Word Game 项目报告

17341097 廖永滨

一、实验内容

用 python 编写“Word Game”游戏，要求实现游戏的各个函数以完成整个游戏项目，具体要求在文档中。

二、实验要求

1.根据已有函数原型，实现一些基本的游戏函数。
2.根据游戏的基础设定、完成用户与电脑之间的基本交互需求并完善游戏规则。主要实现围绕 play_hand 这一函数展开，实现单一词汇的游戏过程。
3.在 2 的基础上，进一步完善游戏。围绕 play_game 这一函数展开，丰富对 play_hand 这一函数的使用，进一步完善游戏规则，加入一些词汇的变动以及总分数的设定。

三、实验流程

Start.

先读入单词表。

这里项目提供了一个 load_words()函数，调用了文件读写的方法。由于这里函数要求有 words.txt 文件的绝对路径或者相对路径。由于不同环境下绝对路径是不一样的，所以选用相对路径。

```
28      WORDLIST_FILENAME = (os.path.abspath(os.path.dirname(__file__))+"\\words.txt")
29
30      def load_words():
31
32          print("Loading word list from file...")
33          # inFile: file
34          inFile = open(WORDLIST_FILENAME, 'r')
35          # wordlist: list of strings
36          wordlist = []
37          for line in inFile:
38              wordlist.append(line.strip().lower())
39          print("  ", len(wordlist), "words loaded.")
40          return wordlist
```

同 PS2，我依旧采用了 os 库里的 os.path.abspath()方法获取了 py 脚本的绝对路径，解决有时候定位错误这个问题。也就是我是先获取脚本的绝对路径，再通过这个路径定位 words.txt 文件的路径。

Problem 1: Word scores

实现基本的 get_word_score 游戏功能函数：获取对应单词的分数

```
def get_word_score(word, n):

    word = word.lower()                                  #规范大小写
    component1 = 0                                       #乘积的第一部分
    for letter in word:
        component1 += SCRABBLE_LETTER_VALUES.get(letter,0)   #component1 : 第一部分的分数；word各个字母对应分数之和
    word_len = len(word)                                 #word_len : 单词长度
    component2 = 7*word_len - 3*(n-word_len)             #按公式计算出来的结果与1作比较，取较大值为乘积第二部分
    component2 = component2 if component2>1 else 1
    return component1*component2                         #返回乘积
```

根据文档中的公式得出分数后与 1 比较取最大即可，由于文档提示要规范大小写，所以就利用 lower()函数就可以解决对应的问题。

Problem 2: Dealing with hands

这一部分，题目要求完成 hands 单词表的更新与设置函数，文档已经提供了：deal_hand（获取随机的单词表）display_hand（展示单词表）get_frequency_dict（将字符串变成单词表）这三个函数，而我们只要实现 update_hand 这一函数即可。

```
def update_hand(hand, word):

    #由于不能修改原字典，所以要用深拷贝
    new_hand = deepcopy(hand)                    #用到 copy 库 要在顶部import
    word = word.lower()                          #规范大小写
    for letter in word:
        if new_hand.get(letter, 0) != 0:         #若e存在于newhand中
            if(new_hand[letter] > 0):
                new_hand[letter] -= 1            #则使对应的值减一

    #删去新hand中值为0的元素
    flag = True
    while(flag):
        flag = False
        for key in new_hand:
            if new_hand[key] == 0:
                del new_hand[key]
                flag = True
                break
    return new_hand
```

由于可以通过 for 遍历字符串的所有字符，所以 get_frequency_dict 这一函数我并没有用到，而是用更为普通的办法，就是让字符串里的每一个字符对应的 value 减掉 1，若减后为 0 代表没有了，就直接删除就可以了。这里文档要求不能修改传来的参数，所以要用深拷贝。

