# 2048游戏

## 《基于Linux平台python OOP的项目案例》

- 机构名称：西部开源技术中心
- 课程方向: python全栈开发与大数据分析

# 目录

CONTENTS

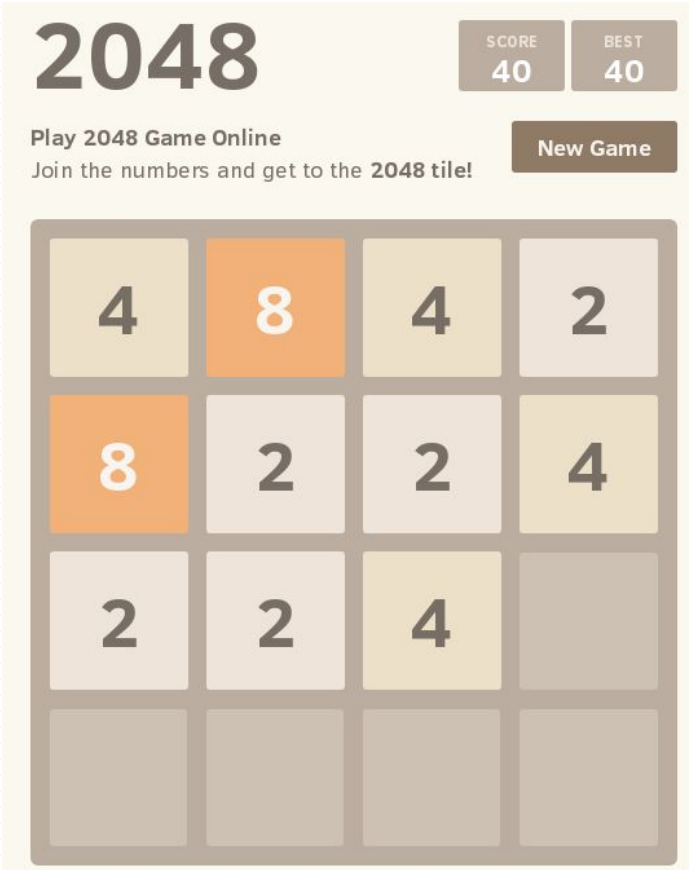
# 01 选题背景

# 01 选题背景

**官方2048网址：** http://2048game.com/

**游戏规则：**

20岁的Gabriele Cirulli开发的一款数字游戏。初衷就是觉得好玩，在将其开源版本放到Github上后，意外走红。这款游戏的玩法很简单，每次可以选择上下左右滑动，每滑动一次，所有的数字方块都会往滑动的方向靠拢，系统也会在空白的地方乱数出现一个数字方块，相同数字的方块在靠拢、相撞时会相加。不断的叠加最终拼凑出2048这个数字就算成功。



# 01 选题背景


- **游戏规则1**

  一开始方格内会出现2或者4等这两个小数字;玩家只需要上下左右其中一个方向来移动出现的数字，所有的数字就会想滑动的方向靠拢，而滑出的空白方块就会随机出现一个数字;

