1.3.1 题目 1**：欢乐球吃球**

**【问题描述】**

用 C 语言编写一个简单的“欢乐球吃球”游戏程序。欢乐球吃球是一款休闲与挑战并

存的游戏。在这个球球的世界里，每个人都成为一颗独特的球球，发展与生存是所有人的目

标，刺激战场极限逃生。程序需实现所要求的基本功能；在完成基本功能后，可以选择实现

数项扩展功能。

**【基本功能要求】**

1、 游戏初始界面如图 1 所示（示例，可自己设计）。

2、 游戏开始 例如：按下空格键后，“空格开始”消失游戏开始。

图 1 欢乐球吃球游戏开始界面示例

3、 游戏开始后：

1） 载入图片，有一个大球和若干随机运动的小球，例如图 2；

2） 玩家球可以在一定范围左右或上下移动，玩家用键盘（例如： WASD

键）和控制上、下、左、右运行。玩家碰到墙就停下来，需要调转方

向才能继续前进。玩家可吃球。玩家每吃一球加 1 分。；

3）每隔几秒钟（例如 3 秒）就会产生一组球。

4） 游戏时间固定（例如 30 秒），游戏剩余时间在屏幕固定位置显示，

并适时更新游戏剩余时间；

5）游戏时间到（例如 30 秒），玩家游戏结束显示分数。- 3 -

6） 记录游戏得分，显示在游戏界面上。

7） 保存每个用户的最高分成绩，单击“保存成绩”按钮，将当前用户的

用户名和本次得分存入文件。（是否要求输入文件名自己决定。同一个用

户每次成绩都保存）

图 2 欢乐球吃球游戏界面示例

**1、优化界面**

界面分为 3 个区域：游戏区、控制区、信息显示区。增加游戏的控制和显示信息。

**2、显示排行榜**

单击“显示排行榜”按钮，在文件中找到当前成绩中的排行榜，显示在信息显示区中。

**3、将游戏的当前状态存盘和导入**

存盘：按指定文件名，将当前游戏的状态、用户名、得分等存入文件，游戏继续。

导入：删除当前游戏（删除前应提问是否存盘），将文件中存储的游戏状态恢复到屏幕

上，导入的游戏继续运行。导入文件中的游戏后，得分恢复到存盘时的得分。

**4、鼠标控制游戏**

**5、自己添加的其他功能**

自己添加的使游戏更加有趣的功能，增加画面内容，或使界面更加美观的功能（例如，

鼠标单击控制区中的按钮时，按钮有被按下去的效果，松开时，有弹回来的效果）。- 4 -

**2 课程设计要求与安排**

**2.1 完成课程设计任务的基本过程与时间安排**

程序设计应采用自顶向下的模块化程序设计方法。按照软件开发规范，本课程设计任务

的基本过程应该经过分析、设计、编码、调试、测试和整合优化程序等阶段。程序设计与开

发中应兼顾程序的可扩展性和健壮性。完成程序之后，最终还要提交课程设计报告。

（

**3.1C 语言显示方式概述**

C 语言有文本和图形两种显示方式，我们编写的控制台程序的默认显示方式为文本显示

方式。常用的 scanf、printf 等库函数均可在文本显示方式下直接使用，但图形显示方式下，

这些函数便无法正常使用，需要用图形方式下的函数。

EasyX 图形库提供了在 Windows 环境下 C 语言的图形显示方式的函数。在 VC 环境下

使用 EasyX 图形库，同学首先需要自行下载并安装。在 www.EasyX.cn 中有完整的 EasyX

的下载、安装以及使用的方法。本章主要包含完成课设的最基本的一些函数的介绍和使用范

例，内容也主要取自 www.EasyX.cn，仅供同学们参考。

首先给出两个简单例子说明文字显示和图形显示。

1. 文字显示

文本显示顾名思义，就是在屏幕上输出文字，无论是在控制台，还是在绘图窗口中。以

前作业是在控制台输出文字，而课设中更适合在绘图窗口中输出才能达到更好的游戏效果。

我们学习的第一个 C 程序“Hello world”就是在控制台以文本方式显示，想必我们已经很了

解这种情况了，所以，在这里仅列举一个在绘图窗口输出文字“Hello world”的例子。

例 1：

//#include "stdafx.h"

#include <graphics.h>

void main()

{

// 初始化绘图窗口

initgraph(50, 50);

//将” Hello world”在当前屏幕的当前位置输出

outtext(\_T("Hello world "));

//发出dos指令"pause"，其意为按任意键继续

system("pause");

//关闭绘图窗口，与initgraph对应

closegraph();

}

这段代码的作用就是在绘图窗口输出“Hello world”字符串，输出当前位置是光标位置，

虽然在绘图环境下看不到光标，但它是实际存在的，初始位置在绘图窗口的左上角，所以，

“Hello world”显示在了窗口左上角。

使用EasyX图形库的时候需要注意，“outtext(\_T("Hello world "))”中\_T为一个宏函数作

用是将ASCII字符串转化成Unicode字符串，Unicode字符串（Unicode和ASCII一样是一种- 11 -

编码标准）。所以该函数的参数只能是字符串常量，编写课设的时候会造成很多不必要的麻

烦。如果使用的是VS编译系统，可以在右侧解决方案资源管理处选择项目右键属性->配置

属性->常规->项目默认值->字符集改为“使用多字节字符集”便可直接使用ASCII字符串作

为参数。

2. 图形显示

一个图形显示的例子。

例 2：

#include "stdafx.h"