Problem 3. Valid words

这一部分要求我们判断输入是否合法（能获得分数）。

```
def is_valid_word(word, hand, word_list):
    #检测单词是否在词库中
    word = word.lower()
    cnt = 0
    if '*' in word:                                  #当输入有'*'时
        flag = 1
        for letter in "aeiou":                       #将'*'分别换成五个元音字母，若对应的五个单词都不存在则返回False
            word_x = word.replace('*', letter)
            if word_x in word_list:
                flag = 0
        if flag:
            return False
    else:                                            #当输入无'*'时
        if word not in word_list:                    #如果单词不存在，则返回False
            return False
    #检测单词是否在对应hand中
    temp_hand = deepcopy(hand)                       #用深拷贝
    for letter in word:                              #对于word的每一个字母
        if temp_hand.get(letter,0) == 0:             #如果hand中无这个字母，则返回False
            return False
        else:                                        #如果有这个字母，则其在字典中的值减一
            temp_hand[letter] = temp_hand[letter] - 1
            if temp_hand[letter] < 0:
                return False
    return True
```

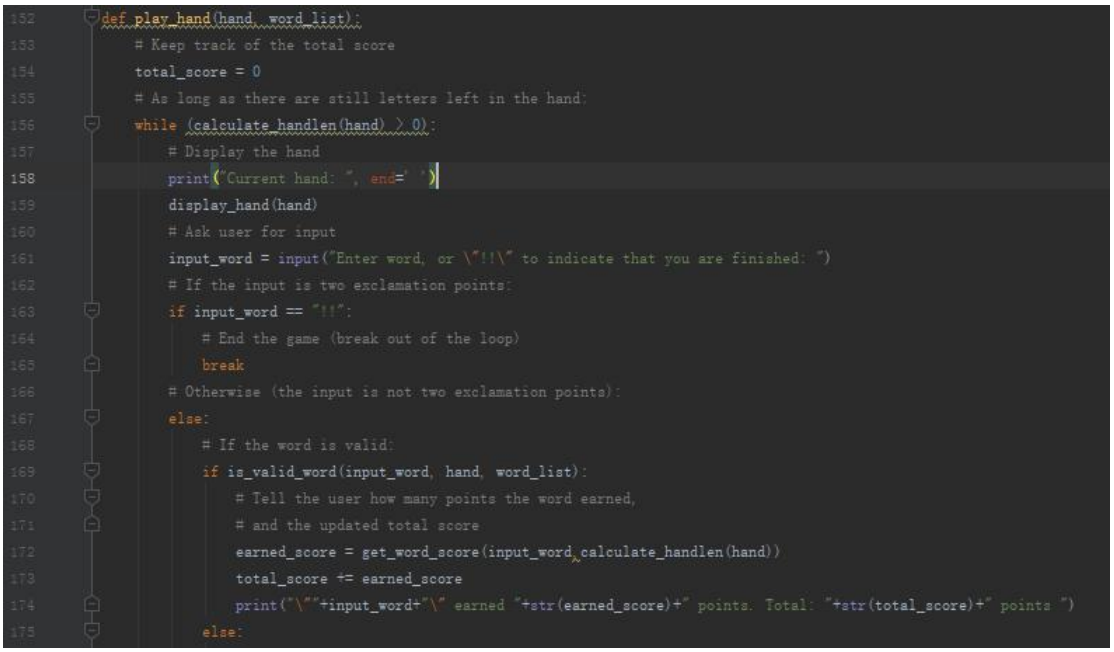
输入单词包含*号时，由于*号至多一个，所以只要遍历单词后替换第一个*号即可，依次替换为“aeiou”中的一个，如果在单词表中则进行第二步，同理无*号时，直接判断是否在单词表中，在则进入第二步。

