GetAsyncKeyState函数

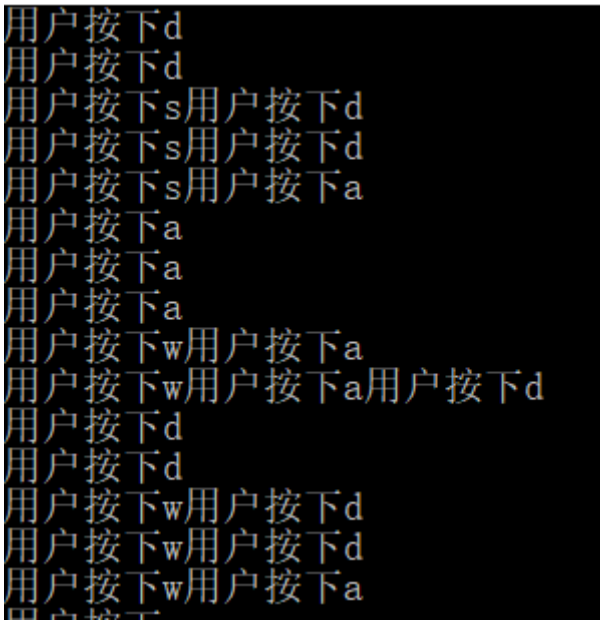
GetAsyncKeyState是一个用来判断函数调用时指定虚拟键的状态，确定用户当前是否按下了键盘上一个键的函数。如果按下，则返回值最高位为1。头文件是“#include<windows.h>”。

GetAsyncKeyState函数判断按键按下，则返回值16位数据，最高位为1，可进行& 0x8000判断是否有按键按下。

该函数可以判断出多个按键同时按下。除了常用按键外，其它按键的VK值：

VK值	对应按键	VK值	对应按键
VK_SHIFT	Shift键	VK_CONTROL	Ctrl键
VK_MENU	Alt键	VK_LBUTTON	鼠标左键
VK_RBUTTON	鼠标右键	VK_ESCAPE	ESC键
VK_RETURN	回车键	VK_SPACE	空格键
VK_TAB	TAB键	VK_DOWN	↓键
VK_UP	↑键	VK_LEFT	←键
VK_RIGHT	→键		

一小段测试程序：编写程序判断用户输入“w”“a”“s”“d”四个按键，用户长按“w”则打窗口打印 “用户输入w”提示。




```c
#include <stdio.h>
#include <windows.h>
int main(void)
{
    while (1)
    {
        int n = 0;  //每一次循环n的初始值为0
        if ((n = GetAsyncKeyState('w')) & 0x8000)
```

```
        {
            printf("用户按下w");
        }
        if ((n = GetAsyncKeyState('S')) & 0x8000)
        {
            printf("用户按下s");
        }
        if ((n = GetAsyncKeyState('A')) & 0x8000)
        {
            printf("用户按下a");
        }
        if ((n = GetAsyncKeyState('D')) & 0x8000)
        {
            printf("用户按下d");
        }
         printf("\n");
        Sleep(100); //等待1s查看结果
    }
    return 0;
}
```

获取按键输入

定义输入数据的枚举类型，方便代码阅读

```
//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
```

获取按键输入的函数

```
//同时获取多个输入，操作飞机
void getInput()
{
    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)
    {
        input |= FIREINPUT;
    }
}
```

处理按键输入

定义一个速度变量，代表每一帧移动多少个像素。

然后根据飞机当前的坐标与输入的数据，计算飞机的下一个位置。

但是飞机不能超出屏幕，所以要做一些判断

```
int speed = 10;
//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;
    }
    input = NOINPUT;
}
```

主函数建立死循环，获取按键输入，处理按键输入，然后显示飞机。

```
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();
    while(1)
    {
        getInput(); //获取输入
        dealInput();//调整飞机位置
        showAircraft();//显示飞机
```

```
        putimage(0, 0, &temp_img);
        Sleep(10);
    }

    _getch();
    closegraph();

    return 0;
}
```

演示现象，发现飞机可以移动，但是感觉很卡，有残影。

使用Sleep需要头文件“#include<windows.h>”。

完整代码：

```
#include "stdafx.h"

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

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

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


//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();

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();
    while(1)
    {
        getInput(); //获取输入
        dealInput();//调整飞机位置
        showAircraft();//显示飞机
        putimage(0, 0, &temp_img);
        Sleep(10);
    }
    _getch();
    closegraph();

    return 0;
}

//飞机数据初始化
void dataInit()
{
    plane = { 150,150 };
    ufoa  = { 0,0 };
    ufob  = { 350,0};
    ufoc  = { 450,200};
}

//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)
{
 if (dstimg == NULL)
 {
  return;
 }
 // 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->getwidth();
 int dst_height = dstimg->getheight();

 // 实现透明贴图  可优化
 for (int iy = 0; iy < src_height; iy++)
 {
  for (int ix = 0; ix < src_width; ix++)
  {
```

