L0: amgame

姓名：张灵毓　　**学号：**171240524

2019 年 3 月 5 日

摘要

实现了一个方形五子棋小游戏，空格开始游戏，上下左右键控制光标移动，空格下棋，规则和常见五子棋规则一致（终端中有提示输出），在游戏结束后可以按 R 键重新开始游戏。

0.1　实验过程

实现的方形五子棋小游戏是基于所给出的 game.c，自己所实现的大致分为如下几个部分：1、背景的绘画。2、光标的初始化及处理。3、键盘图形交互。4、规则逻辑的制定。

0.1.1　基础性操作

为了方便实现，做了如下基础操作，用二维数组来记录屏幕上每一个矩形的状态 (颜色)。

为了尽可能使小游戏在多种分辨率下都能有不错的表现，可以使原来在 game.h 中宏定义的 SIDE 变成一个全局变量，每次根据 $init\_screen$ 函数中得到的分辨率来选取一个合适的 SIDE 大小。

考虑了如何重新开始游戏，在游戏进程 while(1) 外再套一个 while(1) 循环，每次如果发现 R 键被按下（游戏结束后重新开始的命令）则跳出当前死循环，进入外层死循环，重新开始游戏。

0.1.2　屏幕的绘制

实际上在背景的绘制的时候主要解决了一个将屏幕分成若干个矩形，画出一个个格子的问题，其实我们只需要在 $draw\_rec$ 函数对每一个像素点进行绘制的时候，将每个像素点所对对应的矩形的最下和最右的地方渲染上不同的颜色，即可实现，代码如下：

```
for (int i = 0; i < w * h; i++) {
  pixels[i] = color;
  //lines
  if(i > (h-1)*w-1)
      pixels[i] = line;

  if((i+1)%w == 0)
      pixels[i] = line;
  //
}
```

在游戏过程中，每一次调用 *update_screen* 函数都将重画一帧，重画的时候直接遍历记录屏幕状态的二维数组，根据二维数组所记录的值来确定渲染什么颜色，遍历到光标所在的位置的时候，以光标结构体里的数据为准，这样就保证了光标不会因为绘制屏幕而被覆盖。

0.1.3 光标的初始化及处理

使用结构体来记录光标的状态 (坐标，闪烁时的颜色，以及代表的棋子)，初始化函数将两个光标的初始坐标设置为 (0,0)，并确定相对应的棋子颜色。

在处理光标的闪烁时，利用 uptime() 函数，每隔 0.5s 跟换当前光标的颜色 (所代表棋子颜色和棋盘颜色)，然后 *update_screen* 即可

```
//control the twinkling of cursor;
current = uptime();
if(current - last > 500){//every 500ms
    switch(turn){
        case 0: cur[turn].color = (cur[turn].color == WHITE ? GRAY : WHITE); break;//WHITE chess
        case 1: cur[turn].color = (cur[turn].color == BLACK ? GRAY : BLACK); break;//BLACK chess
    }
    last = current;
}
```

光标的移动在下面键盘图形交互中说明。

0.1.4 键盘图形交互

只需要处理六个 keys, 上下左右，space 和 R。在没按 space 开始游戏前，对其他按键均不响应，游戏开始后，上下左右按键移动光标的位置，需要注意的是要判断光标是否已经到达了屏幕边界，如果已经到达，则不在此方向上改变光标位置。游戏开始后，space 键表示在光标所在处下棋，需要特殊处理的是在已下有棋子的位置上不能下棋而是输出这里有棋子的提示。对于 R 键，加入一个新的变量来记录 R 是否在游戏结束后被按下，如果被按下则跳出当前游戏的死循环，重新开始游戏。

0.1.5 规则逻辑

按照五子棋的规则，只要是横竖或者斜线方向有连续五个相同颜色的棋子即为胜利。所以可以如下检测，每下一颗棋子就进行一次 check(), 以刚下的棋子为中心，分上下左右，左上左下，右上右下，八个方向分别检测。

```
//up,down,left,right,lup,rdown,rup,ldown
const int dx[8] = {0,0,-1,1,-1,1,1,-1};
const int dy[8] = {-1,1,0,0,-1,1,-1,1};
```

用一个数组记录八个方向有多少粒与刚下的棋子颜色相同的棋子，如果某一个方向可以计数到四颗，则说明已经胜利。或者上下（左右，左上右下，左下右上）所记录的

棋子数加起来大于等于四即可，因为刚下的这颗棋子算一颗。

0.2 实验思考

遇到的一个问题是，include 头文件"klib.h" 后发现 game.c 里面有 *read_key* 和 *draw_rec* 这两个函数和 klib 中 io.c 中两个函数重名，则会发生重定义现象，于是把 klib.h 中的声明注释并 make run 执行未出现异常。但是之后实现光标闪烁的时候调用了 uptime() 函数，再次 make run 的时候却报错：这两个函数发生重定义现象。最后发现前后区别的原因是因为调用了 uptime() 函数，uptime() 函数也在文件 io.c 中，所以在符号解析的时候，由于 uptime() 函数被调用，所以会去解析 io.c 文件，这时候就发现了对于上述两个函数有两个强符号定义的情况。