# 龙神录编程馆系列教程翻译

## 版权规约及翻译版权附加申明

### 1、版权规约

2007 12/15 更新

※注意※

本人收到了许多关于本文与程序被直接作为毕业作品而提交的报告。

请遵守规定利用本文章。

首先，对于不知道东方的人，在这里事先说明一下，本游戏（译者注：即龙神录）与东方Project系列游戏很类似。

其次，东方Project的游戏都非常出色，如果您有兴趣的话，请务必玩玩东方系列（在东方的主页上也有体验版）。

本站是为了想要学习STG的编程方法的人而制作的网站。

其次，当然也是因为对东方和弹幕有爱而献上这个网站。

看了本站的教程，就能够制作和四圣龙神录类似的STG了。

本文中展示的程序只要在常识范围内请尽情使用。

不过，如果您需要以本文中的程序为参考而发表/发售，抑或向教育机关/企业提交，请向我报告一次。

至于游戏中的素材图片，除了龙的图片以外，其它的都可以二次使用。不过，请在理解了这种发布方式是在和东方相同的许可的前提下使用。

请不要二次使用游戏中的音效/效果音。

（您可以选择自己制作，也可以在The Match Maker寻找效果音，在炼狱庭院寻找音乐，在那里都能寻找到不错的素材。）

|  |  |
| --- | --- |
|  | |
| 使用场合 | 限制 |
| 如果是您个人娱乐的场合 | 您可以任何限制地使用这里的东西。 |
| 如果您利用这里的源代码制作游戏并免费发布的场合 | 请在您的主页或者在游戏内部容易被看见的地方介绍一下龙神录程序编程馆。 |
| 如果是利用这里的源代码制作游戏并出售的场合 | 如果您事先联系本人的话是可以使用的。 |
| 利用这里的源代码提交制作好了的毕业作品、或者是在教育机关/企业等投递作品的场合。 | 在使用的时候请参考一下注释1，在邮件中添加上您的投递目标并联系我。 |
| 向动画站点投稿的场合 | 只要您将使用了本站的源代码这个意思说明了（比如将URL贴出来），无论怎么投稿都没有问题。 |

※管理员对于规定的想法※

对以上海爱丽丝幻乐团做成的游戏为基础做成的游戏，让我来做这样那样的规定实在有些不合情理，我也是喜欢二次创作的人，因此我尽量不去定下非常精细的规则。

如果您想使用程序的话那就尽情使用吧。

不过，如果我偶尔能够收到类似“我参考了馆里的教程制作了游戏！来玩玩吧！”这样子的报告，我会很高兴的。

不管您做了什么游戏，都请让我玩玩。

不过我希望您能尽量避免将本文本作品的一份拷贝直接作为课题、毕业作品之类的东西提交出去的行为。

如果您对程序的内容有不了解的地方、又或者是不知道如何理顺思路的时候，请在揭示板上随便询问。

如果您要作为参考资料记录的话，请参考以下情报。

制作者：Dixq

社团名：RemicalSoft

站点名：龙神录编程馆

在本游戏中使用的库为DX Library。

著作権表示

DX Library Copyright (C) 2001-2006 Takumi Yamada.

如果您要详细DX Library的著作权，请在其主页上浏览：

http://homepage2.nifty.com/natupaji/DxLib/dxlicense.html

联络地址：dixqhp@gmail.com

注释1：

如果是提交专业学校的毕业作品或者作为某种产品向教育机关/企业提交的情况下，请提前联系我。

此时请在联络内容中添加上您的提交目标和您的名字。

### 2、翻译版权附加申明

1）“龙神录编程馆”系列教程由Dixq本人授权Irisine翻译，转载请保留此信息。

2）“龙神录编程馆”系列教程翻译由以下人员完成，翻译版权归翻译者本人所有，如需转载请保留翻译中的任何部分。请勿用于商业用途，因译文所产生的任何纠纷翻译者概不负责。

翻译人员：Hui

3）“龙神录编程馆”系列教程所有源代码及所涉及到的版权知识均归Dixq本人所有。

4）限于翻译人员水平有限，如有错误，请批评指出。

## 序章

龙神录编程馆以“简单易懂”为口号，旨在将射击游戏切实地介绍给大家。

如果您切实地从头到尾地学习了这里的教程，那么您最终也能够做出龙神录这种话程度的射击游戏。

如果您想要了解四圣龙神录请点击这里。

前往四圣龙神录公式网页

龙神录馆使用了基于DirectX的库DX Library。

您需要的最基本的条件就是了解基本的DX Library的使用方法，以及基本的C语言的知识。

我会尽量避免使用晦涩难懂的词汇，使初学者也能非常容易接受地介绍开来。

不过有一点请容我拒绝。

本馆并非围绕游戏程序框架的开发而介绍。

这里并不会刻意考虑效率化、封装化，以及泛用性/可扩展性等，总之并不会使用复杂的构造结构，目的仅仅是对实现既定的功能。

因此，大多数的变量都会直接作为全局变量而使用，所以本馆的设计并不适合大型项目的开发。

至于具有效率性的设计方法，会预定在另外的程序馆中介绍。

作为最必要的知识，您至少需要知道最低程度的C语言的知识。例如：

* 结构体
* 数组
* 自定义函数
* 预处理

等知识。

如果感觉到知识点有点模糊，请在入门站点中再复习一遍。

至于DX Library的基本使用方法，推荐您在游戏编程馆再仔细学习一遍效果会更好。

如果有不明白的地方，请随便在揭示板提问。

那么，龙神录程序馆的讲座就此开始。

## 准备好项目和素材吧

作为基本要求，龙神录程序馆以Visual Studio 2005或者Visual Studio 2008作为最基本的使用要求。

当然，其它的编译器也可以编译，不过发布的工程是使用的Visual Studio的工程文件。

在下面的工程中，龙神录使用的素材已经都放了进去。

下载所有的项目文件。

文件夹的构成如下所示：

在上图中写着“1章”的文件夹中，用于放置许多追加的东西。

今后每追加一章就会追加相应的文件夹。

请点开每一章中的RyuJin.sln文件。

dat文件夹存放了图片、音乐、效果音等和游戏一起发布的数据。

mydat文件夹里放的是我自己添加的头文件、源文件、Photoshop的文件等自己使用的数据。

Include文件夹中放的是DX Library的文件夹。

请点开顶端的“RyuJin.sln”文件。

项目所需要的所有至少需要的文件都已经构成了。

观察解决方案管理器，会发现文件是这样子构成的。

哇，文件怎么这么多。

嘛，不用担心。现在我们什么都还没写呢。无论是哪个文件里面一行代码都没有写进去。

所以还没有必要去理解它们的意义。

要问为什么只有Hard Mode（译者注：玩过东方都知道这模式吧……），那是因为如果我们一开始就学会了做最复杂的弹幕的话，那么做简单的弹幕那就非常轻松了。

先做出最复杂的弹幕，然后减少子弹的数量或者将轨道进行简化，进行这样子的工作这会很容易，但反过来，先做出简单的弹幕，要将其复杂化，那可就不知道要怎么去增加子弹的数量才好了，因此为了增加子弹的数量，很容易破坏全体的平衡性，因此我们一开始就先做很复杂的弹幕吧。

这一小节仅仅能够解说“工程原来有这么一些文件”这一点已经足够了。

如果您很在意文件的构成，那么请参考下面对各个文件的介绍。

从图片中由上至下按顺序：

boss\_shotH.cpp

为Boss的弹幕而设置的文件（Hard）

boss\_shot.cpp

控制Boss弹幕的文件

enemy.cpp

控制杂兵运动的文件

enemy\_act\_pattern.cpp

控制杂兵运动模式的文件。

graph.cpp

与绘制相关的文件

graph\_back.cpp

与背景绘制相关的文件

shotH.cpp

为杂兵的射击而设置的文件（hard）

cshot.cpp

为自机的射击而设置的文件

shot.cpp

控制杂兵射击的文件

char.cpp

控制自机的文件

fps.cpp

控制FPS的文件

ini.cpp

初始化的文件

key.cpp

控制按键的文件

load.cpp

用于载入图片和音乐的文件

main.cpp

main

music.cpp

控制音乐的文件

out.cpp

计算碰撞判定的文件

define.h

定义用的头文件

function.h

extern声明用的头文件

GV.h

为全局变量设置的头文件

struct.h

定义结构体用的头文件

## 第二章 首先我们来执行一下吧

首先我们来执行一下吧。向工程的main.cpp文件里添入在游戏编程馆介绍过的“程序基本骨架”链接。

然后编译，执行一下吧。

--main.cpp变更--

#define GLOBAL\_INSTANCE

#include "../include/GV.h"

int Key[256];

int GetHitKeyStateAll\_2(int GetHitKeyStateAll\_InputKey[]){

char GetHitKeyStateAll\_Key[256];

GetHitKeyStateAll( GetHitKeyStateAll\_Key );

for(int i=0;i<256;i++){

if(GetHitKeyStateAll\_Key[i]==1) GetHitKeyStateAll\_InputKey[i]++;

else GetHitKeyStateAll\_InputKey[i]=0;

}

return 0;

}

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,LPSTR lpCmdLine, int nCmdShow ){

ChangeWindowMode(TRUE);//窗口模式

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初始化和里画面化

while(ProcessMessage()==0/\*消息处理\*/ && ClearDrawScreen()==0/\*清空画面\*/ && GetHitKeyStateAll\_2(Key)==0 /\*保存输入状态\*/&& Key[KEY\_INPUT\_ESCAPE]==0/\*没有按下ESC键\*/){

//这里！！

ScreenFlip();

}

DxLib\_End();

return 0;

}

接下来，窗口就出现了，按下Esc就结束程序。

到此为止，如果您已经成功完成了，那么说明您的准备已经完备了。

## 第三章 注意一下对全局变量的处理吧

刚才我们直接将处理按键输入的GetHitKeyStateAll\_2函数从样例中复制粘贴过来了。

按照前面那样，就必须得把名为Key的函数变成全局变量，但是我们希望要尽量要减少任何地方其值都会被重写的全局变量。

其次，就算是同一个全局变量，我们也尽量尝试使用能降低BUG产生的使用方法。

--key.cpp変更--

#include "../include/GV.h"

unsigned int stateKey[256];

int GetHitKeyStateAll\_2(){

char GetHitKeyStateAll\_Key[256];

GetHitKeyStateAll( GetHitKeyStateAll\_Key );

for(int i=0;i<256;i++){

if(GetHitKeyStateAll\_Key[i]==1) stateKey[i]++;

else stateKey[i]=0;

}

return 0;

}

int CheckStateKey(unsigned char Handle){

return stateKey[Handle];

}

在main文件和另外的key文件里面试着向上面那样写。

到目前为止，我们要想知道ESC键的状态的话，必须要检查Key[KEY\_INPUT\_ESCAPE]的值。

然而，将Key改为stateKey，为了在判断按键时不直接比较stateKey中对应的元素，我们向CheckStateKey函数传入按键的代码，此时如果能够返回输入状态，那么就可以减少按键值被错误地重写的危险性。

现在我们向上面实现的CheckStateKey函数传入按键的代码，这样就能够检查按键的输入状态了。

此外，如果要使用其它文件中实现的变量或者函数，那么就需要extern声明。

//处理现在的输入

extern int GetHitKeyStateAll\_2();

//返回传入的按键的编号的当前状态

extern int CheckStateKey(unsigned char Handle);

将这个写到main.cpp中，那么就可以在main文件中使用key文件中定义的函数了。

此外，把在上一小节写到的主循环中每次都会执行到的一系列处理进行整理，都放到名为ProcessLoop()的函数中。

执行的事情并没有改变。那么试着看看改变过后的main文件。

--main.cpp変更--

#define GLOBAL\_INSTANCE

#include "../include/GV.h"

//处理现在的输入

extern int GetHitKeyStateAll\_2();

//返回传入的按键的编号的当前状态

extern int CheckStateKey(unsigned char Handle);

//循环中必须执行的三大处理

int ProcessLoop(){

if(ProcessMessage()!=0)return -1; //过程处理如果发生错误返回-1

if(ClearDrawScreen()!=0)return -1; //如果画面清空错误返回-1

GetHitKeyStateAll\_2();//进行当前按键处理

return 0;

}

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nCmdShow){

ChangeWindowMode(TRUE); //窗口模式

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1; //初始化和里表面化

while(ProcessLoop()==0){//主循环

if(CheckStateKey(KEY\_INPUT\_ESCAPE)==1)break; //如果按下ESC键跳出循环

ScreenFlip();//里外画面翻转

}

DxLib\_End();//DX Libaray终止处理

return 0;

}

按下Esc键就结束程序运行。

## 第四章 试试描绘角色吧

接下来我们试着描绘一下角色吧。

要做到这一点，首先我们要定义角色数据的结构体，同时定义它的变量。

角色数据的结构体定义如下：

--struct.h变更--

//和角色相关的结构体

typedef struct{

int flag; //flag

int cnt; //计数变量

int power; //power

int point; //点数

int score; //分数

int num; //残机数

int mutekicnt; //无敌状态与计数变量

int shot\_mode; //射击模式

int money; //金钱

int img; //图像

int slow; //是否缓慢移动

double x,y; //坐标

}ch\_t;

结构体的的定义请写在struct.h文件中。

虽然一来就好像有很多成员，不过并不需要都记下。

一个角色说拥有的数据是各种各样的。比如power啊分数啊当前的射击模式之类的。

现在只是暂且把这些需要的东西都准备好。现在我们需要的也仅仅只有坐标，因此我们也就把它也放进结构体了。

需要注意的也就是结构体中有double类型的xy成员这一点而已。

接下来，让我们读入角色的图像。我们将图片的句柄（handle）放到img\_ch变量中。

我们把12张角色图像看为一组。在龙神录中，为了实现变身的效果，需要两组图像，因此我们定义变量

int img\_ch[2][12];

来使用。我们在main文件中定义变量、在下面的load.cpp中对其进行extern声明。

--load.cpp変更--

#include "../include/GV.h"

extern int img\_ch[2][12];

void load(){

LoadDivGraph( "../dat/img/char/0.png" , 12 , 4 , 3 , 73 , 73 , img\_ch[0] ) ;

}

请确认一下在上述过程中读入的画像。上面的函数中的读取过程的意思是，将总共12幅、横4竖3、每幅都长宽73-73像素的图像放入img\_ch[0]中。

如果不明白的话，请在DX Libaray的参考页面中确认函数。

那么我们修改一下执行描绘的graph.cpp文件。

和之前一样，变量的定义在main文件中进行，而在这里通过extern声明进行使用。

从现在开始，和描绘相关的函数将会越来越多，因此为了从main文件中能够很好地进行整体性地控制，我们调用graph\_main函数。

--graph.cpp変更--

#include "../include/GV.h"

extern ch\_t ch;

extern int img\_ch[2][12];

void graph\_ch(){

DrawRotaGraphF(ch.x,ch.y,1.0f,0.0f,img\_ch[0][ch.img],TRUE);

}

void graph\_main(){

graph\_ch();

}

接下来是重要的main文件，main.cpp。

--main.cpp変更--

#define GLOBAL\_INSTANCE

#include "../include/GV.h"

//处理现在的输入

extern int GetHitKeyStateAll\_2();

//返回传入的按键的编号的当前状态

extern int CheckStateKey(unsigned char Handle);

//载入数据

extern void load();

//绘画用的main

extern void graph\_main();

int img\_ch[2][12]; //角色图像12张　X2(变身用)

ch\_t ch; //角色数据

//循环必须处理的三大过程

int ProcessLoop(){

if(ProcessMessage()!=0)return -1; //过程处理如果发生错误返回-1

if(ClearDrawScreen()!=0)return -1; //如果画面清空错误返回-1

GetHitKeyStateAll\_2();//进行当前按键处理

return 0;

}

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nCmdShow){

ChangeWindowMode(TRUE);//窗口模式

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初始化和里画面化

load();//载入数据

while(ProcessLoop()==0){//主循环

graph\_main();//绘制main

if(CheckStateKey(KEY\_INPUT\_ESCAPE)==1)break;//按下ESC键就跳出循环

ScreenFlip();//里外画面翻转

}

DxLib\_End();//DX Library终止处理

return 0;

}

运行结果：

如果您的效果和上面一样的话那么本节的内容便完成了。您应该可以在左上角稍微看到角色。

啊，为什么main文件一下子看不懂了呢……

没关系，我会在下一章里面给大家漂漂亮亮的整理一下。

## 第五章 琢磨一下声明的方式吧

上一小节main文件被我们改得乱七八糟的，这样一来，根本弄不清楚到底在那里定义了变量，又在哪里使用了extern了。

这样一来，我们得好好琢磨一下变量的定义和extern的声明方式了。

我们稍微在这里复习一下上一章讲过的内容。

1、（这次设计的时候）在main文件即main.cpp文件中定义了全局变量。

2、如果想要在其它文件中调用的时候，那么在其它文件中进行extern声明。

大概就是这些内容了吧。

不过，在main文件中声明全局变量，而在其它文件中对同名的变量进行extern，总让人感觉没有什么效率。

反过来，如果我们能在所有的源文件中调用同一个头文件，而在这个头文件中对调用源进行条件分歧检查，进行适当的声明，那么就比较有效率了。

请看下述程序。

#ifdef GLOBAL\_INSTANCE

#define GLOBAL

#else

#define GLOBAL extern

#endif

GLOBAL int img;

前半は

前半部分意味着

如果定义了“GLOBAL\_INSTANCE”，那么有GLOBAL的地方就用空白代替。

如果没有定义“GLOBAL\_INSTANCE”，那么有GLOBALE的地方就用extern代替。

这样一来

如果我们直接写

GLOBAL int img;

的话，结果会是如何的，也即是是否会加上extern这一点就很容易知道了。

因为在main.cpp中调用的时候全局变量必须被定义，所以我们不希望加上extern。

也即是定义GLOBAL\_INSTANCE之后再调用就可以了。

一方面在其它文件中调用的时候我们希望加上extern。

也即是说不定义GLOBAL\_INSTANCE，也就是我们什么都不写直接调用就行了。

请注意main文件的最上方。这里写着：

#define GLOBAL\_INSTANCE

#include "../include/GV.h"

因为现在我们定义了GLOBAL\_INSTANCE然后再调用头文件，那么变量的声明就进行了。

另一方面，从别的文件中调用的变量不应该是定义的状态，因此添加上extern。

这样一来，就可以从别的文件中也调用这个头文件了。

再者，如果我们要在其它文件中调用已经实现了的函数，我们也有必要在main文件中对其进行声明。

由于进行了extern声明的话，无论是其实现的地方还是其调用的地方它们的先后顺序都不存在问题，因此在main文件中声明了的话，无论在哪里实现，只要进行了extern声明那就可以调用。

进一步，对在进行了条件分歧的GV.h中的function.h中进行和上面同样的GLOBAL声明。

嘛，只是看的话肯定不容易理解，所以像下面这样写。

--function.h変更--

//graph.cpp

//绘制main

GLOBAL void graph\_main();

//key.cpp

//处理当前的按键输入

GLOBAL int GetHitKeyStateAll\_2();

//返回传入的按键的编号的当前状态

GLOBAL int CheckStateKey(unsigned char Handle);

//laod.cpp

//载入数据

GLOBAL void load();

--GV.h变更--

#include "../../../include/DxLib.h"

#include "define.h"

#ifdef GLOBAL\_INSTANCE

#define GLOBAL

#else

#define GLOBAL extern

#endif

#include "function.h" //函数声明

//图像用变量声明部分

GLOBAL int img\_ch[2][12]; //角色图像12张　X2(变身用)

//结构体变量声明部分

GLOBAL ch\_t ch; //角色声明

然后我们把“main.cpp”、“graph.cpp”、“load.cpp”里的定义部分和extern声明都删掉。

编译执行，没有出错且效果和第四节中的效果相同。

今后我们如果要增加全局变量的话，就在GV.h中增加，而在main文件中要调用的其它文件中声明了的函数都在function.h中增加。

## 第六章 控制一下在主循环中调用的函数吧

在主循环中调用的函数，会因某个时刻的状况有不同的回馈，没错吧。

也就是说，我们必须要在主循环中控制我们调用的函数。

因此，我们事先准备好一个叫func\_state的变量，向其带入表示状态的值，以此为依据进行条件分歧吧。

由于我们希望将func\_state变量用作全局变量，首先向GV.h中如下追加：

--向GV.h进行如下追加：-

GLOBAL int func\_state;

我们顺便为下一章节做个准备，为其先准备一个初始化函数吧。

--在ini.cpp中进行以下追加--

void first\_ini(){

}

声明要在主循环中调用的函数的时候，需要在function.h里面写上。

--在function.h中进行如下追加--

GLOBAL void first\_ini();

在主循环中以准备好了的func\_state为一句，我们使用switch语句进行条件分歧判断。

--main.cpp変更--

#define GLOBAL\_INSTANCE

#include "../include/GV.h"

//循环必须处理的三大过程

int ProcessLoop(){

if(ProcessMessage()!=0)return -1; //过程处理如果发生错误返回-1

if(ClearDrawScreen()!=0)return -1; //如果画面清空错误返回-1

GetHitKeyStateAll\_2();//进行当前按键处理

return 0;

}

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nCmdShow){

ChangeWindowMode(TRUE);//窗口模式

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初始化和里画面化

while(ProcessLoop()==0){//主循环

switch(func\_state){

case 0:

load(); //载入数据

first\_ini();//最开始的初始化

func\_state=100;

break;

case 100:

graph\_main();//绘制main

break;

default:

printfDx("错误的func\_state\n");

break;

}

if(CheckStateKey(KEY\_INPUT\_ESCAPE)==1)break;//如果按下ESC键则跳出循环

ScreenFlip();//里外画面翻转

}

DxLib\_End();//DX Library终结处理

return 0;

}

从第四节开始我们的运行结果就没有改变。和以前一样执行的话一样会成功。

## 第七章 让我们把面板显示出来吧

我们的角色老是在左上角处于被切了一半的状态，看起来很没意思，这一节就让我们把最低程度的框架给搭好吧。

首先在GV.h里面为了保存面板的图片，我们先定义变量。

--在GV.h中如下追加-

GLOBAL int img\_board[40];

顺便为下一节做准备，我们先准备一个初始化函数吧。

接下来，我们定义表示出角色可以自由运动领域的范围。

FIELD\_MAX\_X,　FILED\_MAX\_Y是各种角色可以自由移动的领域的最大尺寸。

FIELD\_X, FILED\_Y是这一领域的左上角坐标。

下述的数值描述了这样一个角色可运动的区域：从（32,16）坐标开始，横向384纵向448像素的区域。

--define.h变更--

#include "struct.h"

#define FIELD\_MAX\_X 384

#define FIELD\_MAX\_Y 448

#define FIELD\_X 32

#define FIELD\_Y 16

我们使用calc\_ch来计算显示角色的图像。

通过递增计数器，以24次为一个周期。在此期间有四种类型的图像轮流显示。

如果您看了数据文件中的角色图片的话，您就会明白了，角色是四种图片一个动作，因此我们使用下面的式子来循环计算图片的编号。

--char.cpp变更--

#include "../include/GV.h"

void calc\_ch(){

ch.cnt++;

ch.img=(ch.cnt%24)/6;

}

声明在主循环中调用的函数的时候，也要写在function.h中。

--向function.h进行以下追加-

GLOBAL void calc\_ch();

在load.cpp中读入面板的图像数据。

至于为何图片用于编号的数字并不是连续的，这并没有什么特别的意义，不过这样编号的话后面的更换图片会方便一点。其实从0开始连续也完全没有问题。

--load.cpp变更-

#include "../include/GV.h"

void load(){

img\_board[10] = LoadGraph("../dat/img/board/10.png");

img\_board[11] = LoadGraph("../dat/img/board/11.png");

img\_board[12] = LoadGraph("../dat/img/board/12.png");

img\_board[20] = LoadGraph("../dat/img/board/20.png");

LoadDivGraph( "../dat/img/char/0.png" , 12 , 4 , 3 , 73 , 73 , img\_ch[0] ) ;

}

设定角色的初始坐标。

--ini.cpp变更--

#include "../include/GV.h"

//最开始的初始化

void first\_ini(){

ch.x=FIELD\_MAX\_X/2;

ch.y=FIELD\_MAX\_Y\*3/4;

}

为了描绘面板我们使用graph\_board函数。

--graph.cpp变更--

#include "../include/GV.h"

void graph\_ch(){

DrawRotaGraphF(ch.x+FIELD\_X,ch.y+FIELD\_Y,1.0f,0.0f,img\_ch[0][ch.img],TRUE);

}

void graph\_board(){

DrawGraph( 0, 0,img\_board[10],FALSE);

DrawGraph( 0, 16,img\_board[11],FALSE);

DrawGraph( 0,464,img\_board[12],FALSE);

DrawGraph(416, 0,img\_board[20],FALSE);

}

void graph\_main(){

graph\_ch();

graph\_board();

}

然后仅是在main.cpp的主循环中的case 0中加入“first\_ini();”。

--main.cpp变更--

#define GLOBAL\_INSTANCE

#include "../include/GV.h"

//循环必须处理的三大过程

int ProcessLoop(){

if(ProcessMessage()!=0)return -1; //过程处理如果发生错误返回-1

if(ClearDrawScreen()!=0)return -1; //如果画面清空错误返回-1

GetHitKeyStateAll\_2();//进行当前按键处理

return 0;

}

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nCmdShow){

ChangeWindowMode(TRUE);//窗口模式

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初始化和里画面化

while(ProcessLoop()==0){//主循环

switch(func\_state){

case 0:

load(); //载入数据

first\_ini();//最开始的初始化

func\_state=100;

break;

case 100:

calc\_ch();

graph\_main();//绘制main

break;

default:

printfDx("错误的func\_state\n");

break;

}

if(CheckStateKey(KEY\_INPUT\_ESCAPE)==1)break;//按下ESC键则跳出循环

ScreenFlip();//里外画面翻转

}

DxLib\_End();//DX Library终结处理

return 0;

}

运行结果

## 第八章 对游戏的输入进行反馈吧

按键配置对STG而言是非常重要的部分。

如果如果我们在最开始不设定的话，那么随着后面源程序的整个地改变，事情也会变得麻烦，因此我们在最开始就设定好吧。

首先我们要获得按键的输入状态。（这一点我们在之前的小节中已经完成了）

其次要取得手柄的输入状态。由于我们希望无论是键盘输入还是手柄输入我们都有相应的回馈，因此我们将键盘的输入状态放到在config中设定好的手柄信息中，从而两方面我们都能给出回馈。

现在我们使用GetJoypadInputState函数，如果不清楚怎么用的，请到DX Libaray参考文档处确认一下。

那么，我们定义手柄和config会使用的结构体，在结构体中我们把PAD\_MAX设定到16，定义如下：

--在struct.h 如下追加--

//和手柄相关的参数

typedef struct{

int key[PAD\_MAX];

}pad\_t;

//和config相关的文件

typedef struct{

int left,up,right,down,shot,bom,slow,start,change;

}configpad\_t;

--define.h变更--

//区域的尺寸

#define FIELD\_MAX\_X 384

#define FIELD\_MAX\_Y 448

//区域的左上角坐标

#define FIELD\_X 32

#define FIELD\_Y 16

//手柄按键的最大数量

#define PAD\_MAX 16

#include "struct.h"

我们先预置在手柄上使用的按键数目最大为16个，首先我们确定其中的九个按键分别表示“左、上、右、下、发射、Boom、低速运动、开始、变身”。

对于不同的键位使用0~15中的数字去对应，这样一来就可以按照自己喜好的设定来使用手柄上的按键了。

至于手柄上的哪个编号和哪个按钮对应，实际上这需要自己去确认。

就我这边确认的结果，一般使用下述的编号，我们在初始化函数中带入这些值。

--ini.cpp变更--

#include "../include/GV.h"

//最开始的初始化

void first\_ini(){

ch.x=FIELD\_MAX\_X/2;

ch.y=FIELD\_MAX\_Y\*3/4;

configpad.down=0;

configpad.left=1;

configpad.right=2;

configpad.up=3;

configpad.bom=4;

configpad.shot=5;

configpad.slow=11;

configpad.start=13;

configpad.change=6;

}

--向GV.h 进行以下追加--

GLOBAL configpad\_t configpad;

使用GetHitPadStateAll函数可以获得当前手柄的输入状态，与键盘输入状态进行比较，取用按键时间较长的一方，然后带入手柄的输入变量中。对上面的两个函数并没有做任何更改。

在这个时候将使用的参数1和参数2中较大的一方放到参数1中的函数是这样的：

void input\_pad\_or\_key(int \*p, int k)

如果您对在这个函数中使用的三元表达式不熟悉的话，请自行搜索相关资料。

如果向CheackStatePad函数传入在初始化函数中设定好了的手柄编号，那么它会返回其输入状态。

它和已经实现好了的CheckStateKey函数是一个功能。

--key.cpp变更--

#include "../include/GV.h"

unsigned int stateKey[256];

int GetHitKeyStateAll\_2(){

char GetHitKeyStateAll\_Key[256];

GetHitKeyStateAll( GetHitKeyStateAll\_Key );

for(int i=0;i<256;i++){

if(GetHitKeyStateAll\_Key[i]==1) stateKey[i]++;

else stateKey[i]=0;

}

return 0;

}

int CheckStateKey(unsigned char Handle){

return stateKey[Handle];

}

//保存手柄输入状态的变量

pad\_t pad;

//把参数1和参数2中较大的一方传给参数1

void input\_pad\_or\_key(int \*p, int k){

\*p = \*p>k ? \*p : k;

}

//同时检查手柄和键盘输入的函数

void GetHitPadStateAll(){

int i,PadInput,mul=1;

PadInput = GetJoypadInputState( DX\_INPUT\_PAD1 );//获得手柄的输入状态

for(i=0;i<16;i++){

if(PadInput & mul) pad.key[i]++;

else pad.key[i]=0;

mul\*=2;

}

input\_pad\_or\_key(&pad.key[configpad.left] ,CheckStateKey(KEY\_INPUT\_LEFT ));

input\_pad\_or\_key(&pad.key[configpad.up] ,CheckStateKey(KEY\_INPUT\_UP ));

input\_pad\_or\_key(&pad.key[configpad.right] ,CheckStateKey(KEY\_INPUT\_RIGHT ));

input\_pad\_or\_key(&pad.key[configpad.down] ,CheckStateKey(KEY\_INPUT\_DOWN ));

input\_pad\_or\_key(&pad.key[configpad.shot] ,CheckStateKey(KEY\_INPUT\_Z ));

input\_pad\_or\_key(&pad.key[configpad.bom] ,CheckStateKey(KEY\_INPUT\_X ));

input\_pad\_or\_key(&pad.key[configpad.slow] ,CheckStateKey(KEY\_INPUT\_LSHIFT ));

input\_pad\_or\_key(&pad.key[configpad.start] ,CheckStateKey(KEY\_INPUT\_ESCAPE ));

input\_pad\_or\_key(&pad.key[configpad.change] ,CheckStateKey(KEY\_INPUT\_LCONTROL));

}

//返回传入的手柄的按键编号对应的输入状态。如果返回-1表示传入错误。

int CheckStatePad(unsigned int Handle){

if(0<=Handle && Handle<PAD\_MAX){

return pad.key[Handle];

}

else{

printfDx("向CheckStatePad传入了错误的键值\n");

return -1;

}

}

为了向input\_pad\_or\_key函数传入各种参数写了一大堆东西，事实上如果使用数组和指针的话会省事许多，如果您尚有余力，那么可以尝试把这段代码简短一下。

--char.cpp变更--

#include "../include/GV.h"

void calc\_ch(){

ch.cnt++;

ch.img=(ch.cnt%24)/6;

if(CheckStatePad(configpad.left)>0)//如果左键按下的情况

ch.x-=3;//坐标向左增加

if(CheckStatePad(configpad.right)>0)//如果右键按下的情况

ch.x+=3;//坐标向右增加

}

CheckStatePad函数的使用如上。因为设定好了的按键，比如左键的编号已经放进了configpad.left中，所以就直接把这个编号传给了CheckStatePad函数。因为会返回按键按下时间长短的计数器的值，所以如果这个值不是0的话，那说明就没有按下。

接下来我们把这次增加的函数向头文件里追加进去吧。

--向function.h 进行如下追加--

GLOBAL void GetHitPadStateAll();

GLOBAL int CheckStatePad(unsigned int Handle);

在本次的main文件中发生变化的部分在下面用红字标出。

--main.cpp变更--

#define GLOBAL\_INSTANCE

#include "../include/GV.h"

//循环必须处理的三大过程

int ProcessLoop(){

if(ProcessMessage()!=0)return -1; //过程处理如果发生错误返回-1

if(ClearDrawScreen()!=0)return -1; //如果画面清空错误返回-1

GetHitKeyStateAll\_2();//进行当前按键处理

GetHitPadStateAll(); //处理当前的手柄输入

return 0;

}

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nCmdShow){

ChangeWindowMode(TRUE);//窗口模式

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初始化和里表面化

while(ProcessLoop()==0){//主循环

switch(func\_state){

case 0:

load(); //载入数据

first\_ini();//最开始的初始化

func\_state=100;

break;

case 100:

calc\_ch(); //角色计算

graph\_main();//绘制main

break;

default:

printfDx("错误的func\_state\n");

break;

}

if(CheckStateKey(KEY\_INPUT\_ESCAPE)==1)break;//按下ESC键则跳出循环

ScreenFlip();//里外画面翻转

}

DxLib\_End();//DX Library终止处理

return 0;

}

运行结果

接入游戏手柄然后左右按一下，然后在键盘上用十字键左右按一下，如果角色开始动画那样地左右移动的话那就成功了。

## 第九章 试着控制一下角色的移动吧

现在我们试着让我们的角色可以在画面中可移动的区域内上下左右地移动。

由于斜向前进的时候不给给以1/√2的速度的话，一般而言合速度是不会为1的（译者注：这是一个简单的运动学问题，将斜向的速度视为合速度矢量并投影到xy方向上，若斜向速度要保持为1，那么xy速度都应该为1/√2的速度），因此斜向移动的时候，有必要在水平和竖直方向上分别对速度进行变化。

--char.cpp变更--

#include "../include/GV.h"

void calc\_ch(){

ch.cnt++;

ch.img=(ch.cnt%24)/6;

}

void ch\_move(){//控制角色的移动

int i,sayu\_flag=0,joge\_flag=0;

double x,y,mx,my,naname=1;

double move\_x[4]={-4.0,4.0,0,0},move\_y[4]={0,0,4.0,-4.0};//{左,右,下,上}的速度

int inputpad[4];

inputpad[0]=CheckStatePad(configpad.left); inputpad[1]=CheckStatePad(configpad.right);

inputpad[2]=CheckStatePad(configpad.down); inputpad[3]=CheckStatePad(configpad.up);

if(CheckStatePad(configpad.left)>0)//左键按下的情况

ch.img+=4\*2;//图像左移

else if(CheckStatePad(configpad.right)>0)//右键按下的情况

ch.img+=4\*1;//图像右移

for(i=0;i<2;i++)//左右部分

if(inputpad[i]>0)//如果左右方向上有输入

sayu\_flag=1;//设置左右移动的标志

for(i=2;i<4;i++)//上下部分

if(inputpad[i]>0)//如果上下方向有输入

joge\_flag=1;//设置上下移动的标志

if(sayu\_flag==1 && joge\_flag==1)//如果左右、上下方向都有输入说明斜向移动

naname=sqrt(2.0);//将移动速度变为1/√2

for(int i=0;i<4;i++){//4方向分量循环

if(inputpad[i]>0){//i方向的键盘或者手柄有输入

x=ch.x , y=ch.y;//暂时存入当前的坐标

mx=move\_x[i]; my=move\_y[i];//将移动分量带入mx和my if(CheckStatePad(configpad.slow)>0){//如果低速运动

mx=move\_x[i]/3; my=move\_y[i]/3;//将移动速度变为1/3

}

x+=mx/naname , y+=my/naname;//为当前坐标加上移动分量

if(!(x<10 || x>FIELD\_MAX\_X-10 || y<5 || y>FIELD\_MAX\_Y-5)){//如果计算结果在可移动范围内

ch.x=x , ch.y=y;//进行实际的移动

}

}

}

}

因为使用了sqrt函数，所以需要包含math.h头文件。

然后把ch\_move函数添加到function.h文件中。

--在GV.h进行以下追加--

#include "math.h"

--在fnction.h进行以下追加-

GLOBAL void ch\_move();

主函数部分只添加了红字部分。

--main.cpp变更

#define GLOBAL\_INSTANCE

#include "../include/GV.h"

//循环必须处理的三大过程

int ProcessLoop(){

if(ProcessMessage()!=0)return -1; //过程处理如果发生错误返回-1

if(ClearDrawScreen()!=0)return -1; //如果画面清空错误返回-1

GetHitKeyStateAll\_2();//进行当前按键处理

GetHitPadStateAll(); //处理当前的手柄输入

return 0;

}

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nCmdShow){

ChangeWindowMode(TRUE);//窗口模式

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初始化和里表面化

while(ProcessLoop()==0){//主循环

switch(func\_state){

case 0:

load(); //载入数据

first\_ini();//最开始的初始化

func\_state=100;

break;

case 100:

calc\_ch(); //角色计算

ch\_move(); //控制角色的移动

graph\_main();//描绘main

break;

default:

printfDx("错误的c\_state\n");

break;

}

if(CheckStateKey(KEY\_INPUT\_ESCAPE)==1)break;//按下ESC键则跳出循环

ScreenFlip();//里外画面翻转

}

DxLib\_End();//DX Library终结

return 0;

}

运行结果

接上游戏手柄然后左右上下按键测试，同时在键盘上也左右上下按键测试，如果角色开始动画那样地移动的话那就成功了。

## 第十章 来显示敌人吧

既然是要做的是STG，那真想快点开始制作弹幕啊。

但是，在构造弹幕数据之前，我们必须先构造好敌人的数据，所以我们先来搞好敌人的构造吧。

首先我们先写好敌人数据的结构体。

--在struct.h中进行以下追加--

//和敌人相关的结构体

typedef struct{

//flag、计数器、移动模式、方向、敌人的种类、HP最大值、掉落道具、图像

int flag,cnt,pattern,muki,knd,hp,hp\_max,item\_n[6],img;

//坐标、速度x分量、速度y分量、速度、角度

double x,y,vx,vy,sp,ang;

//弹幕开始时间、弹幕的种类、子弹的种类、颜色、状态、待机时间、停止时间

int bltime,blknd,blknd2,col,state,wtime,wait;

}enemy\_t;

这是什么情况！

虽然一大堆变量杂七杂八地混在一起，不过没有必要全部记住。

flag就是flag、cnt就是计数器、pattern就是敌人移动的模式，现在我们用自我表意的名字命名了它们，当我们想要知道这个变量是用来做什么的时候，就看着它们的名字就了解了。

因为敌人肯定拥有许多不同的信息，因此虽然变量一下子变得很多很多，但是这并不复杂。

事先定义好游戏中用于表示计数器的stage\_count，还有用于保存敌人图像句柄的img\_enemy。

在define里面我们把画面上同时显示的最大敌人数使用ENEMY\_MAX定义为30.

--在GV.h中进行以下追加--

GLOBAL int stage\_count;

GLOBAL int img\_enemy[3][9];//敌人的图像每种9张　X3种敌人

GLOBAL enemy\_t enemy[ENEMY\_MAX];

--在define.h中进行以下追加--

//同时表示的最大敌人数

#define ENEMY\_MAX 30

--在load.cpp中进行以下追加--

LoadDivGraph( "../dat/img/enemy/0.png" , 9 , 3 , 3 , 32 , 32 , img\_enemy[0] ) ;

初始化函数分为两种类型：1、游戏启动之后只进行一次的初始化；2、每次新游戏开始时进行的初始化。

根据下面的例子来说，config的出事设定在新游戏开始的时候没有必要进行。

--ini.cpp变更--

#include "../include/GV.h"

//最开始的初始化

void first\_ini(){

configpad.down=0;

configpad.left=1;

configpad.right=2;

configpad.up=3;

configpad.bom=4;

configpad.shot=5;

configpad.slow=11;

configpad.start=13;

configpad.change=6;

}

//新游戏的初始化

void ini(){

stage\_count=1;

memset(&ch,0,sizeof(ch\_t));//自机数据的初始化

ch.x=FIELD\_MAX\_X/2;

ch.y=FIELD\_MAX\_Y\*3/4;

memset(enemy,0,sizeof(enemy\_t)\*ENEMY\_MAX);//敌人数据的初始化

}

作为敌人数据处理中相当重要的流程，当游戏计数器在到达某一点的时候，开始登陆敌人的数据（enemy\_enter）。

当敌人的数据被登录之后，就将处理交给控制函数，这个函数控制了敌人的行动，而这些敌人按照登录了的移动模式移动。

当敌人跑到画面外面的时候，就把登陆flag取消掉（enemy\_act）。

整个流程大概就是这样。

在下面的例子中，尝试着让游戏计数器在100的时候登录编号为0的敌人，使其先往下移动然后又往上移动。

--enemy.cpp变更--

#include "../include/GV.h"

//使用敌人的移动模式0来控制敌人的移动

void enemy\_pattern0(int i){

if(enemy[i].cnt<60){

enemy[i].y+=2.0;

}

if(enemy[i].cnt>60+240){

enemy[i].y-=2.0;

}

}

//登陆敌人的数据

void enemy\_enter(){

if(stage\_count==100){//在新游戏开始后计数器变为100的时候登录

enemy[0].cnt =0;

enemy[0].muki =1;

enemy[0].flag =1;

enemy[0].bltime =150;

enemy[0].hp =1000;

enemy[0].hp\_max =enemy[0].hp;

enemy[0].pattern=0;

enemy[0].x =FIELD\_MAX\_X/2;

enemy[0].y =-20;

}

}

//控制敌人的行动

void enemy\_act(){

int i;

for(i=0;i<ENEMY\_MAX;i++){

if(enemy[i].flag==1){//如果敌人的flag为有效

enemy\_pattern0(i);

enemy[i].cnt++;

enemy[i].img=enemy[i].muki\*3+(enemy[i].cnt%18)/6;

//如果敌人跑到外面去了就把他们销毁

if(enemy[i].x<-50 || FIELD\_MAX\_X+50<enemy[i].x || enemy[i].y<-50 || FIELD\_MAX\_Y+50<enemy[i].y)

enemy[i].flag=0;

}

}

}

//敌人处理main

void enemy\_main(){

enemy\_enter();

enemy\_act();

}

在绘制处理中只需要绘制增加了的敌人数据。

搜索目前已经登录了的敌人数据，然后对其进行绘制。

--graph.cpp变更--

#include "../include/GV.h"

void graph\_enemy(){

int i;

for(i=0;i<ENEMY\_MAX;i++){

if(enemy[i].flag==1){

DrawRotaGraphF(enemy[i].x+FIELD\_X,enemy[i].y+FIELD\_Y,1.0f,0.0f,img\_enemy[0][enemy[i].img],TRUE);

}

}

}

void graph\_ch(){

DrawRotaGraphF(ch.x+FIELD\_X,ch.y+FIELD\_Y,1.0f,0.0f,img\_ch[0][ch.img],TRUE);

}

void graph\_board(){

DrawGraph( 0, 0,img\_board[10],FALSE);

DrawGraph( 0, 16,img\_board[11],FALSE);

DrawGraph( 0,464,img\_board[12],FALSE);

DrawGraph(416, 0,img\_board[20],FALSE);

}

void graph\_main(){

graph\_enemy();

graph\_ch();

graph\_board();

}

main函数中变更了红字部分。

--main.cpp变更--

#define GLOBAL\_INSTANCE

#include "../include/GV.h"

//循环必须进行的三大处理

int ProcessLoop(){

if(ProcessMessage()!=0)return -1;//如果过程处理错误返回-1

if(ClearDrawScreen()!=0)return -1;//如果画面清空处理错误返回-1

GetHitKeyStateAll\_2();//执行当前的输入处理

GetHitPadStateAll(); //执行当前手柄输入处理

return 0;

}

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nCmdShow){

ChangeWindowMode(TRUE);//窗口模式

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初始化和里表面化

while(ProcessLoop()==0){//主循环

switch(func\_state){

case 0://只在游戏启动时处理

load(); //载入数据

first\_ini();//最开始的初始化

func\_state=99;

break;

case 99://STG开始前进行的初始化

ini();

func\_state=100;

break;

case 100://通常处理

calc\_ch(); //角色计算

ch\_move(); //控制角色移动

enemy\_main();//敌人处理main

graph\_main();//绘制main

stage\_count++;

break;

default:

printfDx("错误的func\_state\n");

break;

}

if(CheckStateKey(KEY\_INPUT\_ESCAPE)==1)break;//如果按下ESC键跳出循环

ScreenFlip();//里外画面翻转

}

DxLib\_End();//DX Library终止处理

return 0;

}

现在将本次追加的函数进行登录。

--在function.h 中进行以下处理--

GLOBAL void enemy\_main();

GLOBAL void ini();

运行结果

如果等了一阵子敌人从上面降落下来，过了一会儿又往上回去的话，那就说明成功了。

## 第十一章 使用Excel来设置敌人出现的数据吧

在道中会出来很多敌人。

同时敌人拥有的信息有很多。至少也得有下面这些吧：

--在struct.h进行以下追加-

typedef struct{

//计数器、移动模式、敌人的种类

int cnt,pattern,knd;

//初始化坐标和移动速度

double x,y,sp;

//弹幕开始时间、弹幕种类、颜色、体力、子弹的种类、停止时间、物品（6个种类）

int bltime,blknd,col,hp,blknd2,wait,item\_n[6];

}enemy\_order\_t;

这是和敌人出现信息有关的结构体。

何时、何处、什么样子的敌人、射过来什么样子的子弹、要停止多久、是否回去……之类的信息。

我们必须要指定许多东西。但是，

在计数器达到112时x坐标在这里、y坐标在那里、移动模式是这样、计数器达到187的时候又要这样又要那样……

要是将这些数据全部手动输入然后代入到数据中的话会很麻烦的。

因此我们试试使用Excel来制作敌人的出现的数据吧。

首先，从这里下载已经用Excel写好了的出现信息数据。

Excel数据在这里下载。

在dat文件夹里面增加一个“csv”的文件夹，然后把Excel文件放进去。内部应该是这样子的。

如果没有Excel的话，文本处理器也能够打开，如下：

首先最左边的是敌人出现的时刻。因为这是60fps的游戏，60=1一秒。

也就是说，这表示第一个敌人，在游戏开始之后1.6秒左右出现。

然后置10个计数，也就是说每隔1/6秒就出现一个敌人。

敌人的信息除了x坐标以外其它都是一样的。使用Excel能够简单地作成有同样变化的值，非常方便。

接下来，使用DX Library将文件读入的时候，使用Dx Library的函数（为了使用archive）。

如果不知道DX Library的文件读入函数的使用方法，请到其参考主页上学习。

使用起来基本上和C语言的标准函数fopen（←只有一点区别）、fgetc之类的函数的方法相同。

其次，我认为就算是csv文件，即使没有使用过的经验，将这个文件用文本编辑器打开应该很快就能够明白了。由于这只是简单地将单元用逗号隔开用于隔开的数据，因此只要用逗号隔开然后仔细阅读的话，一般而言就能够掌握和阅读一般文本数据一样的要领了。那么，现在我们来写这个读入函数。

下面我们把从Excel数据读入用于存储敌人出现信息的变量enemy\_order中，虽然用了一个代入函数，但是这没有有必要理解它，只要知道这个函数是用来把数据带入这个变量的就行了。

--在load.cpp中进行以下追加 --

//将敌人出现的信息从Excel读入并保存的函数

void load\_story(){

int n,num,i,fp;

char fname[32]={"../dat/csv/storyH0.csv"};

int input[64];

char inputc[64];

fp = FileRead\_open(fname);//读入文件

if(fp == NULL){

printfDx("read error\n");

return;

}

for(i=0;i<2;i++)//丢掉最开始的两行

while(FileRead\_getc(fp)!='\n');

n=0 , num=0;

while(1){

for(i=0;i<64;i++){

inputc[i]=input[i]=FileRead\_getc(fp);//读入1个字符

if(inputc[i]=='/'){//如果是斜杠

while(FileRead\_getc(fp)!='\n');//循环一直到换行

i=-1;//重置计数器

continue;

}

if(input[i]==',' || input[i]=='\n'){//如果是逗号或者换行

inputc[i]='\0';//将到此的所有文字作为字符串

break;

}

if(input[i]==EOF){//如果到了文件尾

goto EXFILE;//终止

}

}

switch(num){

case 0: enemy\_order[n].cnt =atoi(inputc);break;

case 1: enemy\_order[n].pattern =atoi(inputc);break;

case 2: enemy\_order[n].knd =atoi(inputc);break;

case 3: enemy\_order[n].x =atof(inputc);break;

case 4: enemy\_order[n].y =atof(inputc);break;

case 5: enemy\_order[n].sp =atof(inputc);break;

case 6: enemy\_order[n].bltime =atoi(inputc);break;

case 7: enemy\_order[n].blknd =atoi(inputc);break;

case 8: enemy\_order[n].col =atoi(inputc);break;

case 9: enemy\_order[n].hp =atoi(inputc);break;

case 10:enemy\_order[n].blknd2 =atoi(inputc);break;

case 11:enemy\_order[n].wait =atoi(inputc);break;

case 12:enemy\_order[n].item\_n[0]=atoi(inputc);break;

case 13:enemy\_order[n].item\_n[1]=atoi(inputc);break;

case 14:enemy\_order[n].item\_n[2]=atoi(inputc);break;

case 15:enemy\_order[n].item\_n[3]=atoi(inputc);break;

case 16:enemy\_order[n].item\_n[4]=atoi(inputc);break;

case 17:enemy\_order[n].item\_n[5]=atoi(inputc);break;

}

num++;

if(num==18){

num=0;

n++;

}

}

EXFILE:

FileRead\_close(fp);

}

想完全理解的话太阳都要落山了，那么我们就先把它放到这里继续往下走吧。

在进入具体的处理之前，我先说明几个细微的增加点。

在ini.cpp中，在初始化函数中初始化现在我们增加了的数据。

在define.h中，设定了允许登录的最大数量。

在Gv.h中我们定义了使用到的变量。

--在ini.cpp的ini函数中进行以下追加-

memset(enemy\_order,0,sizeof(enemy\_order\_t)\*ENEMY\_ORDER\_MAX);

--在define.h中进行以下追加--

//敌人出现信息的最大数量

#define ENEMY\_ORDER\_MAX 500

--在GV.h中进行以下追加--

GLOBAL enemy\_order\_t enemy\_order[ENEMY\_ORDER\_MAX];//敌人的出现信息

--在function.h中进行以下追加--

GLOBAL void load\_story();

--main.cpp中的main函数的switch语句中的以下部分变更

case 99://STG开始前的初始化

ini();

load\_story();

func\_state=100;

break;

这样一来，我们就把这次敌人出现信息相关的数据放到enemy\_order里面了。

然后我们把保存了的数据在下面进行登录，而这实际上就是在游戏内登录敌人。

使用enemy\_enter进行敌人数据的登录。

因为enemy\_order[n].cnt中已经存入了敌人应该合适出现的数据，那么便在与当前的游戏计数器一致的时候，进行敌人数据的登录。

进行敌人数据登录的时候，为了找到哪个编号是空的，我们实现一个叫做enemy\_num\_search的函数。

放入敌人数据的enemy数组下标是0~ENEMY\_MAX-1，这个函数将告诉我们哪个编号可以使用。如果全部都不是空的话那么返回-1，这个时候就不再进行登录。

另外，前文已经说过enemy[n].wait表示停止时间，像下面的红字那样使用。

这样一来就可以控制敌人的停止时间了。

再者，之所以敌人移动的计算，既有利用ang和sp组合的计算，也有利用vx、vy组合的计算，这是因为根据情况的不同，用途会不同。

当然，这次在enemy\_parttern中使用到的vy=2也可以和ang=3.14/2、sp=2进行置换，前者更容易理解就是了。

--enemy.cpp变更--

#include "../include/GV.h"

//使用敌人的移动模式0进行移动控制

void enemy\_pattern0(int i){

int t=enemy[i].cnt;

if(t==0)

enemy[i].vy=2;//下降

if(t==60)

enemy[i].vy=0;//静止

if(t==60+enemy[i].wait)//在登录的时间区间内停止

enemy[i].vy=-2;//上升

}

//检测空闲的敌人编号

int enemy\_num\_search(){

for(int i=0;i<ENEMY\_MAX;i++){//搜寻没有立上flag的enemy

if(enemy[i].flag==0){

return i;//返回可以使用的编号

}

}

return -1;//如果全部都使用了返回错误

}

//登录敌人的信息

void enemy\_enter(){//登录、控制敌人行动的函数

int i,j,t;

for(t=0;t<ENEMY\_ORDER\_MAX;t++){

if(enemy\_order[t].cnt==stage\_count){//如果当前计数是是顺序的计数的瞬间

if((i=enemy\_num\_search())!=-1){

enemy[i].flag =1;//flag

enemy[i].cnt =0;//计数器

enemy[i].pattern=enemy\_order[t].pattern;//移动模式

enemy[i].muki =1;//方向

enemy[i].knd =enemy\_order[t].knd;//敌人的种类

enemy[i].x =enemy\_order[t].x;//坐标

enemy[i].y =enemy\_order[t].y;

enemy[i].sp =enemy\_order[t].sp;//速度

enemy[i].bltime =enemy\_order[t].bltime;//子弹的发射时间

enemy[i].blknd =enemy\_order[t].blknd;//弹幕的种类

enemy[i].blknd2 =enemy\_order[t].blknd2;//子弹的种类

enemy[i].col =enemy\_order[t].col;//颜色

enemy[i].wait =enemy\_order[t].wait;//停止时间

enemy[i].hp =enemy\_order[t].hp;//体力

enemy[i].hp\_max =enemy[i].hp;//体力最大值

enemy[i].vx =0;//水平分量的速度

enemy[i].vy =0;//竖直分量的速度

enemy[i].ang =0;//角度

for(j=0;j<6;j++)

enemy[i].item\_n[j]=enemy\_order[t].item\_n[j];//掉落的道具

}

}

}

}

//控制敌人的行动

void enemy\_act(){

int i;

for(i=0;i<ENEMY\_MAX;i++){

if(enemy[i].flag==1){//如果敌人的flag为有效

enemy\_pattern0(i);

enemy[i].x+=cos(enemy[i].ang)\*enemy[i].sp;

enemy[i].y+=sin(enemy[i].ang)\*enemy[i].sp;

enemy[i].x+=enemy[i].vx;

enemy[i].y+=enemy[i].vy;

enemy[i].cnt++;

enemy[i].img=enemy[i].muki\*3+(enemy[i].cnt%18)/6;

//如果敌人跑到画面外面了就销毁

if(enemy[i].x<-20 || FIELD\_MAX\_X+20<enemy[i].x || enemy[i].y<-20 || FIELD\_MAX\_Y+20<enemy[i].y)

enemy[i].flag=0;

}

}

}

//敌人处理main

void enemy\_main(){

enemy\_enter();

enemy\_act();

}

运行结果

敌人连续地落下，等了一会儿又往回跑的话，就说明成功了。

## 第十二章 设置许多敌人的运动模式吧

前面我们已经写好了控制一个敌人移动的函数了。现在我们做很多。

我试着做了0~10共11种控制敌人的行动模式的函数。

另外，这次也做了一个叫rang的函数。这个函数的功能是传入n，然后返回一个从-n~n的double类型的随机数。

下面的代码稍微有点长，不过就是把简单的函数写到一起而已。

--enemy\_act\_pattern.cpp变更--

#include "../include/GV.h"

//返回-ang～ang的随机角度

double rang(double ang){

return ( -ang + ang\*2 \* GetRand(10000)/10000.0 );

}

//移动模式0

//向下移动后停止然后往上移动

void enemy\_pattern0(int i){

int t=enemy[i].cnt;

if(t==0)

enemy[i].vy=3;//向下移动

if(t==40)

enemy[i].vy=0;//停止

if(t==40+enemy[i].wait)//在登录的时间区间内停止

enemy[i].vy=-3;//向上移动

}

//移动模式1

//向下移动然后停止然后向左下移动

void enemy\_pattern1(int i){

int t=enemy[i].cnt;

if(t==0)

enemy[i].vy=3;//向下移动

if(t==40)

enemy[i].vy=0;//停止

if(t==40+enemy[i].wait){//在登录的时间区间内停止

enemy[i].vx=-1;//向左

enemy[i].vy=2;//向下移动

enemy[i].muki=0;//设置方向为左

}

}

//移动模式2

//向下移动然后停止然后向右下移动

void enemy\_pattern2(int i){

int t=enemy[i].cnt;

if(t==0)

enemy[i].vy=3;//向下移动

if(t==40)

enemy[i].vy=0;//停止

if(t==40+enemy[i].wait){// 在登录的时间区间内停止

enemy[i].vx=1;//向右

enemy[i].vy=2;//向下移动

enemy[i].muki=2;//设置方向为右

}

}

//移动模式3

//迅速地下降然后向左移动

void enemy\_pattern3(int i){

int t=enemy[i].cnt;

if(t==0)

enemy[i].vy=5;//向下移动

if(t==30){//途中方向向左

enemy[i].muki=0;

}

if(t<100){

enemy[i].vx-=5/100.0;//左方向加速

enemy[i].vy-=5/100.0;//减速

}

}

//移动模式4

//迅速地下降然后向右移动

void enemy\_pattern4(int i){

int t=enemy[i].cnt;

if(t==0)

enemy[i].vy=5;//向下移动

if(t==30){//途中方向向右

enemy[i].muki=2;

}

if(t<100){

enemy[i].vx+=5/100.0;//右方向加速

enemy[i].vy-=5/100.0;//減速

}

}

//移动模式5

//斜着向左下移动

void enemy\_pattern5(int i){

int t=enemy[i].cnt;

if(t==0){

enemy[i].vx-=1;

enemy[i].vy=2;

enemy[i].muki=0;

}

}

//移动模式6

//斜着向右下移动

void enemy\_pattern6(int i){

int t=enemy[i].cnt;

if(t==0){

enemy[i].vx+=1;

enemy[i].vy=2;

enemy[i].muki=2;

}

}

//移动模式7

//停止然后就那样向左上移动

void enemy\_pattern7(int i){

int t=enemy[i].cnt;

if(t==enemy[i].wait){//在登录的时间区间内停止

enemy[i].vx=-0.7;//向左

enemy[i].vy=-0.3;//向上移动

enemy[i].muki=0;//设置方向为左

}

}

//移动模式8

//停止然后就那样向右上移动

void enemy\_pattern8(int i){

int t=enemy[i].cnt;

if(t==enemy[i].wait){//在登录的时间区间内停止

enemy[i].vx=+0.7;//向右

enemy[i].vy=-0.3;//向上移动

enemy[i].muki=2;//设置方向为右

}

}

//移动模式9

//停止然后就那样向上移动

void enemy\_pattern9(int i){

int t=enemy[i].cnt;

if(t==enemy[i].wait)//在登录的时间区间内停止

enemy[i].vy=-1;//向上移动

}

//移动模式10

//下降然后转着圈往上移动

void enemy\_pattern10(int i){

int t=enemy[i].cnt;

if(t==0) enemy[i].vy=4;//向下移动

if(t==40)enemy[i].vy=0;//停止

if(t>=40){

if(t%60==0){

int r=cos(enemy[i].ang)< 0 ? 0 : 1;

enemy[i].sp=6+rang(2);

enemy[i].ang=rang(PI/4)+PI\*r;

enemy[i].muki=2-2\*r;

}

enemy[i].sp\*=0.95;

}

if(t>=40+enemy[i].wait){

enemy[i].vy-=0.05;

}

}

总之先看看这些函数是描述了怎样的动作的吧。

把Excel文件修改成下面那样。您可以直接下载也可以自己替换。

/计数器 移动模式 敌人的种类 x坐标 y坐标 速度 发射时间 弹幕种类 子弹的颜色 体力 子弹的种类 待机时间 物品1 2 3 4 5 6

/cnt pattern knd x y sp bltime blknd col hp blknd2 wait item\_n

100 0 0 180 -20 0 150 0 0 100 0 120 0 -1 -1 -1 -1 -1

180 1 0 200 -20 0 150 0 0 100 0 120 0 -1 -1 -1 -1 -1

260 2 0 180 -20 0 150 0 0 100 0 120 0 -1 -1 -1 -1 -1

340 3 0 200 -20 0 150 0 0 100 0 120 0 -1 -1 -1 -1 -1

420 4 0 180 -20 0 150 0 0 100 0 120 0 -1 -1 -1 -1 -1

500 5 0 200 -20 0 150 0 0 100 0 120 0 -1 -1 -1 -1 -1

580 6 0 180 -20 0 150 0 0 100 0 120 0 -1 -1 -1 -1 -1

660 7 0 200 150 0 150 0 0 100 0 120 0 -1 -1 -1 -1 -1

740 8 0 180 150 0 150 0 0 100 0 120 0 -1 -1 -1 -1 -1

820 9 0 200 150 0 150 0 0 100 0 120 0 -1 -1 -1 -1 -1

900 10 0 180 -20 0 150 0 0 100 0 360 0 -1 -1 -1 -1 -1

下载Excel

在这里，由于敌人的移动模式的信息放到了enemy[i].pattern中，我们不妨考虑利用放在移动模式信息里面的0~10的值自动地调用enemy\_parttern0~enemy\_parttern10函数。

这样一来我们要使用函数指针这个东西了。

像(\*enemy\_pattern[ENEMY\_PATTERN\_MAX])(int) = { enemy\_pattern0, enemy\_pattern1, .....

