**C语言项目报告**

**项目名称：星际战舰红色十月号**

**项目成员：陈知周、许扬、赵宁远**

1. **摘要（Abstract）**

由于市面上的战舰类游戏主要是以战舰群为主体的整体战略布局游戏，众所周知：不想当将军的士兵不是好士兵，不想上前线的将军不是好将军，我们希望可以细化游戏操作，将单艘战舰为主体来重新设计一款游戏，同时我们的设计可以为之后的战舰群和单艘战舰结合类游戏提供借鉴，建立一款新型的多人组队的战舰类游戏，满足人们对战舰类游戏的需求。

1. **问题描述（Problem Statement）**
2. 项目背景：

现有的多款太空战游戏中，主要以战舰整体为目标的一种策略性游戏。我们希望进一步改进该类游戏，做出一种即可以选择攻击战舰局部，又可以操纵战舰总体布局的一款新型的战舰类游戏。

1. 面临问题：

1）没有其他类似游戏可以供我们借鉴，故此需要自己设计游戏框架及思路。

2）游戏界面的绘制以及战舰的绘制

3）玩家的触觉体验，听觉体验以及视觉体验的需求

1. **组内分工（Group Division）**
2. 陈知周（40%）：作为组长寻找各种需要的游戏素材，完成了动画制作，文本设计，战舰图片绘制，和帧循环中最后结算时的判断，人机对抗时机器部分代码编写，以及对组员代码编写进行指导和与组员共同克服难关。
3. 赵宁远（30%）：负责基础的结构体设置和文件读入以及相对较为简单的按钮和界面中鼠标检测和状态改变，部分UI的绘制。

