200行Python代码实现2048

一、实验说明

1.环境登录

无需密码自动登录，系统用户名shiyanlou

2. 环境介绍

本实验环境采用带桌面的Ubuntu Linux环境，实验中会用到桌面上的程序：

1.LX终端（LXTerminal）: Linux命令行终端，打开后会进入Bash环境，可以使用Linux命令

2.GVim：非常好用的编辑器，最简单的用法可以参考课程Vim编辑器

3. 环境使用

使用GVim编辑器输入实验所需的代码及文件，使用LX终端（LXTerminal）运行所需命令进行操作。

实验报告可以在个人主页中查看，其中含有每次实验的截图及笔记，以及每次实验的有效学习时间（指的是在实验桌面内操作的时间，如果没有操作，系统会记录为发呆时间）。这些都是您学习的真实性证明。

4. 知识点

本节实验中将学习和实践以下知识点：

1.Python基本知识

2.状态机的概念

二、实验内容

是的，又是2048，这回我们是用 Python 实现，只需要200行代码，不用很麻烦很累就可以写一个 2048 游戏出来。

实验楼上已有的 2048 课程：

GO语言开发2048

网页版2048

C语言制作2048

游戏玩法这里就不再赘述了，还会有比亲自玩一遍体会规则更快的的吗：）

2048 原版游戏地址：http://gabrielecirulli.github.io/2048

创建游戏文件 2048.py

首先导入需要的包：

import cursesfrom random import randrange, choicefrom collections import defaultdict

1. 主逻辑

1.1 用户行为

所有的有效输入都可以转换为"上，下，左，右，游戏重置，退出"这六种行为，用 actions 表示

actions = ['Up', 'Left', 'Down', 'Right', 'Restart', 'Exit']

有效输入键是最常见的 W（上），A（左），S（下），D（右），R（重置），Q（退出），这里要考虑到大写键开启的情况，获得有效键值列表：

letter\_codes = [ord(ch) for ch in 'WASDRQwasdrq']

将输入与行为进行关联：

actions\_dict = dict(zip(letter\_codes, actions \* 2))

1.2 状态机

处理游戏主逻辑的时候我们会用到一种十分常用的技术：状态机，或者更准确的说是有限状态机（FSM）

你会发现 2048 游戏很容易就能分解成几种状态的转换。

state 存储当前状态， state\_actions 这个词典变量作为状态转换的规则，它的 key 是状态，value 是返回下一个状态的函数：

Init: init()

oGame

Game: game()

oGame

oWin

oGameOver

oExit

Win: lambda: not\_game('Win')

oInit

oExit

Gameover: lambda: not\_game('Gameover')

oInit

oExit

Exit: 退出循环

状态机会不断循环，直到达到 Exit 终结状态结束程序。

下面是经过提取的主逻辑的代码，会在后面进行补全：

def main(stdscr):

def init():

#重置游戏棋盘

return 'Game'

def not\_game(state):

#画出 GameOver 或者 Win 的界面

#读取用户输入得到action，判断是重启游戏还是结束游戏

responses = defaultdict(lambda: state) ＃默认是当前状态，没有行为就会一直在当前界面循环

responses['Restart'], responses['Exit'] = 'Init', 'Exit' #对应不同的行为转换到不同的状态

return responses[action]

def game():

#画出当前棋盘状态

#读取用户输入得到action

if action == 'Restart':

return 'Init'

if action == 'Exit':

return 'Exit'

#if 成功移动了一步:

if 游戏胜利了:

return 'Win'

if 游戏失败了:

return 'Gameover'

return 'Game'

state\_actions = {

'Init': init,

'Win': lambda: not\_game('Win'),

'Gameover': lambda: not\_game('Gameover'),

'Game': game

}

state = 'Init'

#状态机开始循环

while state != 'Exit':

state = state\_actions[state]()

2. 用户输入处理

阻塞＋循环，直到获得用户有效输入才返回对应行为：

def get\_user\_action(keyboard):

char = "N"

while char not in actions\_dict:

char = keyboard.getch()

return actions\_dict[char]