这样带入的话，这个数组里面也就放入了函数的地址了。

接下来只要我们对数组下标指定前面我们放在enemy[i].pattern里面的值之后就可以传递处理过程了。

实际上使用函数指针调用函数的方法入红字所示。

--在enemy.cpp中进行以下追加--

#include "../include/GV.h"

#define ENEMY\_PATTERN\_MAX 11

extern void enemy\_pattern0(int);extern void enemy\_pattern1(int);extern void enemy\_pattern2(int);

extern void enemy\_pattern3(int);extern void enemy\_pattern4(int);extern void enemy\_pattern5(int);

extern void enemy\_pattern6(int);extern void enemy\_pattern7(int);extern void enemy\_pattern8(int);

extern void enemy\_pattern9(int);extern void enemy\_pattern10(int);

void (\*enemy\_pattern[ENEMY\_PATTERN\_MAX])(int) = {

enemy\_pattern0, enemy\_pattern1, enemy\_pattern2, enemy\_pattern3, enemy\_pattern4,

enemy\_pattern5, enemy\_pattern6, enemy\_pattern7, enemy\_pattern8, enemy\_pattern9,

enemy\_pattern10,

};

//检查空的敌人编号

int enemy\_num\_search(){

for(int i=0;i<ENEMY\_MAX;i++){//查询flag为无效的enemy

if(enemy[i].flag==0){

return i;//返回使用可能的编号

}

}

return -1;//如果全部被使用返回错误

}

//登录敌人情报

void enemy\_enter(){//控制、登录敌人行动的函数

int i,j,t;

for(t=0;t<ENEMY\_ORDER\_MAX;t++){

if(enemy\_order[t].cnt==stage\_count){//如果当前计数是是顺序的计数的瞬间

if((i=enemy\_num\_search())!=-1){

enemy[i].flag =1;//flag

enemy[i].cnt =0;//计数器

enemy[i].pattern=enemy\_order[t].pattern;//移动模式

enemy[i].muki =1;//方向

enemy[i].knd =enemy\_order[t].knd;//敌人的种类

enemy[i].x =enemy\_order[t].x;//坐标

enemy[i].y =enemy\_order[t].y;

enemy[i].sp =enemy\_order[t].sp;//速度

enemy[i].bltime =enemy\_order[t].bltime;//子弹的发射时间

enemy[i].blknd =enemy\_order[t].blknd;//弹幕的种类

enemy[i].blknd2 =enemy\_order[t].blknd2;//子弹的种类

enemy[i].col =enemy\_order[t].col;//颜色

enemy[i].wait =enemy\_order[t].wait;//停止时间

enemy[i].hp =enemy\_order[t].hp;//体力

enemy[i].hp\_max =enemy[i].hp;//体力最大值

enemy[i].vx =0;//水平分量的速度

enemy[i].vy =0;//竖直分量的速度

enemy[i].ang =0;//角度

for(j=0;j<6;j++)

enemy[i].item\_n[j]=enemy\_order[t].item\_n[j];//掉落的道具

}

}

}

}

//控制敌人的行动

void enemy\_act(){

int i;

for(i=0;i<ENEMY\_MAX;i++){

if(enemy[i].flag==1){//如果敌人的flag为有效

if(0<=enemy[i].pattern && enemy[i].pattern<ENEMY\_PATTERN\_MAX){

enemy\_pattern[enemy[i].pattern](i);

enemy[i].x+=cos(enemy[i].ang)\*enemy[i].sp;

enemy[i].y+=sin(enemy[i].ang)\*enemy[i].sp;

enemy[i].x+=enemy[i].vx;

enemy[i].y+=enemy[i].vy;

enemy[i].cnt++;

enemy[i].img=enemy[i].muki\*3+(enemy[i].cnt%18)/6;

//敌人跑到画面外面的话销毁

if(enemy[i].x<-20 || FIELD\_MAX\_X+20<enemy[i].x || enemy[i].y<-20 || FIELD\_MAX\_Y+20<enemy[i].y)

enemy[i].flag=0;

}

else

printfDx("enemy[i].patternの%d的值错误。",enemy[i].pattern);

}

}

}

//敌人处理main

void enemy\_main(){

enemy\_enter();

enemy\_act();

}

接下来为了计算角度，我们设定圆周率的定义。

由于圆周率π或者2π都经常使用，乘积的值我们也事先定义。

--在define.h中进行以下追加--

//圆周率

#define PI 3.1415926535898

#define PI2 (PI\*2)

运行结果

如果像运行结果那样，敌人进行各种各样的行动的话那就成功了。

## 第十三章 让敌人射出子弹吧

接下来，既然我们已经把敌人的信息确立了，那么我们就试着让它们射出子弹吧。

由于本节追加的地方有点多，所以请一边比较一起发布的工程的内容，一边往下阅读。

首先，我们在敌人出现之后的计数器和设定好了的弹幕发射的计数器的值相同的时候进行弹幕的登录。

设定在敌人信息中的弹幕的开始时间是bltime。而敌人出现后的计数器变量是cnt。

因此在以下时刻进行登录。

---- 在enemy.cpp中追加了红字部分 ----

void enter\_shot(int i){

int j;

for(j=0;j<SHOT\_MAX;j++){//搜索flag为无效的敌人

if(shot[j].flag==0){//如果有没有使用过的弹幕数据

memset(&shot[j],0,sizeof(shot\_t));//初始化并登录之

shot[j].flag=1;//设置flag为有效

shot[j].knd=enemy[i].blknd;//子弹的种类

shot[j].num=i;//num=是哪个敌人发射过来的

shot[j].cnt=0;

return ;

}

}

}

//控制敌人的行动

void enemy\_act(){

int i;

for(i=0;i<ENEMY\_MAX;i++){

if(enemy[i].flag==1){//如果敌人的flag为有效的话

if(0<=enemy[i].pattern && enemy[i].pattern<ENEMY\_PATTERN\_MAX){

enemy\_pattern[enemy[i].pattern](i);

enemy[i].x+=cos(enemy[i].ang)\*enemy[i].sp;

enemy[i].y+=sin(enemy[i].ang)\*enemy[i].sp;

enemy[i].x+=enemy[i].vx;

enemy[i].y+=enemy[i].vy;

enemy[i].cnt++;

enemy[i].img=enemy[i].muki\*3+(enemy[i].cnt%18)/6;

//如果敌人跑到画面外面的话销毁之

if(enemy[i].x<-20 || FIELD\_MAX\_X+20<enemy[i].x || enemy[i].y<-20 || FIELD\_MAX\_Y+20<enemy[i].y)

enemy[i].flag=0;

if(enemy[i].bltime==enemy[i].cnt)

enter\_shot(i);

}

else

printfDx("enemy[i].pattern的%d的值错误。",enemy[i].pattern);

}

}

}

现在让我们看看结构体和变脸的定义，它们描述了弹幕数据是什么样的。

请保证

#include "struct.h"

写在define.h的最后。以后的章节中也是这样。

--- 在define.h 中进行以下追加 ---

//一个敌人所拥有的最大子弹的数量

#define SHOT\_BULLET\_MAX 1000

//画面中一帧所能表示的最大的敌人的弹幕数

#define SHOT\_MAX 30

//射击种类的最大数量

#define SHOT\_KND\_MAX 1

//效果音种类的的最大数量

#define SE\_MAX 100

//敌人的行动模式的最大数量

#define ENEMY\_PATTERN\_MAX 11

--- 在struct.h 中进行以下追加 ---

//和子弹相关的结构体

typedef struct{

//flag、种类、计数器、颜色、状态、保证不消失的最短时间、效果的种类

int flag,knd,cnt,col,state,till,eff;

//坐标、角度、速度、基本角度、瞬间记忆速度

double x,y,angle,spd,base\_angle[1],rem\_spd[1];

}bullet\_t;

//和射击有关的结构体

typedef struct{

//flag、种类、计数器、发射的敌人的编号

int flag,knd,cnt,num;

//基本角度、基本速度

double base\_angle[1],base\_spd[1];

bullet\_t bullet[SHOT\_BULLET\_MAX];

}shot\_t;

--- 在GV.h 中进行以下追加 ---

GLOBAL int img\_bullet[10][10]; //子弹的图像

//音乐文件用的变量部分

GLOBAL int sound\_se[SE\_MAX];

GLOBAL int se\_flag[SE\_MAX]; //SE的flag

GLOBAL shot\_t shot[SHOT\_MAX];//射击信息

--- 在ini.cpp 的 ini() 中进行以下追加 ---

memset(shot,0,sizeof(shot\_t)\*SHOT\_MAX);

bullet\_t是和子弹相关的结构体。它内部同样也包含了很多的变量，不过没有必要记住。

只需要一眼看上去，能够知道这个结构体中有这样的变量就行了。

对于子弹来说，flag、飞行的速度、角度、坐标、状态之类的信息自然是必要的。

其中，til表示至少此区间内不会消失的子弹的计数器数。

如果“只要跑出画面就销毁之”，那么就可能出现做不出来自己想做的弹幕。

因此，为了在till设定的时间内子弹就算跑到画面外面也不销毁之，也就设定了这个变量。

eff表示子弹附加了什么样子的效果，base\_angle表示子弹的角度动态变化的时候保存以作为基准的角度。

shot\_t这个构造体表示将弹幕数据作为一个实体来管理的构造体。

每一个弹幕数据都作为拥有各种变量和拥有SHOT\_BULLET\_MAX（在这里是1000）个子弹的结构体。

由于shot\_t是敌人射出的弹幕的信息，我们只需要定义敌人某个时刻所能表现的最大的子弹的数量就行了。

换言之，弹幕数据的数据库中如果事先准备了30个弹幕数据，那么总的子弹的数量就是30XSHOT\_BULLET\_MAX个蛋了。由于现在保存了相当大的数，因此最好不要让bullet\_t中有太多不相关的变量。

其次，由于弹幕数据、子弹数据都使用了登录的方法，搜索登录的信息然后计算之。

因此，弹幕信息main部分在shot.cpp文件内写成这个样子。

--- 在shot.cpp 中进行以下记录 ---

#include "../include/GV.h"

extern void shot\_bullet\_H000(int);

void (\*shot\_bullet[SHOT\_KND\_MAX])(int) ={

shot\_bullet\_H000,

};

//返回登录了的第n号射击的敌人和自机的之间的夹角

double shotatan2(int n){

return atan2(ch.y-enemy[shot[n].num].y,ch.x-enemy[shot[n].num].x);

}

//搜索空着的子弹

int shot\_search(int n){

int i;

for(i=0;i<SHOT\_BULLET\_MAX;i++){

if(shot[n].bullet[i].flag==0){

return i;

}

}

return -1;

}

void shot\_main(){

int i;

for(i=0;i<SHOT\_MAX;i++){//弹幕数据计算

//如果flag为有效且设定的种类没有错误的情况（防止溢出）

if(shot[i].flag!=0 && 0<=shot[i].knd && shot[i].knd<SHOT\_KND\_MAX){

shot\_bullet[shot[i].knd](i);//调用.knd的弹幕计算函数的函数指针

shot\_calc(i);//计算第i号弹幕

shot[i].cnt++;

}

}

}

--- 在shotH.cpp 中进行以下追加 ---

//只发射一发，向自机直线移动

void shot\_bullet\_H000(int n){

int k;

if(shot[n].cnt==0){

if(shot[n].flag!=2 && (k=shot\_search(n))!=-1){

shot[n].bullet[k].knd =enemy[shot[n].num].blknd2;

shot[n].bullet[k].angle =shotatan2(n);

shot[n].bullet[k].flag =1;

shot[n].bullet[k].x =enemy[shot[n].num].x;

shot[n].bullet[k].y =enemy[shot[n].num].y;

shot[n].bullet[k].col =enemy[shot[n].num].col;

shot[n].bullet[k].cnt =0;

shot[n].bullet[k].spd =3;

se\_flag[0]=1;

}

}

}

---在 main.cpp 中进行红字部分的追加 ---

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nCmdShow){

ChangeWindowMode(TRUE);//窗口模式

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初始化和里表面化

while(ProcessLoop()==0){//主循环

music\_ini();

switch(func\_state){

case 0://只在游戏启动时进行的处理

load(); //载入数据

first\_ini();//最开始的初始化

func\_state=99;

break;

case 99://新的STG开始前的初始化

ini();

load\_story();

func\_state=100;

break;

case 100://通常处理

calc\_ch(); //角色计算

ch\_move(); //角色的移动控制

enemy\_main();//敌人处理main

shot\_main();//射击main

graph\_main();//绘制main

stage\_count++;

break;

default:

printfDx("错误的func\_state\n");

break;

}

music\_play();

if(CheckStateKey(KEY\_INPUT\_ESCAPE)==1)break;//按下ESC键则跳出循环

ScreenFlip();//里外画面翻转

}

DxLib\_End();//DX Library终止处理

return 0;

}

直至弹幕数据的最大数量循环搜索已经登录的弹幕，如果有已经登录的数据，那么交给计算部分处理。

从现在开始我们将会制作种类繁多的弹幕，因此这里我们按照敌人行动模式那样子使用函数指针。

实际的计算部分如下所示。（在之前的函数的基础上进行追加）

---在 shot.cpp 中进行以下追加 ---

void shot\_calc(int n){

int i,max=0;

if(enemy[shot[n].num].flag!=1)//如果敌人被打倒的话

shot[n].flag=2;//将之前登录的射击的flag设置为无效

for(i=0;i<SHOT\_BULLET\_MAX;i++){//计算第n号弹幕数据的子弹

if(shot[n].bullet[i].flag>0){//如果那子弹并没有被登录

shot[n].bullet[i].x+=cos(shot[n].bullet[i].angle)\*shot[n].bullet[i].spd;

shot[n].bullet[i].y+=sin(shot[n].bullet[i].angle)\*shot[n].bullet[i].spd;

shot[n].bullet[i].cnt++;

if(shot[n].bullet[i].x<-50 || shot[n].bullet[i].x>FIELD\_MAX\_X+50 ||

shot[n].bullet[i].y<-50 || shot[n].bullet[i].y>FIELD\_MAX\_Y+50){//如果跑到画面外面的话

if(shot[n].bullet[i].till<shot[n].bullet[i].cnt)//且比最低程度不会销毁的时间还要很长

shot[n].bullet[i].flag=0;//销毁之

}

}

}

//查询当前显示中的子弹的熟练是否至少还有一个

for(i=0;i<SHOT\_BULLET\_MAX;i++)

if(shot[n].bullet[i].flag>0)

return;

if(enemy[shot[n].num].flag!=1){

shot[n].flag=0;//終了

enemy[shot[n].num].flag=0;

}

}

由于从弹幕main函数中将弹幕编号作为n传递出去，因此我们要进行第n号弹幕数据的计算。

由于弹幕数据中有SHOT\_BULLET\_MAX个子弹，我们进一步进行在其中登录的数据的计算。

并且，敌人被打倒了还在发射子弹的话看起来会很奇怪，因此我们必须在敌人被打倒之后取消子弹的登录。因此，敌人一旦被打倒，就flag设定而为2.

然后，如果画面中一个子弹都没有了的话，弹幕就终结了。

接下来，Hard的第0号计算函数在shotH.cpp中写为shot\_bullet\_H000，如下进行计算。

--- 在shotH.cpp 中进行以下追加 ---

void shot\_bullet\_H000(int n){

int k;

if(shot[n].cnt==0){//当计数器为0的时候，弹幕开始

//如果敌人没有被打倒，且搜索到的可以登录的子弹的编号是有效的情况

if(shot[n].flag!=2 && (k=shot\_search(n))!=-1){

shot[n].bullet[k].knd =enemy[shot[n].num].blknd2;//子弹的种类

shot[n].bullet[k].angle =shotatan2(n);//角度

shot[n].bullet[k].flag =1;//flag

shot[n].bullet[k].x =enemy[shot[n].num].x;//坐标

shot[n].bullet[k].y =enemy[shot[n].num].y;

shot[n].bullet[k].col =enemy[shot[n].num].col;//颜色

shot[n].bullet[k].cnt =0;//计数器

shot[n].bullet[k].spd =3;//速度

se\_flag[0]=1;//置子弹的发射声音flag为有效

}

}

}

由于shot[n].num是登录了的敌人数据的识别编号，因此对于enemy可以像上面那样使用。

其次，最后se\_flag[0]=1是为了展示子弹的发射声效。

由于音乐文件的管理在某个固定的地方进行会比较好，因此这里只将其flag设置为有效。

在music.cpp文件中追加下面两个函数。

---在 music.cpp 中进行以下追加 ---

void music\_ini(){

memset(se\_flag,0,sizeof(int)\*SE\_MAX);

}

void music\_play(){

int i;

for(i=0;i<SE\_MAX;i++){

if(se\_flag[i]==1)

PlaySoundMem(sound\_se[i],DX\_PLAYTYPE\_BACK);

}

}

使用music\_ini，使得循环开始之前，每个se的flag都为0。

接着，在循环的最后使用music\_play播放所有flag为有效的编号的效果音。

这么一来，PlaySoundMem仅在一个固定的地方使用，无论是管理还是调试都很轻松了。

最后，我们得让在shot.cpp中追加了的函数能够在其它地方调用。

---在 function.h 中进行如下追加 ---

//shot.cpp

GLOBAL double shotatan2(int n);

GLOBAL int shot\_search(int n);

GLOBAL void shot\_main();

//music.cpp

GLOBAL void music\_ini();

GLOBAL void music\_play();

---在 load.cpp 中进行如下追加 ---

//读入子弹的图像文件

LoadDivGraph( "../dat/img/bullet/b0.png" , 5 , 5 , 1 , 76 , 76 , img\_bullet[0] ) ;

LoadDivGraph( "../dat/img/bullet/b1.png" , 6 , 6 , 1 , 22 , 22 , img\_bullet[1] ) ;

LoadDivGraph( "../dat/img/bullet/b2.png" , 10 , 10 , 1 , 5 , 120 , img\_bullet[2] ) ;

LoadDivGraph( "../dat/img/bullet/b3.png" , 5 , 5 , 1 , 19 , 34 , img\_bullet[3] ) ;

LoadDivGraph( "../dat/img/bullet/b4.png" , 10 , 10 , 1 , 38 , 38 , img\_bullet[4] ) ;

LoadDivGraph( "../dat/img/bullet/b5.png" , 3 , 3 , 1 , 14 , 16 , img\_bullet[5] ) ;

LoadDivGraph( "../dat/img/bullet/b6.png" , 3 , 3 , 1 , 14 , 18 , img\_bullet[6] ) ;

LoadDivGraph( "../dat/img/bullet/b7.png" , 9 , 9 , 1 , 16 , 16 , img\_bullet[7] ) ;

LoadDivGraph( "../dat/img/bullet/b8.png" , 10 , 10 , 1 , 12 , 18 ,img\_bullet[8] ) ;

LoadDivGraph( "../dat/img/bullet/b9.png" , 3 , 3 , 1 , 13 , 19 , img\_bullet[9] ) ;

//读入敌人的射击音效

sound\_se[0]=LoadSoundMem("../dat/se/enemy\_shot.wav");

最后就是绘制子弹了。

某个弹幕信息（shot）中的子弹（bullet）发射的时候，其flag应该是有效的。

此时检查整个数组的flag，如果某个种类子弹的flag为有效，那么就显示之。

这里有一个叫做“线性插值绘制”的不好理解的词汇。

当我们使用这个技术的时候，像进行“在坐标为1.5像素的地方绘制”这种，想要显示图像的位置有小数点的场合，便可以非常好地在中间进行插值，然后描绘出来。

现在，请想象一下我们在中学的时候学习的一次函数“y=10x”的图像。它是笔直地倾泻向右上方延伸呢。

当这种射击进行的时候，可以想象，如果只在没有小数点的像素上前进，那么子弹会前进地会变得摇摇晃晃的。

因此如果我们使用被称为双线性（Bilinear）法的方法来描绘的话，就可以在其中进行很好地插值并描绘出来。（译者注：插值，这里也可以译为补间，如果知道补间动画这个概念的人应该会很容易明白这个道理。）

---在 graph.cpp 中进行以下追加 ---

//绘制子弹

void graph\_bullet(){

int i,j;

SetDrawMode( DX\_DRAWMODE\_BILINEAR ) ;//线性插值绘制

for(i=0;i<SHOT\_MAX;i++){//敌人的弹幕数量次循环

if(shot[i].flag>0){//如果敌人的弹幕数据为有效

for(j=0;j<SHOT\_BULLET\_MAX;j++){//弹幕所拥有的子弹的最大数量次循环

if(shot[i].bullet[j].flag!=0){//如果子弹的数据为有效

if(shot[i].bullet[j].eff==1)

SetDrawBlendMode( DX\_BLENDMODE\_ADD, 255) ;

DrawRotaGraphF(

shot[i].bullet[j].x+FIELD\_X, shot[i].bullet[j].y+FIELD\_Y,

1.0, shot[i].bullet[j].angle+PI/2,

img\_bullet[shot[i].bullet[j].knd][shot[i].bullet[j].col],TRUE);

if(shot[i].bullet[j].eff==1)

SetDrawBlendMode( DX\_BLENDMODE\_NOBLEND, 0) ;

}

}

}

}

SetDrawMode(DX\_DRAWMODE\_NEAREST);//还原绘制状态

}

---在 graph.cpp 中追加红字部分 ---

void graph\_main(){

graph\_enemy();

graph\_ch();

graph\_bullet();

graph\_board();

}

---在 load.cpp 中进行红字部分变更 ---

//从Excel中读入敌人出现信息并保存的函数

void load\_story(){

int n,num,i,fp;

char fname[32]={"../dat/csv/13章/storyH0.csv"};

int input[64];

char inputc[64];

fp = FileRead\_open(fname);//读入文件

if(fp == NULL){

printfDx("read error\n");

return;

}

运行结果

如果像运行结果那样敌人进行着各种动作的话那就成功了。

## 第十四章 来制作各种各样的弹幕数据吧

接下来，麻烦的设定已经完成了，现在我们来制作各种各样的弹幕数据吧。

一般地，自机狙的弹幕函数是001.

通过往shotatan2函数传入n，这个函数返回自机和传入的第n号敌人之间的夹角，就简单地可以往angle代入和自机之间的夹角。

//100次计数中发射10发，朝着自机直线发射（一般是自机狙）

void shot\_bullet\_H001(int n){

int t=shot[n].cnt;

int k;

if(t>=0 && t<100 && t%10==0){ //100次计数每10次计数1次

if(shot[n].flag!=2 && (k=shot\_search(n))!=-1) //如果敌人存在且子弹可以被登录

shot[n].bullet[k].knd =enemy[shot[n].num].blknd2;//子弹的登录

shot[n].bullet[k].angle =shotatan2(n);//角度

shot[n].bullet[k].flag =1;//flag

shot[n].bullet[k].x =enemy[shot[n].num].x;//坐标

shot[n].bullet[k].y =enemy[shot[n].num].y;

shot[n].bullet[k].col =enemy[shot[n].num].col;//颜色

shot[n].bullet[k].cnt =0;//计数器

shot[n].bullet[k].spd =3;//速度

se\_flag[0]=1;

}

}

}

如果只是看数值的话肯定不容易理解，请与运行结果对照着来看。

---- shotH.cpp变更 ----

#include "../include/GV.h"

//只发射一发，向着自机直线发射

void shot\_bullet\_H000(int n){

int k;

if(shot[n].cnt==0){

if(shot[n].flag!=2 && (k=shot\_search(n))!=-1){

shot[n].bullet[k].knd =enemy[shot[n].num].blknd2;

shot[n].bullet[k].angle =shotatan2(n);

shot[n].bullet[k].flag =1;

shot[n].bullet[k].x =enemy[shot[n].num].x;

shot[n].bullet[k].y =enemy[shot[n].num].y;

shot[n].bullet[k].col =enemy[shot[n].num].col;

shot[n].bullet[k].cnt =0;

shot[n].bullet[k].spd =3;

se\_flag[0]=1;

}

}

}

//100次计数中发射10发，向着自机直线发射（一般是自机狙）

void shot\_bullet\_H001(int n){

int t=shot[n].cnt;

int k;

if(t>=0 && t<100 && t%10==0){//100カウント中10カウントに1回

//100次计数中每10次计数发射1次

if(shot[n].flag!=2 && (k=shot\_search(n))!=-1){//敌人存在，并且子弹可登录的话

shot[n].bullet[k].knd =enemy[shot[n].num].blknd2;//子弹登录

shot[n].bullet[k].angle =shotatan2(n);//角度

shot[n].bullet[k].flag =1;//flag

shot[n].bullet[k].x =enemy[shot[n].num].x;//坐标

shot[n].bullet[k].y =enemy[shot[n].num].y;

shot[n].bullet[k].col =enemy[shot[n].num].col;//颜色

shot[n].bullet[k].cnt =0;//计数器

shot[n].bullet[k].spd =3;//速度

se\_flag[0]=1;

}

}

}

//100次计数中发射10发，向着自机直线发射（记忆角度）

void shot\_bullet\_H002(int n){

int t=shot[n].cnt;

int k;

if(t>=0 && t<100 && t%10==0){

if(t==0)

shot[n].base\_angle[0]=shotatan2(n);

if(shot[n].flag!=2 && (k=shot\_search(n))!=-1){

shot[n].bullet[k].knd =enemy[shot[n].num].blknd2;

shot[n].bullet[k].angle =shot[n].base\_angle[0];

shot[n].bullet[k].flag =1;

shot[n].bullet[k].x =enemy[shot[n].num].x;

shot[n].bullet[k].y =enemy[shot[n].num].y;

shot[n].bullet[k].col =enemy[shot[n].num].col;

shot[n].bullet[k].cnt =0;

shot[n].bullet[k].spd =3;

se\_flag[0]=1;

}

}

}

//100次计数中10发，向着自机速度变化地直线发射

void shot\_bullet\_H003(int n){

int t=shot[n].cnt;

int k;

if(t>=0 && t<100 && t%10==0){

if(shot[n].flag!=2 && (k=shot\_search(n))!=-1){

shot[n].bullet[k].knd =enemy[shot[n].num].blknd2;

shot[n].bullet[k].angle =shotatan2(n);

shot[n].bullet[k].flag =1;

shot[n].bullet[k].x =enemy[shot[n].num].x;

shot[n].bullet[k].y =enemy[shot[n].num].y;

shot[n].bullet[k].col =enemy[shot[n].num].col;

shot[n].bullet[k].cnt =0;

shot[n].bullet[k].spd =1+5.0/100\*t;

se\_flag[0]=1;

}

}

}

//0.5秒一次地圆形发射

void shot\_bullet\_H004(int n){

int t=shot[n].cnt;

int k;

if(t>=0 && t<120 && t%20==0){

double angle=shotatan2(n);

for(int i=0;i<20;i++){

if(shot[n].flag!=2 && (k=shot\_search(n))!=-1){

shot[n].bullet[k].knd =enemy[shot[n].num].blknd2;

shot[n].bullet[k].angle =angle+PI2/20\*i;

shot[n].bullet[k].flag =1;

shot[n].bullet[k].x =enemy[shot[n].num].x;

shot[n].bullet[k].y =enemy[shot[n].num].y;

shot[n].bullet[k].col =enemy[shot[n].num].col;

shot[n].bullet[k].cnt =0;

shot[n].bullet[k].spd =4;

}

se\_flag[0]=1;

}

}

}

//散射

void shot\_bullet\_H005(int n){

int t=shot[n].cnt;

int k;

if(t>=0 && t<120 && t%2==0){

if(shot[n].flag!=2 && (k=shot\_search(n))!=-1){

shot[n].bullet[k].knd =enemy[shot[n].num].blknd2;

shot[n].bullet[k].angle =shotatan2(n)+rang(PI/4);

shot[n].bullet[k].flag =1;

shot[n].bullet[k].x =enemy[shot[n].num].x;

shot[n].bullet[k].y =enemy[shot[n].num].y;

shot[n].bullet[k].col =enemy[shot[n].num].col;

shot[n].bullet[k].cnt =0;

shot[n].bullet[k].spd =3+rang(1.5);

se\_flag[0]=1;

}

}

}

//散射（减速）

void shot\_bullet\_H006(int n){

int t=shot[n].cnt;

int k;

if(t>=0 && t<120 && t%2==0){

if(shot[n].flag!=2 && (k=shot\_search(n))!=-1){

shot[n].bullet[k].knd =enemy[shot[n].num].blknd2;

shot[n].bullet[k].angle =shotatan2(n)+rang(PI/4);

shot[n].bullet[k].flag =1;

shot[n].bullet[k].x =enemy[shot[n].num].x;

shot[n].bullet[k].y =enemy[shot[n].num].y;

shot[n].bullet[k].col =enemy[shot[n].num].col;

shot[n].bullet[k].cnt =0;

shot[n].bullet[k].spd =4+rang(2);

se\_flag[0]=1;

}

}

for(int i=0;i<SHOT\_BULLET\_MAX;i++){//所有子弹的循环

if(shot[n].bullet[i].spd>1.5)//如果速度大于1.5的话

shot[n].bullet[i].spd-=0.04;//减速

}

}

追加上述弹幕的同时也进行以下变更。

/计数器 移动模式 敌人种类 x坐标 y坐标 速度 发射时间 弹幕种类 子弹的颜色 体力 子弹的种类 待机时间 物品1 2 3 4 5 6

/cnt pattern knd x y sp bltime blknd col hp blknd2 wait item\_n

60 9 0 200 150 0 60 0 0 100 7 120 0 -1 -1 -1 -1 -1

200 9 0 180 150 0 60 1 0 100 8 120 0 -1 -1 -1 -1 -1

340 9 0 200 150 0 60 2 0 100 9 120 0 -1 -1 -1 -1 -1

480 9 0 180 150 0 60 3 0 100 0 120 0 -1 -1 -1 -1 -1

620 9 0 200 150 0 60 4 0 100 7 120 0 -1 -1 -1 -1 -1

760 9 0 180 150 0 60 5 0 100 8 120 0 -1 -1 -1 -1 -1

900 9 0 180 150 0 60 6 0 100 8 120 0 -1 -1 -1 -1 -1

移动模式的话，采用模式9，这个模式在在敌人完全出现之后就往上移动。由于这次制作了0~6的弹幕数据，因此弹幕种类传入了0~6的值。

下载Excel数据

----在 shot.cpp的exntern声明部分和指针的带入部分进行以下变更----

extern void shot\_bullet\_H000(int);

extern void shot\_bullet\_H001(int);

extern void shot\_bullet\_H002(int);

extern void shot\_bullet\_H003(int);

extern void shot\_bullet\_H004(int);

extern void shot\_bullet\_H005(int);

extern void shot\_bullet\_H006(int);

void (\*shot\_bullet[SHOT\_KND\_MAX])(int) ={

shot\_bullet\_H000,

shot\_bullet\_H001,

shot\_bullet\_H002,

shot\_bullet\_H003,

shot\_bullet\_H004,

shot\_bullet\_H005,

shot\_bullet\_H006,

};

----在 function.h 进行以下追加 ----

GLOBAL double rang(double);

----在 define.h 进行以下变更 ----

#define SHOT\_KND\_MAX 7

运行结果

如果和运行结果一样敌人发射了各种各样的弹幕的话那就成功了。

## 第十五章 制作妖梦的弹幕吧みょん

（译者注：这个标题原文为“みょん弾幕を作ってみょん”，“みょん”这个妖妖梦里面的梗我实在不知道怎么翻译才好，干脆直接按照niconico大百科的说法直接翻成妖梦好了……，具体情况还是请各位看官自行百度吧。）

诶？题目很奇怪？别在意这些细节了啦。

接下来，既然我们已经知道了基本弹幕的制作方法了，那么我们就试着做做稍微复杂一点的弹幕吧。

……虽然这么说，要是突然就增加难度那可就不容易理解了，首先我们还是从简单的弹幕开始。

这次，我们先把运行结果给大家看看。

运行结果

具体来说，和妖梦的弹幕比起来还是有一些区别，如果要非常相似的话，计算的式子会很复杂，所以这里就简略化了。

要想这样子射出子弹，需要怎么做呢？

让发射源以敌人所在位置为中心回旋着移动？

从下往上进行半回旋。

从敌人所在的位置开始，从PI/2开始先后在PI/2+PI的方向和PI/2-PI的方向的两个方向之间回旋？

这次我们试着将这个动作在150次计数中进行15次射击来表现。

----在 shotH.cpp中进行以下追加 ----

//妖梦弹幕

void shot\_bullet\_H007(int n){

int t=shot[n].cnt;

int k;

if(t>=0 && t<=150 && t%10==0){

for(int i=0;i<20;i++){

if(shot[n].flag!=2 && (k=shot\_search(n))!=-1){

shot[n].bullet[k].knd =7;

shot[n].bullet[k].angle =PI2/20\*i;

shot[n].bullet[k].flag =1;

shot[n].bullet[k].x =enemy[shot[n].num].x+cos(PI/2+PI/150\*t)\*100;

shot[n].bullet[k].y =enemy[shot[n].num].y+sin(PI/2+PI/150\*t)\*100;

shot[n].bullet[k].col =2;

shot[n].bullet[k].cnt =0;

shot[n].bullet[k].spd =1.2;

}

}

for(int i=0;i<20;i++){

if(shot[n].flag!=2 && (k=shot\_search(n))!=-1){

shot[n].bullet[k].knd =7;

shot[n].bullet[k].angle =PI2/20\*i;

shot[n].bullet[k].flag =1;

shot[n].bullet[k].x =enemy[shot[n].num].x+cos(PI/2-PI/150\*t)\*100;

shot[n].bullet[k].y =enemy[shot[n].num].y+sin(PI/2-PI/150\*t)\*100;

shot[n].bullet[k].col =4;

shot[n].bullet[k].cnt =0;

shot[n].bullet[k].spd =1.2;

se\_flag[0]=1;

}

}

}

}

cos(PI/2+PI/150\*t)\*100;

中的cos的作用是在0~1的范围内计算出x分量，然后乘以圆的半径100。

如果想要变更圆的大小的话，就调整这个100就行了。

其次，如果每一次都要变更声明、定义的话很麻烦，所以这里我们先声明18个弹幕吧。

----在 shotH.cpp中进行以下追加 ----

void shot\_bullet\_H008(int n){}

void shot\_bullet\_H009(int n){}

void shot\_bullet\_H010(int n){}

void shot\_bullet\_H011(int n){}

void shot\_bullet\_H012(int n){}

void shot\_bullet\_H013(int n){}

void shot\_bullet\_H014(int n){}

void shot\_bullet\_H015(int n){}

void shot\_bullet\_H016(int n){}

void shot\_bullet\_H017(int n){}

----在 shot.cpp 中任意地方进行以下修改 ----

extern void shot\_bullet\_H000(int);extern void shot\_bullet\_H001(int);extern void shot\_bullet\_H002(int);

extern void shot\_bullet\_H003(int);extern void shot\_bullet\_H004(int);extern void shot\_bullet\_H005(int);

extern void shot\_bullet\_H006(int);extern void shot\_bullet\_H007(int);extern void shot\_bullet\_H008(int);

extern void shot\_bullet\_H009(int);extern void shot\_bullet\_H010(int);extern void shot\_bullet\_H011(int);

extern void shot\_bullet\_H012(int);extern void shot\_bullet\_H013(int);extern void shot\_bullet\_H014(int);

extern void shot\_bullet\_H015(int);extern void shot\_bullet\_H016(int);extern void shot\_bullet\_H017(int);

void (\*shot\_bullet[SHOT\_KND\_MAX])(int) ={

shot\_bullet\_H000,shot\_bullet\_H001,shot\_bullet\_H002,shot\_bullet\_H003,shot\_bullet\_H004,

shot\_bullet\_H005,shot\_bullet\_H006,shot\_bullet\_H007,shot\_bullet\_H008,shot\_bullet\_H009,

shot\_bullet\_H010,shot\_bullet\_H011,shot\_bullet\_H012,shot\_bullet\_H013,shot\_bullet\_H014,

shot\_bullet\_H015,shot\_bullet\_H016,shot\_bullet\_H017,

};

----对 define.h 的定义进行以下修改 ----

#define SHOT\_KND\_MAX 18

这是Excel数据的修改的地方。

从现在开始，由于弹幕的编号会发生改变，请对“弹幕种类”的编号进行逐一增加。

/计数器 移动模式 敌人的种类 x坐标 y坐标 速度 发射时间 弹幕种类 子弹的颜色 体力 子弹的种类 待机时间 物品1 2 3 4 5 6

/cnt pattern knd x y sp bltime blknd col hp blknd2 wait item\_n

60 9 0 200 150 0 60 7 0 100 0 200 0 -1 -1 -1 -1 -1

Excel数据下载。

## 第十六章 来制作诹访子大人的弹幕吧

在前面的章节中，我们已经完善好了我们的模块，因此下面弹幕的追加将会很轻松。

现在我们只需要修改弹幕函数就行了。

我们用shot\_bullet\_H008来实现诹访子大人的弹幕吧。

诹访子大人的弹幕是东方风神录EX的最后一个弹幕。如果不是很清楚请在视频中确认一下。

http://www.nicovideo.jp/search/%E3%83%9F%E3%82%B7%E3%83%A3%E3%82%B0%E3%82%B8%E3%81%95%E3%81%BE?sort=v

出人意外的是，EX的最后一个弹幕非常简单地就实现出来了……听到这种说法不少东方厨得烧我了吧，但是确实是很简单地就做出了非常相似的弹幕。

----在 shotH.cpp中进行以下追加 ----

//诹访子大人

void shot\_bullet\_H008(int n){

int t=shot[n].cnt;

int k;

if(t>=0 && t<1200 && t%90==0){

double angle=rang(PI);

for(int j=0;j<2;j++){//中途的时候就分裂为两部分

for(int i=0;i<60;i++){//一次60个

if(shot[n].flag!=2 && (k=shot\_search(n))!=-1){

shot[n].bullet[k].knd =8;//第8号子弹

shot[n].bullet[k].angle =angle+PI2/60\*i;//60个圆

shot[n].bullet[k].flag =1;

shot[n].bullet[k].x =enemy[shot[n].num].x;

shot[n].bullet[k].y =enemy[shot[n].num].y;

shot[n].bullet[k].col =4;//第4号颜色

shot[n].bullet[k].cnt =0;

shot[n].bullet[k].state =j;//状态。0或者是1表示不同旋转。

shot[n].bullet[k].spd =2;

}

}

}

se\_flag[0]=1;//播放发射音效

}

for(int i=0;i<SHOT\_BULLET\_MAX;i++){//所有子弹的循环

if(shot[n].bullet[i].flag>0){//如果有登录了的子弹

int state=shot[n].bullet[i].state;

int cnt=shot[n].bullet[i].cnt;

if(30<cnt && cnt<120){//如果是30~120次计数

shot[n].bullet[i].spd-=1.2/90.0;//90次计数总共减去1.2

shot[n].bullet[i].angle+=(PI/2)/90.0\*(state?-1:1);//90次计数总共倾斜90°

}

}

}

}

请保证子弹的编号和bullet文件中的子弹的编号是一样的。

(b8.png)

从最左边开始为0号，由于淡蓝色的编号为1，因此指定颜色为1.

在这里我们让state为0或者1。由于诹访子大人的弹幕是从一个地方开始两个两个地发射，因此中途开始我们就要变成两个轨道。

因此，我们试着让指定为0的子弹角度顺时针旋转，而指定为1的子弹角度为逆时针旋转。

shot[n].bullet[i].angle+=(PI/2)/90.0\*(state?-1:1);

这就是state的代入。问号用来表示三元表达式。如果不清楚的请谷歌一下。

这样以来，我们就可以分成两种角度了。

/计数器 移动模式 敌人的种类 x坐标 y坐标 速度 发射时间 弹幕种类 子弹的颜色 体力 子弹的种类 待机时间 物品1 2 3 4 5 6

/cnt pattern knd x y sp bltime blknd col hp blknd2 wait item\_n

60 9 0 200 150 0 60 8 1 100 0 1200 0 -1 -1 -1 -1 -1

运行结果

## 第十七章 让自机射出子弹吧

慢慢地，游戏开始有点游戏的样子了呢。

现在我们加上自机的射击和碰撞判定，这样一来大体上的构架就完成了。

自机的射击的做法大体上而言和敌人的射击没有多大的区别。

我们先定义保存子弹情报的容器，这个容器里面只需要保存种类、速度、角度等信息就可以了。

只要我们事先设定好了数字，那么计算部分就能够方便地进行轨道和碰撞判定的计算了。

那么，我们和平常一样准备这个容器。

----在 struct.h 追加以下红字部分 ----

//和角色射击相关的结构体

typedef struct{

int flag,power,cnt,knd;//flag、power、计数器、种类

double x,y,angle,spd;//坐标、角度、速度

}cshot\_t;

//和角色相关的结构体

typedef struct{

int flag; //flag

int cnt; //计数器

int power; //power

int point; //point

int score; //score

int num; //残机数

int mutekicnt; //无敌状态及其计数器

int shot\_mode; //射击模式

int money; //金钱

int img;

int slow; //是否缓慢移动

double x,y; //坐标

int shot\_cnt; //射击的计数器

}ch\_t;

----在 define.h 中进行以下追加 ----

//自机射击的子弹的登录最大数

#define CSHOT\_MAX 200

----在 GV.h 中进行以下追加 ----

GLOBAL int img\_cshot[2]; //自机射击用的图像

GLOBAL cshot\_t cshot[CSHOT\_MAX];//自机射击用的变量

----在 load.cpp 的 load()函数中进行以下追加 ----

img\_cshot[0]=LoadGraph("../dat/img/char/bl\_00.png");

img\_cshot[1]=LoadGraph("../dat/img/char/bl\_01.png");

sound\_se[2]=LoadSoundMem("../dat/se/cshot.wav");

----在 ini.cpp 的 ini函数中进行以下追加 ----

memset(cshot,0,sizeof(cshot\_t)\*CSHOT\_MAX);

ch.power=500;

----在 function.h 中进行以下追加 ----

GLOBAL void cshot\_main();

----在 graph.cpp 中进行以下追加 ----

void graph\_cshot(){

for(int i=0;i<CSHOT\_MAX;i++){

if(cshot[i].flag>0){

DrawRotaGraphF(cshot[i].x+FIELD\_X,cshot[i].y+FIELD\_Y,1,0,

img\_cshot[cshot[i].knd],TRUE);

}

}

}

----修改 graph.cpp 的 graph\_main函数 ----

void graph\_main(){

graph\_enemy();

graph\_cshot();

graph\_ch();

graph\_bullet();

graph\_board();

}

---- main.cpp 的main函数内的switch语句部分进行以下变更 ----

case 100://通常处理

calc\_ch(); //角色计算

ch\_move(); //控制角色移动

cshot\_main();//自机射击main

enemy\_main();//敌人处理main

shot\_main();//射击main

graph\_main();//绘制main

stage\_count++;

break;

最后是重要的自机射击登录部分。

我们试着让自机的power在200以下的话一次发射2发子弹，200以上的话一次发射4发。

这4发的位置如下所示。

int cshot0pos\_x[4]={-10, 10,-30, 30};

int cshot0pos\_y[4]={-30,-30,-10,-10};

第1发的位置是（-10，-30），第2发的位置是（10，-30），第3发的位置是（-30，-10）……大概就这样。

这次也和前面的章节中一样，由于先定义了登录数据库，于是在登录射击的时候通过

int search\_cshot()

搜索空着的编号，然后登录之。

---- cshot.cpp 变更 ----

#include "../include/GV.h"

int cshot0num[2] ={2,4};

int cshot0pos\_x[4]={-10, 10,-30, 30};

int cshot0pos\_y[4]={-30,-30,-10,-10};

//返回自机射击登录可能的编号。

int search\_cshot(){

for(int i=0;i<CSHOT\_MAX;i++){

if(cshot[i].flag==0)

return i;

}

return -1;

}

//一般射击的登录

void ch0\_shot\_pattern(){

int k;

for(int i=0;i<cshot0num[ch.power<200?0:1];i++){

if((k=search\_cshot())!=-1){

cshot[k].flag=1;

cshot[k].cnt=0;

cshot[k].angle=-PI/2;

cshot[k].spd=20;

cshot[k].x=ch.x+cshot0pos\_x[i];

cshot[k].y=ch.y+cshot0pos\_y[i];

cshot[k].power=23;

cshot[k].knd=0;

}

}

se\_flag[2]=1;//播放发射音

}

//低速下一般射击的登录

void ch1\_shot\_pattern(){

int k;

for(int i=0;i<cshot0num[ch.power<200?0:1];i++){

if((k=search\_cshot())!=-1){

cshot[k].flag=1;

cshot[k].cnt=0;

cshot[k].angle=-PI/2;

cshot[k].spd=20;

cshot[k].x=ch.x+cshot0pos\_x[i]/3;//如果处于低速状态的话，将位置向中心偏移

cshot[k].y=ch.y+cshot0pos\_y[i]/2;

cshot[k].power=23;

cshot[k].knd=0;

}

}

se\_flag[2]=1;

}

//射击登录部分

void enter\_shot(){

//当按下射击按钮的时候

if(CheckStatePad(configpad.shot)>0){

ch.shot\_cnt++;

if(ch.shot\_cnt%3==0){//每3次计数射击一次

if(CheckStatePad(configpad.slow)>0)//如果处于低速移动中的话

ch1\_shot\_pattern();

else

ch0\_shot\_pattern();

}

}

else

ch.shot\_cnt=0;

}

//射击的移动计算

void calc\_cshot(){

for(int i=0;i<CSHOT\_MAX;i++){

if(cshot[i].flag==1){

int dranx=cshot[i].spd+11/2,drany=cshot[i].spd+55/2;

cshot[i].x+=cos(cshot[i].angle)\*cshot[i].spd;

cshot[i].y+=sin(cshot[i].angle)\*cshot[i].spd;

cshot[i].cnt++;

if(cshot[i].x<-dranx || cshot[i].x>FIELD\_MAX\_X+dranx ||

cshot[i].y<-drany || cshot[i].y>FIELD\_MAX\_Y+drany)//如果跑到画面外了的话

cshot[i].flag=0;

}

}

}

//和角色射击相关的函数

void cshot\_main(){

calc\_cshot();//射击轨道的计算

enter\_shot();//射击登录

}

在calc\_cshot函数中判定射击的子弹从画面内进入画面外的情况时，为何要使用

int dranx=cshot[i].spd+11/2,drany=cshot[i].spd+55/2;

来判定，这里我来补充说明一下。由于下一章将要介绍碰撞判定，因此有必要事先理解碰撞判定，但是由于下一章中这段代码不会出现了，因此我事先说明一下。

如果我们仅仅单纯地使用勾股定理来计算是否接触，是无法进行碰撞判定的。

比如射击的速度是200而子弹的碰撞判定是10，那么就会有190空出来的无用空间。（译者注：本文章的作者Dixq在他的网站的游戏编程馆s11章介绍了用勾股定理进行碰撞判定的方法，这里的描述即是那篇文章中用到的说法。）

也即是说没有必要计算子弹的轨迹去判定是否接触。

由于碰撞判定的计算是和轨迹相关的，如果子弹一旦跑到画面外就让它消失的话就无法计算最后的轨迹了。

也即是说，在之前的帧中已经在画面外且当前帧也是在画面外的情况下，才有必要销毁之。

其次，11,55指的是射击的子弹的图片尺寸。

如果已经超出画面速度+图片尺寸/2这么多像素的话，由于已经知道上一帧也在画面外面，因此这个时候就有必要销毁这个子弹了。

上面那段代码就是为了实现上面的算法进行的处理。

运行结果：

## 第十八章 为自机的射击添上碰撞判定吧

这一章中将要介绍碰撞判定。

如果是做过STG的人的话，大概马上就会说“啊啊，使用勾股定理的话不是很容易吗”，不过在高速射击的的情况下单单这样是不够的。

如果您还不明白碰撞判定中利用勾股定理进行判断的方法的话，请在游戏编程馆s11章中进行确认。链接。

假如碰撞判定的有效判定为10，那么如果子弹的速度是100的话会怎么样？

子弹100、100地前进，而由于碰撞判定是10、10地进行，子弹高速移动的情况下，就会出现没有被检查到的区域。

因此，有可能会出现敌人擦着自机子弹过去的情况。

为了不出现这种情况，必须一点一点地检查自机射击的子弹所经过的地方，不能漏掉一个地方。

因此，虽然有点麻烦，我们还是像下面那样实现吧。

在out\_judge\_cshot函数中对传入的第i号自机的射击数据和第s号敌人数据进行碰撞判定，

＞if(cshot[i].cnt>0){

也即是说，如果子弹至少计算过一次轨道的话，那么回到1帧之前的位置1次，然后反复以子弹和敌人的碰撞判定之合计那么多的量向当前的位置接近，这样，每次都检查一下是否已经碰撞。

下面是上述的实现。

---- out.cpp 变更 ----

#include "../include/GV.h"

#define ENEMY\_RANGE\_MAX 4

#define CSHOT\_RANGE\_MAX 2

//敌人的碰撞判定范围

int enemy\_range[ENEMY\_RANGE\_MAX]={16,30,16,50};

//自机的射击的子弹碰撞判定范围

int cshot\_range[CSHOT\_RANGE\_MAX]={6,};

//判定是否击中

int out\_judge\_cshot(int i,int s){

int j;

if(cshot[i].cnt>0){ //如果射击的轨道至少计算过一次

double x=cshot[i].x-enemy[s].x;//敌人和自机射击的子弹的距离

double y=cshot[i].y-enemy[s].y;

//防止溢出

if(cshot[i].knd>=CSHOT\_RANGE\_MAX || enemy[s].knd>=ENEMY\_RANGE\_MAX)

printfDx("out\_judge\_cshot中溢出");

//敌人的碰撞判定和自机射击的子弹的碰撞判定的合计范围

double r=cshot\_range[cshot[i].knd]+enemy\_range[enemy[s].knd];

//如果有必要计算中间

if(cshot[i].spd>r){

//保存1帧之前的位置

double pre\_x=cshot[i].x+cos(cshot[i].angle+PI)\*cshot[i].spd;

double pre\_y=cshot[i].y+sin(cshot[i].angle+PI)\*cshot[i].spd;

double px,py;

for(j=0;j<cshot[i].spd/r;j++){//前进的分量÷碰撞判定分量次循环

px=pre\_x-enemy[s].x;

py=pre\_y-enemy[s].y;

if(px\*px+py\*py<r\*r)

return 1;

pre\_x+=cos(cshot[i].angle)\*r;

pre\_y+=sin(cshot[i].angle)\*r;

}

}

if(x\*x+y\*y<r\*r)//如果在碰撞判定范围内

return 1;//碰撞

}

return 0;

}

//决定敌人是否死掉

void enemy\_death\_judge(int s){

int i;

se\_flag[8]=1;//击中敌人的声音

if(enemy[s].hp<0){//如果敌人的HP小于0

enemy[s].flag=0;//消灭敌人

se\_flag[1]=1;//敌人击毁的声音（原文是敵のピチュり音，至于这个梗……直接谷歌“ピチュる　とは”就清楚了。）

for(i=0;i<SHOT\_MAX;i++){//敌人总数次循环

if(shot[i].flag!=0){//如果有没有登录的弹幕

if(s==shot[i].num){//且敌人有已经登录了的弹幕

shot[i].flag=2;//改变flag以表示弹幕不再继续

break;

}

}

}

}

}

//碰撞判定main

void out\_main(){

int i,s;

for(i=0;i<CSHOT\_MAX;i++){//自机射击总数

if(cshot[i].flag>0){

for(s=0;s<ENEMY\_MAX;s++){//敌人总数

if(enemy[s].flag>0){

if(out\_judge\_cshot(i,s)){//如果自机子弹射中敌人

cshot[i].flag=0;//将那个自机子弹销毁

enemy[s].hp-=cshot[i].power;//将HP减少子弹的power的量那么多

enemy\_death\_judge(s);//决定敌人是否死掉

}

}

}

}

}

}

在out\_main中，进行所有的自机射击的子弹和所有敌人的总判定。

因此与此同时，我们和之前一样进行以下变更。

----在 main.cpp 中进行以下变更----

case 100://通常处理

calc\_ch(); //角色计算

ch\_move(); //控制角色移动

cshot\_main();//自机射击main

enemy\_main();//敌人处理main

shot\_main(); //射击main

out\_main(); //碰撞计算

graph\_main();//绘制main

stage\_count++;

break;

----在 function.h 中进行以下追加 ----

//out.cpp

GLOBAL void out\_main();

----在 load.cpp 的 load函数中进行以下追加 ----

sound\_se[1]=LoadSoundMem("dat/se/enemy\_death.wav");

sound\_se[2]=LoadSoundMem("dat/se/cshot.wav");

sound\_se[8]=LoadSoundMem("dat/se/hit.wav");

ChangeVolumeSoundMem( 50, sound\_se[0] ) ;//设定各素材的重播音量

ChangeVolumeSoundMem(128, sound\_se[1] ) ;

ChangeVolumeSoundMem(128, sound\_se[2] ) ;

ChangeVolumeSoundMem( 80, sound\_se[8] ) ;

----在 music.cpp 的 music\_play函数进行以下变更 ----

void music\_play(){

int i;

for(i=0;i<SE\_MAX;i++){

if(se\_flag[i]==1){

if(CheckSoundMem(sound\_se[i])!=0){

if(i==8)continue;

StopSoundMem(sound\_se[i]);

}

PlaySoundMem(sound\_se[i],DX\_PLAYTYPE\_BACK);

}

}

}

至于为何要在最后追加的music\_play函数中写上

if(i==8)continue;

那是因为第8号效果音是击中敌人时候的声音，如果它处于播放中，让它停止然后再重播的话，就会听到断断续续不好听的声音。

也就是说，必须要在它重播结束之后在重播才行。

现在的情况下有这种必要的只有第8号，因此就先这样了。

Excel数据也要适当地变更。

运行结果

## 第十九章 给敌人加上消灭特效吧

如果敌人突然就消失了，看起来实在是很寂寞，现在我们试着做出消灭特效吧。

那么，我先追加一个效果用的文件。

在解决方案管理器里面，右键project项目追加一个新的项目，现在我们往mydat文件夹里面的source里面追加“effect.cpp”。

然后像下面那样复制过去。

---- effect.cpp ----

#include "../include/GV.h"

//检查登录为空的效果编号

int search\_effect(){

for(int i=0;i<EFFECT\_MAX;i++){

if(effect[i].flag==0)

return i;

}

return -1;

}

//计算效果

void calc\_effect(){

for(int i=0;i<EFFECT\_MAX;i++){

if(effect[i].flag>0){ //如果效果已经登录了

switch(effect[i].knd){//根据效果的种类进行分歧

case 0://0号的处理

effect[i].cnt++;

effect[i].r+=0.08;//逐渐把效果的大小变大

if(effect[i].cnt>10)//如果计数为10以上

effect[i].brt-=25;//将亮度降低

if(effect[i].cnt>20)//效果为20以上

effect[i].flag=0;//销毁之

break;

default:

printfDx("effect设定错误\n");

break;

}

}

}

}

//检查登录为空的死亡效果编号

int search\_del\_effect(){

for(int i=0;i<DEL\_EFFECT\_MAX;i++){

if(del\_effect[i].flag==0)

return i;

}

return -1;

}

//登录消灭特效

void enter\_del\_effect(int s){

int k=search\_del\_effect();

if(k!=-1){

del\_effect[k].flag=1;//设置flag为有效

del\_effect[k].cnt=0;

del\_effect[k].col=enemy[s].back\_col;//设定敌人的背景色为消灭色

del\_effect[k].x=enemy[s].x;//将敌人的坐标设置为死亡位置

del\_effect[k].y=enemy[s].y;

}

}

//进行消灭特效的计算和特效的登录

void calc\_del\_effect(){

int k;

for(int i=0;i<DEL\_EFFECT\_MAX;i++){

if(del\_effect[i].flag>0){

if(del\_effect[i].cnt%2==0){//每1/30秒登录一次

if((k=search\_effect())!=-1){//搜索空着的编号

memset(&effect[k],0,sizeof(effect\_t));//初始化

effect[k].flag=1;//设置flag为有效

effect[k].brt=255;//将亮度变为最大值

effect[k].ang=rang(PI);//随机角度

effect[k].col=del\_effect[i].col;//设置颜色

effect[k].eff=1;// 1是发光效果。这里设置为光效果。

effect[k].img=img\_del\_effect[effect[k].col];//设置消灭的图像

effect[k].knd=0;//效果的种类是0号（消灭特效的knd一定是0）

effect[k].x=del\_effect[i].x;//设置坐标

effect[k].y=del\_effect[i].y;

}

}

if(del\_effect[i].cnt>8){//如果计数器大于8的话结束

del\_effect[i].flag=0;

}

del\_effect[i].cnt++;

}

}

}

void effect\_main(){

calc\_del\_effect();//消灭特效计算

calc\_effect();//特效计算

}

在enter\_del\_effect中登录消灭特效的话，后面就可以对特效进行全自动地计算，这会非常方便。

这个函数通过参数s传入敌人的识别编号。这样一来，就可以登录s号的敌人的消灭特效了。

现在我们也新定义好了信息数据库。

为了登录消灭特效而定义del\_effect

为了登录特效而定义effect

对于消灭特效和特效两种不同的数据库，可能会有人产生疑问。

消灭特效事实上并不是进行特效，而是负责管理在什么时间点上登录效果、将什么特效往哪里登录之类的事情。

也即是说，实际的效果还是在effect中计算和绘制。

由于这个effect变量是泛用的，因此除了消灭特效以外，还能够登录别的特效。

我们后面将会在特效里面登录许许多多东西，大部分都是在这里处理的。

一旦我们需要许多特效，由于需要使用许多不同的变量，处理起来将会很麻烦么，因此这样做效率比较高。

因此，现在您需要记住“在消灭特效变量之上，还有特效变量。如果仅仅只有消灭特效变量的话并不能形成特效，消灭特效变量的工作只是管理特效”这一点。

在enter\_del\_effect函数中登录的信息将在calc\_del\_effect()函数中计算。

如果您看了注释的话应该就会明白了，一系列的工作是：搜索登录了的消灭特效，如果有的话每1/30秒登录1次特效。然后，搜索效果的登录编号，最后设置信息。

和之前所说的一样，由于效果变量可以使用任何值，因此我们往knd传入0——它用于标识这是哪个种类的特效。而今后我们就把消灭特效设置为0号的特效。

在out\_main中进行自机总射击的子弹与所有敌人的总碰撞判定。

那么与此同时，进行下面那样的修改。

----在 out.cpp 中进行以下部分的变更 ----

extern void enter\_del\_effect(int);

void enemy\_death\_judge(int s){

int i;

se\_flag[8]=1;//击中敌人的声音

if(enemy[s].hp<0){//如果敌人的HP小于0

enemy[s].flag=0;//消灭敌人

se\_flag[1]=1;//敌人的击毁声

enter\_del\_effect(s);

----在 define.h 中进行以下追加 ----

//特效登录最大数

#define EFFECT\_MAX 100

//消灭特效登录最大数

#define DEL\_EFFECT\_MAX 30

----在 struct.h 中进行以下变更 ----

//和敌人相关的结构体

typedef struct{

int flag,cnt,pattern,muki,knd,hp,hp\_max,item\_n[6],img,back\_col;

//flag、计数器、移动模式、方向、敌人的种类、HP最大值、掉落道具、背景色

double x,y,vx,vy,sp,ang;

//坐标、速度x分量、速度y分量、速度、角度

int bltime,blknd,blknd2,col,state,wtime,wait;

//弹幕开始时间、弹幕种类、子弹的种类、颜色、状态、待机时间、停止时间

}enemy\_t;

----在 enemy.cpp的enemy\_enter函数的登录部分进行以下追加----

enemy[i].back\_col=GetRand(4);

----在 function.h 中进行以下追加 ----

//effect.cpp

GLOBAL void effect\_main();

----在 GV.h 中进行以下追加 ----

GLOBAL int img\_del\_effect[5];

GLOBAL del\_effect\_t del\_effect[DEL\_EFFECT\_MAX];//消灭特效

GLOBAL effect\_t effect[EFFECT\_MAX];//特效

----在 ini.cpp 的 ini函数中进行以下追加 ----

memset(effect,0,sizeof(effect\_t)\*EFFECT\_MAX);

memset(del\_effect,0,sizeof(del\_effect\_t)\*DEL\_EFFECT\_MAX);

----在 load.cpp 的 load函数中进行以下追加 ----

LoadDivGraph( "../dat/img/enemy/hit\_effect.png" , 5 , 5 , 1 , 140 , 140 , img\_del\_effect ) ;

----在 main.cpp 的main函数中进行以下变更 ----

case 100://通常处理

calc\_ch(); //角色计算

ch\_move(); //控制角色移动

cshot\_main();//自机射击main

enemy\_main();//敌人处理main

shot\_main(); //射击main

out\_main(); //碰撞判定

effect\_main();//特效main

graph\_main();//绘制main

stage\_count++;

break;

----在 struct.h　中进行以下追加 ----

//特效

typedef struct{

int flag,cnt,col,knd,img,eff,brt;

double x,y,r,ang;

}effect\_t;

//消灭特效

typedef struct{

int flag,cnt,col;

double x,y;

}del\_effect\_t;

----在 graph.cpp 中进行以下追加 ----

void graph\_effect(){

for(int i=0;i<EFFECT\_MAX;i++){

if(effect[i].flag>0){

if(effect[i].eff==1)//如果特效是光特效

SetDrawBlendMode( DX\_BLENDMODE\_ADD, effect[i].brt) ;

DrawRotaGraphF(effect[i].x+FIELD\_X,effect[i].y+FIELD\_Y,effect[i].r,effect[i].ang,effect[i].img,TRUE);

if(effect[i].eff==1)

SetDrawBlendMode( DX\_BLENDMODE\_NOBLEND, 0) ;

}

}

}

void graph\_main(){

graph\_effect();

graph\_enemy();

graph\_cshot();

graph\_ch();

graph\_bullet();

graph\_board();

}

运行结果

## 第二十章 试着描绘背景吧

接下来我们来绘制背景吧。现在我们试着描绘龙神录stage 1的背景。

……虽然这么说可是还是有些难，而数据文件里已经事先放进了拍成了快照的背景（译者注：原作中为3D背景），因此让我们先让它滚动起来吧。（译者注：原文为スクロール，也就是scroll，这里的滚动……嘛，您还是自己想象吧。）

所谓背景的滚动，就是这样一个循环：让1张图像在上下连续地绘制2次制然后滚动，且循环的周期要和图像的尺寸进行周期性吻合。

---- graph\_back.cpp 变更 ----

#include "../include/GV.h"

void graph\_back00(){

SetDrawArea( 32 , 16 , 416 , 464 ) ;//设定可以绘制的区域

DrawGraph(FIELD\_X,count%700+FIELD\_Y-700,img\_back[0],FALSE);

DrawGraph(FIELD\_X,count%700+FIELD\_Y ,img\_back[0],FALSE);

SetDrawArea( 0, 0, 640, 480);//重置区域

}

void graph\_back\_main(){

graph\_back00();

}

首先，由于有很大部分背景是被挤出画面的，因此我们没有必要计算这些不需要的区域，因此我们只把可描绘的区域设定为画面内。

然后，我们把1张图片分别在2个位置错开着绘制（总共绘制2次），以同样的速度往下滚动。

这里我们使用一个计数器count，它每一帧增加1。

嘛，这次仅仅是让照片滚动而已，因此很简单。

----在 graph.cpp 的 graph\_main 函数中进行以下追加 ----

extern void graph\_back\_main();

void graph\_main(){

graph\_back\_main();

----在 GV.h 中进行以下追加 ----

GLOBAL int img\_back[10];//背景用图片

GLOBAL int func\_state,stage\_count,count;

----在 load.cpp 的 load函数中进行以下追加 ----

img\_back[0] = LoadGraph("../dat/img/back/0/back0.png");

----在 main.cpp 的main函数中进行以下追加 ----

case 100://通常处理

calc\_ch(); //角色计算

ch\_move(); //控制角色的移动

cshot\_main();//自机射击main

enemy\_main();//敌人处理main

shot\_main(); //射击main

out\_main(); //碰撞计算

effect\_main();//特效main

graph\_main();//绘制main

stage\_count++;

break;

----在 main.cpp 中进行以下追加 ----

music\_play();

if(CheckStateKey(KEY\_INPUT\_ESCAPE)==1)break;//如果按下ESC键则跳出循环

ScreenFlip();//里外画面翻转

count++;

}

DxLib\_End();//DX Library终结处理

return 0;

}

运行结果

## 第二十一章 为自机加上碰撞判定和无敌处理吧

之前我们已经为敌人增加了碰撞判定，现在我们也为自机加上碰撞判定吧。

首先，在进入具体的处理之前，我们进行一些预处理。

如果要对子弹进行判定处理，我们必须要让子弹自身保有判定范围之类的信息。

因此，我们像下面那样做。

----在 struct.h 中进行以下追加 ----

//子弹的信息

typedef struct{

int size\_x,size\_y,col\_num;

double range;

}bullet\_info\_t;

----在 GV.h 中进行以下追加 ----

GLOBAL bullet\_info\_t bullet\_info[10];//子弹的信息

----在 load.cpp load函数中进行以下追加 ----

sound\_se[3]=LoadSoundMem("../dat/se/char\_death.wav");//自机被击毁的声音

ChangeVolumeSoundMem( 80, sound\_se[3] ) ;

----在 ini.cpp 中进行以下追加 ----

//将传入的信息放入结构体的函数

void input\_bullet\_info(bullet\_info\_t \*binfo,int size\_x,int size\_y,int col\_num,double range){

binfo->size\_x =size\_x; binfo->size\_y =size\_y;

binfo->col\_num=col\_num; binfo->range =range;

}

----在 ini.cpp first\_ini()函数中进行以下追加 ----

//比如：0号子弹，大小76x76像素，有五种颜色，碰撞范围为17像素

input\_bullet\_info(&bullet\_info[0],76, 76, 5,17.0);

input\_bullet\_info(&bullet\_info[1],22, 22, 6, 4.0);

input\_bullet\_info(&bullet\_info[2], 5,120,10, 2.5);

input\_bullet\_info(&bullet\_info[3],19, 34, 5, 2.0);

input\_bullet\_info(&bullet\_info[4],38, 38,10, 2.0);

input\_bullet\_info(&bullet\_info[5],14, 16, 3, 3.5);

input\_bullet\_info(&bullet\_info[6],14, 18, 3, 2.0);

input\_bullet\_info(&bullet\_info[7],16, 16, 9, 2.5);

input\_bullet\_info(&bullet\_info[8],12, 18,10, 1.5);

input\_bullet\_info(&bullet\_info[9],13, 19, 3, 2.0);

first\_ini函数保存着子弹的信息，我为了把这个碰撞判定的范围调整到合适的大小费了不少时间……

这个范围我还是不能接受！……如果您还是这样觉得话，那就自己反复测试然后调整吧。

在这个函数中，保存着子弹的大小、颜色的种类数、碰撞判定范围。

接下来，准备已经做好了，现在开始进入正式的处理了。

……虽然这么说，在上一章节中，我们已经做好了判定的模块，因此这次我们也同样地进行判定就行了，相当简单。

全部复制粘贴过来就OK了。

虽然把几个看起来都差不多的函数一个个地写出来感觉很浪费时间，不过这里要是泛用化的话稍微有些不好确认，因此我们后面在讨论函数的泛用化问题，现在我们只需要一一复制然后稍微更改一下就行了。

基本上来看，敌人的射击和自机的射击的碰撞判定处理是一样的。

---- out.cpp 变更 ----

#include "../include/GV.h"

#define ENEMY\_RANGE\_MAX 4

#define CSHOT\_RANGE\_MAX 2

#define CRANGE 2.0

//敌人的碰撞判定范围

int enemy\_range[ENEMY\_RANGE\_MAX]={16,30,16,50};

//自机射击的子弹的碰撞范围

int cshot\_range[CSHOT\_RANGE\_MAX]={6,};

//判定敌人和自机射击的子弹是否相碰

int out\_judge\_cshot(int i,int s){

int j;

if(cshot[i].cnt>0){ //如果射击的子弹的轨道至少计算过1次

double x=cshot[i].x-enemy[s].x; //敌人和自机射击的子弹的距离

double y=cshot[i].y-enemy[s].y;

//防止溢出

if(cshot[i].knd>=CSHOT\_RANGE\_MAX || enemy[s].knd>=ENEMY\_RANGE\_MAX)

printfDx("out\_judge\_cshot溢出");

//敌人的碰撞判定和自机射击的子弹的碰撞判定的合计范围

double r=cshot\_range[cshot[i].knd]+enemy\_range[enemy[s].knd];

//如果有必要计算中间的话

if(cshot[i].spd>r){

//保存1帧之前的位置

double pre\_x=cshot[i].x+cos(cshot[i].angle+PI)\*cshot[i].spd;

double pre\_y=cshot[i].y+sin(cshot[i].angle+PI)\*cshot[i].spd;

double px,py;

for(j=0;j<cshot[i].spd/r;j++){//前进部分÷碰撞判定部分次循环

px=pre\_x-enemy[s].x;

py=pre\_y-enemy[s].y;

if(px\*px+py\*py<r\*r)

return 1;

pre\_x+=cos(cshot[i].angle)\*r;

pre\_y+=sin(cshot[i].angle)\*r;

}

}

if(x\*x+y\*y<r\*r)//如果在碰撞判定范围内

return 1;//碰撞

}

return 0;

}

//判定自机和敌人射击的子弹是否碰上

int out\_judge\_enemyshot(int s,int n){

int j;

if(shot[s].bullet[n].cnt>0) //如果射击的子弹的轨道至少计算过1次

double x=shot[s].bullet[n].x-ch.x //敌人和自机射击的子弹的距离

double y=shot[s].bullet[n].y-ch.y;

//防止溢出

if(shot[s].bullet[n].knd>=10)

printfDx("out\_judge\_enemyshot内オーバーフロー\n");

//敌人射击的子弹和自机碰撞判定的合计范围

double r=bullet\_info[shot[s].bullet[n].knd].range+CRANGE;

//如果有必要计算中间的话

if(shot[s].bullet[n].spd>r){

//保存1帧前的位置

double pre\_x=shot[s].bullet[n].x+cos(shot[s].bullet[n].angle+PI)\*shot[s].bullet[n].spd;

double pre\_y=shot[s].bullet[n].y+sin(shot[s].bullet[n].angle+PI)\*shot[s].bullet[n].spd;

double px,py;

for(j=0;j<shot[s].bullet[n].spd/r;j++){//前进部分÷碰撞判定部分次循环

px=pre\_x-ch.x;

py=pre\_y-ch.y;

if(px\*px+py\*py<r\*r)

return 1;

pre\_x+=cos(shot[s].bullet[n].angle)\*r;

pre\_y+=sin(shot[s].bullet[n].angle)\*r;

}

}

if(x\*x+y\*y<r\*r)//如果在碰撞判定内的话

return 1;//碰撞

}

return 0;

}

extern void enter\_del\_effect(int);

//决定敌人是否击毁

void enemy\_death\_judge(int s){

int i;

se\_flag[8]=1;//敌人被击中的声音

if(enemy[s].hp<0){//如果敌人HP小于0

enemy[s].flag=0;/消灭敌人

se\_flag[1]=1;//敌人击落声

enter\_del\_effect(s);

for(i=0;i<SHOT\_MAX;i++){//敌人总数次循环

if(shot[i].flag!=0){//如果有已经登录的弹幕数据

if(s==shot[i].num){//并且那个敌人有登录了的弹幕

shot[i].flag=2;//将flag设置为不再继续

break;

}

}

}

}

}

//自机射击和敌人的处理

void cshot\_and\_enemy(){

int i,s;

for(i=0;i<CSHOT\_MAX;i++){//自机射击的子弹总数

if(cshot[i].flag>0){

for(s=0;s<ENEMY\_MAX;s++){//敌人总数

if(enemy[s].flag>0){

if(out\_judge\_cshot(i,s)){//如果自机撞上了敌人

cshot[i].flag=0;//将自机销毁

enemy[s].hp-=cshot[i].power;//敌人减少子弹所持有的power的量那么多的HP

enemy\_death\_judge(s);//决定敌人是否消灭

}

}

}

}

}

}

//敌人射击和自机的处理

void enemyshot\_and\_ch(){

int s,n;

for(s=0;s<SHOT\_MAX;s++){//敌人射击的子弹总数

if(shot[s].flag>0){//如果当前射击被登录了的话

for(n=0;n<SHOT\_BULLET\_MAX;n++){//子弹总数

if(shot[s].bullet[n].flag==1){//如果子弹被登录了的话

if(out\_judge\_enemyshot(s,n)){//且自己与这个子弹接触了

shot[s].bullet[n].flag=0;//销毁子弹

/\*在这里进行决死处理（译者注：决个毛蛋死，咱就喜欢抱B撞233333~）\*/

if(ch.flag==0 && ch.mutekicnt==0){状态为一般状态，且不是无敌状态下

ch.flag =2; //1：正在进行决死处理 2：已经被击毁且新的自机正在从下浮起来

ch.cnt =0;

se\_flag[3]=1;//击毁声

return;