#include <graphics.h>

#include <conio.h> //引入line函数的库

#define HIGH 480

#define WIDE 640

int main()

{

//设置图形显示方式

initgraph(WIDE, HIGH,NULL);

//画线 四个参数分别为起点横纵坐标 终点横纵坐标

line(200, 240, 440, 240);

line(320, 120, 320, 360);

//按任意键继续（另一种方法）

getch();

//退出图形显示方式

closegraph();

return 0;

}

该段代码生成一个宽为 WIDE，高为 HIGH 的绘图窗口，然后画了两条线。最后用

closegraph 来关闭绘图窗口。编译这个代码并运行程序，会看到一个 640x480 的绘图窗口，

其中心画有十字。

**3.2 Visual C++ 的图形显示方式使用**

使用图形显示方式大致需要经历下列基本过程：

（1）设置图形显示方式。

（2）定义视口（不是必须的，缺省为屏幕窗口）。

（3）设置当前位置、前景色和背景色。

（4）绘制图形或文字。

EasyX 图形库提供了丰富的图形相关的函数，在例 1 和例 2 中我们已经开始使用了。下

面用到的函数、对象等都位于相应的头文件之中，安装 EasyX 以后便可以在源代码中包含该

头文件即可使用这些函数。- 12 -

**3.2.1 设置图形显示方式**

1. 进入图形显示方式

使用图形方式之前必须首先设置，进行必要的初始化，如例 1 和例 2 所示。initgraph 函

数完成图形方式的初始化。

HWND initgraph( int width, int height, int flag = NULL);

函数用于初始化绘图环境。

参数：width，绘图环境的宽度；Height，绘图环境的高度；flag，绘图环境的样式，默

认为 NULL。可为以下值：

**值**

**含义**

NOCLOSE

禁用绘图环境的关闭按钮。

NOMINIMIZE

禁用绘图环境的最小化按钮。

SHOWCONSOLE

保留原控制台窗口。

返回值：创建的绘图窗口的句柄。

2. 退出图形显方式

使用函数：closegraph();

应该和 initgraph 函数成对出现。

**3.2.2 定义视口**

文字和图形输出需要显示在窗口上，窗口在缺省的时候就是屏幕能够提供的整个矩形区

域。但是它是可以根据需要指定其大小的。屏幕输出区域我们称之为窗口（window），而它

在图形系统中的术语是视口（viewport），概念都是显示区域。图形函数是在视口上操作。

下面给出定义一个视口的方法：

#include<Windows.h>

int left=0,top=0,right=100,bottom=100;

HRGN Area=CreateRectRgn(int left, int top, int right, int bottom);

解析：

（1）其中，HRGN 是一个句柄，位于 Whindows.h 头文件。为了方便理解，可以把它当成

一个指针。（当然它和指针还是有区别的）

（2）CreateRectRgn 函数的原型为：

HRGN CreateRectRgn(int left, int top, int right, int bottom);

这个函数把一个以（left，top）为左上角，以（right，bottom）为右下角的矩形区域看做了

一个整体，并返回一个指向它的句柄。在上述代码里，这个句柄被赋给了 Area。之后，便

可以通过 Area 句柄来对这一块区域进行一些统一的操作。创建区域后不再使用时，执行

DeleteObject(HRGN hrgn)函数释放该区域对应的系统资源。- 13 -

一些相关的函数：

（1）设置当前绘图设备的裁剪区函数：void setcliprgn(HRGN hrgn);

将句柄 hrgn 指向的区域设置为裁剪区域，这样只有在该区域内进行的绘图才有效果。

（2）取消之前设置的裁剪区函数：setcliprgn(NULL);

（3）清空裁剪区的屏幕内容函数：void clearcliprgn();

（4）用于设置坐标原点函数：void setorigin(int x, int y);

其中，x，y 均是物理坐标。

例：#include <conio.h>

#include <stdio.h>

#include <graphics.h>

#include <Windows.h>

int main()

{

initgraph(640, 480);

HRGN Area,Area2;

Area = CreateRectRgn(0, 0, 100, 100);

Area2 = CreateRectRgn(100, 100, 300, 300);

//将(0,0),(100，100）和（100,100,300,300）

//所表示的矩形区域设置为一个视口

circle(50, 50, 20);

circle(200, 200, 20);

//画两个圆

getch();

//程序暂停，按任意键继续

setcliprgn(Area);

//将 Area 设置为一个裁剪区

clearcliprgn();

//清空这个区域

setcliprgn(NULL);

//取消之前设置的裁剪区

getch();

//程序暂停，按任意键继续

setcliprgn(Area2);

//将 Area2 设置为一个裁剪区

clearcliprgn();

//清空这个区域

system("pause");

}

运行程序后，会画出两个圆，而连续两次按键盘后，这两个圆会被依次清除。- 14 -

此外，windows 有很多创建区域的函数，如：使用 CreateEllipticRgn 创建椭圆形的

区域；使用 CreatePolygonRgn 创建多边形的区域等等。还可以使用 CombineRgn 创建组

合区域。用到时查手册或 MSDN。

**3.2.3 设置当前位置**

在图形显示模式下光标是不显示的，但仍然有一个光标位置，我们把这个位置称为当前

位置。有一些函数的输出是从当前位置开始的，所以需要在程序中控制当前位置。下表是几

个控制当前位置的函数。

函数原形

功能描述

**int getwidth(void);**

返回当前图形显示方式下的最大的 x 坐标值

**int getheight(void);**

返回当前图形显示方式下的最大的 y 坐标值

**int getx(void);**

返回当前位置的 x 坐标

**int gety(void);**

返回当前位置的 y 坐标

**void moveto(int x,int y);**

将当前位置设置到（x,y）位置

**void moverel(int dx,int dy)**

如果当前位置为（x,y），则该函数将当前位置移到

（x+dx,y+dy）

#include <conio.h>

#include <stdio.h>

#include <graphics.h>

int main()

