**NierPlane后期开发记录**

建议格式

{

格式:

日志编号: 性质(增C 删D 改U)\_日期(方便以后 ctrl+ f 查找) \_ 当天第几次日志 //CRUD, 分别是create, retrieve update delete

修改/新增/删除 的文件名

具体变更描述, 在描述较长时, 可用大括号{} 括起.

格式例:

C\_0118\_1 // C- create 即新建文件

文件: Timer.h

对time相关的计时器函数进行包装, 添加了一些功能函数.

}

**1月15日**

U\_0115\_1

文件：starting.h/cpp

将原来所有与某个特定功能相关的全局变量整合到类里面。

**1月16日**

U\_0116\_1

文件：starting.h/cpp

修改了随机雪花的实现，通过随机起点+随机长度+随机线型绘制出随机线，并利用正弦函数做出周期的效果，用动量方法使得相邻的随机线有一定的相关性

**1月18日**

C\_0118\_1

文件：Timer.h

新增文件Timer.h

{

仅.h, 具体实现全在.h文件中,(后续把实现移到.cpp文件中).

Timer.h是计时器的包装类, 为计时器添加了一些常用功能

Timer(long \_timeLength);// 构造函数, 参数为计时长度(ms)

// 当距离上一次updateTime的时间超过 timeLength的时候, 返回true, 否则返回false

bool ifOverTime();

void updateTime(); //更新计时器

//返回现在的时间距离上次updateTime的具体时间长度(ms);

long getLengthBetweenPreviousAndNow();

例:

{

clock\_t time1 = clock(), time2;

while(time2 - time1 > 1000){

time2 = clock();

Sleep(10);

}

cout << time2 - time1 << endl;

}

可改写为:

{

Timer time1(1000);

while(time1.ifOverTime()){

// 判断距离上次更新时间是否超过1000ms(构造函数中给定值)

Sleep(10);

}

cout << time1.getLengthBetweenPreviousAndNow() << endl;

time1.updateTime(); // 更新为当前时间/或理解为归零

}

}

U\_0118\_2

文件：nierPlane.h/cpp

1 . 更改计时器实现, 全部更改为调用Timer实现

2 . 更改暂停实现, 按 [P键] 可 暂停/继续, 暂停后计时器不会继续计时

3 . 更改调试参数显示(即左上角的粒子数, 循环时间等). 改为按 [F3] 可开启/关闭

4 . 新增第二行调试参数, 显示计时器当前计时长度.

**1月19日**

U\_0119\_1

文件：starting.h/cpp

1. 新增了齿轮实现，通过偏移坐标+角度控制完成齿牙、齿轴的绘制与旋转，大小可调
2. 对于由于计算机屏幕显示精度带来的误差，目前仍无较好的解决方案

**1月20日**

U\_0120\_1

文件：common.h

1. 新增了myputimage2函数，实现图像直接覆盖的放置。

D\_0120\_2

文件：starting.cpp

{

删除之前在背景绘制大白色块的代码，改为先把这张图片画出来，再放上去，以不受图像缩放的影响。另外把删除了的代码放在这里保存。

const TCHAR\* bk\_name = \_T("mynierbk.png");

loadimage(&bk\_img, bk\_name, win\_wid, win\_hei);

//SetWorkingImage();

//setfillcolor(RGB(230, 230, 218));

//int pts\_x[14] = { 1090, 750, 750, 760, 760, 800, 800, 810, 810, 820, 820, 900, 900, 1090 };

//int pts\_y[14] = { 386, 386, 370, 370, 360, 360, 350, 350, 340, 340, 330, 330, 320, 320 };

//POINT pts[14];

//for (int i = 0; i <= 13; i++)

// pts[i] = { static\_cast<long>(x\_ratio \* pts\_x[i]) , static\_cast<long>(y\_ratio \* pts\_y[i]) };

//setrop2(R2\_XORPEN);

//fillpolygon(pts, 14);

//IMAGE temp;

//getimage(&temp, 429 \* x\_ratio, 316 \* y\_ratio, 21 \* x\_ratio, 21 \* y\_ratio); // 取一块适当的区域以移植

//myputimage2(455 \* x\_ratio, 315 \* y\_ratio, &temp, 0); // 这里是把第一个字母i上面的点擦除掉

//saveimage(\_T("mynierbk.png"));}

**1月21日**

C\_0121\_1

文件: particle\_planeHit1.h/cpp, particle\_planeHit2.h/cpp, particle\_planeHit3.h/cpp

1. 新增己方飞机受击的效果(1,2,3), 1为内光圈, 2为外光圈+向内光晕, 3为效果2消失是残留的多个矩形块.

U\_0121\_2

文件: nierPlane.cpp

1. 主循环内增加判断 是否有子弹被己方受击产生的光晕(C0121中效果2) 击中的子弹 的分支;
2. 添加 U\_0121\_2\_1.所述效果的实现函数
3. 修复了增援后鼠标可能失效的bug, 但仅在本机测试成功, 不清楚其他机器是否还会出bug

U\_0121\_3

文件: common.h

1. 添加clamp功能函数, 作用为限制变量范围. 用法例 x = clamp(x,0,256); //大于256被降为256, 小于0被升为0.

