实验1-棋盘覆盖

问题分析

- 对于这种规模比较庞大的问题，首先应该考虑到的是能不能把它转化成一个个易于解决的子问题。
- 对于一块棋盘，我们可以把它划分为四个部分，即左上角区域、右上角区域、左下角区域、右下角区域。然后根据特殊方格的坐标去判断该特殊方格在哪个区域。
- 首先考虑一个最简单的问题，即一个2*2大小的棋盘，若其中一个区域含有特殊方格，那么我们要做的就是用L型骨牌覆盖其他三个区域的方格。
- 问题回到更普遍的情况，考虑当前棋盘正中央的四个方格，用L型骨牌覆盖没有特殊方格的三个区域的所属方格。然后再去处理这四个区域的棋盘，这个问题又回到了原问题，因为经过划分处理后每个棋盘都是相同大小，且含有一个特殊方格的棋盘。
- 所以该问题可以用递归来进行解决。直到棋盘的宽度为1时代表递归的终点。

程序只需要用到一个函数

```
void fillboard(int x,int y,int tx,int ty,int size)
```

`(x,y)`为当前棋盘的左上角坐标(用于定位棋盘位置),`(tx,ty)`为特殊方格所在的坐标

`size` 为当前棋盘的宽度(宽度为1即为递归边界)

在函数中对四个区域进行判断，如果特殊坐标在该区域中，则该区域不再放牌，否则在区域的角落（棋盘中间四个方格中属于该区域的一个方格）放置骨牌

对位置的判断可用以下代码(`len=size/2`即分成四个区域后每个区域的宽度)

- `if(tx<x+len&&ty<y+len)` 特殊方格在左上角区域
- `if(tx<x+len&&ty>=y+len)` 特殊方格在右上角区域
- `if(tx>=x+len&&ty<y+len)` 特殊方格在左下角区域
- `if(tx>=x+len&&ty>=y+len)` 特殊方格在右下角区域

以右上角区域为例，右上角棋盘的左上角坐标为`(x,y+len)`

若特殊方格在该区域，则`fillboard()`函数中特殊坐标的参数对应的仍是`tx,ty`，若不在，则右上角区域应放骨牌的区域在该区域的左下角，即对应坐标`(x+len-1,y+len)`

对于其他区域的处理同理。

```
int len=size/2;//偏离距离
//覆盖右上角子棋盘
if(tx<x+len && ty>=y+len)
    fillboard(x,y+len,tx,ty,len);//特殊方格在此棋盘中
else //此棋盘中无特殊方格，用t号L型骨牌覆盖左下角
{
    board[x+len-1][y+len]=t;
    //覆盖其余方格
    fillboard(x,y+len,x+len-1,y+len,len);
}
```

图形界面

工具：Python Tkinter 画布

初始化画布

- 定义好覆盖棋盘用的颜色
- 设置好棋盘的左上角坐标这里为(200,200),定义格子宽度为50
- 生成画布，背景设置为白色，根据总格子数量，设置画布的宽度和高度

```
    colors = ['white', '#36ddd9', '#668B8B', '#7FFFD4', '#66CDAA',
'#458B74', '#C1FFC1', '#9BCD9B', '#00FF00',
              '#008B45', '#6495ED', '#483D8B',
              'orange', 'yellow', 'green', 'cyan', 'blue', 'pink',
'purple', 'red', '#6A5ACD', '#8470FF', '#0000CD',
              '#1E90FF', '#00BFFF', '#87CEFA', '#00CED1']
    startx = 200
    starty = 200
    cellwidth = 50
    root = Tk()
    canvas = Canvas(root, bg="white")
    canvas.pack()
    width = 2 * startx + len(board) * cellwidth
    height = 2 * starty + len(board) * cellwidth
    canvas.config(width=width, height=height)
```