3. 矩阵转置与矩阵逆转

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，看到后面就知道了。

矩阵转置：

def transpose(field):

return [list(row) for row in zip(\*field)]

矩阵逆转（不是逆矩阵）：

def invert(field):

return [row[::-1] for row in field]

4. 创建棋盘

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。

class GameField(object):

def \_\_init\_\_(self, height=4, width=4, win=2048):

self.height = height #高

self.width = width #宽

self.win\_value = 2048 #过关分数

self.score = 0 #当前分数

self.highscore = 0 #最高分

self.reset() #棋盘重置

4.1 棋盘操作

随机生成一个 2 或者 4

def spawn(self):

new\_element = 4 if randrange(100) > 89 else 2

(i,j) = choice([(i,j) for i in range(self.width) for j in range(self.height) if self.field[i][j] == 0])

self.field[i][j] = new\_element

重置棋盘

def reset(self):

if self.score > self.highscore:

self.highscore = self.score

self.score = 0

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

self.spawn()

self.spawn()

一行向左合并

(注：这一操作是在 move 内定义的，拆出来是为了方便阅读)

def move\_row\_left(row):

def tighten(row): # 把零散的非零单元挤到一块

new\_row = [i for i in row if i != 0]

new\_row += [0 for i in range(len(row) - len(new\_row))]

return new\_row

def merge(row): # 对邻近元素进行合并

pair = False

new\_row = []

for i in range(len(row)):

if pair:

new\_row.append(2 \* row[i])

self.score += 2 \* row[i]

pair = False

else:

if i + 1 < len(row) and row[i] == row[i + 1]:

pair = True

new\_row.append(0)

else:

new\_row.append(row[i])

assert len(new\_row) == len(row)

return new\_row

#先挤到一块再合并再挤到一块

return tighten(merge(tighten(row)))

棋盘走一步

通过对矩阵进行转置与逆转，可以直接从左移得到其余三个方向的移动操作

def move(self, direction):

def move\_row\_left(row):

#一行向左合并

moves = {}

moves['Left'] = lambda field: [move\_row\_left(row) for row in field]

moves['Right'] = lambda field: invert(moves['Left'](invert(field)))

moves['Up'] = lambda field: transpose(moves['Left'](transpose(field)))

moves['Down'] = lambda field: transpose(moves['Right'](transpose(field)))

if direction in moves:

if self.move\_is\_possible(direction):

self.field = moves[direction](self.field)

self.spawn()

return True

else:

return False

判断输赢

def is\_win(self):

return any(any(i >= self.win\_value for i in row) for row in self.field)

def is\_gameover(self):

return not any(self.move\_is\_possible(move) for move in actions)

判断能否移动

def move\_is\_possible(self, direction):

def row\_is\_left\_movable(row):

def change(i):

if row[i] == 0 and row[i + 1] != 0: # 可以移动

return True

if row[i] != 0 and row[i + 1] == row[i]: # 可以合并

return True

return False

return any(change(i) for i in range(len(row) - 1))

check = {}

check['Left'] = lambda field: any(row\_is\_left\_movable(row) for row in field)

check['Right'] = lambda field: check['Left'](invert(field))

check['Up'] = lambda field: check['Left'](transpose(field))

check['Down'] = lambda field: check['Right'](transpose(field))

if direction in check:

return check[direction](self.field)

else:

return False

4.2 绘制游戏界面

（注：这一步是在棋盘类内定义的）

def draw(self, screen):

help\_string1 = '(W)Up (S)Down (A)Left (D)Right'

help\_string2 = ' (R)Restart (Q)Exit'

gameover\_string = ' GAME OVER'

win\_string = ' YOU WIN!'

def cast(string):

screen.addstr(string + '\n')

