tmp = a.x * b.y - a.y * b.x;
    return tmp;
}
//判断点(x,y)是否在三角形内
int isPointInTriangle(triangle tri, float x, float y)
{
    vector pa = getVector(tri.ax, tri.ay, x, y);//向量pa，是a-p
    vector pb = getVector(tri.bx, tri.by, x, y);//向量pb，是b-p
    vector pc = getVector(tri.cx, tri.cy, x, y);//向量pc，是c-p
    float t1 = crossProduct(pa, pb);
    float t2 = crossProduct(pb, pc);
    float t3 = crossProduct(pc, pa);
    return t1 * t2 >= 0 && t1 * t3 >= 0 && t2 * t3 >= 0;
}
//传入2组坐标，生成向量
vector getVector(float x1, float y1, float x2, float y2)
{
    vector tmp;
    tmp.x = x2 - x1;
    tmp.y = y2 - y1;
    return tmp;
}
//根据飞机图片的x 与 y坐标 来构建一个三角形，用于碰撞判断
triangle getPlaneTriangle()
{
    triangle tmp;
```

```
    //a 是最上边的点，b 是右下， c是左下。
    tmp.ax = plane.x + plane.width / 2;
    tmp.ay = plane.y;
    tmp.bx = plane.x + plane.width;
    tmp.by = plane.y + plane.height;
    tmp.cx = plane.x;
    tmp.cy = plane.y + plane.height;
    return tmp;
}


//判断UFO的子弹是否击中飞机，执行相应的加分与减命的操作,参数是UFO的子弹链表
void bulletHitPlane(list* bullet_list)
{
    for (list* cur = bullet_list; cur != NULL; cur = cur->pnext)
    {
        //子弹在飞机的矩形图片内时，再判断是否在飞机的三角形内，减少计算量
        if ((cur->x > plane.x) && (cur->x < plane.x + plane.width))
        {
            if ((cur->y > plane.y) && (cur->y < plane.y + plane.height))
            {
                triangle tri = getPlaneTriangle();//获取飞机的三角形参数
                if (isPointInTriangle(tri, cur->x, cur->y))//子弹与飞机相撞
                {
                    plane.life -= cur->damage;
                    cur->isExist = 0;
                }
            }
        }
    }
}
```

飞机被撞击

```
//UFO是否与飞机碰撞  判断UFO下方中间的点,中间偏左与中间偏右的点，是否处于飞行器的范围内
void ufoCrash(aircraft* tmp)
{
    triangle tri = getPlaneTriangle();//获取飞机的三角形参数
    if (isPointInTriangle(tri, tmp->x + tmp->width / 2, tmp->y + tmp->height)
        || isPointInTriangle(tri, tmp->x + tmp->width / 4, tmp->y + tmp->height
/ 2)
        || isPointInTriangle(tri, tmp->x + tmp->width - tmp->width / 4, tmp->y +
tmp->height / 2))
    {
        plane.life -= PLANE_LIFE / 2;//飞船掉一半的最大生命值
        tmp->life = 0;//产生撞击的UFO死掉
    }
}
//飞机的撞击判断
void ufosCrashJudge()
{
    ufoCrash(&ufoa);
    ufoCrash(&ufob);
    ufoCrash(&ufoc);
}
```

主函数增加撞击判断

目前飞机撞谁谁死

```
#include "stdafx.h"

#define HEIGHT 720  // 游戏画面尺寸
#define WIDTH 1280
#define PI 3.1415926

//定义飞机的结构体
struct aircraft
{
    int x ;
    int y;
    int width;
    int height;
    int speed;
    int life;
    int new_born_flg;
};
aircraft plane, ufoa, ufob, ufoc;

//向量的结构体
struct vector
{
    float x, y;
};
//三角形的结构体
struct triangle
{
    float ax, ay, bx, by, cx, cy;
};