3. 许扬（30%）：负责相对复杂的区块中鼠标检测和状态改变和移动版块代码编写以及后期查bug，以及部分UI的绘制。

**四．需求分析（Analysis）**

用户的需求可分为视觉需求，听觉需求以及触觉反馈。

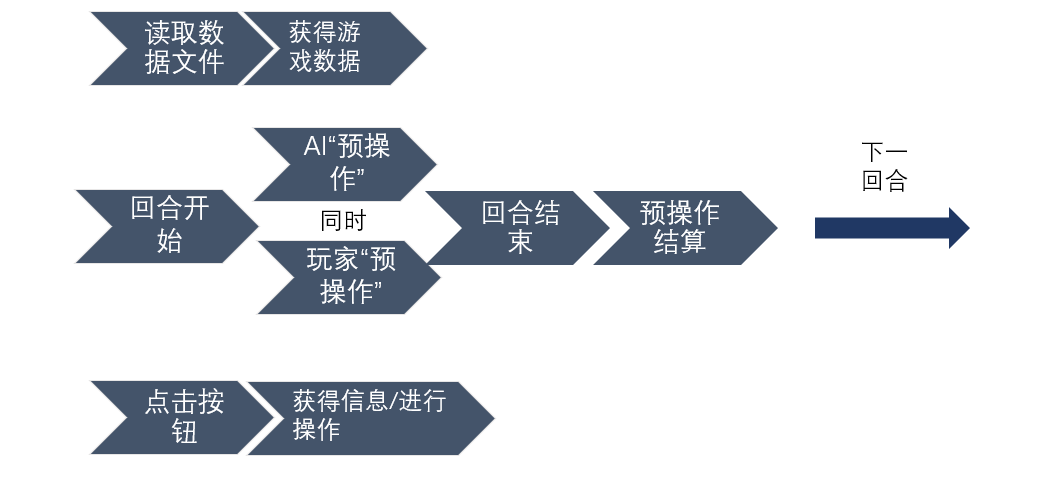
我们分别通过制作富有代入感的动画，风格协调的原创UI；在开始界面加入背景音乐；

以及进行60帧率的输入检测，屏幕刷新来实现用户需求的满足。

1. **设计（Design）**

主体框架分为三个模块：界面设计，玩家操作以及电脑操作。

设计逻辑：我们先设计了游戏界面，再用随机数代替AI完成电脑操作（前期的简单代替），最后编写了关于玩家操作的一系列代码。

****

1. **实施（Implementation）**

1.模块设计：

1.1 输入检测：

鼠标移动到按钮所在区域，则按钮会高亮显示，点击按钮，改变程序状态；

1.2 瞄准与攻击：

切换到瞄准界面（底端第四个按钮），可用的武器会高亮显示，点击该武器，可以

瞄准对应的敌人区块；在回合结束后发射武器，造成对应的伤害。敌人同理。

1.3 状态显示：

左上、右上屏幕会显示舰船受损的状态，右中屏幕显示敌舰真实图像，雷达界面显示各舰船的实时位置等。

1.4 动画播放：

武器开火，舰船被摧毁时爆炸等

1.5 移动：

在雷达屏幕选择舰船下回合将要移动的位置，敌人同时也会做出移动；

1.6 数值结算：

武器对对应的区块造成伤害，计算能量消耗

1.7 AI算法：

敌人的AI如何根据现场的状态做出反应，采取何种战术攻击玩家。

1.8 存/读取数据：

保存游戏进度，读取武器、舰船的数值

2.数据存储

2.1 GUI的状态位

定义当前该显示什么界面，所有相关的数据都存储于UIFLAG、BTFL两个结构体中。

其中，BTFL存储的是对整个程序较为关键的数值。

定义如下：

struct UIFLAG //用于定义UI的状态位，在刷新屏幕时使用

{

int w=0; //“view details”选择状态，未选择为0

int r=1; //雷达状态， 有5个值 0=图像，1=显示板，2=雷达，3=星图，4=武器

int lscr = 0; //左中显示器显示状态，有3个值 1=来袭预警，2=引擎状态，3=武器状态 0=空

int rscr = 0; //右侧显示器显示状态，有2个值 1=敌舰图像，2=视频通话

int p\_e = 0; //鼠标指向能量面板状态

int p\_w = 0; //鼠标指向武器面板状态 值取1-5

int p\_enscr = 0;int p\_plscr = 0; //鼠标指向左上、右上面板状态

int p\_l = 0; //鼠标指向左上按钮状态

int p\_r1 = 0; int p\_r2 = 0; int p\_r3 = 0; int p\_r4 = 0; //鼠标指向左下4按钮状态

int p\_mov = 0;//指向“移动”

int p\_sec = -1; //取1-16

int p\_sec\_pl = -1; //指向了哪个玩家区块

int p\_dpl\_rw = 0; //指向“Deploy B.W”

int p\_turn = 0; //指向 “结束回合”

int sec\_slected[16] = { 0 };//区块有无被瞄准

int move = 0; //点击移动

int dpl\_rw = 0;//RAD界面点击开火

int fire[10] = { 0,0,0,0,0,0,0,0,0,0 }; //正经的开火

struct ship\_slec { //三元数组，瞄准舰船-》使用的武器-》瞄准的区块

int slec\_sec[5] = { -1,-1,-1,-1,-1 };//取0-16

}ship\_slec[20];

int s\_asys = 0; //左中小屏幕缩进状态

int m\_w = -1; //左中武器面板，开火状态

int m\_wd = 0; //武器受损状态

int m\_ed = 8; //引擎面板，受损状态

bool ani\_click = false; //在动画中点击了

}UIFLAG;

注：1.带有”p\_”的为记录鼠标是否移动到了某个可互动区域上，若值为1，则会高亮显示

2.结构体中嵌套的结构体数组用于实现三元数（被瞄准舰船，使用的武器，瞄准的区块）的存储。

struct BUTTONFLAG //用于定义按钮的状态位（是否按了）

{

int wp\_c[10] = { 0,0,0,0,0,0,0,0,0,0 }; //在普通界面，武器有无被点击

int wp\_c4[10] = { 0,0,0,0,0,0,0,0,0,0 }; //在瞄准敌舰界面，武器有无被点击 0=unselected 1=selecting 2=selected

int movslec = 0; //是否选择了移动

int radar\_clicked = 0; //点击了雷达上的哪个东西，取值为0~20？

}BTFL;