- **游戏规则2**

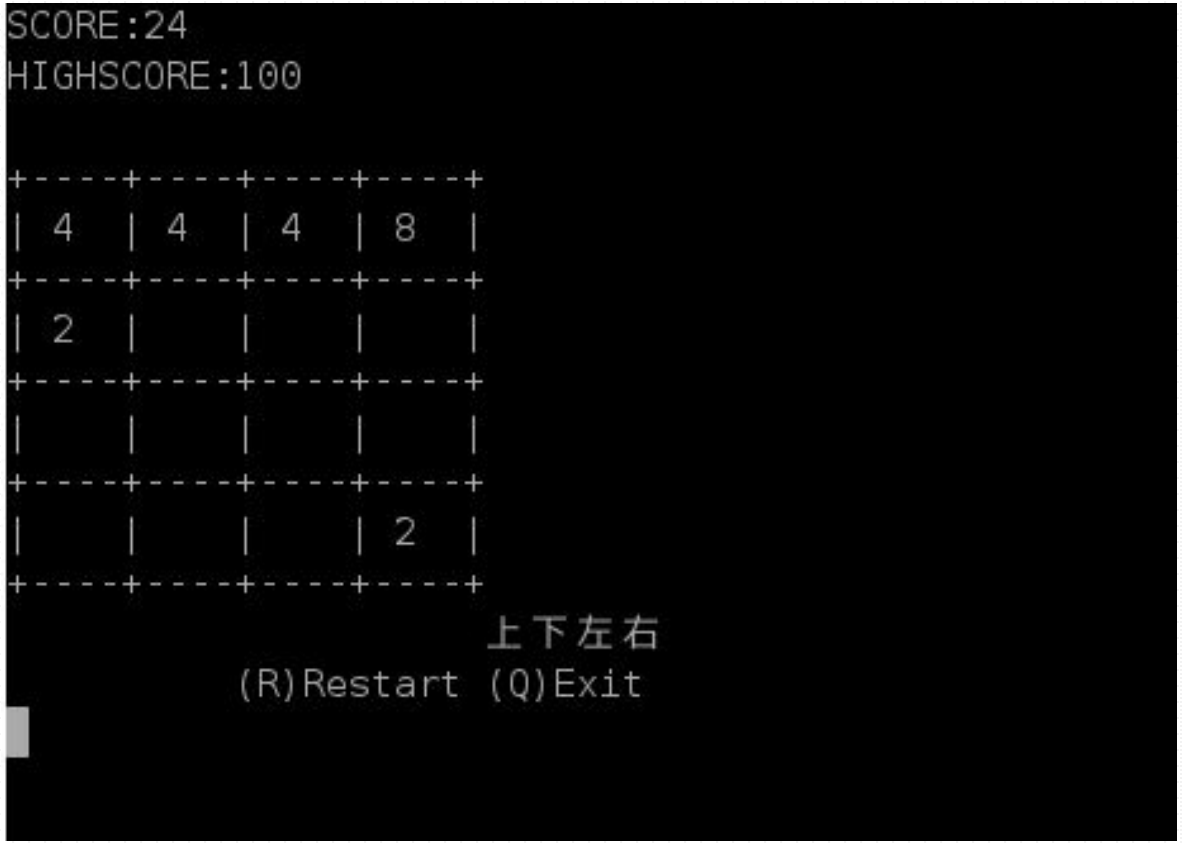
  相同的数字相撞时会叠加靠拢，然后一直这样，不断的叠加最终拼凑出2048这个数字就算成功。

- **游戏规则3**

  如果向上向下向左向右均不可移动， Game Over!
  否则游戏继续执行!

