

实验报告 ps3

姓名：张铁山 学号：15352418

【题意理解】

实现一个游戏，这个游戏类似于 word with friends 游戏，

(1) 由程序给你一些字母集 hand，hand 可以是多个，每个 hand 用字典表示其字母和对应的数量。

(2) 用 hand 里面的字母凑单词，每个字母有其对应的分值，这个字母分值对应表已经给出，单词必须是有效的，才可以得到分数，具体有效判断和计分规则在下面解题思路中会详细介绍。

要完成整个游戏实现过程，需要一步步实现一些函数，这些函数的功能必须保证实现正确才能保证游戏的正确实现，因此每个函数需要自己去测试其功能的正确性，当然，题目也给出了 test_ps3.py 文件帮助判断。

为增加本报告的可读性，将具体每个 problem 的理解跟解题思路放一起了，使阅读更流畅一点，避免来回查看题意理解及对应代码实现。

【解题思路】

Problem 1: Word scores

这个部分的任务是计算一个单词的得分，计算过程如下：

1：给定的字典对象 SCRABBLE_LETTER_VALUES 存储了各字母的分数，因为其中键值全部为小写，所以必须先将所给单词变成全部小写。

2：按照题意分别先计算出构成总得分的两部分 sum 和 second_com，其中 sum 表示单词中字母对应分值之和，second_com 为得分构成第 2 部分，返回两者之积 score。

函数实现代码及测试结果如下：

```
#获取单词得分
def get_word_score(word, n):

    word_low=word.lower() #将单词转换成全小写
    word_length=len(word_low) #字母个数
    if word_length==0: #单词为空字符串的情况
        return 0
    sum=0 # component 1
    for key in word_low:
        sum+=SCRABBLE_LETTER_VALUES[key]
    temp=7*word_length-3*(n-word_length)
    second_com=max(1, temp) # component 2

    score=sum*second_com # 得分
    return score
```

Example1: 单词 'it' n=7

计算过程依次是：Sum=1+1； temp=-1； second_com=1； score=2*1=2

Example2: 单词 'Apple' n=7

计算过程依次是：Sum=1+3+3+1+1=9； temp=29； second_com=29； score=9*29=261

```
>>> import ps3
>>> ps3.get_word_score('it', 7)
2
>>> ps3.get_word_score('Apple', 7)
261
```

执行 text_ps3.py 测试文件结果：可知函数功能实现正确

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.17134.706]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Users\86156>python test_ps3.py
Loading word list from file...
  83667 words loaded.
-----
Testing get_word_score...
SUCCESS: test_get_word_score()
-----
```

Problem 2: Dealing with hands

这个部分主要是对 hand 的操作，具体如下：

1: display_hand(hand): 以一种更友好的、直观的格式输出 hand 内容，实质就是按数量输出 hand 中字母，实现过程已给出；

2: deal_hand(n): 随机获取一个 hand，这个 hand 由元音字母和辅音字母组成，其中元音字母个数不超过 $n/3$ 的向上取整，剩下的则随机获取辅音字母；

3: update_hand(hand, word): 更新 hand，并返回更新后的 hand，就是从 hand 中减去 word 中用掉的字母。注意，word 可能是无效的，包括单词本身是无效的，或者 word 使用了 hand 中没有的字母，但是这并不对更新操作造成影响，无论 word 是否有效或者是否使用了 hand 中不存在的字母，对于更新操作，我们只关注 word 中用掉的 hand 之中的字母。实现代码如下：

```
def update_hand(hand, word):
    new_hand=hand.copy() #浅复制copy
    word_low=word.lower() #将单词全部转换成小写
    for x in word_low: #减去已经用掉的字母
        if x in new_hand and new_hand[x]>0:
            new_hand[x]-=1
    return new_hand
```

值得注意的是 new_hand 是采用直接赋值 (new_hand=hand) 还是采用浅拷贝(采用 copy 方法)。两者的区别就在于采用直接赋值，其实 new_hand 是 hand 的引用，new_hand 会随 hand 改变而改变；而在这里，采用 copy() 方法则不会。但是在这里两种方法并无影响，因为我们可以直接在 new_hand 上操作，最终返回的是 new_hand。当然，采用 copy() 方法不失为一种保险的做法。当然，也不是说 copy() 不会随父对象的改变而改变，采用 copy() 实际上也是一种浅拷贝：深拷贝父对象（一级目录），子对象（二级目录）不拷贝，还是引用。举个例子：