2.2 舰船数据

定义了shipentity结构体，存储所有与舰船相关的数据。

struct shipentity //舰船实体

{

bool initialized = false;//是否被初始化了

int stance = 2; //0=中立，1=友好，-1=敌对，2=未知

char nam[20] = { 0 };

int t = 0; //type

int energy = 0;

int mov = 0; //移动力，决定先手/后手，移动力低的敌人的移动路径将会显示

int wpsys\_num = 1;

int englv[3] = {0,0,0};

int shield = 0;

int shield\_r = 0; //实时护盾

int shieldsys\_int\_ful[3]; int shield\_int[3]; int shield\_in[3] = {-1,-1,- 1} ; //各系统完整生命值，及当前生命值,开始新战斗时，别忘了把当前生命值恢复！

int armorsys\_int\_ful[3]; int armor\_int[3]; int armor\_in[3] = {-1, -1, -1};

int wpsys\_int\_ful[3]; int wp\_int[3]; int wpsys\_in[3] = { -1,-1,-1 };

int eng\_int\_ful[3]; int eng\_int[3]; int eng\_ins[3] = { -1,-1,-1 };

int reac\_int\_ful[3]; int reac\_int[3]; int reac\_ins[3] = { -1,-1,-1 };

int carrier\_int\_ful[3]; int carrier\_int[3]; int carr\_ins[3] = { -1,-1,-1 };

int wp\_[10] = { 0,0,0,0,0,0,0,0,0,0 }; //武器有无检测。0=无武器，1或更多=有武器

struct {

int wp\_aim[10] = { -1,-1,-1,-1,-1,-1,-1,-1,-1,-1 };//瞄准哪里

}shipaim[20];

int wp\_aim\_pl[10] = { -1,-1,-1,-1,-1,-1,-1,-1,-1,-1 };//瞄准玩家哪里

int wp\_c[10] = { 0,0,0,0,0,0,0,0,0,0 }; //充能状态

int wp\_chg\_ct[10]= { 0,0,0,0,0,0,0,0,0,0 };//充能计数

int sector[16] = { 0 }; //区块,表示是否有区块,0=无，1=有

int secinteg[16] = { -1,-1,-1,-1,-1,-1,-1,-1,-1,-1, -1,-1,-1,-1,-1,-1 }; //区块生命值

int secarmor[16] = { -1,-1,-1,-1,-1,-1,-1,-1,-1,-1, -1,-1,-1,-1,-1,-1 };

int secinteg\_r[16] = { -1,-1,-1,-1,-1,-1,-1,-1,-1,-1, -1,-1,-1,-1,-1,-1 }; //实时区块生命值

int secarmor\_r[16] = { -1,-1,-1,-1,-1,-1,-1,-1,-1,-1, -1,-1,-1,-1,-1,-1 };

struct seclocs {

int secloc[4] = { 0 };

} seclocs[16] ; //小号的区块坐标，大号的区块坐标

int AItype=0;

int aim\_ship0 = 0;//瞄准了玩家 0=未瞄准、1=瞄准了

int eng\_ecost = 0;//引擎能量消耗

int shd\_ecost = 0;//护盾能量消耗

int arm\_ecost = 0;};

shipentity ship\_0;

shipentity ship[30], ship\_in\_dat[20],\*pp;

shipentity类设置两个结构体数组和一个单独的结构体。ship[30]负责接受、读取自文件的每一种舰船的信息，数值是静态的。ship\_in\_dat是场地中的舰船，其值是动态的。单独的结构体代表玩家的舰船，\*pp是读取文件时用的指针。