}

}

}

}

}

}

}

//碰撞判定main

void out\_main(){

cshot\_and\_enemy();

enemyshot\_and\_ch();

}

enemyshot\_and\_ch是进行敌人射击和自机的碰撞判定处理的函数，

不过，击中之后为什么要ch.flag=2？

事实上中子弹之后角色会进入两个阶段的状态。

首先，中子弹的瞬间会有“正在进行决死处理”。将这个状态设置为ch.flag=1。

如果在规定时间内没有按下Boom键的话接下来就会进入从画面下方将新的自机浮上来的状态。

将这个状态设置为ch.flag=2。

浮上来进入画面内的时候，如果进行了按键输入的话，就将状态还原。

其次在角色死掉的顺江让角色进入无敌状态。

而是否是无敌状态的flag是另外一个标记。

ch.mutekicnt如果不为0表示是无敌的。这个时候不进行中子弹处理。

其次，在graph函数内绘制角色的时候，每2帧绘制1次。

这样一来自机会一闪一闪地，给人的无敌的感觉。

无敌状态在适当的时间之后消失。

现在我们就进行上述处理的变更/追加处理吧。

----在 char.cpp 中变更以下函数 ----

void calc\_ch(){

if(ch.cnt==0 && ch.flag==2){//如果当前瞬间死掉的话

ch.x=FIELD\_MAX\_X/2;//设置坐标

ch.y=FIELD\_MAX\_Y+30;

ch.mutekicnt++;//进入无敌状态

}

if(ch.flag==2){//如果已经死掉正在上浮的话

unsigned int push=CheckStatePad(configpad.left)+CheckStatePad(configpad.right)

+CheckStatePad(configpad.up)+CheckStatePad(configpad.down);

ch.y-=1.5;//将角色往上移动

//1秒以上，或者角色已经到达某个的位置且按下了某个键

if(ch.cnt>60 || (ch.y<FIELD\_MAX\_Y-20 && push)){

ch.cnt=0;

ch.flag=0;//将角色的状态还原

}

}

if(ch.mutekicnt>0){//如果无敌状态不为0的话

ch.mutekicnt++;

if(ch.mutekicnt>120)//如果已经2秒以上的话

ch.mutekicnt=0;//还原

}

ch.cnt++;//角色计数器自增

ch.img=(ch.cnt%24)/6;//决定当前的图像

}

----在 graph.cpp 的以下函数中变更 ----

//自机绘制

void graph\_ch(){

if(ch.mutekicnt%2==0)

DrawRotaGraphF(ch.x+FIELD\_X,ch.y+FIELD\_Y,1.0f,0.0f,img\_ch[0][ch.img],TRUE);

}

运行结果

注：由于录制的FPS的原因，在上面的动画中并没有很好地展示角色亮灭的效果。这个动画和实际的效果不太一样。

请自己编译后运行确认结果。

## 第二十二章 让自机放出Boom吧

本章我们讨论Boom。

我们再次和之前一样，先制作登录信息数据库，然后在数据库里面登录数据，之后让其自动计算。

在前面的章节中，我们准备了用于“特效”执行的变量。

之前我们将其用于敌人的消灭特效，这次我们试着在同一个数据库里面放入Boom的特效吧。

之前也说过了，在这个特效登录函数里面登录什么都是可以的。

只要我们往特效的识别编号knd里面放入决定好了的编号，然后和之前一样计算的话，我们就可以使用同一个数据库了。

在龙神录的特效里面，有“biu……咚”的特效（这个是特效编号1），以及特效线上下左右移动的特效（这个是特效编号2），还有角色突然往上移动的特效。（这个是特效编号3）

登录Boom的时候，用这些特效编号分别登录，然后试着实现它们吧。

首先做以下准备。

----在 struct.h 中进行以下追加 ----

//特效

typedef struct{

int flag,cnt,col,knd,img,eff;

double x,y,r,ang,spd,mvang,brt;

}effect\_t;

//Boom的信息

typedef struct{

int flag,cnt,knd;

double x,y;

}bom\_t;

//画面亮度的信息

typedef struct{

unsigned char brt;

}bright\_set\_t;

//咚！地晃动画面的信息

typedef struct{

int flag,cnt,time,size;

int x,y;

}dn\_t;

----在 GV.h 中进行以下追加 ----

GLOBAL int img\_eff\_bom[5]; //Boom的特效图像

GLOBAL bom\_t bom; //Boom

GLOBAL bright\_set\_t bright\_set; //绘制的亮度

GLOBAL dn\_t dn; //咚地摇动

----在 load.cpp load函数内进行以下追加 ----

img\_eff\_bom[0] = LoadGraph( "../dat/img/effect/bom0.png" );

img\_eff\_bom[1] = LoadGraph( "../dat/img/effect/bom1.png" );

img\_eff\_bom[2] = LoadGraph( "../dat/img/char/body.png" );

img\_eff\_bom[3] = LoadGraph( "../dat/img/bullet/bom\_title0.png" );

sound\_se[14]=LoadSoundMem("../dat/se/bom0.wav");

sound\_se[15]=LoadSoundMem("../dat/se/bom1.wav");

----在 ini.cpp 的 ini 函数中进行以下追加 ----

memset(&bom,0,sizeof(bom\_t));

memset(&bright\_set,0,sizeof(bright\_set\_t));

memset(&dn,0,sizeof(dn\_t));

bright\_set.brt=255;

----在 out.cpp 的out\_main函数中进行以下变更 ----

//碰撞判定main

void out\_main(){

cbom\_and\_enemy();

cshot\_and\_enemy();

enemyshot\_and\_ch();

}

----在 out.cpp 的任意地方进行以下变更 ----

//让s编号的敌人承受power的伤害

void hit\_enemy(int s,int power){

enemy[s].hp-=power;//减少子弹所有的power量的HP

enemy\_death\_judge(s);//判断敌人是否击毁

}

//Boom和敌人的处理

void cbom\_and\_enemy(){

int s;

if(bom.flag!=1)return;

for(s=0;s<ENEMY\_MAX;s++){//敌人总数

if(enemy[s].flag>0)

hit\_enemy(s,ch.power/20);//让s号敌人承受ch.power/20程度的伤害

}

}

//自机射击和敌人的处理

void cshot\_and\_enemy(){

int i,s;

for(i=0;i<CSHOT\_MAX;i++){//自机射击的子弹的总数

if(cshot[i].flag>0){

for(s=0;s<ENEMY\_MAX;s++){//敌人总数

if(enemy[s].flag>0){

if(out\_judge\_cshot(i,s)){//如果自机射击的子弹与敌人碰上

cshot[i].flag=0;//将子弹销毁

hit\_enemy(s,cshot[i].power);//给予编号为s的敌人power程度的伤害

}

}

}

}

}

}

----在 graph.cpp 的 graph\_effect中追加红字部分 ----

void graph\_effect(int knd){

for(int i=0;i<EFFECT\_MAX;i++){

if(effect[i].flag>0 && effect[i].knd==knd){

if(effect[i].eff==1)//特效是闪光特效的情况下

SetDrawBlendMode( DX\_BLENDMODE\_ADD, effect[i].brt) ;

if(effect[i].eff==2)//特效是α特效的情况下

SetDrawBlendMode(DX\_BLENDMODE\_ALPHA, effect[i].brt) ;

DrawRotaGraphF(effect[i].x+FIELD\_X+dn.x,effect[i].y+FIELD\_Y+dn.y,effect[i].r,effect[i].ang,effect[i].img,TRUE);

if(effect[i].eff==1 || effect[i].eff==2)

SetDrawBlendMode( DX\_BLENDMODE\_NOBLEND, 0) ;

}

}

}

----在 graph.cpp 的 graph\_main 中进行变更 ----

void graph\_main(){

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_back\_main();//背景绘制main

graph\_effect(0);//敌人击毁的特效

graph\_enemy();//敌人的描绘

graph\_cshot();//自机射击的描绘

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_ch();//自机的描绘

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_bullet();//子弹的描绘

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_effect(1);// Boom特效

graph\_effect(2);// Boom线特效

graph\_effect(3);// Boom角色特效

graph\_board();//面板绘制

}

这个章节中我们往结构体中新加入这样三个成员：

・保存Boom信息的bom\_t

・设定画面亮度的bright\_set\_t

・咚！ 地摇晃画面的信息dn\_t

用bright\_set\_t保存的变量通常是255，如果有必要降低亮度的话再修改它即可。

用dn\_t保存的变量用和之前一样的方法，在想要登录、计算、摇动画面的时候使用。

Boom的变量的使用方法和前面一样。

我们已经完成了预先准备。那么我们试着开始特效的计算吧。

---- effect.cpp 变更 ----

#include "../include/GV.h"

//搜索登录为空的特效编号

int search\_effect(){

for(int i=0;i<EFFECT\_MAX;i++){

if(effect[i].flag==0)

return i;

}

return -1;

}

//计算特效

void calc\_effect(){

for(int i=0;i<EFFECT\_MAX;i++){

if(effect[i].flag>0){//如果特效已经被登录了

switch(effect[i].knd){//按照特效的种类进行分歧

case 0://敌人的消灭特效

effect[i].cnt++;

effect[i].r+=0.08;//特效的大小逐渐变大

if(effect[i].cnt>10)//如果计数超过了10

effect[i].brt-=25;//将亮度变暗

if(effect[i].cnt>20)//如果计数超过了20

effect[i].flag=0;//销毁

break;

case 1:// Boom的特效

//坐标计算

effect[i].x+=cos(effect[i].mvang)\*effect[i].spd;

effect[i].y+=sin(effect[i].mvang)\*effect[i].spd;

//速度计算

if(effect[i].cnt<60)

effect[i].spd-=(0.2+effect[i].cnt\*effect[i].cnt/3000.0);

if(effect[i].cnt==60){

effect[i].spd=0;

se\_flag[15]=1;

dn.flag=1;

dn.cnt=0;

dn.size=11;

dn.time=20;

}

//亮度和大小的计算

effect[i].r+=0.015;

if(effect[i].cnt<51)

effect[i].brt+=5;

if(effect[i].cnt>=60){

effect[i].r+=0.04;

effect[i].brt-=255/30.0;

}

//计数器自增以及消去计算

effect[i].cnt++;

if(effect[i].cnt>=90)

effect[i].flag=0;

break;

case 2:// Boom特效（角色）

//座標計算 坐标计算

effect[i].x+=cos(effect[i].mvang)\*effect[i].spd;

effect[i].y+=sin(effect[i].mvang)\*effect[i].spd;

//亮度计算

if(effect[i].cnt<51)

effect[i].brt+=4;

if(effect[i].cnt>130-51)

effect[i].brt-=4;

//计数器自增及消去计算

effect[i].cnt++;

if(effect[i].cnt>=130)

effect[i].flag=0;

break;

case 3://Boom的特效（线）

//坐标计算

effect[i].x+=cos(effect[i].mvang)\*effect[i].spd;

effect[i].y+=sin(effect[i].mvang)\*effect[i].spd;

//亮度计算

if(effect[i].cnt<51)

effect[i].brt+=2;

if(effect[i].cnt>130-51)

effect[i].brt-=2;

//计数器自增与消去计算

effect[i].cnt++;

if(effect[i].cnt>=130)

effect[i].flag=0;

break;

default:

printfDx("effect设定错误\n");

break;

}

}

}

}

//搜索登录为空的消灭特效的编号

int search\_del\_effect(){

for(int i=0;i<DEL\_EFFECT\_MAX;i++){

if(del\_effect[i].flag==0)

return i;

}

return -1;

}

//登录消灭特效

void enter\_del\_effect(int s){

int k=search\_del\_effect();

if(k!=-1){

del\_effect[k].flag=1;//设置flag为有效

del\_effect[k].cnt=0;

del\_effect[k].col=enemy[s].back\_col;//将敌人的背景色设定为消灭色

del\_effect[k].x=enemy[s].x;//将敌人的坐标设置为消灭位置

del\_effect[k].y=enemy[s].y;

}

}

//计算消灭特效、登录特效

void calc\_del\_effect(){

int k;

for(int i=0;i<DEL\_EFFECT\_MAX;i++){

if(del\_effect[i].flag>0){

if(del\_effect[i].cnt%2==0){//每1/30秒设定一次

if((k=search\_effect())!=-1){//搜索空编号

memset(&effect[k],0,sizeof(effect\_t));//初始化

effect[k].flag=1;//设置flag为有效

effect[k].brt=255;//亮度设置为最大值

effect[k].ang=rang(PI);//随机角度

effect[k].col=del\_effect[i].col;//设置颜色

effect[k].eff=1;// 1是闪光特效。闪光特效为1.

effect[k].img=img\_del\_effect[effect[k].col];//设置消灭图像

effect[k].knd=0;//特效的种类是0号（消灭特效为0）

effect[k].x=del\_effect[i].x;//设置坐标

effect[k].y=del\_effect[i].y;

effect[k].spd=0;

}

}

if(del\_effect[i].cnt>8){

del\_effect[i].flag=0;

}

del\_effect[i].cnt++;

}

}

}

//登录Boom

void enter\_bom(){

int k;

bom.flag=1;

bom.cnt=0;

bom.knd=0;

ch.mutekicnt=1;//设置为无敌

se\_flag[14]=1;//播放确定声（译者注：原文为キュイーン音，至于这具体指啥声音，我也不太清楚……）

//竖线

if((k=search\_effect())!=-1){

effect[k].flag=1;

effect[k].cnt=0;

effect[k].knd=3;// Boom的线的特效

effect[k].brt=0;//α值

effect[k].ang=PI/2;//图像的方向

effect[k].mvang=-PI/2;//特效前进的方向

effect[k].spd=1;//速度

effect[k].r=1;//大小

effect[k].eff=2;//α混合

effect[k].img=img\_eff\_bom[3];

effect[k].x=70;//坐标

effect[k].y=300;

}

//横线

if((k=search\_effect())!=-1){

effect[k].flag=1;

effect[k].cnt=0;

effect[k].knd=3;// Boom的线的特效

effect[k].brt=0;

effect[k].ang=0;

effect[k].mvang=0;

effect[k].spd=1;

effect[k].r=1;

effect[k].eff=2;

effect[k].img=img\_eff\_bom[3];

effect[k].x=100;

effect[k].y=350;

}

//角色

if((k=search\_effect())!=-1){

effect[k].flag=1;

effect[k].cnt=0;

effect[k].knd=2;//Boom的角色、线的特效

effect[k].brt=0;

effect[k].ang=0;

effect[k].mvang=-PI/2;

effect[k].spd=0.7;

effect[k].r=1;

effect[k].eff=2;

effect[k].img=img\_eff\_bom[2];

effect[k].x=260;

effect[k].y=300;

}

}

//Boom计算

void bom\_calc(){

int n,k,shot\_angle[4]={0,PI,PI/2,PI\*1.5};//4发特效飞行的角度

if(ch.flag==0 && bom.flag==0){//角色如果是通常状态且没有放Boom

if(CheckStatePad(configpad.bom)==1){//如果Boom键被按下

enter\_bom();

}

}

if(bom.flag==1){//如果Boom已经被登录

if(bom.cnt%10==0){//1/6秒一次

n=(bom.cnt/10);

if(n<4 && (k=search\_effect())!=-1){

effect[k].flag=1;

effect[k].cnt=0;

effect[k].knd=1;//特效编号1：Boom的特效

effect[k].brt=0;

effect[k].ang=rang(PI);//随机的图像方向

effect[k].mvang=shot\_angle[n]-PI/4;//飞行方向

effect[k].spd=13+rang(2);//速度

effect[k].r=0.5;//大小

effect[k].eff=2;//α混合

effect[k].img=img\_eff\_bom[(bom.cnt/10)/3];//图像

effect[k].x=ch.x;//坐标

effect[k].y=ch.y;

}

}

bom.cnt++;

if(bom.cnt<40)

bright\_set.brt=255-bom.cnt\*5;//设定画面的亮度（暗）

if(bom.cnt>90)

bright\_set.brt=255-40\*5+(bom.cnt-90)\*5;//设定画面的的亮度（亮）

if(bom.cnt>130){

bom.flag=0;

bright\_set.brt=255;

}

}

}

//咚梆地摇动画面的处理

void dn\_calc(){

if(dn.flag==1){

dn.x=(int)rang(dn.size);

dn.y=(int)rang(dn.size);

dn.cnt++;

if(dn.cnt>dn.time){//如果超过了预设的时间的话就结束

dn.flag=0;

dn.x=0;

dn.y=0;

}

}

}

void effect\_main(){

dn\_calc();//咚梆地摇晃画面的处理

calc\_del\_effect();//消灭特效的计算

calc\_effect();//特效的计算

bom\_calc();//Boom计算

}

在enter\_bom中登录Boom。登录了的Boom在130次计数中计算，进行特效的登录、画面的亮度的变更等处理。

在calc\_effect()函数中进行特效的计算，在本章中追加了的特效编号为1、2、3的处理。

运行结果

## 第二十三章 试着做出决死吧

既然我们已经做好了Boom，那么现在我们来做决死吧。

在这里，我们先复习一下角色变化的flag。

ch.flag

在0的时候为通常。

在1的时候为正在进行决死。

在2的时候为已经被击毁正在上浮的状态。

中子弹的瞬间，将flag设置为1，在为1的时间范围内，开始进行Boom的处理。

并不是特地要去要去追加Boom处理，在

if(ch.flag==0 || ch.flag == 1)

的时候只要能够登录Boom即可，需要修改的地方很少。

我们决死处理中会把画面变暗，然后显示出特效吧。

特效也和之前那样，在特效登录用的变量中进行登录。

我们试着让自机猛地把敌人被击毁时使用的图像发射出去吧。

首先做一些预先准备。

----在 char.cpp 的 calc\_ch函数中进行以下追加 ----

void calc\_ch(){

if(ch.flag==1){//如果正在进行决死处理

bright\_set.brt=80;//变暗

if(ch.cnt>20){//0.33秒中进行决死处理

ch.flag =2; //1：正在进行决死处理 2：被击毁正在上浮中

ch.cnt =0;

bright\_set.brt=255;

}

}

if(ch.cnt==0 && ch.flag==2){//如果在当前瞬间被击毁的话

ch.x=FIELD\_MAX\_X/2;//设置坐标

ch.y=FIELD\_MAX\_Y+30;

ch.mutekicnt++;//进入无敌状态

}

if(ch.flag==2){//如果被击毁正在上浮的话

unsigned int push=CheckStatePad(configpad.left)+CheckStatePad(configpad.right)

+CheckStatePad(configpad.up)+CheckStatePad(configpad.down);

ch.y-=1.5;//将角色往上移动

//1秒钟以上，或者角色已经到达某个位置并且按下某键

if(ch.cnt>60 || (ch.y<FIELD\_MAX\_Y-20 && push)){

ch.cnt=0;

ch.flag=0;//将角色状态还原

}

}

if(ch.mutekicnt>0){//如果无敌计数不为0的话

ch.mutekicnt++;

if(ch.mutekicnt>150)//在150以上

ch.mutekicnt=0;//还原

}

ch.cnt++;//角色计数器自增

ch.img=(ch.cnt%24)/6;//决定当前的图像

}

---- effect.cpp 中的以下函数变更 ----

//计算特效

void calc\_effect(){

for(int i=0;i<EFFECT\_MAX;i++){

if(effect[i].flag>0){//如果特效已经被登录了

switch(effect[i].knd){//按照特效的种类进行分歧

case 0://敌人的消灭特效

effect[i].cnt++;

effect[i].r+=0.08;//特效的大小逐渐变大

if(effect[i].cnt>10)//如果计数超过了10

effect[i].brt-=25;//将亮度变暗

if(effect[i].cnt>20)//如果计数超过了20

effect[i].flag=0;//销毁

break;

case 1:// Boom的特效

//坐标计算

effect[i].x+=cos(effect[i].mvang)\*effect[i].spd;

effect[i].y+=sin(effect[i].mvang)\*effect[i].spd;

//速度计算

if(effect[i].cnt<60)

effect[i].spd-=(0.2+effect[i].cnt\*effect[i].cnt/3000.0);

if(effect[i].cnt==60){

effect[i].spd=0;

se\_flag[15]=1;

dn.flag=1;

dn.cnt=0;

dn.size=11;

dn.time=20;

}

//亮度和大小的计算

effect[i].r+=0.015;

if(effect[i].cnt<51)

effect[i].brt+=5;

if(effect[i].cnt>=60){

effect[i].r+=0.04;

effect[i].brt-=255/30.0;

}

//计数器自增以及消去计算

effect[i].cnt++;

if(effect[i].cnt>=90)

effect[i].flag=0;

break;

case 2:// Boom特效（角色）

//座標計算 坐标计算

effect[i].x+=cos(effect[i].mvang)\*effect[i].spd;

effect[i].y+=sin(effect[i].mvang)\*effect[i].spd;

//明るさ計算 亮度计算

if(effect[i].cnt<51)

effect[i].brt+=4;

if(effect[i].cnt>130-51)

effect[i].brt-=4;

//计数器自增及消去计算

effect[i].cnt++;

if(effect[i].cnt>=130)

effect[i].flag=0;

break;

case 3://Boom的特效（线）

//坐标计算

effect[i].x+=cos(effect[i].mvang)\*effect[i].spd;

effect[i].y+=sin(effect[i].mvang)\*effect[i].spd;

//亮度计算

if(effect[i].cnt<51)

effect[i].brt+=2;

if(effect[i].cnt>130-51)

effect[i].brt-=2;

//计数器自增与消去计算

effect[i].cnt++;

if(effect[i].cnt>=130)

effect[i].flag=0;

break;

case 4://决死特效

//亮度计算

if(effect[i].cnt>=6)

effect[i].brt-=255/6;

effect[i].r+=0.12;

effect[i].cnt++;

if(effect[i].cnt>=12 || ch.flag!=1)

effect[i].flag=0;

break;

default:

printfDx("effect设定错误\n");

break;

}

}

}

}

//Boom计算

void bom\_calc(){

int n,k,shot\_angle[4]={0,PI,PI/2,PI\*1.5};//4发特效飞行的角度

if((ch.flag==0||ch.flag==1) && bom.flag==0){//角色如果是通常状态或者在决死状态，且没有放Boom

if(CheckStatePad(configpad.bom)==1){//如果Boom键被按下

enter\_bom();

}

}

if(bom.flag==1){//如果Boom已经被登录

if(bom.cnt%10==0){//1/6秒一次

n=(bom.cnt/10);

if(n<4 && (k=search\_effect())!=-1){

effect[k].flag=1;

effect[k].cnt=0;

effect[k].knd=1;//特效编号1：Boom的特效

effect[k].brt=0;

effect[k].ang=rang(PI);//随机的图像方向

effect[k].mvang=shot\_angle[n]-PI/4;//飞行方向

effect[k].spd=13+rang(2);//速度

effect[k].r=0.5;//大小

effect[k].eff=2;//α混合

effect[k].img=img\_eff\_bom[(bom.cnt/10)/3];//图像

effect[k].x=ch.x;//坐标

effect[k].y=ch.y;

}

}

bom.cnt++;

if(bom.cnt<40)

bright\_set.brt=255-bom.cnt\*5;//设定画面的亮度（暗）

if(bom.cnt>90)

bright\_set.brt=255-40\*5+(bom.cnt-90)\*5;//设定画面的的亮度（亮）

if(bom.cnt>130){

bom.flag=0;

bright\_set.brt=255;

}

}

}

void enter\_crybom\_effect(){

int k;

if(ch.flag==1){

if(ch.cnt%2==0){

if((k=search\_effect())!=-1){

effect[k].flag=1;

effect[k].cnt=0;

effect[k].knd=4;/决死特效

effect[k].brt=255;

effect[k].ang=rang(PI);

effect[k].spd=0;

effect[k].r=0;

effect[k].eff=2;

effect[k].img=img\_del\_effect[GetRand(4)];

effect[k].x=ch.x;

effect[k].y=ch.y;

}

}

}

}

void effect\_main(){

dn\_calc();//咚梆地摇动画面的计算

calc\_del\_effect();//消灭特效计算

calc\_effect();//特效计算

bom\_calc();//Boom计算

enter\_crybom\_effect();

}

---- out.cpp enemyshot\_and\_ch函数----

//敌人射击和自机的处理

void enemyshot\_and\_ch(){

int s,n;

for(s=0;s<SHOT\_MAX;s++){//敌人射击的子弹总数

if(shot[s].flag>0){//如果射击登录了

for(n=0;n<SHOT\_BULLET\_MAX;n++){//子弹总数

if(shot[s].bullet[n].flag==1){//如果子弹已经登录

if(out\_judge\_enemyshot(s,n)==1){//且已经与自机接触

shot[s].bullet[n].flag=0;//销毁子弹

if(ch.flag==0 && ch.mutekicnt==0){

ch.cnt=0;

ch.flag=1;

se\_flag[3]=1;

}

}

}

}

}

}

}

---- graph.cpp 的 graph\_main函数变更----

void graph\_main(){

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_back\_main();//背景描绘main

graph\_effect(0);//敌人击毁特效

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_effect(4);//决死特效

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_enemy();//敌人的描绘

graph\_cshot();//自机射击的描绘

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_ch();//自机的描绘

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_bullet();//子弹的描绘

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_effect(1);//Boom的特效

graph\_effect(2);//Boom线的特效

graph\_effect(3);//Boom角色的特效

graph\_board();//面板绘制

}

---- char.cpp 的 ch\_move函数变更 ----

void ch\_move(){

int i,sayu\_flag=0,joge\_flag=0;

double x,y,mx,my,naname=1;

double move\_x[4]={-4.0,4.0,0,0},move\_y[4]={0,0,4.0,-4.0};

int inputpad[4];

inputpad[0]=CheckStatePad(configpad.left); inputpad[1]=CheckStatePad(configpad.right);

inputpad[2]=CheckStatePad(configpad.down); inputpad[3]=CheckStatePad(configpad.up);

if(ch.flag==1)return;//如果正在进行决死处理就不能移动

没有说明

运行结果

## 第二十四章 来制作Boss吧

接下来，我们终于快要进入弹幕制作了。

不过要制作Boss的弹幕的话，还得先制作Boss才行。

要制作Boss的框架非常麻烦，大家一起努力吧。

不过在此之前，我们得先讨论一些物理的东西。

在龙神录中，Boss倏倏地在到处移动。

虽然希望Boss能够遵从“在这个时间内从这里开始到这里移动”这样规定的指示，但是只是在匀速运动中移动到达指定地点之后突然停止的运动看起来会很没意思。

我们想看到的是Boss先加速、再减速然后华丽地停下来，这样子漂亮的动作。

因此我们来研究一下能够处理“在这个时间内仅在这个距离内漂亮地移动”这样的指示的计算式吧。

如果您的数学不太好的话，下面的计算式可能会让您感觉厌烦，不过这是非常简单的计算，所以还是请看看吧。

在物理的力学中，匀加速运动有三大方程。

也即是在下面的方程的1~3，其中v是速度，a是加速度、y是距离、t是时间。

现在我们考虑这样一个过程：咚地发射出去，然后在指定的时间ty内在指定的距离距离y内减速，然后停下来。

以初速度v0飞出的物体在时间ty内前进到ymax停止的时候，由于最后速度变为0，因此往式①中带入，有。

然后我们带入③中，有

然后把⑤带入这个式子，有，

两边约去v0，有，

然后把⑥带入⑤中，有，

最后把⑥、⑦带入②中，有，

这样一来，我们就把式A导出来了。

ty是指定时间、ymax是移动的距离。现在只要把计数器的值传给t然后计算，就能够得到y的值了。

在水平分量和竖直分量分别计算的话就能够实现漂亮的运动了。

下面是我们对上面的式子的实现。

input\_phy(登录)

calc\_phy(计算)

请参看注释。

---- boss\_shot.cpp 变更 ----

#include "../include/GV.h"

#include "../include/func.h"

#define V0 10.0

int search\_boss\_shot(){//返回空编号

for(int i=0;i<BOSS\_BULLET\_MAX;i++){

if(boss\_shot.bullet[i].flag==0)

return i;

}

return -1;

}

double bossatan2(){//自机和敌人所成的夹角

return atan2(ch.y-boss.y,ch.x-boss.x);

}

//进行物理计算的登录（在指定时间t内回到固定位置）

void input\_phy(int t){//t=

附加在移动上的时间

doubleymax\_x,ymax\_y;if(t==0)t=1;

boss.phy.flag=1;//登录有效

boss.phy.cnt=0;//计数器初始化

boss.phy.set\_t=t;//设置附加移动时间

ymax\_x=boss.x-BOSS\_POS\_X;//想要移动的水平距离

boss.phy.v0x=2\*ymax\_x/t;//水平分量的初速度

boss.phy.ax =2\*ymax\_x/(t\*t);//水平分量的加速度

boss.phy.prex=boss.x;//初始x坐标

ymax\_y=boss.y-BOSS\_POS\_Y;//想要移动的竖直距离

boss.phy.v0y=2\*ymax\_y/t;//竖直分量的初速度

boss.phy.ay =2\*ymax\_y/(t\*t);//数值分量的加速度

boss.phy.prey=boss.y;//初始y坐标

}

//物理上的角色移动计算

void calc\_phy(){

double t=boss.phy.cnt;

boss.x=boss.phy.prex-((boss.phy.v0x\*t)-0.5\*boss.phy.ax\*t\*t);//计算当前应当所在的x坐标

boss.y=boss.phy.prey-((boss.phy.v0y\*t)-0.5\*boss.phy.ay\*t\*t);//计算当前应当所在的y坐标

boss.phy.cnt++;

if(boss.phy.cnt>=boss.phy.set\_t)//如果超过附加移动的时间的话

boss.phy.flag=0;//オフ 设置移动为无效

}

//计算Boss的弹幕

void boss\_shot\_calc(){

int i;

boss.endtime--;

if(boss.endtime<0)

boss.hp=0;

for(i=0;i<BOSS\_BULLET\_MAX;i++){

if(boss\_shot.bullet[i].flag>0){

boss\_shot.bullet[i].x+=cos(boss\_shot.bullet[i].angle)\*boss\_shot.bullet[i].spd;

boss\_shot.bullet[i].y+=sin(boss\_shot.bullet[i].angle)\*boss\_shot.bullet[i].spd;

boss\_shot.bullet[i].cnt++;

if(boss\_shot.bullet[i].cnt>boss\_shot.bullet[i].till){

if(boss\_shot.bullet[i].x<-50 || boss\_shot.bullet[i].x>FIELD\_MAX\_X+50 ||

boss\_shot.bullet[i].y<-50 || boss\_shot.bullet[i].y>FIELD\_MAX\_Y+50)

boss\_shot.bullet[i].flag=0;

}

}

}

boss\_shot.cnt++;

}

//设置弹幕

void enter\_boss\_shot(){

memset(&boss\_shot , 0, sizeof(boss\_shot\_t));//子弹信息初始化

boss\_shot.flag=1;

boss.wtime=0;//待机时间0

boss.state=2;//设置为弹幕中的状态

boss.hp=boss.set\_hp[boss.knd];//HP设定

boss.hp\_max=boss.hp;

}

//设置Boss

void enter\_boss(int num){

if(num==0){//中路Boss开始时的情况

memset(enemy,0,sizeof(enemy\_t)\*ENEMY\_MAX);//销毁杂兵

memset(shot,0,sizeof(shot\_t)\*SHOT\_MAX);//销毁弹幕

boss.x=FIELD\_MAX\_X/2;//设置Boss的初始坐标

boss.y=-30;

boss.knd=-1;//弹幕的种类

}

boss.flag=1;

boss.hagoromo=0;//是否扇状扩散的flag

boss.endtime=99\*60;//剩余时间

boss.state=1;//设置状态为待机中

boss.cnt=0;

boss.graph\_flag=0;//恢复绘制flag

boss.knd++;

boss.wtime=0;//初始化待机时间

memset(&boss\_shot,0,sizeof(boss\_shot\_t));//初始化Boss的弹幕信息

input\_phy(60);//附加上60次计数在物理计算中回到固定位置

}

//Boss的待机处理

void waitandenter(){

int t=140;

boss.wtime++;

if(boss.wtime>t) //如果待机140次计数的话设置弹幕

enter\_boss\_shot();

}

//Boss的弹幕main

void boss\_shot\_main(){

if(stage\_count==boss.appear\_count[0] && boss.flag==0)//如果到了开始时间的话

enter\_boss(0);//开始

if(boss.flag==0)//如果Boss还没有登录那么什么都不做

return;

if(boss.phy.flag==1)//如果物理移动计算有效

calc\_phy();//进行物理计算

if(boss.state==2 && (boss.hp<=0 || boss.endtime<=0)){//如果在弹幕中体力为0的话

enter\_boss(1);//进入下一次弹幕

}

if(boss.state==1){//弹幕之间的待机时间

waitandenter();

}

if(boss.state==2){//如果在弹幕中

boss\_shot\_bullet[boss.knd]();//进入弹幕函数

boss\_shot\_calc();//计算弹幕

}

boss.cnt++;

}

呀~Boss的控制真是麻烦啊。

首先，如果stage\_count这个游戏计数器的值和boss.appear\_count这个决定Boss是否出现的计数器的值相同的话，那么就让Boss出现。由于为Boss定义了boss这个变量，因此我们在这里将flag设置为有效。

我们让boss.flag=1。

boss.state在1的时候，这是弹幕之间的冷却时间。敌人处于待机状态。

boss.state在2的时候，正处于弹幕发射的时候。

向弹幕函数传入的方法就是之前在射击处理中用到的函数指针。

这次我们在最上面实现的

double bossatan2()

函数的作用是，调用整个函数就可以返回自机和敌人之间的角度，非常方便。在制作弹幕的时候一定要利用这个。

首先我们在ini函数中为这些项目传入合适的值：合适Boss出现、体力为多少等信息。

---- ini.cpp ini()变更 ----

//游戏的初始化

void ini(){

stage\_count=1;

memset(&ch,0,sizeof(ch\_t));

memset(enemy,0,sizeof(enemy\_t)\*ENEMY\_MAX);

memset(enemy\_order,0,sizeof(enemy\_order\_t)\*ENEMY\_ORDER\_MAX);

memset(shot,0,sizeof(shot\_t)\*SHOT\_MAX);

memset(cshot,0,sizeof(cshot\_t)\*CSHOT\_MAX);

memset(effect,0,sizeof(effect\_t)\*EFFECT\_MAX);

memset(del\_effect,0,sizeof(del\_effect\_t)\*DEL\_EFFECT\_MAX);

memset(&bom,0,sizeof(bom\_t));

memset(&bright\_set,0,sizeof(bright\_set\_t));

memset(&dn,0,sizeof(dn\_t));

memset(&boss,0,sizeof(boss\_t));

ch.x=FIELD\_MAX\_X/2;

ch.y=FIELD\_MAX\_Y\*3/4;

ch.power=500;

boss.appear\_count[0]=100;

for(int i=0;i<DANMAKU\_MAX;i++)

boss.set\_hp[i]=5000;

bright\_set.brt=255;

}

我们也来事先进行变量和结构体的定义。

----在 struct.h 中追加 ----

//和Boss射击相关的结构体

typedef struct{

//flag、种类、计数器、发射过来的敌人的编号、颜色

int flag,knd,cnt,num;

//基本角度、基本速度

double base\_angle[1],base\_spd[1];

bullet\_t bullet[BOSS\_BULLET\_MAX];

}boss\_shot\_t;

//用于物理计算的结构体

typedef struct{

int flag,cnt,set\_t;

double ax,v0x,ay,v0y,vx,vy,prex,prey;

}phy\_t;

//Boss的信息

typedef struct{

int flag,cnt,knd,wtime,state,endtime,hagoromo,graph\_flag;

int hp,hp\_max;

int appear\_count[2],set\_hp[DANMAKU\_MAX];

double x,y,ang,spd;

phy\_t phy;

}boss\_t;

----在 define.h 中追加 ----

//Boss的固定位置

#define BOSS\_POS\_X (FIELD\_MAX\_X/2)

#define BOSS\_POS\_Y 100.0

//Boss所有的子弹的最大数量

#define BOSS\_BULLET\_MAX 3000

//弹幕的最大数

#define DANMAKU\_MAX 50

----在 GV.h 中追加 ----

GLOBAL int img\_dot\_riria[8];//莉莉安的像素画（译者注：莉莉安是游戏里面的角色，大概吧，嗯。）

GLOBAL boss\_shot\_t boss\_shot;// Boss射击的信息

GLOBAL boss\_t boss; //Boss的信息

----在 main.cpp 的main函数中进行以下部分的变更----

case 100://通常处理

calc\_ch(); //角色计算

ch\_move(); //控制角色的移动

cshot\_main();//自机射击main

enemy\_main();//敌人处理main

shot\_main(); //射击main

boss\_shot\_main();

out\_main(); //碰撞计算

effect\_main();//特效main

graph\_main();//绘制main

if(boss.flag==0)

stage\_count++;

----在 load.cpp load()函数内追加 ----

LoadDivGraph( "../dat/img/char/riria.png" , 8 , 8 , 1 , 100 , 100 , img\_dot\_riria ) ;

----在 enemy.cpp enemy\_enter()函数内进行如下追加 ----

void enemy\_enter(){//登录、控制敌人的行动的函数

int i,j,t;

if(boss.flag!=0)return;

for(t=0;t<ENEMY\_ORDER\_MAX;t++){

----在 graph.cpp 中进行以下函数的变更、追加 ----

void graph\_boss(){

if(boss.flag==0)return;

DrawRotaGraphF(boss.x+FIELD\_X+dn.x,boss.y+FIELD\_Y+dn.y,1.0f,0.0f,img\_dot\_riria[0],TRUE);

}

//子弹的描绘

void graph\_bullet(){

int i,j;

SetDrawMode( DX\_DRAWMODE\_BILINEAR ) ;//线性插值绘制

for(i=0;i<SHOT\_MAX;i++){//敌人弹幕数次循环

if(shot[i].flag>0){//如果弹幕数据为有效

for(j=0;j<SHOT\_BULLET\_MAX;j++){//弹幕所有的子弹的最大数次循环

if(shot[i].bullet[j].flag!=0){//如果子弹数据为有效

if(shot[i].bullet[j].eff==1)

SetDrawBlendMode( DX\_BLENDMODE\_ADD, 255) ;

DrawRotaGraphF(

shot[i].bullet[j].x+FIELD\_X+dn.x, shot[i].bullet[j].y+FIELD\_Y+dn.y,

1.0, shot[i].bullet[j].angle+PI/2,

img\_bullet[shot[i].bullet[j].knd][shot[i].bullet[j].col],TRUE);

if(shot[i].bullet[j].eff==1)

SetDrawBlendMode( DX\_BLENDMODE\_NOBLEND, 0) ;

}

}

}

}

//Boss

if(boss\_shot.flag>0){//弹幕数据有效的话

for(j=0;j<BOSS\_BULLET\_MAX;j++){//弹幕所有的子弹的最大数次循环

if(boss\_shot.bullet[j].flag!=0){//子弹数据有效的话

if(boss\_shot.bullet[j].eff==1)

SetDrawBlendMode( DX\_BLENDMODE\_ADD, 255) ;

DrawRotaGraphF(

boss\_shot.bullet[j].x+FIELD\_X+dn.x, boss\_shot.bullet[j].y+FIELD\_Y+dn.y,

1.0, boss\_shot.bullet[j].angle+PI/2,

img\_bullet[boss\_shot.bullet[j].knd][boss\_shot.bullet[j].col],TRUE);

if(boss\_shot.bullet[j].eff==1)

SetDrawBlendMode( DX\_BLENDMODE\_NOBLEND, 0) ;

}

}

}

SetDrawMode(DX\_DRAWMODE\_NEAREST);//恢复绘制状态

}

void graph\_develop(){

DrawFormatString(0,0,GetColor(255,255,255),"%d",stage\_count);

}

void graph\_main(){

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_back\_main();//背景绘制main

graph\_effect(0);//敌人击毁特效

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_effect(4);//决死特效

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_boss();

graph\_enemy();//人的绘制

graph\_cshot();//自机射击的绘制

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_ch();//自机的绘制

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_bullet();//子弹的描绘

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_effect(1);//Boom的特效

graph\_effect(2);// Boom线的特效

graph\_effect(3);// Boom角色的特效

graph\_board();//面板的绘制

graph\_develop();

}

----在 function.h 中追加 ----

//boss\_shot.cpp

GLOBAL void boss\_shot\_main();

弹幕的函数和杂兵射击的时候一样，用函数指针保存。

本章中，include文件夹下追加一个func.h文件。

---- func.cpp 变更 ----

extern void boss\_shot\_bulletH000();

void (\*boss\_shot\_bullet[DANMAKU\_MAX])() =

{

boss\_shot\_bulletH000,

};

试着先准备一个弹幕。

---- boss\_shotH.cpp 变更 ----

#include "../include/GV.h"

extern int search\_boss\_shot();

extern double bossatan2();

void boss\_shot\_bulletH000(){

#define TM000 120

int i,k,t=boss\_shot.cnt%TM000;

double angle;

if(t<60 && t%10==0){

angle=bossatan2();

for(i=0;i<30;i++){if((k=search\_boss\_shot())!=-1){

boss\_shot.bullet[k].col = 0;

boss\_shot.bullet[k].x = boss.x;

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 8;

boss\_shot.bullet[k].angle = angle+PI2/30\*i;

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 3;

se\_flag[0]=1;

}

}

}

for(i=0;i<BOSS\_BULLET\_MAX;i++){

if(boss\_shot.bullet[i].flag>0){

}

}

}

虽然并没有加上碰撞判定，不过还是先看看弹幕吧。

运行结果

## 第二十五章 为Boss加上碰撞判定吧

现在我们为Boss加上碰撞判定，同时也为Boss的子弹加上碰撞判定吧。

同时在本章中我们也加上在前面一直忘记的角色的装饰。

接下来，前面我们已经用FILED\_MAX\_X来定义区域的宽度，不过写起来很麻烦（汗）

FIELD\_MAX\_X -> FMX

FIELD\_MAX\_Y -> FMY

FIELD\_X -> FX

FIELD\_Y -> FY

然后我们做了以上变更，在这里先事先说明一下。

接下来，我们说说碰撞判定的话题。

在前面的章节中，我们写了碰撞判定的函数。在那个时候我们说过我们后面来泛用化地写函数。

请和前面的函数对比起来看。

进行以下变更。

//射击的坐标：1 进行碰撞判定的物体：2

int out\_judge(double x1, double y1, double x2, double y2,

double range1, double range2, double spd1,double angle1){

int j;

double x=x1-x2;//敌人和自机射击的子弹的距离

double y=y1-y2;

//敌人的碰撞判定和自机双色机的碰撞判定的合计范围

double r=range1+range2;

//如果有必要计算中间部分

if(spd1>r){

//保存1帧以前的位置

double pre\_x=x1+cos(angle1+PI)\*spd1;

double pre\_y=y1+sin(angle1+PI)\*spd1;

double px,py;

for(j=0;j<spd1/r;j++){//前进部分÷碰撞判定部分次循环

px=pre\_x-x2;

py=pre\_y-y2;

if(px\*px+py\*py<r\*r)

return 1;

pre\_x+=cos(angle1)\*r;

pre\_y+=sin(angle1)\*r;

}

}

if(x\*x+y\*y<r\*r)//如果在碰撞判定内的话

return 1;//碰撞

return 0;

}

直到上个章节为止，我们要为每个变量都写一次函数，这次我们使这个函数能对任何变量都进行计算。

只需要变更要处理的参数的形式，算法和处理的内容完全没有变化。

这是射击的子弹和物体是否碰撞的判定函数，

x1，y1等有“1”的参数表示射击的信息

x2，y2等有“2”的参数表示物体的信息。

根据这个，我们在out文件中进行替换。

---- out.cpp 变更 ----

#include "../include/GV.h"

#define ENEMY\_RANGE\_MAX 4

#define CSHOT\_RANGE\_MAX 2

#define CRANGE 2.0

#define BRANGE 40.0

//敌人的碰撞判定范围

int enemy\_range[ENEMY\_RANGE\_MAX]={16,30,16,50};

//自机设计的碰撞判定范围

int cshot\_range[CSHOT\_RANGE\_MAX]={6,};

//碰撞判定

//射击的坐标：1 进行碰撞判定的物体：2

int out\_judge(double x1, double y1, double x2, double y2,

double range1, double range2, double spd1,double angle1){

int j;

double x=x1-x2;//敌人和自机射击的子弹的距离

double y=y1-y2;

//敌人的碰撞判定和自机设计的碰撞判定的合计范围

double r=range1+range2;

//如果有必要计算中间的话

if(spd1>r){

//记录1帧之前的位置

double pre\_x=x1+cos(angle1+PI)\*spd1;

double pre\_y=y1+sin(angle1+PI)\*spd1;

double px,py;

for(j=0;j<spd1/r;j++){//前进部分÷碰撞判定部分次循环

px=pre\_x-x2;

py=pre\_y-y2;

if(px\*px+py\*py<r\*r)

return 1;

pre\_x+=cos(angle1)\*r;

pre\_y+=sin(angle1)\*r;

}

}

if(x\*x+y\*y<r\*r)//如果在碰撞判定内

return 1;//碰撞

return 0;

}

//判断敌人和自机射击是否碰上

int out\_judge\_cshot(int i,int s){

if(cshot[i].cnt>0){ //射击的子弹的轨道如果至少计算过一次

if(out\_judge(cshot[i].x,cshot[i].y,enemy[s].x,enemy[s].y,

cshot\_range[cshot[i].knd],enemy\_range[enemy[s].knd],

cshot[i].spd,cshot[i].angle)){

return 1;

}

}

return 0;

}

//判定Boss是否和自机射击碰撞

int out\_judge\_cshot\_boss(int i){

if(cshot[i].cnt>0){//如果射击的子弹的轨道至少计算过一次

if(out\_judge(cshot[i].x,cshot[i].y,boss.x,boss.y,

cshot\_range[cshot[i].knd],BRANGE,cshot[i].spd,cshot[i].angle)){

return 1;

}

}

return 0;

}

//判定自机是否和敌人射击碰上

int out\_judge\_enemyshot(int s,int n){

if(shot[s].bullet[n].cnt>0){//如果射击的子弹的轨道至少计算过一次

if(out\_judge(

shot[s].bullet[n].x,shot[s].bullet[n].y,ch.x,ch.y,

bullet\_info[shot[s].bullet[n].knd].range,CRANGE,

shot[s].bullet[n].spd,shot[s].bullet[n].angle

)){

return 1;

}

}

return 0;

}

//是否自机是否和Boss射击碰上

int out\_judge\_bossshot(int n){

if(boss\_shot.bullet[n].cnt>0){//射击的子弹的轨道至少计算过一次的话

if(out\_judge(

boss\_shot.bullet[n].x,boss\_shot.bullet[n].y,ch.x,ch.y,

bullet\_info[boss\_shot.bullet[n].knd].range,CRANGE,

boss\_shot.bullet[n].spd,boss\_shot.bullet[n].angle

)){

return 1;

}

}

return 0;

}

extern void enter\_del\_effect(int);

//判定敌人是否击毁

void enemy\_death\_judge(int s){

int i;

se\_flag[8]=1;//碰上敌人的声音

if(enemy[s].hp<0){//敌人的HP小于0的话

enemy[s].flag=0;//消灭敌人

se\_flag[1]=1;//敌人的击毁声

enter\_del\_effect(s);

for(i=0;i<SHOT\_MAX;i++){//敌人总数次循环

if(shot[i].flag!=0){//如果有登录了的弹幕数据

if(s==shot[i].num){//如果敌人有已经登录的弹幕的话

shot[i].flag=2;//设置弹幕不再继续

break;

}

}

}

}

}

//让s号敌人承受power伤害

void hit\_enemy(int s,int power){

enemy[s].hp-=power;//减少子弹所持有的power量的HP

enemy\_death\_judge(s);//决定敌人是否击毁

}

//让Boss承受power伤害

void hit\_boss(int power){

boss.hp-=power;//让HP减少子弹所持有的power的量的

}

//自机射击和敌人的处理

void cshot\_and\_enemy(){

int i,s;

for(i=0;i<CSHOT\_MAX;i++){//自机射击的子弹总数

if(cshot[i].flag>0){

for(s=0;s<ENEMY\_MAX;s++){//敌人总数

if(enemy[s].flag>0){

if(out\_judge\_cshot(i,s)){//如果自机射击的子弹和敌人碰上

cshot[i].flag=0;//将自己子弹销毁

hit\_enemy(s,cshot[i].power);

break;

}

}

}

//Boss出现且其不描绘flag为无效且在射击中

if(boss.flag==1 && boss.graph\_flag==0 && boss.state==2){

if(out\_judge\_cshot\_boss(i)){

cshot[i].flag=0;

hit\_boss(cshot[i].power);

}

}

}

}

}

//敌人射击和自机的处理

void enemyshot\_and\_ch(){

int s,n;

//杂兵的射击

for(s=0;s<SHOT\_MAX;s++){//敌人的射击的子弹总数

if(shot[s].flag>0){//如果射击已经登录

for(n=0;n<SHOT\_BULLET\_MAX;n++){//子弹总数

if(shot[s].bullet[n].flag==1){//如果子弹已经登录的话

if(out\_judge\_enemyshot(s,n)==1){//且自机与子弹接触

shot[s].bullet[n].flag=0;//销毁子弹

if(ch.flag==0 && ch.mutekicnt==0){

ch.cnt=0;

ch.flag=1;

se\_flag[3]=1;

}

}

}

}

}

}

//Boss射击

if(boss\_shot.flag>0){//如果射击已经登录

for(n=0;n<BOSS\_BULLET\_MAX;n++){//子弹总数

if(boss\_shot.bullet[n].flag==1){//如果子弹已经登录

if(out\_judge\_bossshot(n)==1){//如果子弹已经和自机接触

boss\_shot.bullet[n].flag=0;//销毁子弹

if(ch.flag==0 && ch.mutekicnt==0){

ch.cnt=0;

ch.flag=1;

se\_flag[3]=1;

}

}

}

}

}

}

//让敌人承受Boom的伤害

void cbom\_and\_enemy(){

int s;

if(bom.flag!=1)return;

for(s=0;s<ENEMY\_MAX;s++){//敌人总数

if(enemy[s].flag>0)//如果敌人是存在的

hit\_enemy(s,ch.power/20);//给予伤害

}

//Boss存在且其不绘制flag为无效且在射击中

if(boss.flag==1 && boss.graph\_flag==0 && boss.state==2)

hit\_boss(ch.power/20);//承受伤害

}

//碰撞判定main

void out\_main(){

cbom\_and\_enemy();//使敌人承受Boom的伤害

cshot\_and\_enemy();//自机射击和敌人的处理

enemyshot\_and\_ch();//敌人射击和自机的处理

}

虽然代码有点长，不过基本上来看，Boss的处理只是放在和杂兵一样的处理里面，因此在构造完全相同。

其次，在龙神录中SC的弹幕的时候还会变化背景呢。

在什么弹幕的时候有什么样子的背景，为了设定这一点我们为Boss增加back\_knd这个变量。

首先，我们在ini中把所有的SC都准备好吧。

----在 ini.cpp ini()中进行变更、追加 ----

boss.appear\_count[0]=50;//中路Boss出现的时刻

for(int i=0;i<DANMAKU\_MAX;i++){//各自弹幕的HP

boss.set\_hp[i]=10000;

boss.hp\_max=10000;

}

for(int i=0;i<DANMAKU\_MAX;i++)//弹幕的各自背景种类

boss.back\_knd[i]=1;

变量和结构体也事先定义。

----在 struct.h 中追加 ----

//Boos的信息

typedef struct{

int flag,cnt,knd,wtime,state,endtime,hagoromo,graph\_flag;

int hp,hp\_max;

int appear\_count[2],set\_hp[DANMAKU\_MAX],back\_knd[DANMAKU\_MAX];

double x,y,ang,spd;

phy\_t phy;

}boss\_t;

----在 define.h 中进行以下变更 ----

//区域的尺寸

#define FMX 384

#define FMY 448

//区域 左上角坐标

#define FX 32

#define FY 16

----在 GV.h 中追加 ----

GLOBAL int img\_chetc[10]; //和角色相关的其它图像

GLOBAL int img\_etc[50];//其它图像

这样就在etc中定义了和角色的装饰、Boss的HP表示相关的图像。

我们将它们读取出来然后绘制。

----在 load.cpp 的load函数中追加 ----

img\_chetc[0] = LoadGraph( "../dat/img/char/atari.png" );//碰撞判定

img\_chetc[2] = LoadGraph( "../dat/img/char/ball.png" );//球

img\_etc[1] = LoadGraph( "../dat/img/enemy/hp.png" );

img\_etc[7] = LoadGraph( "../dat/img/enemy/hp\_boss.png" );//Boos 的HP

----在 graph.cpp中追加、变更 ----

//绘制Boss

void graph\_boss(){

int i;

if(boss.flag==0)return;

DrawRotaGraphF(boss.x+FX+dn.x,boss.y+FY+dn.y,1.0f,0.0f,img\_dot\_riria[0],TRUE);

if(boss.hp\_max==0){printfDx("graph\_boss的0%\n");return;}

for(i=0;i<FMX\*0.98\*boss.hp/boss.hp\_max;i++){

if(boss.back\_knd[boss.knd]==1)

DrawGraph(3+FX+i+dn.x,2+FY+dn.y,img\_etc[7],FALSE);

else

DrawGraph(3+FX+i+dn.x,2+FY+dn.y,img\_etc[1],FALSE);

}

}

//自机绘制

void graph\_ch(){

double sx,sy,ny=(sin(2.0\*PI\*(count%50)/50)\*3),ang=2.0\*PI\*(count%120)/120;

if(CheckStatePad(configpad.slow)>0)//如果低速移动中

sx=15,sy=15+ny;//拉近

else

sx=30,sy=30+ny;//往普通的位置移动

DrawRotaGraphF( ch.x-sx+FX, ch.y+sy+FY, 1.0f, ang, img\_chetc[2], TRUE );

DrawRotaGraphF( ch.x+sx+FX, ch.y+sy+FY, 1.0f, -ang, img\_chetc[2], TRUE );

if(ch.mutekicnt%2==0){//无敌中的话亮灭

//自机显示

DrawRotaGraphF(ch.x+FX+dn.x,ch.y+FY+dn.y,1.0f,0.0f,img\_ch[0][ch.img],TRUE);

if(CheckStatePad(configpad.slow)>0)//如果低速移动中的话显示出碰撞判定

DrawRotaGraphF( ch.x+FX, ch.y+FY, 1.0f, 2.0\*PI\*(count%120)/120, img\_chetc[0], TRUE );

}

}

现在来绘制SC用的背景。

boos.knd的指示编号boos.back\_knd在1的时候表示SC，这一章中我们全部置为1.

然后我们背景什么都不绘制，让它一片漆黑。

---- graph\_back.cpp 变更 ----

#include "../include/GV.h"

void graph\_back00(){//通常背景

SetDrawArea( 32 , 16 , 416 , 464 ) ;//设定绘制可能的区域

DrawGraph(FX,count%700+FY-700,img\_back[0],FALSE);

DrawGraph(FX,count%700+FY ,img\_back[0],FALSE);

SetDrawArea( 0, 0, 640, 480);//还原区域

}

void graph\_back01(){//SC用背景

SetDrawArea( 32 , 16 , 416 , 464 ) ;//设置绘制可能的区域

//在这里绘制SC用的背景

SetDrawArea( 0, 0, 640, 480);//还原区域

}

void graph\_back\_main(){

//Boos为有效且在放SC

if(boss.flag==1 && boss.back\_knd[boss.knd]==1)

graph\_back01();

else//除此之外

graph\_back00();

}

---- func.h 变更 ----

extern void boss\_shot\_bulletH000();

void (\*boss\_shot\_bullet[DANMAKU\_MAX])() =

{

boss\_shot\_bulletH000,

boss\_shot\_bulletH000,

boss\_shot\_bulletH000,

boss\_shot\_bulletH000,

boss\_shot\_bulletH000,

boss\_shot\_bulletH000,

boss\_shot\_bulletH000,

};

在func.h中写入弹幕情报。

现在我们只做了H000这个弹幕，后面在增加弹幕的时候，这个也会增加。

因此在这里加入的函数的顺序就是弹幕的顺序。

现在，在弹幕结束的时候从0号弹幕开始。

7次弹幕结束了的话，会发生指针错误请注意一下。

运行结果

## 第二十六章 为Boss加上特效吧

只是让Boos梆地射出弹幕的话一点也不好看，现在我们让Boos周围回旋起纹章，加上特效吧。

其次，在这一章节中，我们会稍微扩展一下敌人的移动控制函数。

Boss上下地慢慢移动的时候看起来就很漂亮了。

显示用的坐标为了与x、y区分使用dx、dy。我们往这两个变量中代入显示用的坐标。

接下来，我们读入必要的图像，然后绘制Boss的特效吧。

----在 boss\_shot.cpp 中进行以下追加 ----

void calc\_boss(){

boss.dx=boss.x;

boss.dy=boss.y+sin(PI2/130\*(count%130))\*10;

}

----在load函数中进行以下追加 ----

img\_etc[2] = LoadGraph( "../dat/img/enemy/bossback.png" );

img\_etc[5] = LoadGraph( "../dat/img/enemy/bossback3.png" );

img\_etc[6] = LoadGraph( "../dat/img/enemy/bossback4.png" );

----在graph.cpp 的以下部分进行修正

void graph\_boss\_effect(){

SetDrawBlendMode( DX\_BLENDMODE\_ALPHA, 150) ;

DrawRotaGraphF(boss.dx+FX, boss.dy+FY,

(0.4+0.05\*sin(PI2/360\*(count%360)))\*3,

2\*PI\*(count%580)/580,

img\_etc[5], TRUE);

DrawRotaGraphF(boss.dx+FX, boss.dy+FY,

(0.5+0.1\*sin(PI2/360\*(count%360)))\*2,

2\*PI\*(count%340)/340,img\_etc[2], TRUE);

DrawRotaGraphF(boss.dx+60\*sin(PI2/153\*(count%153))+FX,

boss.dy+80\*sin(PI2/120\*(count%120))+FY,

0.4+0.05\*sin(PI2/120\*(count%120)),

2\*PI\*(count%30)/30,img\_etc[6], TRUE);

DrawRotaGraphF(boss.dx+60\*sin(PI2/200\*((count+20)%200))+FX,

boss.dy+80\*sin(PI2/177\*((count+20)%177))+FY,

0.3+0.05\*sin(PI2/120\*(count%120)),

2\*PI\*(count%35)/35,img\_etc[6], TRUE);

SetDrawBlendMode( DX\_BLENDMODE\_NOBLEND, 0) ;

DrawRotaGraphF(boss.dx+60\*sin(PI2/230\*((count+40)%230))+FX,

boss.dy+80\*sin(PI2/189\*((count+40)%189))+FY,

0.6+0.05\*sin(PI2/120\*(count%120)),

2\*PI\*(count%40)/40,img\_etc[6], TRUE);

}

void graph\_boss(){

int i;

if(boss.flag==0)return;

graph\_boss\_effect();

DrawRotaGraphF(boss.dx+FX+dn.x,boss.dy+FY+dn.y,

1.0f,0.0f,img\_dot\_riria[0],TRUE);

其次，我们为前面章节中利用物理移动的方程式进行移动计算的函数加上后缀，以做出这样子的函数，它能够完成“从当前地点开始在多少计数下往离开多远的某个位置移动”这样子的指定。

----在 boss\_shot.cpp 中进行追加----

//进行与某点的指定距离的物理计算的登录（在指定时间t内回到固定位置）

void input\_phy\_pos(double x,double y,int t){//t=移動にかける時間 移动附加时间

double ymax\_x,ymax\_y;

if(t==0)t=1;

boss.phy.flag=1;//登录有效

boss.phy.cnt=0;//计时器初始化

boss.phy.set\_t=t;//设置移动附加时间

ymax\_x=boss.x-x;//想要移动的水平距离

boss.phy.v0x=2\*ymax\_x/t;//水平分量的初速度

boss.phy.ax =2\*ymax\_x/(t\*t);//水平分量的加速度

boss.phy.prex=boss.x;//初始x坐标

ymax\_y=boss.y-y;//想要移动的水平距离

boss.phy.v0y=2\*ymax\_y/t;//水平分量的初速度

boss.phy.ay =2\*ymax\_y/(t\*t);//水平分量的加速度

boss.phy.prey=boss.y;//初始y坐标

}

//在t次计数中从当前位置到离开dist这么远的位置移动

int move\_boss\_pos(double x1,double y1,double x2,double y2,double dist, int t){

int i=0;

double x,y,angle;

for(i=0;i<1000;i++){

x=boss.x,y=boss.y;//设置当前Boss的位置

angle=rang(PI);//適当地决定前进方向

x+=cos(angle)\*dist;//向着那个地方移动

y+=sin(angle)\*dist;

if(x1<=x&&x<=x2 && y1<=y&&y<=y2){//如果那个点在移动可能的范围内的话

input\_phy\_pos(x,y,t);

return 0;

}

}

return -1;//1000如果1000次尝试都不能的话就返回错误

}

---- boss\_shot.cpp 的 boss\_shotmain 中进行以下红字部分追加 ----

//

void boss\_shot\_main(){

if(stage\_count==boss.appear\_count[boss.knd] && boss.flag==0)//如果到了开始时间

enter\_boss(boss.knd);//开始

if(boss.flag==0)//如果Boss没有被登录则返回

return;

calc\_boss();

if(boss.phy.flag==1)//如果物理计算移动为有效

----在 boss\_shotH.cpp 的头部进行以下追加 ----

extern int move\_boss\_pos(double x1,double y1,double x2,double y2,double dist, int t);

----在 struct.h 中进行以下追加 ----

//Boss的信息

typedef struct{

int flag,cnt,knd,wtime,state,endtime,hagoromo,graph\_flag;

int hp,hp\_max;

int appear\_count[2],set\_hp[DANMAKU\_MAX],back\_knd[DANMAKU\_MAX];

double x,y,dx,dy,ang,spd;

phy\_t phy;

}boss\_t;

move\_boss\_pos函数中，Boos能够在x1以上x2以下、y1以上y2以下的范围内在t次计数内在离开dist那么远的位置。

适当地在各个地方决定其以合适的方向前进的目标位置。

要是1000次反复测试都不行的话那就放弃。

这个函数将从下个章节开始使用，运行结果如下所示。

运行结果

## 第二十七章 制作弹幕——“沉默的圣奈”

弹幕名：沉默的圣奈（サイレントセレナ）（Silence Seirai）

制作难易度：★　(1/10)

运行结果

第一回弹幕制作讲座我们将制作弹幕——“沉默的圣奈”。

这个弹幕为圆形散开连续发射的子弹和随机地往下掉落子弹组成，非常简单，因此实现起来也很轻松。

.base\_angle中保存每1回弹幕开始时自机与Boss的角度，以此为基准进行圆形发射。

程序的流程请参考注释。

----在boss\_shotH.cpp 中进行以下追加 ----

//沉默的圣奈

void boss\_shot\_bulletH001(){

#define TM001 60

int i,k,t=boss\_shot.cnt%TM001,t2=boss\_shot.cnt;

static int cnum;

double angle;

if(t2==0)//最开始的初始化

cnum=0;

if(t==0){//每1次弹幕最开始的初始化

boss\_shot.base\_angle[0]=bossatan2();//自机与Boss的角度

if(cnum%4==3){// 4次弹幕移动一次

move\_boss\_pos(40,30,FMX-40,120,60, 60);

}

}

//1次弹幕的最开始是自机狙，到了一半之后从自机狙错开

if(t==TM001/2-1)

boss\_shot.base\_angle[0]+=PI2/20/2;

//1次弹幕发射10次圆形子弹

if(t%(TM001/10)==0){

angle=bossatan2();//自机-Boss之间的角度

for(i=0;i<20;i++){//20个

if((k=search\_boss\_shot())!=-1){

boss\_shot.bullet[k].col = 4;

boss\_shot.bullet[k].x = boss.x;

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 8;

//从基本角度开始旋转20次并发射

boss\_shot.bullet[k].angle = boss\_shot.base\_angle[0]+PI2/20\*i;

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 2.7;//速度

se\_flag[0]=1;

}

}

}

//4次计数掉落一次的子弹的登录

if(t%4==0){

if((k=search\_boss\_shot())!=-1){

boss\_shot.bullet[k].col = 0;

boss\_shot.bullet[k].x = GetRand(FMX);

boss\_shot.bullet[k].y = GetRand(200);

boss\_shot.bullet[k].knd = 8;

boss\_shot.bullet[k].angle = PI/2;//直角

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 1+rang(0.5);

se\_flag[0]=1;

}

}

if(t==TM001-1)

cnum++;

}

也请确认变更了的func.h。

## 第二十八章 制作弹幕——“完美冻结”

弹幕名：完美冻结（パーフェクトフリーズ）（Perfect Freeze）

制作难易度：★★　(2/10)

运行结果

第二回弹幕制作讲座我们将制作弹幕——“完美冻结”。

本章中我们需要考虑的关键是随机数的生成。

请注意

rang(PI2/20)+PI2/10\*t

这个生成随机角度的地方。

如果是生成360°的随机方向的话调用rang(PI)就可以了呢。

由于rang(n)函数返回的是-n<=θ<=n的角度，那么使用这个函数我们应该就可以生成全方向的随机方向了。

但是，随机数是个可怕的东西，很有可能某个时候某个地点会出现聚集现象，也即无法产生“概率均等的随机数”。

于是，我们干脆来生成“假定随机数”吧。

现在，我们希望让0~99分别对应全方向的角度，然后生成0~99之间的随机数。

这个时候

第1帧中返回0~9之间的随机数

第2帧中返回10~19之间的随机数

第3帧中返回20~29之间的随机数

第4帧中返回30~39之间的随机数……

如此继续下去，一直到99，然后又回到0重新开始，进入这样一个循环。

这么一来，在10帧内必定可以往各不相同的10个方向发射。

即便是随机方向，由于10帧中在10个区分范围内进行随机方向的前进，因此不可能偶尔出现0~9中聚集现象。

这么一来，1帧1帧地区分角度的话就会循环起来，事实上虽然是在循环，但是由于是1/6秒循环一周，实际上看上去也就根本没有循环。完美冻结的随机发射如果采取这样的方式，您还会在意循环么？（译者注：这里我复述一下作者的意思。其实作者就是想说明10个随机的方向虽然是分别产生的，但是由于分别产生于10帧中，这个时间相当短，根本可以认为就是同时产生的，因此不用在意0~9这个变化过程，显示出来的时候看起来也就是同时往10个方向前进了。）

这么一来，我们就可以生成假想的随机数了。

由于这对于高质量的弹幕的制作而言相当重要，因此我们要防止在任何地方产生聚集性的随机数，深思熟虑过后再生成之。

完美冻结的算法中进行了如下流程：

分别向所有方向散射出子弹

冻结！！停止子弹的运动然后变白色。

连续向以自机所在位置为基准的八个方向

停止的子弹朝着全方向开始移动。

----在 boss\_shotH.cpp 中进行以下追加 ----

//完美冻结

void boss\_shot\_bulletH002(){

#define TM002 650

int i,k,t=boss\_shot.cnt%TM002;

double angle;

if(t==0 || t==210){

//在40<x<FMX-40 50<y<150的范围内80次计数中移动到100单位远的地方

move\_boss\_pos(40,50,FMX-40,150,100, 80);

}

//最开始的随机发射

if(t<180){

for(i=0;i<2;i++){//1次计数发射2次

if((k=search\_boss\_shot())!=-1){

boss\_shot.bullet[k].col = GetRand(6);//随机像是7种颜色

boss\_shot.bullet[k].x = boss.x;//发射初始坐标为Boss的坐标

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 7;//子弹的种类

boss\_shot.bullet[k].angle = rang(PI2/20)+PI2/10\*t;

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 3.2+rang(2.1);

boss\_shot.bullet[k].state = 0;//子弹的状态

boss\_shot.bullet[k].kaiten= 1;//决定子弹是否旋转的flag

}

}

if(t%10==0)

se\_flag[0]=1;

}

//根据自机的位置往8个方向发射

if(210<t && t<270 && t%3==0){

angle=bossatan2();

for(i=0;i<8;i++){

if((k=search\_boss\_shot())!=-1){

boss\_shot.bullet[k].col = 0;

boss\_shot.bullet[k].x = boss.x;

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 7;

//以自机与Boos所成的夹角为基准往8个方向发射

boss\_shot.bullet[k].angle = angle-PI/2\*0.8+PI\*0.8/7\*i+rang(PI/180);

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 3.0+rang(0.3);

boss\_shot.bullet[k].state = 2;

boss\_shot.bullet[k].kaiten= 1;

}

}

if(t%10==0)

se\_flag[0]=1;

}

for(i=0;i<BOSS\_BULLET\_MAX;i++){

if(boss\_shot.bullet[i].flag>0){

//t在190的时候将所有的子弹都停止下来，然后变白色，重置计数器

if(boss\_shot.bullet[i].state==0){

if(t==190){

boss\_shot.bullet[i].kaiten=0;//停止子弹的旋转

boss\_shot.bullet[i].spd =0;

boss\_shot.bullet[i].col =9;

boss\_shot.bullet[i].cnt =0;

boss\_shot.bullet[i].state =1;//将状态设置为1

}

}

//开始往随机方向移动

if(boss\_shot.bullet[i].state==1){

if(boss\_shot.bullet[i].cnt==200){

boss\_shot.bullet[i].angle=rang(PI);//全方向随机

boss\_shot.bullet[i].kaiten=1;//设置旋转flag为有效

}

if(boss\_shot.bullet[i].cnt>200)

boss\_shot.bullet[i].spd+=0.01;//逐渐加速

}

}

}

}

也请确认变更了的func.h。

## 第二十九章 制作弹幕——“恋之迷路”

弹幕名：恋之迷路（恋の迷路）

制作难易度：★★★★　(4/10)

运行结果

第三回弹幕制作讲座我们将制作弹幕——“恋之迷路”。

恋之迷路稍微有点麻烦。

让子弹一边旋转一边发射，并且在特定的部分不发射的处理相当重要。

旋转的方向由计数器决定。

使用计数器进行旋转方向的计数，当计数器到达1周次数-2或者1周次数-1的话那就不发射。

如果1周的计数器已经达到1周次数 -1的话，则接下来将会有1次计数错开而不发射。那么我们来试着实现吧。

（译者注：我这里重新解释一遍作者的意思。通过阅读源码可以发现，作者设置了三个静态变量cnt、tcnt和cnum，这3个变量在每次Boss开始发放弹幕的时候分别设置为0、2和0。其中作者设置了所谓的“1周”的计数为DF003——它的值为20，有兴趣的话您也可以把这个数字换成其它数字试试。

tcnt的大小用于固定“1周”的数值，它的设置条件为tcnt-1 == cnt的时候。

现在我们来考虑第1次弹幕发射的时候，这时tcnt-1 = 1并不等于cnt，所以tcnt没有被置数，这个时候Boss发射第一次弹幕，然后cnt++。然后，第2次进入弹幕发射的时候由于tcnt-1 = 1 = cnt了，那么tcnt被置数为tcnt += DF003-2，事实上这里就确定了允许发射的1周的最终一次为cnt变为cnt-2的时候，此时cnt-2=tcnt。可以认为是cnt从1变为18这个区间内都要发射弹幕，而当cnt=19或者20的时候就不发射了——这里也就产生了视频中的“空缺部分”，而当cnt=19的时候tcnt+=DF003-2又被执行，这样一来，相当于计数器被重置……如此循环也就形成了“有空缺”的弹幕。如果您还看不懂的话我也没辙了。）

----在 boss\_shotH.cpp 中进行以下追加 ----

void boss\_shot\_bulletH003(){

#define TM003 600

#define DF003 20

int i,j,k,t=boss\_shot.cnt%TM003,t2=boss\_shot.cnt;

static int tcnt,cnt,cnum;

double angle;

if(t2==0){

//在40<x<FMX-40 50<y<150的范围内80次计数中移动到100单位外的位置

input\_phy\_pos(FMX/2,FMY/2, 50);

cnum=0;

}

if(t==0){

boss\_shot.base\_angle[0]=bossatan2();

cnt=0;

tcnt=2;

}

if(t<540 && t%3){

angle=bossatan2();

if(tcnt-2==cnt || tcnt-1==cnt){

if(tcnt-1==cnt){

boss\_shot.base\_angle[1]=boss\_shot.base\_angle[0]+PI2/DF003\*cnt\*(cnum?-1:1)-PI2/(DF003\*6)\*3;

tcnt+=DF003-2;

}

}

else{

for(i=0;i<6;i++){

if((k=search\_boss\_shot())!=-1){

boss\_shot.bullet[k].col = cnum?1:4;

boss\_shot.bullet[k].x = boss.x;

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 8;

boss\_shot.bullet[k].angle

= boss\_shot.base\_angle[0]+PI2/DF003\*cnt\*(cnum?-1:1)+PI2/(DF003\*6)\*i\*(cnum?-1:1);

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 2;

se\_flag[0]=1;

}

}

}

cnt++;

}

if(40<t&&t<540 && t%30==0){

for(j=0;j<3;j++){

angle=boss\_shot.base\_angle[1]-PI2/36\*4;

for(i=0;i<27;i++){

if((k=search\_boss\_shot())!=-1){

boss\_shot.bullet[k].col = cnum?6:0;

boss\_shot.bullet[k].x = boss.x;

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 7;

boss\_shot.bullet[k].angle = angle;

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 4-1.6/3\*j;

se\_flag[0]=1;

}

angle-=PI2/36;

}

}

}

// for(i=0;i<BOSS\_BULLET\_MAX;i++){

// if(boss\_shot.bullet[i].flag>0){

//

// }

// }

if(t==TM003-1)

cnum++;

}

也请确认变更了的func.h。

## 第三十章 制作弹幕——小小青蛙不畏风雨

弾幕名：小小青蛙不畏风雨（ケロちゃん風雨に負けず）

（译者注：这是风神录EX诹访子的SC，Dixq这是多爱诹访子啊……）

制作难易度：★★★　(3/10)

运行结果

第四回弹幕制作讲座我们将制作弹幕——“小小青蛙不畏风雨”。

首先，如果仔细观察这个弹幕的话，可以发现这个弹幕往上方发射8路子弹。

只要让这8路子弹慢慢下落并加速就行了。

8路子弹的中心以PI/6为振幅左右摇摆。

为了让变化看起来美观，我们使用sin。由于sin能够漂亮地产生0→1→0→-1→0这样子的循环，所以我们来使用它吧。

angle=PI\*1.5+PI/6\*sin(PI2/tm\*t2);

以tm为基准，以递增的计数器t2来进行角度的变化。

由于角度的中心是正上方，所以我们设置为PI\*1.5.

