# Linux项目报告

# 项目名称

Shell脚本实现扫雷小程序

# 项目结构

1. 显示雷区
2. 创建游戏逻辑
3. 创建判断单元格是否可选的逻辑
4. 记录可用和已查明（已排雷）单元格的个数
5. 创建游戏结束逻辑

# 项目要素

在扫雷中，游戏界面是一个由 2D 数组（列和行）组成的不透明小方格。每一格下都有可能藏有地雷。玩家的任务就是找到那些不含雷的方格，并且在这一过程中，不能点到地雷。这个 shell 版本的扫雷使用 16x16 的矩阵，实际逻辑则由一个简单的 shell 数组来完成。

# 变量

score=0 # 会用来存放游戏分数

# 下面这些变量，用来随机生成可排除地雷的实际值

a="1 16 -16 -1"

b="-1 0 1"

c="0 1"

d="-1 0 1 -2 -3"

e="1 2 20 21 10 0 -10 -20 -23 -2 -1"

f="1 2 3 35 30 20 22 10 0 -10 -20 -25 -30 -35 -3 -2 -1"

g="1 4 6 9 10 15 20 25 30 -30 -24 -11 -10 -9 -8 -7"

#

接下来，我会用列（0-15）和行（a-j）显示出游戏界面，并且使用一个 16x16 矩阵作为雷区。（M[16][16] 是一个索引从 0-9255，有 256 个值的数组。）

printf '\n\n'

printf '%s' "     a   b   c   d   e   f   g   h   i   j"

printf '\n   %s\n' "-----------------------------------------"

**创建玩家逻辑**

玩家操作背后的逻辑在于，先从 stdin 中读取数据作为坐标，然后再找出对应位置实际包含的值。这里用到了 shell的参数扩展，来设法得到行列数。然后将代表列数的字母传给分支语句，从而得到其对应的列数。为了更好地理解这一过程，可以看看下面这段代码中，变量 o 所对应的值。 举个例子，玩家输入了 c3，这时 shell将其分成两个字符：c 和 3。为了简单起见，我跳过了如何处理无效输入的部分。

colm=${opt:0:1} # 得到第一个字符，一个字母

ro=${opt:1:1} # 得到第二个字符，一个整数

case $colm in

a ) o=1;; # 最后，通过字母得到对应列数。

b ) o=2;;

    c ) o=3;;

    d ) o=4;;

    e ) o=5;;

    f ) o=6;;

    g ) o=7;;

    h ) o=8;;

    i ) o=9;;

    j ) o=10;;

......

  Esac

**创建判断单元格是否可选的逻辑**

为了找到地雷，在将坐标转化，并找到实际位置之后，程序会检查这一单元格是否可选。如不可选，程序会显示一条警告信息，并要求玩家重新输入坐标。如果可选，则重置单元格对应的值，并更新分数。反之，因为其对应值不为点，则设置变量 not\_allowed

**记录已选择和可用单元格的个数**

这个程序需要记录游戏界面中哪些单元格是可选择的。否则，程序会一直让用户输入数据，即使所有单元格都被选中过。为了实现这一功能，我创建了一个叫 free\_fields 的变量，初始值为 0。用一个 for 循环，记录下游戏界面中可选择单元格的数量。 如果单元格所对应的值为点（.），则 free\_fields 加一。

get\_free\_fields()

{

  free\_fields=0

  for n in $(seq 1 ${#room[@]}); do

    if [[ "${room[$n]}" = "." ]]; then

      ((free\_fields+=1))

    fi

  done

}

# 运行结果