#绘制水平分割线

def draw\_hor\_separator():

line = '+' + ('+------' \* self.width + '+')[1:]

separator = defaultdict(lambda: line)

if not hasattr(draw\_hor\_separator, "counter"):

draw\_hor\_separator.counter = 0

cast(separator[draw\_hor\_separator.counter])

draw\_hor\_separator.counter += 1

def draw\_row(row):

cast(''.join('|{: ^5} '.format(num) if num > 0 else '| ' for num in row) + '|')

screen.clear()

cast('SCORE: ' + str(self.score))

if 0 != self.highscore:

cast('HIGHSCORE: ' + str(self.highscore))

for row in self.field:

draw\_hor\_separator()

draw\_row(row)

draw\_hor\_separator()

if self.is\_win():

cast(win\_string)

else:

if self.is\_gameover():

cast(gameover\_string)

else:

cast(help\_string1)

cast(help\_string2)

5. 完成主逻辑

完成以上工作后，我们就可以补完主逻辑了！

def main(stdscr):

def init():

#重置游戏棋盘

game\_field.reset()

return 'Game'

def not\_game(state):

#画出 GameOver 或者 Win 的界面

game\_field.draw(stdscr)

#读取用户输入得到action，判断是重启游戏还是结束游戏

action = get\_user\_action(stdscr)

responses = defaultdict(lambda: state) #默认是当前状态，没有行为就会一直在当前界面循环

responses['Restart'], responses['Exit'] = 'Init', 'Exit' #对应不同的行为转换到不同的状态

return responses[action]

def game():

#画出当前棋盘状态

game\_field.draw(stdscr)

#读取用户输入得到action

action = get\_user\_action(stdscr)

if action == 'Restart':

return 'Init'

if action == 'Exit':

return 'Exit'

if game\_field.move(action): # move successful

if game\_field.is\_win():

return 'Win'

if game\_field.is\_gameover():

return 'Gameover'

return 'Game'

state\_actions = {

'Init': init,

'Win': lambda: not\_game('Win'),

'Gameover': lambda: not\_game('Gameover'),

'Game': game

}

curses.use\_default\_colors()

game\_field = GameField(win=32)

state = 'Init'

#状态机开始循环

while state != 'Exit':

state = state\_actions[state]()

6. 运行

填上最后一行代码：

curses.wrapper(main)

运行看看吧！

$ python 2048.py

全部代码

#-\*- coding:utf-8 -\*-

import curses

from random import randrange, choice # generate and place new tile

from collections import defaultdict

letter\_codes = [ord(ch) for ch in 'WASDRQwasdrq']

actions = ['Up', 'Left', 'Down', 'Right', 'Restart', 'Exit']

actions\_dict = dict(zip(letter\_codes, actions \* 2))

def get\_user\_action(keyboard):

char = "N"

while char not in actions\_dict:

char = keyboard.getch()

return actions\_dict[char]

def transpose(field):

return [list(row) for row in zip(\*field)]

def invert(field):

return [row[::-1] for row in field]

class GameField(object):

def \_\_init\_\_(self, height=4, width=4, win=2048):

self.height = height

self.width = width

self.win\_value = win

self.score = 0

self.highscore = 0

self.reset()

def reset(self):

if self.score > self.highscore:

self.highscore = self.score

self.score = 0

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

self.spawn()

self.spawn()

def move(self, direction):

def move\_row\_left(row):

def tighten(row): # squeese non-zero elements together

new\_row = [i for i in row if i != 0]

new\_row += [0 for i in range(len(row) - len(new\_row))]

return new\_row

def merge(row):

pair = False

new\_row = []

for i in range(len(row)):

if pair:

new\_row.append(2 \* row[i])

self.score += 2 \* row[i]

pair = False

else:

if i + 1 < len(row) and row[i] == row[i + 1]:

pair = True

new\_row.append(0)

else:

new\_row.append(row[i])

assert len(new\_row) == len(row)

return new\_row

return tighten(merge(tighten(row)))

moves = {}

moves['Left'] = lambda field: \

[move\_row\_left(row) for row in field]