振幅为PI/6。只要从把-1到1分别乘以这个值，那么就可以让最终值

PI\*1.5-PI/6~PI\*1.5+PI/6

之间变化。

只要以这个为中心射出8路子弹就行了。

这一次，如果使用angle来移动的话会很麻烦，因此我们利用x和y方向的速度来移动。

所以我们不使用spd。

vx是水平方向的速度而vy是竖直方向的速度。

这个计算在函数中进行。

此外，这样一来为了让显示的图像的角度不稳定（译者注：作者在这里的表述有些不清楚，这里我多说一句，其实这里就是让落下的子弹的旋转角度与速度的夹角角度相同，您可以把atan2函数改成一个定值看看效果），为了决定显示的图像的角度，对vx和vy使用atan2来计算角度。

----在 boss\_shotH.cpp 中进行以下追加 ----

void boss\_shot\_bulletH004(){

#define TM004 200

int i,j,k,n,t=boss\_shot.cnt%TM004,t2=boss\_shot.cnt;

static int tm;

double angle;

//周期的最开始设置tm

if(t==0)

tm=190+rang(30);

angle=PI\*1.5+PI/6\*sin(PI2/tm\*t2);

//每4次计数往8路射出子弹

if(t2%4==0){

for(n=0;n<8;n++){

if((k=search\_boss\_shot())!=-1){

boss\_shot.bullet[k].flag=1;

boss\_shot.bullet[k].state=0;

boss\_shot.bullet[k].cnt=0;

boss\_shot.bullet[k].knd=4;

boss\_shot.bullet[k].col=0;

boss\_shot.bullet[k].angle=0;

boss\_shot.bullet[k].eff\_detail=1;

boss\_shot.bullet[k].x=boss.x;

boss\_shot.bullet[k].y=boss.y;

boss\_shot.bullet[k].vx = cos(angle-PI/8\*4+PI/8\*n+PI/16)\*3;

boss\_shot.bullet[k].vy = sin(angle-PI/8\*4+PI/8\*n+PI/16)\*3;

}

}

se\_flag[0]=1;

}

if(t%1==0 && t2>80){

int num=1;

if(t%2)

num=2;

for(n=0;n<num;n++){

if((k=search\_boss\_shot())!=-1){

angle=PI\*1.5-PI/2+PI/12\*(t2%13)+rang(PI/15);

boss\_shot.bullet[k].flag=1;

boss\_shot.bullet[k].state=1;

boss\_shot.bullet[k].cnt=0;

boss\_shot.bullet[k].knd=8;

boss\_shot.bullet[k].eff\_detail=0;

boss\_shot.bullet[k].col=4;

boss\_shot.bullet[k].angle=0;

boss\_shot.bullet[k].x=boss.x;

boss\_shot.bullet[k].y=boss.y;

boss\_shot.bullet[k].vx = cos(angle)\*1.4\*1.2;

boss\_shot.bullet[k].vy = sin(angle)\*1.4;

}

}

se\_flag[7]=1;

}

for(i=0;i<BOSS\_BULLET\_MAX;i++){

if(boss\_shot.bullet[i].flag>0){

if(boss\_shot.bullet[i].state==0){

if(boss\_shot.bullet[i].cnt<150)

boss\_shot.bullet[i].vy+=0.03;

boss\_shot.bullet[i].x+=boss\_shot.bullet[i].vx;

boss\_shot.bullet[i].y+=boss\_shot.bullet[i].vy;

}

if(boss\_shot.bullet[i].state==1){

if(boss\_shot.bullet[i].cnt<160)

boss\_shot.bullet[i].vy+=0.03;

boss\_shot.bullet[i].x+=boss\_shot.bullet[i].vx;

boss\_shot.bullet[i].y+=boss\_shot.bullet[i].vy;

boss\_shot.bullet[i].angle=atan2(boss\_shot.bullet[i].vy,boss\_shot.bullet[i].vx);

}

}

}

}

也请确认变更了的func.h。

## 第三十一章 制作弹幕——早苗风格☆状弹幕

弹幕名：无　（早苗风格☆状弹幕的样板）

制作难易度：★★★★　(4/10)

运行结果

第五回弹幕制作讲座是以练习为目的的弹幕。

我们来模仿早苗风格那样绘制☆形状的漂亮曲线吧。

首先我们学习一下☆这个形状有什么性质吧。（译者注：前方数学高能，平面几何不好的速速退散）

α+β=γ（译者注：三角形的一个外角等于其不相邻的两个内角和）这个定理我们在中学的时候就已经学过了呢。

同样地，a+b=c。

这样考虑的话，把α、β、a……五个角加起来就是180°了呢。

既然这样，我们也就知道α的角度是π/5了。

下图中以描绘“始点”的位置开始为最初的点，向stt0速度的方向引一条线。

星星的中心是Boos的坐标。从那里开始到始点位置的距离以红色的线标识。

sst在0的时候一直前进直到与圆相交。我们可以这样判断：如果从中心点开始到当前位置的距离超过半径r的话那就相交了。如果判定相交的话，就把sst设置为1然后把角度变成-(PI-PI/5)然后继续前进。

然后沿着sst1的速度方向前进，并且同样地一直到与圆相交为止。

一直到4为止都前进，到5的话就停止。

----在 boss\_shotH.cpp 中进行以下追加 ----

//☆弾

void boss\_shot\_bulletH005(){

#define TM005 820

#define RANGE005 100.0

#define LEM005 4

int i,j,k,n,t=boss\_shot.cnt%TM005,t2=boss\_shot.cnt;

static int sst,bnum;

static double sx,sy,sangle;

//1个周期的最开始的话

if(t==0){

sst=0;

sx=boss.x;

sy=boss.y-RANGE005;

sangle=PI/5/2+PI/2;

bnum=0;

}

//绘制星星

if(sst<=4){

for(i=0;i<2;i++){

sx+=cos(sangle)\*LEM005;//计算坐标

sy+=sin(sangle)\*LEM005;

//如果与圆相交的话

if((sx-boss.x)\*(sx-boss.x)+(sy-boss.y)\*(sy-boss.y)>RANGE005\*RANGE005){

sangle-=(PI-PI/5);//变换方向

sst++;//状态变换

if(sst==5)break;//如果到5的话停止

}

for(j=0;j<5;j++){//绘制5个星星

if((k=search\_boss\_shot())!=-1){

boss\_shot.bullet[k].flag=1;

boss\_shot.bullet[k].state=j;

boss\_shot.bullet[k].cnt=0;

boss\_shot.bullet[k].spd=0;

boss\_shot.bullet[k].knd=10;

boss\_shot.bullet[k].col=j;

boss\_shot.bullet[k].angle=-PI/2+PI2/5\*j;

boss\_shot.bullet[k].x=sx;

boss\_shot.bullet[k].y=sy;

boss\_shot.bullet[k].vx = cos(sangle)\*1.4\*1.2;

boss\_shot.bullet[k].vy = sin(sangle)\*1.4;

boss\_shot.bullet[k].base\_angle[0]=sangle-PI+PI/20\*bnum;

}

}

bnum++;

}

se\_flag[0]=1;

}

for(i=0;i<BOSS\_BULLET\_MAX;i++){

if(boss\_shot.bullet[i].flag>0){

int cnt=boss\_shot.bullet[i].cnt;

//如果状态不满10

if(boss\_shot.bullet[i].state<10){

if(t==150){//如果是150的话把星星往五个方向发射

boss\_shot.bullet[i].spd=4;//速度

boss\_shot.bullet[i].cnt=0;//设置计数器

boss\_shot.bullet[i].state+=10;//状态编号变更

}

}

else if(boss\_shot.bullet[i].state<20){//10~19なら

if(cnt<=80){//减速

boss\_shot.bullet[i].spd-=0.05;

}

if(cnt==100){//从设置好的基准角度出发绘制漂亮的曲线

boss\_shot.bullet[i].angle=boss\_shot.bullet[i].base\_angle[0];

}

if(cnt>=100 && cnt<160){//加速

boss\_shot.bullet[i].spd+=0.015;

}

}

}

}

}

也请确认变更了的func.h。

不过有一个可怕的问题，那就是全部的弹都是在固定的地方安置的，如果在意这个问题的话，可以适当地用随机数调整一下角度。

## 第三十二章 练习“然后都不见了？”风格的弹幕吧！（1）

弹幕名：无　（“然后都不见了？”（「そして誰もいなくなるか？」）的样板）

（译者注：由于本人只玩过日文版的东方，所以对于其中的SC译名都是按照自己的想法翻译的，如果和现行的汉化版中的译名不同请见谅。）

制作难易度：★★★★　(4/10)

运行结果

第六回弹幕制作讲座是以练习为目的的弹幕。

我们来试着做出“然后都不见了？”中那样从四面八方逼过来的子弹。

在前面的章节中我们为了描绘☆而采取了一边运动坐标一边放置子弹的方法，这次我们也和前面一样。

也即是说，我们再来让坐标一边移动一边放置子弹吧。

这一次我们的计算式子就没有☆的这么复杂了，只要我们把方程列出来就能够完成了。

为了今后的扩充，我们使用和之前一样的方法。

接下来，以画面左上角为起始点，首先往画面右上方其角度ange=0的方向移动。

当到达画面右上角的话，使angle+=PI/2，接着往下移动。然后当到达画面右下角的话就angle+=PI/2……

如此进行，一直到绕着画面四个角走了一圈。

为了仅仅修改一个宏定义，就可以漂亮地自动变更所有的子弹的放置间隔，我们也就稍微试着这样子漂亮地写一下吧。

#define DNUMX 15

是画面水平方向放置的子弹的个数。

画面宽度÷放置的个数

这样子计算的话就可以求得子弹的放置间隔。

以这个间隔为基准去除画面的高度，也就可以求得纵向放置的个数，另外，我们用画面的高度除以个数就可以求得竖直方向放置的间隔了。（译者注：我实在没弄懂作者这样处理的意图，大概是为了精度处理吧。）

在下面的声明部分进行了上述过程。

之后只需要往画面中央发射就行了。

现在我们来实现bossatan3这个函数，它用来求得第k号子弹往x，y坐标前进所需的角度。

传入子弹的编号以及想要前进的坐标就可以返回角度。

这次我们制作3个种类的子弹的运动。

由于只要稍微改变一下速度、角度等参数就能够形成各种各样的角度，我们来试着这样子做一下吧。

----在 boss\_shotH.cpp 中进行以下追加 ----

//从四面八方往中央发射

void boss\_shot\_bulletH006(){

#define TM006 300

//横竖方向放置的子弹的个数

#define DNUMX 15

int i,j,k,t=boss\_shot.cnt%TM006,t2=boss\_shot.cnt;

double xlen=FMX/DNUMX;//要在画面上横向放置DNUM个子弹需要xlen的间隔

int dnumy=FMY/xlen;/以xlen为间隔的话再竖直方向需要多少个子弹

int cnum;

if(dnumy==0){printfDx("boss\_shot\_bulletH006除以0\n"); return ;}

double ylen=FMY/dnumy;//需要ylen的放置区间

double x,y,angle;

static int num;

if(t2==0)num=0;

if(t==0){

x=0,y=0,angle=0,cnum=0;

//从画面左上角开始右→下→左→上 四个方向移动

for(j=0;j<4;j++){

//dnumy 水平方向DNUMX 竖直方向dnumy

int to = j%2 ? dnumy: DNUMX ;

//由于水平、竖直方向放置的个数不一样，进行to次子弹放置

for(i=0;i<=to;i++){

if((k=search\_boss\_shot())!=-1){

boss\_shot.bullet[k].x = x;//坐标

boss\_shot.bullet[k].y = y;

boss\_shot.bullet[k].knd = 4;//子弹的种类

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

switch(num){

case 0://发射种类1

boss\_shot.bullet[k].col = 0;//子弹的颜色

boss\_shot.bullet[k].angle = bossatan3(k,FMX/2,FMY/2);//角度

boss\_shot.bullet[k].spd = 1.3;//速度

boss\_shot.bullet[k].state = 0;//状态

break;

case 1://发射种类2

boss\_shot.bullet[k].col = 3;//子弹的颜色

boss\_shot.bullet[k].angle = bossatan3(k,FMX/2,FMY/2);//角度

boss\_shot.bullet[k].spd

= 1.4+ ((j%2 ? -1 : 1) \* ((cos(PI2/to\*i-PI)+1)/2))\*0.4;//速度

boss\_shot.bullet[k].state = 1;//状态

break;

case 2://发射种类3

boss\_shot.bullet[k].col = 6;//子弹的颜色

boss\_shot.bullet[k].angle = bossatan3(k,FMX/2,FMY/2);//角度

boss\_shot.bullet[k].spd = 1.3;//速度

boss\_shot.bullet[k].state = 2;//状态

boss\_shot.bullet[k].base\_angle[0]

= PI/1000 \* (j%2 ? -1 : 1) \* ((cos(PI2/to\*i-PI)+1)/2);//合计角度

break;

}

se\_flag[0]=1;

cnum++;

}

x+=cos(angle)\*xlen;

y+=sin(angle)\*ylen;

}

angle+=PI/2;

}

}

for(i=0;i<BOSS\_BULLET\_MAX;i++){

if(boss\_shot.bullet[i].flag>0){

int cnt=boss\_shot.bullet[i].cnt;

switch(boss\_shot.bullet[i].state){

case 2://如果是发射种类3

//角度加和

boss\_shot.bullet[i].angle+=boss\_shot.bullet[i].base\_angle[0];

break;

default:

break;

}

}

}

if(t==TM006-1)

num=(++num)%3;

}

也请确认变更了的func.h。

由于还有固定放置的子弹，因此有必要调整一下。

## 第三十三章 练习“然后都不见了？”风格的弹幕吧！（2）

弹幕名：无　（“然后都不见了？”（「そして誰もいなくなるか？」）的样板）

制作难易度：★　(1/10)

运行结果

第七回弹幕制作讲座是以练习为目的的弹幕。

在前面的章节中我们已经完成了“然后都不见了？”那样从四面八方逼过来的子弹了呢。

在前面我们从1个固定的地方只发射1发子弹，而现在我们从一个固定的地方发射两发子弹。

和前面的工程相比基本没什么改变。

变更了的地方用红字标识出来。

----在 boss\_shotH.cpp 中进行以下追加 ----

//从四面八方直线发射

void boss\_shot\_bulletH007(){

#define TM007 300

#define DNUMX 9

int i,j,k,s,t=boss\_shot.cnt%TM007,t2=boss\_shot.cnt;

double xlen=FMX/DNUMX;//要在画面上横向放置DNUM个子弹需要xlen的间隔

int dnumy=FMY/xlen;/以xlen为间隔的话再竖直方向需要多少个子弹

int cnum;

if(dnumy==0){printfDx("boss\_shot\_bulletH006除以0\n"); return ;}

double ylen=FMY/dnumy;//需要ylen的放置区间

double x,y,angle;

static int num;

if(t2==0)num=0;

if(t==0){

x=0,y=0,angle=0,cnum=0;

for(j=0;j<4;j++){

int to=j%2 ? dnumy: DNUMX ;

for(i=0;i<=to;i++){

for(s=0;s<2;s++){

if((k=search\_boss\_shot())!=-1){

boss\_shot.bullet[k].x = x;//坐标

boss\_shot.bullet[k].y = y;

boss\_shot.bullet[k].knd = 4;//子弹种类

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

switch(num){

case 0://直线

boss\_shot.bullet[k].col = 0;//子弹颜色

boss\_shot.bullet[k].angle = angle+PI/2;//角度

boss\_shot.bullet[k].spd = 1.1+0.5\*s;//速度

boss\_shot.bullet[k].state = 0;//状态

break;

case 1://斜向交叉

boss\_shot.bullet[k].col = 3;//子弹的颜色

boss\_shot.bullet[k].angle = angle+PI/2-PI/14+PI/7\*s;//角度

boss\_shot.bullet[k].spd = 1.3;//速度

boss\_shot.bullet[k].state = 1;//状态

break;

}

se\_flag[0]=1;

cnum++;

}

}

x+=cos(angle)\*xlen;

y+=sin(angle)\*ylen;

}

angle+=PI/2;

}

}

if(t==TM006-1)

num=(++num)%2;

}

也请确认变更了的func.h。

由于还有固定放置的子弹，因此有必要调整一下。

## 第三十四章 试着制作激光吧

制作难易度：★★★★★★★★　(8/10)

运行结果

如果说要制作像东方那样的弹幕的话，肯定要碰壁的那就是制作激光了。

激光到底是基于什么构造呢。

估计大家都抱着这样子的疑问吧。

在这里我就为大家介绍一下以我自己的方式的激光实现方法吧。

所以，我并不知道这到底是不是游戏业界通常使用的方法，因此先说明一下我自己也不知道这是不是一个合适的方法。

其次，就有必要讨论一下勾股定理、点与直线的距离以及向量的话题了。

虽然这是基础级别的数学讨论，但是如果有“这种东西根本就忘光了嘛！”这类人的话，还请你在谷歌上输入关键词搜一下吧。

首先我来说明一下激光的基本想法。

上图中红色的长方形部分是碰撞判定部分。

如果显示激光的图像的话也就将那个特定的长方形部分作为碰撞判定部分了。

这样一来无论是小的、大的、斜着的还是笔直的激光都能够做到了。

指定图像的4个顶点需要使用使用

DrawModiGraph函数

使用这个函数能够表示出任何朝向的长方形。

那么，碰撞判定需要怎么做呢？

其实现在就变成“长方形和圆的相交问题”了。

用于计算点和直线的距离、圆与圆的距离的方程都是大家都是知道的，但是判定长方形与圆的相交的方程式没有普通的方法的。

我们试着按着以下的步骤进行判定。

1、圆中是否有长方形的顶点

如果有判定为接触

如果没有如果没有的话进行后面的判定式的判定

2、圆是否进入了长方形中

如果进入的话判定为接触

如果没有的话进行后面的判定式的判定

3、圆的中心和长方形边的距离是否在半径以内

如果在半径以内判定为接触

除此之外可以判定为完全没有接触

首先我们来考虑“1、圆中是否有长方形的顶点”吧。

这里的处理只需要我们一个一个地计算“顶点和圆的中心”的距离就好了，非常简单。

-使用勾股定理计算点与点之间的距离。

这样一来

r = sqrt( (x\*x) + (y\*y) )

我们判定这个r是否在半径以内，就可以进行接触判定了。

//点是否在圆中 0：没有 1：是

double question\_point\_and\_circle(pt\_t p, pt\_t rp,double r){

double dx=p.x-rp.x,dy=p.y-rp.y;

if(dx\*dx + dy\*dy < r\*r) return 1;

else return 0;

}

/\*中间省略\*/

/\*判定圆中是否有长方形4个顶点中的某一个\*/

for(i=0;i<4;i++){

if(question\_point\_and\_circle(pt[i],rpt,r)==1)

return 1;

}

这样子写的话实现好了。

接下来我们来考虑“2、圆是否进入了长方形中”吧。

如果a，b，r所成的角θ和 d，c，r所成的角θ’都在π/2以内的话，那么就可以判定圆的中心在长方形中。

那么我们试着求这个θ。

利用向量A和向量B求解cosθ的方法我们在高中的时候就学过了呢。

向量A和B的内积定义：

在C语言中能够如下进行简单计算。

向量A和B的内积与外积定义：

将两者一起使用的话再C语言中就能够简单地利用下面的函数来求角度。

我们试着实现上述结论吧。

/\* 平面向量 \*/

typedef struct {

double x, y;

} Vector2\_t;

/\* diff ← 向量 p - q \*/

void Vector2Diff(Vector2\_t \*diff, const Vector2\_t \*p, const Vector2\_t \*q)

{

diff->x = p->x - q->x;

diff->y = p->y - q->y;

}

/\* 向量 p 和 q 的内积（译者注：线性代数中这叫内积，高中好像教的是“数量积”、“点乘”？） \*/

double Vector2InnerProduct(const Vector2\_t \*p, const Vector2\_t \*q)

{

return p->x \* q->x + p->y \* q->y;

}

/\* 向量 p 和 q 的外积

（译者注：注意，2维空间中的叉乘是：

V1(x1, y1) X V2(x2, y2) = x1y2 – y1x2

看起来像个标量，事实上叉乘的结果是个向量，方向在z轴上。上述结果是它的模。在二维空间里，让我们暂时忽略它的方向，将结果看成一个向量，我们有：

A x B = |A||B|Sin(θ)

然而角度 θ和点乘的角度有一点点不同，他是有正负的，是指从A到B的角度。）\*/

double Vector2OuterProduct(const Vector2\_t \*p, const Vector2\_t \*q)

{

return p->x \* q->y - p->y \* q->x;

}

/\*中略\*/

/\*中间省略\*/

/\* 求向量 C→P 和 C→Q 所成的角θ以及旋转方向。\*/

Vector2\_t c, p, q; /\* 输入数据 \*/

Vector2\_t cp; /\* 向量 C→P \*/

Vector2\_t cq; /\* 向量 C→Q \*/

double s; /\* 外积：(C→P) × (C→Q) \*/

double t; /\* 内积：(C→P) ・ (C→Q) \*/

double theta,theta2;/\* θ (弧度) \*/

/\*将 c，p，q 设定为想要的值。\*/

c.x = pt[0].x; c.y = pt[0].y;

p.x = pt[1].x; p.y = pt[1].y;

q.x = x; q.y = y;

/\* 计算旋转方向以及角度θ。\*/

Vector2Diff(&cp, &p, &c); /\* cp ← p - c \*/

Vector2Diff(&cq, &q, &c); /\* cq ← q - c \*/

s = Vector2OuterProduct(&cp, &cq); /\* s ← cp × cq \*/

t = Vector2InnerProduct(&cp, &cq); /\* t ← cp ・ cq \*/

theta = atan2(s, t);

接下来考虑“3、圆的中心和长方形边的距离是否在半径以内”吧。

我还记得在高中的时候学习过求点和直线的距离的方法，现在我们来求点和“线段”的距离。

将长方形的顶点设定为α、β的时候，首先判定圆的中心r是否在长方形中。

如果像上面那样r在长方形中的话，向量ab和向量ac所形成的θ应该有cosθ>0。

如果在长方形中的话，则从r开始向αβ引的垂线的长度就是距离了。

现在利用垂线向量和线段αβ的内积为0的结论进行以下实现。

//求点和线段的距离

double get\_distance(double x, double y, double x1, double y1,

double x2, double y2){

double dx,dy,a,b,t,tx,ty;

double distance;

dx = (x2 - x1); dy = (y2 - y1);

a = dx\*dx + dy\*dy;

b = dx \* (x1 - x) + dy \* (y1 - y);

t = -b / a;

if (t < 0) t = 0;

if (t > 1) t = 1;

tx = x1 + dx \* t;

ty = y1 + dy \* t;

distance = sqrt((x - tx)\*(x - tx) + (y - ty)\*(y - ty));

return distance;

}

/\*中间省略\*/

/\*求线段和点的距离\*/

for(i=0;i<4;i++){

if(get\_distance(rpt.x,rpt.y,pt[i].x,pt[i].y,pt[(i+1)%4].x,pt[(i+1)%4].y)<r)

return 1;

}

现在试着将上述内容作为一束激光的判定函数来实现吧。

----在 out\_lazer.cpp中进行的新规追加如下----

#include "../include/GV.h"

#include <math.h>

#include <stdio.h>

/\* 平面向量 \*/

typedef struct {

double x, y;

} Vector2\_t;

/\* diff ← 向量 p - q \*/

void Vector2Diff(Vector2\_t \*diff, const Vector2\_t \*p, const Vector2\_t \*q){

diff->x = p->x - q->x;

diff->y = p->y - q->y;

}

/\* 向量 p 和 q 的内积 \*/

double Vector2InnerProduct(const Vector2\_t \*p, const Vector2\_t \*q){

return p->x \* q->x + p->y \* q->y;

}

/\* 向量 p 和 q 的外积 \*/

double Vector2OuterProduct(const Vector2\_t \*p, const Vector2\_t \*q){

return p->x \* q->y - p->y \* q->x;

}

//求点和线段的距离

double get\_distance(double x, double y, double x1, double y1,

double x2, double y2){

double dx,dy,a,b,t,tx,ty;

double distance;

dx = (x2 - x1); dy = (y2 - y1);

a = dx\*dx + dy\*dy;

b = dx \* (x1 - x) + dy \* (y1 - y);

t = -b / a;

if (t < 0) t = 0;

if (t > 1) t = 1;

tx = x1 + dx \* t;

ty = y1 + dy \* t;

distance = sqrt((x - tx)\*(x - tx) + (y - ty)\*(y - ty));

return distance;

}

//返回点与点的距离

double get\_pt\_and\_pt(pt\_t p1, pt\_t p2){

return sqrt((p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y));

}

//点是否在圆中。0：没有 1：在

double question\_point\_and\_circle(pt\_t p, pt\_t rp,double r){

double dx=p.x-rp.x,dy=p.y-rp.y;

if(dx\*dx + dy\*dy < r\*r) return 1;

else return 0;

}

//交换

void swap\_double(double \*n, double \*m){

double t=\*m;

\*m=\*n;\*n=t;

}

//通过3个点返回所成夹角

double get\_sita(pt\_t pt0,pt\_t pt1,pt\_t rpt){

/\* 计算向量 C→P 和 C→Q 所形成的角θ以及旋转方向\*/

Vector2\_t c, p, q; /\* 输入数据 \*/

Vector2\_t cp; /\* 向量 C→P \*/

Vector2\_t cq; /\* 向量 C→Q \*/

double s; /\* 外积：(C→P) × (C→Q) \*/

double t; /\* 内积：(C→P) ・ (C→Q) \*/

double theta; /\* θ (弧度) \*/

/\*将 c，p，q 设定为所想的值。\*/

c.x = pt0.x; c.y = pt0.y;

p.x = pt1.x; p.y = pt1.y;

q.x = rpt.x; q.y = rpt.y;

/\* 计算旋转方向以及角度θ。\*/

Vector2Diff(&cp, &p, &c); /\* cp ← p - c \*/

Vector2Diff(&cq, &q, &c); /\* cq ← q - c \*/

s = Vector2OuterProduct(&cp, &cq); /\* s ← cp × cq \*/

t = Vector2InnerProduct(&cp, &cq); /\* t ← cp ・ cq \*/

theta = atan2(s, t);

return theta;

}

//判定长方形和圆是否相碰

int hitjudge\_square\_and\_circle(pt\_t pt[4], pt\_t rpt, double r){

int i;

double a[4],b[4];//a:傾き　b:y切片

double x=rpt.x,y=rpt.y;

/\*判断圆中是否有长方形4个顶点中的某一个\*/

for(i=0;i<4;i++){

if(question\_point\_and\_circle(pt[i],rpt,r)==1)

return 1;

}

/\*到此为止\*/

/\*判定长方形中是否有物体进入\*/

theta =get\_sita(pt[0],pt[1],rpt);//3点所成的角1

theta2=get\_sita(pt[2],pt[3],rpt);//3点所成的角2

if(0<=theta && theta<=PI/2 && 0<=theta2 && theta2<=PI/2)

return 1;

/\*到此为止\*/

/\*求线段和点的距离\*/

for(i=0;i<4;i++){

if(get\_distance(rpt.x,rpt.y,pt[i].x,pt[i].y,pt[(i+1)%4].x,pt[(i+1)%4].y)<r)

return 1;

}

/\*到此为止\*/

return 0;//如果哪里都没有碰撞的话就没有碰撞判定

}

int out\_lazer(){

int i,j;

pt\_t sqrp[4],rpt={ch.x,ch.y};//长方形四个顶点和圆的中心

//激光数次循环

for(i=0;i<LAZER\_MAX;i++){

//如果激光已经登录且设定了碰撞判定

if(lazer[i].flag>0 && lazer[i].hantei!=0){

for(j=0;j<4;j++){//设定激光的四个顶点

sqrp[j].x=lazer[i].outpt[j].x;

sqrp[j].y=lazer[i].outpt[j].y;

}

//长方形和圆的接触判定

if(hitjudge\_square\_and\_circle(sqrp,rpt,CRANGE))

return 1;

}

}

return 0;

}

另外，前面我们使用物理的运动来控制敌人，而激光也要漂亮地旋转，表现出在既定的坐标附近接连地缓慢旋转、随后突然停止的效果，因此我们也对此进行物理上的计算。

激光的旋转速度在这里我们称为角速度。

我们现在从力学的三大公式开始，逐步推导出激光的旋转速度吧。

以初速度ω0飞出的激光在时间ty内转动θmax角后停止的时候，由于最终的角速度为0，因此将ω=0带入①中，有

然后，带入③中，有，

然后把这个式子带入⑤中，有，

两边约掉ω0，有，

然后将⑥带入⑤中，有，

将⑥、⑦带入②中，有，

这样就导出A了。由于能够写出在ty次计数类进行θmax度角旋转的方程，那么我们根据这个方程来计算角度的话就能够完成漂亮地加速了。

因此，我们进行以下追加以实现激光。

---- struct.h 追加 ----

//为激光的物理计算处理准备的结构体

typedef struct{

int conv\_flag;//是否旋转的flag

double time,base\_ang,angle;//旋转时间、基本角度、角度

double conv\_x,conv\_y,conv\_base\_x,conv\_base\_y;//旋转前的坐标、旋转的基本坐标

}lphy\_t;

//激光的结构体

typedef struct{

int flag,cnt,knd,col,state;// flag、计数器、种类、颜色

double haba,angle,length,hantei;// 幅度、角度、长度、判定范围（相对显示的幅度，指定0~1中的值）

pt\_t startpt,disppt[4],outpt[4];//发射激光的初始点、显示坐标、碰撞判定范围

lphy\_t lphy;

}lazer\_t;

----在 boss\_shot.cpp 中追加红字部分 ----

void enter\_boss(int num){

if(num==0){//中路Boss开始的时候

memset(enemy,0,sizeof(enemy\_t)\*ENEMY\_MAX);//销毁杂兵敌人

memset(shot,0,sizeof(shot\_t)\*SHOT\_MAX);//销毁弹幕

boss.x=FMX/2;//Boss的初始坐标

boss.y=-30;

boss.knd=-1;//弹幕的种类

}

boss.flag=1;

boss.hagoromo=0;//是否扇状扩展的flag

boss.endtime=99\*60;//剩余时间

boss.state=1;//变为待机中状态

boss.cnt=0;

boss.graph\_flag=0;//还原绘制flag

boss.knd++;

boss.wtime=0;//初始化待机时间

memset(lazer,0,sizeof(lazer\_t)\*LAZER\_MAX);//初始化激光信息

memset(&boss\_shot,0,sizeof(boss\_shot\_t));//初始化Boss的弹幕信息

input\_phy(60);// 60次计数后在物理计算下回到固定位置

}

激光旋转了。不过是在离开旋转基准点的位置进行旋转的。

因此，我们设定

控制在哪里旋转的conv\_x，conv\_y

控制以哪个地方为基准 选装的conv\_base\_x，conv\_base\_y

设定激光从哪里发射的startpt.x，startpt.y

然后使用坐标的旋转以变换坐标。

将p旋转β角变为p’，我们可以用以下方法计算p’。

在这里，

因此，

因此用矩阵表示就是，

因此我们将这个进行坐标转换的函数

//坐标旋转

//将点(x0,y0)以(mx,my)为基准进行ang角旋转后的坐标传入(x,y)中。

void conv\_pos0(double \*x, double \*y, double x0, double y0, double mx, double my,double ang){

double ox=x0-mx,oy=y0-my;

\*x=ox\*cos(ang) +oy\*sin(ang);

\*y=-ox\*sin(ang)+oy\*cos(ang);

\*x+=mx;

\*y+=my;

}

像这样子进行实现。

激光的幅度用“haba”设定。（译者注：看了前面的代码的读者可能会注意到作者对有些变量的命名并不是用英文，而是用日语的罗马音标注的。）

激光的长度用“lenght”设定。

激光的碰撞判定范围用hantei设定。带入0~1的值，就可以确定是设定为显示出的总幅度的多大比例的范围了。

由于要把这些结果全部以运行结果的方式来展示很困难，因此我们将会在下一章中展示的反魂蝶的激光判定的绘制结果。

弹幕的话请在下一章中阅读。

谢辞：

这些理论参考了C言語何でも質問サイト（译者注：Dixq先生的网站）里大家的意见。非常感谢。

## 第三十五章 制作弹幕——“反魂蝶~八部盛开~”

弹幕：“反魂蝶~八部盛开~”（ 「反魂蝶～八部咲き～」）

制作难易度：★★★★★★　(6/10)

运行结果

点击这里观看包含音乐的高画质版本

首先如果您还没有理解激光的话是没法做出反魂蝶的，因此还没有学习“试着制作激光吧”这一章节的读者请先学习后再往下看。

……接下来上面激光的解说部分已经把我累坏了，这里的制作部分就让我稍微轻松一下吧。（汗）

void boss\_shot\_bulletH009(){

#define TM009 420

#define DIST 60

int i,j,k,s,t=boss\_shot.cnt%TM009,t2=boss\_shot.cnt;

double angle;

static int num;

if(t2==0)num=4;

if(t==0){

for(j=0;j<2;j++){

for(i=0;i<num;i++){

int plmn=(j ? -1 : 1);

if((k=search\_lazer())!=-1){

lazer[k].col = j;//子弹的颜色

lazer[k].knd = 0;//子弹的种类

lazer[k].angle = PI2/num\*i+PI2/(num\*2)\*j+PI2/(num\*4)\*((num+1)%2);//角度

lazer[k].startpt.x= boss.x+cos(lazer[k].angle)\*DIST;//开始绘制的角度

lazer[k].startpt.y= boss.y+sin(lazer[k].angle)\*DIST;

lazer[k].flag = 1;//是否显示　0:否　1:是

lazer[k].cnt = 0;

lazer[k].haba = 2;//幅度

lazer[k].state = j;//

lazer[k].length = 240;//长度

lazer[k].hantei = 0;//是否碰撞判定　0:是　1:否

lazer[k].lphy.conv\_flag=1;//是否旋转　0:否　1：是

lazer[k].lphy.conv\_base\_x=boss.x;//旋转的基准坐标

lazer[k].lphy.conv\_base\_y=boss.y;

lazer[k].lphy.conv\_x=lazer[k].startpt.x;//旋转的激光位置

lazer[k].lphy.conv\_y=lazer[k].startpt.y;

input\_lphy(&lazer[k],80,PI/num\*plmn);//登录第k号激光在80次计数内只旋转PI/num\*plmn角度的信息

}

}

}

se\_flag[33]=1;

}

//60次计数一下每10次计数1次

if(t==50){

angle=rang(PI);//自机与Boss所成角度

for(s=0;s<2;s++){

for(t=0;t<3;t++){

for(j=0;j<3;j++){

for(i=0;i<30;i++){

if((k=search\_boss\_shot())!=-1){

boss\_shot.bullet[k].col = s;//子弹的颜色

boss\_shot.bullet[k].x = boss.x;//坐标

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 11;//子弹的种类

boss\_shot.bullet[k].angle = angle+PI2/30\*i+PI2/60\*s;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 1.8-0.2\*j+0.1\*s;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = t;

}

se\_flag[0]=1;

}

}

}

}

}

if(t>=170 && t<310 && (t-170)%35==0){

int div=((t-170)%70==0) ? -1 : 1;

angle=rang(PI);//自机与Boss所成的角度

for(s=0;s<2;s++){//有两类速度不同的子弹

for(t=0;t<3;t++){//从一个地方一分为三

for(i=0;i<30;i++){//1周30个

if((k=search\_boss\_shot())!=-1){

boss\_shot.bullet[k].col = 2;//子弹的颜色

boss\_shot.bullet[k].x = boss.x;//坐标

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 11;//子弹的种类

boss\_shot.bullet[k].angle = angle+PI2/30\*i;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 2-0.3\*s;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 10+t;

boss\_shot.bullet[k].base\_angle[0] = PI/300\*div;

}

}

}

se\_flag[0]=1;

}

}

if(t==360){

angle=rang(PI);//自机与Boss所成的角度

for(t=0;t<3;t++){//从一个地方一分为三

for(i=0;i<30;i++){

if((k=search\_boss\_shot())!=-1){

boss\_shot.bullet[k].col = 1;//子弹的颜色

boss\_shot.bullet[k].x = boss.x;//坐标

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 0;//子弹的种类

boss\_shot.bullet[k].angle = angle+PI2/30\*i;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 1.8;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 20+t;

}

}

}

se\_flag[0]=1;

}

for(i=0;i<BOSS\_BULLET\_MAX;i++){

if(boss\_shot.bullet[i].flag>0){

int cnt=boss\_shot.bullet[i].cnt;

int state=boss\_shot.bullet[i].state;

if(state%10==0){

if(cnt>90 && cnt<=100)

boss\_shot.bullet[i].spd-=boss\_shot.bullet[i].spd/220;

}

if(state%10==1){

if(cnt>50)

boss\_shot.bullet[i].spd+=boss\_shot.bullet[i].spd/45;

}

if(state%10==2){

if(cnt>65)

boss\_shot.bullet[i].spd+=boss\_shot.bullet[i].spd/90;

}

if(10<=state && state<=12){

if(cnt>15 && cnt<=80)

boss\_shot.bullet[i].angle+=boss\_shot.bullet[i].base\_angle[0];

}

}

}

for(i=0;i<LAZER\_MAX;i++){

if(lazer[i].flag>0){

int cnt=lazer[i].cnt;

int state=lazer[i].state;

if(state==0 || state==1){

if(cnt==80){

lazer[i].haba=30;

lazer[i].hantei=0.5;

}

if(cnt>=260 && cnt<=320){

if(cnt==280)

lazer[i].hantei=0;

lazer[i].haba=10\*(60-(cnt-260))/60.0;

if(cnt==320)

lazer[i].flag=0;

}

}

}

}

if(t==TM009-1)

num=(++num);

}

## 第三十六章 来仿制“深弹幕结界”吧

弹幕：“深弹幕结界”（「深弾幕結界」）仿制

制作难易度：无法测定　(非常困难/10)

运行结果

点击这里观看包含音乐的高画质版本

首先呢，我并不准备为大家介绍这个弹幕的制作方法。（诶？）

请认为本章仅仅是向大家说明“这样子的弹幕只要努力的话就能做出来哦”这种程度的介绍而已。m(\_ \_)m

为了非常贴近地完成这个弹幕，由于无法进行完全相同的动作，因此我不断地截屏重播动画、停止动画、再截屏，然后在画板之类的工具里面确认子弹的坐标、求角度、推算计算式以求与原作中基本相同……一直这样重复工作。具体要我说明的话那就太麻烦了orz。

我大致说明一下计算式的制作方法。

首先，反复多次重播动画，然后确认这些动作是怎么进行的。

这样的话可以发现，魔法阵在以区域的中心为基准进行旋转，从那里（译者注：指魔法阵的外围）一个个地发出朝中心方向发射出去的子弹。

这个弹幕的第1发子弹朝着中心飞去，然后稍微改变一下子弹的前进角度，最后发射出去的子弹以在某种程度上稍微偏离中心的“不同角度”飞去。（译者注：如果您还记得高中物理电学部分的话，可以考虑这样一个类似的模型：带正电的运动粒子朝同样带正电的固定粒子飞去，一开始是朝着固定粒子的中心飞去，但是由于电场力的作用使运动粒子的运动方向逐渐偏离固定粒子中心，然后远离固定粒子飞走。）

至于这个“不同角度”是多少，需要像之前那样使用利用像素坐标求角度的方法来计算。

举个例子，最后发射的子弹会从某个坐标开始朝着某个方向飞去。

首先，从截图中我们确认一下坐标。我确认的坐标是[87,78]。而中心坐标是[192,215]。而子弹实际上射出去的目标坐标为[185,389]。

现在，通过这3个点，我们就可以解出这样两条直线了：最后发射的子弹和中心连线的直线、最后发射的子弹的轨迹的直线。

当我们计算出两条直线后，也就能够知道两条直线的夹角了。

由于已经知道了中心的坐标，接下来只要我们知道了需要变化几次（译者注：从源代码可以看出，这个变化次数事实上由您自己利用时间来确定，当这个时间值越小则轨道越精确，但是计算量也越大），也就知道要以多大角度移动了。

因此我们试着写这样一个工具程序：输入4个点返回两条直线所成的角度。

#include <stdio.h>

#include <math.h>

#define PI 3.141592653589793238462643383279

typedef struct{

double x,y;

}p\_t;

int main(){

p\_t p[4];

char st[2][5]={{"始点"},{"终点"}};

double angle0,angle1;

for(int i=0;i<4;i++){

printf("输入第%d个的%s坐标(x,y)\n",i/2+1,st[i%2]);

scanf(" %lf %lf",&p[i].x,&p[i].y);

}

angle0=atan2(p[1].y-p[0].y,p[1].x-p[0].x);

angle1=atan2(p[3].y-p[2].y,p[3].x-p[2].x);

printf("\n角度 = %.8f[rad]\n角度 = %.2f[°]\n"

,angle1-angle0,(angle1-angle0)/PI\*180);

return 0;

}

运行结果

角度 = 0.34866626[rad]

角度 = 19.98[°]

输入第1个始点的坐标(x,y)

87 78

输入第1个终点的坐标( (x,y)

192 215

输入第2个始点的坐标( (x,y)

87 78

输入第2个终点的坐标( (x,y)

185 389

角度 = 0.34866626[rad]

角度 = 19.98[°]

使用这个程序试着计算角度之后，得知结果为19.98°。

我们干脆忽略小于1°的误差，直接考虑使用20°。

如果是60fps的话1帧就是16.66ms，我们通过距离和时间来求速度，然后写出符合原作中轨道的计算式。

这些的内容非常麻烦，弹幕也只做到2周目为止了，请见谅。m(\_ \_;)m

然后，将设定子弹暂时停止的时间的cnt\_till、设定子弹开始运动的时间的cnt\_stt追加到子弹的结构体中，不过这只是本章才会使用的变量，也就在本章的工程中追加了。

另外，我们还要让作为子弹的发射位置的魔法阵一遍旋转一边移动呢。

我们通过child这个结构体来完成这个工作。

由于这个机能我想在稍微后面再介绍，因此在这里我就不多说明了。

这种事情也是能实现的，请抱着这样的想法来阅读代码。

//深弹幕结界

void boss\_shot\_bulletH010(){

#define TM010 9000 //最大周期(并没有特别设定的意义)

#define DIST010 (FMX/2\*1.18) // childl在1周目圆周运动的大小

#define DIST0101 (FMX/2\*0.95) // child在2周目圆周运动的大小

#define HANSHU 120 // child在1周目半圆周运动的时间

#define HANSHU1 180 // child在2周目半圆周运动的时间

#define GOOUT010 90 //child从中央往外移动的时间

#define KAISHI010 (HANSHU) //child开始发射的时间

#define KAISHI010\_1 (HANSHU1\*0.6)

#define CHILD\_TIME (HANSHU\*5+HANSHU/3) //child的存在时间

#define CHILD\_TIME1 (HANSHU1\*3+HANSHU1\*0.4) //child的存在时间

#define CHILD\_SHOT\_TIME (CHILD\_TIME-KAISHI010)//child实际发射中的时间

#define CHILD\_SHOT\_TIME1 (CHILD\_TIME1-KAISHI010\_1)// child实际发射中的时间

#define ANG0 PI/9 //1周目的角度

#define ANG1 PI/6 //2周目的角度

#define TIME1 900 //1周目结束的时间

#define RAG 20 //往外发射后直到往内发射之间的延时

#define TERM0 20 //全部发射后直到开始运动的时间

#define ST\_ED 130 //最初开始运动后直到最后的子弹开始运动的时间

#define ST0 (CHILD\_SHOT\_TIME+TERM0) //最开始子弹运动的时间

#define ED0 (CHILD\_SHOT\_TIME+TERM0+ST\_ED) //最后的子弹运动的时间

#define TERM1 47

#define ST\_ED1 105

#define ST1 (CHILD\_SHOT\_TIME1+TERM1)

#define ED1 (CHILD\_SHOT\_TIME1+TERM1+ST\_ED1)

int i,j,k,t=boss\_shot.cnt%TM010,t2=boss\_shot.cnt;

int tt1=boss\_shot.cnt-TIME1;

static int num,flag,knum;

static double child\_dist,child\_angle,child\_dist2,child\_angle2;

if(t2==0){//如果是最开始

input\_phy\_pos(FMX/2,FMY/2, 50);

num=-1;

flag=0;

}

//周期的最开始

if(t==0 || t2==TIME1){

num++;

//child数据的初始化

child\_dist=0;

child\_angle=0;

child\_dist2=0;

child\_angle2=0;

knum=0;

}

//child的登录

if(t2==GOOUT010 || t2==TIME1){

int j=2;

for(i=j-2;i<j;i++){

child[i].flag =1;

child[i].x =boss.x;

child[i].y =boss.y;

child[i].range=0.5;

child[i].spd =1;

child[i].angle=0;

child[i].knd =0;

child[i].col =0;

child[i].cnt =0;

child[i].state=i;

}

}

//1周目

if(num==0){

if(GOOUT010<=t){

//出现后直到将要往外扩展为止

if(GOOUT010<=t && t<GOOUT010+KAISHI010)

child\_dist+=DIST010/KAISHI010;

//一般地旋转

child\_angle+=PI/HANSHU;

}

}

//2周目

if(num==1){

//出现后直到将要往外扩展为止

if(child[0].cnt<KAISHI010\_1)

child\_dist+=DIST0101/KAISHI010\_1;

//一般地旋转

child\_angle-=PI/HANSHU1;

}

//child数据计算

for(i=0;i<CHILD\_MAX;i++){

if(child[i].flag>0){//如果已经登录的话

//轨道计算

child[i].x=boss.x+cos(child\_angle+PI\*child[i].state)\*child\_dist;

child[i].y=boss.y+sin(child\_angle+PI\*child[i].state)\*child\_dist;

//第一次

if(num==0){

if(KAISHI010<child[i].cnt){//如果超过发射开始的计数

if(((t+6)%36)/4<=5 && t%4==0){//这个时候登录子弹

for(j=0;j<3;j++){//往外发射3路

k=knum++;

boss\_shot.bullet[k].col = child[i].state;//子弹的颜色

boss\_shot.bullet[k].x = child[i].x;//坐标

boss\_shot.bullet[k].y = child[i].y;

boss\_shot.bullet[k].knd = 6;//子弹的种类

boss\_shot.bullet[k].angle

= bossatan3(k,boss.x,boss.y)+PI-PI/6+PI/6\*j;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].state = 10+j+10\*child[i].state;//状态

boss\_shot.bullet[k].spd = 0.4;//速度

se\_flag[0]=1;

}

}

else if(t%4==0){//除此之外，每4次执行1次

for(j=0;j<3;j++){

k=knum++;

boss\_shot.bullet[k].col = child[i].state;//子弹的颜色

boss\_shot.bullet[k].x = child[i].x;//坐标

boss\_shot.bullet[k].y = child[i].y;

boss\_shot.bullet[k].knd = 6;//子弹的种类

boss\_shot.bullet[k].angle

= bossatan3(k,boss.x,boss.y)+PI-PI/6+PI/6\*j;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].state = 30;

boss\_shot.bullet[k].spd = 0.4;//速度

}

}

}

//如果到了销毁的时间则销毁之

if(child[i].cnt>CHILD\_TIME)

child[i].flag=0;

}

//第1次end

//第二次

if(num==1){

//如果超过发射计数

if(KAISHI010\_1<child[i].cnt){

if((tt1-55)%(3\*22)<(3\*15) && t%3==0){//这个时候登录

for(j=0;j<3;j++){//往外发射

k=knum++;

boss\_shot.bullet[k].col = child[i].state;//子弹的颜色

boss\_shot.bullet[k].x = child[i].x;//坐标

boss\_shot.bullet[k].y = child[i].y;

boss\_shot.bullet[k].knd = 6;//子弹的种类

boss\_shot.bullet[k].angle

= bossatan3(k,boss.x,boss.y)+PI-PI/6+PI/6\*j;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].state = 10+j+10\*child[i].state;

boss\_shot.bullet[k].spd = 0.4;//速度

se\_flag[0]=1;

}

}

else if(t%3==0){//除此之外每3次执行一次

for(j=0;j<3;j++){//向外发射3路

k=knum++;

boss\_shot.bullet[k].col = child[i].state;//子弹的颜色

boss\_shot.bullet[k].x = child[i].x;//坐标

boss\_shot.bullet[k].y = child[i].y;

boss\_shot.bullet[k].knd = 6;//子弹的种类

boss\_shot.bullet[k].angle

= bossatan3(k,boss.x,boss.y)+PI-PI/6+PI/6\*j;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].state = 30;

boss\_shot.bullet[k].spd = 0.4;//速度

}

}

}

//如果到了销毁的时间就销毁之

if(child[i].cnt>CHILD\_TIME1)

child[i].flag=0;

}

}

}

for(i=0;i<BOSS\_BULLET\_MAX;i++){

if(boss\_shot.bullet[i].flag>0){

int cnt=boss\_shot.bullet[i].cnt;

int state=boss\_shot.bullet[i].state;

//0的话是第1周目内侧发射弹，100的话是2周目内侧发射弹

if(state==0 || state==100){

//如果到了停止的时间的话则停止并修改状态

if(boss\_shot.bullet[i].cnt\_till==cnt){

boss\_shot.bullet[i].spd=0;

boss\_shot.bullet[i].state=1;

if(state==100)

boss\_shot.bullet[i].state+=100;

}

}

//1,101是在上面状态下停止后的状态

if(state==1 || state==101){

//如果到了开始运动的时间的话运动开始并改变状态

if(boss\_shot.bullet[i].cnt\_stt==cnt){

boss\_shot.bullet[i].state++;

if(flag==0)

flag=1;

boss\_shot.bullet[i].cnt=0;

}

}

//2,102是在上面状态下开始加速后（开始移动）的状态

if(state==2 || state==102){

if(flag==1)

boss\_shot.bullet[i].spd+=0.05;

if(boss\_shot.bullet[i].spd>2.0 && flag==1)

flag=2;

if(flag==2){

if(state==2)

if(boss\_shot.bullet[i].spd<2.0)

boss\_shot.bullet[i].spd+=0.05;

if(state==102)

if(boss\_shot.bullet[i].spd<2.7)

boss\_shot.bullet[i].spd+=0.05;

}

}

//11,21是各种1周目、2周目的外侧发射弹

if(state==11 || state==21){

if(cnt==RAG){

double zero\_one;

double ang,spd;

if(num==0){

zero\_one=(double)(t-GOOUT010-RAG-HANSHU)/CHILD\_SHOT\_TIME;

ang=ANG0\*zero\_one;

spd=2.3;

}

if(num==1){

zero\_one=(double)(tt1-RAG-KAISHI010\_1)/CHILD\_SHOT\_TIME1;

ang=PI/6-(PI/6+PI/4)\*zero\_one;

if(tt1==143)

ang=PI/3;

spd=2.3\*0.55;

}

k=knum++;

boss\_shot.bullet[k].col = boss\_shot.bullet[i].state==11 ? 0 : 1;//子弹的颜色

boss\_shot.bullet[k].x = boss\_shot.bullet[i].x;//坐标

boss\_shot.bullet[k].y = boss\_shot.bullet[i].y;

boss\_shot.bullet[k].knd = 6;//子弹的种类

boss\_shot.bullet[k].angle = bossatan3(k,boss.x,boss.y)+ang;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

if(num==0){

boss\_shot.bullet[k].state = 0;

boss\_shot.bullet[k].cnt\_till= (int)(150\*zero\_one+10);

boss\_shot.bullet[k].cnt\_stt

= (int)((ST0-(ST\_ED+TERM0))\*(1-zero\_one)+(ST\_ED+TERM0)+10);

}

if(num==1){

boss\_shot.bullet[k].state = 100;

boss\_shot.bullet[k].cnt\_till= (int)(150\*zero\_one);

boss\_shot.bullet[k].cnt\_stt

= (int)((ST1-(ST\_ED1+TERM1))\*(1-zero\_one)+(ST\_ED1+TERM1));

}

boss\_shot.bullet[k].spd = spd;

}

}

}

}

}

谢辞：

36章承蒙array的鼓励与建议才顺利完成了。非常感谢。

请让我介绍一下array所制作的以编写36章为契机而制作的动画。

## 第三十七章 把警告给消除了吧

这一章要说的话本应该在第一章就说明的，不过还是拖到了现在。

编译的时候估计大家已经注意到编译器的警告了吧。

可能会出现这样子的警告：

1>c:\龍神録プロジェクト\15章\mydat\source\graph.cpp(7) : warning C4244:

'参数' : 'double' 到 'float' 的变换。可能会丢失数据。

这是因为DrawRotaGraphF函数的x,y坐标的参数只能够接受float类型，由于我们传入了double型的变量，因此编译器会警告您可能会丢失值。

虽然我们无视这样子的警告也无所谓，不过警告数量为0的话看起来会好一点，所以我们来消除这些警告吧。

现在我们使用函数重载的机能来解决这个问题。

不想特别去记的人也无所谓。即便不知道这个对以后的制作也不会有问题。

我们使用C++的机能的话，就可以声明函数名相同但内容不同的函数了。

比如

int my\_calc(double a){

}

和

int my\_calc(int a){

}

这两个函数是可以共存的。如果我们传入double型的参数的话会调用上面那个函数，而传入int型的函数的话会调用下面那个函数。

如果我们在这里传入float之类类型的参数的话，那可就因为参数模糊编译器就不通过了。

我们试着使用这个特性来写出接受double型参数的同名函数，清楚地转换类型后像调用普通的函数那样调用吧。

int DrawRotaGraphF( double xd, double yd, double ExRate,

double Angle, int GrHandle, int TransFlag, int TurnFlag = FALSE ) {

return DrawRotaGraphF((float)xd,(float)yd,ExRate,Angle,GrHandle,TransFlag,TurnFlag);

}

int DrawModiGraphF( double x1, double y1, double x2, double y2,

double x3, double y3, double x4, double y4, int GrHandle, int TransFlag ){

return DrawModiGraphF( (float)x1, (float)y1, (float)x2, (float)y2,

(float)x3, (float)y3, (float)x4, (float)y4, GrHandle, TransFlag ) ;

}

在graph.cpp函数的include语句正下方这样子写。由于这样一来就写好了接受double型参数的函数了，首先这边调用，然后转换参数类型进行处理过程的传递，使得警告变为0了。

虽然在一些地方写了一些东西看起来并不好看，其实就是单纯地将double转换为float类型然后调用函数仅此而已。

事实上在稍微前几章中就这样做了，因此可能会有读者已经注意到警告已经是0了。

这次我们只是把警告消除了，因此运行结果并没有改变。

## 第三十八章 来绘制Stage Title吧

敌人的基本行动、Boss、弹幕我们都做好了，看起来我们现在很想完成的就是一个完整的Stage了呢。

从现在开始，要完成一个Stage，我想我们需要做许多细微的工作了。

无论是龙神录还是东方，Stage开始后稍微过了一会儿敌人便会出现，同时Title也显示出来了，而Stage这个时候才算真正开始了呢。现在我们试着展示一下这个Stage Ttile。

与前面所作的工作相同，我们需要编写这样子的程序：定义Stage Title用到的结构体，当到了应该显示Title的时候我们将flag设置为有效，之后自动地进行计算。

登录/计算/绘制，一般而言这三个过程是最基本的格式。

那么，这次因为要增加计算部分，所以请在calc.cpp文件中进行追加。

在calc.cpp中我们将要写入本次以及以后的制作中所需要的各种各样细小的计算处理。

在主循环中只需要调用将在后面实现的cacl\_main函数，而在calc\_main函数中我们决定哪个函数被调用。

因此，首先我们来进行结构体的定义以及变量的声明吧。

然后，为了设定本次的Stage，我们定义这样一些东西：用于标识当前为哪个stage的stage变量以及一共有多少个stage的宏STAGE\_NUM。

----在 define.h 中进行以下追加 ----

//stage数

#define STAGE\_NUM 5

----在 GV.h 中追加以下红字部分----

//flag/状态变量

GLOBAL int func\_state,stage\_count,count,stage; //用于控制函数的变量

GLOBAL int se\_flag[SE\_MAX]; // SE标志

//设定用变量

GLOBAL int stage\_title\_count[STAGE\_NUM];

～略～

GLOBAL stage\_title\_t stage\_title;// Stage Title

----在 function.h 中进行以下追加 ----

//calc.cpp

GLOBAL void calc\_main();

----在 struct.h 中进行以下追加 ----

//游戏Stage Title

typedef struct{

//flag、图像句柄、计数器、亮度、出现计数

int flag,img,cnt,brt,appear\_cnt;

}stage\_title\_t;

接下来，首先我们进行初始化吧。

本次我们定义了“stage\_title\_count”这个变量了呢。

我们往这个变量里面放入用于确定到多少计数的时候才开始显示Title的计数值。

如果我们想在Stage 1中600次计数后开始显示Title，那么那么我们就往state\_title\_count[0]中代入600就行了。

当游戏的计数器值到达这个计数的时候就会进行自动登录。

我们在stage\_title.appear\_cnt中设置在Stage中开始显示Title的计数值。

那么我们就这么干吧。

----在 first\_ini.cpp 中进行以下追加 ----

stage=0;

stage\_title\_count[0]=200;

----在 ini.cpp 中进行以下红字部分追加 ----

//游戏初始化

void ini(){

stage\_count=1;

memset(&ch,0,sizeof(ch\_t));

memset(enemy,0,sizeof(enemy\_t)\*ENEMY\_MAX);

memset(lazer,0,sizeof(lazer\_t)\*LAZER\_MAX);

memset(enemy\_order,0,sizeof(enemy\_order\_t)\*ENEMY\_ORDER\_MAX);

memset(shot,0,sizeof(shot\_t)\*SHOT\_MAX);

memset(cshot,0,sizeof(cshot\_t)\*CSHOT\_MAX);

memset(effect,0,sizeof(effect\_t)\*EFFECT\_MAX);

memset(del\_effect,0,sizeof(del\_effect\_t)\*DEL\_EFFECT\_MAX);

memset(&bom,0,sizeof(bom\_t));

memset(&bright\_set,0,sizeof(bright\_set\_t));

memset(&dn,0,sizeof(dn\_t));

memset(&boss,0,sizeof(boss\_t));

memset(child,0,sizeof(child\_t)\*CHILD\_MAX);

memset(&stage\_title,0,sizeof(stage\_title\_t));

ch.x=FMX/2;

ch.y=FMY\*3/4;

ch.power=500;

stage\_title.appear\_cnt=stage\_title\_count[stage];

/\*弹幕的各种设定。在后面的章节中我们再来好好设定吧。\*/

boss.appear\_count[0]=2000;//中路Boss出现的时刻

for(int i=0;i<DANMAKU\_MAX;i++){//弹幕的各种HP

boss.set\_hp[i]=10000;

boss.hp\_max=10000;

}

for(int i=0;i<DANMAKU\_MAX;i++)//弹幕的各种背景种类

boss.back\_knd[i]=1;

/\*到此为止\*/

bright\_set.brt=255;//最开始亮度为最大

}

我们把这些处理试着写在calc.cpp中吧。

如果和出现计数值一致了的话，就登录。

那之后我们渐渐地进行变亮的处理，稍微保持一会儿又进行变暗的处理，如果到了时间点的话就结束显示。

这次，由于我们对这个图像的使用仅仅是瞬间，因此没有必要作为让图像数据在内存上一直保存到游戏结束。

因此，我们只需要在登录的时候读入，在不需要它的时候释放它。

----在 calc.cpp 中进行以下追加 ----

#include "../include/GV.h"