```
28 dict1={'num':1, 'arr':[1,2,3]}
29 dict2=dict1.copy()
30
31 dict1['arr'].pop()
32 print("dict2为: ",dict2)
```

```
In [3]: runfile('C:/Users/
86156/.spyder-py3/temp.py',
wdir='C:/Users/86156/.spyder-py3')
dict2为: {'num': 1, 'arr': [1, 2]}
```

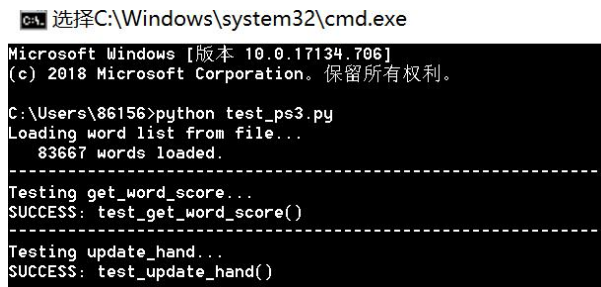
可知，dict1 的子对象发生了改变，dict2 随之而变。

回到 update_hand(hand, word)函数，手动测试结果如下：

```
>>> import ps3
>>> hand = {'a':1, 'q':1, 'l':2, 'm':1, 'u':1, 'i':1}
>>> ps3.display_hand(hand)
a q l l m u i
>>> new_hand=ps3.update_hand(hand, 'quail')
>>> new_hand
{'a': 0, 'q': 0, 'l': 1, 'm': 1, 'u': 0, 'i': 0}
>>> ps3.display_hand(new_hand)
l m
>>> ps3.display_hand(hand)
a q l l m u i
```

```
>>> hand = {'j':2, 'o':1, 'l':1, 'w':1, 'n':2}
>>> ps3.display_hand(hand)
j j o l w n n
>>> hand=ps3.update_hand(hand, 'jolly')
>>> hand
{'j': 1, 'o': 0, 'l': 0, 'w': 1, 'n': 2}
>>> ps3.display_hand(hand)
j w n n
```

执行 text_ps3.py 测试文件结果：可知函数功能实现正确



```
C:\> 选择C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.17134.706]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Users\86156>python test_ps3.py
Loading word list from file...
  83667 words loaded.

-----
Testing get_word_score...
SUCCESS: test_get_word_score()
-----
Testing update_hand...
SUCCESS: test_update_hand()
```

Problem 3. Valid words

这里主要是实现函数 is_valid_word(word, hand, word_list)，判断所给单词是否有效，有效是指 2 个条件：（1）这个单词在 word_list 之中；（2）word 中所用的每个字母都在 hand 中，并且该字母所用的数量小于等于 hand 中该字母的数量。

此处可暂时忽略条件（1），应该是为了方便对于这个函数的功能测试，而且条件（1）只需要用个条件判断 加 in 关键字即可实现。在确保条件（2）测试功能测试正常后添加进程序即可。

思路：将当前 hand 用个数组 hand_array 存起来，考虑到某些字母数量大于 1 的情况，可以借助 display_hand 实现的思想，用两个 for 循环逐个添加到数组中即可。遍历 word 中字母，如果该字母不在 hand_array 中，显然单词无效，如果该字母在 hand_array 中，则在 hand_array 中将对应位置替换为 '_'，不能直接删除，因为需要考虑对应字母的数量。具体代码如下：

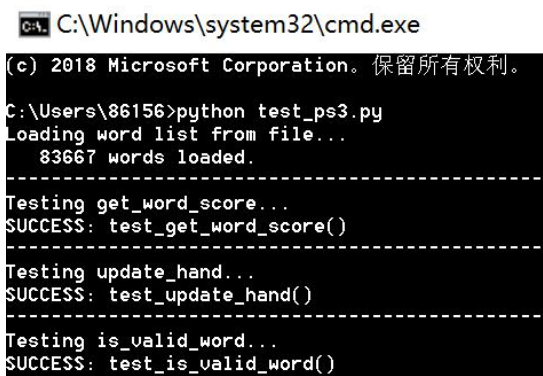
```
def is_valid_word(word, hand, word_list):
    #word 转化为全小写
    word_low=word.lower()
    #判断word是否来自word_list
    if word_low not in word_list:
        return False