//定义子弹的结构体, 组成链表
typedef struct bullet
{
    float x, y;
    float vx, vy;
    int damage;//伤害
    int isExist;//判断子弹是否需要删除
    struct bullet* pnext;//指向下一个子弹节点的指针
}list;//给结构体起一个别名list,头结点可以表示列表的信息

list* plane_bullet_list = NULL; // 飞机子弹列表的头节点
list* ufob_bullet_list = NULL;  //UFOB的子弹列表的开头
list* ufoa_bullet_list = NULL;  //UFOA的子弹列表的开头

IMAGE img_bk, img_plane,temp_img,
    img_ufoa,img_ufob,img_ufoc,
    img_plane_bullet,img_ufoa_bullet,img_ufob_bullet;

int PLANE_LIFE = 100;
int UFOA_LIFE = 1500;
int UFOB_LIFE = 150;
int UFOC_LIFE = 100;
```

```
//按键输入的枚举列表
enum GAMEINPUT
{
    NOINPUT = 0X0,
    UPINPUT = 0X1,
    DOWNINPUT = 0X2,
    LEFTINPUT = 0X4,
    RIGHTINPUT = 0X8,
    FIREINPUT = 0X10
};
int input = NOINPUT;//判断输入变量
int speed = 10;

void dataInit();
void loadRes();
void drawAlpha(IMAGE* dstimg, int x, int y, IMAGE* srcimg);
void showAircraft();
void getInput();
void dealInput();
void ctrlFps(int start_time);
void ufoaMove();
void ufobMove();
void ufocMove();
void showBullet();
void listPushBack(list** pplist, list* newNode);
list* creatPlaneBullet(float vx, float vy);
list* creatUFOA_Bullet(float vx, float vy);
list* creatUFOB_Bullet(float vx, float vy);
void listChangeXY(list** pplist);
void listRemoveNode(list** pplist);

void aircraftReborn(aircraft* tmp, int life);
void aircraftLifeJudge();

//处理三角形相关函数
vector getVector(float x1, float y1, float x2, float y2);//生成向量
float crossProduct(vector a, vector b);//2个向量的叉乘
int isPointInTriangle(triangle tri, float x, float y);//判断点(x,y)是否在三角形内
triangle getPlaneTriangle();//构建一个三角形，用于碰撞判断

//碰撞相关函数
void bulletHitPlane(list* bullet_list);//判断UFO的子弹是否击中飞机
void bulletHitUFO(aircraft* tmp);//判断飞机的子弹是否击中UFO
void ufoCrash(aircraft* tmp);//UFO是否与飞机碰撞
void ufosCrashJudge();//整个UFO是否与飞机碰撞

int _tmain(int argc, _TCHAR* argv[])
{
    dataInit();//初始化所有飞机，设置飞机坐标
    initgraph(WIDTH, HEIGHT);// 创建绘图窗口
    loadRes();
    clock_t start_time;
    BeginBatchDraw();
    srand(time(NULL));
    while(1)
    {
```

```
        ufoaMove();
        ufobMove();
        ufocMove();
        start_time = clock();
        getInput(); //获取输入
        dealInput();//调整飞机位置
        showAircraft();//显示飞机
        showBullet();
        putimage(0, 0, &temp_img);
        aircraftLifeJudge();//飞机生存判断
        ufosCrashJudge();//飞机敌机碰撞判断
        ctrlFps(start_time);//控制Fps在60左右
        FlushBatchDraw();
    }
    EndBatchDraw();
    _getch();
    closegraph();

    return 0;
}

//飞机数据初始化
void dataInit()
{
    plane = { 150,150,80,80,10,100,1 };
    ufoa = { 0,0 ,300,150,2,1000,1 };
    ufob = { 350,0,150,50,4,150,1 };
    ufoc = { 450,200 ,100,60,10,100,1 };
}