//Title处理

void calc\_stage\_title(){

if(stage\_title.appear\_cnt==stage\_count){//如果现在是开始显示的时刻的话

stage\_title.flag=1;

stage\_title.img = LoadGraph("../dat/img/board/subject0.png");

}

if(stage\_title.flag==0)return;

if(stage\_title.cnt<128)//在最初的128次计数中逐渐变量

stage\_title.brt+=2;

if(stage\_title.cnt>128+128)//逐渐变暗

stage\_title.brt-=2;

if(stage\_title.cnt>128+128+128){//结束

stage\_title.cnt=0;

stage\_title.brt=0;

stage\_title.flag=0;

DeleteGraph(stage\_title.img);//释放图像

return ;

}

stage\_title.cnt++;

//强制结束的时候需要释放图像

}

void calc\_main(){

calc\_stage\_title();//Title处理

}

----在 graph.cpp 中追加以下红字部分 ----

//Title的显示

void graph\_stage\_title(){

if(stage\_title.flag>0){

SetDrawBlendMode( DX\_BLENDMODE\_ALPHA, stage\_title.brt );

DrawGraph(120+FX+dn.x,10+FY+dn.y,stage\_title.img,TRUE);

SetDrawBlendMode( DX\_BLENDMODE\_NOBLEND, 0 );

}

}

void graph\_main(){

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_back\_main();//背景显示main

graph\_effect(0);//敌人的消灭特效

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_effect(4);//决死的特效

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_child();

graph\_boss();

graph\_enemy();//敌人的绘制

graph\_cshot();//自机射击的绘制

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_ch();//自机的绘制

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_lazer();//激光的绘制

graph\_bullet();//子弹的绘制

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_effect(1);//Boom的特效

graph\_effect(2);//Boom线的特效

graph\_effect(3);// Boom角色的特效

graph\_stage\_title();//Title的显示

graph\_board();//面板的绘制

graph\_develop();

}

----在 main.cpp的main函数中进行以下红字部分变更 ----

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nCmdShow){

ChangeWindowMode(TRUE);//窗口模式

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初始化与里表面化

while(ProcessLoop()==0){//主循环

music\_ini();

switch(func\_state){

case 0://只在游戏开始时进行的处理

load(); //读入数据

first\_ini();//最开始的初始化

func\_state=99;

break;

case 99://STG开始前进行的初始化

ini();

load\_story();

func\_state=100;

break;

case 100://一般处理

calc\_ch(); //角色计算

ch\_move(); //控制角色的移动

cshot\_main();//自机射击main

enemy\_main();//敌人处理main

boss\_shot\_main();

shot\_main(); //射击main

out\_main(); //碰撞计算

effect\_main();//特效main

calc\_main();//游戏Title显示计算

graph\_main();//绘制main

if(boss.flag==0)

stage\_count++;

break;

default:

printfDx("错误的func\_state\n");

break;

}

music\_play();

if(CheckStateKey(KEY\_INPUT\_ESCAPE)==1)break;//按下ESC键则跳出徐U你还

ScreenFlip();//里外画面翻转

count++;

}

DxLib\_End();//DX Library终止处理

return 0;

}

----在 load.cp中进行以下红字变更 ----

void load\_story(){

int n,num,i,fp;

char fname[32]={"../dat/csv/38章/storyH0.csv"};

int input[64];

char inputc[64];

运行结果

无意中也修改了Excel数据，不过由于Stage的制作将会在后面进行，因此现在请随便看一下吧。所做的修改也不过是为了像真正的Stage开始那样随便改动了一下而已。

## 第三十九章 让敌人掉落道具吧

这次我们来让敌人掉落道具吧。

在实现和道具相关的内容的时候，我们也和平常一样，先定义结构体、声明变量，然后进行“登录/计算/绘制”这三个标准步骤。☆1

不过和道具相关的实现中增加了这些内容之后，就有必要进行和自机的碰撞判定计算以及和道具接触时的状态变化的处理了。

因此，我们试着在这一章节中先来制作“☆1”的部分。

那么我们来定义结构体。

另外如果我们在往收物线（Borderline）上方移动的话，就可以自动地收分了。

这意味着道具的状态会改变。

因此，现在我们试着在一般的结构体的之外再追加“state”变量吧。

初始化和图像的加载也先写出来。

----在 define.h 中进行以下追加 ----

//道具显示的最大数量

#define ITEM\_MAX 100

//获得道具的收物线

#define ITEM\_GET\_BORDER\_LINE 100

//道具的收取范围

#define ITEM\_INDRAW\_RANGE 70

----在 struct.h 中进行以下追加 ----

//道具的结构体

typedef struct{

//flag、计数器、种类、状态

int flag,cnt,knd,state;

//速度、坐标、大小

double v,x,y,r;

}item\_t;

----在 GV.h 中进行以下追加----

GLOBAL int img\_item[6][2];//道具的图像

GLOBAL item\_t item[ITEM\_MAX];//道具

----在 ini.cpp 的 ini() 中进行以下追加 ----

memset(item,0,sizeof(item\_t)\*ITEM\_MAX);

----在 load.cpp 的 load() 中进行以下追加 ----

LoadDivGraph( "../dat/img/item/p0.png" , 2 , 2 , 1 , 35 , 35 , img\_item[0] ) ;

LoadDivGraph( "../dat/img/item/p1.png" , 2 , 2 , 1 , 35 , 35 , img\_item[1] ) ;

LoadDivGraph( "../dat/img/item/p2.png" , 2 , 2 , 1 , 15 , 15 , img\_item[2] ) ;

LoadDivGraph( "../dat/img/item/p3.png" , 2 , 2 , 1 , 35 , 35 , img\_item[3] ) ;

LoadDivGraph( "../dat/img/item/p4.png" , 2 , 2 , 1 , 35 , 35 , img\_item[4] ) ;

LoadDivGraph( "../dat/img/item/p5.png" , 2 , 2 , 1 , 35 , 35 , img\_item[5] ) ;

----在 load.cpp 中变更以下红字部分 ----

void load\_story(){

int n,num,i,fp;

char fname[32]={"../dat/csv/39章/storyH0.csv"};

int input[64];

char inputc[64];

好的，预先准备已经做好了。

那么首先我们来登录吧。

道具出现的时刻为消灭敌人或者自机被击毁的时候。

那么在敌人被击毁的时候我们调用enter\_item函数使得道具出现。

至于会出现什么道具，这个信息已经先保存在敌人的item\_n中了。

由于我们实在Excel数据中决定道具的掉落的，因此去确认Excel文件。

当第1个道具在敌人的位置一次出现并且有多个将要出现的时候，那么从第2个道具开始就在适当地在离开一段距离的地方出现。

----在 out.cpp 中进行红字部分变更 ----

int serch\_item(){

for(int i=0;i<ITEM\_MAX;i++)

if(item[i].flag==0)

return i;

return -1;

}

//道具登录

//道具　0:小power 1:小点 2:弹点 3:小金 4:大power 5:大金

void enter\_item(double x, double y, int item\_n[], int num){//x,y,道具的种类，数量

int k;

double r[6]={0.6,0.6,1.0,0.6,1.0,1.0};//dat/img/item的图像放大率

for(int i=0;i<num;i++){//1个敌人掉落的道具最大数为6个

if(item\_n[i]!=-1){//如果在Excel中指定的道具是-1（空）的话就终止

if((k=serch\_item())!=-1){//查询可以登录的编号

item[k].flag=1;

item[k].v =-3.5; //速度

item[k].cnt =0;

item[k].state=0;

item[k].x =x;

item[k].y =y;

if(i>0){//如果不止一个的话就适当地分散一下

item[k].x+=rang(40);

item[k].y+=rang(40);

}

item[k].knd =item\_n[i];//指定了的道具掉落

item[k].r =r[item[k].knd];

}

}

}

}

//敌人掉落的道具

void enter\_enemy\_item(int s){

enter\_item(enemy[s].x,enemy[s].y,enemy[s].item\_n, sizeof(enemy[s].item\_n)/sizeof(int));

}

//角色掉落的道具

void enter\_char\_item(){

int item\_n[4]={4,4,4,4};

enter\_item(ch.x, ch.y, item\_n, 4);

}

//决定敌人是否被击毁

void enemy\_death\_judge(int s){

int i;

se\_flag[8]=1;//敌人被击中的声音

if(enemy[s].hp<0){//如果敌人的HP小于0

enemy[s].flag=0;//消灭敌人

se\_flag[1]=1;//敌人的击毁声

enter\_del\_effect(s);

enter\_enemy\_item(s);//s号敌人掉落道具（39章）

for(i=0;i<SHOT\_MAX;i++){//敌人总数次

----在 char.cpp 中进行以下变更 ----

extern void enter\_char\_item();//(39章)

void calc\_ch(){

if(ch.flag==1){//正在处理决死

bright\_set.brt=80;//变暗

if(ch.cnt>20){//0.33秒处理决死

ch.flag =2; //1:正在处理决死　2:自机被击毁正在上浮

ch.cnt =0;

bright\_set.brt=255;

}

}

if(ch.cnt==0 && ch.flag==2){//当前瞬间被击毁的话

enter\_char\_item();//登录自机的道具(39章)

ch.x=FMX/2;//设置坐标

ch.y=FMY+30;

ch.mutekicnt++;//进入无敌状态

}

接下来是计算和绘制。

道具的状态如果是0的话那么就是普通地降落。

如果是1的话就是被收取。

如果自机在ITEM\_GET\_BORDER\_LINE以上的话那么item的状态就变成1以被收取。

低速移动中如果道具在自机的半径ITEM\_INDRAW\_RANGE以内的话状态即便为0也会收取一部分。

普通下落的时候我们只要在y方向将坐标加上速度分量就行了。

吸收的时候，自机与道具的角度使用atan2来求得，然后像平常那样用sin、cos来计算就行了。

绘制的时候我们按照“基础1，基础2，文字”的顺序绘制3次。

基础1是等倍率旋转绘制，而基础2是0.8返回旋绘制。

----在 calc.cpp 中进行以下红字部分追加/变更 ----

//道具的收取处理

void calc\_item\_indraw(int i){

double v = item[i].state ? 8 : 3;//state如果是1的话速度为8否则为3

double angle=atan2(ch.y-item[i].y,ch.x-item[i].x);//计算与自机的角度

item[i].x += cos(angle)\*v;

item[i].y += sin(angle)\*v;

}

//计算道具的移动等

void calc\_item(){

for(int i=0;i<ITEM\_MAX;i++){

if(item[i].flag>0){

if(item[i].state==0)

if(ch.y<ITEM\_GET\_BORDER\_LINE)//如果是收取状态的话

item[i].state=1;

if(item[i].state==0){//如果是普通状态的话

double x=ch.x-item[i].x,y=ch.y-item[i].y;

//如果在低速状态中的自机附近的话

if(CheckStatePad(configpad.slow)>0 &&

x\*x+y\*y<ITEM\_INDRAW\_RANGE\*ITEM\_INDRAW\_RANGE){

calc\_item\_indraw(i);//收取一部分

}

else{

if(item[i].v<2.5)//增加速度

item[i].v+=0.06;

item[i].y+=item[i].v;//移动

}

}

else{//如果是收取状态的话

calc\_item\_indraw(i);

}

item[i].cnt++;

if(item[i].y>FMY+50)

item[i].flag=0;

}

}

}

void calc\_main(){

calc\_stage\_title();

calc\_item();

}

----在 graph.cpp 中进行以下红字部分的变更/追加 ----

void graph\_item(){

int i;

for(i=0;i<ITEM\_MAX;i++){

if(item[i].flag==1){

DrawRotaGraphF(item[i].x+FX+dn.x,item[i].y+FY+dn.y,

item[i].r,PI2\*(count%120)/120,img\_item[item[i].knd][1],TRUE);

DrawRotaGraphF(item[i].x+FX+dn.x,item[i].y+FY+dn.y,

item[i].r\*0.8,-PI2\*(count%120)/120,img\_item[item[i].knd][1],TRUE);

DrawRotaGraphF(item[i].x+FX+dn.x,item[i].y+FY+dn.y,

item[i].r,0,img\_item[item[i].knd][0],TRUE);

}

}

}

void graph\_main(){

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_back\_main();//背景绘制main

graph\_effect(0);//敌人消灭特效

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_effect(4);//决死的特效

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_child();

graph\_item();//道具的绘制

graph\_boss();

graph\_enemy();//敌人的绘制

graph\_cshot();//自机射击的绘制

运行结果

由于现在我们还没有完成道具的碰撞判定，因此与道具接触之后什么都没有发生，并且收取了的道具也仅仅是跟着自机跑而已。

在下一章中我们完成这些东西。

## 第四十章 追加道具获得的处理吧

事到如今我有一种其实完全没有必要把章节一分为二的想法了。（汗）

本章是上一章节的继续，我们来试着实现道具的接触判定以及接触的时候由道具引起的状态变化吧。

这一章中要追加的处理非常少量。

不管怎么说我们先试着定义碰撞判定，然后当道具那个范围内的时候进行状态变化吧。

碰撞判定在英语中称为Hitbox，大家不妨作为小知识记住吧。b

----在 define.h 中进行以下追加 ----

//道具的碰撞判定

#define ITEM\_HIT\_BOX 20

----在 out.cpp 中追加以下红字部分 ----

//与道具接触

//道具　0:小power 1:小点 2:弹点 3:小金 4:大power 5:大金

void ch\_and\_item(){

for(int i=0;i<ITEM\_MAX;i++){

if(item[i].flag>0){

double x=item[i].x-ch.x,y=item[i].y-ch.y;

if(x\*x+y\*y<ITEM\_HIT\_BOX\*ITEM\_HIT\_BOX){//如果在碰撞判定内的话

switch(item[i].knd){//根据种类变更

case 0: ch.power+=3; break;

case 1: ch.point+=1; break;

case 2: ch.score+=1; break;

case 3: ch.money+=1; break;

case 4: ch.power+=50;break;

case 5: ch.money+=10;break;

}

if(ch.power>500)ch.power=500;

if(ch.point>9999)ch.point=9999;

if(ch.money>999999)ch.money=999999;

if(ch.score>999999999)ch.score=999999999;

item[i].flag=0;

se\_flag[34]=1;//获取音

}

}

}

}

//碰撞判定main

void out\_main(){

cbom\_and\_enemy();//让敌人承受Boom的伤害

cshot\_and\_enemy();//自机射击与敌人的处理

enemyshot\_and\_ch();//敌人射击与自机的处理

ch\_and\_item();//道具接触处理

}

----在 load.cpp 的load函数中进行以下追加 ----

sound\_se[34]=LoadSoundMem("../dat/se/item\_get.wav");

在进行变化状态而增加值的时候，不妨先像上面那样决定好上限值吧。

我曾经认为不可能会出现能得到9亿点score的人（译者注：指在四圣龙神录中），因此我没有设置一个上限值，也正是因为这个原因偏偏有人就超过了上限值而引起了游戏的BUG。

像score这种不可能为负数的值我们将其设置为unsigned int虽然也是一个解决方案，但是如果像上面那样设定的话就没有问题了。

我们将会在下一章中进行说明，这个上限值是有它的意义的。

运行结果

## 番外篇第一章 ~兴致来潮追加的小游戏~

这一章您继不继续看下去都是没关系的。即便不阅读对以后也没有任何障碍。

刚才我试着想象了一下39章中的场景——那时我们还无法获取道具，突然想到如果有这么一个游戏——自机在低速移动中暂时收取道具，同时在下方放置一个吊篮用来把收取到的道具放进去的话，不是很像一个小游戏么？

角色不能在最下方。也不能下降到画面的3/4左右的位置。

这是一个“漂亮地把道具放进篮子里来挣分吧”的游戏。

要想把这个想法好好地作成小游戏的话，由于在低速状态下连续移动在某种程度上会比较轻松，因此对于道具暂时保留在自机上的时间以及最大个数的设定非常有必要反复调整。

运行结果

就请不要吐槽运行示例中的非常糟糕的实现了。w

在这一章中变更的内容与其他章节中毫无关系。

“番外1章”的工程也被放进了打包的文件中了，如果您有兴趣的话可以看看。

变更了的地方都用“//(b1)”标注了出来，所以您可以直接搜索找到变更的地方。

说不定下一作的龙神录的迷你游戏中就会加入这样子的游戏呢。W

## 第四十一章 试着进行Score面板的显示吧

本章中我们试着进行右侧面板的显示吧。

本章的重点在于，使用图像进行多位数的显示。

虽然普通的数字的显示我们用%d来表示的话不用考虑位数就可以显示了，但是本章中我们将要组合0-9的10种图片来表示多位数。

在这里“余数”的概念就很重要了。只要我们表示出用10去除那个数得到的余数就能够表示出那1位数字了。

假如现在我们有“29”这个数字，同时0-9分别对应第0-9号的图像编号。

首先我们绘制第1位数字。

考虑

int score = 29;

那么29 % 10就是9了呢。那么我们就显示编号为9的图像。

稍微将显示坐标往左挪挪，之后

score /= 10;

然后，由于

socre % 10

相当于

2 % 10

因此我们在这里显示编号为2的图像。

这样一看，我们就可以显示29了。

就像这样，用10去除然后显示余数，再用10去除显示余数，再用10去除……

直至数归零为止，就可以完成显示了。

我们试着在graph.cpp中进行这个处理吧。

另外，由于从现在开始我们使用了颜色，因此我们定义颜色的变量。

我们标上(41)，用以表示这个处理是在第41章中追加的。

----在 GV.h 中进行以下追加 ----

GLOBAL int img\_num[3][12];//数字的图像(41)

//其它变量

GLOBAL int color[20];//颜色(41)

----在 struct.cpp 中进行红字部分变更 ----

//与角色相关的结构体

typedef struct{

int flag; //flag

int cnt; //计数器

int power; //power

int point; //point

int score; //score

int num; //残机数

int mutekicnt; //无敌状态与计数器

int shot\_mode; //射击模式

int money; //钱

int img;

int slow; //是否缓慢移动

double x,y; //坐标

int shot\_cnt; //射击计数

int graze; //擦弹 (41)

}ch\_t;

----在 ini.cpp 的 ini() 函数中进行以下变更/追加 ----

ch.power=0;//初始化power(41)

ch.num=5;//初始化残机数(41)

----在 load.cpp 的load函数中进行以下追加 ----

LoadDivGraph( "../dat/img/num/2.png" , 10 , 10 , 1 , 16 , 18 , img\_num[0] ) ;//(41)

//(41)

color[0] = GetColor(255,255,255);//白色

color[1] = GetColor( 0, 0, 0);//黑色

color[2] = GetColor(255, 0, 0);//红色

color[3] = GetColor( 0,255, 0);//绿色

color[4] = GetColor( 0, 0,255);//蓝色

color[5] = GetColor(255,255, 0);//黄色

color[6] = GetColor( 0,255,255);//蓝绿色

color[7] = GetColor(255, 0,255);//紫色

----在 graph.cpp 中追加以下红字部分 ----

//显示Score面板(41)

void graph\_board\_states(){

int i;

int score=ch.score;

int power=ch.power;

int graze=ch.graze;

int point=ch.point;

int money=ch.money;

for(i=0;i<9;i++){//显示Score

DrawRotaGraph(625-15\*i,30,1.0f,0.0f,img\_num[0][score%10],TRUE);

DrawRotaGraph(625-15\*i,50,1.0f,0.0f,img\_num[0][score%10],TRUE);

score/=10;

}

for(i=0;i<ch.num;i++)//显示残机数

DrawGraph(499+12\*i,63,img\_etc[8],TRUE);

DrawRotaGraph(547,91,0.9f,0.0f,img\_num[0][power%10],TRUE);power/=10;

DrawRotaGraph(536,91,0.9f,0.0f,img\_num[0][power%10],TRUE);power/=10;

DrawRotaGraph(513,91,1.0f,0.0f,img\_num[0][power%10],TRUE);

DrawString(522,82,".",color[0]);//チョン

for(i=0;i<6;i++){//擦弹显示

DrawRotaGraph(578-14\*i,111,1.0f,0.0f,img\_num[0][graze%10],TRUE);

graze/=10;

}

for(i=0;i<4;i++){//point表示

DrawRotaGraph(550-14\*i,131,1.0f,0.0f,img\_num[0][point%10],TRUE);

point/=10;

}

for(i=0;i<6;i++){//金钱显示

DrawRotaGraph(578-14\*i,154,1.0f,0.0f,img\_num[0][money%10],TRUE);

money/=10;

}

}

//面板绘制

void graph\_board(){

DrawGraph( 0, 0,img\_board[10],FALSE);

DrawGraph( 0, 16,img\_board[11],FALSE);

DrawGraph( 0,464,img\_board[12],FALSE);

DrawGraph(416, 0,img\_board[20],FALSE);

graph\_board\_states();//(41)

}

在前面的章节中决定的上限就是在这里所能显示的最大值呢。

运行结果

## 第四十二章 试着制作闪屏（Flash）特效吧

本章的内容并不是特别重要。由于考虑到今后自己可能会使用这个特效，因此就实现了。

那就是“Flash”。一瞬间让画面变白进行闪屏。

是的，这一次我们也按照老规矩来。

定义结构体、声明变量，然后进行“登录、计算、绘制”这一过程。

我们制作两种闪屏。

种类0的闪屏效果是在指定时间内逐渐变亮然后再逐渐变暗。

种类1的闪屏效果是一开始突然屏幕全白，然后在指定的时间内渐渐变暗——这是一般的闪屏类型。

----在 GV.h 中进行以下追加 ----

GLOBAL flash\_t flash;//(42)

----在 struct.cpp 中进行以下追加 ----

//闪屏(42)

typedef struct{

//flag、计数器、闪屏的种类、在多少计数值中显示

int flag,cnt,knd,to\_cnt;

//亮度

double brt;

}flash\_t;

----在 ini.cpp 的 ini() 函数中进行以下变更/追加 ----

boss.appear\_count[0]=20;

----在 boss\_shotH.cpp 中进行以下追加 ----

//闪屏样例(42)

void boss\_shot\_bulletH011(){

#define TM009 420

#define DIST 60

int t=boss\_shot.cnt%TM009,t2=boss\_shot.cnt;

if(t2==60){

flash.flag=1;

flash.knd=0;//逐渐变量、逐渐变暗

flash.to\_cnt=30;//30カウントで変化 在30次计数中变化

flash.cnt=0;

}

if(t2==180){

flash.flag=1;

flash.knd=1;//闪！

flash.to\_cnt=90;// 90次计数中变化

flash.cnt=0;

}

if(t2==260){

flash.flag=1;

flash.knd=0;//逐渐变亮，逐渐变暗

flash.to\_cnt=15;//1在15次计数中变化

flash.cnt=0;

}

if(t2==380){

flash.flag=1;

flash.knd=1;//闪！

flash.to\_cnt=30;//在30次计数中变化

flash.cnt=0;

}

}

----在 func.h 追加变更红字部分 ----

extern void boss\_shot\_bulletH011();//(42)

void (\*boss\_shot\_bullet[DANMAKU\_MAX])() =

{

boss\_shot\_bulletH011,

boss\_shot\_bulletH010,

boss\_shot\_bulletH009,

boss\_shot\_bulletH008,

boss\_shot\_bulletH007,

boss\_shot\_bulletH006,

boss\_shot\_bulletH005,

boss\_shot\_bulletH004,

boss\_shot\_bulletH003,

boss\_shot\_bulletH002,

boss\_shot\_bulletH001,

boss\_shot\_bulletH000,

boss\_shot\_bulletH000,

boss\_shot\_bulletH000,

boss\_shot\_bulletH000,

boss\_shot\_bulletH000,

boss\_shot\_bulletH000,

};

----在 calc.cpp 中追加以下红字部分 ----

//闪屏(42)

void calc\_flash(){

if(flash.flag>0){

if(flash.knd==0){

if(flash.cnt==0)flash.brt=0;

if(flash.cnt<flash.to\_cnt)

flash.brt+=255.0/flash.to\_cnt;

else

flash.brt-=255.0/flash.to\_cnt;

}

if(flash.knd==1){

if(flash.cnt==0)flash.brt=255;

flash.brt-=255.0/flash.to\_cnt;

}

flash.cnt++;

if(flash.knd==0){

if(flash.cnt>flash.to\_cnt\*2)

flash.flag=0;

}

if(flash.knd==1){

if(flash.cnt>flash.to\_cnt)

flash.flag=0;

}

}

}

void calc\_main(){

calc\_stage\_title();

calc\_item();

calc\_flash();//闪屏(42)

}

----在 graph.cpp 中进行以下变更追加 ----

//闪屏绘制(42)

void graph\_flash(){

if(flash.flag>0){

SetDrawBlendMode( DX\_BLENDMODE\_ALPHA, (int)(flash.brt) );

DrawBox(FX,FY,FX+FMX,FY+FMY,color[0],TRUE);

SetDrawBlendMode( DX\_BLENDMODE\_NOBLEND, 0 );

}

}

void graph\_main(){

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_back\_main();//背景绘制main

graph\_effect(0);//敌人击毁特效

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_effect(4);//决死特效

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_child();

graph\_item();//道具绘制

graph\_boss();

graph\_enemy();//敌人的绘制

graph\_cshot();//自机射击的绘制

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_ch();//自机的绘制

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_lazer();//激光的绘制

graph\_bullet();//子弹的绘制

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_effect(1);//Boom的特效

graph\_effect(2);//Boom线的特效

graph\_effect(3);//Boom角色的特效

graph\_stage\_title();//Title的显示

graph\_board();//面板的绘制

graph\_flash();//闪屏的绘制

graph\_develop();

}

运行结果

## 番外篇第二章~与音乐共舞的弹幕

（译者注：这一章简直高能！如果您能上niconico的话请务必看看本章弹幕的视频！）

这章简直要疯了。

首先请您看一下本章制作出来弹幕的视频。

请务必将音箱音量调高来听。

（这是1500kbps的高画质版。虽然有点大，但是在Zoome上都可以观看。）（译者注：Zoome已经在2011年8月31日正式停止运营）

（niconico动画在这里）

采纳了“不妨把按照自己兴趣做出来的弹幕也开源大家看看如何”的建议，因此我试着在本次第43章中这样做了（译者注：后改名为番外篇第二章），不过代码看起来不是那么适合阅读就是了^^;。

这就是写好的弹幕函数（只有一个函数）。

呀~我可没说您一定要把它看完哦……

为了对别的文件不产生影响因此我自行定义了一些变量，因此请不要以此作为参考哦。（啥？）

--- 编写新的文件 boss\_shotElse.cpp，进行以下追加 ----

#include "../include/GV.h"

extern int search\_boss\_shot();

extern int search\_lazer();

extern int search\_child();

extern double bossatan2();

extern double bossatan3(int k,double x,double y);//指定坐标和子弹所成的角

extern double bossatan4(int k);//制定子弹和自机所成的角

extern int move\_boss\_pos(double x1,double y1,double x2,double y2,double dist, int t);

extern void input\_phy\_pos(double x,double y,int t);

extern void input\_lphy(lazer\_t \*laz, int time, double angle);

extern int search\_boss\_shot\_n(int from, int to);//返回指定的空编号

//曲子的节奏（rhythm）数据

int tm12[1000]={

459,509,559,605,653,703,751,798,848,893,941,990,1038,1086,1134,1183,1231,1277,1325,1374,1420,1469,1520,

1571,1620,1679,1738,1814,1837,1858,1878,1897,1915,1934,1955,1974,1996,2015,2037,2057,2076,2097,2117,2137,

2157,2176,2198,2218,2239,2258,2278,2297,2318,2337,2358,2380,2400,2421,2440,2460,2481,2500,2520,2540,2559,

2579,2599,2621,2640,2660,2680,2701,2721,2742,2762,2782,2802,2822,2842,2863,2884,2904,2924,2944,2964,2984,

3005,3025,3045,3065,3085,3105,3124,3144,3165,3226,3246,3265,3286,3348,3368,3387,3408,3469,3490,3508,3528,

3592,3611,3630,3650,3710,3730,3749,3770,3832,3852,3871,3893,3913,3933,3954,3973,3993,4014,4075,4096,4116,

4136,4156,4174,4195,4215,4237,4258,4278,4298,4318,4337,4358,4378,4398,4418,4438,4458,4477,4498,4520,4540,

4559,4580,4601,4622,4642,4662,4680,4701,4720,4740,4760,4780,4801,4822,4842,4863,4883,4903,4924,4944,4965,

4985,5005,5024,5043,5064,5086,5106,5127,5150,5170,5188,5208,5227,5248,5267,5288,5308,5328,5404

};

//音乐的视觉效果用的数据

#define VMX 52

#define VMY 30

BYTE signal[10][VMX]={

{2,3,4,5,6,5,4,3,2,3,4,3,2,4,6,5,4,3,2,3,4,5,6,5,4,7,10,7,4,4,4,6,8,6,4,3,2,5,8,7,6,4,2,4,6,7,8,6,4,6,8,8},

{4,5,6,8,10,7,4,6,8,5,2,4,6,8,10,9,8,7,6,7,8,9,10,11,12,8,4,4,4,6,8,6,4,7,10,8,6,7,8,9,10,7,4,4,4,5,6,5,4,5,6,6,},

{8,7,6,4,2,3,4,6,8,6,4,4,4,5,6,6,6,8,10,7,4,3,2,3,4,7,10,9,8,6,4,5,6,8,10,7,4,6,8,9,10,11,12,10,8,7,6,6,6,5,4,4,},

{6,5,4,7,10,7,4,4,4,3,2,6,10,7,4,3,2,4,6,5,4,5,6,8,10,7,4,3,2,3,4,3,2,6,10,6,2,3,4,3,2,3,4,3,2,5,8,5,2,4,6,6,},

{2,3,4,7,10,9,8,5,2,3,4,3,2,3,4,6,8,9,10,6,2,4,6,8,10,8,6,5,4,5,6,5,4,7,10,7,4,3,2,2,2,5,8,7,6,5,4,3,2,3,4,4,},

{2,3,4,5,6,8,10,11,12,15,18,13,8,7,6,7,8,11,14,18,22,20,18,20,22,21,20,18,16,17,18,20,22,17,12,15,18,16,14,15,16,13,10,8,6,5,4,6,8,5,2,2,},

{2,6,10,7,4,7,10,18,26,18,10,9,8,7,6,9,12,17,22,22,22,16,10,9,8,6,4,7,10,12,14,19,24,19,14,13,12,9,6,8,10,14,18,22,26,23,20,16,12,8,4,4,},

{4,7,10,14,18,13,8,14,20,18,16,17,18,14,10,15,20,14,8,8,8,7,6,7,8,9,10,13,16,15,14,12,10,9,8,9,10,14,18,12,6,8,10,9,8,13,18,12,6,5,4,4,},

{6,8,10,11,12,16,20,17,14,11,8,6,4,5,6,8,10,14,18,14,10,14,18,14,10,12,14,18,22,19,16,13,10,8,6,7,8,9,10,8,6,7,8,14,20,17,14,15,16,15,14,14,},

{2,5,8,12,16,10,4,5,6,6,6,11,16,11,6,11,16,10,4,7,10,14,18,17,16,13,10,8,6,8,10,8,6,6,6,7,8,15,22,15,8,7,6,6,6,5,4,9,14,9,4,4,},

};

//视觉效果用的结构体

typedef struct{

int cnt;

BYTE num[VMX];

}visual\_effect\_t;

visual\_effect\_t visual\_effect;

//视觉效果的正方形的边长

#define VWIDE 8

void boss\_shot\_bulletElse(){

int i,j,k,s,n,t=boss\_shot.cnt,t2=0;

static int num,sound\_handle,base\_col;

static double lzv,base\_angle;

if(t==0){

//如果启动注释了的这三行的话，就从num所设定音乐的地方开始

//num是tm12中设定的音乐节奏的数据，一直到188。

// num=20;

// t=t2=tm12[num];

boss.endtime=tm12[188]+800;

lzv=0;

base\_col=0;

boss.graph\_flag=1;

sound\_handle=LoadSoundMem("../dat/music/Rage.wav");

// SetSoundCurrentTime( (int)(t2\*16.666666-100) , sound\_handle);

PlaySoundMem(sound\_handle, DX\_PLAYTYPE\_BACK, FALSE);

boss\_shot.cnt=t2;

base\_angle=-PI/2;

memset(&visual\_effect,0,sizeof(visual\_effect\_t));

}

if(t==tm12[num]){//计算节奏

num++;

}

//取得与音乐同步的处理

if(t%60==59 && t<tm12[188]){

int gettm=GetSoundCurrentTime(sound\_handle);

int nowflame=(int)(t\*16.6666);

// printfDx("%d\n",gettm-nowflame);

if(abs(gettm-nowflame)>100){

StopSoundMem(sound\_handle);

SetSoundCurrentTime( (int)(t\*16.666666) , sound\_handle);

PlaySoundMem(sound\_handle, DX\_PLAYTYPE\_BACK,FALSE);

// printfDx("调整后%d\n",GetSoundCurrentTime(sound\_handle));

}

}

//最初的激光乱射~

if(t<180){

for(i=0;i<2;i++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 0;//子弹的颜色

boss\_shot.bullet[k].x = FMX/2;//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 2;//子弹的种类

boss\_shot.bullet[k].angle = PI/120\*t;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 1.5;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 4;//子弹的颜色

boss\_shot.bullet[k].x = FMX/2;//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 2;//子弹的种类

boss\_shot.bullet[k].angle = (PI+PI/240)-PI/120\*t;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 2;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

}

//接下来是蓝色的蝴蝶从上面飘下来

if(t==240){

for(j=0;j<15;j++){

for(i=0;i<30;i++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 1;//子弹的颜色

boss\_shot.bullet[k].x = FMX/2;//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 11;//子弹的种类

boss\_shot.bullet[k].angle = PI/30\*i;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 3-0.2\*j;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

}

}

//闪屏

if(t==tm12[0]){

flash.flag=1;

flash.knd=1;//闪！

flash.to\_cnt=90;//在90次计数中变化

flash.cnt=0;

}

//和音乐一起涌上来的激光

if(t==tm12[0]+1){

{

k=0;

lazer[k].col = 0;//子弹的颜色

lazer[k].knd = 0;//子弹的种类

lazer[k].angle = PI;//角度

lazer[k].startpt.x= FMX;//坐标

lazer[k].startpt.y= 0;

lazer[k].flag = 1;

lazer[k].cnt = 0;

lazer[k].haba = 20;//幅度

lazer[k].state = 0;//状态

lazer[k].length = FMX+60;//激光的长度

lazer[k].hantei = 0.6;

lazer[k].lphy.conv\_flag=0;//旋转flag

lazer[k].lphy.conv\_base\_x=0;//旋转基准位置

lazer[k].lphy.conv\_base\_y=0;

lazer[k].lphy.conv\_x=0;//旋转元的位置

lazer[k].lphy.conv\_y=0;

input\_lphy(&lazer[k],0,0);//代入

}

{

k=1;

lazer[k].col = 0;//子弹的颜色

lazer[k].knd = 0;//子弹的种类

lazer[k].angle = 0;//角度

lazer[k].startpt.x= 0;//坐标

lazer[k].startpt.y= FMY;

lazer[k].flag = 1;

lazer[k].cnt = 0;

lazer[k].haba = 20;//幅度

lazer[k].state = 0;//状态

lazer[k].length = FMX+60;//激光的长度

lazer[k].hantei = 0.6;

lazer[k].lphy.conv\_flag=0;//旋转flag

lazer[k].lphy.conv\_base\_x=0;//旋转基准位置

lazer[k].lphy.conv\_base\_y=0;

lazer[k].lphy.conv\_x=0;//旋转元的位置

lazer[k].lphy.conv\_y=0;

input\_lphy(&lazer[k],0,0);//代入

}

}

//对角线方向飞散

if(tm12[1]-15<=t && t<=tm12[7]){

for(i=1;i<=8;i++){

if(t==tm12[i]){

for(s=0;s<2;s++){

for(j=0;j<3;j++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 2;//子弹的颜色

boss\_shot.bullet[k].x = boss.x;//坐标

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 0;//子弹的种类

boss\_shot.bullet[k].angle = PI\*0.75+rang(PI/10)+PI\*s+PI/2\*(num%2);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 6-0.7\*j;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

for(j=0;j<5;j++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 1;//子弹的颜色

boss\_shot.bullet[k].x = boss.x;//坐标

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 1;//子弹的种类

boss\_shot.bullet[k].angle = PI\*0.75+rang(PI/8)+PI\*s+PI/2\*(num%2);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 5.5-0.6\*j;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

for(j=0;j<7;j++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 0;//子弹的颜色

boss\_shot.bullet[k].x = boss.x;//坐标

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 11;//子弹的种类

boss\_shot.bullet[k].angle = PI\*0.75+rang(PI/6)+PI\*s+PI/2\*(num%2);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 5-0.6\*j;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

for(j=0;j<15;j++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 0;//子弹的颜色

boss\_shot.bullet[k].x = boss.x;//坐标

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 10;//子弹的种类

boss\_shot.bullet[k].angle = PI\*0.75+rang(PI/4)+PI\*s+PI/2\*(num%2);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 4-0.2\*j;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

}

}

}

for(i=1;i<=8;i++){

if(tm12[i]-15<=t && t<=tm12[i]){

lzv+=0.2;

lazer[0].startpt.y+=lzv;

lazer[1].startpt.y-=lzv;

}

if(t==tm12[i])

lzv=0;

}

}

//闪屏！

if(t==tm12[7]){

flash.flag=1;

flash.knd=1;//闪屏！

flash.to\_cnt=90;//在90次计数中进行

flash.cnt=0;

lazer[0].flag=0;

lazer[1].flag=0;

}

//一大波激光

if(t==tm12[7]){

for(i=0;i<6;i++){

k=i;

lazer[k].col = 1;//子弹的颜色

lazer[k].knd = 0;//子弹的种类

lazer[k].angle = PI/2+PI\*i;//角度

lazer[k].startpt.x= 160\*i+40;//坐标

lazer[k].startpt.y= FMY\*(i%2);

lazer[k].flag = 1;

lazer[k].cnt = 0;

lazer[k].haba = 2;//幅度

lazer[k].state = 0;//状态

lazer[k].length = FMY+60;//激光的长度

lazer[k].hantei = 0;

lazer[k].lphy.conv\_flag=0;//旋转flag

lazer[k].lphy.conv\_base\_x=0;//旋转基准位置

lazer[k].lphy.conv\_base\_y=0;

lazer[k].lphy.conv\_x=0;//旋转元的位置

lazer[k].lphy.conv\_y=0;

input\_lphy(&lazer[k],0,0);//代入

}

}

if(tm12[8]-15<=t && t<=tm12[14]){

for(i=8;i<=14;i++){

if(tm12[i]-15<=t && t<=tm12[i]){

lzv+=0.2;

for(j=0;j<6;j++){

lazer[j].haba+=0.7;

lazer[j].startpt.x-=lzv;

lazer[j].hantei = 0.6;

}

}

if(t==tm12[i])

lzv=0;

}

}

//对角线方向飞散

if(tm12[8]-15<=t && t<=tm12[14]){

for(i=8;i<=14;i++){

if(t==tm12[i]){

for(s=0;s<2;s++){

for(j=0;j<3;j++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 1;//子弹的颜色

boss\_shot.bullet[k].x = boss.x;//坐标

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 0;//子弹的种类

boss\_shot.bullet[k].angle = PI\*0.75+rang(PI/10)+PI\*s+PI/2\*(num%2);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 5-0.6\*j;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

for(j=0;j<5;j++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 0;//子弹的颜色

boss\_shot.bullet[k].x = boss.x;//坐标

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 1;//子弹的种类

boss\_shot.bullet[k].angle = PI\*0.75+rang(PI/8)+PI\*s+PI/2\*(num%2);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 4.5-0.5\*j;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

for(j=0;j<7;j++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 3;//子弹的颜色

boss\_shot.bullet[k].x = boss.x;//坐标

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 11;//子弹的种类

boss\_shot.bullet[k].angle = PI\*0.75+rang(PI/6)+PI\*s+PI/2\*(num%2);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 4-0.5\*j;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

for(j=0;j<15;j++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 1;//子弹的颜色

boss\_shot.bullet[k].x = boss.x;//坐标

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 10;//子弹的种类

boss\_shot.bullet[k].angle = PI\*0.75+rang(PI/4)+PI\*s+PI/2\*(num%2);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 3-0.1\*j;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

}

}

}

}

if(t==tm12[14]){

flash.flag=1;

flash.knd=1;//闪屏！

flash.to\_cnt=90;//在90次计数中变化

flash.cnt=0;

for(i=0;i<6;i++)

lazer[i].flag=0;

}

//激光

if(t==tm12[14]){

for(i=0;i<6;i++){

k=i;

lazer[k].col = i%2;//子弹的颜色

lazer[k].knd = 0;//子弹的种类

lazer[k].angle = PI\*i;//角度

lazer[k].startpt.x= FMX\*(i%2);//坐标

lazer[k].startpt.y= 160\*i;

lazer[k].flag = 1;

lazer[k].cnt = 0;

lazer[k].haba = 2;//幅度

lazer[k].state = 0;//状态

lazer[k].length = FMX+60;//激光的长度

lazer[k].hantei = 0;

lazer[k].lphy.conv\_flag=0;//旋转flag

lazer[k].lphy.conv\_base\_x=0;//旋转基准位置

lazer[k].lphy.conv\_base\_y=0;

lazer[k].lphy.conv\_x=0;//旋转元的位置

lazer[k].lphy.conv\_y=0;

input\_lphy(&lazer[k],0,0);//代入

}

}

if(tm12[15]-15<=t && t<=tm12[20]){

for(i=14;i<=20;i++){

if(tm12[i]-15<=t && t<=tm12[i]){

lzv+=0.2;

for(j=0;j<6;j++){

lazer[j].haba+=0.4;

lazer[j].startpt.y-=lzv;

lazer[j].hantei = 0.6;

}

}

if(t==tm12[i])

lzv=0;

}

}

//对角线方向飞散

if(tm12[15]-15<=t && t<=tm12[20]){

for(i=14;i<=20;i++){

if(t==tm12[i]){

for(s=0;s<2;s++){

for(j=0;j<1;j++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = (num+1)%2;//子弹的颜色

boss\_shot.bullet[k].x = FMX\*((s+num)%2);//坐标

boss\_shot.bullet[k].y = FMY\*(s);

boss\_shot.bullet[k].knd = 0;//子弹的种类

boss\_shot.bullet[k].angle = bossatan4(k);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 4-0.6\*j;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

for(j=0;j<3;j++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = num%2;//子弹的颜色

boss\_shot.bullet[k].x = FMX\*((s+num)%2);//坐标

boss\_shot.bullet[k].y = FMY\*(s);

boss\_shot.bullet[k].knd = 1;//子弹的种类

boss\_shot.bullet[k].angle = bossatan4(k)+rang(PI/28);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 3.5-0.6\*j;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

for(j=0;j<5;j++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 3\*(num%2);//子弹的颜色

boss\_shot.bullet[k].x = FMX\*((s+num)%2);//坐标

boss\_shot.bullet[k].y = FMY\*(s);

boss\_shot.bullet[k].knd = 11;//子弹的种类

boss\_shot.bullet[k].angle = bossatan4(k)+rang(PI/26);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 3-0.5\*j;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

for(j=0;j<10;j++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = num%2;//子弹的颜色

boss\_shot.bullet[k].x = FMX\*((s+num)%2);//坐标

boss\_shot.bullet[k].y = FMY\*(s);

boss\_shot.bullet[k].knd = 10;//子弹的种类

boss\_shot.bullet[k].angle = bossatan4(k)+rang(PI/20);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 3-0.1\*j;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

}

}

}

}

if(t==tm12[20]){

flash.flag=1;

flash.knd=1;//闪屏！

flash.to\_cnt=90;//

flash.cnt=0;

for(i=0;i<6;i++)

lazer[i].flag=0;

}

//激光乱射~

if(tm12[21]-20<=t && t<=tm12[21]+30 && t%2){

for(i=0;i<2;i++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 0;//子弹的颜色

boss\_shot.bullet[k].x = FMX;//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 2;//子弹的种类

boss\_shot.bullet[k].angle = PI-PI/2/50\*(t-(tm12[21]-20));//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 3;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

}

if(tm12[23]-20<=t && t<=tm12[23]+30 && t%2){

for(i=0;i<2;i++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 4;//子弹的颜色

boss\_shot.bullet[k].x = 0;//坐标

boss\_shot.bullet[k].y = FMY;

boss\_shot.bullet[k].knd = 2;//子弹的种类

boss\_shot.bullet[k].angle = -PI/2/50\*(t-(tm12[23]-20));//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 3;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

}

//从四面八方发射

double x4[4]={0,FMX,FMX,0};

double y4[4]={0,0,FMY,FMY};

if(t==tm12[25]){

for(j=0;j<4;j++){

for(i=0;i<90;i++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 1;//子弹的颜色

boss\_shot.bullet[k].x = x4[j];//坐标

boss\_shot.bullet[k].y = y4[j];

boss\_shot.bullet[k].knd = 2;//子弹的种类

boss\_shot.bullet[k].angle = PI/2\*j+PI/2/90\*i;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 3;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 1;

}

}

}

}

if(t==tm12[27]){

flash.flag=1;

flash.knd=1;//闪屏！

flash.to\_cnt=90;//在90次计数中变化

flash.cnt=0;

for(i=0;i<6;i++)

lazer[i].flag=0;

}

//阶段。视觉特效

if(t>=tm12[27] && t<tm12[180] && (t-tm12[28])%20==0){

visual\_effect.cnt=0;

int r=GetRand(4);

if(num<100 && (num-27)%16==0)

r+=5;

for(j=0;j<VMX;j++){

n=signal[r][j]\*0.7-visual\_effect.num[j];

for(i=0;i<n;i++){

k=VMY\*j+visual\_effect.num[j]+i;

boss\_shot.bullet[k].col = 0;//子弹的颜色

boss\_shot.bullet[k].x = k/VMY\*VWIDE;//坐标

boss\_shot.bullet[k].y = (k%VMY)\*VWIDE;

boss\_shot.bullet[k].knd = 12;//子弹的种类

boss\_shot.bullet[k].angle = 0;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;//状態

boss\_shot.bullet[k].spd = 0;//速度

}

if(n>0)

visual\_effect.num[j]+=n;

}

}

visual\_effect.cnt++;

for(j=0;j<VMX;j++){

if(visual\_effect.cnt>=5 && GetRand(3)==0){

if(visual\_effect.num[j]==0)continue;

if(visual\_effect.num[j]<=2 && visual\_effect.cnt%2==0)continue;

if(visual\_effect.num[j]<=4 && visual\_effect.cnt%4==0)continue;

if(visual\_effect.num[j]<=6 && visual\_effect.cnt%6==0)continue;

k=VMY\*j+visual\_effect.num[j];

boss\_shot.bullet[k].flag=0;

visual\_effect.num[j]--;

}

}

//激光

if(t>=tm12[26] && t<tm12[90]){

for(i=26;i<=90;i++){

if(t==tm12[i] && (num-26+6)%8==0 && num<=57 ||

t==tm12[i] && (num-58+2)%4==0 && num>=58 && num<=90 ){

// double angle=atan2(ch.y,ch.x-FMX/2);

if(num<85){

double angle=rang(PI);

for(j=0;j<15;j++){

for(s=0;s<40;s++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 0;//子弹的颜色

boss\_shot.bullet[k].x = FMX/2;//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 11;//子弹的种类

boss\_shot.bullet[k].angle = angle+PI2/40\*s;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 5-3.0/15.0\*j;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

}

}

for(j=0;j<11;j++){

if((k=search\_lazer())!=-1){

lazer[k].col = 0;//子弹的颜色

lazer[k].knd = 0;//子弹的种类

lazer[k].angle = -PI/2+PI/10\*j+rang(PI/20);//角度

lazer[k].startpt.x= 0;//坐标

lazer[k].startpt.y= FMY/2+rang(50);

lazer[k].flag = 1;

lazer[k].cnt = 0;

lazer[k].haba = 1;//幅度

lazer[k].state = 1;//状态

if(num>=58)

lazer[k].state=2;

lazer[k].length = FMX\*1.2;//激光的长度

lazer[k].hantei = 0;

lazer[k].lphy.conv\_flag=0;//旋转flag

lazer[k].lphy.conv\_base\_x=0;//旋转基准位置

lazer[k].lphy.conv\_base\_y=0;

lazer[k].lphy.conv\_x=0;//旋转元的位置

lazer[k].lphy.conv\_y=0;

input\_lphy(&lazer[k],0,0);//代入

}

}

for(j=11;j<22;j++){

if((k=search\_lazer())!=-1){

lazer[k].col = 1;//子弹的颜色

lazer[k].knd = 0;//子弹的种类

lazer[k].angle = PI+PI/2+PI/10\*j+rang(PI/20);//角度

lazer[k].startpt.x= FMX;//坐标

lazer[k].startpt.y= FMY/2+rang(50);

lazer[k].flag = 1;

lazer[k].cnt = 0;

lazer[k].haba = 1;//幅度

lazer[k].state = 1;//状态

if(num>=58)

lazer[k].state=2;

lazer[k].length = FMX\*1.2;//激光的长度

lazer[k].hantei = 0;

lazer[k].lphy.conv\_flag=0;//旋转flag

lazer[k].lphy.conv\_base\_x=0;//旋转基准位置

lazer[k].lphy.conv\_base\_y=0;

lazer[k].lphy.conv\_x=0;//旋转元的位置

lazer[k].lphy.conv\_y=0;

input\_lphy(&lazer[k],0,0);//代入

}

}

}

}

}

//从91开始哒~哒~哒~哒~↑♪

int st,ed;

st=91; ed=94;

if(tm12[st]<=t && t<=tm12[ed]){

for(i=st;i<=ed;i++){

if(tm12[i]==t){

for(s=0;s<3;s++){

int setx[3]={0,-30,30};

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 0;//子弹的颜色

boss\_shot.bullet[k].x = setx[s]+50+(FMX-100)/3\*(num-1-st);//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 0;//子弹的种类

boss\_shot.bullet[k].angle = PI/2;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 6-s;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

for(s=0;s<6;s++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 1;//子弹的颜色

boss\_shot.bullet[k].x = rang(30)+50+(FMX-100)/3\*(num-1-st);//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 1;//子弹的种类

boss\_shot.bullet[k].angle = PI/2;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 5.5-3.0/6\*s;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

for(s=0;s<10;s++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 0;//子弹的颜色

boss\_shot.bullet[k].x = rang(20)+50+(FMX-100)/3\*(num-1-st);//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 10;//子弹的种类

boss\_shot.bullet[k].angle = PI/2;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 5.5-4.0/10\*s;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

}

}

}

//哒~哒~哒~哒~↑♪第二次

st=95; ed=98;

if(tm12[st]<=t && t<=tm12[ed]){

for(i=st;i<=ed;i++){

if(tm12[i]==t){

for(s=0;s<3;s++){

int setx[3]={0,-30,30};

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 1;//子弹的颜色

boss\_shot.bullet[k].x = setx[s]+FMX-50-(FMX-100)/3\*(num-1-st);//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 0;//子弹的种类

boss\_shot.bullet[k].angle = PI/2;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 6-s;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

for(s=0;s<6;s++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 0;//子弹的颜色

boss\_shot.bullet[k].x = rang(30)+FMX-50-(FMX-100)/3\*(num-1-st);//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 1;//子弹的种类

boss\_shot.bullet[k].angle = PI/2;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 5.5-3.0/6\*s;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

for(s=0;s<10;s++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 1;//子弹的颜色

boss\_shot.bullet[k].x = rang(20)+FMX-50-(FMX-100)/3\*(num-1-st);//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 10;//子弹的种类

boss\_shot.bullet[k].angle = PI/2;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 5.5-4.0/10\*s;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

}

}

}

//斜方向哒~哒~哒~哒~↑♪

st=99; ed=102;

if(tm12[st]<=t && t<=tm12[ed]){

for(i=st;i<=ed;i++){

if(tm12[i]==t){

for(s=0;s<3;s++){

double setx[3]={0,-PI/20,PI/20};

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 0;//子弹的颜色

boss\_shot.bullet[k].x = 0;//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 0;//子弹的种类

boss\_shot.bullet[k].angle = setx[s]+PI/2-(PI/2-PI/10)/3\*(num-1-st);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 6-s;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

for(s=0;s<6;s++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 1;//子弹的颜色

boss\_shot.bullet[k].x = 0;//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 1;//子弹的种类

boss\_shot.bullet[k].angle = PI/2-(PI/2-PI/10)/3\*(num-1-st)+rang(PI/20);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 5.5-3.0/6\*s;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

for(s=0;s<10;s++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 0;//子弹的颜色

boss\_shot.bullet[k].x = 0;//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 10;//子弹的种类

boss\_shot.bullet[k].angle = PI/2-(PI/2-PI/10)/3\*(num-1-st)+rang(PI/30);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 5.5-4.0/10\*s;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

}

}

}

st=103; ed=106;

if(tm12[st]<=t && t<=tm12[ed]){

for(i=st;i<=ed;i++){

if(tm12[i]==t){

for(s=0;s<3;s++){

double setx[3]={0,-PI/20,PI/20};

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 1;//子弹的颜色

boss\_shot.bullet[k].x = FMX;//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 0;//子弹的种类

boss\_shot.bullet[k].angle = setx[s]+PI/2+(PI/2-PI/10)/3\*(num-1-st);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 6-s;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

for(s=0;s<6;s++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 0;//子弹的颜色

boss\_shot.bullet[k].x = FMX;//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 1;//子弹的种类

boss\_shot.bullet[k].angle = PI/2+(PI/2-PI/10)/3\*(num-1-st)+rang(PI/20);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 5.5-3.0/6\*s;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

for(s=0;s<10;s++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 1;//子弹的颜色

boss\_shot.bullet[k].x = FMX;//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 10;//子弹的种类

boss\_shot.bullet[k].angle = PI/2+(PI/2-PI/10)/3\*(num-1-st)+rang(PI/30);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 5.5-4.0/10\*s;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

}

}

}

//哒~哒~哒~最后的谢幕（last curtain）

st=106;

if(tm12[st]==t){

for(n=st;n<st+3;n++){

int setx[3]={0,-30,30};

for(s=0;s<3;s++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = n%2 ? 1 : 0;//子弹的颜色

boss\_shot.bullet[k].x = setx[s]+FMX-50-(FMX-100)/2\*(n-st);//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 0;//子弹的种类

boss\_shot.bullet[k].angle = PI/2;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 6-s;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

for(s=0;s<6;s++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = n%2 ? 0 : 1;//子弹的颜色

boss\_shot.bullet[k].x = rang(30)+FMX-50-(FMX-100)/2\*(n-st);//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 1;//子弹的种类

boss\_shot.bullet[k].angle = PI/2;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 5.5-3.0/6\*s;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

for(s=0;s<10;s++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = n%2 ? 1 : 0;//子弹的颜色

boss\_shot.bullet[k].x = rang(20)+FMX-50-(FMX-100)/2\*(n-st);//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 10;//子弹的种类

boss\_shot.bullet[k].angle = PI/2;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 5.5-4.0/10\*s;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

}

}

//上升后抛洒（shower）

st=107;

if(tm12[st]<=t && t<=tm12[st]+60 && t%5==0){

for(s=0;s<=150;s++){

if(s%6<4)continue;

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = (s/6)%9;//子弹的颜色

boss\_shot.bullet[k].x = boss.x;//坐标

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 2;//子弹的种类

boss\_shot.bullet[k].angle = 0;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 0;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 2;

boss\_shot.bullet[k].vx = -2.5+5.0/150\*s;

boss\_shot.bullet[k].vy = -8;

}

}

}

//两端下方

/\* st=107;

if(tm12[111]-(tm12[111]-tm12[110])/3==t){

for(s=0;s<=30;s++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 0;//子弹的颜色

boss\_shot.bullet[k].x = 0;//坐标

boss\_shot.bullet[k].y = FMY;

boss\_shot.bullet[k].knd = 11;//子弹的种类

boss\_shot.bullet[k].angle = -PI/4+rang(PI/4);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 1.5+rang(1);//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

for(s=0;s<=30;s++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 0;//子弹的颜色

boss\_shot.bullet[k].x = FMX;//坐标

boss\_shot.bullet[k].y = FMY;

boss\_shot.bullet[k].knd = 11;//子弹的种类

boss\_shot.bullet[k].angle = PI+PI/4+rang(PI/4);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 1.5+rang(1);//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

}\*/

//上升后抛洒（shower）

st=111; ed=114;

if(tm12[st]<=t && t<=tm12[ed]){

for(i=st;i<=ed;i++){

if(tm12[i]<=t && t<tm12[i]+60){

for(s=150/4\*(num-st-1);s<=150/4\*(num-st);s++){

if(s%6<4)continue;

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = (s/6)%9;//子弹的颜色

boss\_shot.bullet[k].x = boss.x;//坐标

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 2;//子弹的种类

boss\_shot.bullet[k].angle = 0;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 0;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 2;

boss\_shot.bullet[k].vx = -2.5+5.0/150\*s;

boss\_shot.bullet[k].vy = -8;

}

}

}

}

}

//激光

st=115; ed=123;

if(tm12[st]<=t && t<=tm12[ed]){

for(i=st;i<=ed;i++){

if(tm12[i]==t){

for(s=0;s<2;s++){

if((k=search\_lazer())!=-1){

lazer[k].col = 0;//子弹的颜色

lazer[k].knd = 0;//子弹的种类

lazer[k].angle = PI/2+rang(PI/2)-PI/4+PI/2\*s;//角度

lazer[k].startpt.x= FMX/2;//坐标

lazer[k].startpt.y= 0;

lazer[k].flag = 1;

lazer[k].cnt = 0;

lazer[k].haba = 1;//幅度

lazer[k].state = 3;//状态

lazer[k].length = FMY\*1.2;//激光的长度

lazer[k].hantei = 0;

lazer[k].lphy.conv\_flag=0;//旋转flag

lazer[k].lphy.conv\_base\_x=0;//旋转基准位置

lazer[k].lphy.conv\_base\_y=0;

lazer[k].lphy.conv\_x=0;//旋转元的位置

lazer[k].lphy.conv\_y=0;

input\_lphy(&lazer[k],0,0);//代入

}

if((k=search\_lazer())!=-1){

lazer[k].col = 1;//子弹的颜色

lazer[k].knd = 0;//子弹的种类

lazer[k].angle = -PI/2+rang(PI/2)-PI/4+PI/2\*s;//角度

lazer[k].startpt.x= FMX/2;//坐标

lazer[k].startpt.y= FMY;

lazer[k].flag = 1;

lazer[k].cnt = 0;

lazer[k].haba = 1;//幅度

lazer[k].state = 3;//状态

lazer[k].length = FMX\*1.2;//激光的长度

lazer[k].hantei = 0;

lazer[k].lphy.conv\_flag=0;//旋转flag

lazer[k].lphy.conv\_base\_x=0;//旋转基准位置

lazer[k].lphy.conv\_base\_y=0;

lazer[k].lphy.conv\_x=0;//旋转元的位置

lazer[k].lphy.conv\_y=0;

input\_lphy(&lazer[k],0,0);//代入

}

}

}

}

}

if(tm12[124]==t){

for(i=0;i<8;i++){

for(s=0;s<=40;s++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = 0;//子弹的颜色

boss\_shot.bullet[k].x = FMX/2;//坐标

boss\_shot.bullet[k].y = 0;

boss\_shot.bullet[k].knd = 11;//子弹的种类

boss\_shot.bullet[k].angle = PI/40\*s+rang(PI/80);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 4.5-3.5/8\*i+rang(0.2);//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 0;

}

}

}

}

if(t==tm12[125]){

flash.flag=1;

flash.knd=1;//闪屏！

flash.to\_cnt=50;//在90次计数中变化

flash.cnt=0;

}

//闪~啊~闪~

if(tm12[133]<=t && t<tm12[153] && t%3==0){

for(i=0;i<2;i++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = base\_col;//子弹的颜色

boss\_shot.bullet[k].x = boss.x;//坐标

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 1;//子弹的种类

boss\_shot.bullet[k].angle = base\_angle-PI\*0.15+PI\*0.3\*i;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 9;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 3;

}

}

}

if(tm12[133]==t||tm12[137]==t||tm12[141]==t){

if((k=search\_lazer())!=-1){

lazer[k].col = 1;//子弹的颜色

lazer[k].knd = 0;//子弹的种类

lazer[k].angle = bossatan2();//角度

lazer[k].startpt.x= boss.x;//坐标

lazer[k].startpt.y= boss.y;

lazer[k].flag = 1;

lazer[k].cnt = 0;

lazer[k].haba = 1;//幅度

lazer[k].state = 4;//状态

lazer[k].length = FMX;//激光的长度

lazer[k].hantei = 0;

lazer[k].lphy.conv\_flag=0;//旋转flag

lazer[k].lphy.conv\_base\_x=0;//旋转基准位置

lazer[k].lphy.conv\_base\_y=0;

lazer[k].lphy.conv\_x=0;//旋转元的位置

lazer[k].lphy.conv\_y=0;

input\_lphy(&lazer[k],0,0);//代入

}

}

if(tm12[137]==t){

base\_angle+=PI/6+rang(PI/20);

base\_col=1;

}

if(tm12[141]==t){

base\_angle-=PI/6\*2+rang(PI/20);

base\_col=0;

}

if(tm12[145]==t){

base\_angle+=PI/4+rang(PI/20);

base\_col=1;

}

if(tm12[147]==t){

base\_angle=-PI/2;

base\_col=0;

}

if(tm12[147]<=t && t<tm12[161] && t%3==0){

for(i=0;i<3;i++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = base\_col;//子弹的颜色

boss\_shot.bullet[k].x = boss.x;//坐标

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 1;//子弹的种类

boss\_shot.bullet[k].angle = base\_angle-PI\*0.15+PI\*0.15\*i;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 9;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 3;

}

}

}

if(tm12[147]==t||tm12[151]==t||tm12[155]==t){

for(i=0;i<3;i++){

if((k=search\_lazer())!=-1){

lazer[k].col = i%2;//子弹的颜色

lazer[k].knd = 0;//子弹的种类

lazer[k].angle = bossatan2()-PI/6+PI/6\*i;//角度

lazer[k].startpt.x= boss.x;//坐标

lazer[k].startpt.y= boss.y;

lazer[k].flag = 1;

lazer[k].cnt = 0;

lazer[k].haba = 1;//幅度

lazer[k].state = 4;//状态

lazer[k].length = FMX;//激光的长度

lazer[k].hantei = 0;

lazer[k].lphy.conv\_flag=0;//旋转flag

lazer[k].lphy.conv\_base\_x=0;//旋转基准位置

lazer[k].lphy.conv\_base\_y=0;

lazer[k].lphy.conv\_x=0;//旋转元的位置

lazer[k].lphy.conv\_y=0;

input\_lphy(&lazer[k],0,0);//代入

}

}

}

if(tm12[151]==t){

base\_angle-=PI/10+rang(PI/20);

base\_col=1;

}

if(tm12[155]==t){

base\_angle+=PI/5+rang(PI/20);

base\_col=0;

}

if(tm12[159]==t){

base\_angle-=PI/4+rang(PI/20);

base\_col=1;

}

if(tm12[161]==t){

base\_angle=-PI/2;

base\_col=0;

}

if(tm12[161]<=t && t<tm12[173] && t%3==0){

for(i=0;i<4;i++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = base\_col;//子弹的颜色

boss\_shot.bullet[k].x = boss.x;//坐标

boss\_shot.bullet[k].y = boss.y;

boss\_shot.bullet[k].knd = 1;//子弹的种类

boss\_shot.bullet[k].angle = base\_angle-PI\*0.18+PI\*0.12\*i;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 9;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 3;

}

}

}

if(tm12[161]==t||tm12[165]==t||tm12[169]==t){

for(i=0;i<6;i++){

if((k=search\_lazer())!=-1){

lazer[k].col = (i+1)%2;//子弹的颜色

lazer[k].knd = 0;//子弹的种类

lazer[k].angle = bossatan2()+PI2/6\*i;//角度

lazer[k].startpt.x= boss.x;//坐标

lazer[k].startpt.y= boss.y;

lazer[k].flag = 1;

lazer[k].cnt = 0;

lazer[k].haba = 1;//幅度

lazer[k].state = 4;//状态

lazer[k].length = FMX;//激光的长度

lazer[k].hantei = 0;

lazer[k].lphy.conv\_flag=0;//旋转flag

lazer[k].lphy.conv\_base\_x=0;//旋转基准位置

lazer[k].lphy.conv\_base\_y=0;

lazer[k].lphy.conv\_x=0;//旋转元的位置

lazer[k].lphy.conv\_y=0;

input\_lphy(&lazer[k],0,0);//代入

}

}

}

if(tm12[165]==t){

base\_angle-=PI/10+rang(PI/20);

base\_col=1;

}

if(tm12[169]==t){

base\_angle+=PI/5+rang(PI/20);

base\_col=0;

}

if(tm12[161]==t||tm12[165]==t||tm12[169]==t){

for(i=0;i<6;i++){

if((k=search\_lazer())!=-1){

lazer[k].col = (i+1)%2;//子弹的颜色

lazer[k].knd = 0;//子弹的种类

lazer[k].angle = bossatan2()+PI2/6\*i;//角度

lazer[k].startpt.x= boss.x;//坐标

lazer[k].startpt.y= boss.y;

lazer[k].flag = 1;

lazer[k].cnt = 0;

lazer[k].haba = 1;//幅度

lazer[k].state = 4;//状态

lazer[k].length = FMX;//激光的长度

lazer[k].hantei = 0;

lazer[k].lphy.conv\_flag=0;//旋转flag

lazer[k].lphy.conv\_base\_x=0;//旋转基准位置

lazer[k].lphy.conv\_base\_y=0;

lazer[k].lphy.conv\_x=0;//旋转元的位置

lazer[k].lphy.conv\_y=0;

input\_lphy(&lazer[k],0,0);//代入

}

}

}

if(tm12[175]==t){

for(i=0;i<6;i++){

if((k=search\_lazer())!=-1){

lazer[k].col = 0;//子弹的颜色

lazer[k].knd = 0;//子弹的种类

lazer[k].angle = base\_angle+PI2/6\*i;//角度

lazer[k].startpt.x= boss.x;//坐标

lazer[k].startpt.y= boss.y;

lazer[k].flag = 1;

lazer[k].cnt = 0;

lazer[k].haba = 1;//幅度

lazer[k].state = 5;//状态

lazer[k].length = FMX;//激光的长度

lazer[k].hantei = 0;

lazer[k].lphy.conv\_flag=0;//旋转flag

lazer[k].lphy.conv\_base\_x=0;//旋转基准位置

lazer[k].lphy.conv\_base\_y=0;

lazer[k].lphy.conv\_x=0;//旋转元的位置

lazer[k].lphy.conv\_y=0;

input\_lphy(&lazer[k],0,0);//代入

}

}

}

if(tm12[179]==t){

for(i=0;i<6;i++){

if((k=search\_lazer())!=-1){

lazer[k].col = 1;//子弹的颜色

lazer[k].knd = 0;//子弹的种类

lazer[k].angle = base\_angle+PI2/6\*i;//角度

lazer[k].startpt.x= boss.x;//坐标

lazer[k].startpt.y= boss.y;

lazer[k].flag = 1;

lazer[k].cnt = 0;

lazer[k].haba = 1;//幅度

lazer[k].state = 6;//状态

lazer[k].length = FMX;//激光的长度

lazer[k].hantei = 0;

lazer[k].lphy.conv\_flag=0;//旋转flag

lazer[k].lphy.conv\_base\_x=0;//旋转基准位置

lazer[k].lphy.conv\_base\_y=0;

lazer[k].lphy.conv\_x=0;//旋转元的位置

lazer[k].lphy.conv\_y=0;

input\_lphy(&lazer[k],0,0);//代入

}

}

}

if(tm12[183]-60==t){

double sx=FMX/2,sy=420;

for(i=0;i<20;i++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = i%9;//子弹的颜色

boss\_shot.bullet[k].x = sx+cos(PI2/20\*i)\*400;//坐标

boss\_shot.bullet[k].y = sy+sin(PI2/20\*i)\*400;

boss\_shot.bullet[k].knd = 4;//子弹的种类

boss\_shot.bullet[k].angle = PI2/20\*i+PI;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 4;//速度