用不同颜色的骨牌覆盖棋盘

遍历之前得到的board数组，根据预先处理好的颜色，给相应格子涂上颜色

格子的左上角坐标根据(i,j)偏移关系算出，右下角坐标即为左上角坐标分边加上格子宽度

```
index = board[i][j]
color = colors[index]
cellx = startx + i * 50
celly = starty + j * 50
canvas.create_rectangle(cellx, celly, cellx + cellwidth, celly +
cellwidth,fill=color, outline="black")
```

为了使棋盘覆盖的过程中，组成L型骨牌的三个格子被同时涂上颜色，所以当我们遍历到一个型号的格子，把其他另外两个同型号的格子一同给涂上颜色。而一个格子的相邻格子有八个，对这个八个格子的型号进行判断，型号一样的覆盖上相同颜色即可。

- 横坐标可取i-1,i,i+1
- 纵坐标可取j-1,j,j+1

这里需要注意边界，有些格子越过了棋盘的边界，用特判i-1>=0,i+1<len来排除这些边界情况。按此方法遍历完所有格子即可完成棋盘的颜色覆盖

```
board[i][j] = -1
for t1 in [(i - 1 if(i-1>=0)else i), i, (i + 1 if (i + 1 < len(board))
else i)]:
  for t2 in [(j - 1if(j-1>=0)else j), j, (j + 1 if (j + 1 < len(board))
else j)]:
    if board[t1][t2] == index:
      cellx = startx + t1 * 50
      celly = starty + t2 * 50
      canvas.create_rectangle(cellx, celly, cellx + cellwidth, celly +
cellwidth,fill=color, outline="black")
```