//以相对路径载入所有素材
void loadRes()
{
    loadimage(&img_bk, _T("res\\background.png"));
    loadimage(&temp_img, _T("res\\background.png"));
    loadimage(&img_plane, _T("res\\plane.png"));
    loadimage(&img_ufoa, _T("res\\ufoa.png"));
    loadimage(&img_ufob, _T("res\\ufob.png"));
    loadimage(&img_ufoc, _T("res\\ufoc.png"));
    loadimage(&img_plane_bullet, _T("res\\plane_bullet.png"));
    loadimage(&img_ufoa_bullet, _T("res\\ufoa_bullet.png"));
    loadimage(&img_ufob_bullet, _T("res\\ufob_bullet.png"));
}

// 根据透明度绘图
void drawAlpha(IMAGE *dstimg, int x, int y, IMAGE *srcimg)
{
    // 变量初始化
    DWORD *dst = GetImageBuffer(dstimg);
    DWORD *src = GetImageBuffer(srcimg);
    int src_width = srcimg->getwidth();
    int src_height = srcimg->getheight();
    int dst_width = (dstimg == NULL ? getwidth() : dstimg->getwidth());
    int dst_height = (dstimg == NULL ? getheight() : dstimg->getheight());

    // 计算贴图的实际长宽
    int iwidth = (x + src_width > dst_width) ? dst_width - x : src_width;
// 处理超出右边界
```

```
    int iheight = (y + src_height > dst_height) ? dst_height - y : src_height;
// 处理超出下边界
    if (x < 0) { src += -x;             iwidth -= -x;   x = 0; }
// 处理超出左边界
    if (y < 0) { src += src_width * -y; iheight -= -y;  y = 0; }
// 处理超出上边界

    // 修正贴图起始位置
    dst += dst_width * y + x;
    // 实现透明贴图
    for (int iy = 0; iy < iheight; ++iy)
    {
        for (int i = 0; i < iwidth; ++i)
        {
            int sa = ((src[i] & 0xff000000) >> 24);//获取阿尔法值
            if (sa != 0)//假如是完全透明就不处理
            if (sa == 255)//假如完全不透明则直接拷贝
            dst[i] = src[i];
            else//真正需要阿尔法混合计算的图像边界才进行混合
            dst[i] = ((((src[i] & 0xff0000) >> 16) + ((dst[i] & 0xff0000) >> 16)
* (255 - sa) / 255) << 16) |((((src[i] & 0xff00) >> 8) + ((dst[i] & 0xff00) >>
8) * (255 - sa) / 255) << 8) | ((src[i] & 0xff) + (dst[i] & 0xff) * (255 - sa) /
255);
        }
        dst += dst_width;
        src += src_width;
    }
}
//绘制所有的飞机
void showAircraft()
{
    drawAlpha(&temp_img,0, 0, &img_bk);
    drawAlpha(&temp_img,plane.x, plane.y, &img_plane);
    drawAlpha(&temp_img,ufoa.x, ufoa.y, &img_ufoa);
    drawAlpha(&temp_img,ufob.x, ufob.y, &img_ufob);
    drawAlpha(&temp_img,ufoc.x, ufoc.y, &img_ufoc);
}

//同时获取多个输入，操作飞机
void getInput()
{
    int reload_time = 100;//飞机发子弹的间隔时间，单位ms
    static int fire_start = 0;//静态变量，储存开火的时间
    int tmp = clock();
    if (GetAsyncKeyState('W') & 0x8000)
    {
        input |= UPINPUT;
    }
    if (GetAsyncKeyState('S') & 0x8000)
    {
        input |= DOWNINPUT;
    }
    if (GetAsyncKeyState('A') & 0x8000)
    {
        input |= LEFTINPUT;
    }
    if (GetAsyncKeyState('D') & 0x8000)
    {
```

```
        input |= RIGHTINPUT;
    }
    if (GetAsyncKeyState('K') & 0x8000)
    {
        if (tmp - fire_start >= reload_time)
        {
            input |= FIREINPUT;
            fire_start = tmp;
        }
    }
}