    #将hand转化成列表
    hand_array=[]
    for key in hand:
        for i in range(hand[key]):
            hand_array.append(key)
    for x in word_low:
        #判断word 中字母是否都来自hand
        if x not in hand_array:
            return False
        #hand_array中减去已经用掉的字母
        else:
            pos=hand_array.index(x)
            hand_array[pos]='_'
    return True
```

题目没有给出测试案例，自己想了几个能够说明问题的案例，测试时将相关 word_list 判定条件注释掉了：

```
>>> import ps3
>>> hand = {'j':2, 'o':1, 'l':1, 'w':1, 'n':2}
>>> ps3.display_hand(hand)
j j o l w n n
>>> ps3.is_valid_word('jolly', hand)
False
>>> ps3.is_valid_word('joll', hand)
False
>>> ps3.is_valid_word('jjol', hand)
True
>>> ps3.is_valid_word('jollww', hand)
False
>>> ps3.is_valid_word('', hand)
True,
```

将条件 (1) word 在 word_list 之中添加到程序之中，执行 test_ps3.py 测试程序，结果如下，可知函数功能实现正确



```
C:\Windows\system32\cmd.exe
(c) 2018 Microsoft Corporation. 保留所有权利。
C:\Users\86156>python test_ps3.py
Loading word list from file...
83667 words loaded.
-----
Testing get_word_score...
SUCCESS: test_get_word_score()
-----
Testing update_hand...
SUCCESS: test_update_hand()
-----
Testing is_valid_word...
SUCCESS: test_is_valid_word()
```

Problem 4. Wildcards

实现通配符的功能，这里通配符用 * 号表示，具体如下：

(1) 改动 deal_hand(n)函数，随机获取的 hand 中必须保证有且仅有一个 *号，且辅音字母个数不变。只需减少获取元音字母 1 个即可，再往 hand 中添加键值对 '*' :1 可：代码如下：

```
hand={}
num_vowels = int(math.ceil(n / 3)) #向上取整

for i in range(num_vowels-1):
    x = random.choice(VOWELS) #获取随机元音字母
    hand[x] = hand.get(x, 0) + 1
hand['*']=1
```

(2) 相应的单词中可以出现 * 号的情况需要考虑，在 is_valid_word(word, hand, word_list)基础上添加一个条件判断，有 *号，则将其替换为 5 个元音字母中的任意一个，即出现 * 号的单词可以有 5 个变体，只要 5 个之中有一个在 word_list 之中，并且满足所用字符均出自当前的 hand 即可。具体过程在代码中已给出详细的注释。

```
def is_valid_word(word, hand, word_list):
    word_low=word.lower()#word 转化为全小写 * 号不影响
    if '*' in word_low: #有*号的情况
        word_after=[]# 存储*号变成任一元音字母之后的结果
        word_arr=list(word_low)#将单词转换为数组形式，方便操作
        pos=word_arr.index('*') #找到 * 号的位置
        for i in range(5):#5个元音字母
            word_temp=word_arr
            word_temp[pos]=VOWELS[i] #将*号变成元音字母
            string_temp="" #用于将*号处理后的单词数组变成字符串形式
            word_after.append(string_temp.join(word_temp))

        #只要word_after中有一个字符串在word_list中，且满足后续条件，单词就是有效的
        flag=False
        for i in range(5):
            if word_after[i] in word_list:
                flag=True
        if not flag:
            return False
    #没有通配符情况
    elif word_low not in word_list:
        return False

    hand_array=[] #将hand转化成列表
    for key in hand:
        for i in range(hand[key]):
            hand_array.append(key)
    for x in word_low:
        #判断word 中字母是否都来自hand
        if x not in hand_array:
            return False

        else:#hand_array中减去已经用掉的字母
            pos=hand_array.index(x)
            hand_array[pos]='_'
    return True
```

剩余判断单词字符是否均出自 hand 的代码不变。

注意：这里的 *号对于单词转小写是没有影响的，证明如下：