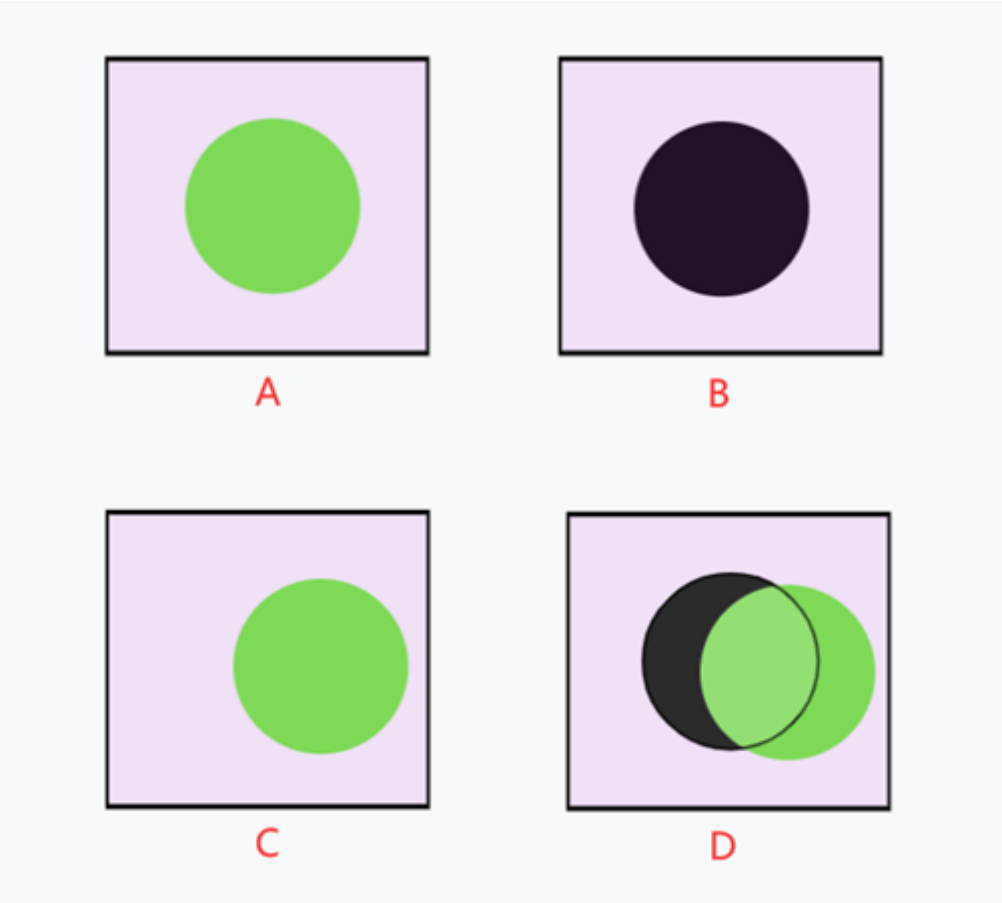
```
    int srcX = ix + iy * src_width;
    int sa = ((src[srcX] & 0xff000000) >> 24);
    int sr = ((src[srcX] & 0xff0000) >> 16);
    int sg = ((src[srcX] & 0xff00) >> 8);
    int sb = src[srcX] & 0xff;
    if (x + ix >= 0 && x + ix < dst_width
     && y + iy >= 0 && y + iy < dst_height)
    {
     int dstX = (x + ix) + (y + iy) * dst_width;
     int dr = ((dst[dstX] & 0xff0000) >> 16);
     int dg = ((dst[dstX] & 0xff00) >> 8);
     int db = dst[dstX] & 0xff;
     dst[dstX] = ((sr * sa / 255 + dr * (255 - sa) / 255) << 16)
      | ((sg * sa / 255 + dg * (255 - sa) / 255) << 8)
      | (sb * sa / 255 + db * (255 - sa) / 255);
    }
   }
  }
 }
}

//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);
    //暂时也绘制子弹
    drawAlpha(&temp_img,0, 300, &img_plane_bullet);
    drawAlpha(&temp_img,50, 300, &img_ufoa_bullet);
    drawAlpha(&temp_img,100, 300, &img_ufob_bullet);
}

//同时获取多个输入，操作飞机
void getInput()
{
    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)
    {
        input |= FIREINPUT;
    }
}
```

```
//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;
    }
    input = NOINPUT;
}
```