//同时处理多个输入，调整飞机的位置
void dealInput()
{
    if ((input & UPINPUT) && (plane.y >= 0))
    {
        plane.y -= speed;
    }
    if ((input & DOWNINPUT) && (plane.y <= HEIGHT - 120))
    {
        plane.y += speed;
    }
    if ((input & LEFTINPUT) && (plane.x >= 0))
    {
        plane.x -= speed;
    }
    if ((input & RIGHTINPUT) && (plane.x <= WIDTH - 120))
    {
        plane.x += speed;
    }
    if (input & FIREINPUT)
    {
        listPushBack(&plane_bullet_list, creatPlaneBullet(0, -20));//水平方向无速
度，垂直向上速度20
        listPushBack(&plane_bullet_list, creatPlaneBullet(-10, -17.32));//30°散开
20*cos30约等于17.32
        listPushBack(&plane_bullet_list, creatPlaneBullet(10, -17.32));
    }

    input = NOINPUT;
}

//根据从开始到现在的时间，设置休眠的时间
void ctrlFps(int start_time)
{
    clock_t running_time = clock() - start_time;
    if((13 - running_time) >= 0)//防止睡眠函数使用负数
      Sleep(13 - running_time);//动态睡眠
    TCHAR time_text[50];
    int FPS = 1000 / (clock() - start_time);
    _stprintf_s(time_text, _T("FPS:%d"), FPS);
    settextstyle(60, 0, _T("黑体")); //为了演示，显示fps字体大小不宜太大
    outtextxy(0, 0, time_text);
}

//UFOA缓慢向前移动，到达一定的位置回去,转圈发射子弹
void ufoaMove()
```

```
{
    static int cnt = 0;
    static int dir = 1;//前进
    int ufoa_reload_cnt = 80;//发射子弹的计数器，数值越小发射子弹越快
    int ufoa_fire_num = 20;  //UFOA共发射多少路子弹
    if (1 == ufoa.new_born_flg)//新出生的ufoa ， 重置x,y的坐标
    {
        ufoa.new_born_flg = 0;
        ufoa.x = rand() % (WIDTH - ufoa.width);
        ufoa.y = -50 ;
    }
    if(ufoa.y > 200)//纵坐标大于300，改为后退
    {
        dir = 0; //后退

    }
    else if(ufoa.y < -150)
    {
        dir = 1;
        ufoa.new_born_flg = 1;
    }

    if(dir == 1) //前进
        ufoa.y += ufoa.speed;
    else  //后退
        ufoa.y -= ufoa.speed;

    //cnt到达指定的间隔以后，转圈发射子弹
    if (++cnt % ufoa_reload_cnt == 0)
    {
        for (int i = 0; i < ufoa_fire_num; i++)
        {
            float angle = i * 2 * PI / ufoa_fire_num;
            float vx = 4 * sin(angle);
            float vy = 4 * cos(angle);
            listPushBack(&ufoa_bullet_list, creatUFOA_Bullet(vx, vy));
        }
    }
    if (cnt > 999999) cnt = 0;
}


//UFOB左右快速移动，慢速向下移动
void ufobMove()
{
    static int step = ufob.speed;//spep表示方向与速度
    static int cnt = 0;
    int ufob_reload_cnt = 60;//UFOb  发射子弹的计数器，数值越小发射子弹越快
    if (1 == ufob.new_born_flg)//新出生的ufob ， 重置x,y的坐标
    {
        ufob.new_born_flg = 0;
        ufob.x = rand() % (WIDTH - ufob.width);
        ufob.y = -ufob.height;
    }
    //水平撞墙返回
    if ((ufob.x <= 0) || (ufob.x + ufob.width >= WIDTH))
        step = -step;
    ufob.x += step;
```