```
s='ABCD*EFG'
print(s.lower()) #结果abcd*efg
```


考虑到 *号对于计分的影响, *号本身对应分值为 0, 所以只需在程序顶部 字母: 分值字典 SCRABBLE_LETTER_VALUES 中添加键值对 '*': 0 即可。

执行 test_ps3.py 测试程序, 结果如下, 可知 problem4 功能实现正确

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.17134.706]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Users\86156>python test_ps3.py
Loading word list from file...
 83667 words loaded.

-----
Testing get_word_score...
SUCCESS: test_get_word_score()
-----
Testing update_hand...
SUCCESS: test_update_hand()
-----
Testing is_valid_word...
SUCCESS: test_is_valid_word()
-----
Testing wildcards...
SUCCESS: test_wildcard()
All done!
```

Problem 5. Playing a hand

实现玩游戏的过程, 即实现人机交互, 完善 play_hand 函数。先实现辅助函数 calculate_handlen, 用于计算当前 hand 中字符个数:

```
def calculate_handlen(hand):
    length=0
    for key in hand:
        length+=hand[key]
    return length
```

功能测试:

```
>>> import ps3
>>> hand={'a':2, '*':1, 'c':2, 'b':1, 'd':2}
>>> ps3.calculate_handlen(hand)
8
>>> hand_other={'a':2, '*':1, 'c':0, 'b':0, 'd':2}
>>> ps3.calculate_handlen(hand_other)
5
```

接下来就是 play_hand 函数的实现: 显然这是一个循环处理的过程, 只要 hand 不为空或者玩家没有自己输入!!号游戏就不会终止, 因此采用 while 循环, 再用条件判断语句依次判断 hand 是否为空, 玩家是否输入退出游戏的!!符号, 以及单词是否有效, 再用 break 和 continue 在不同的条件下, 代码已给出详细注释, 具体如下

```

def play_hand(hand, word_list):
    length=calculate_handlen(hand) #赋大于0的初值，用于执行while循环
    total_score=0 #总分
    while length>0:
        length=calculate_handlen(hand)
        if length==0: #hand中字母用光了的情况
            print("Ran out of letters. Total score: {} points".format(total_score))
            print("")
            break
        print("Current Hand: ", end=" ")
        display_hand(hand)
        print('Enter word, or "!!" to indicate that you are finished:', end=" ")
        word=input() #输入单词
        if not is_valid_word(word, hand, word_list):
            #单词无效，继续下一次循环
            print("That is not a valid word. Please choose another word.")
            print("")
            continue

        elif word=="!!":#玩家结束游戏
            print("Total score: {} points".format(total_score))
            break #输出总分后，跳出循环

        temp_score=get_word_score(word, length) #当前单词得分
        total_score+=temp_score #该轮游戏总得分
        print(' {} earned {} points. Total: {} points'.format(word, temp_score, total_score))
        hand=update_hand(hand, word) #更新hand

```

为测试函数功能是否正常实现，将最下方代码块做了一些改动：

```

if __name__ == '__main__':
    word_list = load_words()
    #play_game(word_list)
    hand1={'a':1, 'j':1, 'e':1, 'f':1, '*':1, 'r':1, 'x':1}
    hand2={'a':1, 'c':1, 'f':1, 'i':1, '*':1, 't':1, 'x':1}
    play_hand(hand2, word_list)

```

执行脚本，多番调试，所得结果与给出的案例完全一致，另外测试了一些案例，判断 play_hand 功能实现正常。

```

C:\Users\86156>python ps3.py
Loading word list from file...
 83667 words loaded.
Current Hand:  a j e f * r x
Enter word, or "!!" to indicate that you are finished: jar
jar earned 90 points. Total: 90 points

Current Hand:  e f * x
Enter word, or "!!" to indicate that you are finished: f*x
f*x earned 216 points. Total: 306 points

Current Hand:  e
Enter word, or "!!" to indicate that you are finished: !!
Total score: 306 points

```

```

C:\Users\86156>python ps3.py
Loading word list from file...
  83667 words loaded.
Current Hand: a c f i * t x
Enter word, or "!!" to indicate that you are finished: fix
fix earned 117 points. Total: 117 points

Current Hand: a c * t
Enter word, or "!!" to indicate that you are finished: ac
That is not a valid word. Please choose another word.

Current Hand: * t
Enter word, or "!!" to indicate that you are finished: *t
*t earned 14 points. Total: 131 points

Ran out of letters. Total score: 131 points