2.3 武器数据

定义了stdwpgeneral结构体，存储所有与武器相关的数据。

struct stdwpgeneral

{

//…定义过于冗长，在此省略

}；

stdwpgeneral类同样包括静态部分和动态部分，类似shipentity。

2.4 图像数据

图像源文件存储于程序根目录下，由getimage()函数读取。

专门设置了含有图像的结构体，动态取值，存储场内所有舰船/武器的图像资源，方便格式化绘制图片。

struct ship\_in

{

//…定义过于冗长，在此省略

}ship\_in\_pic[20],ship\_0\_pic;

struct wp\_in { //场内的武器图像

PIMAGE wp\_static; //贴图

PIMAGE anim[30];//动画

PIMAGE proj;//抛射物

PIMAGE proj\_virtual; //抛射物在左中、左上屏幕

};

2.5 其他数据

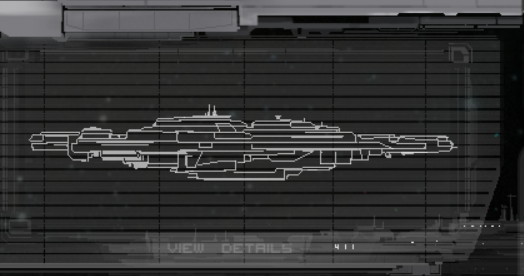
如雷达上物体的坐标等，同样使用结构体进行存储。

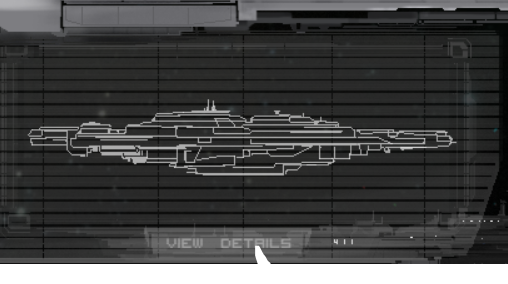
3.输入检测与状态显示

3.1 介绍

输入探测是整个程序必不可少的部分。本节将叙述程序如何检测输入信息，以及怎么实现根据输入的信息更改显示状态。

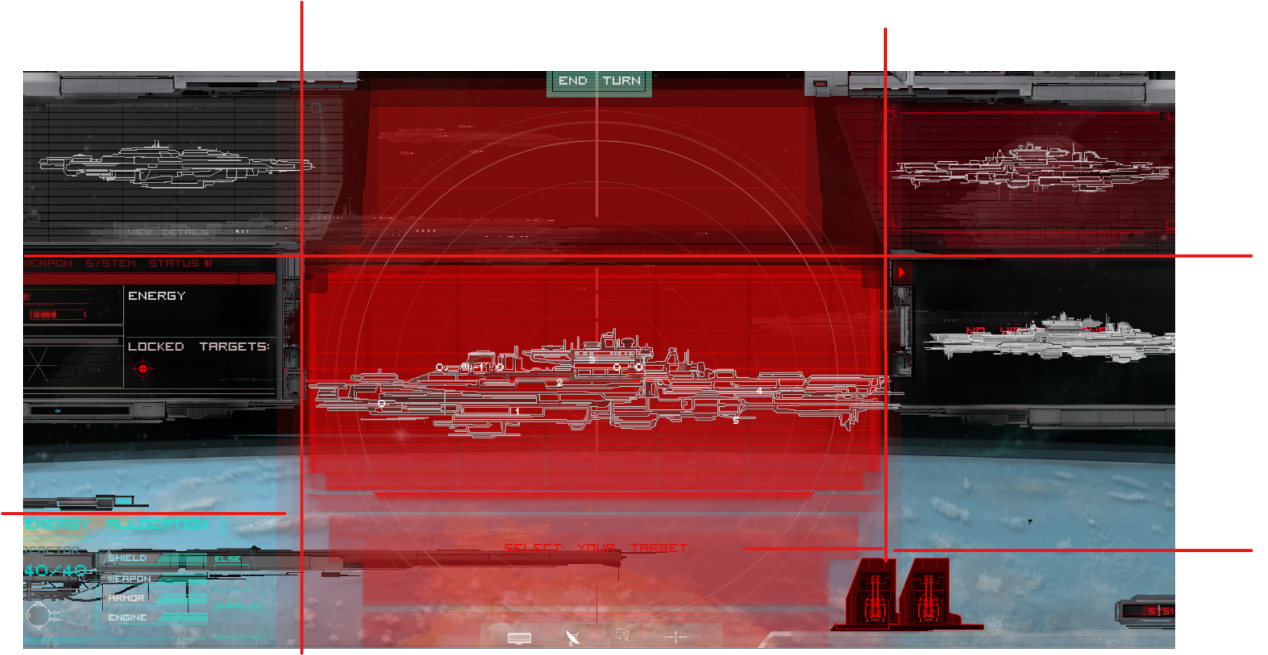
示例：





可以见到，鼠标移到“VIEW DETAILS”上之后，该按钮被高亮了

3.2 交互界面设计



屏幕大致可以分为7个部分（如图中红线所示）

左上显示玩家舰船的实时状态，右上显示敌舰的实时状态，左中显示玩家舰船子系统的实时状态，右中显示对敌舰的情报，左下显示当前的能量状态，右下显示当前的武器状态，正中央的界面可以切换，显示敌舰的状态，或者雷达图像，下方的四个按钮负责此功能。

3.3 相关函数

包括4个与按钮相关的函数：button, button\_c, button\_r, revbutton。

用途分别为检测鼠标是否指向指定区域，是否左键点击，是否右键点击，是否点击了指定区域以外的部分。

最重要的函数是mousecheck函数，集成了所有鼠标检测功能。其主体结构是一个while循环，条件是当检测到鼠标信息时。第一行代码mesg\_1=mousemsg();的目的是更新存储鼠标信息的结构体，使鼠标的位置为实时信息。