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 4;

boss\_shot.bullet[k].till = 1000;

}

}

}

if(t==tm12[187]){

flash.flag=1;

flash.knd=1;//闪屏！

flash.to\_cnt=120;//在90次计数中变化

flash.cnt=0;

}

int rflag=0;

double rx,ry;

for(i=VMX\*VMY;i<BOSS\_BULLET\_MAX;i++){

if(boss\_shot.bullet[i].flag>0){

int cnt=boss\_shot.bullet[i].cnt;

int state=boss\_shot.bullet[i].state;

double x=boss\_shot.bullet[i].x,y=boss\_shot.bullet[i].y;

if(t==tm12[0]||t==tm12[7]||t==tm12[14]||t==tm12[20]||t==tm12[27])

boss\_shot.bullet[i].flag=0;

if(state==1){

if(cnt>35)

boss\_shot.bullet[i].spd\*=0.983;

}

if(state==2){

if(boss\_shot.bullet[i].vy<8)

boss\_shot.bullet[i].vy+=0.15;

boss\_shot.bullet[i].x+=boss\_shot.bullet[i].vx;

boss\_shot.bullet[i].y+=boss\_shot.bullet[i].vy;

boss\_shot.bullet[i].angle=atan2(boss\_shot.bullet[i].vy,boss\_shot.bullet[i].vx);

}

if(state==3){

if(x<0+9 || FMX-9<x || y<0+9){

if(x<0+9 || FMX-9<x )

boss\_shot.bullet[i].angle=PI-boss\_shot.bullet[i].angle;

if(y<0+9)

boss\_shot.bullet[i].angle=PI2-boss\_shot.bullet[i].angle;

}

}

if(state==4){

if(cnt==100){

boss\_shot.bullet[i].spd=0;

}

if(tm12[187]==t){

boss\_shot.bullet[i].vy=10;

boss\_shot.bullet[i].state=5;

}

}

if(state==5){

if(boss\_shot.bullet[i].vy>0){

boss\_shot.bullet[i].vy-=0.15;

if(boss\_shot.bullet[i].vy<0)

boss\_shot.bullet[i].vy=0;

}

boss\_shot.bullet[i].y-=boss\_shot.bullet[i].vy;

if(t==tm12[188]){

rx=boss\_shot.bullet[i].x;

ry=boss\_shot.bullet[i].y;

rflag=1;

boss\_shot.bullet[i].flag=0;

}

}

if(state==6){

if(cnt<180)

boss\_shot.bullet[i].spd\*=0.98;

if(cnt>180)

boss\_shot.bullet[i].spd/=0.99;

if(cnt>480)

boss\_shot.bullet[i].spd/=0.99;

}

}

}

if(rflag==1){

for(j=0;j<=20;j++){

for(i=0;i<100-j\*4;i++){

if((k=search\_boss\_shot\_n(VMX\*VMY,BOSS\_BULLET\_MAX))!=-1){

boss\_shot.bullet[k].col = GetRand(8);//子弹的颜色

boss\_shot.bullet[k].x = rx;//坐标

boss\_shot.bullet[k].y = ry;

boss\_shot.bullet[k].knd = 11;//子弹的种类

boss\_shot.bullet[k].angle = PI2/(100-j\*4)\*i+rang(PI/200);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 9-9/20.0\*j+rang(0.2);//速度

if(boss\_shot.bullet[k].spd<0.1)

boss\_shot.bullet[k].spd=0.1;

boss\_shot.bullet[k].eff = 1;

boss\_shot.bullet[k].state = 6;

boss\_shot.bullet[k].till = 0;

}

}

}

}

for(i=0;i<LAZER\_MAX;i++){

if(lazer[i].flag>0){

int cnt=lazer[i].cnt;

int state=lazer[i].state;

if(state==1){

if(cnt>=20\*4-7 && cnt<20\*4){

lazer[i].haba+=1.5;

}

if(cnt==20\*4+1)

lazer[i].hantei=0.6;

if(cnt>=20\*8-30-7 && cnt<20\*8-30){

lazer[i].hantei=0;

lazer[i].haba-=1.5;

}

if(cnt==20\*8-30){

lazer[i].flag=0;

}

}

if(state==2){

if(cnt>=20\*2-5 && cnt<20\*2){

lazer[i].haba+=1.5;

}

if(cnt==20\*2+1)

lazer[i].hantei=0.6;

if(cnt>=20\*6-30-5 && cnt<20\*6-30){

lazer[i].hantei=0;

lazer[i].haba-=1.5;

}

if(cnt==20\*6-30){

lazer[i].flag=0;

}

}

if(state==3){

if(t>=tm12[124]-7 && t<tm12[124]){

lazer[i].haba+=1;

}

if(t==tm12[124]+2)

lazer[i].hantei=0.6;

if(t==tm12[125]){

lazer[i].flag=0;

}

}

if(state==4){

if(cnt>=20\*2-10 && cnt<20\*2){

lazer[i].haba+=1;

}

if(cnt==20\*2+2)

lazer[i].hantei=0.6;

if(cnt>=20\*4-10 && cnt<20\*4){

lazer[i].hantei=0;

lazer[i].haba-=1;

}

if(cnt==20\*4){

lazer[i].flag=0;

}

}

if(state==5){

lazer[i].angle+=PI/180;

if(30<cnt && cnt<40){

lazer[i].haba+=3.0;

}

if(cnt==45)

lazer[i].hantei=0.6;

if(cnt>=70 && cnt<80){

lazer[i].hantei=0;

lazer[i].haba-=3.0;

}

if(cnt==80){

lazer[i].flag=0;

}

}

if(state==6){

lazer[i].angle-=PI/180;

if(30<cnt && cnt<40){

lazer[i].haba+=3.0;

}

if(cnt==45)

lazer[i].hantei=0.6;

if(cnt>=70 && cnt<80){

lazer[i].hantei=0;

lazer[i].haba-=3.0;

}

if(cnt==80){

lazer[i].flag=0;

}

}

}

}

}

为了体现一个函数一个弹幕的风格，因此就这样写到一个函数里面了……真是长啊……（汗）

这次追加了控制FPS的处理这样的东西，因此这章的内容将在下一章中继续说明。

非常抱歉连我自己都不知道修改了多少地方了，因此实际上我准备在其它章节中介绍追加了的处理。

其次，至于音乐的同捆问题，由于曲子的制作者还没有答复是否同意，因此还没有放入工程里面。

现在如果您要试着玩这个弹幕的话只能您自己准备音乐了。

http://www.nicovideo.jp/watch/sm3822557

niconico动画中有这个曲子（上面的链接中的“5.　20:30 Rage/Godspeed”。）

http://dixq.net/sr/niconico.html

您可以参考这里的介绍来把曲子扒下来。

不过，从20:30开始（实际上还要提前一点）到某个地方为止，您必须要编辑得和介绍动画里面的音乐一样才行。

可以使用Audacity之类的软件编辑得和介绍动画中一样。

……这些怎么样都行但又麻烦到家的事情又不能不做，您如果没兴趣的话这章就可以略过了。m(\_ \_||)m

如果还有进展的话我大概会重新写一次吧。

## 第四十三章 准确地控制FPS吧

本章中介绍STG中非常重要的FPS控制的处理。

本章的着眼点在于轻巧利落地完成待机这一工作。

在这里先说一些作为预备知识的东西。

一般而言游戏都是以60fps进行的。

所谓fps，就是frame per second的简称，也即1秒钟进行多少帧的意思。

1秒钟如果进行60次绘制的话那就是60FPS。

那么1帧又是多少秒呢？如果是60FPS的话

（毫秒=1/1000秒，简写为ms）

1000 / 60fps = 16.666666…(ms)

也就是说1帧需要保证的毫秒数是无法用int来表示的。

假定以1帧16ms来待机的话，

1000ms / 16ms = 62.5fps

又如果以1帧17ms来待机话，

1000ms / 17ms = 58.823...fps

结果会是这样子，那就不能严格按照60FPS进行了。

如果注意到16.666666…这个数字就是(16+17+17)/3的话，可能就会想到“第1次待机为16ms，然后以17ms待机两次，然后这样循环进行不就行了？”

然而，估计大家都用Sleep函数来待机吧（WaitTimer函数会执行一些多余的处理，不适合短时间的待机），

Sleep( int time );

不过向

Sleep(int time)

的参数time传入的并不是“待机的毫秒数”。

这个函数并不是进行待机指定毫秒数的工作，而是进行“超过time毫秒再返回”的工作。在Windows这样子多任务的操作系统中，无法测定正确的时间，即便写了Sleep(100)实际上待机时间可能是101ms。虽然会认为1/1000秒的误差看起来微不足道，但是我们已经在16ms还是17ms这个问题上都很苦恼了，要是有1ms的偏差那么计算结果可就大变样了。现在我们看看以1秒为单位的计算。

以16ms为单位考虑的话1ms的误差是很大的，而如果我们以1000ms为单位来计算的话，那么就可以很好地控制误差可能造成的错误了。

在第1帧中出现的误差我们在第2帧中修正，接着在第2帧中出现的误差我们在第3帧中修正……

这样子反复进行，一直到60帧结束，这样一来全体产生的误差都可以修正了。

具体而言就是，

在第0帧的时候保存当前时刻，以此为基准进行待机。

假定现在是第1帧，那么从第0帧开始计数，进行(int)(16.6666…\*1)ms的待机就行了，也就是16ms。

假定现在是第2帧，那么从第0帧开始计数，进行(int)(16.6666…\*2)ms的待机就行了，也就是33ms。

假定现在是第3帧，那么从第0帧开始计数，进行(int)(16.6666…\*3)ms的待机就行了，也就是49ms。

……（略）

假定现在是第60帧，那么从第0帧开始计数，进行(int)(16.6666…\*60)ms的待机就行了，也就是999ms。

这样一来，1秒的误差我们就控制到0.001秒这个数量级了。

然而，既然第1秒待机1秒这一点是很清楚的，那么从这个时候开始一直到60帧前记录的时刻+1000ms为止待机就行了（译者注：事实上这里作者的说法可能会产生误导，会让成产生可能有if(lastTime + 1000ms <= curTime)这种类型的判断的想法，事实上从源代码来看并不是这样的，作者直接采用的阶段等待的方法来减小误差）。

这么一来1秒之间的误差就只有Sleep函数的误差那么多了。

我们试着在游戏中实现吧。

----在 function.h 中进行以下追加 ----

//fps.cpp

GLOBAL void fps\_wait();

GLOBAL void draw\_fps(int x, int y);

----在 main.cpp 的main函数中进行以下追加/修正 ----

switch(func\_state){

case 0://游戏启动时进行的处理

load(); //读入数据

first\_ini();//最开始的初始化

func\_state=99;

break;

case 99://STG开始前进行的初始化

ini();

load\_story();

func\_state=100;

break;

case 100://通常处理

calc\_ch(); //角色计算

ch\_move(); //控制角色的移动

cshot\_main();//自机射击main

enemy\_main();//敌人处理main

boss\_shot\_main();

shot\_main(); //射击main

out\_main(); //碰撞计算

effect\_main();//特效main

calc\_main();//游戏Title显示计算

graph\_main();//绘制main

draw\_fps(0,465);//fps显示

if(boss.flag==0)

stage\_count++;

break;

default:

printfDx("错误的func\_state\n");

break;

}

fps\_wait();//计算帧等待

music\_play();

if(CheckStateKey(KEY\_INPUT\_ESCAPE)==1)break;//如果按下ESC键则跳出循环

ScreenFlip();

----在 fps.cpp 中进行以下追加 ----

#include "../include/GV.h"

//fps

#define FLAME 60

//fps的计数器，60帧1次记录作为基准的时刻

int fps\_count,count0t;

//为了进行平均计算,记录60次1个周期的时间

int f[FLAME];

//平均fps

double ave;

//以FLAME fps为目标进行fps的计算和控制

void fps\_wait(){

int term,i,gnt;

static int t=0;

if(fps\_count==0){//60帧1次的话

if(t==0)//如果是最开始的话不等待

term=0;

else//基于上一次记录了的时间计算

term=count0t+1000-GetNowCount();

}

else //应该等待的时间=当前的理论时刻-当前的实际时刻（译者注：GetNowCount()是DX Library中类似GetTickCount()的函数，用于返回系统启动后到当前为止的毫秒精度的时间计数）

term = (int)(count0t+fps\_count\*(1000.0/FLAME))-GetNowCount();

if(term>0)//只等待应该等待的时间（译者注：如果term大于0的话，说明当前帧率应该到达的值还超过了当前的时刻，那么就等待这个差值即可；反过来则无需等待，因为说明当前帧等待的时间早就被超过了，之所以出现这种情况一般是因为绘制过程太多影响了效率）

Sleep(term);

gnt=GetNowCount();

if(fps\_count==0)// 60帧进行1次基准变更

count0t=gnt;

f[fps\_count]=gnt-t;//记录1个周期的时间

t=gnt;

//平均计算

if(fps\_count==FLAME-1){

ave=0;

for(i=0;i<FLAME;i++)

ave+=f[i];

ave/=FLAME;

}

fps\_count = (++fps\_count)%FLAME ;

}

//在x,y的位置显示fps

void draw\_fps(int x, int y){

if(ave!=0){

DrawFormatString(x, y,color[0],"[%.1f]",1000/ave);

}

return;

}

实际上，只要修改

#define FLAME 60

这里的数字，就可以变成您自己想要的fps了，您可以自行确认一下。

另外，如果自己使用的显示器的刷新率是60的话，就没法显示60以上了，请注意一下。

运行结果

## 第四十四章 制作支持DEBUG的函数吧

接下来，随着处理的增加，我们发现渐渐地当错误出现的时候，我们已经很难弄清楚到底原因何在呢。

另外，我想也很难找出到底哪一块的代码是无用的。

因此，我们试着让在主循环中调用的每一个自定义函数都显示出各自在处理上消耗了多少时间吧。

这样一来，假设我们有大量的无用处理，或者在哪个地方混杂着奇怪的程序的话，那么肯定会在某个特定的地方有长时间的处理，那么我们就可以直接去看有问题的地方并进行修正就行了。

即便没有特别错误的地方，如果存在在没有充分考虑过的处理中耗时的函数的话，也很有必要去检查这个函数是否能够被优化，这么一来显示出各个函数的处理时间是很有作用的。

作为实现的方法，

时间测定函数();

自定义函数();

时间测定函数();

像这样，将自定义函数用时间测定函数夹在中间，那么就能够获得处理时间了。

试着实现这个方法吧。

其次，本章中也试着对文字的绘制使用字体数据。

先定义保存字体句柄的font变量，然后我们在读取函数中把字体数据构建好。

---- 新增加check\_time.cpp 文件，在其中进行以下追加----

#include "../include/GV.h"

//名称的最大字数

#define STR\_MAX 64

//可以检查的函数的最大数

#define FUNC\_MAX 30

//计数当前是第几个

int func\_count;

//保存上一次测定的时间

LONGLONG lt;

//保存测定数据用的变量

typedef struct{

int tm;

char str[STR\_MAX];

}func\_tm\_t;

func\_tm\_t func\_tm[FUNC\_MAX];

//登录在st变量中接受到的数据。如果flag为1的话就重置

void enter\_func\_tm(char st[], int flag){

int i;

LONGLONG nowtm;

if(func\_count>=FUNC\_MAX){

printfDx("func\_count的值%d异常\n",func\_count);

return ;

}

nowtm=GetNowHiPerformanceCount() ;

if(nowtm-lt<INT\_MAX){//如果在int可以表示的范围内的话

func\_tm[func\_count].tm=(int)(nowtm-lt);//保存处理时间

memcpy(func\_tm[func\_count].str,st,STR\_MAX-1);//复制字符串

func\_tm[func\_count].str[STR\_MAX-1]=0;//放入终止符（\0）

}

else//如果超出的话

func\_tm[func\_count].tm=-1;//错误

lt=nowtm ;//保存上一次记录的时刻

if(flag==1){//重置

for(i=func\_count+1;i<FUNC\_MAX;i++)

func\_tm[i].str[0]=0;

func\_count=0;

}

else

func\_count++;

}

//绘制数据

void draw\_func\_tm(int x, int y){

int i;

unsigned int total=0;

for(i=0;i<FUNC\_MAX;i++){

if(func\_tm[i].str[0]==0)break;

DrawFormatStringToHandle(x,y+14\*i,color[0],font[0],"%05.2f:%s",func\_tm[i].tm/1000.0,func\_tm[i].str );

total+=func\_tm[i].tm;

}

DrawFormatStringToHandle(x,y+14\*i,color[0],font[0],"合計:%05.2f",total/1000.0);

}

现在我们为了进行精确的时间测定，试着使用了100万分之1秒（微秒）精度的GetNowHiPerformanceCount()函数。

既然有100万分之1秒的测定精度，那么1000秒钟就有10亿微秒，很快就会超过int类型的上限了。因此，返回值使用的是LONGLONG型，这一点请注意。

不过，1个函数的处理时间不可能有几千秒以上，因此我们还是将保存时间差的变量用int类型定义。

其次这个enter\_func\_tm以字符串作为参数。

我们把作为参数的字符串和测定了的时间放到一起在后面来显示。

为此我们定义把设置名称和测定时间放到一起func\_tm\_t类型。

函数的调用的情况，我们还是在其它地方一起来进行观察吧。

啊，由于另外还有一点就是在上一章中介绍了的draw\_fps函数也放到绘制函数中了，请参照下面的介绍来确认这一点。

----在 GV.h 中进行以下追加 ----

//其它变量

GLOBAL int font[20];

----在 function.h 中进行以下追加 ----

//check\_time.cpp

GLOBAL void enter\_func\_tm(char st[], int flag = 0);

GLOBAL void draw\_func\_tm(int x, int y);

----在 load.cpp 的 load() 中进行以下追加 ----

font[0] = CreateFontToHandle( "HGPｺﾞｼｯｸE" , 15 , 2 , DX\_FONTTYPE\_ANTIALIASING\_EDGE);

----在 graph.cpp 的头部进行以下追加 ----

extern void draw\_fps(int,int);

extern void draw\_func\_tm(int,int);

----在 graph.cpp 的 graph\_main() 函数中进行以下变更 ----

void graph\_main(){

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_back\_main();//背景绘制main

graph\_effect(0);//敌人的消灭特效

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_effect(4);//决死特效

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_child();

graph\_item();//绘制道具

graph\_boss();

graph\_enemy();//敌人的绘制

graph\_cshot();//自机射击的绘制

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_ch();//自机的绘制

if(bright\_set.brt!=255)SetDrawBright(bright\_set.brt,bright\_set.brt,bright\_set.brt);

graph\_lazer();//激光的绘制

graph\_bullet();//子弹的绘制

if(bright\_set.brt!=255)SetDrawBright(255,255,255);

graph\_effect(1);//Boom的特效

graph\_effect(2);//Boom线的特效

graph\_effect(3);//Boom角色的特效

graph\_stage\_title();//Title的显示

graph\_board();//面板的绘制

graph\_flash();//闪屏的绘制

graph\_develop();

draw\_fps(0,465);//fps绘制

draw\_func\_tm(450,250);//处理时间绘制

}

----在 main.cpp 的main函数中如下述注释那样变更----

---- 在enter\_func\_tm函数中进行变更追加 ----

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nCmdShow){

ChangeWindowMode(TRUE);//窗口模式

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初期化と裏画面化

while(ProcessLoop()==0){//主循环

music\_ini();

switch(func\_state){

case 0://游戏启动时的处理

load(); //读入数据

first\_ini();//最开始的初始化

func\_state=99;

break;

case 99://STG开始前的初始化

ini();

load\_story();

func\_state=100;

break;

case 100://通常处理

enter\_func\_tm("最初");

calc\_ch(); enter\_func\_tm("角色計算");

ch\_move(); enter\_func\_tm("角色移动");

cshot\_main(); enter\_func\_tm("自机射击main");

enemy\_main(); enter\_func\_tm("敌人处理main");

boss\_shot\_main();enter\_func\_tm("Boss射击main");

shot\_main(); enter\_func\_tm("射击main");

out\_main(); enter\_func\_tm("碰撞判定");

effect\_main(); enter\_func\_tm("特效main");

calc\_main(); enter\_func\_tm("计算main");

graph\_main(); enter\_func\_tm("绘制main");

if(boss.flag==0)

stage\_count++;

break;

default:

printfDx("错误的func\_state\n");

break;

}

music\_play(); enter\_func\_tm("音乐播放");

fps\_wait(); enter\_func\_tm("待机的时间",1);

if(CheckStateKey(KEY\_INPUT\_ESCAPE)==1)break;//按下ESC键则跳出循环

ScreenFlip();//里外画面翻转

count++;

}

DxLib\_End();//DX Library终止处理

return 0;

}

我们大可不必关注“最初”和“待机时间”这两项。

因为这两个分别是调整FPS的待机时间和与垂直同步配合的待机时间，因此值肯定会变大。

虽然这个和电脑的配置有关，不过各个自定义函数的处理时间尽量能在1ms以内就很不错了。

如果全部时间合计超过16.6ms的话，就是“丢帧（处理滞后）”的状态了。

请尽量不要变成这样。

运行结果

## 第四十五章 实现中路和终路Boss吧

一直到上一章为止Boss都还在一直放着弹幕不会停止，反复地多次重复弹幕的话就会出现指针错误。

本章中我们将要让Boss的弹幕停止，同时也要做出中路Boss和终路Boss。

虽然说是中路Boss和终路Boss，但是它们都是敌人，在龙神录和东方中，Boss在中路出现一次被干掉跑掉后，在最后会完整地再出现一次。我们试着实现这个吧。

在boss结构体中追加danmaku\_num[2]。

这是设定中路Boss发射多少个弹幕、终路Boss发射多少个弹幕的变量。

在接下来的弹幕临近的时候，如果当前的弹幕数和这个数字一样的话那么就让Boss被销毁。

另外本章中，由于进行画面咚！地摇动的dn处理全部在一个文件中实现了，因此请配合那里一起阅读。

dn的登录需要使用enter\_dn。

----在 struct.h 中进行以下部分变更 ----

//Boss的信息

typedef struct{

//flag、计数器、种类、待机时间、状态、直至弹幕结束的计数器、背后的羽衣、显示flag

int flag,cnt,knd,wtime,state,endtime,hagoromo,graph\_flag;

//HP,最大HP

int hp,hp\_max;

//出现计数、各个弹幕的的HP、背景的种类、中路Boss-终路Boss弹幕数（45）

int

appear\_count[2],set\_hp[DANMAKU\_MAX],back\_knd[DANMAKU\_MAX],danmaku\_num[2];

//坐标

double x,y,dx,dy,ang,spd;

//物理移动的变量

phy\_t phy;

}boss\_t;

----在 function.h 中进行以下追加 ----

//dn.cpp

GLOBAL void enter\_dn(int size,int time);

----在 define.h 进行以下变更

#define BOSS\_POS\_Y (FMY/4)

----在 func.h中进行以下变更 ----

void (\*boss\_shot\_bullet[DANMAKU\_MAX])() =

{

boss\_shot\_bulletH000,

boss\_shot\_bulletH001,

boss\_shot\_bulletH000,

boss\_shot\_bulletH009,

boss\_shot\_bulletH000,

};

----在 ini.cpp 中进行以下变更 ----

int back\_knd[DANMAKU\_MAX]={//背景的种类

0,1,0,1,// 0为通常 1为SC用

};

//游戏的初始化

void ini(){

stage\_count=1;

memset(&ch,0,sizeof(ch\_t));

memset(enemy,0,sizeof(enemy\_t)\*ENEMY\_MAX);

memset(lazer,0,sizeof(lazer\_t)\*LAZER\_MAX);

memset(enemy\_order,0,sizeof(enemy\_order\_t)\*ENEMY\_ORDER\_MAX);

memset(shot,0,sizeof(shot\_t)\*SHOT\_MAX);

memset(cshot,0,sizeof(cshot\_t)\*CSHOT\_MAX);

memset(effect,0,sizeof(effect\_t)\*EFFECT\_MAX);

memset(del\_effect,0,sizeof(del\_effect\_t)\*DEL\_EFFECT\_MAX);

memset(&bom,0,sizeof(bom\_t));

memset(&bright\_set,0,sizeof(bright\_set\_t));

memset(&dn,0,sizeof(dn\_t));

memset(&boss,0,sizeof(boss\_t));

memset(child,0,sizeof(child\_t)\*CHILD\_MAX);

memset(&stage\_title,0,sizeof(stage\_title\_t));

memset(item,0,sizeof(item\_t)\*ITEM\_MAX);

ch.x=FMX/2;

ch.y=FMY\*3/4;

ch.power=0;//初始化power(41)

ch.num=5;//初始化残机数(41)

stage\_title.appear\_cnt=stage\_title\_count[stage];

/\*弹幕的各种设定。留在后面好好设定吧。\*/

boss.appear\_count[0]=800;//中路Boss出现的时刻(42)

boss.appear\_count[1]=1100;//终路Boss出现的时刻(44)

boss.danmaku\_num[0]=1;//中路Boos要放的弹幕个数

boss.danmaku\_num[1]=3;//终路Boos要放的弹幕个数

for(int i=0;i<DANMAKU\_MAX;i++){//弹幕的各自HP

boss.set\_hp[i]=1000;

boss.hp\_max=1000;

}

for(int i=0;i<DANMAKU\_MAX;i++)//弹幕的各自背景的种类

boss.back\_knd[i]=back\_knd[i];

/\*到此为止\*/

bright\_set.brt=255;//最开始亮度为最大

}

----在 effect.cpp 的头部进行以下追加 ----

extern void dn\_calc();

----在 effect.cpp 的 calc\_effect()进行以下变更 ----

case 1://Boom的特效

//速度計算

if(effect[i].cnt<60)

effect[i].spd-=(0.2+effect[i].cnt\*effect[i].cnt/3000.0);

if(effect[i].cnt==60){

effect[i].spd=0;

se\_flag[15]=1;

enter\_dn(11,20);//(45)

}

//亮度和大小的计算

effect[i].r+=0.015;

if(effect[i].cnt<51)

effect[i].brt+=5;

if(effect[i].cnt>=60){

effect[i].r+=0.04;

effect[i].brt-=255/30.0;

}

//计数器自增与消去计算

effect[i].cnt++;

if(effect[i].cnt>=90)

effect[i].flag=0;

break;

----在 dn.cpp 进行以下的新追加 ----

#include "../include/GV.h"

void enter\_dn(int size, int time){

dn.flag=1;

dn.cnt=0;

dn.time=time;

dn.size=size;

}

//咚绑！地摇动画面的处理

void dn\_calc(){

if(dn.flag==1){

dn.x=(int)rang(dn.size);

dn.y=(int)rang(dn.size);

dn.cnt++;

if(dn.cnt>dn.time){//如果超过指定的时间的话停止

dn.flag=0;

dn.x=0;

dn.y=0;

}

}

}

在boss.knd里面传入Boss当前发射哪个种类的弹幕（译者注：这里作者）。

如果这个值和ini函数中保存了的boss.danmaku\_num的值一样的话就销毁Boss。（译者注：前提是boss.knd的这个弹幕已经放完）

在销毁的时候发出咚梆！的声音同时加以特效吧。

boss.danmaku\_num[0]是中路Boss的终止值而[1]是终路Boss的终止值。

由于编号从0开始的，那么想要让中路Boss显示在func.h的void (\*boss\_shot\_bullet[DANMAKU\_MAX])()中指定的到第2个弹幕为止的这些弹幕的话，也即是说我们要让中路Boss显示到编号为1为止，那么我们就指定1。（译者注：这里我多说两句，由于全局只有一个保存弹幕函数指针的全局变量boss\_shot\_bullet，因此对于一面的所有Boss而言，它们的knd的变化事实上应该是这个样子的：第1个Boss：0~x1，第2个Boss：x1+1~x2，第3个Boss：x2+1~x3……终路Boss：xk~n-1，这里的n是boss\_shot\_bullet中这一面所有的弹幕的数量，这里通过对boss.knd的累加来实现弹幕的变换——这是非面向对象化的方法）

----在 boss\_shot.cpp 中进行以下变更 ----

//设置Boss

void enter\_boss(int num){

if(num==0){//中路Boss开始的时候

memset(enemy,0,sizeof(enemy\_t)\*ENEMY\_MAX);//销毁杂兵

memset(shot,0,sizeof(shot\_t)\*SHOT\_MAX);//销毁弹幕

boss.x=FMX/2;//Boss的初始坐标

boss.y=-30;

if(stage\_count==boss.appear\_count[0])//最开始的话

boss.knd=-1;//弹幕的种类

}

boss.phy.flag=1;

boss.flag=1;

boss.hagoromo=0;//是否扇形扩展开的flag

boss.endtime=99\*60;//剩余时间

boss.state=1;//变为待机中状态

boss.cnt=0;

boss.graph\_flag=0;//恢复绘制flag

boss.knd++;

boss.wtime=0;//初始化待机时间

input\_phy(60);//60次计数内物理计算下回到固定位置

}

//Boss的弹幕main

void boss\_shot\_main(){

if((stage\_count==boss.appear\_count[0] || stage\_count==boss.appear\_count[1])

&& boss.flag==0){//如果到了开始时间

enter\_boss(0);//开始

}

if(boss.flag==0)//如果Boss没有登录的话直接返回

return;

calc\_boss();

if(boss.phy.flag==1)//如果物理计算移动为有效

calc\_phy();//进行物理计算

if(boss.state==2 && (boss.hp<=0 || boss.endtime<=0)){//弹幕中如果体力没有了的话

se\_flag[1]=1;//敌人的击毁音效

se\_flag[11]=1;

input\_phy(30);//30次计数内回到固定位置

memset(&boss\_shot,0,sizeof(boss\_shot\_t));//初始化Boss的弹幕信息

memset(&lazer,0,sizeof(lazer\_t)\*LAZER\_MAX);//初始化Boss的激光信息

flash.flag=0;//清除闪屏

if(boss.knd==boss.danmaku\_num[0] || boss.knd==boss.danmaku\_num[1]){//如果弹幕全部放完了

boss.flag=0;//销毁

enter\_dn(10,40);//(45)咚梆！的处理

se\_flag[9]=1;//Boss击毁声效

return ;

}

else

enter\_boss(1);//登录下一个弹幕

}

if(boss.state==1){//弹幕之间的待机时间

waitandenter();

}

if(boss.state==2){//如果在弹幕中的话

boss\_shot\_bullet[boss.knd]();//开始弹幕

boss\_shot\_calc();//弹幕计算

}

boss.cnt++;

}

运行结果

## 第四十六章 整理一下绘制函数的参数吧

一直到上一章位置，graph.cpp的DrawGraph和DrawRotaGraph函数之类的坐标都是○○+FX+dn.x这个样子的，我们必须每次都要把○○之外的多余部分写进去。

FX是为了让区域从0,0点稍微往右下方向错开而设置的X坐标分量。

而dn.x是画面咚！地摇动的时候的用于摇动控制的放入x坐标的变量。

这些东西要是每次都写的话就太麻烦了，而且毫无用处地横着把代码变得那么长看起来也不容易。

比如

void graph\_item(){

int i;

for(i=0;i<ITEM\_MAX;i++){

if(item[i].flag==1){

DrawRotaGraphF(item[i].x+FX+dn.x,item[i].y+FY+dn.y,item[i].r.....);

DrawRotaGraphF(item[i].x+FX+dn.x,item[i].y+FY+dn.y,item[i].r.....);

DrawRotaGraphF(item[i].x+FX+dn.x,item[i].y+FY+dn.y,item[i].r.....);

}

}

}

这些紫色的部分是每次都必须要写的部分，也是毫无意义的部分。

那么因此，

我们不妨这样考虑：“在进行必须这样写的处理的时候，干脆直接调用自动追加这些部分的函数不就好了？”

现在我们说DrawRotaGraph函数是直接在区域上绘制的版本（译者注：DrawRotaGraph函数是DX Library的函数，它始终使用GDI坐标系即左上角坐标为(0,0)，而现在要实现的版本是以游戏区域为原点进行绘制的版本），因此编写了DrawRotaGraphFdF这个函数。而且从现在开始我们就开始使用这些函数吧。

这个函数的参数和之前是一样的。只不过调用这些函数的时候，将绘制的坐标往上述的部分自动移动然后显示出来。

比如像下面这样将上面的函数用新的函数替换。

void graph\_item(){

int i;

for(i=0;i<ITEM\_MAX;i++){

if(item[i].flag==1){

DrawRotaGraphFdF(item[i].x,item[i].y,item[i].r....);

DrawRotaGraphFdF(item[i].x,item[i].y,item[i].r....);

DrawRotaGraphFdF(item[i].x,item[i].y,item[i].r....);

}

}

}

不过全部手动替换的话要全部变更就太麻烦了。

因此我们使用“替换”的功能。

graph.cpp中的

+FX

+FY

+dn.x

+dn.y

请把这些全部删除。

调出替换窗口，在搜索字符串窗口中输入上面那些文字，而置换部分什么都不写就行了。

接下来同样地在graph.cpp把下面的

DrawRotaGraphF　→　DrawRotaGraphFdF

DrawModiGraphF　→　DrawModiGraphFdF

像这样子替换。

graph.cpp的graph\_bullet()函数中的DrawGraph也变为DrawGraphFd。

在graph.cpp上进行以下追加

int DrawRotaGraphF( double xd, double yd, double ExRate, double Angle, int GrHandle, int TransFlag, int TurnFlag = FALSE ) {

return DrawRotaGraphF((float)xd,(float)yd,ExRate,Angle,GrHandle,TransFlag,TurnFlag);

}

int DrawModiGraphF( double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4, int GrHandle, int TransFlag ){

return DrawModiGraphF( (float)x1, (float)y1, (float)x2, (float)y2, (float)x3, (float)y3, (float)x4, (float)y4, GrHandle, TransFlag ) ;

}

int DrawRotaGraphFdF( double xd, double yd, double ExRate, double Angle, int GrHandle, int TransFlag, int TurnFlag = FALSE ) {

return DrawRotaGraphF((float)xd+FX+dn.x,(float)yd+FY+dn.y,ExRate,Angle,GrHandle,TransFlag,TurnFlag);

}

int DrawModiGraphFdF( double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4, int GrHandle, int TransFlag ){

return DrawModiGraphF( (float)x1+FX+dn.x, (float)y1+FY+dn.y, (float)x2+FX+dn.x, (float)y2+FY+dn.y,

(float)x3+FX+dn.x, (float)y3+FY+dn.y, (float)x4+FX+dn.x, (float)y4+FY+dn.y, GrHandle, TransFlag ) ;

}

int DrawGraphFd( int xd, int yd, int GrHandle, int TransFlag) {

return DrawGraph(xd+FX+dn.x,yd+FY+dn.y,GrHandle,TransFlag);

}

这样就行了。

graph\_back.cpp的函数也请这样变更

#include "../include/GV.h"

extern int DrawGraphFd( int xd, int yd, int GrHandle, int TransFlag);

void graph\_back00(){//通常背景

SetDrawArea( FX , FY , FX+FMX , FY+FMY ) ;//设定可以绘制的区域

DrawGraphFd(0,count%700-700,img\_back[0],FALSE);

DrawGraphFd(0,count%700 ,img\_back[0],FALSE);

SetDrawArea( 0, 0, 640, 480);//还原区域

}

void graph\_back01(){//SC用背景

SetDrawArea( FX , FY , FX+FMX , FY+FMY ) ;//设定可以绘制的区域

DrawGraphFd(0,-count%480,img\_back[11],FALSE);

DrawGraphFd(0,-count%480+480,img\_back[11],FALSE);

DrawGraphFd(0,0,img\_back[10],TRUE);

SetDrawArea( 0, 0, 640, 480);//还原区域

}

void graph\_back\_main(){

//Boss如果有效且为SC得话

if(boss.flag==1 && boss.back\_knd[boss.knd]==1)

graph\_back01();

else//除此之外

graph\_back00();

}

如何？在此之前往参数传入的

+FX+dn.x,

+FY+dn.y,

全部都消失了，一样看去结构是不是变清晰了呢？

运行结果

本章没有动画。因为没有什么看上去的变化。

## 第四十七章 来制作一面吧

基本的部分我们全都做出来了。

作为继续对细小部分的调整，现在我们来试着大致地制作一面吧。

虽然这么说，由于计算处理、登录处理、绘制处理等处理早已完成了，因此现在我们只需要稍微修改Excel数据和设定部分就行了。

嘛，虽然说这个“稍微”在Excel数据上反映出来并不是那么容易就是了……

直到Stage的标题名显示为止的操作都和之前的一样。

敌人散乱地落下来并射出自机狙……

之类的，至于要做什么样子的Stage比起用文字说明还是您自行看看动画会比较好吧。

由于这次的动画长度稍微有点长，所以放到了别的页面里。

请点击这里的图像进入动画页面。

Excel数据的话就是这个感觉的：

/计数器 移动模式 敌人的种类 x坐标 y坐标 速度 发射时间 弹幕种类 子弹的颜色 体力 子弹种类 待机时间 道具1 2 3 4 5 6

/cnt pattern knd x y sp bltime blknd col hp blknd2 wait item\_n

100 0 0 40 -20 0 60 0 0 1 7 180 0 -1 -1 -1 -1 -1

一行以“/”开始的是注释。

计数器的值是在游戏画面左上角显示的数字。

嘛，细节的设定之前已经说过了。

建议您观看动画以确认各个值和运行结果动画是如何对应的。

此外，这里我按照自己的想法摆弄了一下，您也可以试试按照自己的想法自行安排。

这个时候请注意不要出现这种情况：改变了子弹的种类但是却指定了没有对应颜色的编号。

移动模式是“enemy\_act\_pattern.cpp”

弹幕的种类是“shotH.cpp”

子弹种类的图像文件夹是bullet文件夹

道具的种类的话您可以试着阅读out.cpp的第102行来进行各种调整。

另外，现在我们也进行了以下部分的变更。

----在 func.h 中进行以下部分的变更 ----

void (\*boss\_shot\_bullet[DANMAKU\_MAX])() =

{

//中路Boss

boss\_shot\_bulletH000,//normal

boss\_shot\_bulletH002,// SC~完美冻结~

//终路Boss

boss\_shot\_bulletH001,// normal~沉默的圣奈~

boss\_shot\_bulletH004,//SC~小小青蛙~

boss\_shot\_bulletH009,//SC~反魂蝶

//终止

boss\_shot\_bulletH000,//防止出现NullPointrException用（这个弹幕不会显示）（译者注：ぬるぽ这个词也能用说明Dixq没少逛2ch啊……）

};

----在 load.cpp 的 load\_story() 进行红字部分变更 ----

char fname[32]={"../dat/csv/47章/storyH0.csv"};

----在 ini.cpp 中进行红字部分变更 ----

/\*我们以后再在这里进行完整的设定\*/

int back\_knd[DANMAKU\_MAX]={

0,1,0,1,1,

};

int boss\_hp[DANMAKU\_MAX]={

5000,10000,8000,20000,30000,

};

/\*到此为止\*/

//游戏的初始化

void ini(){

stage\_count=1;

memset(&ch,0,sizeof(ch\_t));

memset(enemy,0,sizeof(enemy\_t)\*ENEMY\_MAX);

memset(lazer,0,sizeof(lazer\_t)\*LAZER\_MAX);

memset(enemy\_order,0,sizeof(enemy\_order\_t)\*ENEMY\_ORDER\_MAX);

memset(shot,0,sizeof(shot\_t)\*SHOT\_MAX);

memset(cshot,0,sizeof(cshot\_t)\*CSHOT\_MAX);

memset(effect,0,sizeof(effect\_t)\*EFFECT\_MAX);

memset(del\_effect,0,sizeof(del\_effect\_t)\*DEL\_EFFECT\_MAX);

memset(&bom,0,sizeof(bom\_t));

memset(&bright\_set,0,sizeof(bright\_set\_t));

memset(&dn,0,sizeof(dn\_t));

memset(&boss,0,sizeof(boss\_t));

memset(child,0,sizeof(child\_t)\*CHILD\_MAX);

memset(&stage\_title,0,sizeof(stage\_title\_t));

memset(item,0,sizeof(item\_t)\*ITEM\_MAX);

ch.x=FMX/2;

ch.y=FMY\*3/4;

ch.power=0;//初始化power(41)

ch.num=5;//初始化残机数(41)

stage\_title.appear\_cnt=stage\_title\_count[stage];

/\*弹幕的各自设定。放在后面好好设定吧。\*/

boss.appear\_count[0]=1650;//中路Boss出现的时刻(42)(47)

boss.appear\_count[1]=2850;//终路Boss出现的时刻(44)(47)

boss.danmaku\_num[0]=1;//中路Boss放多少个弹幕

boss.danmaku\_num[1]=4;//终路Boss放多少个弹幕 (47)

for(int i=0;i<DANMAKU\_MAX;i++){//弹幕的各自HP(47)

boss.set\_hp[i]=boss\_hp[i];

}

boss.hp\_max=boss.set\_hp[0];

for(int i=0;i<DANMAKU\_MAX;i++)//弹幕的各自背景种类

boss.back\_knd[i]=back\_knd[i];

/\*到此为止\*/

bright\_set.brt=255;//初期亮度最大

}

弹幕的顺序就是放入boss\_shot\_bullet中的顺序。

您可以对照着boss\_shotH.cpp来决定顺序。

back\_knd和boss\_hp是各自弹幕时候的背景种类以及敌人的体力。

在这里为了便于理解适当地放入了变量中，我们将在后面完整地进行处理。

如何？很好地运行了么？

是不是终于开始有一种这是游戏的感觉了呢？

下一章我们将会让音乐也播放出来让游戏更有气氛。

## 第四十八章 试着加上音乐吧

本章中我们试着加上音乐吧。

音乐的话这里我使用了炼狱庭园的曲子。

不妨请听听dat文件夹的music文件夹中中放入的曲子。

如果您打不开ogg文件的话，请安装GOM Player之类的软件然后来听。

虽然这音乐听起来不像是1面的曲子，您也可以变成您喜欢的曲子。

这一章我们使用

SetLoopPosSoundMem这个函数。

虽然用PlaySoundMem函数也能够播放曲子，但是有个问题，那就是只要是曲子，大多数都是有前奏的。

因此，想要在游戏内循环重播、且曲子到达某个部分之后要回到最初的时候，那么会直接回到整个曲子的开头这很不方便了。

为了解决这个问题，我们使用这个函数来让循环播放时跳回去的地方定位到指定的毫秒位置。

我们在Audacity之类的软件里面确认曲子的应该是循环位置的部分吧。

比如试着将终路Boss的曲子0\_1.ogg中像是循环位置的部分放大来看之后，就会发现如果把26.672秒也即26672毫秒设置为循环位置的话就行了。

--------------------------------------------------------------------

这一次，我们定义了area\_t类型的结构体，对于每一个区域（译者注：这里包括下文中的区域指的是每一面）中需要保存的信息都要用到一个这个结构体来保存。

在其中我们定义了bgm\_t型结构体，它是与BGM相关的变量的集合。

----在 struct.h 中进行以下追加 ----

//BGM信息(在area\_t中使用(48)

typedef struct{

//BGM的播放flag、播放句柄、播放时机的种类编号、循环位置

int flag,handle,knd[2],loop\_pos[2];

}bgm\_t;

//区域所持有的信息

typedef struct{

bgm\_t bgm;

}area\_t;

----在 GV.h 中进行以下追加 ----

GLOBAL area\_t area;//区域所持有的数据 (48)

----在 function.h 中进行以下追加 ----

GLOBAL void delete\_bgm();

GLOBAL void bgm\_main();

----在 load.cpp 变更以下部分 ----

适当地调整了ChangeVolumeSoundMem函数内的音量。详情省略。

在bgm\_t中定义了

flag,　handle,　knd[2],　loop\_pos[2];

这些变量。音乐的播放状态按照下面那样0~4的状态依次变化。

0:还没有播放

1:通常BGM播放中

2:通常BMG播放停止中

3:终路Boss的BGM播放中

4:终路Boss的BGM播放停止中

状态0是当前区域的游戏刚开始的状态。这个时候读入数据。

在状态1下何时开始播放由knd来设定。在样例中种类0的重播方式是开始之后60次计数时开始。

由于我们希望对于每一个设定了的种类都要变更重播的时机，因此今后我们在music.cpp中增加重播的方式吧。

状态2是终路Boss临近的状态。此时停止通常BMG，删除数据，读入终路Boss的音乐数据并待机。

状态3则是开始播放。

状态4的时候虽然是停止，但是停止的处理还没有在示例中完成。

loop\_pos是前面说过的循环位置。在ini函数中设置。

[0]是一般BGM的设定，[1]是终路Boss的BGM的设定。

另外，在load\_bgm函数中也写好路径了。

如果我们将

0\_0作为1面的通常BGM

0\_1作为1面的终路Boss的BGM

1\_0作为2面的通常BGM

1\_1作为2面的终路Boss的BGM

的话，那么我们就可以利用这个函数沿袭着继续扩展下去了。

----在 ini.cpp 中进行以下追加 ----

memset(&area,0,sizeof(area\_t));//(48)

area.bgm.knd[0]=0;//通常BGM在什么时刻播放取决于music.cpp(48)

area.bgm.knd[1]=0;// 终路Boss的BGM在什么时刻播放取决于music.cpp(48

area.bgm.loop\_pos[0]=26672;//通常BGM的循环位置 (48)

area.bgm.loop\_pos[1]=27010;//Boss的BGM的循环位置(48)

----在 GV.h 中进行以下追加 ----

GLOBAL area\_t area;//当前区域所持有的数据 (48)

----在 function.h 中进行以下追加 ----

GLOBAL void delete\_bgm();

GLOBAL void bgm\_main();

----在 load.cpp 中进行以下部分变更 ----

在ChangeVolumeSoundMem函数中适当地调整了音量。详情省略。

----在 music.cpp 中进行以下追加 ----

void load\_bgm(int flag){

char str[64];

sprintf(str,"../dat/music/%d\_%d.ogg",stage,flag);

area.bgm.handle=LoadSoundMem(str);

}

void delete\_bgm(){

DeleteSoundMem(area.bgm.handle);

}

void bgm\_main(){

if(area.bgm.flag==0){//最开始读入

load\_bgm(0);

SetLoopPosSoundMem( area.bgm.loop\_pos[0], area.bgm.handle );//设置循环位置

area.bgm.flag=1;

}

if(area.bgm.flag==1){//按照种类开始播放

switch(area.bgm.knd[0]){

case 0://如果是播放种类0的话

if(stage\_count==60){//开始60次计数后开始播放

PlaySoundMem(area.bgm.handle,DX\_PLAYTYPE\_LOOP);

area.bgm.flag=2;

}

break;

default:

break;

}

}

if(area.bgm.flag==2){//在Boss出现时读入

if(stage\_count==boss.appear\_count[1]){//终路Boss出现的话读入

StopSoundMem(area.bgm.handle);

delete\_bgm();

load\_bgm(1);

SetLoopPosSoundMem( area.bgm.loop\_pos[1], area.bgm.handle );//设置循环位置

area.bgm.flag=3;

}

}

if(area.bgm.flag==3){

switch(area.bgm.knd[0]){

case 0://如果是播放种类0的话

if(stage\_count==boss.appear\_count[1]){//终路Boss出现后再播放

PlaySoundMem(area.bgm.handle,DX\_PLAYTYPE\_LOOP);

area.bgm.flag=4;

}

break;

default:

break;

}

}

}

----在 main.cpp 中追加以下红字部分 ----

case 100://通常处理

enter\_func\_tm("最初");

calc\_ch(); enter\_func\_tm("角色计算");

ch\_move(); enter\_func\_tm("角色移动");

cshot\_main(); enter\_func\_tm("自机射击main");

enemy\_main(); enter\_func\_tm("敌人处理main");

boss\_shot\_main();enter\_func\_tm("Boss射击main");

shot\_main(); enter\_func\_tm("射击main");

out\_main(); enter\_func\_tm("碰撞判定");

effect\_main(); enter\_func\_tm("特效main");

calc\_main(); enter\_func\_tm("计算main");

graph\_main(); enter\_func\_tm("绘制main");

bgm\_main(); enter\_func\_tm("BGMmain");

if(boss.flag==0)

stage\_count++;

break;

default:

printfDx("错误的func\_state\n");

break;

运行结果

也就是有没有音乐的区别，因此看起来没有区别。

※试着练习吧※

不妨运行这个工程，在Boss战中故意优哉一下听听BGM。这样应该就能够明白是在哪里进行循环的了。

这首曲子是炼狱庭园的曲子“裏桜華-CODE：桜樺-”。

http://www.rengoku-teien.com/mp3\_wahuu.html

您可以下载曲子，然后自己对曲子进行编辑，如果能够编辑、设定到无缝循环的程度的话就非常不错了。

※管理员的话※

在龙神录中考虑到容量的问题，出于无奈只好把这首“裏桜華-CODE：桜樺-”进行剪短再循环了。

然而这首曲子有4分多钟的长度，前半部分和后半部分都非常好听。

如果您已经多次玩过龙神录了，然后听了全曲的话可能会有“诶？原来这首曲子还有这样一些部分啊”这样子的想法。

这首曲子真的很不错，可以的话请务必从头至尾地听一遍。

## 第四十九章 为自机的射击加上Option吧

（译者注：Option即オプション，在东方之类的STG里面有特殊意义的东西，说简单点就是选择自机的时候选到的武器，而后面出现的Bonbon也就是围绕在自机周围放子弹的东西，具体可以看图，这里我不知道汉化版中的通用称谓是什么，这里我直接写为Option）

这一次我们为自机的射击加上Option吧。

自动追踪的处理我们将在下一章中讲解，本章中我们先准备好发射Option所需要的Bonbon，然后显示它，最后再来发射Option。

那么和之前一样，我们来定义Option的Bonbon的结构体option\_bb，声明并计算之。

由于Option的图像现在是固定的，因此在初始化函数中就事先设定好了。

----在 struct.h 中进行一下追加 ----

//和角色射击相关的结构体

typedef struct{

int flag; //flag

int power; //power

int cnt; //计数器

int knd; //种类

int img; //图像(49)

double x,y; //坐标

double angle; //角度

double spd; //速度

}cshot\_t;

//射击的Option的发射源Bonbon

typedef struct{

int flag; //flag

int img; //图像

int cnt; //计数器

double x,y; //坐标

}option\_bb\_t;

----在 GV.h 中进行以下追加 ----

GLOBAL int img\_cshot[3]; //自机射击用图像(49)

GLOBAL option\_bb\_t option\_bb[2];//Option的发射源Bonbon (49)

----在 load.cpp 中进行以下追加 ----

img\_cshot[2]=LoadGraph("../dat/img/char/ball.png");

----在 ini.cpp 中进行以下部分变更 ----

memset(option\_bb,0,sizeof(option\_bb\_t)\*2);//(49)

option\_bb[0].img=img\_cshot[2];//代入Option的Bonbon的图像 (49)

option\_bb[1].img=img\_cshot[2];

Option是独立地在运动，发射时从Option开始。

自机的power在100~299的时候从Option发射2路子弹。

超过299的话发射4路子弹。

Option的基本位置在定义中已经写好了，4路子弹的各自的角度在option0ang中设定。

Option的Bonon在calc\_option\_bb中计算。

低速移动中将Option靠近自机并固定不动。

----在 cshot.cpp 中进行以下追加 ----

#define OPTION\_X 25

#define OPTION\_Y 35

int option0num[2] ={2,4};

double option0ang[4] = {-PI/2,-PI/2,-PI/2-PI/4,-PI/2+PI/4};//Option的发射角度(49)

//返回自机射击可以登录的编号

int search\_cshot(){

for(int i=0;i<CSHOT\_MAX;i++){

if(cshot[i].flag==0)

return i;

}

return -1;

}

//登录通常射击

void ch0\_shot\_pattern(){

int k;

for(int i=0;i<cshot0num[ch.power<200?0:1];i++){

if((k=search\_cshot())!=-1){

cshot[k].flag=1;

cshot[k].cnt=0;

cshot[k].angle=-PI/2;

cshot[k].spd=20;

cshot[k].x=ch.x+cshot0pos\_x[i];

cshot[k].y=ch.y+cshot0pos\_y[i];

cshot[k].power=23;

cshot[k].knd=0;

cshot[k].img=img\_cshot[0];//图像编号代入(49)

}

}

se\_flag[2]=1;//发射音有效

if(ch.power>=100){

for(int i=0;i<(ch.power<300?2:4);i++){

if((k=search\_cshot())!=-1){

cshot[k].flag=1;

cshot[k].cnt=0;

cshot[k].angle=option0ang[i];

cshot[k].spd=20;

cshot[k].x=option\_bb[i%2].x;

cshot[k].y=option\_bb[i%2].y;

cshot[k].power=20;

cshot[k].knd=1;

cshot[k].img=img\_cshot[1];//画像番号代入(49)

}

}

}

}

//低速通常射击登录

void ch1\_shot\_pattern(){

int k;

for(int i=0;i<cshot0num[ch.power<200?0:1];i++){

if((k=search\_cshot())!=-1){

cshot[k].flag=1;

cshot[k].cnt=0;

cshot[k].angle=-PI/2;

cshot[k].spd=20;

cshot[k].x=ch.x+cshot0pos\_x[i]/3;//如果低速中的话将位置往中心偏移

cshot[k].y=ch.y+cshot0pos\_y[i]/2;

cshot[k].power=23;

cshot[k].knd=0;

cshot[k].img=img\_cshot[0];//图像编号代入(49)

}

}

se\_flag[2]=1;

if(ch.power>=100){

for(int i=0;i<(ch.power<300?2:4);i++){

if((k=search\_cshot())!=-1){

cshot[k].flag=1;

cshot[k].cnt=0;

cshot[k].angle=option0ang[i];

cshot[k].spd=20;

cshot[k].x=option\_bb[i%2].x;

cshot[k].y=option\_bb[i%2].y;

cshot[k].power=20;

cshot[k].knd=1;

cshot[k].img=img\_cshot[1];//图像编号代入(49)

}

}

}

}

//Option的Bonbon计算

void calc\_option\_bb(){

if(ch.power>=100){//power在100以上的话增加Option

for(int i=0;i<2;i++){

if(CheckStatePad(configpad.slow)==0){//如果不是低速移动中的话

option\_bb[i].x=ch.x+OPTION\_X\*(i==0 ? -1 : 1);

option\_bb[i].y=ch.y+OPTION\_Y+sin(PI2/150\*option\_bb[i].cnt)\*20;

}

else{

option\_bb[i].x=ch.x+OPTION\_X/2\*(i==0 ? -1 : 1);

option\_bb[i].y=ch.y+OPTION\_Y/2;

}

option\_bb[i].cnt++;

}

if(option\_bb[0].flag==0)

option\_bb[0].flag = option\_bb[1].flag = 1;

}

else{

if(option\_bb[0].flag==1)

option\_bb[0].flag = option\_bb[1].flag = 0;

}

}

(略)

//和角色射击相关的函数

void cshot\_main(){

calc\_option\_bb();//Option的Bonon计算 (49)

calc\_cshot();//射击的启动计算

enter\_shot();//登录射击

}

----在 graph.cpp 中进行以下变更 ----

//自机绘制

void graph\_ch(){

double sx,sy,ny=(sin(2.0\*PI\*(count%50)/50)\*3),ang=2.0\*PI\*(count%120)/120;

if(CheckStatePad(configpad.slow)>0)//低速移动中的话

sx=15,sy=15+ny;//拉近

else

sx=30,sy=30+ny;//在普通的位置

// DrawRotaGraphFdF( ch.x-sx, ch.y+sy, 1.0f, ang, img\_chetc[2], TRUE );

// DrawRotaGraphFdF( ch.x+sx, ch.y+sy, 1.0f, -ang, img\_chetc[2], TRUE );

if(ch.mutekicnt%2==0){//无敌状态的话亮灭

//自机显示

DrawRotaGraphFdF(ch.x,ch.y,1.0f,0.0f,img\_ch[0][ch.img],TRUE);

if(CheckStatePad(configpad.slow)>0)//低速移动中的话显示碰撞判定（译者注：即碰撞判定点）

DrawRotaGraphFdF( ch.x, ch.y, 1.0f, 2.0\*PI\*(count%120)/120, img\_chetc[0], TRUE );

}

}

//自机射击的绘制

void graph\_cshot(){

for(int i=0;i<CSHOT\_MAX;i++){

if(cshot[i].flag>0){

DrawRotaGraphFdF(cshot[i].x,cshot[i].y,1,cshot[i].angle+PI/2,cshot[i].img,TRUE);

}

}

for(int i=0;i<2;i++){//(49)

if(option\_bb[i].flag>0){

DrawRotaGraphFdF(option\_bb[i].x,option\_bb[i].y,1,0,option\_bb[i].img,TRUE);

}

}

}

运行结果

自动最终处理将会在下一章中追加。

## 第五十章 增加自动追踪的功能吧

在49章中我们为自机的射击增加了Option了呢。这一章我们试着为射击增加自动追踪的处理吧。

各个子弹如果会寻找最就近的敌人，然后往敌人的方向飞去。

两个点与点的角度计算和之前一样使用atan2函数。

自动最终处理有必要分为Boss存在与Boss不存在两种情况。

即便boss.flag为有效的，由于当Boss的HP为0的时候不存在碰撞判定，Boss的flag为有效且Boss的HP大于0的时候我们才进行Boss的自动追踪处理。

其次，自机射击这个东西，单纯地进行增加是不行的（译者注：指的是单纯地增加子弹数）。

刚开始是1束子弹，但每次power up之后会像2束……6束这样子增加，在这个时候会单纯地变为之前6倍的强度。

如果这样的话，在power max的情况下10秒钟能解决的敌人在power为0的时候要花1分钟。

实际上总体的强度的增加并不是1倍、2倍……这样子增加，而是以0.1倍、0.2倍为单位来增加，这样的话会比较合理。

因此，在同时发射的子弹数量增加的同时，我们同时进行削减子弹威力的处理吧。

（如果您认为“power up反而要削弱子弹的威力？”有点不可思议的话，我在这里多说明一下。比如从1束子弹变为2束（同时发射2个子弹）的时候，假设1束子弹每1个子弹的power是10的话，如果我们把2束子弹的每1个子弹的power降低6，那么整体来看其实是增加了1.2倍。也就是这个道理。）

如果子弹变得更加有力了的话，作为玩家仅凭这一点也就能够产生的确变强了的印象。

power up的时候，实际上子弹的威力并不是突然就增加，为了让展示显得没有跳跃，我们让子弹的威力在一段时间内微微地进行提升。

由于这里的调整如果不一边玩一边调整的话会很困难的，不管怎么说，那边的调整也就先做到合适就行了。

----在 cshot.cpp 中进行以下追加 ----

//通常射击登录

void ch0\_shot\_pattern(){

int k;

int n=cshot0num[ch.power<200?0:1];

for(int i=0;i<n;i++){

if((k=search\_cshot())!=-1){

cshot[k].flag=1;

cshot[k].cnt=0;

cshot[k].angle=-PI/2;

cshot[k].spd=20;

cshot[k].x=ch.x+cshot0pos\_x[i];

cshot[k].y=ch.y+cshot0pos\_y[i];

cshot[k].power= (n<2 ? 20 : 12) + ch.power/100;//2束子弹的话20,4束子弹的话12 (50)

cshot[k].knd=0;

cshot[k].img=img\_cshot[0];//图像编号代入(49)

}

}

se\_flag[2]=1;//发射声有效

if(ch.power>=100){

for(int i=0;i<(ch.power<300?2:4);i++){

if((k=search\_cshot())!=-1){

cshot[k].flag=1;

cshot[k].cnt=0;

cshot[k].angle=option0ang[i];

cshot[k].spd=20;

cshot[k].x=option\_bb[i%2].x;

cshot[k].y=option\_bb[i%2].y;

cshot[k].power=10-7\*(i/2);//注意不要为负数(50)

cshot[k].knd=1;

cshot[k].img=img\_cshot[1];//图像编号代入 (49)

}

}

}

}

//低速通常射击登录

void ch1\_shot\_pattern(){

int k;

int n=cshot0num[ch.power<200?0:1];

for(int i=0;i<n;i++){

if((k=search\_cshot())!=-1){

cshot[k].flag=1;

cshot[k].cnt=0;

cshot[k].angle=-PI/2;

cshot[k].spd=20;

cshot[k].x=ch.x+cshot0pos\_x[i]/3;//低速中的话将位置往中心偏移

cshot[k].y=ch.y+cshot0pos\_y[i]/2;

cshot[k].power= (n<2 ? 20 : 12) + ch.power/100;//2束的话20,4束的话12(50)

cshot[k].knd=0;

cshot[k].img=img\_cshot[0];//图像编号代入(49)

}

}

se\_flag[2]=1;

if(ch.power>=100){

for(int i=0;i<(ch.power<300?2:4);i++){

if((k=search\_cshot())!=-1){

cshot[k].flag=1;

cshot[k].cnt=0;

cshot[k].angle=option0ang[i];

cshot[k].spd=20;

cshot[k].x=option\_bb[i%2].x;

cshot[k].y=option\_bb[i%2].y;

cshot[k].power=10-7\*(i/2);//注意不要为负数(50)

cshot[k].knd=1;

cshot[k].img=img\_cshot[1];//图像编号代入(49)

}

}

}

}

（略）

/寻找就近的敌人并设置角度

void calc\_homing(int k){

int i,d, num=-1, min=-1;

double x,y;

if(boss.flag==0){//Boss不在的时候

for(i=0;i<ENEMY\_MAX;i++){//敌人的总数次

if(enemy[i].flag==1){

x=(enemy[i].x-cshot[k].x);

y=(enemy[i].y-cshot[k].y);

d=(int)(x\*x+y\*y);//子弹与敌人的距离

if(d<min || min==-1){//计算结果为最小值或者还没有保存任何值

num=i;//编号记录

min=d;//距离记录

}

}

}

}

//如果找到就近的敌人的话，又或者Boss存在且HP不为0的时候设置角度

if(num!=-1 || boss.flag==1 && boss.hp>0){

if(boss.flag==0){

x=enemy[num].x-cshot[k].x;

y=enemy[num].y-cshot[k].y;

}

else{

x=boss.x-cshot[k].x;

y=boss.y-cshot[k].y;

}

cshot[k].angle=atan2(y,x);

}

}

//射击的移动计算

void calc\_cshot(){

for(int i=0;i<CSHOT\_MAX;i++){

if(cshot[i].flag==1){

int dranx=(int)(cshot[i].spd+11/2),drany=(int)(cshot[i].spd+55/2);

if(cshot[i].knd==1)

calc\_homing(i);

cshot[i].x+=cos(cshot[i].angle)\*cshot[i].spd;

cshot[i].y+=sin(cshot[i].angle)\*cshot[i].spd;

cshot[i].cnt++;

if(cshot[i].x<-dranx || cshot[i].x>FMX+dranx ||

cshot[i].y<-drany || cshot[i].y>FMY+drany)//在画面外往外跑的话

cshot[i].flag=0;

}

}

}

运行结果

## 番外篇第三章~试着完成弹幕——“正直者之死”

我接受了制作东方永夜抄的弹幕“正直者之死”的委托而完成了这个弹幕，因为很不容易所以向大家介绍一下。

和之前一样，因为是番外篇，因此不看这一章也没问题。

(↓niconico动画在这里)

程序的介绍就省略了，如果有兴趣的话您可以直接看看工程里面的源代码。

因为新增加了一些子弹，所以在之前工程的基础上频繁地进行了变更。

## 第五十一章 来制作完成汉字弹幕的工具吧（1）

各位，好久不见。

在四圣龙神录Plus中

出现了这种风格的，图片和文字的弹幕。因为我收到了多个拜托我把这种弹幕的制作方法介绍一下的委托，因此我想就在这里开始讲解吧。

那么，我想既然这一章是“制作完成汉字弹幕的工具吧”，那么和龙神录的工程就完全不一样了而是另外一个“辅助工具制作”的工程了，因此在这一章公众我们不使用前面我们一直在使用的工程。

我们使用每次写程序都用到的框架<http://dixq.net/g/#41>，从头开始制作。如果是手动创建工程的话，请自行创建一个新的工程。

至于这工具要做成什么样子，首先请仔细看一下。另外，由于这个工具会在下一章中完成，因此这一章的运行结果看起来就有一种alpha版本的感觉。

就象这样，在下面先铺上自己想要描绘的汉字的图像，然后在上面像临摹一样放置子弹。

子弹的放置，就像在画图工具中拖拉的间断线那样。

每点击一次鼠标，就在鼠标的指针那里引出一条线，在下一次单击的时候就以单击的位置为终点在一条直线上放置子弹。

也即是说，如果两次在同一个地方单机的话那么就只在那里放置一个子弹，如果想在一条直线放置子弹的话，那就像拖拉间断线那样做就行了。

另外，当你正在专心致志地进行汉字数据的制作，在将要完成的时候突然发现“啊，画错了”，但是这个时候却没办法消去或者撤销，那可就泪奔了。

因此，在后面我们试着实现撤销的功能吧。

接下来，可能会有人觉得这会很困难，不过其实这很简单的。

我简单地说明一下处理的流程。

～鼠标部分～

查询鼠标是否单击

如果是第1次单击鼠标，那么记下那个坐标。

如果是第2次单击鼠标，那么从第1次单击的地方开始向第2次单击的地方前进，同时反复进行这样子的工作：空出一定的距离然后登陆一次子弹的坐标。

～绘制部分～

绘制背景

绘制已经登录了的子弹

如果已经是第1次单击的状态的话，那么从已经记忆了的地方开始往当前鼠标指针所在的坐标引线，仅此而已。

那么，实际的操作请直接看程序部分吧。

由于这次我们使用了鼠标。因此我们使用了这样一些函数。

监视鼠标按键状态的函数

由于这个函数的内部处理不用清楚了解就能够使用，因此我就不做特别说明了。

另外，因为这次并不是和龙神录的工程一起说明的东西，所以把文件给分开了。

--- Key\_Mouse.h ---

typedef struct{

int x;

int y; //坐标

unsigned int Button[8]; //按钮的按键状态

int WheelRotVol;//滚轮的旋转量

}Mouse\_t;

int GetHitKeyStateAll\_2(int GetHitKeyStateAll\_InputKey[]){

char GetHitKeyStateAll\_Key[256];