```
    ufob.y++;
    //超出下边界，重生，y坐标重置，x坐标随机
    if (ufob.y >= HEIGHT)
        ufob.new_born_flg = 1;
    if (++cnt % ufob_reload_cnt == 0)
    {
        listPushBack(&ufob_bullet_list, creatUFOB_Bullet(0, 5));
    }
    if (cnt > 999999) cnt = 0;//当计数器超过999999 进行清零
}

//ufoc撞向飞机
void ufocMove()
{
    static float dist_x = 0, dist_y = 0;//ufoc出生时，记录与飞机的横竖距离
    static float tmp_x = 0, tmp_y = 0; //储存x,y坐标的临时变量，浮点型方便计算
    static float vx = 0, vy = 0;
    float step = 1000 / ufoc.speed;       //调整UFO速度
    if (1 == ufoc.new_born_flg)
    {
        ufoc.new_born_flg = 0;
        tmp_x = rand() % (WIDTH - ufoc.width);
        tmp_y = -ufoc.height;
        dist_x = plane.x - tmp_x;
        dist_y = plane.y - tmp_y;
        vx = dist_x / step;
        vy = dist_y / step;
    }
    tmp_x += vx;
    tmp_y += vy;
    ufoc.x = (int)(tmp_x + 0.5);
    ufoc.y = (int)(tmp_y + 0.5);
    //边界判断，可以超出画面，但不超出太多
    if (ufoc.x < -ufoc.width)
        ufoc.new_born_flg = 1;
    else if (ufoc.x > WIDTH+ufoc.width)
        ufoc.new_born_flg = 1;
    //超出下边界，重生，y坐标重置，x坐标随机
    if (ufoc.y >= HEIGHT)
        ufoc.new_born_flg = 1;
}


//在某链表尾部插入一个数据
void listPushBack(list** pplist, list* newNode)
{
    if (*pplist == NULL)//如果链表为空，那么新增的节点就是第一个
    {
        *pplist = newNode;
        return;
    }
    list* cur = *pplist;
    while (cur->pnext != NULL)//找到最后一个节点
    {
        cur = cur->pnext;
    }
    cur->pnext = newNode;//插入新的节点
}
```

```
//飞机发射的子弹，增加一个节点
list* creatPlaneBullet(float vx, float vy)
{
    list* p = (list*)malloc(sizeof(list));
    p->x = plane.x + plane.width / 2+10;//飞机头部的位置
    p->y = plane.y;
    p->vx = vx;
    p->vy = vy;//速度
    p->damage = 50;
    p->isExist = 1;
    p->pnext = NULL;
    return p;
}

//ufoa发射的子弹，增加一个节点
list* creatUFOA_Bullet(float vx, float vy)
{
    list* p = (list*)malloc(sizeof(list));
    p->x = ufoa.x + ufoa.width / 2;//中间
    p->y = ufoa.y + ufoa.height;   //下方
    p->vx = vx;
    p->vy = vy;//速度
    p->damage = 10;
    p->isExist = 1;
    p->pnext = NULL;
    return p;
}

//ufoB发射的子弹，增加一个节点
list* creatUFOB_Bullet(float vx, float vy)
{
    list* p = (list*)malloc(sizeof(list));
    p->x = ufob.x + ufob.width / 2;//中间
    p->y = ufob.y + ufob.height;   //下方
    p->vx = vx;
    p->vy = vy;//速度
    p->damage = 20;
    p->isExist = 1;
    p->pnext = NULL;
    return p;
}
//修改某链中所有节点的坐标。
void listChangeXY(list** pplist)
{
    if (*pplist == NULL)//如果链表为空，那么新增的节点就是第一个
        return;
    list* cur = *pplist;//curret指向第一个节点
    while (cur != NULL)//遍历链表
    {
        cur->x += cur->vx;
        cur->y += cur->vy;
        //判断子弹是否离开视野
        if ((cur->y < -20)||(cur->y > HEIGHT)||(cur->x > WIDTH)||(cur->y < -20))
            cur->isExist = 0;
        cur = cur->pnext;//指向下一个节点
    }
}
```