解决画面闪烁的问题

以绿色的小球在黑色的背景移动为例，这是因为每次画一个小球，不论是绘制新的小球，还是绘制背景色的小球，对于小球当前位置和下个位置重叠的部分，会被重复操作，看上去是绿色→黑色→绿色。如下图所示。先是绘制了绿球（A区域），延时后绘制黑球遮挡住刚才绘制的绿球（B区域），改变小球的坐标后，在更新的坐标在绘制绿球（C区域）。

重叠的部分绿色和黑色交替出现（D区域），便会有闪烁的现象。



EasyX提供了BeginBatchDraw()，FlushBatchDraw()，EndBatchDraw()批量绘图函数处理画面闪烁的问题。

BeginBatchDraw()函数用于开始批量绘图，执行后任何绘图操作都将暂时不输出到屏幕上，直到执行FlushbatchDraw()函数或者EndBatchDraw()函数才会将之前的绘图输出；FlushbatchDraw()函数用于执行未完成的绘制任务，执行批量绘制；EndBatchDraw()函数用于结束批量绘制并执行未完成的绘制任务。三个函数均没有参数。

因为使用批量绘图函数后，直到执行FlushbatchDraw()函数或者EndBatchDraw()函数才会将之前的绘图输出，所以绘制了遮挡上一秒的黑球后，没有直接显示在窗口中，再绘制下一秒显示的绿球后，再将这些内容同时显示出来，避免了绿球和黑球交替出现导致的闪烁问题。

使用BeginBatchDraw()，FlushBatchDraw()，EndBatchDraw()批量绘图函数对代码进行优化。

修改代码：

```
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();
    BeginBatchDraw();
    while(1)
    {
        getInput(); //获取输入
        dealInput();//调整飞机位置
        showAircraft();//显示飞机
        putimage(0, 0, &temp_img);
        Sleep(10);
        FlushBatchDraw();
    }
    EndBatchDraw();
```

```
    _getch();
    closegraph();

    return 0;
}
```

显示FPS

FPS就是刷新率（单位为Hz），一般液晶显示器为60Hz刷新率，现在流行144HZ乃至更高的刷新率。例如：60Hz的刷新率刷也就是指屏幕一秒内只扫描60次，即60帧/秒。而当刷新率太低时，肉眼就能感觉到屏幕的闪烁，不连贯。而刷新率过高也会造成画面撕裂，并且对计算机性能的要求也会变高。

本项目中，将FPS控制在60左右，并将FPS数值显示到程序中。计算FPS需要获取时间，使用clock函数可以获取时间。编写函数ctrlFps，获取时间并显示FPS，首先显示FPS。

```
void ctrlFps(int start_time)
{
    clock_t running_time = clock() - start_time;
//    if((5 - running_time) >= 0)//防止睡眠函数使用负数
//      Sleep(5 - running_time);//动态睡眠13ms
    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);
}
```

使用clock函数需要头文件#include <time.h>

修改主函数

```
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();
    while(1)
    {
        start_time = clock();
        getInput(); //获取输入
        dealInput();//调整飞机位置
        showAircraft();//显示飞机
        putimage(0, 0, &temp_img);
        //Sleep(10);
        ctrlFps(start_time);//控制Fps在60左右
        FlushBatchDraw();
    }
    EndBatchDraw();
    _getch();
    closegraph();

    return 0;
}
```

然后发现FPS只有45左右。

原因是贴图函数太浪费时间，寻求代码优化。

优化贴图并控制FPS

搜集资料：

https://codebus.cn/yangw/knowledge-of-picture-material

https://codebus.cn/yangw/transparent-putimage

https://codebus.cn/lingyin/0

修改贴图函数：

```
// 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;
    }
}
```

FPS能够达到500，速度飞快。

控制FPS，动态休眠，在每次贴图数量不同的情况下，休眠时间也能大致相等。

```
//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);
}
```

完整代码

```
#include "stdafx.h"

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

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

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


//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);

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();
    while(1)
    {
        start_time = clock();
        getInput(); //获取输入
        dealInput();//调整飞机位置
        showAircraft();//显示飞机
        putimage(0, 0, &temp_img);
        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 };
    ufoa  = { 0,0 };
    ufob  = { 350,0};
    ufoc  = { 450,200};
}

//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);
    //暂时也绘制子弹
    drawAlpha(&temp_img,0, 300, &img_plane_bullet);
    drawAlpha(&temp_img,50, 300, &img_ufoa_bullet);
    drawAlpha(&temp_img,100, 300, &img_ufob_bullet);
}

//同时获取多个输入，操作飞机
void getInput()
{
    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)
    {
        input |= FIREINPUT;
    }
}

//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;
    }
    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);
}
```