GetHitKeyStateAll( GetHitKeyStateAll\_Key );

for(int i=0;i<256;i++){

if(GetHitKeyStateAll\_Key[i]==1) GetHitKeyStateAll\_InputKey[i]++;

else GetHitKeyStateAll\_InputKey[i]=0;

}

return 0;

}

int GetHitMouseStateAll\_2(Mouse\_t \*Nezumi){

if(GetMousePoint( &Nezumi->x, &Nezumi->y ) == -1){ //获得鼠标位置

return -1;

}

int MouseInput=GetMouseInput(); //获得鼠标的按键状态

for(int i=0; i<8; i++){ //确认鼠标最多8个按键

if( (MouseInput & 1<<i ) != 0 ) Nezumi->Button[i]++; //如果有按键的话计数器增加

else Nezumi->Button[i] = 0; //如果没有按键的话就是0

}

Nezumi->WheelRotVol = GetMouseWheelRotVol() ; //获得滚轮的旋转量

return 0;

}

好的，我们就用这个文件名保存它。随便保存到哪里都行。

如果想要完全不做改变而使用这个工程的话，请参考下载了的工程。放在“mydata/source”里面。

接下来我们来说明定义的结构体。

//float型的坐标结构体

typedef struct{

float x,y;

}fPt\_t;

//每个子弹的信息

typedef struct{

int Knd;//子弹的种类

int Col;//子弹的颜色

float Angle;//子弹的角度

float x,y;

}Bl\_t;

//子弹全体的信息

typedef struct{

int Num;//登录了的个数

Bl\_t Bl[ PMAX ]; //登录的子弹的信息

}BlPoint\_t;

//操作设定部分信息

typedef struct{

int State;//状态

int Knd;//种类

int Col;//颜色

int Space;//空白

float Angle;//角度

int flag;//是否显示的flag

fPt\_t fPt1;//第1次单击的地方

fPt\_t fPt2;//第2次单击的地方

}Operate\_t;

这次就像这个样子定义4个结构体。第1个结构体就不再多说。全部都和注释说明的一样，第2个结构体用于保存每个子弹的信息，第3个结构体用于设置全体子弹，最后一个是和操作设定有关的结构体。

看起来并不是很难嘛。由于子弹有种类、颜色、角度以及坐标这些属性因此放到Bl\_t中。Bl即是Bullet。

至于BlPoint\_t，假如我们用PMAX表示构成汉字的子弹的数量的话，那么就有必要记录这PMAX的一整个信息，因此我们定义了这个结构体。

Num中传入登录了的子弹的个数

关于设定部分正如你所见。都是与放置什么样子的子弹的设定之类相关。

那么，我们来看看登录子弹时候的处理吧。

第2次单击的时候，就从第1次单击的坐标（以后称“第1次的坐标”）的位置开始以某个间隔在一条直线上放置子弹。

具体而言，就是用atan2来计算用第1次的坐标和第2次的坐标的角度，这个和射击的移动是一个道理，将第1次的坐标作为初始地点，在计算好了的角度下往第2次的坐标的方向进行类似飞过去那样前进就行了。

在设定项目中设定的间隔“Space”即是速度。我们以此在每一个计算好了的地方放置空白那么子弹也就放好了。

一开始就把第一次和第二次坐标之间的距离计算好，然后按照之前说明的顺序反复进行登录，要是移动的距离超过了的话那就停止登录。

我们使用以下三个函数来进行上述计算。

//登录子弹

void InputBlData(float x, float y, int Knd, int Col, float Angle){

BlPoint.Bl[BlPoint.Num].x = x;//记录当前的位置

BlPoint.Bl[BlPoint.Num].y = y;

BlPoint.Bl[BlPoint.Num].Knd = Knd;//子弹的种类

BlPoint.Bl[BlPoint.Num].Col = Col;//颜色

BlPoint.Bl[BlPoint.Num].Angle = Angle;//角度

BlPoint.Num++;//表示当前有多少个子弹的计数器自增

}

//放置子弹时候的计算

void CalcBullet(){

float x = Operate.fPt1.x, y = Operate.fPt1.y;//最开始的位置

//最开始单击的位置和最后单击的位置之间的角度

float Angle = atan2( Operate.fPt2.y - Operate.fPt1.y, Operate.fPt2.x - Operate.fPt1.x );

float xlen = Operate.fPt2.x - Operate.fPt1.x;//x的距离

float ylen = Operate.fPt2.y - Operate.fPt1.y;//y的距离

float Length = sqrt( xlen \* xlen + ylen \* ylen );//点与点的距离

float Proceeded = 0;//当前前进了的距离

//当前前进的距离在允许前进的距离以内的区间内，且在登录可能的个数下循环

while( BlPoint.Num < PMAX ){

InputBlData(x,y,Operate.Knd,Operate.Col,Operate.Angle);

x += cos( Angle ) \* Operate.Space;//往应该前进的方向前进

y += sin( Angle ) \* Operate.Space;

Proceeded += Operate.Space;//合计已经前进的距离

if(Length < Proceeded) break;

}

}

//鼠标的计算

void CalcMouse(){

if( Mouse.Button[0]==1 ){//如果单击了左键

switch( Operate.State ){

case 0://第1次单击的时候

//记录那个时候的位置

Operate.fPt1.x = (float)Mouse.x;

Operate.fPt1.y = (float)Mouse.y;

Operate.State = 1;

break;

case 1://第2次单击的时候

//记录那个时候的位置

Operate.fPt2.x = (float)Mouse.x;

Operate.fPt2.y = (float)Mouse.y;

Operate.State = 0;

//从第1次单击的位置开始到第2次单击的位置为止登录子弹

CalcBullet();

break;

}

}

}

绘制的部分非常简单。

绘制背景、进行登录了的子弹个数次绘制，如果是单击了的状态的话，就从记录了的坐标开始到当前鼠标指针的位置绘制线段就行了。

内容如下。

//显示数据

void Show(){

int i;

//绘制背景

DrawGraph(0,0,ImgBack,FALSE);

//绘制子弹

for(i=0; i<BlPoint.Num; i++){

DrawCircle( (int)BlPoint.Bl[ i ].x, (int)BlPoint.Bl[ i ].y, 3, Red, TRUE );

}

//绘制引出来的线

if( Operate.State==1 ){

DrawLine( (int)Operate.fPt1.x, (int)Operate.fPt1.y, Mouse.x, Mouse.y, Red );

}

//在鼠标指针位置显示子弹

DrawCircle( Mouse.x, Mouse.y, 3, Red, TRUE );

//绘制完子弹之后，再绘制有多少当前设定了的空格

DrawLine(Mouse.x,Mouse.y,Mouse.x+Operate.Space,Mouse.y,Blue);

}

之后是main函数。

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,LPSTR lpCmdLine, int nCmdShow ){

ChangeWindowMode(TRUE);//窗口模式

SetGraphMode(640,640,32);//画面尺寸变更

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初始化与里表面化

SetMouseDispFlag( TRUE ) ;//鼠标显示有效

ini();//初始化

load();//载入

while(ProcessMessage()==0 /\*消息处理\*/&& ClearDrawScreen()==0/\*清空画面\*/ && GetHitKeyStateAll\_2(Key)==0 /\*保存输入状态\*/&& Key[KEY\_INPUT\_ESCAPE]==0/\*没有按下ESC \*/){

GetHitMouseStateAll\_2(&Mouse);

CalcMouse();//鼠标计算

CalcOperate();//操作计算

Show();//显示

ScreenFlip();

}

DxLib\_End();

return 0;

}

数据载入部分之类的在这里就省略不写了，这样子就是所有了。那么最后清除地来看一下整体吧。

--- main.cpp ---

#include "../../../include/DxLib.h"

#include "math.h"

#include "Key\_Mouse.h"

#define PI2 (3.141562f\*2) //圆周率\*2

#define PMAX 1000 //登录的子弹的最大数量

//float型坐标结构体

typedef struct{

float x,y;

}fPt\_t;

//单个子弹的信息

typedef struct{

int Knd;//子弹的种类

int Col;//子弹的颜色

float Angle;//子弹的角度

float x,y;

}Bl\_t;

//子弹全体信息

typedef struct{

int Num;//已登录的个数

Bl\_t Bl[ PMAX ]; //登录的子弹的信息

}BlPoint\_t;

BlPoint\_t BlPoint;

//操作设定类情报

typedef struct{

int State;

int Knd;

int Col;

int Space;

float Angle;

int flag;

fPt\_t fPt1;

fPt\_t fPt2;

}Operate\_t;

int Key[256];//按键

int Red,White,Blue;//颜色

int ImgBullet[14][10],ImgBack;//子弹的图像和背景图像

Mouse\_t Mouse;//鼠标

Operate\_t Operate;//操作设定

//初始化

void ini(){

Operate.Knd=7;//将子弹的初始种类设置为7

Operate.Space=20;//Spac最开始为20

Operate.flag=1;

}

//载入

void load(){

White=GetColor(255,255,255);

Red = GetColor(255,0,0);

Blue = GetColor(0,255,255);

ImgBack = LoadGraph("mydat/img/ryu.png");

}

//登录子弹

void InputBlData(float x, float y, int Knd, int Col, float Angle){

BlPoint.Bl[BlPoint.Num].x = x;//记录当前位置

BlPoint.Bl[BlPoint.Num].y = y;

BlPoint.Bl[BlPoint.Num].Knd = Knd;//子弹的种类

BlPoint.Bl[BlPoint.Num].Col = Col;//颜色

BlPoint.Bl[BlPoint.Num].Angle = Angle;//角度

BlPoint.Num++;//表示当前有多少个子弹的计数器自增

}

//放置子弹时候的计算

void CalcBullet(){

float x = Operate.fPt1.x, y = Operate.fPt1.y;//最开始的地点

//最开始单击的位置和最后单击的位置的角度

float Angle = atan2( Operate.fPt2.y - Operate.fPt1.y, Operate.fPt2.x - Operate.fPt1.x );

float xlen = Operate.fPt2.x - Operate.fPt1.x;//x的距离

float ylen = Operate.fPt2.y - Operate.fPt1.y;//y的距离

float Length = sqrt( xlen \* xlen + ylen \* ylen );//点与点的距离

float Proceeded = 0;//当前已经前进的距离

//当前前进的距离在允许前进的距离以内的区间内，且在登录可能的个数下循环

while( BlPoint.Num < PMAX ){

InputBlData(x,y,Operate.Knd,Operate.Col,Operate.Angle);

x += cos( Angle ) \* Operate.Space;//往应该前进的方向前进

y += sin( Angle ) \* Operate.Space;

Proceeded += Operate.Space;//合计已经前进的距离

if(Length < Proceeded) break;

}

}

//鼠标的计算

void CalcMouse(){

if( Mouse.Button[0]==1 ){//如果单击了左键

switch( Operate.State ){

case 0://第1次单击的时候

//记录那个时候的位置

Operate.fPt1.x = (float)Mouse.x;

Operate.fPt1.y = (float)Mouse.y;

Operate.State = 1;

break;

case 1://第2次单击的时候

//记录那个时候的位置

Operate.fPt2.x = (float)Mouse.x;

Operate.fPt2.y = (float)Mouse.y;

Operate.State = 0;

//从第1次单击的位置开始到第2次单击的位置为止登录子弹

CalcBullet();

break;

}

}

}

void CalcOperate(){

}

//显示数据

void Show(){

int i;

//绘制背景

DrawGraph(0,0,ImgBack,FALSE);

//绘制子弹

for(i=0; i<BlPoint.Num; i++){

DrawCircle( (int)BlPoint.Bl[ i ].x, (int)BlPoint.Bl[ i ].y, 3, Red, TRUE );

}

//绘制引出来的线

if( Operate.State==1 ){

DrawLine( (int)Operate.fPt1.x, (int)Operate.fPt1.y, Mouse.x, Mouse.y, Red );

}

//在鼠标指针位置显示子弹

DrawCircle( Mouse.x, Mouse.y, 3, Red, TRUE );

//绘制完子弹之后，再绘制有多少当前设定了的空格

DrawLine(Mouse.x,Mouse.y,Mouse.x+Operate.Space,Mouse.y,Blue);

}

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,LPSTR lpCmdLine, int nCmdShow ){

ChangeWindowMode(TRUE);//窗口模式

SetGraphMode(640,640,32);//画面尺寸变更

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初始化与里表面化

SetMouseDispFlag( TRUE ) ;//鼠标显示有效

ini();//初始化

load();//载入

while(ProcessMessage()==0 /\*消息处理\*/&& ClearDrawScreen()==0/\*清空画面\*/ && GetHitKeyStateAll\_2(Key)==0 /\*保存输入状态\*/&& Key[KEY\_INPUT\_ESCAPE]==0/\*没有按下ESC \*/){

GetHitMouseStateAll\_2(&Mouse);

CalcMouse();//鼠标计算

CalcOperate();//操作计算

Show();//显示

ScreenFlip();

}

DxLib\_End();

return 0;

}

## 第五十二章 来制作完成汉字弹幕的工具吧（2）

这一章是51章的继续。在这一章中我们把在上一章中完成的原型（Proto Type）版给完成。

第51章的程序只是暂时地保留子弹的位置并绘制之。

现在我们要实现：

Ctrl + Z能够撤销操作（和windows标准的Ctrl+Z的功能一样）

Ctrl + Y 能够将被撤销的操作还原（和windows标准的Ctrl+Y的功能一样）

按着S键然后按左右方向键可以更改Space ※1

使用左右方向键来变更子弹的角度

C（color）键选择子弹的颜色

K（kind）键选择 子弹的种类

空格键可以切换显示

※1: 按着鼠标拖动就会拉出一条红色的线，单击鼠标的话就会自动地按一定的间隔在一条直线上放置若干个子弹。也就是调整这个间隔的的功能。

我们追加上这7个功能，以及把制作好了的数据写入文件里面的功能吧。

现在我们首先看一下完成版是怎么操作的。

感性上已经有了个大概的体会了吧。

那么，现在来看看具体的程序代码吧。

由于这一次我们使用子弹的图像，因此读入子弹的图像。这个和之前没有区别。

---在 load.cpp 中进行以下追加 ---

LoadDivGraph( "../dat/img/bullet/b0.png" , 5 , 5 , 1 , 76 , 76 , ImgBullet[0] ) ;

LoadDivGraph( "../dat/img/bullet/b1.png" , 6 , 6 , 1 , 22 , 22 , ImgBullet[1] ) ;

LoadDivGraph( "../dat/img/bullet/b2.png" , 10 , 10 , 1 , 5 , 120 , ImgBullet[2] ) ;

LoadDivGraph( "../dat/img/bullet/b3.png" , 5 , 5 , 1 , 19 , 34 , ImgBullet[3] ) ;

LoadDivGraph( "../dat/img/bullet/b4.png" , 10 , 10 , 1 , 38 , 38 , ImgBullet[4] ) ;

LoadDivGraph( "../dat/img/bullet/b5.png" , 3 , 3 , 1 , 14 , 16 , ImgBullet[5] ) ;

LoadDivGraph( "../dat/img/bullet/b6.png" , 3 , 3 , 1 , 14 , 18 , ImgBullet[6] ) ;

LoadDivGraph( "../dat/img/bullet/b7.png" , 10 , 10 , 1 , 16 , 16 , ImgBullet[7] ) ;

LoadDivGraph( "../dat/img/bullet/b8.png" , 10 , 10 , 1 , 12 , 18 , ImgBullet[8] ) ;

LoadDivGraph( "../dat/img/bullet/b9.png" , 3 , 3 , 1 , 13 , 19 , ImgBullet[9] ) ;

LoadDivGraph( "../dat/img/bullet/b10.png", 8 , 8 , 1 , 8 , 8 , ImgBullet[10]) ;

LoadDivGraph( "../dat/img/bullet/b11.png", 8 , 8 , 1 , 35 , 32 , ImgBullet[11]) ;

LoadDivGraph( "../dat/img/bullet/b12.png", 10 , 10 , 1 , 12 , 12 , ImgBullet[12]) ;

LoadDivGraph( "../dat/img/bullet/b13.png", 10 , 10 , 1 , 22 , 22 , ImgBullet[13]) ;

这次加入了实际变更的函数除了上面的函数以外，仅仅只有CalcOperate()、Show()这两个函数了。

其它的函数从上一章开始就没有变更了，因此只看这些函数。

根据键盘输入来变更操作。由于和变更的操作相关的变量是下面这个结构体，所以请事先确认。

//操作设定信息

typedef struct{

int State;//状态

int Knd;//种类

int Col;//颜色

int Space;//空格

float Angle;//角度

int flag;//是否显示flag

fPt\_t fPt1;//第1次单击的位置

fPt\_t fPt2;//第2次单击的位置

}Operate\_t;

那么，我们试着来实现将这个变量按照

Ctrl + Z能够撤销操作（和windows标准的Ctrl+Z的功能一样）

Ctrl + Y 能够将被撤销的操作还原（和windows标准的Ctrl+Y的功能一样）

按着S键然后按左右方向键可以更改Space ※1

使用左右方向键来变更子弹的角度

C（color）键选择子弹的颜色

K（kind）键选择 子弹的种类

空格键可以切换显示

的顺序变更的功能吧。

现在我们来看看要实现的CalcOpertate函数的内部吧。

void CalcOperate(){

//各个子弹的颜色数

int Col[14]={5,6,10,5,10,3,3,10,10,3,8,8,10,10};

//左控制键（左Ctrl键）被按下的时候

现在来实现功能1。

BLPoint.Num是当前显示的子弹的数量。假设放置了5个子弹的话这个变量的值就为5。

因此如果我们把这个变量改为4的话，那么表面上显示出来的也就是到第4个为止的已经登录子弹了。另外，超过0.5秒还在按下的话，我们还要让它能够一口气地连续操作。（译者注：即连续后退）

if( Key[KEY\_INPUT\_LCONTROL]>0 ){

//Z键按下的话

if( Key[KEY\_INPUT\_Z]==1 || Key[KEY\_INPUT\_Z]>30){

//操作退后一步

if( BlPoint.Num>0 ){

BlPoint.Num--;

}

}

现在实现功能2。

这个功能是把之前用Z键撤销的操作再反过来回复的功能。

我们通过减少Bl.Point.Num实现了把已经进行了的动作给撤销掉。

我们要反向回复的话，只需要增加BL.Point.Num的值就可以了 。

不过，在前面的顺序下假设我们回复到4之后，是不可能恢复到并未登陆的6的。

因此，我们应该反向回复到已经登录了的编号为止。

由于我们用x,y坐标都为0来表示当前子弹还没有登录，如果子弹的不符合这个条件的话我们就增加一次。

//Y键按下的情况

if( Key[KEY\_INPUT\_Y]==1 || Key[KEY\_INPUT\_Y]>30 ){

//按下多少次Y键回复多少次，且只能恢复到之前存在过的样子

if( BlPoint.Num<PMAX-1 && !(BlPoint.Bl[BlPoint.Num].x==0 && BlPoint.Bl[BlPoint.Num].y==0) ){

BlPoint.Num++;

}

}

}

现在来实现功能3。

我们用Operate.Space来调整子弹的空白。嘛，这个是很简单的吧。

//S键被按下的情况

if( Key[KEY\_INPUT\_S]>0 ){

//如果左方向键被按下

if( Key[KEY\_INPUT\_LEFT]==1 || Key[KEY\_INPUT\_LEFT]>30){

//增加空白

if( Operate.Space>2 ){

Operate.Space--;

}

}

//右方向键

if( Key[KEY\_INPUT\_RIGHT]==1 || Key[KEY\_INPUT\_RIGHT]>30){

//增加空白

if( Operate.Space<300 ){

Operate.Space++;

}

}

}

现在来实现功能4。

如果别的什么键都不按而按下左右方向键，就可以调整子弹的角度。

嘛，这个也不困难呢。

else{

if( Key[KEY\_INPUT\_LEFT]>0 )//左方向键

Operate.Angle-=PI2/360;//左转

if( Key[KEY\_INPUT\_RIGHT]>0 )//右方向键

Operate.Angle+=PI2/360;//右转

}

现在来实现功能5。

每次按下C键都会变更子弹的颜色。

在这里，请注意每个子弹的颜色的最大数都是不同的。

请稍微看一下「dat/img/bullet/」。

0号子弹b0.png的子弹颜色有5种。

1号子弹b1.png的子弹颜色有6种。

因此，比如如果要表示0号子弹的第6个颜色那就是不可能的了。

因此，我们在这个函数的开始定义保存颜色最大数的数组Col。

通过这个变量我们来控制颜色吧。

if( Key[KEY\_INPUT\_C]==1 )//颜色

Operate.Col = (Operate.Col+1)%Col[Operate.Knd];

嘛，剩下的工作就和之前一样了。

按下Space键的时候，将会让flag交互变换，使显示或隐藏。

用K键进行变更。变更的时候自动将颜色变为0。

if( Key[KEY\_INPUT\_SPACE]==1 )//显示flag

Operate.flag\*=-1;

if( Key[KEY\_INPUT\_K]==1 ){//种类

Operate.Knd = (++Operate.Knd)%14;

Operate.Col = 0;

}

}

之后我们只要完成显示数据内容的Show函数就OK了。

前半部分完成的内容和上一章没有区别。

只需要在Operate变量的内容中追加用于显示的部分就行了。

//显示数据

void Show(){

int i;

//背景を描画

//绘制背景

DrawGraph(0,0,ImgBack,FALSE);

//弾を描画

//绘制子弹

for(i=0; i<BlPoint.Num; i++){

DrawRotaGraphF( BlPoint.Bl[ i ].x, BlPoint.Bl[ i ].y,1.0,BlPoint.Bl[i].Angle,

ImgBullet[BlPoint.Bl[i].Knd][BlPoint.Bl[i].Col], TRUE );

}

//绘制引出来的线

if( Operate.State==1 ){

DrawLine( (int)Operate.fPt1.x, (int)Operate.fPt1.y, Mouse.x, Mouse.y, Red );

}

//鼠标指针位置绘制子弹

DrawRotaGraph( Mouse.x, Mouse.y, 1.0, Operate.Angle, ImgBullet[Operate.Knd][Operate.Col], TRUE );

//绘制完子弹之后，再绘制有多少当前设定了的空格

DrawLine(Mouse.x,Mouse.y,Mouse.x+Operate.Space,Mouse.y,Blue);

//如果显示flag为有效的话，显示当前的操作设定内容

if(Operate.flag==1){

SetDrawBlendMode( DX\_BLENDMODE\_ALPHA , 128 ) ;

DrawBox(0,0,230,120,0,TRUE);

SetDrawBlendMode( DX\_BLENDMODE\_NOBLEND, 0 ) ;

DrawFormatString(0, 0,White,"坐标[%3d,%3d]",Mouse.x,Mouse.y);

DrawFormatString(0, 20,White,"种类 [%2d] : K键",Operate.Knd);

DrawFormatString(0, 40,White,"颜色　 [%2d] : C键",Operate.Col);

DrawFormatString(0, 60,White,"角度[%5.1f°] : ←→键",Operate.Angle/PI2\*360.0f);

DrawFormatString(0, 80,White,"空白 [%3d] : S+←→键",Operate.Space);

DrawFormatString(0,100,White,"按下Space键不显示");

}

}

唉呀，我们还忘记写把我们好不容易做出来的数据给写入到文件中的函数了。

//写出数据

void Output(){

int i;

FILE \*fp;

//座標データを-1～1に変換する

//坐标数据变换到-1~1之间

for(i=0; i<BlPoint.Num; i++){

BlPoint.Bl[i].x -= WINDOW\_SIZE\_X/2;

BlPoint.Bl[i].x /= WINDOW\_SIZE\_X/2;

BlPoint.Bl[i].y -= WINDOW\_SIZE\_Y/2;

BlPoint.Bl[i].y /= WINDOW\_SIZE\_Y/2;

}

fp = fopen( "Output.dat" , "wb" );

if( fp == NULL )

return;

fwrite( &BlPoint, sizeof(BlPoint), 1, fp );

fclose(fp);

}

文件的输入输出用fopen函数就可以进行。

着并不是什么特别的函数，而是C的标准函数。

如果不知道如何使用的话，请谷歌“fopen“学习一下。

我们把结构体一次性写出到文件中。

然后读入的时候调用fread函数就能一次性读入了。

关于读入将会在下一章中说明。

在这里稍微注意一下。

为什么在写入数据之前要用for语句来处理一下呢？这是一个要点。

现在，各个坐标数据(x,y)都用0~640(WINDOW\_SIZE\_X, WINDOW\_SIZE\_Y)之间的值来表示出来了呢。（严格来说是639）

在上面的程序中(0,0) ~(640,640)都是允许的。

不过这么一来，画面的中心就是(320,320)呢。我们希望这一点能作为原点(0,0)。为什么这样子会更好我们会在下一章中实际体会到的。

再者，如果我们用-1~1的范围来展示坐标，表示出离中心有多远的话，会使子弹速度的设定之类的工作会更加方便，因此我们将

(0,0)~(640,640)的数据变换到

(-1,-1)~(1,1)去。

这个处理我们也写好了呢。

那么最后我们最后来清晰地看一下完整的代码吧。

---在 main.cpp 中进行一下部分追加---

#define WINDOW\_SIZE\_X 640

#define WINDOW\_SIZE\_Y 640

--- (中略) ---

void CalcOperate(){

//各弾の色数

//各个子弹的颜色数

int Col[14]={5,6,10,5,10,3,3,10,10,3,8,8,10,10};

//左Ctrl键被着的时候

if( Key[KEY\_INPUT\_LCONTROL]>0 ){

//Z键如果被按下

if( Key[KEY\_INPUT\_Z]==1 || Key[KEY\_INPUT\_Z]>30){

//操作往前退一步

if( BlPoint.Num>0 ){

BlPoint.Num--;

}

}

//Y键按下的情况

if( Key[KEY\_INPUT\_Y]==1 || Key[KEY\_INPUT\_Y]>30 ){

//按下多少次Y键回复多少次，且只能恢复到之前存在过的样子

if( BlPoint.Num<PMAX-1 && !(BlPoint.Bl[BlPoint.Num].x==0 && BlPoint.Bl[BlPoint.Num].y==0) ){

BlPoint.Num++;

}

}

}

}

//S键被按着时候

if( Key[KEY\_INPUT\_S]>0 ){

//左方向键如果被按下

if( Key[KEY\_INPUT\_LEFT]==1 || Key[KEY\_INPUT\_LEFT]>30){

//减少空白

if( Operate.Space>2 ){

Operate.Space--;

}

}

//右方向键

if( Key[KEY\_INPUT\_RIGHT]==1 || Key[KEY\_INPUT\_RIGHT]>30){

//增加空白

if( Operate.Space<300 ){

Operate.Space++;

}

}

}

else{

if( Key[KEY\_INPUT\_LEFT]>0 )//左方向键

Operate.Angle-=PI2/360;//左转

if( Key[KEY\_INPUT\_RIGHT]>0 )//右方向键

Operate.Angle+=PI2/360;//右转

}

if( Key[KEY\_INPUT\_C]==1 )//颜色

Operate.Col = (++Operate.Col)%Col[Operate.Knd];

if( Key[KEY\_INPUT\_SPACE]==1 )//显示flag

Operate.flag\*=-1;

if( Key[KEY\_INPUT\_K]==1 ){//种类

Operate.Knd = (++Operate.Knd)%14;

Operate.Col = 0;

}

}

void Show(){

int i;

//背景を描画

//绘制背景

DrawGraph(0,0,ImgBack,FALSE);

//弾を描画

//绘制子弹

for(i=0; i<BlPoint.Num; i++){

DrawRotaGraphF( BlPoint.Bl[ i ].x, BlPoint.Bl[ i ].y,1.0,BlPoint.Bl[i].Angle,

ImgBullet[BlPoint.Bl[i].Knd][BlPoint.Bl[i].Col], TRUE );

}

//绘制引出来的线

if( Operate.State==1 ){

DrawLine( (int)Operate.fPt1.x, (int)Operate.fPt1.y, Mouse.x, Mouse.y, Red );

}

//鼠标指针位置绘制子弹

DrawRotaGraph( Mouse.x, Mouse.y, 1.0, Operate.Angle, ImgBullet[Operate.Knd][Operate.Col], TRUE );

//绘制完子弹之后，再绘制有多少当前设定了的空格

DrawLine(Mouse.x,Mouse.y,Mouse.x+Operate.Space,Mouse.y,Blue);

//如果显示flag为有效的话，显示当前的操作设定内容

if(Operate.flag==1){

SetDrawBlendMode( DX\_BLENDMODE\_ALPHA , 128 ) ;

DrawBox(0,0,230,120,0,TRUE);

SetDrawBlendMode( DX\_BLENDMODE\_NOBLEND, 0 ) ;

DrawFormatString(0, 0,White,"坐标[%3d,%3d]",Mouse.x,Mouse.y);

DrawFormatString(0, 20,White,"种类 [%2d] : K键",Operate.Knd);

DrawFormatString(0, 40,White,"颜色　 [%2d] : C键",Operate.Col);

DrawFormatString(0, 60,White,"角度[%5.1f°] : ←→键",Operate.Angle/PI2\*360.0f);

DrawFormatString(0, 80,White,"空白 [%3d] : S+←→键",Operate.Space);

DrawFormatString(0,100,White,"按下Space键不显示");

}

}

void Output(){

int i;

FILE \*fp;

//坐标数据变换到-1~1之间

for(i=0; i<BlPoint.Num; i++){

BlPoint.Bl[i].x -= WINDOW\_SIZE\_X/2;

BlPoint.Bl[i].x /= WINDOW\_SIZE\_X/2;

BlPoint.Bl[i].y -= WINDOW\_SIZE\_Y/2;

BlPoint.Bl[i].y /= WINDOW\_SIZE\_Y/2;

}

fp = fopen( "Output.dat" , "wb" );

if( fp == NULL )

return;

fwrite( &BlPoint, sizeof(BlPoint), 1, fp );

fclose(fp);

}

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,LPSTR lpCmdLine, int nCmdShow ){

ChangeWindowMode(TRUE);//窗口模式

SetGraphMode(WINDOW\_SIZE\_X,WINDOW\_SIZE\_Y,32);//变更画面尺寸

SetWindowSizeChangeEnableFlag(TRUE);//允许调整画面的大小

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初始化与里表面化

SetMouseDispFlag( TRUE ) ;//鼠标显示有效

ini();//初始化

load();//载入

while(ProcessMessage()==0/\*消息处理\*/ && ClearDrawScreen()==0 /\*清空画面\*/&& GetHitKeyStateAll\_2(Key)==0 /\*保存输入状态\*/&& Key[KEY\_INPUT\_ESCAPE]==0/\* ESC键没有被按下\*/){

GetHitMouseStateAll\_2(&Mouse);

CalcMouse();//鼠标计算

CalcOperate();//操作计算

Show();//显示

ScreenFlip();

}

Output();//写出数据

DxLib\_End();

return 0;

}

这样一来我们就可以用子弹来描图画或者文字，描完 之后按下ESC键就可以吧子弹的坐标数据写到Output.data文件中。

在下一章中我们来使用这个数据来制作弹幕吧。

## 第五十三章 试着制作汉字弹幕吧

接下来我们把在52章中制作好的汉字弹幕数据显示出来吧。

首先，我们做一些预先准备。并且，本次我们追加了2种新的子弹。

--- 在define.h 中进行以下追加 ---

//用于文字弹幕的子弹最大数(53)

#define FONT\_BULLET\_MAX 1000

//用于文字弹幕的文字最大数(53)

#define FONT\_NUM\_MAX 10

---在 GV.h 中进行以下追加 ---

GLOBAL BlPoint\_t BlPoint[FONT\_NUM\_MAX];//汉字弹幕用变量 (53)

---在 func.h 中进行一下追加 ---

extern void boss\_shot\_bulletH000();

extern void boss\_shot\_bulletH001();

(中间省略)

extern void boss\_shot\_bulletH011();

extern void boss\_shot\_bulletH012();//(53)

extern void boss\_shot\_bulletH013();//(53)

void (\*boss\_shot\_bullet[DANMAKU\_MAX])() =

{

boss\_shot\_bulletH012,//汉字弹幕(53)

boss\_shot\_bulletH013,//汉字弹幕(53)

//中路Boss

boss\_shot\_bulletH000,//Normal

(中间胜率)

};

---在 struct.h 中进行以下追加 ---

/单个汉字子弹的信息(53)

typedef struct{

int Knd;//子弹的种类

int Col;//子弹的颜色

float Angle;//子弹的角度

float x,y;

}Bl\_t;

//汉字子弹整体信息(53)

typedef struct{

int Num;//已经登录的个数

Bl\_t Bl[ FONT\_BULLET\_MAX ];//登录的子弹信息

}BlPoint\_t;

---在 function.h 中进行以下追加 ---

//载入汉字弹幕的子弹坐标数据(53)

GLOBAL int load\_font\_dat(char name[32], BlPoint\_t \*Bp);

---在 ini.cpp 中进行以下变更 ---

boss.appear\_count[0]=50;//中路Boss出现时刻(42)(47)(53)

---在 load.cpp 的 load() 中进行以下追加 ---

LoadDivGraph( "../dat/img/bullet/b12.png", 10 , 10 , 1 , 12 , 12 , ImgBullet[12]) ;

LoadDivGraph( "../dat/img/bullet/b13.png", 10 , 10 , 1 , 22 , 22 , ImgBullet[13]) ;

接下来，我们来读入弹幕数据，并显示出来吧。

首先就是读入。既然我们的数据是用fwrite把结构体的内容一口气写出去的，因此我们就可以用fread一口气把结构体的内容都读进来。

紧接着，汉字坐标数据的保存我们用BlPoint这个数组来进行。

我们试着来做出这样的函数，我们只需要将想要用来保存汉字弹幕数据的变量以及文件名作为参数传到这个函数中，那么就可以将汉字弹幕数据保存到那个变量中。

//文字弹幕的子弹坐标数据载入(53)

int load\_font\_dat(char name[64], BlPoint\_t \*Bp){

int i;

char fname[128];

FILE \*fp;

sprintf(fname,"../dat/font/%s.dat",name);//将接收到的文件名放入路径中

fp = fopen( fname , "rb" );//使用路径以及文件名来打开文件

if( fp == NULL )

return -1;

fread( Bp, sizeof(BlPoint\_t), 1, fp );

fclose(fp);

return 0;

}

汉字弹幕数据我们存到了“dat/font/龍.dat”中。

传入的name字符串比如有“龍”这个字，为了让它变为“data/font/龍.dat”这个样子，我们使用sprintf函数向上面那样子写就行了。

接下来，只要用fread函数读入那个路所指向的文件即可。如果读入出现错误返回-1。

那么接下来试着制作弹幕吧。

这次我们试着制作基本的2个弹幕。

首先，弹幕数据是这个样子的。

这次为了让颜色鲜明我们弄得五颜六色，嘛，到现在为止我想还没有人认为我的颜色使用是很没品的吧。（诶？）

如果你能够明白接下来我们能够像这样子，同时处理许多种颜色以及许多种类的子弹的话我就很欣慰了。

使用这个数据我们试着制作下面那样的2个弹幕吧。

第1个弹幕，我们让弹幕从中心扩散进行绘制。

这个弹幕的实现，首先我们要利用在-1~1范围内表示的坐标，用离开中心有多远来决定速度。

由于最开始子弹击中在像素-1~1之间，因此可以看见它们在中心位置重叠。

然后，用离开中心的距离来设定相应的速度，以这个速度从中心向外扩张。

一般而言，我们利用勾股定理来求得离开中心的距离，也即是√(x\*x+y\*y)这个公式。

至于子弹方向的计算，我们也利用坐标是用-1~1来表示的这一点，我们只需要把这个值传入计算角度的函数atan里面就行了。

另外，当然像下面那样，子弹的颜色和种类没有必要利用到弹幕数据中的内容。

这是因为在程序内变更就行了。

在这里，有一点需要注意，如果(0,0)有坐标数据，以及这种情况下调用atan2函数的时候会出现问题，由于我们是利用离开中心的距离来决定速度的，那么既然(0,0)有子弹，那么它会永远禁止在那里，这个时候我们应该将其作为错误情况。

因此制作扩散系的弹幕数据的时候，我们不要紧紧地靠近中心来制作。

另外，靠近中心的子弹如果不进行加速的话很难往外移动，因此我们有必要对其加速。

---在 boss\_shotH.cpp 中进行以下追加 ---

//汉字弹幕(53)

void boss\_shot\_bulletH012(){

#define TM012 200

int i,k,t=boss\_shot.cnt%TM012,t2=boss\_shot.cnt;

static int num;

if(t2==0){

input\_phy\_pos(FMX/2,FMY/2, 50);//向正中间移动

num=0;//次数初始化

if(load\_font\_dat("龍", &BlPoint[0])!=0){

printfDx("文字弹幕数据读入错误\n");

}

}

if(t==50){

for(i=0;i<BlPoint[0].Num;i++){//子弹个数次循环

if((k=search\_boss\_shot())!=-1){

float sx = BlPoint[0].Bl[i].x, sy = BlPoint[0].Bl[i].y;

if(sx==0 && sy==0){

printfDx("汉字坐标数据中有(0,0)坐标数据\n");

return ;

}

int col=BlPoint[0].Bl[i].Col, knd=BlPoint[0].Bl[i].Knd;

if(num==1){

col = 0;

knd = 4;

}

boss\_shot.bullet[k].col = col;//子弹的颜色

boss\_shot.bullet[k].x = boss.x+BlPoint[0].Bl[i].x;//坐标

boss\_shot.bullet[k].y = boss.y+BlPoint[0].Bl[i].y;

boss\_shot.bullet[k].knd = knd;//子弹的种类

boss\_shot.bullet[k].angle = atan2(BlPoint[0].Bl[i].y, BlPoint[0].Bl[i].x);//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd =

sqrt(BlPoint[0].Bl[i].x\*BlPoint[0].Bl[i].x+BlPoint[0].Bl[i].y\*BlPoint[0].Bl[i].y);//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

}

for(i=0;i<BOSS\_BULLET\_MAX;i++){//加速

if(boss\_shot.bullet[i].flag>0){

if(boss\_shot.bullet[i].state==0){

boss\_shot.bullet[i].spd \*= 1.01;

}

}

}

if(t==TM012-1){

num = (num+1)%2;

}

}

在前面我们好不容易在弹幕坐标数据中完成了的Angle并没有使用。

这是由于我们规定移动时候的Angle是前进方向。

因此，在弹幕坐标数据中完成的子弹角度，会在像上面的视频中的第2个弹幕那样子的情况下使用。

将这个展示如下。

---在 boss\_shotH.cpp --- 中进行一下部分追加

//汉字弹幕(53)

void boss\_shot\_bulletH013(){

int i,n,k,t=boss\_shot.cnt,t2=boss\_shot.cnt;

if(t2==0){

input\_phy\_pos(FMX/2,FMY/2, 50);//向正中间移动

if(load\_font\_dat("龍", &BlPoint[0])!=0){

printfDx("文字弹幕数据读入错误\n");

}

}

if(t==50){

for(i=0;i<BlPoint[0].Num;i++){

if((k=search\_boss\_shot())!=-1){

float sx = BlPoint[0].Bl[i].x, sy = BlPoint[0].Bl[i].y;

if(sx==0 && sy==0){

printfDx("汉字坐标数据中有(0,0)坐标数据\n");

return ;

}

boss\_shot.bullet[k].col = BlPoint[0].Bl[i].Col;//子弹的颜色

boss\_shot.bullet[k].x = boss.x+BlPoint[0].Bl[i].x\*200;//坐标

boss\_shot.bullet[k].y = boss.y+BlPoint[0].Bl[i].y\*200;

boss\_shot.bullet[k].knd = BlPoint[0].Bl[i].Knd;//子弹的种类

boss\_shot.bullet[k].angle = BlPoint[0].Bl[i].Angle;//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = 0;//速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

}

for(i=0;i<BOSS\_BULLET\_MAX;i++){

if(boss\_shot.bullet[i].flag>0){

if(boss\_shot.bullet[i].state==0){

int cnt = boss\_shot.bullet[i].cnt;

if(60<cnt && cnt<=120){

boss\_shot.bullet[i].x += 1;

}

if(120<cnt && cnt<=240){

boss\_shot.bullet[i].x -= 1;

}

if(240<cnt && cnt<=300){

boss\_shot.bullet[i].x += 1;

}

if(350==cnt){

boss\_shot.bullet[i].spd = 1;

}

}

}

}

}

我们可以使用把spd置为0，强制其不使用spd进行移动、或者在Angle的方向上使用spd进行移动的方法。

强制移动并不只是进行横向移动，再加以缩小扩大、旋转的处理不也很有趣么。

不过，由于spd为0的移动方法并不是设想中的移动，因此并没有轨道计算然后碰撞判定计算，所以请注意有子弹运动过快的情况（即有子弹擦过去的情况）。

另外，像第2个弹幕最后那样，如果制作按照决定好了的Angle方向飞过去的子弹的话，那么我们就可以让子弹在单纯地利用计算式没法描述的飞行模式下飞行了。

接下来就是您自行发挥的时候了☆。

## 第五十四章 图像→弹幕 试着制作变换工具吧（1）

本章，我们来试着把图像变换为弹幕数据吧。

我们通过在有勾线的图像的勾线上放置子弹，来制作弹幕数据。

在这一章中使用的线画是这个。我先新建了一个叫做“AA画像”的文件夹然后把这张图片放进去。

这是拜托緑青黒羽先生绘制的图像。非常感谢。（緑青黒羽先生的站点点此进入）

这一次也是，先新建一个工程。如果觉得新建工程很麻烦的话，您可以直接使用教程附带的project中的项目，如果您不这么认为那么还是请自行新建工程。

接下来，这一次由于我们要处理图像，因此有必要先了解图像相关的知识。

有一种称为Bitmap文件(.bmp)的体积相当庞大的文件格式呢。

这是无压缩图像文件，数据被完完整整地写进文件。

因此虽然数据很庞大但是操作起来相当简单。

在DX Library中，颜色是用光的三原色的各自的亮度分量0~255来表示的。

Bitmap也是一样。每个像素都是红、蓝、绿三种颜色各自的亮度分量在0~255的范围内表示的。

也即是说，读入数据的话，

typedef struct{

unsigned char col[3];

}img\_t;

把颜色数据放入这种感觉的结构体中似乎是个不错的想法。

不过，文件有一个叫做“文件头”的东西。

“这幅图像的尺寸是○X○哦。容量是怎么样的哦。规格又是什么哦。”

这样子的信息放在了数据的开头。这个文件头部分大小为54bit，之后才是图像本身的数据。

因此，读入的时候我们扔掉54bit的数据之后再读入就行了。

关于Bitmap的内容，在这里有详细的说明。

另外，我们要决定子弹是否放置在那里，并没有必要使用0~255的亮度分量。

倒不如说，这里只需要分为“是否有颜色”2个种类即可。

这种方法称为“二值化“。为了进行二值化，我们将所有亮度分量的平均值分为128以上和128一下即可。

总之，在这一章中

1. 读入
2. 把颜色信息放入保存颜色的结构体中
3. 进行二值化
4. 绘制

那么我们来试着实现吧。

/\* Bitmap图像的宽度 \*/

#define BMP\_YOKO 400

/\* Bitmap图像的高度 \*/

#define BMP\_TATE 400

/\* Bitmap的文件头大小 \*/

#define HEAD 54

/\* 总共大小 \*/

#define TOTAL (BMP\_YOKO\*BMP\_TATE\*3+HEAD)

/\* 作成AA数据合计子弹数 \*/

#define BULLET\_MAX 4000

/\* 用于Bitmap颜色信息存储的结构体 \*/

typedef struct{

unsigned char col[3];

}img\_t;

/\* TOTAL大小的数据 \*/

unsigned char data[TOTAL];

/\* 图像的像素个数大小的颜色存储用结构体\*/

img\_t img[BMP\_TATE][BMP\_YOKO];

/\* 存储二值化后的信息的数组\*/

BYTE Pixel[BMP\_TATE][BMP\_YOKO];

总之我们先准备好这样一些变量和宏定义。

把图像设定为400x400。

TOTAL是包含文件头后的数据的比特大小。

BULLET\_MAX现在暂时不使用。

/\* 读入Bitmap并保存到data中的函数 \*/

int ReadBmp(){

char name[256]="../AA画像/aisha.bmp";

FILE \*fp;

fp = fopen( name , "rb" );

if( fp == NULL ){

printfDx( "找不到%s。",name);

return -1;

}

fread( data, TOTAL, 1, fp );

fclose(fp);

return 0;

}

这样子就读入了Bitmap图像。这真是相当简单呢。

/\* 将Bitmap的完整数据重新放入用于各个像素的颜色保存的结构体中 \*/

void ConvData(){

int x,y,c,t;

t=HEAD;

for(y=BMP\_TATE-1;y>=0;y--){

for(x=0;x<BMP\_YOKO;x++){

for(c=0;c<3;c++){

img[y][x].col[c]=data[t];

t++;

}

}

}

}

数据是像

“蓝绿红蓝绿红蓝绿红……”

这样子连续放入data中的，因此为了解析方便我们对将其重新放进img结构体中。

/\* 二值化 \*/

void Binarization(){

int x,y,c;

int sum;

for(y=0;y<BMP\_TATE;y++){

for(x=0;x<BMP\_YOKO;x++){

sum = 0;

//计算颜色的平均亮度

for(c=0;c<3;c++){

sum += img[y][x].col[c];

}

sum /= 3;

//0~255的平均亮度如果在128以上的话（亮的话）为0，不满128的话（暗的话）为1

if( sum >= 128){

Pixel[y][x] = 0;

} else {

Pixel[y][x] = 1;

}

}

}

}

这样子就行了二值化。我们用sum来计算3个颜色分量的平均亮度，用这个平均亮度师傅在128以上来进行判断。

二值化后，保存到Pixel中的将会是这样：如果那里没有颜色（亮）为0，如果有颜色（暗）为1。

接下来就只是Pixel内容的绘制了。

那么我们来看看整个程序吧。

和鼠标、键盘相关的文件和上一章一样。

#include "../../../include/DxLib.h"

#include "Key\_Mouse.h"

/\* Bitmap图像宽度 \*/

#define BMP\_YOKO 400

/\* Bitmap图像高度 \*/

#define BMP\_TATE 400

/\* Bitmap文件头大小 \*/

#define HEAD 54

/\* 总共大小 \*/

#define TOTAL (BMP\_YOKO\*BMP\_TATE\*3+HEAD)

/\* 作成AA数据的总计子弹数 \*/

#define BULLET\_MAX 4000

/\* 保存Bitmap的颜色信息用的结构体 \*/

typedef struct{

unsigned char col[3];

}img\_t;

int Key[256];

/\* TOTAL大小的数据 \*/

unsigned char data[TOTAL];

/\* 图像的像素个数大小的颜色存储用结构体\*/

img\_t img[BMP\_TATE][BMP\_YOKO];

/\* 存储二值化后的信息的数组 \*/

BYTE Pixel[BMP\_TATE][BMP\_YOKO];

/\* 读入Bitmap并保存到data中的函数 \*/

int ReadBmp(){

char name[256]="../AA画像/aisha.bmp";

FILE \*fp;

fp = fopen( name , "rb" );

if( fp == NULL ){

printfDx( "找不到%s。",name);

return -1;

}

fread( data, TOTAL, 1, fp );

fclose(fp);

return 0;

}

/\* 将Bitmap的完整数据重新放入用于各个像素的颜色保存的结构体中 \*/

void ConvData(){

int x,y,c,t;

t=HEAD;

for(y=BMP\_TATE-1;y>=0;y--){

for(x=0;x<BMP\_YOKO;x++){

for(c=0;c<3;c++){

img[y][x].col[c]=data[t];

t++;

}

}

}

}

/\* 二值化 \*/

void Binarization(){

int x,y,c;

int sum;

for(y=0;y<BMP\_TATE;y++){

for(x=0;x<BMP\_YOKO;x++){

sum = 0;

//计算颜色的平均亮度

for(c=0;c<3;c++){

sum += img[y][x].col[c];

}

sum /= 3;

//0~255的平均亮度如果在128以上的话（亮的话）为0，不满128的话（暗的话）为1

if( sum >= 128){

Pixel[y][x] = 0;

} else {

Pixel[y][x] = 1;

}

}

}

}

/\* 描画 \*/

void Graph(){

int x,y;

static int Black = GetColor(0,0,0);

DrawBox(0,0,BMP\_YOKO,BMP\_TATE,GetColor(255,255,255),TRUE);

for(y=0;y<BMP\_TATE;y++){

for(x=0;x<BMP\_YOKO;x++){

if( Pixel[y][x] == 1){

DrawPixel(x,y,Black);

}

}

}

}

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,LPSTR lpCmdLine, int nCmdShow ){

SetGraphMode(BMP\_YOKO,BMP\_TATE,16);//400x400に

SetWindowSizeChangeEnableFlag(TRUE);//允许变更画面的大小

ChangeWindowMode(TRUE);//窗口模式

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初始化和里画面化

ReadBmp();//读入Bitmap

ConvData();//保存到图像的颜色信息结构体中

Binarization();//二值化

while(ProcessMessage()==0/\*消息处理\*/ && ClearDrawScreen()==0 /\*清空画面\*/&& GetHitKeyStateAll\_2(Key)==0 /\*保存输入状态\*/&& Key[KEY\_INPUT\_ESCAPE]==0/\* ESC键没有被按下\*/){

Graph();//绘制

ScreenFlip();

}

DxLib\_End();

return 0;

}

运行结果

由于二值化后的Pixel数组的内容会作为进行DrawPixel的基础，因此应该可以理解从Biitmap图像中完整地读入数据然后变换得以进行的这一操作。

## 第五十五章 图像→弹幕 试着制作变换工具吧（2）

那么我们来完成在54章中制作的画像→弹幕的变换工具吧。

这一次我们应该追加的处理就是“放置子弹”的处理了。

以决定好了的间隔，在线上放置子弹。

将全部的像素从头至尾地遍历一次，当某个像素上有颜色的话那么再考虑是否放置子弹。

首先，计算所有当前已经登录了的子弹和当前位置位置的距离，如果所有子弹和当前位置的距离都在已经决定好了的距离以上的话，那么就可以在那里放置子弹了。

这样一来，在一定的距离以内子弹之间不会重叠，然后再在线上放置子弹吧。

另外，我们还要事先让子弹放置的间隔可以用左右键来调整。

进一步，我们也加上通过按下空格键，能在“只显示子弹”、“显示线画和子弹”、“只显示线画”这样子的显示方法之间进行变换的功能。

总之，我们来看看代码吧。

首先我们定义子弹的结构体。

//子弹的坐标(55)

typedef struct{

float x,y;

}Pt\_t;

//放置了的子弹的结构体

typedef struct{

int num;//现在已经放置了的子弹数

Pt\_t Pt[ BULLET\_MAX ];

}Bl\_t;

然后定义坐标的结构体Pt\_t以及和与集中了BULLET\_MAX个放置好了的子弹相关的结构体Bl\_t。

typedef struct{

int state; //状态

double len; //间隔

}Operate\_t;

Bl\_t Bl;

Operate\_t Operate;

这一次也定义和操作相关的结构体Operate\_t。其中它的成员有和显示方法相关的.state以及和子弹放置的距离相关的.len。

int img\_bullet[4] ;

int img\_back;

因为我们使用了图像，因此我们定义保存图像句柄用的变量。

/\* 子弹放置计算 \*/

void CalcPut(){

int i,x,y;

double lx,ly,len;

//置当前登录了的子弹个数为0个

Bl.num = 0;

//Bitmap图像大小次循环

for(y=0;y<BMP\_TATE;y++){

for(x=0;x<BMP\_YOKO;x++){

//如果子弹已经无法登录了那么跳出西循环

if( Bl.num >= BULLET\_MAX-1 ){

break;

}

//如果当前位置有颜色

if( Pixel[y][x] == 1){

//当前登录了的子弹个数次循环

for(i=0; i<Bl.num; i++){

//所有的子弹和当前位置的距离在Operate.len以上的话不跳出循环

lx = x - Bl.Pt[i].x;

ly = y - Bl.Pt[i].y;

if( lx\*lx + ly\*ly < Operate.len\*Operate.len){

break;

}

}

//没有中途放弃=没有一个子弹在附近的的话

if( i == Bl.num ){

//登录

Bl.Pt[ Bl.num ].x = x;

Bl.Pt[ Bl.num ].y = y;

Bl.num++;

}

}

}

}

}

我们从头开始检查图像。

如果找到有颜色的像素点，那么考虑是否在那里放置子弹。

计算当前位置和当前已经登录的所有子弹的距离，如果和任何子弹的距离都在规定间隔以上的话就在那个地方登录子弹。

/\* 操作计算 \*/

void CalcOperate(){

//每按下Space键就变化一次状态

if( Key[KEY\_INPUT\_SPACE] == 1 ){

Operate.state = ( Operate.state+1 )%3;

}

//如果左方向键被按下则减少.len

if( Key[KEY\_INPUT\_LEFT] == 1 || Key[KEY\_INPUT\_LEFT] > 20 ){

if( Operate.len > 1 ){

Operate.len -= 0.2;

}

CalcPut();

}

//如果右方向键被按下则增加.len

if( Key[KEY\_INPUT\_RIGHT] == 1 || Key[KEY\_INPUT\_RIGHT] > 20 ){

if( Operate.len < 50 ){

Operate.len += 0.2;

}

CalcPut();

}

}

我们这样子计算操作。每次按下Space键则变更显示状态。

另外，每按下左右方向键能够变更规定间隔。一直按下能够快速计算。

/\* 绘制 \*/

void Graph(){

int i;

//绘制模式随状态的变化而变化

if( Operate.state == 1 || Operate.state == 2 ){

DrawGraph(0,0,img\_back,FALSE);

}

if( Operate.state == 0 || Operate.state == 1 ){

for(i=0; i<Bl.num; i++){

DrawRotaGraph(Bl.Pt[i].x,Bl.Pt[i].y,1.0,0.0,img\_bullet[1],TRUE);

}

}

SetDrawBlendMode( DX\_BLENDMODE\_ALPHA , 64 ) ;

DrawBox(0,0,100,40,0,TRUE);

SetDrawBlendMode( DX\_BLENDMODE\_NOBLEND , 0 ) ;

DrawFormatString(0,0,GetColor(255,255,255),"间隔=%.1f",Operate.len);

DrawFormatString(0,20,GetColor(255,255,255),"子弹数=%d",Bl.num);

}

根据操作状态的不同，可以分为只显示线画、只显示子弹等显示状态。

我们也显示了当前已经登录的子弹数和间隔。

//写出数据

void Output(){

int i;

FILE \*fp;

fp = fopen( "Output.dat" , "wb" );

for(i=0; i<Bl.num; i++){//把坐标数据变换到-1~1之间

Bl.Pt[i].x -= BMP\_YOKO/2;

Bl.Pt[i].y -= BMP\_TATE/2;

Bl.Pt[i].x /= BMP\_YOKO/2;

Bl.Pt[i].y /= BMP\_TATE/2;

}

if( fp == NULL ){

return;

}

fwrite( &Bl, sizeof(Bl\_t), 1, fp );

fclose(fp);

}

和数据写出相关的东西和前一章相同。

那么我们来试着看一下整个源码吧。

--- main.cpp ---

#include "../../../include/DxLib.h"

#include "Key\_Mouse.h"

/\* Bitmap图像宽度 \*/

#define BMP\_YOKO 400

/\* Bitmap图像高度 \*/

#define BMP\_TATE 400

/\* Bitmap文件头大小 \*/

#define HEAD 54

/\* 总共大小 \*/

#define TOTAL (BMP\_YOKO\*BMP\_TATE\*3+HEAD)

/\* 作成AA数据的总计子弹数 \*/

#define BULLET\_MAX 4000

/\* 保存Bitmap的颜色信息用的结构体 \*/

typedef struct{

unsigned char col[3];

}img\_t;

int Key[256];

//子弹的坐标(55)

typedef struct{

float x,y;

}Pt\_t;

//放置的子弹的结构体

typedef struct{

int num;//当前放置了的子弹数

Pt\_t Pt[ BULLET\_MAX ];

}Bl\_t;

typedef struct{

int state; //状态

double len; //间隔

}Operate\_t;

Bl\_t Bl;

Operate\_t Operate;

/\* TOTAL大小的数据 \*/

unsigned char data[TOTAL];

/\* 图像的像素个数大小的颜色存储用结构体\*/

img\_t img[BMP\_TATE][BMP\_YOKO];

/\* 存储二值化后的信息的数组 \*/

BYTE Pixel[BMP\_TATE][BMP\_YOKO];

int img\_bullet[4] ; //(55)

int img\_back;

/\* 读入Bitmap并保存到data中的函数 \*/

int ReadBmp(){

char name[256]="../AA画像/aisha.bmp";

FILE \*fp;

fp = fopen( name , "rb" );

if( fp == NULL ){

printfDx( "找不到%s。",name);

return -1;

}

fread( data, TOTAL, 1, fp );

fclose(fp);

return 0;

}

/\* 将Bitmap的完整数据重新放入用于各个像素的颜色保存的结构体中 \*/

void ConvData(){

int x,y,c,t;

t=HEAD;

for(y=BMP\_TATE-1;y>=0;y--){

for(x=0;x<BMP\_YOKO;x++){

for(c=0;c<3;c++){

img[y][x].col[c]=data[t];

t++;

}

}

}

}

/\* 二值化 \*/

void Binarization(){

int x,y,c;

int sum;

for(y=0;y<BMP\_TATE;y++){

for(x=0;x<BMP\_YOKO;x++){

sum = 0;

//计算颜色的平均亮度

for(c=0;c<3;c++){

sum += img[y][x].col[c];

}

sum /= 3;

//0~255的平均亮度如果在128以上的话（亮的话）为0，不满128的话（暗的话）为1

if( sum >= 128){

Pixel[y][x] = 0;

} else {

Pixel[y][x] = 1;

}

}

}

}

/\* 子弹放置计算 \*/

void CalcPut(){

int i,x,y;

double lx,ly,len;

//置当前登录了的子弹个数为0个

Bl.num = 0;

//Bitmap图像大小次循环

for(y=0;y<BMP\_TATE;y++){

for(x=0;x<BMP\_YOKO;x++){

//如果子弹已经无法登录了那么跳出西循环

if( Bl.num >= BULLET\_MAX-1 ){

break;

}

//如果当前位置有颜色

if( Pixel[y][x] == 1){

//当前登录了的子弹个数次循环

for(i=0; i<Bl.num; i++){

//所有的子弹和当前位置的距离在Operate.len以上的话不跳出循环

lx = x - Bl.Pt[i].x;

ly = y - Bl.Pt[i].y;

if( lx\*lx + ly\*ly < Operate.len\*Operate.len){

break;

}

}

//没有中途放弃=没有一个子弹在附近的的话

if( i == Bl.num ){

//登录

Bl.Pt[ Bl.num ].x = x;

Bl.Pt[ Bl.num ].y = y;

Bl.num++;

}

}

}

}

}

/\* 操作计算 \*/

void CalcOperate(){

//每按下Space键就变化一次状态

if( Key[KEY\_INPUT\_SPACE] == 1 ){

Operate.state = ( Operate.state+1 )%3;

}

//如果左方向键被按下则减少.len

if( Key[KEY\_INPUT\_LEFT] == 1 || Key[KEY\_INPUT\_LEFT] > 20 ){

if( Operate.len > 1 ){

Operate.len -= 0.2;

}

CalcPut();

}

//如果右方向键被按下则增加.len

if( Key[KEY\_INPUT\_RIGHT] == 1 || Key[KEY\_INPUT\_RIGHT] > 20 ){

if( Operate.len < 50 ){

Operate.len += 0.2;

}

CalcPut();

}

}

/\* 绘制 \*/

void Graph(){

int i;

//绘制模式随状态的变化而变化

if( Operate.state == 1 || Operate.state == 2 ){

DrawGraph(0,0,img\_back,FALSE);

}

if( Operate.state == 0 || Operate.state == 1 ){

for(i=0; i<Bl.num; i++){

DrawRotaGraph(Bl.Pt[i].x,Bl.Pt[i].y,1.0,0.0,img\_bullet[1],TRUE);

}

}

SetDrawBlendMode( DX\_BLENDMODE\_ALPHA , 64 ) ;

DrawBox(0,0,100,40,0,TRUE);

SetDrawBlendMode( DX\_BLENDMODE\_NOBLEND , 0 ) ;

DrawFormatString(0,0,GetColor(255,255,255),"间隔=%.1f",Operate.len);

DrawFormatString(0,20,GetColor(255,255,255),"子弹数=%d",Bl.num);

}

//写出数据

void Output(){

int i;

FILE \*fp;

fp = fopen( "Output.dat" , "wb" );

for(i=0; i<Bl.num; i++){//把坐标数据变换到-1~1之间

Bl.Pt[i].x -= BMP\_YOKO/2;

Bl.Pt[i].y -= BMP\_TATE/2;

Bl.Pt[i].x /= BMP\_YOKO/2;

Bl.Pt[i].y /= BMP\_TATE/2;

}

if( fp == NULL ){

return;

}

fwrite( &Bl, sizeof(Bl\_t), 1, fp );

fclose(fp);

}

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,LPSTR lpCmdLine, int nCmdShow ){

SetGraphMode(BMP\_YOKO,BMP\_TATE,16);//400x400に

SetWindowSizeChangeEnableFlag(TRUE);//允许变更画面的大小

ChangeWindowMode(TRUE);//窗口模式

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初始化和里画面化

Operate.len = 5;//设置子弹间的间隔为5

img\_back = LoadGraph( "../AA画像/aisha.bmp" );

LoadDivGraph( "../dat/img/bullet/14.png" , 4 , 4 , 1 , 6 , 6 , img\_bullet ) ;//读入子弹(55)

ReadBmp();//读入Bitmap

ConvData();//保存到图像的颜色信息结构体中

Binarization();//二值化

CalcPut();//子弹放置计算

while(ProcessMessage()==0/\*消息处理\*/ && ClearDrawScreen()==0 /\*清空画面\*/&& GetHitKeyStateAll\_2(Key)==0 /\*保存输入状态\*/&& Key[KEY\_INPUT\_ESCAPE]==0/\* ESC键没有被按下\*/){

CalcOperate();//操作计算

Graph();//绘制

ScreenFlip();

}

Output();//写出文件

DxLib\_End();

return 0;

}

运行结果

这样一来我们就可以在线上放置子弹了。

我们使用这些子弹在下一章中制作弹幕。

## 第五十六章 试着制作图像弹幕吧

这次，对本篇的程序进行追加的部分和上一章制作汉字弹幕是同一个顺序。

虽然使用一样的变量也没问题，不过好歹为了容易理解因此我们分开使用不同的变量。

另外，在55章中做好了的爱莎的图像弹幕坐标数据放到了“data/font/”中

其次，我们在“data/img/bullet/”中追加了名为b14.png的子弹图像。

由于变量定义的工作、补充的函数的实现都在了一起，因此我省略了详细的说明，一起来看源码吧。

---在 define.h 中进行以下追加---

//文字弹幕所使用的子弹的最大数(56)

#define AA\_BULLET\_MAX 4000

//文字弹幕所使用的文字的最大数(56)

#define AA\_NUM\_MAX 10

---在 struct.h 中进行以下追加 ---

//每一个AA子弹的信息(56)

typedef struct{

float x,y;

}BlAA\_t;

//全体汉字子弹的信息(56)

typedef struct{

int Num;//已经登录的子弹数

BlAA\_t BlAA[ AA\_BULLET\_MAX ];//登录的子弹的信息

}BlAAPoint\_t;

---在 GV.h 中进行以下追加 ---

GLOBAL BlAAPoint\_t BlAAPoint[ AA\_NUM\_MAX ];//AA弹幕用变量(56)

---在 func.h 中追加红字部分---

extern void boss\_shot\_bulletH011();

extern void boss\_shot\_bulletH012();//(53)

extern void boss\_shot\_bulletH013();//(53)

extern void boss\_shot\_bulletH014();//(56)

void (\*boss\_shot\_bullet[DANMAKU\_MAX])() =

{

boss\_shot\_bulletH014,//汉字弹幕(56)

boss\_shot\_bulletH012,//汉字弹幕(53)

boss\_shot\_bulletH013,//汉字弹幕(53)

---在 function.h 中进行以下追加 ---

//读入AA弹幕的子弹坐标数据 (56)

GLOBAL int load\_AA\_dat(char name[64], BlAAPoint\_t \*Bp);

---在 load.cpp 的 load() 中进行一下追加---

LoadDivGraph( "../dat/img/bullet/b14.png", 4 , 4 , 1 , 6 , 6 , img\_bullet[14]) ;//(56)

---在 load.cpp 中进行以下追加---

//读入文字弹幕的子弹坐标数据(53)

int load\_AA\_dat(char name[64], BlAAPoint\_t \*Bp){

int i;

char fname[128];

FILE \*fp;

sprintf(fname,"../dat/font/%s.dat",name);//将获取都的文件名放入路径中

fp = fopen( fname , "rb" );//使用路径和文件名打开文件

if( fp == NULL )

return -1;

fread( Bp, sizeof(BlAAPoint\_t), 1, fp );

fclose(fp);

return 0;

}

---在 boss\_shotH.cpp 中进行以下追加---

//汉字弹幕(56)

void boss\_shot\_bulletH014(){

int i,k,t=boss\_shot.cnt%400,t2=boss\_shot.cnt;

if( t2==0 ){

input\_phy\_pos(FMX/2,FMY/2, 50);//向正中间移动

if(load\_AA\_dat("アイシャAA", &BlAAPoint[0])!=0){

printfDx("AA弹幕数据读入错误\n");

}

}

if( t==50 ){

for(i=0; i<BlAAPoint[0].Num; i++){

if((k=search\_boss\_shot())!=-1){

float sx = BlAAPoint[0].BlAA[i].x, sy = BlAAPoint[0].BlAA[i].y;

if(sx==0 && sy==0){

printfDx("坐标数据中有 (0,0)坐标数据\n");

return ;

}

boss\_shot.bullet[k].col = 1;//子弹的颜色

boss\_shot.bullet[k].x = boss.x+sx;//坐标

boss\_shot.bullet[k].y = boss.y+sy;

boss\_shot.bullet[k].knd = 14;//子弹的种类

boss\_shot.bullet[k].angle = atan2( sy, sx );//角度

boss\_shot.bullet[k].flag = 1;

boss\_shot.bullet[k].cnt = 0;

boss\_shot.bullet[k].spd = sqrt( sx\*sx + sy\*sy ); //速度

boss\_shot.bullet[k].eff = 0;

boss\_shot.bullet[k].state = 0;

}

}

se\_flag[0]=1;

}

for(i=0;i<BOSS\_BULLET\_MAX;i++){

if(boss\_shot.bullet[i].flag>0){

int cnt = boss\_shot.bullet[i].cnt;

boss\_shot.bullet[i].spd \*= 1.001;

if(cnt > 180){

boss\_shot.bullet[i].spd \*= 1.01;