{

initgraph(640, 480);

outtext("I'm here!");

//开始时光标在（0,0）处

getch();

moveto(50, 50);

//将光标移动至（50,50）

outtext("I'm here!");

getch();

moverel(10, -20);

//将光标移动到当前位置的横坐标加 10，纵坐标减去 20 的新位置

outtext("I'm here!");

getch();}

注意：在使用 outtext 函数之前，请先去项目->属性->常规里面将使用的字符集设置成

使用多字节字符集。**3.2.4 设置颜色**

在图形显示方式下，下面 4 个标准函数可以用来设置或获取前景颜色和背景颜色。

**void setcolor(COLORREF color)** 将前景颜色设置为 color

注：前景颜色指的是在窗口上绘制的图形，输出的字符等等的颜色。

**void setbkcolor(COLORREF color)** 将背景颜色设置为 color

注：背景颜色包括窗口的背景和输出的文字的背景色。设置背景颜色以后，initgraph 函

数生成的窗口依旧是黑色的背景，要再调用一次 cleardevice 函数才能显示你所设置的背景

颜色。

**int getcolor(void)** 返回当前的前景颜色

**int getbkcolor(void)** 返回当前的背景颜色

其中，COLORREF 是图形方式下表达颜色的类型，实际上是一个 4 字节的数字，可以

用 RGB 函数给其赋值：

COLORREF color = RGB(140,40,60);

RGB 函数三个参数分别为颜色的红色部分、绿色部分、蓝色部分，每部分取值为 0~255

EasyX 定义了如下的符号常量，

BLACK（黑色）BLUE（蓝色）GREEN（绿色）CYAN（青色）RED（红色）MAGENTA

（洋红）BROWN（棕色）LIGHTGRAY（亮灰色）DARKGRAY（深灰）LIGHTBLUE（浅

蓝）LIGHTGREEN（浅绿）LIGHTCYAN（浅蓝绿）LIGHTRED（浅红）LIGHTMAGENTA、

YELLOW（黄色）WHITE（白色）

这些常量可以直接使用，不用再用 RGB 函数赋值。

**3.2.5 在图形显示方式下显示文字和数值**

printf、puts 等函数均不能在图形显示方式下显示文字或数字，需要用下面的函数：

**void outtext(LPCTSTR buf)** —— 在当前位置显示字符串 buf

**void outtextxy(int x, int y ,LPCTSTR buf)** —— 在（x, y）坐标位置显示字符串 buf

若要显示数值，也需把数值转换为字符串后才能使用这 2 个函数。在转换时使用下面的

函数会很方便：

(注：VC2008 中，使用以上两个函数需要将工程中的字符集设定为”多字节字符集”)

**sprintf(char \*buf, char \*format, …)** 它的功能与 printf 的几乎相同，只是把输出结果输

出到字符数组 buf 中，而不是屏幕上。调用时，除第 1 个参数是一个数组的首地址外，后面

的参数与 printf 完全一样。例如：

char buf[100];

int x=10;

float y=12.345;

sprintf(buf, “x=%3d y=%8f” , x, y);