随后均是检测部分。格式为

if (#特定前提条件) //有些情况下部分按钮要失效

if(按钮函数检测到点击)

UIFALG.[某个状态位]= 值被修改。

所有更改图像的功能均放置于主函数中，因为仅有在主函数中才能进行图形操作。

通过switch/if ，根据状态位的值，放置对应图像。

举例：

if (UIFLAG.p\_l == 0) //左上按钮没有被指向

putimage\_withalpha(NULL, ul\_bt, 0, 0);

if (UIFLAG.p\_l == 1) { //左上按钮被指向了

putimage\_withalpha(NULL, ul\_bt, 0, 0);

putimage\_withalpha(NULL, ul\_bt, 0, 0);//高亮图像

}

3.4 函数输入/输出

4个按钮函数的输入均为特定区域的边界坐标，输出为0（未检测到点击）或1（检测到点击）

Mousecheck 直接对结构体中数据进行操作，没有输入/输出。

3.5 函数定义

int button(int dwn, int up, int lft, int rht) { //在循环内使用的按钮函数,仅供检测

if ( mesg\_1.x > lft && mesg\_1.x<rht && mesg\_1.y>dwn && mesg\_1.y < up)

{

return 1;

}

else return 0;

return 0;

}

int button\_c(int dwn, int up, int lft, int rht) { //在循环内使用的按钮函数,供检测和点击

if (mesg\_1.is\_left() && mesg\_1.is\_up() && mesg\_1.x > lft && mesg\_1.x<rht && mesg\_1.y>dwn && mesg\_1.y < up)

{

return 1;

}

else return 0;

return 0;

}

int button\_r(int dwn, int up, int lft, int rht) { //在循环内使用的按钮函数,供检测和右键点击

if (mesg\_1.is\_right() && mesg\_1.is\_up() && mesg\_1.x > lft && mesg\_1.x<rht && mesg\_1.y>dwn && mesg\_1.y < up)