{

template<class T>

static T clamp(T x, T min, T max) {

if (x > max) {

return max;

}

if (x < min) {

return min;

}

return x;

}

}

U\_0121\_4

文件：starting.cpp

1. 将绘制开始界面主程序running()进行简化，任何部件的初始化及其绘制尽可能地集中到对应类的成员函数中去，开始着手文字出现与消失特效开发

**1月22日**

C\_0122\_1

文件: MyMusic.h/cpp

1. 新增函数 turn\_up\_volumn() /turn\_down\_volumn()用来调节音乐音量(暂时仅对音乐有效, 子弹音效暂未设置;

C\_0122\_2

文件: particle\_planeHit4.h/cpp

1. 新增受击时 屏幕边缘显示红光的特效

U\_0122\_3

文件: nierPlane.h/cpp

1. 细微修改主循环流程, 将读取键盘操作尽量集中到函数myGetKeyBoardState()中, 添加功能键 ↑, ↓

目前的功能键

( W: 飞机向上, A: 飞机向左, S: 飞机向下, D: 飞机向右, P: 暂停, F3: 输出测试信息, ↑: 调高音量, ↓: 调低音量.

U\_0122\_4

文件：starting.h/cpp

1. 对Text类新增了文字逐个出现、文字淡入、淡出、以及文字下划线淡入、淡出的实现。文字特效完成。开始着手菜单的实现。
2. 把void PARTICLE::show\_particle中的三个平滑函数分别放入多线程中，能使一个循环所花费的时间减少10-15ms
3. 对于实现上一输出文字与下一输出文字之间的串联，采用类静态成员变量完成。PS：静态成员变量对于实现不同成员之间的联系，真的方便。

**1月24日**

U\_0124\_1

文件：starting.h/cpp

1. 实现了菜单功能，能进行上下菜单的选择。上是‘W’||‘w'，下是’S‘||’s'，确认是Enter
2. 由于之前有一些功能耦合在一起，在这次制作菜单的功能时，曾想不管耦合的问题，强行实现当菜单转换时的一些过渡特效，之后发现几乎是不可能的，导致这次做菜单做得比较久。之后进行了解耦，使得对一些特效的操作更加清晰和方便。
3. 本次使用const成员函数为开发带来很多的清晰指引。这是得益于这个类型限定符能清晰地表明调用哪些函数，能无须担心类成员被不小心修改。

**NierPlane开发时一些值得记录下的点**

1. 在做开始界面的光晕特效时，首先得益于后面对子弹光晕形成的研究，使用数字图像处理中的平滑方法对光晕进行处理，一开始能取得比较好的效果，尤其是得到了那种“晕”的视觉效果。但是后面发现该光晕中出现了格网状分布的暗点，根据这一分布特点，推测是由于在制作平滑模板时，平滑模板的各参数采用了类似于棋盘距离的计算方式确定参数，导致了格网状的暗点。因此将模板内趋于中部的参数值尽可能调大一些，而位于边角的参数值尽可能调小一些，这样做了之后取得了一个不错的效果，具体表现为光晕内的分布均匀，肉眼未能观测到暗点。
2. 在制作游戏部分的敌人出场时的红色粒子的运动特效时，大的方案做了三个，前两个都未能取得满意的效果。在做第三个方案时，决定仔细研究原游戏中红色粒子运动轨迹特点，最后猜想红色粒子应该是被置于一个引力场中，其运动特点应该符合天体运动的特点。便决定在画布中模拟出一个引力场，并对粒子的初始速度加以控制，最终取得了一个不错的效果。
3. 在制作游戏部分的敌人运动轨迹时，由于在偶然的机会下，了解过控制论中的PID算法以及知道它比较适合做出一种震荡的运动轨迹，又是在偶然的机会下，了解过在训练神经网络中SGD算法中的Momentum方法，能使得运动轨迹趋于平滑，于是将这两种方法结合应用在运动轨迹的制作中。后面再对PID算法中的原点设置从固定的变为实时随机设定的，为运动轨迹提供了一个很好的外驱动力。
4. 在对游戏中所有需要被旋转的图形，对其的绘制采用了基准坐标+相对坐标的画法，再采用线性代数中的旋转矩阵，利用该矩阵对相对坐标进行矩阵乘法，实现图形的旋转。

**规范**

1. 对于单纯常量，尽量用const或enum替换#define。详见条款02。
2. 对于形似函数的宏（macros），最好改用inline函数替换#define。详见条款02。
3. 尽可能使用const。详见条款03。
4. 构造函数最好是用成员初值列（member initialization list），而不要在构造函数本体内使用赋值操作（assignment）。初值列列出的成员变量，其排列次序应该和它们在class中的声明次序相同。详见条款04。
5. 为免除“跨编译单元之初始化次序“问题，请以local static对象替换non-local static对象。详见条款04。
6. 为多态基类声明virtual析构函数，该析构函数要写定义，否则连接器在一些情形下会报错。详见条款07。