即可把 x、y 按第 2 个参数指定的格式输出到字符数组 buf 中，之后就可把 buf 中的字

符串用 outtext 输出到图形显示方式下的屏幕上，与用 printf 函数向屏幕输出的内容、格式完

- 15 -- 16 -

全一样。

**3.2.6 绘制图形**

下表是 EasyX 提供的部分绘制基本图形的标准函数。

函数原型

功能（除 putpixel 外均使用当前前景色）

**void putpixel(int x,int y,COLORREF**

**color);**

用 color 颜色在(x,y)象素上画一个点

**COLORREF getpixel(int x,int y);**

返回在(x,y)象素上的颜色值

**void line(int x1,int y1,int x2,int y2);**

画一条从（x1,y1）到（x2,y2）的直线

**void lineto(int x,int y);**

画一条从当前位置到（x,y）的直线

**void linerel(int dx,int dy);**

画一条从当前位置(x,y)到(x+dx,y+dy)的直线

**void circle(int x,int y,int radius);**

画一个圆心位于（x,y），半径为 radius 的圆形

**void rectangle(int x1,int y1,int x2,**

**int y2);**

画一个左上角为(x1,y1)，右下角为(x2,y2)的矩形框

**void bar(int x1,int y1,int x2,int y2);**

画一个左上角为(x1,y1)，右下角为(x2,y2)的矩形

条，没有边框，用当前颜色和填充模式填充这个

矩形条

在绘制各种基本图形之前，可以对所画线型进行设置。设置线型的标准函数原型为：

**void setlinestyle(int linestyle,unsigned upattern=NULL,int thickness=1);**

其中，linestyle 表示线的形状，它可以是：SOLID\_LINE（实线）、DOTTED\_LINE（点

虚线）、DASHED\_LINE（长虚线）等常量；thickness 表示线宽，它可以是：NORM\_WIDTH

（一个像素宽）和 THIC\_WIDTH（三个像素宽）；upattern 表示用户自定义的线型，当第一

个参数为 PS\_USERSTYLE 时该参数有效。函数调用时，后两个参数可以不传参，系统将以

默认的 NULL 和 1 传入。

**3.2.7 封闭图形的填充**

填充就是用规定的颜色和模式填满一个封闭图形。

设置填充模式和颜色的函数原型为：

**void setfillstyle(COLORREF color,int pattern,const char \*pupattern=NULL);**

其中的 pattern 取值可以从解决方案管理器->外部依赖项->EasyX.h 之中查询而得。

第三个参数在第二个参数为 PATTERN\_FILL 的时候可用，表达用户自定义的填充类型，

其指针应该指向一个长度为 8 的 char 数组，每个 char 变量由 8 位二进制表示，从低位到高

位表示从左到右，从第 1 个元素到第 8 个元素表示第一行到第八行，所以该参数表示一个- 17 -

8x8 的点阵。

在设置填充模式和颜色之后，可以利用 floodfill()函数对封闭图形进行填充。该函数的

原型为：

**void floodfill(int x,int y,COLORREF border);**

其中，（x,y）是封闭图形中的**任意一点**，border 是**边界的颜色**。

**注意**：（x,y）必须是封闭图形中的任意一点，若是图形外的一点，则会在封闭图形外面

涂色。

**3.3 制造动画**

（1）基本的动画的实现

实现动画的基本思路：绘制一帧图像->暂停一定毫秒->绘制下一帧图片。

**3.3.1 擦除/重绘法**

#include <graphics.h>

void main()

{

initgraph(640, 480);//初始化图形方式

setbkcolor(BLACK);

cleardevice();

int i = 0;

for (i = 10; i < 200; ++i)

{

circle(i, 50, 20);//在新的位置画出新的图形

Sleep(20);

clearcircle(i, 50, 21);//将之前画的擦除

}

closegraph();//关闭图形方式

}

/\*如代码所示，程序通过绘图->暂停->擦除->重绘的流程来实现了一个圆移动的基本动画\*/

备注：Clearcircle 函数可以通过 cleardevice 函数（清空屏幕）来实现，但是在屏幕中有

许多其它图像的时候，这样会把其它图像删除- 18 -

**3.3.1.1 实现图像在背景上面的移动**

如下面代码所示，如果采用上面介绍的方法的话，移动的圆形会擦除作为背景的矩形。这显

然不是我们想要看到的。

#include <graphics.h>

#include <stdlib.h>

void main()

{

initgraph(640, 480);//初始化图形方式

setbkcolor(BLACK);

cleardevice();

int i = 0;

rectangle(100,20, 200, 200);

for (i = 10; i < 200; ++i)

{

circle(i, 50, 20);//在新的位置画出新的图形

Sleep(20);

clearcircle(i, 50, 21);//将之前画的擦除

}

closegraph();//关闭图形方式

}

那么如何解决这种局面呢？幸亏我们拥有 IMAGE 对象以及相关的 get/putimage 函数。这两

个函数结合起来，我们可以先用 getimage 来保存即将画图的那片区域，然后就可以肆无忌

惮地在这片区域画图了。之后我们只需用 putimage 函数把保存的图像归位，就可以清除之

前的绘图。之后便可以保存下一个绘图区域的背景图像...以此类推

具体实现如下：

#include <graphics.h>

void main()

{

initgraph(640, 480);//初始化图形方式

setbkcolor(BLACK);

cleardevice();- 19 -

int i = 0;

rectangle(100, 20, 200, 200);

IMAGE temp;//定义一个 IMAGE 对象

for (i = 20; i < 500; ++i)

{

getimage(&temp, i-21, 19, i+22, 71);//预先保存绘图区域原有的图像

circle(i, 50, 20);//在新的位置画出新的图形

Sleep(20);

putimage(i - 21, 19, &temp);//用覆盖的方式代替擦除

}

closegraph();//关闭图形方式

}

/\*如代码所示，程序通过预先保存绘图区->绘图->等待->用预先保存的图像覆盖绘制的图像

来实现图像在背景上面的移动。这样圆形就可以随意移动而不会擦除背景中的矩形了\*/

另一个例子：

#include <graphics.h>

#include <conio.h>

void main()

{

initgraph(1024,768);//初始化图形方式

setcolor(WHITE);//设置线条颜色为白色

setfillstyle(RED);//设置填充颜色为红色

IMAGE img2;//定义一个 IMAGE 对象，用于存放图像

getimage(&img2,60,60,140,140);//将对应的底部圆圈图像保存到 img2 中

IMAGE img1;//定义一个 IMAGE 对象，用于存放图像

circle(100,100,40);//画一个圆圈

floodfill(100,100,WHITE);//用红色填充它

getimage(&img1,60,60,140,140);//将这个画好的圆圈图像保存到 img1 中

cleardevice();//清空屏幕，准备开始动画

for(int i=50; i<600; i++)

{

putimage(i,100,&img1);//将 img1 的内容输出到屏幕上

Sleep(10);//暂停毫秒- 20 -

putimage(i,100,&img2);

}

closegraph();//关闭图形方式

}

上面的程序显示了一个圆从左运动到右。getimage 有很多种使用方法，根据参数的不同函

数会有不同的效果。getimage/putimage 函数的具体使用方法可参照 EasyX 的帮助文档。

**3.3.2 双缓冲法**

擦除重绘的局限性很大，而且直接使用 loadimage 函数和 putimage 函数会导致 gif 格式

图片的透明背景变成黑色（当然擦除重绘也可以把这个背景变成透明只是不使用 putimage）。

这样就没有办法达到彩色背景上的动画效果，因为不擦除会留下残影，擦除会闪烁。那么这

时候就需要用到双缓冲技术。

首先介绍什么是双缓冲技术，我个人的理解是在内存中绘图并直接覆盖当前窗口的图

像，也就是说除了覆盖当前窗口的图像以为，所有对图像的处理都是在内存中进行的，包括

擦除，重绘，读取图片，把图片放到指定位置等。

接下来用一个简单的例子实现双缓冲动画：

//创建两个HDC数据结构，前者用作获取当前窗口的绘图设备，后者用作获取内存的绘图设

备

HDC current\_hdc = NULL, buffer\_hdc = NULL;