{

return 1;

}

else return 0;

return 0;

}

int revbutton(int dwn, int up, int lft, int rht) { //反向按钮

if ( mesg\_1.is\_down() &&(mesg\_1.x < lft || mesg\_1.x>rht || mesg\_1.y<dwn || mesg\_1.y > up))

{

return 1;

}

else return 0;

}

Mousecheck函数代码超过300行，由于功能已经介绍清楚，在此不做引用。

4.文件存储/读取

4.1 介绍

·· 市面上的游戏大多把设定的数值存在程序之外，本项目也是如此。舰船数据和武器数据分别存储于resources\\data下的shipentity.txt和stdwp.txt。对话的文本存储于dialogue\\dialogue\_lev0.txt

4.2 相关函数

基于fgets与fscanf，格式化读取数据，并直接送到对应存储数据的结构体中

对于舰船、武器数据，由于设置的读取循环的条件是文件尚未终止，所以可以通过在文件中依照格式添加条目的方式，很方便地在游戏中添加新的舰船/武器。

4.3 函数定义

void loaddata() {

//加载文字

FILE\* dia\_fp; //对话文件指针

dia\_fp = fopen("resource\\dialogue\\dialogue\_lev0.txt", "r");

fgets(lev0strs[0].lev0\_str1, 40, dia\_fp);

fclose(dia\_fp);

//加载数据

FILE\* data\_fp; //数据文件指针

data\_fp = fopen("resource\\data\\shipentity.txt", "r");

for (int i = 0; !feof(data\_fp); i++, pp++) { //持续读取

fscanf(data\_fp, "%s %d %d %d %d ", pp->nam, &pp->energy, &pp->englv[0], &pp->mov, &pp->t);

for (int i = 0; i <= 15; i++)//加载区块位置 2\*16

fscanf(data\_fp, "%d %d %d %d", &pp->seclocs[i].secloc[0], &pp->seclocs[i].secloc[1], &pp->seclocs[i].secloc[2], &pp->seclocs[i].secloc[3]); //infx、infy、supx、supy

fscanf(data\_fp, "%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d", &pp->sector[0], &pp->sector[1], &pp->sector[2], &pp->sector[3], &pp->sector[4], &pp->sector[5], &pp->sector[6], &pp->sector[7], &pp->sector[8], &pp->sector[9], &pp->sector[10], &pp->sector[11], &pp->sector[12], &pp->sector[13], &pp->sector[14], &pp->sector[15]); //区块信息

fscanf(data\_fp, "%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d", &pp->secinteg[0], &pp->secinteg[1], &pp->secinteg[2], &pp->secinteg[3], &pp->secinteg[4], &pp->secinteg[5], &pp->secinteg[6], &pp->secinteg[7], &pp->secinteg[8], &pp->secinteg[9], &pp->secinteg[10], &pp->secinteg[11], &pp->secinteg[12], &pp->secinteg[13], &pp->secinteg[14], &pp->secinteg[15]); //区块生命值

fscanf(data\_fp, "%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d", &pp->secarmor[0], &pp->secarmor[1], &pp->secarmor[2], &pp->secarmor[3], &pp->secarmor[4], &pp->secarmor[5], &pp->secarmor[6], &pp->secarmor[7], &pp->secarmor[8], &pp->secarmor[9], &pp->secarmor[10], &pp->secarmor[11], &pp->secarmor[12], &pp->secarmor[13], &pp->secarmor[14], &pp->secarmor[15]); //区块装甲值

fscanf(data\_fp, "%d %d %d %d %d %d %d %d %d %d ", &pp->wp\_[0], &pp->wp\_[1], &pp->wp\_[2], &pp->wp\_[3], &pp->wp\_[4], &pp->wp\_[5], &pp->wp\_[6], &pp->wp\_[7], &pp->wp\_[8], &pp->wp\_[9]); //武器状态

fscanf(data\_fp, "%d %d %d %d %d %d %d %d %d %d ", &pp->wp\_c[0], &pp->wp\_c[1], &pp->wp\_c[2], &pp->wp\_c[3], &pp->wp\_c[4], &pp->wp\_c[5], &pp->wp\_c[6], &pp->wp\_c[7], &pp->wp\_c[8], &pp->wp\_c[9]); //武器状态