```

Problem 6. Playing a game

实现 `substitute_hand` 和 `play_game` 函数功能，来完成整个游戏。

`substitute_hand(hand,letter)`功能：

(1)、由玩家选择一个字母，若该字母在 `hand` 中，则 `hand` 中该键会被一个新字母替代作为新键，但是对应的值（也就是原字母的数量将作为新字母的数量），这个新字母 `new_letter` 将会从最上面给出的元音和辅音字母中随机选择一个，这个 `new_letter` 是不确定的。若 `letter` 不在 `hand` 之中，则对 `hand` 无影响。代码如下：

```

def substitute_hand(hand, letter):
    #随机获取一个新字母，用于替代
    new_letter=random.choice(VOWELS+CONSONANTS)
    #用户输入字符在hand中，则删除该键值对，并用new_letter替代
    if letter in hand:
        num=hand[letter] #先将数量保存起来
        del hand[letter] #删除要替代的键
        hand[new_letter]=num #添加新键值对，完成替换

    #letter不在hand中对hand 没有影响，所以无需操作，直接返回hand
    return hand # 返回处理后的hand

```

手动测试一下，判断 `substitute_hand (hand,letter)`功能实现正常

```

>>> import ps3
>>> ps3.substitute_hand({'h':1, 'e':1, 'l':2, 'o':1}, 'x')#letter不在hand中
{'h': 1, 'e': 1, 'l': 2, 'o': 1}
>>> ps3.substitute_hand({'h':1, 'e':1, 'l':2, 'o':1}, 'l')#letter在hand中
{'h': 1, 'e': 1, 'o': 1, 'f': 2}
>>> ps3.substitute_hand({'h':1, 'e':1, 'l':2, 'o':1}, 'x')#letter在hand中
{'h': 1, 'e': 1, 'l': 2, 'o': 1}

```

`play_game(word_list)`功能：

(1) 我们要处理不止一个 `hand`，而这个 `hand` 的数量 `hand_number` 是由用户输入决定的，本轮游戏得分 `sum_score` 为该轮游戏所有的 `hand` 得分之和；

(2) 每处理一个 hand, 需要提醒玩家是否需要替换 hand 中字母, 但是只有一次机会, 也就是说, 如果用户输入 yes 表示需要替换 hand 中字母后, 之后的 hand 便无须向用户询问是否需要替换 hand 中字母, 因为机会只有一次;

(3) 对每一个 hand, 还需要提供重玩一次的选择, 如果用户输入 yes 选择重新玩一次, 则这个 hand 的得分取两次得分中的最高分;

(4) hand 中 letter 个数 hand_size 也不一定就是 7。尝试实现不同的 hand_size 来完这个游戏;

(5) 这个要求题目没说, 但应该是默认要的: 对于用户输入的单词有效性处理应该要和之前的 play_hand 中一样, 给出相对应的提示, 单个 hand 的计分规则也和上面的一样。

这个函数没有绝对的标准实现, 因为 hand 的随机性以及 substitute_hand(hand, letter) 中的随机替代, 加上自己猜测的结果 word 难以保证一定会在 word_list 之中。所给示例中的 hand_size 也都是 7, 并没有体现题目需求。在这里, 我将 hand_size 的决定权交给了用户, 因此在输出格式上会与示例有点差别, 但是保证功能的实现和相关输入输出格式的合理性, 以及尽量符合所给示例的输出格式。考虑各种随机性, 只能尽量做到在一个测试案例中体现上述所有的功能, 但是不能保证, 因此将会多个测试案例中分别体现其功能, 代码相对较长, 在此就不贴出, 详见 ps3.py 文件:

下面给出一些能说明实现功能的测试案例, 要尽可能在一个例子中体现多个功能, 还是费了不少时间的:

```

Loading word list from file...
83667 words loaded.
Enter total number of hands: 2
Please input a number as the size of the hand: 7
Current hand: u a * y q v k
Would you like to substitute a letter? no
Current Hand: u a * y q v k
Please Enter word, or "!!" to indicate that you are finished: u*a
u*a earned 45 points. Total: 45 points

Current Hand: u y q k
Please Enter word, or "!!" to indicate that you are finished: usp
That is not a valid word. Please choose another word.

Current Hand: y q k
Please Enter word, or "!!" to indicate that you are finished: !!
Total score for this hand: 45
-----
Would you like to replay the hand? yes
Current hand: u a * y q v k
Would you like to substitute a letter? yes
Which letter would you like to replace: k

Current Hand: u a * y q v t
Please Enter word, or "!!" to indicate that you are finished: u*a
u*a earned 45 points. Total: 45 points

Current Hand: u y q t
Please Enter word, or "!!" to indicate that you are finished: ty
That is not a valid word. Please choose another word.

Current Hand: u q
Please Enter word, or "!!" to indicate that you are finished: qu
That is not a valid word. Please choose another word.

Ran out of letters. Total score for this hand: 45
-----
Please input a number as the size of the hand: 10
Current hand: e i u * d f f z j b
Current Hand: e i u * d f f z j b
Please Enter word, or "!!" to indicate that you are finished: die
die earned 4 points. Total: 4 points

Current Hand: u * f f z j b
Please Enter word, or "!!" to indicate that you are finished: !!
Total score for this hand: 4
-----
Total score over all hands: 49