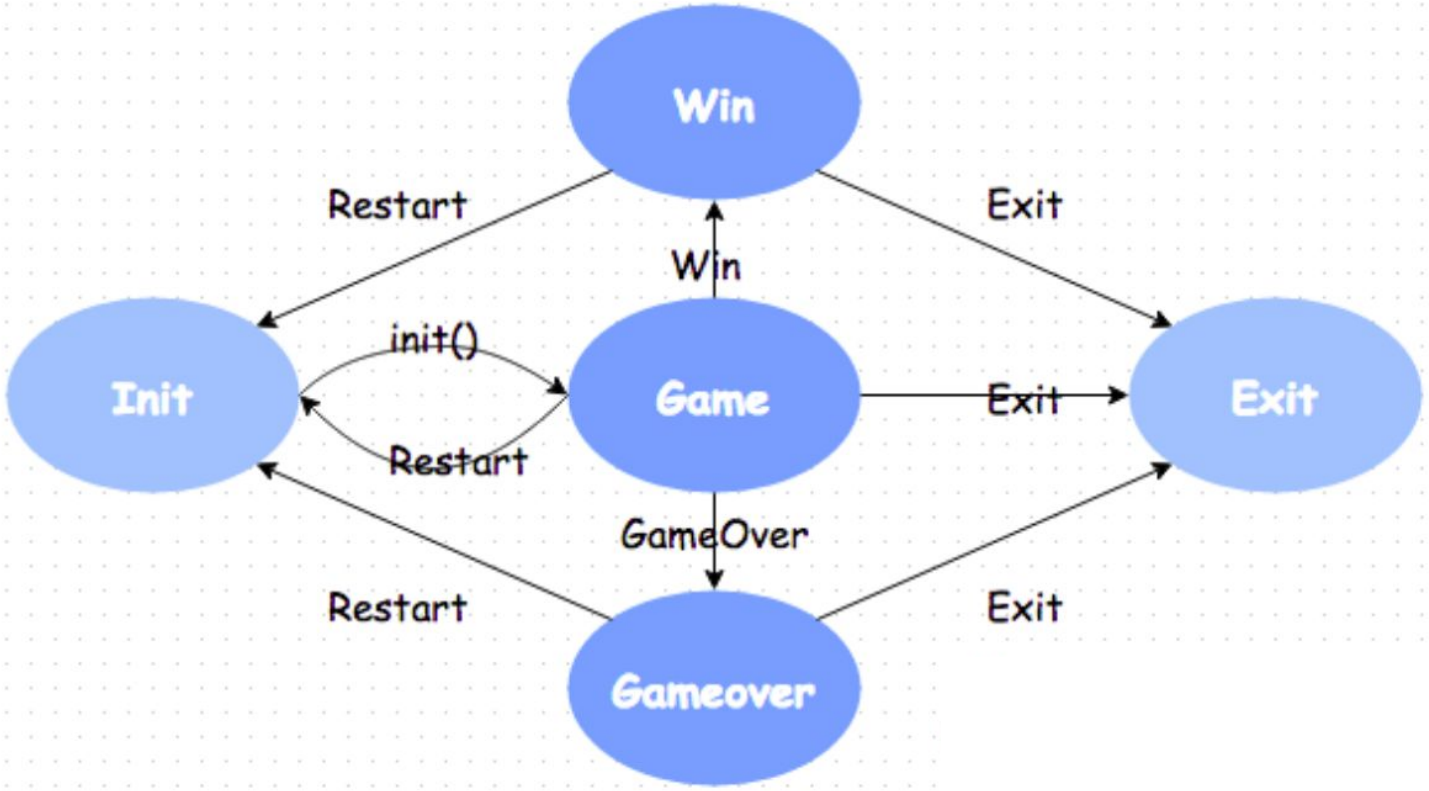
# 02 效果展示

# 02 效果展示



# 03 思路分析

# 03 思路分析-主程序




主程序: 状态机会不断循环,直到达到 Exit 终结状态结束程序。

参考资料：实验楼

# 03 思路分析-主程序

```
state_actions = {
    'Init': init,
    'Win': lambda: not_game('Win'),
    'GameOver': lambda: not_game('Gameover'),
    'Game': game
}

game_field = GameField(win_value=16)
state = 'Init'

while state != 'Exit':
    state = state_actions[state]()
```

主程序: 状态机会不断循环,直到达到 Exit 终结状态结束程序。

# 03 思路分析-获取用户操作

```
def get_user_action (stdscr):
    action = stdscr.getch()

    if action == curses.KEY_UP:
        return  'Up'
    elif action == curses.KEY_DOWN:
        return 'Down'
    elif action == curses.KEY_LEFT:
        return 'Left'
    elif action == curses.KEY_RIGHT:
        return 'Right'
    elif action == ord('r'):
        return 'Restart'
    elif action == ord('q'):
        return 'Exit'
```

利用cursor模块中的getch()方法。

# 04 难点剖析

# 04 思路分析-绘制棋盘

```
from random import choice, randint

width = 4
height = 4

field = [[0 for i in range(width)] for j in range(height)]


def random_create():
    """初始化棋盘时， 在随机位置生成2或者4， 2的可能性大， 4的可能性少"""
    # 可能出现的问题：随机生成的i,j位置原本已经有值。 解决方法：
    while True:
        i, j = choice(range(width)), choice(range(height))
        if field[i][j] == 0:
            field[i][j] = 4 if randint(1, 100) > 80 else 2
            break


random_create()
random_create()
print(field)
```

# 04 思路分析-绘制棋盘

```
def draw_sep():
    line = '+' + '----+' * width
    print(line)
def draw_row(row):  # [2,0,2,0]
    draw_one_row = "".join(['|{:^4}'.format(num)
                            if num!=0  else '|    ' for num in row])+'|'

    print(draw_one_row)

def draw_field():
    for row in field:
        draw_sep()
        draw_row(row)
    draw_sep()

draw_field()
```