运行结果截图

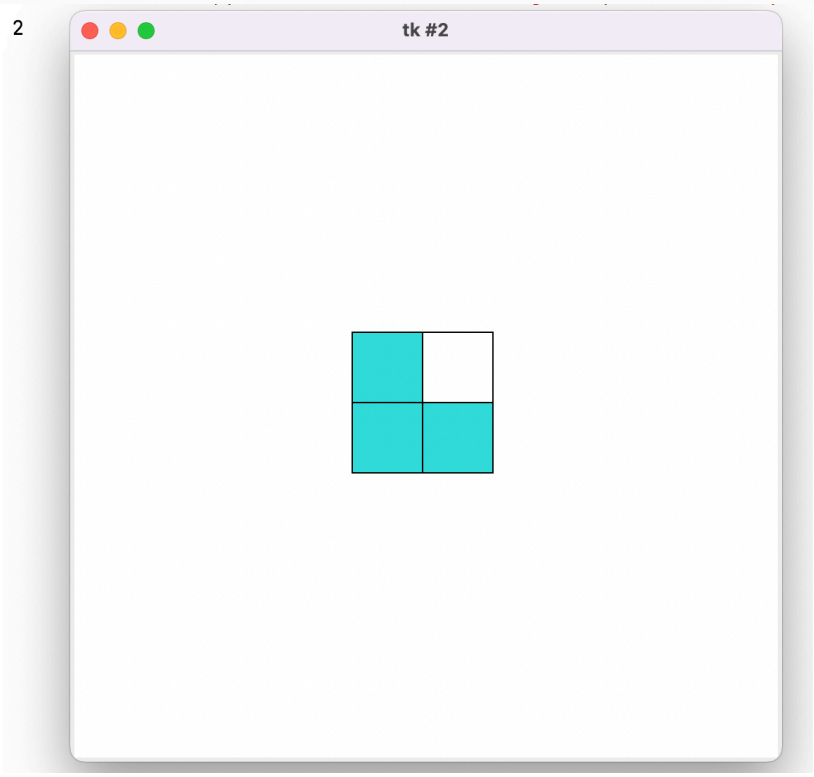
数字形式输出

```
3 4 3
Case 1: n=8
3   3   4   4   8   8   9   9
3   2   2   4   8   7   7   9
5   2   6   6   10  10  7   11
5   5   #   6   1   10  11  11
13  13  14  1   1   18  19  19
13  12  14  14  18  18  17  19
15  12  12  16  20  17  17  21
15  15  16  16  20  20  21  21
4 7 9
Case 2: n=16
4   4   5   5   9   9   10  10  25  25  26  26  30  30  31  31
4   3   3   5   9   8   8   10  25  24  24  26  30  29  29  31
6   3   7   7   11  11  8   12  27  24  28  28  32  32  29  33
6   6   7   2   2   11  12  12  27  27  28  23  23  32  33  33
14  14  15  2   19  19  20  20  35  35  36  36  23  40  41  41
14  13  15  15  19  18  18  20  35  34  34  36  40  40  39  41
16  13  13  17  21  18  22  22  #   37  34  38  42  39  39  43
16  16  17  17  21  21  22  1   37  37  38  38  42  42  43  43
46  46  47  47  51  51  52  1   1   67  68  68  72  72  73  73
46  45  45  47  51  50  52  52  67  67  66  68  72  71  71  73
48  45  49  49  53  50  50  54  69  66  66  70  74  74  71  75
48  48  49  44  53  53  54  54  69  69  70  70  65  74  75  75
56  56  57  44  44  61  62  62  77  77  78  65  65  82  83  83
56  55  57  57  61  61  60  62  77  76  78  78  82  82  81  83
58  55  55  59  63  60  60  64  79  76  76  80  84  81  81  85
58  58  59  59  63  63  64  64  79  79  80  80  84  84  85  85
```

```
4 3 4
Case 3: n=16
4   4   5   5   9   9   10  10  25  25  26  26  30  30  31  31
4   3   3   5   9   8   8   10  25  24  24  26  30  29  29  31
6   3   7   #   11  11  8   12  27  24  28  28  32  32  29  33
6   6   7   7   2   11  12  12  27  27  28  23  23  32  33  33
14  14  15  2   2   19  20  20  35  35  36  36  23  40  41  41
14  13  15  15  19  19  18  20  35  34  34  36  40  40  39  41
16  13  13  17  21  18  18  22  37  37  34  38  42  39  39  43
16  16  17  17  21  21  22  22  1   37  38  38  42  42  43  43
46  46  47  47  51  51  52  1   1   67  68  68  72  72  73  73
46  45  45  47  51  50  52  52  67  67  66  68  72  71  71  73
48  45  49  49  53  50  50  54  69  66  66  70  74  74  71  75
48  48  49  44  53  53  54  54  69  69  70  70  65  74  75  75
56  56  57  44  44  61  62  62  77  77  78  65  65  82  83  83
56  55  57  57  61  61  60  62  77  76  78  78  82  82  81  83
58  55  55  59  63  60  60  64  79  76  76  80  84  81  81  85
58  58  59  59  63  63  64  64  79  79  80  80  84  84  85  85
3 5 7
Case 4: n=8
3   3   4   4   8   8   9   9
3   2   2   4   8   7   7   9
5   2   6   6   10  10  7   11
5   5   6   1   1   10  11  11
13  13  14  1   18  18  #   19
13  12  14  14  18  17  19  19
15  12  12  16  20  17  17  21
15  15  16  16  20  20  21  21
```