```
//删除链表中isExist为0的节点
void listRemoveNode(list** pplist)
{
    if (*pplist == NULL)//如果链表为空，就没有可删除的节点了
        return;
    list* cur = *pplist;//curret先指向第一个节点
    list* prev = NULL;  //previous指向上一个节点的指针
    while (cur != NULL)//遍历链表
    {
        if (cur->isExist == 0)//判断节点是否需要删除
        {
            if (*pplist == cur)//如果删除的是第一个节点
            {
                *pplist = cur->pnext;  //更改链表的地址，让下一个节点作为头结点 ，如果没有节点，则链表为空
                free(cur);             //释放当前节点（第一个节点的）空间
                cur = *pplist;         //让cur指向下一个节点
            }
            else
            {
                prev->pnext = cur->pnext;  //记录下一个节点的地址
                free(cur);                 //释放当前节点空间
                cur = prev;                //当前节点变成前一个节点
            }
        }
        else //如果不需要删除节点，储存当前节点为前一个节点，然后指向下一个节点
        {
            prev = cur;
            cur = cur->pnext;
        }
    }
}

//绘制发出来的子弹
void showBullet()
{
        //判断飞机子弹是否击中，更改敌机生命和子弹状态
    bulletHitUFO(&ufoa);
    bulletHitUFO(&ufob);
    bulletHitUFO(&ufoc);

    //判断敌机子弹是否击中，更改飞机生命和子弹状态
    bulletHitPlane(ufoa_bullet_list);
    bulletHitPlane(ufob_bullet_list);

    //飞机发射的子弹
    listChangeXY(&plane_bullet_list);//计算子弹新的位置
    listRemoveNode(&plane_bullet_list);//超出视野或者击中飞行器的子弹删除掉
    for (list* cur = plane_bullet_list; cur != NULL; cur = cur->pnext)
    {
        drawAlpha(&temp_img,cur->x, cur->y, &img_plane_bullet);
    }

    //ufoa发出的子弹
    listChangeXY(&ufoa_bullet_list);//计算子弹新的位置
    listRemoveNode(&ufoa_bullet_list);//超出视野或者击中飞行器的子弹删除掉
    for (list* cur = ufoa_bullet_list; cur != NULL; cur = cur->pnext)
    {
```

```
        drawAlpha(&temp_img,cur->x-10, cur->y-10, &img_ufoa_bullet);
    }

    //ufob发出的子弹
    listChangeXY(&ufob_bullet_list);//计算子弹新的位置
    listRemoveNode(&ufob_bullet_list);//超出视野或者击中飞行器的子弹删除掉
    for (list* cur = ufob_bullet_list; cur != NULL; cur = cur->pnext)
    {
        drawAlpha(&temp_img,cur->x-15, cur->y-30, &img_ufob_bullet);
    }

}

//判断飞机的子弹是否击中UFO，执行相应的加分与减命的操作
void bulletHitUFO(aircraft* tmp)
{
    for (list* cur = plane_bullet_list; cur != NULL; cur = cur->pnext)
    {
        //子弹在UFO的矩形图片内时，认为击中
        if ((cur->x > tmp->x) && (cur->x < tmp->x + tmp->width))
        {
            if ((cur->y > tmp->y) && (cur->y < tmp->y + tmp->height))
            {
                tmp->life -= cur->damage;//飞行器生命值 - 子弹的伤害
                cur->isExist = 0;  //清除子弹的存在标记
            }
        }
    }
}