```

上图说明的功能有

- (1) 总分功能, $45+4=49$ 两个 hand 的总分, 长方形方框部分
- (2) 改变 hand_size 获取随机 hand 的功能, 如上面的 10 和 7, 椭圆部分
- (3) 询问玩家是否需要 substitute_hand 功能, 询问玩家是否需要 replay 当前 hand, 并且当询问完一次, 玩家使用替换功能后, 之后的 hand 便不再询问, 如上面的例子, 第 1 个 hand 询问了, 但是第 2 个 hand 便没有再问。
- (4) 正常的输入检查及相应处理, 如输入单词无效的提示和处理, 玩家输入!!结束游戏的处理, hand 中字母用光的处理等。

下面的例子将着重说明其他功能:

```

C:\Users\86156>python ps3.py
Loading word list from file...
  83667 words loaded.
Enter total number of hands: 1
Please input a number as the size of the hand: 10
Current hand: o e u * w h l z f d
Would you like to substitute a letter? no
Current Hand: o e u * w h l z f d
Please Enter word, or "!!" to indicate that you are finished: do
do earned 3 points. Total: 3 points

Current Hand: e u * w h l z f
Please Enter word, or "!!" to indicate that you are finished: !!
Total score for this hand: 3
-----
Would you like to replay the hand? yes
Current hand: o e u * w h l z f d
Would you like to substitute a letter? yes
Which letter would you like to replace: f

Current Hand: o e u * w h l z d
Please Enter word, or "!!" to indicate that you are finished: do
do earned 3 points. Total: 3 points

Current Hand: e u * w h l z
Please Enter word, or "!!" to indicate that you are finished: whe×l
whe×l earned 290 points. Total: 293 points

Current Hand: u z
Please Enter word, or "!!" to indicate that you are finished: uz
That is not a valid word. Please choose another word.

Ran out of letters. Total score for this hand: 293
-----
Total score over all hands: 293

```

本例子体现功能：同一个 hand，玩家重玩一次，最终得分取高者，上面的例子只有一个 hand，能派出其他的因素，显然最终得分的 3 和 293 中的高的。

【实验总结】

客观来说，这个问题集的设计还是很有意义的，注重引导，没有一些非常复杂的算法，注重对题目的理解和逻辑实现，尤其是函数式编程的思想，模块化编程思想，似乎到目前为止的 problem set 都具备这些特点，估计之后的 problem set 也会是这个特点。

鉴于上述特点，独立耐心去做完每一个 problem set 收获还是挺大的，在实现每一个函数之后，都对其功能进行了手动的验证，然后再执行 test_ps3.py 脚本测试，确保功能实现正确。这个 problem set 主要考察了一些字典的操作，当然也不可避免的会有大多的列表、字符串等操作。

跟上一个 ps2 对比，个人感觉难度上略微升了一点点，一开始对于题意的理解还不是很透彻，不过文档给出了非常详细的过程，慢慢引导，通过实现一个个函数的过程，对题意加深了理解。比 ps2 需要花费更多的时间，特别是最后一个函数 play_game 的实现过程，为了展示出实现的功能，调试过程是费了很多工夫的。

通过这个以及之前的 problem set 的历练，对于一些稍大一点的项目，思路会更加清晰一点了，python 的学习也不再只是简单停留在利用基本语法和一些 python 本身的便捷操作去做题的阶段上。客观的说，如果题目没有给出一些列的实现过程，告诉我们要实现什么函数，最后一步步走到 play_game 的实现，这些 problem set 难度就顿时提高几倍了，不过这应该也不是这门课带给我们的意义。