moves['Right'] = lambda field: \

invert(moves['Left'](invert(field)))

moves['Up'] = lambda field: \

transpose(moves['Left'](transpose(field)))

moves['Down'] = lambda field: \

transpose(moves['Right'](transpose(field)))

if direction in moves:

if self.move\_is\_possible(direction):

self.field = moves[direction](self.field)

self.spawn()

return True

else:

return False

def is\_win(self):

return any(any(i >= self.win\_value for i in row) for row in self.field)

def is\_gameover(self):

return not any(self.move\_is\_possible(move) for move in actions)

def draw(self, screen):

help\_string1 = '(W)Up (S)Down (A)Left (D)Right'

help\_string2 = ' (R)Restart (Q)Exit'

gameover\_string = ' GAME OVER'

win\_string = ' YOU WIN!'

def cast(string):

screen.addstr(string + '\n')

def draw\_hor\_separator():

line = '+' + ('+------' \* self.width + '+')[1:]

separator = defaultdict(lambda: line)

if not hasattr(draw\_hor\_separator, "counter"):

draw\_hor\_separator.counter = 0

cast(separator[draw\_hor\_separator.counter])

draw\_hor\_separator.counter += 1

def draw\_row(row):

cast(''.join('|{: ^5} '.format(num) if num > 0 else '| ' for num in row) + '|')

screen.clear()

cast('SCORE: ' + str(self.score))

if 0 != self.highscore:

cast('HIGHSCORE: ' + str(self.highscore))

for row in self.field:

draw\_hor\_separator()

draw\_row(row)

draw\_hor\_separator()

if self.is\_win():

cast(win\_string)

else:

if self.is\_gameover():

cast(gameover\_string)

else:

cast(help\_string1)

cast(help\_string2)

def spawn(self):

new\_element = 4 if randrange(100) > 89 else 2

(i,j) = choice([(i,j) for i in range(self.width) for j in range(self.height) if self.field[i][j] == 0])

self.field[i][j] = new\_element

def move\_is\_possible(self, direction):

def row\_is\_left\_movable(row):

def change(i): # true if there'll be change in i-th tile

if row[i] == 0 and row[i + 1] != 0: # Move

return True

if row[i] != 0 and row[i + 1] == row[i]: # Merge

return True

return False

return any(change(i) for i in range(len(row) - 1))

check = {}

check['Left'] = lambda field: \

any(row\_is\_left\_movable(row) for row in field)

check['Right'] = lambda field: \

check['Left'](invert(field))

check['Up'] = lambda field: \

check['Left'](transpose(field))

check['Down'] = lambda field: \

check['Right'](transpose(field))

if direction in check:

return check[direction](self.field)

else:

return False

def main(stdscr):

def init():

#重置游戏棋盘

game\_field.reset()

return 'Game'

def not\_game(state):

#画出 GameOver 或者 Win 的界面

game\_field.draw(stdscr)

#读取用户输入得到action，判断是重启游戏还是结束游戏

action = get\_user\_action(stdscr)

responses = defaultdict(lambda: state) #默认是当前状态，没有行为就会一直在当前界面循环

responses['Restart'], responses['Exit'] = 'Init', 'Exit' #对应不同的行为转换到不同的状态

return responses[action]

def game():

#画出当前棋盘状态

game\_field.draw(stdscr)

#读取用户输入得到action

action = get\_user\_action(stdscr)

if action == 'Restart':

return 'Init'

if action == 'Exit':

return 'Exit'

if game\_field.move(action): # move successful

if game\_field.is\_win():

return 'Win'

if game\_field.is\_gameover():

return 'Gameover'

return 'Game'

state\_actions = {

'Init': init,

'Win': lambda: not\_game('Win'),

'Gameover': lambda: not\_game('Gameover'),

'Game': game

}

curses.use\_default\_colors()

game\_field = GameField(win=32)

state = 'Init'

#状态机开始循环

while state != 'Exit':

state = state\_actions[state]()

curses.wrapper(main)