//这个HDC结构 用作读取磁盘中的图片

HDC srcDC = NULL;

//创建两个IMAGE对象，前者存前景（人物），后者存（背景）

IMAGE fore = NULL, back = NULL;

//获取当前窗口DC

current\_hdc = GetDC(GetHWnd());

//获取内存DC

buffer\_hdc = CreateCompatibleDC(NULL);

//创建与指定的设备环境相关的设备兼容的位图

HBITMAP bmp = CreateCompatibleBitmap(current\_hdc, fbreadth, fheight);

//用bmp替换内存DC中的位图

SelectObject(buffer\_hdc, bmp);

//读取背景到srcDC

loadimage(&back, “背景路径”);

srcDC = GetImageHDC(&back);

//将背景绘入内存DC

TransparentBlt(buffer\_hdc, 0, 0, 前景宽度, 前景高度, srcDC, 前景图片左上角横坐标,

前景图片左上角纵坐标, 前景宽度, 前景高度, 0x000000);//将前景载入srcDC，注意这时的srcDC已经和之前的背景没有任何关系了

loadimage(&fore, “前景路径”);

srcDC = GetImageHDC(&fore);

//将前景绘入内存DC

TransparentBlt(buffer\_hdc, 0, 0, 前景宽度, 前景高度, srcDC, 0, 0, 前景宽度, 前景

高度, 0x000000);

//将内存DC中的图像按以像素为单位，绘入当前绘图窗口

BitBlt(current\_hdc, 前景图片左上角横坐标, 前景图片左上角纵坐标, 前景宽度, 前景

高度, buffer\_hdc, 0, 0, SRCCOPY);

//释放内存

DeleteDC(buffer\_hdc);

DeleteDC(current\_hdc);

DeleteDC(srcDC);

DeleteObject(bmp);

这里重点说下TransparentBlt函数和BitBlt函数：

**TransparentBlt函数：**

函数功能：该函数对指定的源设备环境中的矩形区域像素的颜色数据进行位块

（bit\_block）转换，并将结果置于目标设备环境。

函数原型：BOOL TransparentBlt(HDC hdcDest, int nXOriginDest, int nYOriginDest,

int nWidthDest, int hHeightDest, HDC hdcSrc, int nXOriginSrc, int nYOriginSrc, int

nWidthSrc, int nHeightSrc, UINT crTransparent)；

参数：

nXOriginDest：指定目标矩形左上角的X轴坐标，坐标以逻辑单位表示。

nYOriginDest：指定目标矩形左上角的Y轴坐标，坐标以逻辑单位表示。

nHeightDest：指定目标矩形的高度。

nXOriginSrc：指定源矩形（左上角）的X轴坐标，坐标以逻辑单位表示。

nYOriginsrc：指定源矩形（左上角）的Y轴坐标，坐标以逻辑单位表示。

nWidthSrc：指定源矩形的宽度。

nHeightSrc：指定源矩形的高度。

crTransparent：源位图中的RGB值当作透明颜色。(用RGB(0,0,0)也就是黑色不行)返回

值：如果函数执行成功，那么返回值为TRUE；如果函数执行失败，那么返回值为FALSE。

- 21 -关于最后一个参数，我的实例中就是选黑色做透明色，因为GIF图片的透明背景被载入

IMAGE对象后会变成纯黑色，所以以黑色做背景效果很好。并且，这里推荐使用GIF格式图片

做前景，因为RGB（0,0,0）是纯黑，稍有偏差都会出现毛边等影响视觉的效果。

**BitBlt函数：**

函数功能：该函数对指定的源设备环境区域中的像素进行位块（bit\_block）转换，以

传送到目标设备环境。（功能看着是一样的）

函数原型：BOOL BitBlt( HDC hdcDest, int nXDest, int nYDest, int

nWidth, int nHeight, HDC hdcSrc, int nXSrc, int nYSrc, DWORD

dwRop);