# 04 思路分析-绘制棋盘

效果显示:

```
[[0, 0, 0, 0], [0, 0, 0, 2], [2, 0, 0, 0], [0, 0, 0, 0]]
+----+----+----+----+
|    |    |    |    |
+----+----+----+----+
|    |    |    | 2  |
+----+----+----+----+
| 2  |    |    |    |
+----+----+----+----+
|    |    |    |    |
+----+----+----+----+
```

# 04 思路分析-矩阵转置和反转

```python
def invert (field):
    """反转列表元素"""
    return [row[::-1] for row in field]


def transpose (field):
    """元素转置"""
    # zip一一对应组合， 返回一个zip对象， 每个元素是一个元组；
    return [list(row) for row in zip(*field)]
```

加入这两个操作可以大大节省我们的代码量,减少重复劳动。

# 04 思路分析-判断棋盘是否可移动

```
def move_left_possible(row):
    """判断列表中的一行是否可以移动"""
    # 0 2, 0 4, 2 2, 4 2
    # 0 0
    # # 1. 判断两个元素是否可以移动？
    # 4,2,2,2
    def is_change(i):
        if row[i] == 0 and row[i + 1] != 0:
            return True
        if row[i] != 0 and row[i + 1] == row[i]:
            return True
        return False

    # len(row)-1 =4-1 = 3
    # i= 0,1,2
    # 依次遍历每一行的每一个元素， 判断是否可以移动， 只要有一个时可以移动的， 返回True
    return any([is_change(i) for i in range(len(row) - 1)])
```

# 04 思路分析-判断棋盘是否可移动

```
check = {}

check['Left'] = lambda  field:  any([move_left_possible(row) for row in field])
# check['Right'] = lambda  field: any([ move_left_possible(row) for row in invert(field)])
# 判断每行内容反转后的field能否向左移动，  即原field能否向右移动；
check['Right'] = lambda field: check['Left'](invert(field))
# 判断转置后的field能否向左移动，  即原field能否向上移动；
check['Up'] = lambda  field: check['Left'](tranpose(field))
# 判断转置后的field能否向右移动，  即原field能否向下移动；
check['Down'] = lambda  field: check['Right'](tranpose(field))

print(check['Left']([[2, 4, 2, 4], [2, 4, 2, 0], [2, 4, 2, 4], [2, 4, 2, 4]]))
print(check['Right']([[2, 4, 2, 4], [2, 4, 2, 0], [2, 4, 2, 4], [2, 4, 2, 4]]))
print(check['Up']([[2, 4, 2, 4], [2, 4, 2, 0], [2, 4, 2, 4], [2, 4, 2, 4]]))
print(check['Down']([[2, 4, 2, 4], [2, 4, 2, 0], [2, 4, 2, 4], [2, 4, 2, 4]]))
```

# 04 思路分析-移动棋盘

```
# tight()   merge()  tight()
# [0,2,2,4]=== [2,2,4,0] == [4,0,4,0] == [4,4,0,0]



score = 0
def tight(row):
    """把所有非0的聚集在最左边"""
    # [0,2,2,4]  === [2,2,4]
    new_row = [item for item in row if item!=0]
    # if len(new_row) != len(row):
        # [2,2,4]+ [0] ==> new_row=new_row+[0]
    # range(0)不报错;
    new_row += [0 for item in range(len(row)-len(new_row))]
    return new_row
```

# 04 思路分析-移动棋盘

```
def merge(row):
    # 从左向右依次遍历，  如果两个值相等，  那么左边*2，  右边=0，
    for i in range(len(row)-1):  # 0,1,2,3
        if row[i] == row[i+1]:
            row[i] *= 2
            row[i+1] = 0
            global  score
            score += row[i]
    return  row


print(tight(merge([2,2,0,0])))
print(tight(merge([2,2,2,0])))
print(tight(merge([2,2,2,2])))
```

# 04 思路分析-绘制棋盘

```
class GameField(object):
    def __init__ (self, height=4, width=4, win_value=2048):
        self.height = height
        self.width = width
        self.win_value = win_value
        self.score = 0
        self.highscore = 100
        self.reset()
```

初始化棋盘的参数,可以指定棋盘的高和宽以及游戏胜利条件,默认是最经典的 4x4~2048。

# 05 动手实践

# 感谢聆听！

# THANK YOU!

## 《基于Linux平台python OOP的项目案例》

- 机构名称：西部开源技术中心
- 课程方向: python全栈开发与大数据分析