//飞行器重生  参数为飞行器的地址，以及重生的血量
void aircraftReborn(aircraft* tmp, int life)
{
    tmp->new_born_flg = 1;
    tmp->life = life;
}
//飞行器生存判断
void aircraftLifeJudge()
{
    if (ufoa.life <= 0)
    {
        //aircraftReborn(&ufoa, ufoa.life);//这是错误写法
        aircraftReborn(&ufoa, UFOA_LIFE);
    }
    if (ufob.life <= 0)
    {
        aircraftReborn(&ufob, UFOB_LIFE);
    }
    if (ufoc.life <= 0)
    {
        aircraftReborn(&ufoc, UFOC_LIFE);
    }
    if (plane.life <= 0)
    {
       // game_over = 1;
    }
}
//UFO是否与飞机碰撞  判断UFO下方中间的点,中间偏左与中间偏右的点，是否处于飞行器的范围内
```

```
void ufoCrash(aircraft* tmp)
{
    triangle tri = getPlaneTriangle();//获取飞机的三角形参数
    if (isPointInTriangle(tri, tmp->x + tmp->width / 2, tmp->y + tmp->height)
        || isPointInTriangle(tri, tmp->x + tmp->width / 4, tmp->y + tmp->height
/ 2)
        || isPointInTriangle(tri, tmp->x + tmp->width - tmp->width / 4, tmp->y +
tmp->height / 2))
    {
        plane.life -= PLANE_LIFE / 2;//飞船掉一半的最大生命值
        tmp->life = 0;//产生撞击的UFO死掉
    }
}
//飞机的撞击判断
void ufosCrashJudge()
{
    ufoCrash(&ufoa);
    ufoCrash(&ufob);
    ufoCrash(&ufoc);
}
//2个向量的叉乘，结果仍然是向量，正负可以表示方向
float crossProduct(vector a, vector b)
{
    float tmp = a.x * b.y - a.y * b.x;
    return tmp;
}
//判断点(x,y)是否在三角形内
int isPointInTriangle(triangle tri, float x, float y)
{
    vector pa = getVector(tri.ax, tri.ay, x, y);//向量pa，是a-p
    vector pb = getVector(tri.bx, tri.by, x, y);//向量pb，是b-p
    vector pc = getVector(tri.cx, tri.cy, x, y);//向量pc，是c-p
    float t1 = crossProduct(pa, pb);
    float t2 = crossProduct(pb, pc);
    float t3 = crossProduct(pc, pa);
    return t1 * t2 >= 0 && t1 * t3 >= 0 && t2 * t3 >= 0;
}
//传入2组坐标，生成向量
vector getVector(float x1, float y1, float x2, float y2)
{
    vector tmp;
    tmp.x = x2 - x1;
    tmp.y = y2 - y1;
    return tmp;
}
//根据飞机图片的x 与 y坐标 来构建一个三角形，用于碰撞判断
triangle getPlaneTriangle()
{
    triangle tmp;
    //a 是最上边的点，b 是右下， c是左下。
    tmp.ax = plane.x + plane.width / 2;
    tmp.ay = plane.y;
    tmp.bx = plane.x + plane.width;
    tmp.by = plane.y + plane.height;
    tmp.cx = plane.x;
    tmp.cy = plane.y + plane.height;
    return tmp;
}
```

```
//判断UFO的子弹是否击中飞机，执行相应的加分与减命的操作,参数是UFO的子弹链表
void bulletHitPlane(list* bullet_list)
{
    for (list* cur = bullet_list; cur != NULL; cur = cur->pnext)
    {
        //子弹在飞机的矩形图片内时，再判断是否在飞机的三角形内，减少计算量
        if ((cur->x > plane.x) && (cur->x < plane.x + plane.width))
        {
            if ((cur->y > plane.y) && (cur->y < plane.y + plane.height))
            {
                triangle tri = getPlaneTriangle();//获取飞机的三角形参数
                if (isPointInTriangle(tri, cur->x, cur->y))//子弹与飞机相撞
                {
                    plane.life -= cur->damage;
                    cur->isExist = 0;
                }
            }
        }
    }
}
```