hDestDC：指向目标设备环境的句柄。

x：指定目标矩形区域左上角的X轴逻辑坐标。

y：指定目标矩形区域左上角的``````Y轴逻辑坐标。

nWidth：指定源在目标矩形区域的逻辑宽度。

nHeight：指定源在目标矩形区域的逻辑高度。

hSrcDC：指向源设备环境的句柄。

xSrc：指定源矩形区域左上角的X轴逻辑坐标。

ySrc：指定源矩形区域左上角的Y轴逻辑坐标。

dwRop：指定光栅操作代码。这些代码将定义源矩形区域的颜色数据，如何与目标矩形

区域的颜色数据组合以完成最后的颜色。

下面列出了一些常见的光栅操作代码：

BLACKNESS：表示使用与物理调色板的索引0相关的色彩来填充目标矩形区域，（对缺省

的物理调色板而言，该颜色为黑色）。

DSTINVERT：表示使目标矩形区域颜色取反。

MERGECOPY：表示使用布尔型的AND（与）操作符将源矩形区域的颜色与特定模式组合一

起。

MERGEPAINT：通过使用布尔型的OR（或）操作符将反向的源矩形区域的颜色与目标矩形

区域的颜色合并。

NOTSRCCOPY：将源矩形区域颜色取反，于拷贝到目标矩形区域。

NOTSRCERASE：使用布尔类型的OR（或）操作符组合源和目标矩形区域的颜色值，然后

将合成的颜色取反。

PATCOPY：将特定的模式拷贝到目标位图上。

PATPAINT：通过使用布尔OR（或）操作符将源矩形区域取反后的颜色值与特定模式的颜

色合并。然后使用OR（或）操作符将该操作的结果与目标矩形区域内的颜色合并。

PATINVERT：通过使用XOR（异或）操作符将源和目标矩形区域内的颜色合并。

SRCAND：通过使用AND（与）操作符来将源和目标矩形区域内的颜色合并。

SRCCOPY：将源矩形区域直接拷贝到目标矩形区域。

SRCERASE：通过使用AND（与）操作符将目标矩形区域颜色取反后与源矩形区域的颜色

值合并。

SRCINVERT：通过使用布尔型的XOR（异或）操作符将源和目标矩形区域的颜色合并。

SRCPAINT：通过使用布尔型的OR（或）操作符将源和目标矩形区域的颜色合并。

- 22 -- 23 -

WHITENESS：使用与物理调色板中索引1有关的颜色填充目标矩形区域。（对于缺省物理

调色板来说，这个颜色就是白色）。

这个函数说的不是太多，我们只是用它实现双缓冲，所以按照实例做就可以了。

**3.4 响应消息**

响应消息说的简单一点，就是对程序接受的信息做出反应。比如游戏中，我们按”←”

游戏人物会向左移动这就是一种对键盘消息的响应。

**3.4.1 键盘消息响应**

键盘上大多数按键对应了一个 ASCII 码，getch()便可接收按键返回这个 ASCII 码，并且

getch()是一个一个字符接收没有回显（即将输入内容显示到显示屏上）非常适合用于做游戏。

当然也不是说其他函数就不合适。接下来我举一个简单的键盘消息响应的例子。

#include "stdafx.h"

#include <conio.h>

#include <stdlib.h>

#include <stdio.h>

#define ESC 27 //宏替换将ESC替换成27 这里只是为了提高程序可读性

int main(void)

{

char ch = 0;

while (true) //无限循环 制作游戏时常用

{

printf("Hello World\n");

if (kbhit()) //检测是否有键盘输入

ch = getch();

if (ch == ESC) //因为键盘上Esc对应整形值27所以 在按下Esc时 将会跳出循环

break;

}

printf("End!\n");

system("pause");//按任意键退出

return 0;

}

这个程序会不断输出"Hello World"直到用户按下 Esc。这里使用了 kbhit()监测是否有键盘输入（有

则返回非 0 值，无则返回 0 值），getch()获取键盘输入，返回对应 ASCII 码。

接着是一个稍微复杂的例子：- 24 -

#include <windows.h>

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

int main(void)

{

while (true)

{

printf("Hello World!\n");

if ((GetAsyncKeyState(VK\_F1) & 1) == 1)

break;

}

printf("End!\n");

system("pause");

return 0;