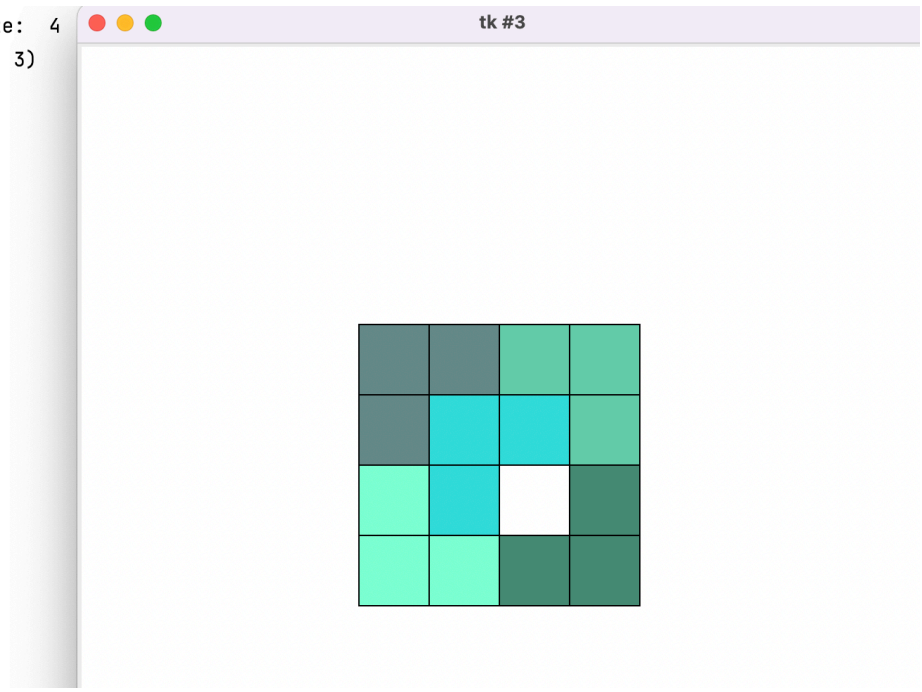
图形化输出

```
size:  2
(1, 2)
```



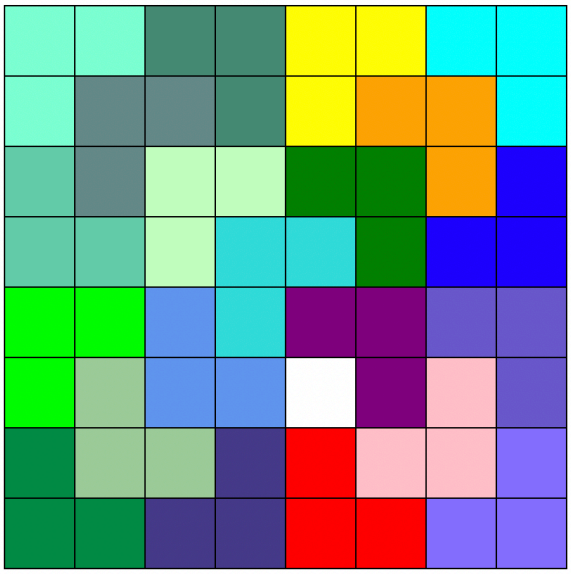
- k=1,x=1,y=2

```
size:  4
(3, 3)
```



- k=2,x=3,y=3

```
size:  8
(6, 5)
```



- k=3,x=6,y=5

设计调试中的问题

- 首次写递归程序时，因忘记写递归终点的判断，导致程序陷入死循环，在函数开头加上`if(size==1) return;`即可
- 在图形化界面的程序中，边界没有处理好，导致画出来的棋盘有多余的格子，分析程序后发现有部分坐标出现越界，造成了结果的错误输出。加上坐标范围的特判即可

实验体会

这是算法这门课程的第一次实验，整体来看比较简单。这次实验让我再一次体会到了递归与分治这一策略的巧妙性，它让看似非常庞大繁琐的问题变得易于处理，其关键就在于对问题的划分进而让我们面对的是一个简单的小规模问题。因此，在以后对程序设计题目的求解过程中，条件允许的话，我们要去用递归与分治的策略，进而巧妙地去解决问题。图形化界面的制作让覆盖的结果更加美观，看着一个个不同颜色的骨牌呈现在屏幕上，内心也感到满足。这次实验让我受益匪浅，通过这次实验，我加深了对递归策略的理解，并学会了怎么用Python 的Tkinter 画布去做一些简单的图形处理。