}

fclose(data\_fp); pp = 0;

data\_fp = fopen("resource\\data\\stdwp.txt", "r");

for (int i = 1; !feof(data\_fp); i++) {

fscanf(data\_fp, "%s %d %d %f %d %d", &stdwp[i].nam, &stdwp[i].ecost, &stdwp[i].charge, &stdwp[i].range, &stdwp[i].damage, &stdwp[i].damage\_integ);

fscanf(data\_fp, "%s %d %d %d %d %d", &stdwp[i].is\_blast, &stdwp[i].integ, &stdwp[i].time, &stdwp[i].track, &stdwp[i].type, &stdwp[i].blast\_range);

fscanf(data\_fp, "%d %d %d %d %d", &stdwp[i].price, &stdwp[i].rare, &stdwp[i].ani\_time/\*UNIT: FPS\*/, &stdwp[i].ani\_fps, &stdwp[i].proj\_spd);

}

fclose(data\_fp);

return;

}

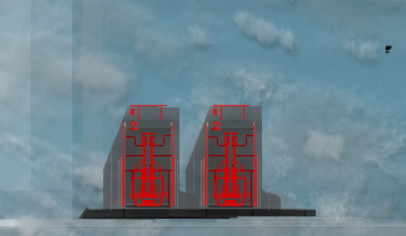
5.战斗机制

5.1 介绍

·· 本部分介绍游戏的玩法以及背后的实现原理，包括瞄准/攻击，移动，伤害结算、能量几个子部分。

5.2 瞄准/攻击

第一步：为右下角的武器充能



第二步：

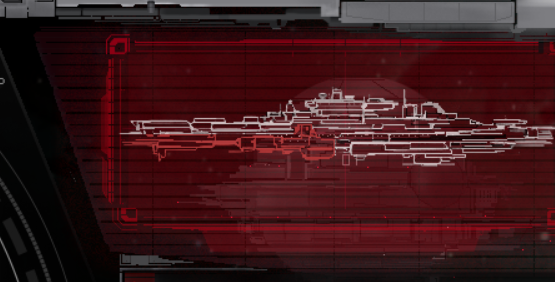


充能数归零后，则充能完毕，可以开火。



切换到瞄准界面，点击武器，可以瞄准目标

第三步：



点击上方的“END TURN”按钮，则结束回合，开火，敌舰被击中的区块受损

区块结构值归零时，该区块即被摧毁。摧毁的区块达到一定数目，或摧毁核心区块，敌舰就会爆炸。

5.3 AI的操作

采用随机函数rand(); AI的每个武器都会随机瞄准一个玩家区块。

5.4 移动

点击MOVE按钮后，会进入移动模式，鼠标点击雷达屏上的任意位置，就会

让玩家舰船在下一回合移动到该位置。

5.5 代码实现

5.5.1 瞄准：

本质上每一个区块都是一个按钮。鼠标移动/点击到该按钮上的判断在mousecheck函数中。Mousecheck会修改UIFLAG.sec\_slected状态位和p\_sec状态位，实现点击瞄准该区块后改变颜色。

5.5.2 攻击:

在主函数末尾，设置了检测函数。一旦所有的动画都进行完毕，则结算伤害。在某舰船有区块被攻击的条件满足后，程序将修改被攻击舰船对应结构体的数据，使该结构体中存储的实时区块生命值的成员减去对应武器的伤害值。

假如舰船有护盾，则优先对护盾造成伤害，护盾清零后再损失装甲，装甲清零后则伤害结构。

1. **测试（测试）**