}

这个程序与上个程序唯一的不同在于用 GetAsyncKeyState()函数代替了 kbhit()和 getch(),

它的优势在于可以对键盘上所有的键做出响应。GetAsyncKeyState()，接受一个虚拟键码（即键盘

对应的一个整型数值），监测虚拟键状态，返回一个 16 位数值。当从上次调用到本次调用

之间，该按键被按过，则这个数值的第一个二进制位被设为 1，否则为 0（第一次调用均为

0）。若该按键目前正在处于被按下状态，则第 16 位二进制位为 1，否则为 0。

换句话说第一次调用 GetAsyncKeyState(VK\_F1)我们没有没有按下 F1，该函数返回 0，

在第二次调用之前我们按下了 F1，该函数会返回 16 位二进制数的 0000 0000 0000 0001，

如果在 F1 被按下的状态下调用了该函数，则会返回即16 位二进制数的1000 0000 0000 0000。

(GetAsyncKeyState(VK\_F1) & 1）中&为“按位与”运算符意为二进制位上的数同为 1 则结果的二

进制位上的数位 1，举个例子：

1011&0010 的结果就是 0010，注意这都是二进制表示。

那么怎么在代码中使用这个运算符呢？还是刚才这个式子，我们需要将它转化为 10 进制数，1011

的十进制表示为 2^3+2^1+2^0=8+2+1=11，0010 的十进制表示为 2^1=2。那么原表达式的写法即位 11&2

结果为 2。

那么回到(GetAsyncKeyState(VK\_F1) & 1），其中按位与运算符的作用在于提取二进制第一位如

果想提取第 16 位请&32768。所以 if 中的判断条件即为 F1 被按下则停止输出”Hello World”转为输

出”End!”。

**3.4.2 鼠标消息响应**

鼠标消息响应跟键盘消息响应一样，只不过是对鼠标的动作做出反应。关于鼠标的信息- 25 -

有鼠标左键点击、鼠标右键点击、鼠标坐标等等。这些信息均可通过特定函数单独获得，但

是我们的目的是制作课设，所以推荐直接使用 GetMouseMsg()获取鼠标事件，这样可以获

得上述所有有关鼠标的信息，并保存在一个 MOUSEMSG 结构体中。

其中，MOUSEMSG 结构体的定义：

struct MOUSEMSG

{

UINT uMsg; // 当前鼠标消息

bool mkCtrl; // Ctrl 键是否按下

bool mkShift; // Shift 键是否按下

bool mkLButton; // 鼠标左键是否按下

bool mkMButton; // 鼠标中键是否按下

bool mkRButton; // 鼠标右键是否按下

int x; // 当前鼠标 x 坐标

int y; // 当前鼠标 y 坐标

int wheel; // 鼠标滚轮滚动值

}

**成员（在这里就是变量）：**

uMsg：

指定鼠标消息类型，可为以下值：

**值**

**含义**

WM\_MOUSEMOVE

鼠标移动消息。

WM\_MOUSEWHEEL

鼠标滚轮拨动消息。

WM\_LBUTTONDOWN

左键按下消息。

WM\_LBUTTONUP

左键弹起消息。

WM\_LBUTTONDBLCLK

左键双击消息。

WM\_MBUTTONDOWN

中键按下消息。

WM\_MBUTTONUP

中键弹起消息。

WM\_MBUTTONDBLCLK

中键双击消息。

WM\_RBUTTONDOWN

右键按下消息。

WM\_RBUTTONUP

右键弹起消息。

WM\_RBUTTONDBLCLK

右键双击消息。

mkCtrl- 26 -

Ctrl 键是否按下

mkShift

Shift 键是否按下

mkLButton

鼠标左键是否按下

mkMButton

鼠标中键是否按下

mkRButton

鼠标右键是否按下

x

当前鼠标 x 坐标（物理坐标）

y

当前鼠标 y 坐标（物理坐标）

wheel

鼠标滚轮滚动值，为 120 的倍数。

并附上一个简单的使用案例：

#include "stdafx.h"

#include "graphics.h"

#include <conio.h>

#include <stdio.h>

#include <process.h>

int main()

{

initgraph(1024, 768);

MOUSEMSG m = GetMouseMsg();//设置鼠标信息

char ch[20];//字符临时变量

while (true)

{

if (MouseHit())//如果检测到鼠标事件

m = GetMouseMsg();//获取鼠标事件

switch (m.uMsg)//对事件进行分类

{- 27 -

case WM\_MOUSEMOVE://如果是鼠标移动事件

sprintf(ch, "x:%d y:%d ", m.x, m.y);//将坐标输入到ch

outtextxy(100, 100, ch);//将字符串ch输出到(100,100)

break;

case WM\_LBUTTONDOWN://如果是点击鼠标左键，在点击位置出现数组ch中的坐标

outtextxy(m.x, m.y, ch);

break;

case WM\_RBUTTONDOWN://如果是点击鼠标右键

return 0;//退出

default://如果遇到其他事件

continue;//继续循环

}

}

closegraph();

return 0;

}

这个程序的效果可以概括为输出鼠标当前坐标，点击左键时在当前鼠标位置输出当前的

鼠标坐标，右键退出程序。

**3.4.3 系统时间**

**3.4.3.1 time 函数及应用**

有的课设内容需要做时间的限制，所以接下来将介绍获得系统时间的方法。可以用 time、

localtime 函数等来获得系统时间，也可以用 time 来进行时间控制。我们比较熟悉 time()，所

以接下来先介绍 time，一般的计时功能 time()完全能够胜任。

函数 time 的原型在 time.h 中，其原型为：

**time\_t time(time\_t \*t);**

其中 time\_t 类型就是 long 类型，在 time.h 中已经声明。

time 函数返回一个长整数，其值以秒为单位。2 次调用此函数，它们返回值的差，即为

2 次调用之间的时间差（秒数）。使用时，参数可使用空指针。

举一个简单的例子：

#include "stdafx.h"

#include <stdio.h>

#include <time.h>

#include <Windows.h>

int main()

**批注 [A1]:** 响应- 28 -

{

time\_t t1, t2;

int interval;

t1 = time(NULL);

for (int i = 0; i < 30000; i++)

printf("%d\n",i);

t2 = time(NULL);

interval = t2 - t1;

printf("%d\n", interval);

system("pause");

return 0;

}

这个程序的作用是输出“输出 3 万次”的时间。

需要注意的是 time()返回值得单位是秒，只能进行最小以秒为单位的时间控制。

**3.4.3.2 localtime 函数及应用**

接下来我们介绍 localtime()：

函数 localtime 可以把用长整型数值表示的时间变为年、月、日、星期、小时、分、秒

等。其原型（在 time.h 中）是：

**struct tm \*localtime(time\_t \*t);**

返回值是指向 **struct tm** 结构类型的指针。

结构类型 struct tm 已经在 time.h 中声明，内容是：

struct tm

{

int tm\_sec; // 秒

int tm\_min; // 分

int tm\_hour; // 小时

int tm\_mday; // 日

int tm\_mon; // 月

int tm\_year; // 年

int tm\_wday; // 星期几（0表示星期天）

int tm\_yday; // 本年天数

int tm\_isdst; // 是否使用了夏时制

};

函数参数是一个指针，指向一个存放用长整型数值表示的时间的变量，通常是指向 time

函数的返回值。至于为什么要用 time 的返回值，因为 time 函数返回的是从 1970 年 1 月 1

日 00:00:00 到现在的此时此刻所经过的秒数。所以 localtime 函数只是将这个数值计算成- 29 -

结构体里的变量并赋给结构体，因为 time()的最小分度是秒，所以 localtime()最小分度也

是秒。至于 localtime()对于课设的作用，可以在存档的时候记个时间吧。

**3.4.3.3 clock 函数及应用**

time 函数是获取系统的实际时间，而 clock 函数是获取进程使用的 cpu 时间单元总数。

函数原型：clock\_t clock(void);

