程序编写提示

0 写在前面的话

程序总体仍使用面向过程的方式构造，各玩家、卡牌等作为全局变量在 item.cpp 中声明

card_inf 为长度 160 的 card_t 数组，用于储存卡牌信息，其中除 owner 为游戏进行中拥有者，其余均为固定属性
general_inf 为 general_t 数组，储存武将信息，具体函数中武将技能触发通过调用 id 判定
game 储存游戏运行数据，如牌堆、当前进行回合等

本程序中“空”“无”的概念常用**-1** 而非 0 表示，因为 0 会作为多个对象的下标

1 图形界面与交互的实现

注：本条所述黑色 RGB 为**(0, 0, 0)**

1.1 按钮点击的判定

```
for(; is_run(); delay_fps(100))
{
    //如下 2 行为检测鼠标事件并获取位置,所用变量在 gui.cpp 声明
    while (mousemsg()) msg = getmouse();
    mousepos(&mouse_x, &mouse_y);

    if(msg.is_down() && mouse_x >= x1 && mouse_x <= x2 && mouse_y >= y1 && mouse_y <= y2)
    {
        //若点击位置在(x1, y1)与(x2, y2)对应矩形范围内，执行语句
    }
    if(msg.is_down() && mouse_x >= x3 && mouse_x <= x4 && mouse_y >= y3 && mouse_y <= y4)
    {
        break;  //此结构通过 break 结束判断
    }
}
```

使用此方法有时会出现鼠标连续判定的情况，如导致回合内取消某牌或技能使用后直接结束回合的情况，需要通过 **goto** 或 **longjmp**(相当于跨文件 goto)而不是 break 退出循环

1.2 图层的构建与使用

item.cpp 中声明有多个 PIMAGE 结构体作为图层，图层建立后通过如下方式初始化：

```
setbkcolor(BLACK);    //设置背景色(默认为黑色)，若背景为黑则无须添加此语句
getimage(gui.background, 0, 0, 1200, 600);    //截取当前显示的内容,此次为整个界面
setbkmode(TRANSPARENT, gui.background);  //设置添加文字背景透明
```

在函数中临时声明的图层，也可使用 **getimage(temp, (char*)".\\Textures\\picture.png");** ，此时图层大小与内容设置为该图片

注意，图片路径中的字符串（包括拼接函数中的小段字符串）需要强转为**(char*)**

将内容绘制到 **NULL** 上，可在界面中显示
实际运行中先将内容绘制到各图层上，再通过 **DrawGui()**将各图层依次绘制在 NULL 以显示

1.3 图形的绘制

大部分图形通过基本绘制函数实现，但常用功能在 app.h 中进行了封装
图形界面使用函数如下：
char* Myitoa(int num); 将数字转换为对应字符串
事实上，C 语言存在转换函数 itoa，但此函数使得返回值为一个 char*型
也就是说，该函数可直接作为一个字符串（如在 Link）使用
但由于字符串信息需要设置为 static 储存，在一个语句中重复调用会导致信息丢失
（如： **Link(Myitoa(1), Myitoa(2));** 会返回字符串"22"而不是"12"），故另构造功能完全相同的函数 **MyitoaII**
char* Link(char* str1, char* str2); 连接字符串
类似地，同一语句多次调用 Link 应使用嵌套而非并列
（如：**Link(Link(Link(str1, str2), str3), str4);** 不能写作 **Link(Link(str1, str2), Link(str3, str4));**）

void Rect(int left, int top, int right, int bottom, color_t color, PIMAGE img);
画矩形线框
void Tri(int x1, int y1, int x2, int y2, int x3, int y3, color_t color, PIMAGE img);
画三角形线框
void LineRect(int left, int top, int right, int bottom, color_t color, PIMAGE img = NULL, int wide = 3);
画矩形外框，直线粗细为 wide（向外扩展）
void PasteImage(char* path, int x, int y, PIMAGE img = NULL, int mode = 0, color_t color = BLACK);
将图片粘贴到指定图层，若不透明粘贴，则只需填写前 4 个参数
若设置透明色（默认为黑色），mode 为 **TRANSPARENT**
void Pastecard(int x, int y, int id, PIMAGE img);
将卡牌图片（包括花色、点数）输出到图层上，id 为 card_inf 中的下标
void Putcard(int id);
将卡牌图片输出到 gui.throwcard 图层上，位置、角度随机，形成牌堆的效果