}

}

}

}

运行结果

由于不知道为什么没法录制动画，动画我会在日后加上。（译者注：时至今日原作者仍然没有加上动画）

## 第五十七章 试着制作3D背景吧（1）

~~DX Library是2D专用的库。~~

~~然而也有唯一一个能够使用3D的函数。~~

（译者注：DX Library目前已经完全支持3D，以上两句话已经被原作者撤销掉。）

（→DX Library从ver3开始，已经完全对应3D了。现在我们制作3D程序已经可以不用使用这么复杂的手段了。）

这就是DrawPolygon3D这个函数。（关于这个函数的使用，请尽量先通过链接阅读一下DX Library参考手册（Reference））。

使用这个函数，用DX Library也来模拟出3D背景吧！——这就是本章的主题。

なお、今回も新しいプロジェクトを使用します。自分で準備する方は準備したプロジェクトに、57章の「mydat/img」にある画像を持ってきてください。

另外，这一次我们也使用新的工程。如果您是自行准备工程的话，请把已经57章的“mydat/img”下的图像给拷到已经准备好的工程中。

那么，现在我就想进入3D相关的解说了。

首先，在此之前我们一直在使用的矩形图像已经不能继续像以前那样使用了。我们没有“四角形”的说法。

在此之前我们使用的这样子的形状，我们使用“2个三角形”来表示。

如果我们用2个三角形来表示四角形的话，那么任何形状的四角形我们都能表示出来了。

我们试着分开绘制三角形abc和三角形def吧。

那么，坐标要怎么办呢。

这里需要大家注意的是“3D情况下坐标原点在左下角”这一点。迄今为止我们都认为往下是增加y的值。那是因为之前原点在左上角呢。

由于在3D下左下角为原点，因此请注意y的增加是往上走的。

因此，比如中心点为(u,v)的话，a的坐标的就变为(u-x/2, v+y/2)了。

另外，还请注意图像的尺寸只能使用2的幂次大小，。

只能处理2,4,8,16,32,64,256,512...这样子尺寸的图像。※1

如果想要使用90x120这样子的图像的话，必须要把90x120的图像放入128x128的图像中，然后“只使用图像中90x120的部分”。

另外，像(128x256)、(512x32)这样，只要是2的幂次，横向和纵向的尺寸不一样也无所谓。

那么我们具体地进行程序讲解吧。

在DX Library使用的每一个3D坐标我们像下面那样通过结构体管理。

struct VECTOR

{

　　float x, y, z ;

} ;

struct VERTEX\_3D

{

　　VECTOR pos ;

　　unsigned char b, g, r, a ;

　　float u, v ;

} ;

可能会考虑到对于每一个坐标会有相当多的变量。

必要的信息除了(x,y,z)的坐标数据以外，还有r,g,b这样子的颜色信息、a这样子的透明度。

以及在※1中说到的表示使用图像的哪个部分的u,v。

对一个多边形而言，它是需要相当多的变量来表示的。

不过为每一个坐标指定颜色和透明度这样子的分量，似乎就可以做到在图像中表现出彩虹色、或者表现出仅在图像的边角逐渐渗透，这样子精致漂亮的效果。

在这里可能有人不是很明白为什么我们要再加上u,v这个分量。

通过设置u,v的值在0.0f~1.0f之间，我们可以指定出使用图像的哪个部分。u相当于x，v相当于y。

比如用前面的例子来说明，如果图像为128x128，我们想要使用的部分为90x120的话

我们将a设置为(u,v)=(0.0f,0.0f)，

把b设置为(u,v)=(90.0f/128.0f,0)。如果在横向128像素中我们只想使用到90为止的话，那么就像90/128这样子指定，不管哪个都要指定0~1中的值。

这样一来，比如

d就是i(u,v)=(90.0f/128.0f,120.0f/128.0)。

要为每一个坐标都加上这样的信息稍微有点麻烦呢。

使用前面的VERTEX\_3D在Vertex[6]中保存6个点的时候，要各个点要如何保存这些信息才好呢？

([0]为a、[1]为b....、[5]为f)

现在，为了描绘中心点为(320,240)、宽度为100、高度为100、深度为Z的图像，

在对a的位置代入必要信息的时候，我们像下面那样为之代入。

// 对画面中央以100的宽高进行绘制

Vertex[0].pos.x = 320.0F - 50.0F ; Vertex[0].pos.y = 240.0F + 50.0F ; Vertex[0].pos.z = Z ;

Vertex[0].u = 0.0F ; Vertex[0].v = 0.0F ;

在此之上，我们试着看看放在DX Library的参考文档中的范例吧（我修正了一部分）。

在这个范例中，我们使用上面所说的内容，让2个三角形像是促使它们拼成四角形一般彼此靠近着绘制，而在Z轴上时而远离时而靠近，这样子使画像动起来。

※ 此处的源代码借用了DX Library原站中的参考手册。

--- main.cpp ---

#include "../../../include/DxLib.h"

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,

LPSTR lpCmdLine, int nCmdShow )

{

int GrHandle ;

float Z, ZAdd ;

VERTEX\_3D Vertex[6] ; //因为有2个多边形，因此顶点为6个

ChangeWindowMode(TRUE);

//DX Library初始化处理

if( DxLib\_Init() == -1 )

return -1 ; // 如果发生错误直接终止

// 将绘制目标变为里画面

SetDrawScreen( DX\_SCREEN\_BACK ) ;

// 读入纹理

GrHandle = LoadGraph( "mydat/img/kabe.png" ) ;

// 初始化z值

Z = 0.0F ;

// 初始化z值的附加值（最开始为靠近）

ZAdd = -1.0F ;

// 循环直至某个键被按下

while( CheckHitKeyAll() == 0 )

{

// 消息处理

if( ProcessMessage() != 0 ) break ;

// 画面初始化

ClearDrawScreen() ;

// z值的处理

Z += ZAdd ;

// 当超过规定边界的时候翻转前进方向

if( Z < -300.0F || Z > 300.0F ) ZAdd = -ZAdd ;

// 设置顶点信息

{

// 对画面中央以100宽高进行绘制

Vertex[0].pos.x = 320.0F - 50.0F ; Vertex[0].pos.y = 240.0F + 50.0F ; Vertex[0].pos.z = Z ;

Vertex[0].u = 0.0F ;

Vertex[0].v = 0.0F ;

Vertex[1].pos.x = 320.0F + 50.0F ; Vertex[1].pos.y = 240.0F + 50.0F ; Vertex[1].pos.z = Z ;

Vertex[1].u = 1.0F ;

Vertex[1].v = 0.0F ;

Vertex[2].pos.x = 320.0F - 50.0F ; Vertex[2].pos.y = 240.0F - 50.0F ; Vertex[2].pos.z = Z ;

Vertex[2].u = 0.0F ;

Vertex[2].v = 1.0F ;

Vertex[3].pos.x = 320.0F + 50.0F ; Vertex[3].pos.y = 240.0F - 50.0F ; Vertex[3].pos.z = Z ;

Vertex[3].u = 1.0F ;

Vertex[3].v = 1.0F ;

Vertex[4].pos.x = 320.0F - 50.0F ; Vertex[4].pos.y = 240.0F - 50.0F ; Vertex[4].pos.z = Z ;

Vertex[4].u = 0.0F ;

Vertex[4].v = 1.0F ;

Vertex[5].pos.x = 320.0F + 50.0F ; Vertex[5].pos.y = 240.0F + 50.0F ; Vertex[5].pos.z = Z ;

Vertex[5].u = 1.0F ;

Vertex[5].v = 0.0F ;

// 亮度分量全部为100%

Vertex[0].r = Vertex[0].g = Vertex[0].b = 255 ;

Vertex[1].r = Vertex[1].g = Vertex[1].b = 255 ;

Vertex[2].r = Vertex[2].g = Vertex[2].b = 255 ;

Vertex[3].r = Vertex[3].g = Vertex[3].b = 255 ;

Vertex[4].r = Vertex[4].g = Vertex[4].b = 255 ;

Vertex[5].r = Vertex[5].g = Vertex[5].b = 255 ;

// a也为最大值

Vertex[0].a = 255 ;

Vertex[1].a = 255 ;

Vertex[2].a = 255 ;

Vertex[3].a = 255 ;

Vertex[4].a = 255 ;

Vertex[5].a = 255 ;

}

// 不使用透明度绘制两个多边形

DrawPolygon3D( Vertex, 2, GrHandle, TRUE ) ;

// 将里画面的内容反映到表画面中

ScreenFlip() ;

}

// DX Library使用终止处理

DxLib\_End() ;

// 结束程序

return 0 ;

}

运行结果

如果您不太明白，请认真琢磨这个范例，玩味这个范例程序直到各方面都领会为止。

## 第五十八章 试着制作3D背景吧（2）

在上一章中，我们仅仅是绘制一张图像都相当麻烦呢。

然而，有很多事是明摆着要去做的，正因为“写了很多难以理解的东西”，似乎才能够使程序效率化。

在这一章，我们的目的是努力使3D绘制轻松地进行，以及图像的淡入淡出功能（fade-in, fade-out）的附加。

所谓淡入，就是从透明状态渐渐地变得清晰起来的效果，而淡出与之相反。

遠くにある物体が近づいてくるとき、どこかの地点から急にはっきり見えたりしませんよね。

远处的物体靠近的时候，从某一个地方看过去，一下子是不会看得很清楚呢。

因此，我们添加上可以慢慢地清楚看见的淡入，以及与之相反功能的淡出吧。

首先，我们来看一下在这一章中完成的程序的运行结果。

运行结果

由于可能您还有点不明白我们现在正在做什么，因此也请同时看看59章的运行结果动画。

在画面的左侧有一个墙壁呢。让这个墙壁一会儿跑到里面一会儿跑到外面的效果就是58章的运行结果动画。

在普通DX Library下使用α混合等来使图像透明的时候，整个图像都会有一定的透明度。

然而，就像从运行结果中看到的那样，即便是在一张图像中，我们也能够让透明度产生变化。

正如同在前面的章节中也说过的那样，这是因为我们能够对每一个坐标都设定透明度。

通过这样子，“越远的地方越透明”这种事情就是可以办到的了。

请稍微看一下这个图像。

现在，我们假定墙壁从远处朝着自己靠近。

为了设定淡入淡出，我们把这些数据设定到变量中：从哪里开始开始绘制图像、淡入到哪里为止、从哪里开始淡出、绘制到哪里为止。

我们把开始绘制图像的z地点设定为FromZ。

把图像淡入终止的z地点设定为FadeFromZ。

把图像淡出开始的z地点设定为FadeToZ。

把图像绘制终止的z地点设定为ToZ。

每一张图像的大小放在LargeX和LargeY中，中心坐标设为(x,y,z)。

把这些东西整理一下我们放在一个结构体中。

//与每一个纹理相关的结构体

typedef struct{

int Type; //0：于画面平行、1：于画面垂直（墙壁）

int Img; //图像

float x,y,z; //中心点

float LargeX,LargeY; //横纵的大小（Type为1的时候，LargeX扮演LargeZ的职能

float u,v; //使用正在使用的图像的哪个部分

float FromZ,ToZ; //设定z深度移动为从哪里开始到哪里为止

float FadeFromZ,FadeToZ; //是否设定从哪儿到哪儿进行淡入淡出（消失的瞬间淡出，显示的瞬间淡入）

VERTEX\_3D Vertex[6] ; //绘制用的6个顶点

}Object\_t;

结构体中我们还另外加入了Type这个变量。

在3D的情况下绘制ID时候，许多时候会有这样一些情况：

・平行于画面

・垂直于画面(墙壁)

・垂直于画面(地面)

当然，台阶、坡道等也能倾斜着绘制来但我们暂时不管；总之，一旦我们设置0或者1中的某个值的话，我们就让它能够自动地按照平行或者垂直于画面来计算。在这一章中我们只制作和画面垂直的墙壁。

另外，在前面的章节中，我们写过这样子的程序呢。

// 向画面中央以宽高100进行绘制

Vertex[0].pos.x = 320.0F - 50.0F ; Vertex[0].pos.y = 240.0F + 50.0F ; Vertex[0].pos.z = Z ;

Vertex[0].u = 0.0F ;

Vertex[0].v = 0.0F ;

Vertex[1].pos.x = 320.0F + 50.0F ; Vertex[1].pos.y = 240.0F + 50.0F ; Vertex[1].pos.z = Z ;

Vertex[1].u = 1.0F ;

Vertex[1].v = 0.0F ;

Vertex[2].pos.x = 320.0F - 50.0F ; Vertex[2].pos.y = 240.0F - 50.0F ; Vertex[2].pos.z = Z ;

Vertex[2].u = 0.0F ;

Vertex[2].v = 1.0F ;

Vertex[3].pos.x = 320.0F + 50.0F ; Vertex[3].pos.y = 240.0F - 50.0F ; Vertex[3].pos.z = Z ;

Vertex[3].u = 1.0F ;

Vertex[3].v = 1.0F ;

Vertex[4].pos.x = 320.0F - 50.0F ; Vertex[4].pos.y = 240.0F - 50.0F ; Vertex[4].pos.z = Z ;

Vertex[4].u = 0.0F ;

Vertex[4].v = 1.0F ;

Vertex[5].pos.x = 320.0F + 50.0F ; Vertex[5].pos.y = 240.0F + 50.0F ; Vertex[5].pos.z = Z ;

Vertex[5].u = 1.0F ;

Vertex[5].v = 0.0F ;

看起来似乎是很难懂的程序，总之这只是按照顺序对abc,def进行坐标指定而已。

仔细看的话，比如就[0]而言，

x = 中心 - ○, y = 中心 + ○, u = 0, v = 0

就是这样的，也就是说，

x = 中心 + ○\*a, y = 中心 + ○\*b, u = ■+□\*c, v = ▲+△\*d

预先这样子设定的话

(a,b,c,d) = (-1,1,0,0)

就能这样子设定呢。我们把这个替换进所有的坐标中的话，

typedef struct{

float x,y;

float u,v;

}VtPm\_t;

VtPm\_t VtPm[6]={{-1,1,0,0},{1,1,1,0},{-1,-1,0,1},{1,-1,1,1},{-1,-1,0,1},{1,1,1,0}};

这样一来， 我们就能在循环中计算出来了。

……诶？不明白什么意思？

仔细研究上面的程序的话，就会发现仅仅是对同一个数字的加减而已。

而关于u,v就是哪里设置为0哪里设置为1的问题而已。

把这个规则用于变量，将数据代入变量的话就能够通过循环计算出来了，也就这么一会事。

由于并没有必要对此进行反复鼓捣理解，对此如果懵懵懂懂的话大可扔到一边就OK了。（真是马虎随便）

那么，我们把目光放到实际的实例上来吧。

Object\_t Object;

我们声明前面的Object\_t类型的结构体。

然后是结构体的初始化。

void ini (){

int i;

Object.Img = LoadGraph( "mydat/img/kabe.png" );

Object.LargeX = 48.0f;//总之对绘制的大小进行适当地设定。横纵比和素材一样

Object.LargeY = 60.0f;

Object.Type = 1;// type设置为垂直

Object.x = 220.0f;//总之将绘制的中心位置设定为中心偏左。

Object.y = 240.0f;

Object.z = 0.0f;

Object.u = 0.763671875f;//使用画面的哪个部分

Object.v = 1.0f;

Object.FromZ = 200;//绘制开始位置

Object.FadeFromZ = 100;//绘制淡入开始位置

Object.FadeToZ = -100;//绘制淡出开始位置

Object.ToZ = -200;//绘制结束位置

for(i=0; i<6; i++){

Object.Vertex[i].r = 255;

Object.Vertex[i].b = 255;

Object.Vertex[i].g = 255;

Object.Vertex[i].a = 255;

Object.Vertex[i].u = Object.u \* VtPm[i].u;

Object.Vertex[i].v = Object.v \* VtPm[i].v;

}

}

我想没有必要特别地说明。

LargeX,LargeY是绘制的大小，我们先对其进行适当地设定，之后可以一边观察运行结果一边进行调整。

不过，如果不设定为与图像的横纵比相同的比例的话显示出来会很奇怪，这一点请注意。

我们往.u放入的意义不明的数值我想大家看了素材大概就明白了。

在横向512像素下，如果512设定1的话，那么使用的实际宽度就是0.7636…f了。

至于.FromZ~.ToZ，它们和上面说明的一样呢。我们用它们来设定绘制的范围和淡入淡出的范围。

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,

LPSTR lpCmdLine, int nCmdShow )

{

int i;

float z=0;

ChangeWindowMode(TRUE);//窗口模式

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初始化和里表面化

ini();

// 循环直至某个键被按下

while(ProcessMessage()==0 && ClearDrawScreen()==0 && CheckHitKey(KEY\_INPUT\_ESCAPE)==0){

Object.z=z;

if(CheckHitKey(KEY\_INPUT\_Z)>0){

z+=1.4f;

}

if(CheckHitKey(KEY\_INPUT\_Y)>0){

z-=1.4f;

}

我们根据绘制对象的类型（type）来变换计算。

如果是与画面平行的情况，就这样子代入大小就可以了，而如果与画面垂直（.Type==1）的情况，X的大小则用于深度。

switch(Object.Type){

case 0:

for(i=0;i<6;i++){

Object.Vertex[i].pos.x = Object.x + Object.LargeX \* VtPm[i].x ;

Object.Vertex[i].pos.y = Object.y + Object.LargeY \* VtPm[i].y ;

Object.Vertex[i].pos.z = Object.z ;

}

break;

case 1:

for(i=0;i<6;i++){

Object.Vertex[i].pos.x = Object.x;

Object.Vertex[i].pos.y = Object.y + Object.LargeY \* VtPm[i].y ;

Object.Vertex[i].pos.z = Object.z + Object.LargeX \* VtPm[i].x ;

}

break;

}

在这里我们计算淡入淡出。

.FromZ~.FadeFromZ之间为淡入。

.FadeToZ~.ToZ之间为淡出。

/\*

z

Object.FromZ 200

z

Object.FadeFromZ 100

z

Object.FadeToZ -100

z

Object.ToZ -200

z

\*/

if( Object.FromZ - Object.FadeFromZ <= 0 ){

printfDx(".From的设定有问题\n");

}

else if( Object.FadeToZ - Object.ToZ <= 0 ){

printfDx(".To的设定有问题\n");

}

else{

for(i=0; i<6; i++){

float z = Object.Vertex[i].pos.z;

//如果当前位置比绘制范围还要远的话设置透明度为0

if (z < Object.ToZ){

Object.Vertex[i].a = 0;

}

//（正在靠近的时候）如果在淡入的位置的话

else if(Object.ToZ < z && z <=Object.FadeToZ){

Object.Vertex[i].a = (unsigned char)(255.0f / (Object.FadeToZ - Object.ToZ) \* (z - Object.ToZ)) ;

}

//如果在通常绘制位置的话

else if(Object.FadeToZ <= z && z <= Object.FadeFromZ){

Object.Vertex[i].a = 255;

}

//（靠近的时候）如果在淡入位置的话

else if(Object.FadeFromZ <= z && z < Object.FromZ){

Object.Vertex[i].a = (unsigned char)(255.0f / (Object.FromZ - Object.FadeFromZ) \* (Object.FromZ - z)) ;

}

//如果当前位置比绘制范围还要近的话透明度为0

else if(Object.FromZ < z){

Object.Vertex[i].a = 0;

}

}

}

描画多边形然后结束。

// 不使用透明度绘制2个多边形

DrawPolygon3D( Object.Vertex, 2, Object.Img, TRUE ) ;

DrawFormatString(0,0,GetColor(255,255,255),"%f",z);

// 将里画面的内容反映到表画面上

ScreenFlip() ;

}

// DX Library使用终止处理

DxLib\_End() ;

// 程序结束

return 0 ;

}

那么，我们试着看看整个程序吧。

在运行结果中，试着通过“Y键”“Z键”来让物体能够远离或者靠近。

--- main.cpp ---

#include "../../../include/DxLib.h"

//用于用2个三角形多边形来绘制四边形而设定的数值。因为数值是固定的，因此没有必要记住。

typedef struct{

float x,y;

float u,v;

}VtPm\_t;

VtPm\_t VtPm[6]={{-1,1,0,0},{1,1,1,0},{-1,-1,0,1},{1,-1,1,1},{-1,-1,0,1},{1,1,1,0}};

//与每一个纹理相关的结构体

typedef struct{

int Type; //0：于画面平行、1：于画面垂直（墙壁）

int Img; //图像

float x,y,z; //中心点

float LargeX,LargeY; //横纵的大小（Type为1的时候，LargeX扮演LargeZ的职能

float u,v; //使用正在使用的图像的哪个部分

float FromZ,ToZ; //设定z深度移动为从那里开始到哪里为止

float FadeFromZ,FadeToZ; //是否设定从哪儿到哪儿进行淡入淡出（消失的瞬间淡出，显示的瞬间淡入）

VERTEX\_3D Vertex[6] ; //绘制用的6个顶点

}Object\_t;

Object\_t Object;

void ini (){

int i;

Object.Img = LoadGraph( "mydat/img/kabe.png" );

Object.LargeX = 48.0f;//总之对绘制的大小进行适当地设定。横纵比和素材一样

Object.LargeY = 60.0f;

Object.Type = 1;//type设置为垂直

Object.x = 220.0f;//总之将绘制的中心位置设定为中心偏左。

Object.y = 240.0f;

Object.z = 0.0f;

Object.u = 0.763671875f;//使用画面的哪个部分

Object.v = 1.0f;

Object.FromZ = 200;//绘制开始位置

Object.FadeFromZ = 100;//绘制淡入开始位置

Object.FadeToZ = -100;//绘制淡出开始位置

Object.ToZ = -200;//绘制结束位置

for(i=0; i<6; i++){

Object.Vertex[i].r = 255;

Object.Vertex[i].b = 255;

Object.Vertex[i].g = 255;

Object.Vertex[i].a = 255;

Object.Vertex[i].u = Object.u \* VtPm[i].u;

Object.Vertex[i].v = Object.v \* VtPm[i].v;

}

}

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,

LPSTR lpCmdLine, int nCmdShow )

{

int i;

float z=0;

ChangeWindowMode(TRUE);//窗口模式

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初始化与里画面化

ini();

// 循环直至某个键被按下

while(ProcessMessage()==0 && ClearDrawScreen()==0 && CheckHitKey(KEY\_INPUT\_ESCAPE)==0){

Object.z=z;

if(CheckHitKey(KEY\_INPUT\_Z)>0){

z+=1.4f;

}

if(CheckHitKey(KEY\_INPUT\_Y)>0){

z-=1.4f;

}

switch(Object.Type){

case 0:

for(i=0;i<6;i++){

Object.Vertex[i].pos.x = Object.x + Object.LargeX \* VtPm[i].x ;

Object.Vertex[i].pos.y = Object.y + Object.LargeY \* VtPm[i].y ;

Object.Vertex[i].pos.z = Object.z ;

}

break;

case 1:

for(i=0;i<6;i++){

Object.Vertex[i].pos.x = Object.x;

Object.Vertex[i].pos.y = Object.y + Object.LargeY \* VtPm[i].y ;

Object.Vertex[i].pos.z = Object.z + Object.LargeX \* VtPm[i].x ;

}

break;

}

/\*

z

Object.FromZ 200

z

Object.FadeFromZ 100

z

Object.FadeToZ -100

z

Object.ToZ -200

z

\*/

if( Object.FromZ - Object.FadeFromZ <= 0 ){

printfDx(".From的设定有问题\n");

}

else if( Object.FadeToZ - Object.ToZ <= 0 ){

printfDx(".To的设定有问题\n");

}

else{

for(i=0; i<6; i++){

float z = Object.Vertex[i].pos.z;

//如果当前位置比绘制范围还要远的话设置透明度为0

if (z < Object.ToZ){

Object.Vertex[i].a = 0;

}

//（正在靠近的时候）如果在淡入的位置的话

else if(Object.ToZ < z && z <=Object.FadeToZ){

Object.Vertex[i].a = (unsigned char)(255.0f / (Object.FadeToZ - Object.ToZ) \* (z - Object.ToZ)) ;

}

//如果在通常绘制位置的话

else if(Object.FadeToZ <= z && z <= Object.FadeFromZ){

Object.Vertex[i].a = 255;

}

//（靠近的时候）如果在淡入位置的话

else if(Object.FadeFromZ <= z && z < Object.FromZ){

Object.Vertex[i].a = (unsigned char)(255.0f / (Object.FromZ - Object.FadeFromZ) \* (Object.FromZ - z)) ;

}

//如果当前位置比绘制范围还要近的话透明度为0

else if(Object.FromZ < z){

Object.Vertex[i].a = 0;

}

}

}

// 不使用透明度绘制2个多边形

DrawPolygon3D( Object.Vertex, 2, Object.Img, TRUE ) ;

DrawFormatString(0,0,GetColor(255,255,255),"%f",z);

// 将里画面的内容反映到表画面上

ScreenFlip() ;

}

// DX Library使用终止处理

DxLib\_End() ;

// 程序结束

return 0 ;

}

## 第五十九章 试着制作3D背景吧（3）

虽然之前大家已经看过了，不过首先我们还是看一下运行结果。

运行结果

可以看到墙壁连续地在滚动（Scroll）呢。

然而，我们要对这个墙壁一个个地登录、独立起来计算、进行淡入淡出计算，看起来实在是麻烦啊。

因此，

通过进行

“把这个图像像墙壁一样给我表示在这里！”

“把这个图像作为地面给我表示在这这里！”

这样子的初始化方式，在恰到好处的间隔下一个一个地设定多边形，而登录计算都自动地进行的话就相当方便了。

在这一章中，我们来试着考虑只需要写一行初始化函数，墙壁和的地面要多少都被追加多少的程序吧。

一开始虽然很麻烦，不过从后面开始就很轻松了。

首先，由于我们要把许多多边形都统一管理，因此我们有必要定义一个进一步来统括前面的多边形信息的结构体的结构体。

统括许多多边形的结构体我们编写为Object\_t。

而每一个多边形作为Object\_t的孩子，在这个意义下我们将其编写为ObChild\_t。

结构体和之前的结构体比起来并没有什么太大的变化，它像下面那样：

//每一个纹理相关的结构体

typedef struct{

float x,y,z;//中心点

VERTEX\_3D Vertex[6] ; //绘制用的6个顶点

} ObChild\_t;

//大量的纹理集中在一起的信息

//比如在左边不断地连续显示的许多墙壁将在这里管理许多ObChild\_t。

//Obchild\_t的集合

typedef struct{

int Type; // 0：平行于画面、1：垂直于画面

int Img; //图像

int ImgSize;

int ImgX1,ImgX2,ImgY1,ImgY2;

float LargeX,LargeY;//横纵的大小（Type为1的时候LargeX扮演LargeZ的职能）

float Zhaba; //深度的幅度

float FromZ,ToZ; //是否设定从哪儿到哪儿z深度上的移动

float FadeFromZ,FadeToZ; //是否设定从哪儿到哪儿进行淡入淡出（消失的瞬间淡出，显示的瞬间淡入）

int ObchindMax;

ObChild\_t ObChild[ OBCHILD\_MAX ];

} Object\_t;

这里新出现一个Zhaba的变量。它表示“深度Z的幅度”。（译者注：这里的幅度可以理解为范围）

要说这是什么的话，

像这样，由于要把多张纹理连续地贴上去，如果在执行“在FromZ~ToZ之间，给我显示五张纹理”这样子的命令的时候，可以自动地计算纹理结构中的Zhaba然后计算出纹理的深度的话会很方便，因此我们增加了这个变量。

另外，如果说“在FromZ~ToZ之间，给我显示5张纹理”的话，那么同时表示的纹理需要6张，这一点请注意。

要说为什么的话，这是因为前进方向的反方向端假设没有纹理的话，如果考虑只有正当中的5张纹理向自己靠近过来，那么就会发现“FromZ”的位置就没有用于绘制的纹理了。

为此，像下面的图像中一样，后面还需要一张纹理。

图像的张数我们保存在ObchildMax这个变量中。

另外，还有一个重要的地方。那就是“必须要先绘制深处的东西”这个事情。

要是我们先绘制靠近外面的东西，之后 再绘制靠近立面的东西的话，显示出来就会很奇怪。

因此，一般而言我们需要用Z来进行排序。

由于在范例中只在同一个对象里面排序，因此如果不同的对象之间出现重叠的时候有必要对全体进行排序这一点请您牢记。

我们注意着这一点然后来看看程序吧。

首先是结构体的初始化。由于初始化所需要的信息变得相当地多了，因此看起来会很困难，大家努力地看吧。（汗）

/\*

int ImgHandle : 图像句柄

int ImgSize : 图像尺寸

int ImgX1 : 图像使用部分的左上角坐标

int ImgY1 : 图像使用部分的左上角坐标

int ImgX2 : 图像使用部分的右下角坐标

int ImgY2 : 图像使用部分的右下角坐标

float LargeX : 绘制的大小（横向）

float LargeY : 绘制的大小（纵向）

int Type : 绘制的种类 0：与画面一个方向 1：与画面垂直（墙壁） 2：与画面垂直（地面）

float FromZ : 绘制开始的深度

float FadeFromZ : 淡入开始的深度

float FadeToZ : 淡出开始的深度

float ToZ : 绘制种子的深度

float GraphX : 绘制的中心点

float GraphY : 绘制的中心点

int ObchildMax : 只在type为0的时候，决定同时显示多少个

\*/

初始化所需要的信息就有这么多。IniObj函数的参数除了一开始的\*Ob意外，其他的都按照上面的顺序来接受，请参考一下。

void IniObj(Object\_t \*Ob, int ImgHandle, int ImgSize, int ImgX1, int ImgY1, int ImgX2, int ImgY2, float LargeX, float LargeY,

int Type, float FromZ, float FadeFromZ, float FadeToZ, float ToZ, float GraphX, float GraphY, int ObchildMax){

int i,s;

统括大量纹理的结构体Object在范例中最大可以登录3个。

如果您有余力的话，可以试着对其进行动态登录。（译者注：这里可以用malloc来进行动态处理，如果是C++的话就更方便了）

if( ObjectNum >= OBJECT\_NUM\_MAX-1 ){

printfDx("Object登录溢出\n");

return ;

}

ObjectNum++;//Object登录数量增加

后面就都是代入的事情了。

Ob->Img = ImgHandle;//图像句柄

Ob->ImgSize = ImgSize;//图像尺寸

Ob->ImgX1 = ImgX1;

Ob->ImgY1 = ImgY1;

Ob->ImgX2 = ImgX2;

Ob->ImgY2 = ImgY2;

Ob->LargeX = LargeX;//总之我们适当地设定绘制的大小。横纵比和素材一样。

Ob->LargeY = LargeY;

Ob->Type = Type;//将Type设定为垂直

Ob->FromZ = FromZ;//绘制开始位置

Ob->FadeFromZ = FadeFromZ;//绘制淡入开始位置

Ob->FadeToZ = FadeToZ;//绘制淡出开始位置

Ob->ToZ = ToZ;//绘制结束位置

Ob->ObchindMax = OBCHILD\_MAX;

Type为0的时候，表示垂直于画面。

也就是说，在范例中对应于樱花的显示。

我们让垂直于画面的物体能够被指定绘制的个数。

至于除此之外的Type，在范例中我们让其使用预定义的个数。

if( Ob->Type == 0 ){

Ob->ObchindMax = ObchildMax;

}

if( Ob->ObchindMax - 1 <= 0 ){

printfDx("显示数量的设定异常\n");

return ;

}

//Z的幅度计算

Ob->Zhaba = (Ob->FromZ - Ob->ToZ) / (Ob->ObchindMax-1);

至于为什么这里是-1，正如上面说明的那样，这是因为允许绘制的最大数-1章纹理将成为在.FromZ~.ToZ之间的纹理数。（译者注：多余的一张纹理用于追加显示）

计算Z的幅度，然后自动计算从开始绘制的位置开始到结束绘制的位置为止这一范围内的多边形的位置并完成对它们的一系列设置。

float ou1 = (float)Ob->ImgX1 / Ob->ImgSize, ou2 = (float)(Ob->ImgX2 - Ob->ImgX1) / Ob->ImgSize;

float ov1 = (float)Ob->ImgY1 / Ob->ImgSize, ov2 = (float)(Ob->ImgY2 - Ob->ImgY1) / Ob->ImgSize;

for(s=0; s<Ob->ObchindMax; s++){

Ob->ObChild[s].x = GraphX;

Ob->ObChild[s].y = GraphY;

Ob->ObChild[s].z = Ob->ToZ - Ob->Zhaba + Ob->Zhaba \* s;;

for(i=0; i<6; i++){

Ob->ObChild[s].Vertex[i].r = Ob->ObChild[s].Vertex[i].g = Ob->ObChild[s].Vertex[i].b = Ob->ObChild[s].Vertex[i].a = 255;

Ob->ObChild[s].Vertex[i].u = ou1 + ou2 \* VtPm[i].u;

Ob->ObChild[s].Vertex[i].v = ov1 + ov2 \* VtPm[i].v;

}

}

}

初始化函数在这里。只要为对象的初始化函数iniObj传入必要的信息，那么“墙壁”和“地面”地面就被增加进去了。

在这里需要注意的，是同一张图像没有多次读入这一点。

在3D函数中，允许进行“将图像从这里到这里进行使用”这样子的指定方式。

因此，在一个图像文件中经常出现里面放入了多张图像的情况。在范例中就是这样。

这个时候，请只生成一次图像句柄。如果两次载入的话，那么程序会吃掉2倍的内存量。

因为要传给函数的信息是相当的多，每写一个参数就换一行并写上注释的话我认为会比较容易阅读（因为在范例中为了减少行数我省了这个过程）。

void ini (){

int ImgHandle;

ObjectNum = 0;

ImgHandle = LoadGraph( "mydat/img/tex.png" );

IniObj(&Object[0], ImgHandle, 512, 0, 0, 256, 128, 250, 50, 2, 1000, 400, -200, -400, 320, 240-90, OBCHILD\_MAX);

IniObj(&Object[1], ImgHandle, 512, 60,270, 405, 512, 180,125, 0, 1000, 400, -200, -400, 470, 275, 6);

ImgHandle = LoadGraph( "mydat/img/kabe.png" );

IniObj(&Object[2], ImgHandle, 512, 0, 0, 390, 512, 73, 90, 1, 1000, 400, -200, -400, 170, 240, OBCHILD\_MAX);

}

接下来进行对象的计算。

首先，按照Type的不同来计算坐标。感觉上和前面一章很相似呢。

void ClacObject(){

int t,s,i;

for(t=0; t<ObjectNum; t++){

for(s=0; s<Object[t].ObchindMax; s++){

Object[t].ObChild[s].z-=3;

for(i=0;i<6;i++){

switch(Object[t].Type){

case 0://与画面平行

Object[t].ObChild[s].Vertex[i].pos.x = Object[t].ObChild[s].x + Object[t].LargeX \* VtPm[i].x ;

Object[t].ObChild[s].Vertex[i].pos.y = Object[t].ObChild[s].y + Object[t].LargeY \* VtPm[i].y ;

Object[t].ObChild[s].Vertex[i].pos.z = Object[t].ObChild[s].z ;

break;

case 1://与画面垂直(墙壁)

Object[t].ObChild[s].Vertex[i].pos.x = Object[t].ObChild[s].x;

Object[t].ObChild[s].Vertex[i].pos.y = Object[t].ObChild[s].y + Object[t].LargeY \* VtPm[i].y ;

Object[t].ObChild[s].Vertex[i].pos.z = Object[t].ObChild[s].z + Object[t].Zhaba/2\* VtPm[i].x ;

break;

case 2://与画面锤子(地板)

Object[t].ObChild[s].Vertex[i].pos.x = Object[t].ObChild[s].x + Object[t].LargeX \* VtPm[i].x;

Object[t].ObChild[s].Vertex[i].pos.y = Object[t].ObChild[s].y ;

Object[t].ObChild[s].Vertex[i].pos.z = Object[t].ObChild[s].z + Object[t].Zhaba/2\* VtPm[i].y;

break;

}

}

}

接下来是淡入淡出。这个和前面也是一样的。

if( Object[t].FromZ - Object[t].FadeFromZ <= 0 ){

printfDx("Object[%d].From的设定有问题\n",t);

}

else if( Object[t].FadeToZ - Object[t].ToZ <= 0 ){

printfDx("Object[%d].To的设定有问题\n",t);

}

else{

for(s=0; s<Object[t].ObchindMax; s++){

for(i=0; i<6; i++){

float z = Object[t].ObChild[s].Vertex[i].pos.z;

//当前位置如果比绘制范围还要远的话设置透明度为0

if (z < Object[t].ToZ){

Object[t].ObChild[s].Vertex[i].a = 0;

}

//（靠近的时候）如果在淡入的位置的话

else if(Object[t].ToZ < z && z <=Object[t].FadeToZ){

Object[t].ObChild[s].Vertex[i].a = (unsigned char)(255.0f / (Object[t].FadeToZ - Object[t].ToZ) \* (z - Object[t].ToZ)) ;

}

//如果在通常绘制的位置的话

else if(Object[t].FadeToZ <= z && z <= Object[t].FadeFromZ){

Object[t].ObChild[s].Vertex[i].a = 255;

}

//（靠近的时候）如果在淡出的位置的话

else if(Object[t].FadeFromZ <= z && z < Object[t].FromZ){

Object[t].ObChild[s].Vertex[i].a = (unsigned char)(255.0f / (Object[t].FromZ - Object[t].FadeFromZ) \* (Object[t].FromZ - z)) ;

}

//如果比绘制范围还要近的话设置透明度为0

else if(Object[t].FromZ < z){

Object[t].ObChild[s].Vertex[i].a = 0;

}

}

在这里，我们进行循环图像的处理。

当图像来到最靠前的时候，又把它移动到最深处，循环滚动（Loop Scroll），我们像这样子进行处理。

//靠近以致看不见的话

if(Object[t].ObChild[s].z < Object[t].ToZ - Object[t].Zhaba\*0.5f){

//把它移动到最靠近那边的那一侧

float sub = (Object[t].ToZ - Object[t].Zhaba\*0.5f)- Object[t].ObChild[s].z;

Object[t].ObChild[s].z = Object[t].FromZ + Object[t].Zhaba\*0.5f - sub;

}

//远离以致看不见了的话

else if(Object[t].ObChild[s].z > Object[t].FromZ + Object[t].Zhaba\*0.5f){

//将它移动到最靠近这边的这一侧

float sub = Object[t].ObChild[s].z - (Object[t].FromZ + Object[t].Zhaba\*0.5f);

Object[t].ObChild[s].z = Object[t].ToZ - Object[t].Zhaba\*0.5f + sub;

}

}

}

}

}

接下来，我们通过Z来对纹理排序。我们用最简单易懂的简单选择排序算法来进行排序。

排序相当简单，就是简单选择的排序。

void SwapObChild(ObChild\_t \*Ob1,ObChild\_t \*Ob2){

ObChild\_t t = \*Ob1;

\*Ob1 = \*Ob2;

\*Ob2 = t;

}

void SortObject(){

int i,j,t;

for(t=0; t<ObjectNum; t++){

for (i = 0; i < Object[t].ObchindMax ; i++) {

for (j = i + 1; j < Object[t].ObchindMax ; j++) {

if ( Object[t].ObChild[i].z < Object[t].ObChild[j].z ) {

SwapObChild( &Object[t].ObChild[i], &Object[t].ObChild[j] );

}

}

}

}

}

那么我们来试着看看整个程序吧。

--- main.cpp ---

#include "../../../include/DxLib.h"

//不能够太少。因此我们要-1后使用。

#define OBCHILD\_MAX 11

#define OBJECT\_NUM\_MAX 10

//用于用2个三角形多边形来绘制四边形而设定的数值。因为数值是固定的，因此没有必要记住。

typedef struct{

float x,y;

float u,v;

}VtPm\_t;

VtPm\_t VtPm[6]={{-1,1,0,0},{1,1,1,0},{-1,-1,0,1},{1,-1,1,1},{-1,-1,0,1},{1,1,1,0}};

//每一个纹理相关的结构体

typedef struct{

float x,y,z;//中心点

VERTEX\_3D Vertex[6] ; //绘制用的6个顶点

} ObChild\_t;

//大量的纹理集中在一起的信息

//比如在左边不断地连续显示的许多墙壁将在这里管理许多ObChild\_t。

//Obchild\_t的集合

typedef struct{

int Type; // 0：平行于画面、1：垂直于画面

int Img; //图像

int ImgSize;

int ImgX1,ImgX2,ImgY1,ImgY2;

float LargeX,LargeY;//横纵的大小（Type为1的时候LargeX扮演LargeZ的职能）

float Zhaba; //深度的幅度

float FromZ,ToZ; //是否设定从哪儿到哪儿z深度上的移动

float FadeFromZ,FadeToZ; //是否设定从哪儿到哪儿进行淡入淡出（消失的瞬间淡出，显示的瞬间淡入）

int ObchindMax;

ObChild\_t ObChild[ OBCHILD\_MAX ];

} Object\_t;

int ObjectNum;

Object\_t Object[OBJECT\_NUM\_MAX];

/\*

int ImgHandle : 图像句柄

int ImgSize : 图像尺寸

int ImgX1 : 图像使用部分的左上角坐标

int ImgY1 : 图像使用部分的左上角坐标

int ImgX2 : 图像使用部分的右下角坐标

int ImgY2 : 图像使用部分的右下角坐标

float LargeX : 绘制的大小（横向）

float LargeY : 绘制的大小（纵向）

int Type : 绘制的种类 0：与画面一个方向 1：与画面垂直（墙壁） 2：与画面垂直（地面）

float FromZ : 绘制开始的深度

float FadeFromZ : 淡入开始的深度

float FadeToZ : 淡出开始的深度

float ToZ : 绘制种子的深度

float GraphX : 绘制的中心点

float GraphY : 绘制的中心点

int ObchildMax : 只在type为0的时候，决定同时显示多少个

\*/

void IniObj(Object\_t \*Ob, int ImgHandle, int ImgSize, int ImgX1, int ImgY1, int ImgX2, int ImgY2, float LargeX, float LargeY,

int Type, float FromZ, float FadeFromZ, float FadeToZ, float ToZ, float GraphX, float GraphY, int ObchildMax){

int i,s;

if( ObjectNum >= OBJECT\_NUM\_MAX-1 ){

printfDx("Object登录溢出\n");

return ;

}

ObjectNum++;//Object登录数量增加

Ob->Img = ImgHandle;//图像句柄

Ob->ImgSize = ImgSize;//图像尺寸

Ob->ImgX1 = ImgX1;

Ob->ImgY1 = ImgY1;

Ob->ImgX2 = ImgX2;

Ob->ImgY2 = ImgY2;

Ob->LargeX = LargeX;//总之我们适当地设定绘制的大小。横纵比和素材一样。

Ob->LargeY = LargeY;

Ob->Type = Type;//将Type设定为垂直

Ob->FromZ = FromZ;//绘制开始位置

Ob->FadeFromZ = FadeFromZ;//绘制淡入开始位置

Ob->FadeToZ = FadeToZ;//绘制淡出开始位置

Ob->ToZ = ToZ;//绘制结束位置

Ob->ObchindMax = OBCHILD\_MAX;

if( Ob->Type == 0 ){

Ob->ObchindMax = ObchildMax;

}

if( Ob->ObchindMax - 1 <= 0 ){

printfDx(""显示数量的设定异常\n");

return ;

}

//Z的幅度计算

Ob->Zhaba = (Ob->FromZ - Ob->ToZ) / (Ob->ObchindMax-1);

float ou1 = (float)Ob->ImgX1 / Ob->ImgSize, ou2 = (float)(Ob->ImgX2 - Ob->ImgX1) / Ob->ImgSize;

float ov1 = (float)Ob->ImgY1 / Ob->ImgSize, ov2 = (float)(Ob->ImgY2 - Ob->ImgY1) / Ob->ImgSize;

for(s=0; s<Ob->ObchindMax; s++){

Ob->ObChild[s].x = GraphX;

Ob->ObChild[s].y = GraphY;

Ob->ObChild[s].z = Ob->ToZ - Ob->Zhaba + Ob->Zhaba \* s;;

for(i=0; i<6; i++){

Ob->ObChild[s].Vertex[i].r = Ob->ObChild[s].Vertex[i].g = Ob->ObChild[s].Vertex[i].b = Ob->ObChild[s].Vertex[i].a = 255;

Ob->ObChild[s].Vertex[i].u = ou1 + ou2 \* VtPm[i].u;

Ob->ObChild[s].Vertex[i].v = ov1 + ov2 \* VtPm[i].v;

}

}

}

void ini (){

int ImgHandle;

ObjectNum = 0;

ImgHandle = LoadGraph( "mydat/img/tex.png" );

IniObj(&Object[0], ImgHandle, 512, 0, 0, 256, 128, 250, 50, 2, 1000, 400, -200, -400, 320, 240-90, OBCHILD\_MAX);

IniObj(&Object[1], ImgHandle, 512, 60,270, 405, 512, 180,125, 0, 1000, 400, -200, -400, 470, 275, 6);

ImgHandle = LoadGraph( "mydat/img/kabe.png" );

IniObj(&Object[2], ImgHandle, 512, 0, 0, 390, 512, 73, 90, 1, 1000, 400, -200, -400, 170, 240, OBCHILD\_MAX);

}

void ClacObject(){

int t,s,i;

for(t=0; t<ObjectNum; t++){

for(s=0; s<Object[t].ObchindMax; s++){

Object[t].ObChild[s].z-=3;

for(i=0;i<6;i++){

switch(Object[t].Type){

case 0://与画面平行

Object[t].ObChild[s].Vertex[i].pos.x = Object[t].ObChild[s].x + Object[t].LargeX \* VtPm[i].x ;

Object[t].ObChild[s].Vertex[i].pos.y = Object[t].ObChild[s].y + Object[t].LargeY \* VtPm[i].y ;

Object[t].ObChild[s].Vertex[i].pos.z = Object[t].ObChild[s].z ;

break;

case 1://与画面垂直(墙壁)

Object[t].ObChild[s].Vertex[i].pos.x = Object[t].ObChild[s].x;

Object[t].ObChild[s].Vertex[i].pos.y = Object[t].ObChild[s].y + Object[t].LargeY \* VtPm[i].y ;

Object[t].ObChild[s].Vertex[i].pos.z = Object[t].ObChild[s].z + Object[t].Zhaba/2\* VtPm[i].x ;

break;

case 2://与画面锤子(地板)

Object[t].ObChild[s].Vertex[i].pos.x = Object[t].ObChild[s].x + Object[t].LargeX \* VtPm[i].x;

Object[t].ObChild[s].Vertex[i].pos.y = Object[t].ObChild[s].y ;

Object[t].ObChild[s].Vertex[i].pos.z = Object[t].ObChild[s].z + Object[t].Zhaba/2\* VtPm[i].y;

break;

}

}

}

if( Object[t].FromZ - Object[t].FadeFromZ <= 0 ){

printfDx("Object[%d].From的设定有问题\n",t);

}

else if( Object[t].FadeToZ - Object[t].ToZ <= 0 ){

printfDx("Object[%d].To的设定有问题\n",t);

}

else{

for(s=0; s<Object[t].ObchindMax; s++){

for(i=0; i<6; i++){

float z = Object[t].ObChild[s].Vertex[i].pos.z;

//当前位置如果比绘制范围还要远的话设置透明度为0

if (z < Object[t].ToZ){

Object[t].ObChild[s].Vertex[i].a = 0;

}

//（靠近的时候）如果在淡入的位置的话

else if(Object[t].ToZ < z && z <=Object[t].FadeToZ){

Object[t].ObChild[s].Vertex[i].a = (unsigned char)(255.0f / (Object[t].FadeToZ - Object[t].ToZ) \* (z - Object[t].ToZ)) ;

}

//如果在通常绘制的位置的话

else if(Object[t].FadeToZ <= z && z <= Object[t].FadeFromZ){

Object[t].ObChild[s].Vertex[i].a = 255;

}

//（靠近的时候）如果在淡出的位置的话

else if(Object[t].FadeFromZ <= z && z < Object[t].FromZ){

Object[t].ObChild[s].Vertex[i].a = (unsigned char)(255.0f / (Object[t].FromZ - Object[t].FadeFromZ) \* (Object[t].FromZ - z)) ;

}

//如果比绘制范围还要近的话设置透明度为0

else if(Object[t].FromZ < z){

Object[t].ObChild[s].Vertex[i].a = 0;

}

}

//靠近以致看不见的话

if(Object[t].ObChild[s].z < Object[t].ToZ - Object[t].Zhaba\*0.5f){

//一番向こう側へ

//把它移动到最靠近那边的那一侧

float sub = (Object[t].ToZ - Object[t].Zhaba\*0.5f)- Object[t].ObChild[s].z;

Object[t].ObChild[s].z = Object[t].FromZ + Object[t].Zhaba\*0.5f - sub;

}

//远离以致看不见了的话

else if(Object[t].ObChild[s].z > Object[t].FromZ + Object[t].Zhaba\*0.5f){

//将它移动到最靠近这边的这一侧

float sub = Object[t].ObChild[s].z - (Object[t].FromZ + Object[t].Zhaba\*0.5f);

Object[t].ObChild[s].z = Object[t].ToZ - Object[t].Zhaba\*0.5f + sub;

}

}

}

}

}

void SwapObChild(ObChild\_t \*Ob1,ObChild\_t \*Ob2){

ObChild\_t t = \*Ob1;

\*Ob1 = \*Ob2;

\*Ob2 = t;

}

//用Z来对纹理排序

void SortObject(){

int i,j,t;

for(t=0; t<ObjectNum; t++){

for (i = 0; i < Object[t].ObchindMax ; i++) {

for (j = i + 1; j < Object[t].ObchindMax ; j++) {

if ( Object[t].ObChild[i].z < Object[t].ObChild[j].z ) {

SwapObChild( &Object[t].ObChild[i], &Object[t].ObChild[j] );

}

}

}

}

}

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow ){

int s,t;

ChangeWindowMode(TRUE);//窗口模式

if(DxLib\_Init() == -1 || SetDrawScreen( DX\_SCREEN\_BACK )!=0) return -1;//初始化于里画面化

ini();

while(ProcessMessage()==0 && ClearDrawScreen()==0 && CheckHitKey(KEY\_INPUT\_ESCAPE)==0){

ClacObject();

SortObject();

SetDrawMode( DX\_DRAWMODE\_BILINEAR ) ;//为了让多边形看起来平滑我们使用“双线性内插法”的绘制方法

for(t=0; t<ObjectNum; t++){

for(s=0; s<Object[t].ObchindMax; s++){

DrawPolygon3D( Object[t].ObChild[s].Vertex, 2, Object[t].Img, TRUE ) ;

}

}

SetDrawMode(DX\_DRAWMODE\_NEAREST);//还原绘制方式

ScreenFlip() ;

}

DxLib\_End() ;

return 0 ;

}

在mian函数的循环中我们使用了“双线性内插法”（Bi-linear Interpolation）的绘制方法。

这种方法在扩大图像的时候锯齿（译者注：原文为dot，这里意译为锯齿）不会太显眼。

由于对点与点之间进行插值再绘制，因此可以进行平滑的绘制。

至于如果不使用这个方法会怎么样，还请您自行尝试一下。尤其是在樱花扩大的时候您会很容易体会到的。

不过，由于某种原因需要画像不那么清晰，在不需要插值的时候，我们就用标准的绘制方法来绘制吧。

## 第六十章 在游戏中试着使用3D背景吧

在这一章中，我们把在第59章中完成的3D背景给适用到我们的游戏中吧。

源代码基本上没有必要改变。

只需要改变初始化函数的名字、改变main函数的名字，然后稍微变更一下main循环的处理即可。

这次将这个59章中的main.cpp文件以3Dbackground.cpp的名字追加到我们的游戏中吧。

在解决方案管理器的“graph”文件夹上右键->“添加”->“新项目”，然后追加3Dbackground.cppp。

复制粘贴到那里就行了。

ini()变更为ini3Dbackground()这个函数名。

main函数变更为Main3Dbackground()这个函数名，其内部的处理稍微进行了变更，请您确认。

具体修改的位置用水色标识出来了。

dat/img/back/3D/

另外，这一次追加了的图像都放在上面那个文件夹中。如果您是自己建立项目的话，请确认一下。

---在 3Dbackground.cpp 进行以下追加 ---

#include "../include/GV.h"

//不能够太少。因此我们要-1后使用。

#define OBCHILD\_MAX 11

#define OBJECT\_NUM\_MAX 10

//用于用2个三角形多边形来绘制四边形而设定的数值。因为数值是固定的，因此没有必要记住。

typedef struct{

float x,y;

float u,v;

}VtPm\_t;

VtPm\_t VtPm[6]={{-1,1,0,0},{1,1,1,0},{-1,-1,0,1},{1,-1,1,1},{-1,-1,0,1},{1,1,1,0}};

//每一个纹理相关的结构体

typedef struct{

float x,y,z;//中心点

VERTEX\_3D Vertex[6] ; //绘制用的6个顶点

} ObChild\_t;

//大量的纹理集中在一起的信息

//比如在左边不断地连续显示的许多墙壁将在这里管理许多ObChild\_t。

//Obchild\_t的集合

typedef struct{

int Type; // 0：平行于画面、1：垂直于画面

int Img; //图像

int ImgSize;

int ImgX1,ImgX2,ImgY1,ImgY2;

float LargeX,LargeY;//横纵的大小（Type为1的时候LargeX扮演LargeZ的职能）

float Zhaba; //深度的幅度

float FromZ,ToZ; //是否设定从哪儿到哪儿z深度上的移动

float FadeFromZ,FadeToZ; //是否设定从哪儿到哪儿进行淡入淡出（消失的瞬间淡出，显示的瞬间淡入）

int ObchindMax;

ObChild\_t ObChild[ OBCHILD\_MAX ];

} Object\_t;

int ObjectNum;

Object\_t Object[OBJECT\_NUM\_MAX];

/\*

int ImgHandle : 图像句柄

int ImgSize : 图像尺寸

int ImgX1 : 图像使用部分的左上角坐标

int ImgY1 : 图像使用部分的左上角坐标

int ImgX2 : 图像使用部分的右下角坐标

int ImgY2 : 图像使用部分的右下角坐标

float LargeX : 绘制的大小（横向）

float LargeY : 绘制的大小（纵向）

int Type : 绘制的种类 0：与画面一个方向 1：与画面垂直（墙壁） 2：与画面垂直（地面）

float FromZ : 绘制开始的深度

float FadeFromZ : 淡入开始的深度

float FadeToZ : 淡出开始的深度

float ToZ : 绘制种子的深度

float GraphX : 绘制的中心点

float GraphY : 绘制的中心点

int ObchildMax : 只在type为0的时候，决定同时显示多少个

\*/

void IniObj(Object\_t \*Ob, int ImgHandle, int ImgSize, int ImgX1, int ImgY1, int ImgX2, int ImgY2, float LargeX, float LargeY,

int Type, float FromZ, float FadeFromZ, float FadeToZ, float ToZ, float GraphX, float GraphY, int ObchildMax){

int i,s;

if( ObjectNum >= OBJECT\_NUM\_MAX-1 ){

printfDx("Object登录溢出\n");

return ;

}

ObjectNum++;//Object登录数量增加

Ob->Img = ImgHandle;//图像句柄

Ob->ImgSize = ImgSize;//图像尺寸

Ob->ImgX1 = ImgX1;

Ob->ImgY1 = ImgY1;

Ob->ImgX2 = ImgX2;

Ob->ImgY2 = ImgY2;

Ob->LargeX = LargeX;//总之我们适当地设定绘制的大小。横纵比和素材一样。

Ob->LargeY = LargeY;

Ob->Type = Type;//将Type设定为垂直

Ob->FromZ = FromZ;//绘制开始位置

Ob->FadeFromZ = FadeFromZ;//绘制淡入开始位置

Ob->FadeToZ = FadeToZ;//绘制淡出开始位置

Ob->ToZ = ToZ;//绘制结束位置

Ob->ObchindMax = OBCHILD\_MAX;

if( Ob->Type == 0 ){

Ob->ObchindMax = ObchildMax;

}

if( Ob->ObchindMax - 1 <= 0 ){

printfDx(""显示数量的设定异常\n");

return ;

}

//Z的幅度计算

Ob->Zhaba = (Ob->FromZ - Ob->ToZ) / (Ob->ObchindMax-1);

float ou1 = (float)Ob->ImgX1 / Ob->ImgSize, ou2 = (float)(Ob->ImgX2 - Ob->ImgX1) / Ob->ImgSize;

float ov1 = (float)Ob->ImgY1 / Ob->ImgSize, ov2 = (float)(Ob->ImgY2 - Ob->ImgY1) / Ob->ImgSize;

for(s=0; s<Ob->ObchindMax; s++){

Ob->ObChild[s].x = GraphX;

Ob->ObChild[s].y = GraphY;

Ob->ObChild[s].z = Ob->ToZ - Ob->Zhaba + Ob->Zhaba \* s;;

for(i=0; i<6; i++){

Ob->ObChild[s].Vertex[i].r = Ob->ObChild[s].Vertex[i].g = Ob->ObChild[s].Vertex[i].b = Ob->ObChild[s].Vertex[i].a = 255;

Ob->ObChild[s].Vertex[i].u = ou1 + ou2 \* VtPm[i].u;

Ob->ObChild[s].Vertex[i].v = ov1 + ov2 \* VtPm[i].v;

}

}

}

void Ini3Dbackground(){

int ImgHandle;

ObjectNum = 0;

ImgHandle = LoadGraph( "../dat/img/back/3D/tex.png" );

IniObj(&Object[0], ImgHandle, 512, 0, 0, 256, 128, 250, 50, 2, 1000, 400, -200, -400, 320, 240-90, OBCHILD\_MAX);

IniObj(&Object[1], ImgHandle, 512, 60,270, 405, 512, 180,125, 0, 1000, 400, -200, -400, 470, 275, 6);

ImgHandle = LoadGraph( "../dat/img/back/3D/kabe.png" );

IniObj(&Object[2], ImgHandle, 512, 0, 0, 390, 512, 73, 90, 1, 1000, 400, -200, -400, 170, 240, OBCHILD\_MAX);

}

void ClacObject(){

int t,s,i;

for(t=0; t<ObjectNum; t++){

for(s=0; s<Object[t].ObchindMax; s++){

Object[t].ObChild[s].z-=3;

for(i=0;i<6;i++){

switch(Object[t].Type){

case 0://与画面平行

Object[t].ObChild[s].Vertex[i].pos.x = Object[t].ObChild[s].x + Object[t].LargeX \* VtPm[i].x ;

Object[t].ObChild[s].Vertex[i].pos.y = Object[t].ObChild[s].y + Object[t].LargeY \* VtPm[i].y ;

Object[t].ObChild[s].Vertex[i].pos.z = Object[t].ObChild[s].z ;

break;

case 1://与画面垂直(墙壁)

Object[t].ObChild[s].Vertex[i].pos.x = Object[t].ObChild[s].x;

Object[t].ObChild[s].Vertex[i].pos.y = Object[t].ObChild[s].y + Object[t].LargeY \* VtPm[i].y ;

Object[t].ObChild[s].Vertex[i].pos.z = Object[t].ObChild[s].z + Object[t].Zhaba/2\* VtPm[i].x ;

break;

case 2://与画面锤子(地板)

Object[t].ObChild[s].Vertex[i].pos.x = Object[t].ObChild[s].x + Object[t].LargeX \* VtPm[i].x;

Object[t].ObChild[s].Vertex[i].pos.y = Object[t].ObChild[s].y ;

Object[t].ObChild[s].Vertex[i].pos.z = Object[t].ObChild[s].z + Object[t].Zhaba/2\* VtPm[i].y;

break;

}

}

}

if( Object[t].FromZ - Object[t].FadeFromZ <= 0 ){

printfDx("Object[%d].From的设定有问题\n",t);

}

else if( Object[t].FadeToZ - Object[t].ToZ <= 0 ){

printfDx("Object[%d].To的设定有问题\n",t);

}

else{

for(s=0; s<Object[t].ObchindMax; s++){

for(i=0; i<6; i++){

float z = Object[t].ObChild[s].Vertex[i].pos.z;

//当前位置如果比绘制范围还要远的话设置透明度为0

if (z < Object[t].ToZ){

Object[t].ObChild[s].Vertex[i].a = 0;

}

//（靠近的时候）如果在淡入的位置的话

else if(Object[t].ToZ < z && z <=Object[t].FadeToZ){

Object[t].ObChild[s].Vertex[i].a = (unsigned char)(255.0f / (Object[t].FadeToZ - Object[t].ToZ) \* (z - Object[t].ToZ)) ;

}

//如果在通常绘制的位置的话

else if(Object[t].FadeToZ <= z && z <= Object[t].FadeFromZ){

Object[t].ObChild[s].Vertex[i].a = 255;

}

//（靠近的时候）如果在淡出的位置的话

else if(Object[t].FadeFromZ <= z && z < Object[t].FromZ){

Object[t].ObChild[s].Vertex[i].a = (unsigned char)(255.0f / (Object[t].FromZ - Object[t].FadeFromZ) \* (Object[t].FromZ - z)) ;

}

//如果比绘制范围还要近的话设置透明度为0

else if(Object[t].FromZ < z){

Object[t].ObChild[s].Vertex[i].a = 0;

}

}

//靠近以致看不见的话

if(Object[t].ObChild[s].z < Object[t].ToZ - Object[t].Zhaba\*0.5f){

//一番向こう側へ

//把它移动到最靠近那边的那一侧

float sub = (Object[t].ToZ - Object[t].Zhaba\*0.5f)- Object[t].ObChild[s].z;

Object[t].ObChild[s].z = Object[t].FromZ + Object[t].Zhaba\*0.5f - sub;

}

//远离以致看不见了的话

else if(Object[t].ObChild[s].z > Object[t].FromZ + Object[t].Zhaba\*0.5f){

//将它移动到最靠近这边的这一侧

float sub = Object[t].ObChild[s].z - (Object[t].FromZ + Object[t].Zhaba\*0.5f);

Object[t].ObChild[s].z = Object[t].ToZ - Object[t].Zhaba\*0.5f + sub;

}

}

}

}

}

void SwapObChild(ObChild\_t \*Ob1,ObChild\_t \*Ob2){

ObChild\_t t = \*Ob1;

\*Ob1 = \*Ob2;

\*Ob2 = t;

}

//用Z来对纹理排序

void SortObject(){

int i,j,t;

for(t=0; t<ObjectNum; t++){

for (i = 0; i < Object[t].ObchindMax ; i++) {

for (j = i + 1; j < Object[t].ObchindMax ; j++) {

if ( Object[t].ObChild[i].z < Object[t].ObChild[j].z ) {

SwapObChild( &Object[t].ObChild[i], &Object[t].ObChild[j] );

}

}

}

}

}

void Main3Dbackground(){//(60)

int t,s;

ClacObject();

SortObject();

SetDrawArea(FX,FY,FX+FMX,FY+FMY);//设定允许绘制的区域

SetDrawMode( DX\_DRAWMODE\_BILINEAR ) ;// 了让多边形看起来平滑我们使用“双线性内插法”的绘制方法

for(t=0; t<ObjectNum; t++){

for(s=0; s<Object[t].ObchindMax; s++){

DrawPolygon3D( Object[t].ObChild[s].Vertex, 2, Object[t].Img, TRUE ) ;

}

}

SetDrawMode(DX\_DRAWMODE\_NEAREST);//还原绘制方式

SetDrawArea(0,0,640,480);//还原允许绘制的区域

}

---在 func.h 中把下面这些注释掉---

// boss\_shot\_bulletH014,//漢字弾幕(56)

// boss\_shot\_bulletH012,//漢字弾幕(53)

// boss\_shot\_bulletH013,//漢字弾幕(53)

---在 ini.cpp 中进行红字部分追加---

//游戏初始化

void ini(){

Ini3Dbackground();//(60)

stage\_count=1;

memset(&ch,0,sizeof(ch\_t));

---在 function.h 中进行以下追加 ---

//3Dbackground.cpp//(60)

GLOBAL void Ini3Dbackground();

GLOBAL void Main3Dbackground();

---在 function.h 中进行以下追加---

//读入AA弹幕的子弹坐标数据(56)

GLOBAL int load\_AA\_dat(char name[64], BlAAPoint\_t \*Bp);

---在 graph\_back.cpp 中进行以下修正 ---

void graph\_back\_main(){

//Boss为有效且为SC

if(boss.flag==1 && boss.back\_knd[boss.knd]==1)

graph\_back01();

else//除此之外

Main3Dbackground();//(60)

// graph\_back00();

}

运行结果

另外，如果我们载入的图像为kabe2.png的话就会是这个样子。

这个纹理要怎样贴上去我想是很容易理解的吧。

## 译者后记

大概在2012年的时候我就知道Dixq先生的个人网站“C言語何でも質問サイト”了，那个时候他就已经基本完成了许多关于DX Library的教程，我那时因为各种原因只是匆匆看了一眼，但那个时候就觉得这写得相当不错。后来自己写游戏引擎，目前也基本成型，后来对于STG开发比较感兴趣于是想到了 Dixq先生的教程，于是我决定着手进行他的一系列教程的翻译。

Dixq先生的教程浅显易懂，很容易上手，但是这么好的教程却因为语言不通的原因使得国人对游戏开发抱有兴趣的人难以阅读，因此怀揣着在国内宣传Dx Library这个强大的游戏库的心情完成了翻译。60章正文+3章番外的《龙神录编程馆系列教程》的翻译前前后后用了一个半月多点的时间完成，中间因为一些个人原因耽搁了许久实在是抱歉。

通过翻译Dixq先生的文章我也自己学到了许多东西，虽然这篇教程中的代码是用C语言写的，而且许多地方都用的硬编码，但是其内在的编程思想与算法确实能够适用并移植开去的，简而言之这篇教程只是提供给大家提供了敲门砖，而想要继续深入下去的还需要各自的自行努力，所谓师父领进门修行在个人就是这个意思了。

纵观国内的游戏开发领域，优秀的教程方面实在是过于匮乏，技术高明的人不愿或懒得书写简单易懂的详细教程，而想入门的人又找不到合适的入门资料——只能从技术高明的人的只言片语那里自行揣摩道路，这使得国内的游戏开发领域呈现出封闭沉闷的感觉，缺乏资料分享的朝气。笔者认为，无私而落实的技术分享是相当重要的，这也是国内和国外游戏开发圈层的区别。

这里个人发表这样的观点或许有些武断，但是个人觉得当前国内游戏开发圈确实是能者自视清高而冷眼旁观，而不能者则是求门无路而苦苦挣扎，个人希望这个怪圈能够被打破，虽然笔者能力甚是微弱，但是希望这篇翻译以及以后的文章翻译能对此产生一些帮助——尤其是对那些想入门而难寻道路的人。

如果您想与我做更深的交流或者对文中翻译不对的地方有所异议，不吝赐教，我的邮箱如下：

[10261212@qq.com](mailto:10261212@qq.com)

[a1026121287@hotmail.com](mailto:a1026121287@hotmail.com)

谨以此文，献予爱游戏的人。

Hui

2015-1-9