原型在：time.h 头文件中

功能：用于测试 2 次调用之间的时间间隔。

返回值类型：clock\_t，（实际就是 long 类型）已经在头文件中声明，程序中可直接使用。

返 回 值 ： 从 程 序 运 行 时 到 本 函 数 被 调 用 经 过 的 处 理 器 时 间 。 此 值 除 以

CLOCKS\_PER\_SEC（即 18.2，已经在头文件中声明）则换算为秒数。

#include "stdafx.h"

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void isleep(clock\_t wait);

void main(void)

{

long i = 60000000L;

clock\_t start, finish;

double duration;

// Delay for a specified time.

printf("Delay for three seconds\n");

isleep((clock\_t)3 \* CLOCKS\_PER\_SEC);

printf("Done!\n");

//\* Measure the duration of an event.

printf("Time to do %ld empty loops is ", i);

start = clock();

printf("%ld\n", start);

while (i--)

;

finish = clock();

printf("%ld\n", finish);

duration = (double)(finish - start) / CLOCKS\_PER\_SEC;

printf("%10.2f seconds\n", duration);

}

// Pauses for a specified number of milliseconds.

void isleep(clock\_t wait)- 30 -

{

clock\_t goal;

goal = wait + clock();

while (goal > clock())

;

}

其中，2 次调用 clock()函数获得的返回值的差再除以 CLOCKS\_PER\_SEC，就得到 2 次

调用 clock()函数所间隔的秒数。这个函数对课设来说用处不大。

**3.4.4 产生声音**

Windows 提供了产生声音的 API 函数，可以利用计算机自带的蜂鸣器（不使用声卡和

音响设备）发出不同频率的简单声音。下面介绍 2 个函数，其原型在 windows.h 中。

**Beep 函数声名：**

bool Beep(int frequency,int Duration);

Beep()以 frequency 为频率使蜂鸣器发声,发声的长度为 Duration 毫秒。

在使用时要注意：频率 frequency 应选择在人耳可以听到的频率（大约是 25——5000）

范围内。

下面的程序用 50、100、150、…、1000 的频率发声，每个频率的声音保持 0.5 秒。最

后关闭蜂鸣器。

#include "stdafx.h"

#include <windows.h>

void main()

{

int freq;

for (freq = 50; freq<1000; freq += 50){

Beep(freq, 500);

}

}

需要注意的是有的电脑听不到，并且课设也基本不会用到这个函数，但是这里还是需要提一

下。

接下来再介绍一个更有用一点的函数：PlaySound 函数，它同样是 API 函数。

**PlaySound 函数声名：**

BOOL PlaySound(LPCSTR pszSound, HMODULE hwnd, DWORD fdwSound);

参数 pszSound 是指定了要播放声音的字符串，该参数可以是 WAVE 文件的名字，或是

WAV 资源的名字，或是内存中声音数据的指针，或是在系统注册表 WIN.INI 中定义的系统事

件声音。如果该参数为 NULL 则停止正在播放的声音。

参数 hwnd 是应用程序的实例句柄，除非 pszSound 的指向一个资源标识符（即 fdwSound- 31 -

被定义为 SND\_RESOURCE），否则必须设置为 NULL。

参数 fdwSound 是标志的组合，如下表所示。若成功则函数返回 TRUE，否则返回 FALSE。

**播放标志以及含义：**

SND\_APPLICATION

用应用程序指定的关联来播放声音。

SND\_ALIAS

pszSound 参数指定了注册表或 WIN.INI 中的系统事件的别名。

SND\_ALIAS\_ID

pszSound 参数指定了预定义的声音标识符。

**SND\_ASYNC**

用异步方式播放声音，PlaySound 函数在开始播放后立即返回。（理解成多线程就好，意思

就是不和主函数同步，即不需要播放完成后再执行下一个语句，课设的时候非常有用）

**SND\_FILENAME**

pszSound 参数指定了 WAVE 文件名。

**SND\_LOOP**

重复播放声音，必须与 SND\_ASYNC 标志一块使用。

SND\_MEMORY

播放载入到内存中的声音，此时 pszSound 是指向声音数据的指针。

SND\_NODEFAULT

不播放缺省声音，若无此标志，则 PlaySound 在没找到声音时会播放缺省声音。

SND\_NOSTOP

PlaySound 不打断原来的声音播出并立即返回 FALSE。

SND\_NOWAIT

如果驱动程序正忙则函数就不播放声音并立即返回。

**SND\_PURGE**

停止所有与调用任务有关的声音。若参数 pszSound 为 NULL，就停止所有的声音，否则，停

止 pszSound 指定的声音。

SND\_RESOURCE

pszSound 参数是 WAVE 资源的标识符，这时要用到 hmod 参数。

SND\_SYNC

同步播放声音，在播放完后 PlaySound 函数才返回。（这个做课设的时候，我觉得是使用不

到）

推荐编写课设时的三个用法：

//用作游戏背景音乐 异步播放 播放指定WAV文件 重复播放

PlaySound("相对路径", NULL, SND\_ASYNC | SND\_FILENAME | SND\_LOOP);

//触发指定音频 异步播放 播放指定WAV文件

PlaySound("相对路径", NULL, SND\_ASYNC | SND\_FILENAME );

//停止播放指定音频 第一个参数为NULL是停止全部音频

PlaySound("相对路径", NULL, SND\_FILENAME | SND\_PURGE);当然还有许多其他用法，上面的参数已经说得很清楚了，可以自己尝试一下，达到自己想要的效果。