每执行一个操作后记得 **DrawGui()**刷新界面

1.4 文字居中显示

本程序采用如下方式实现：

```
char str[121] = "";   //声明字符串,可拼接
setcolor(color, image);   //设置某一图层的文字颜色
setfont(20, 0, "隶书", image);      //设置某一图层的字号
```

outtextxy(x - strlen(str) * 5, y, str, image);　//其中 x 为中心坐标，y 为顶部坐标
其中横坐标通过如下方式确定:（以下设字号为 size）
汉字为 size*size，占 2 个 char
字母与数字为 0.5size*size，占 1 个 char
由此，可得每个 char 所占长度为 0.5size，总长度为 0.5size·strlen(str)，文字左端与中心位置相差 0.25size·strlen(str)

2 游戏功能的实现

注：本条中定义**角色**为任何一个 player_t（储存在数组 player 中），**玩家**为人所操控的角色

2.1 函数中角色的调用

每个 player_t 中的 **id** 属性对应其在 **player 中的下标**
在具体行为的函数（如摸牌、弃牌）中涉及的玩家使用 **executor**（作为行为的实施者）与 **recipient**（一般作为行为的接受者，或是执行弃牌的角色）表示，在调用此类函数时注意传入的实参为 player_t 的地址

若需要引用某特定 id 的角色，写为 **player[id]**
当引用**某角色上下家**时，使用 **id 号±1** 的形式，如:
player[(executor->id + 1) % 4]　表示 executor 的下家
注意，计算下标时需要将最后结果限制在[0, 3]之间，特别是上家（此处应写类似于 **player[(executor->id + 5) % 4]**　的形式，因为 n % 4 在 n 取负时小于 0）

引用**玩家**时 id 为 **game.humanid**，**当前回合角色** id 为 **game.active**

2.2 游戏总体的运行

game 中，使用 **game.active** 与 **game.period** 表示当前进行回合的角色及当前阶段
每个回合开始时判断存活情况与翻面情况，此后若进行回合，则依次执行 6 个阶段，伴随 game.period 从 0 至 5 的变化

2.3 卡牌与牌堆机制

每名角色卡牌有 3 个区域：手牌，装备区，判定区
手牌为 int 数组 **card**，装备区为 int 数组 **equips**（**0~3 分别对应武器、防具、-1 马、+1 马**）
判定区为二维数组 judges，其中每行第一个元素为使用卡牌 id，第二个元素为类型 id，两元素的值通常相同，但存在特例，如:
【方片 A 诸葛连弩】被当做【乐不思蜀】使用，放置于某角色判定区的第 1 张，则该角色（recipient）有：　**recipient->judge[0][0] = 120**　而　**recipient->judge[0][1] = 160**（LE）

当判断某牌类型时使用以下方式:
executor->card[0] == SHA　或　**(int)executor->card[0] >= 0x90 && (int)executor->card[0] >= 0x10**

其中第二种将卡牌类型与数值比较，需要强转为 int（通常用于贴图或大类的判断）
另外，在 Execard 中，type 为 **int** 型

game.card 为长度为 160 的 int 数组，储存牌堆**从上到下**卡牌的 id
当卡牌的拥有者发生变化时，注意**先执行“添加”与“输出”操作，再执行“减少”操作**，如 Drawcard 中

```
recipient->card[recipient->cardamount] = game.card[0];
game.card[0] = -1;
```

的顺序不能互换

弃牌及获得其他角色牌时，卡牌 id 可表示区域，具体表现为：
当此牌在手牌时，id 与 card 的下标相同
而装备区、判定区的 id 在下标基础上加 **0x100** 与 **0x200**，在后续的判断中通过 **switch(id >> 8)**来判断所在区域，**id | 0xFF** 获取 card_inf 中下标

2.4 信息传递

在实际执行过程中，很多操作存在特殊情况，如【火攻】需要弃置特定花色的牌，【贯石斧】不能弃置武器等
相比于另外写对应函数，此程序采用添加参数 **add** 等表示情形（程序并未使用 C++中 class 等概念，因此未采用继承等概念构造）

例如，在 Throwcard 中通过 add 以及其他参数限制：
mode 低 8 位表示允许弃置的花色、类型以及是否允许不弃置，函数中另外进行解算

```
int suit = mode & 15;    //若不对此进行限制，则全设为 1
int type = (mode & 112) >> 4;
int cancel = (mode & 128) >> 7;
```

而判断类型时则进行类似验证：

```
if (suit & (1 << (int)card_inf[recipient->card[i]].suit) &&    //suit 的第[花色]位若为 1 则符合条件
    (type & (1 << TypeIdentify(card_inf[recipient->card[i]].type))))
  accord++;
```

或是如下形式的判断：（此处为麒麟弓发动时，选择武器与防具无效并继续循环）

```
if(add == 0x50 && tothrow < 2) continue;
```

3 AI 算法

本程序中 AI 对出牌、弃牌等判断采用**贪心算法**，具体通过计算“权重”（或“优先级”）实现
如对弃自己手牌的思路如下：
从 0 开始对每张手牌遍历，根据场上形势赋给此牌不同‘权重’，如类型为【杀】时的计算：

```
if(card_inf[recipient->card[i]].type == SHA || card_inf[recipient->card[i]].type == LEISHA || card_inf[recipient->card[i]].type == HUOSHA)
{
    double temp[2] = {0, 0};    //对于指定单一目标且不限于自身的牌，一般分开考虑对不同角色的收益
```

```
    for(int j = 0; j <= 1; j++)
    {
        //考虑此牌是否可使用与有效，若是在出牌阶段此问题更为关键
        if(enemy[j].controller == DEAD) continue;
        /* 判断目标防具 */
        /* 判断目标距离与攻击范围 */
        //计算若使用此牌，得到的收益，大体采用“1 体力=2 牌”的近似
        /* 根据对方手牌计算有闪的期望 */
        //回合外降低此部分收益
        if(game.active != recipient->id || game.period > 3) profit *= 0.7;
    }
    //卡牌留在手中可响应其他角色的出牌，额外计算收益（数据较平均、合理即可），且与自身体力相关
    //用于响应决斗(3),南蛮入侵(3),借刀杀人(2)时视为 1 收益,当体力更低时提高(0.2 每损失体力)权重
    for(int j = 0; j <= 1; j++)   temp[i] += 0.125 + (recipient->maxhealth - recipient->health) * 0.2;
    //取较大者为理论收益
    profit = temp[0] > temp[1] ? temp[0] : temp[1];
}
```

最后，选出收益最大（或最小）的卡牌，传回其在 player.card 中的 id 以执行下一步

4 杂项

4.1 贴图文件夹

所有贴图放在文件夹.\Texture 中，其中卡牌、武将等图片均以**数字.png** 命名，这与相关的枚举相对应，在输出图像时一般进行字符串拼接

如选将后输出武将图片：

```
PasteImage(Link( (char*)".\\Textures\\Generals\\", Link( (char*)Myitoa(forselect[i]), (char*)".png")), 310 + 150 * i, 240, gui.selector);
```

4.2 坐标

窗口大小为 1200*600

gui.cpp 内定义 **pos[8]**表示武将显示框（从位于下方的玩家开始逆时针排列）的坐标，其中偶数下标表示 x 坐标，奇数坐标表示 y 坐标

游戏界面中常用内容的坐标图见**.\Textures\position.png**