第二步判断单词能否由 hand 中的字母凑成，如果不能则返回 flase，能则返回 true。（此处不能修改 hand 字典，所以要深拷贝一个新副本）

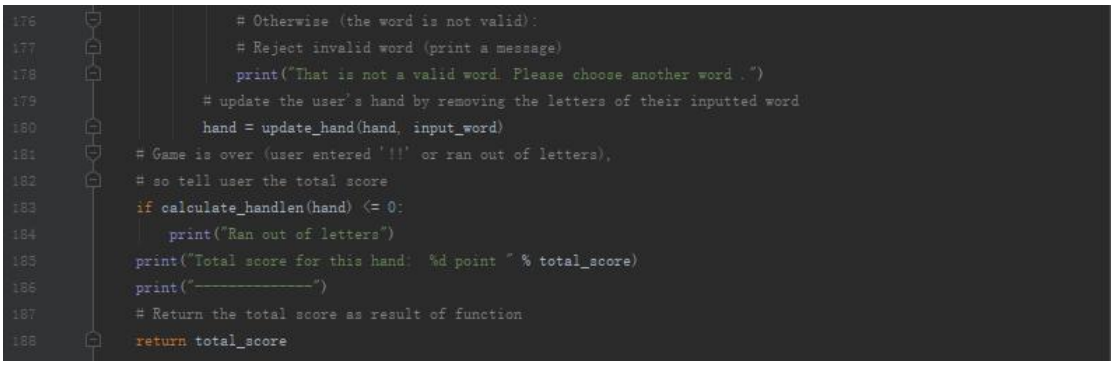
Problem 4. Wildcards

这一部分提到的*号（wildcards）能替换所有元音字母，具体实现在 Problem3 中判断是否为合法单词时已经实现。

Problem 5. Playing a hand



```
def play_hand(hand, word_list):
    # Keep track of the total score
    total_score = 0
    # As long as there are still letters left in the hand:
    while (calculate_handlen(hand) > 0):
        # Display the hand
        print("Current hand: ", end=' ')
        display_hand(hand)
        # Ask user for input
        input_word = input("Enter word, or \"!!\" to indicate that you are finished: ")
        # If the input is two exclamation points:
        if input_word == "!!":
            # End the game (break out of the loop)
            break
        # Otherwise (the input is not two exclamation points):
        else:
            # If the word is valid:
            if is_valid_word(input_word, hand, word_list):
                # Tell the user how many points the word earned,
                # and the updated total score
                earned_score = get_word_score(input_word,calculate_handlen(hand))
                total_score += earned_score
                print("\""+input_word+"\" earned "+str(earned_score)+" points. Total: "+str(total_score)+" points ")
            else:
```

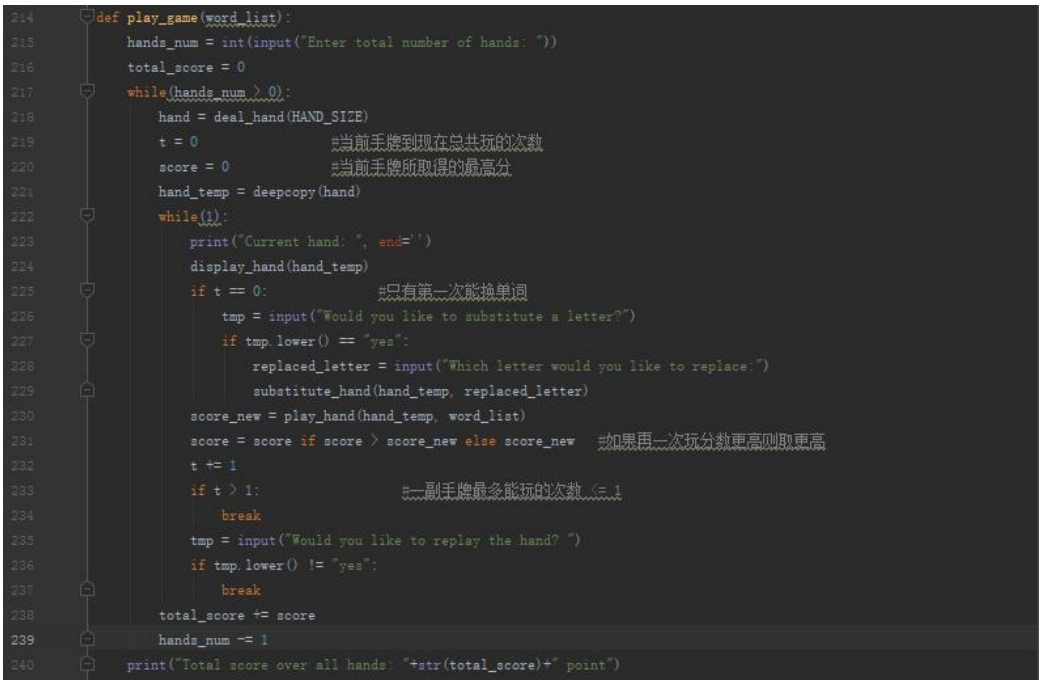


```
                # Otherwise (the word is not valid):
                # Reject invalid word (print a message)
                print("That is not a valid word. Please choose another word .")
            # update the user's hand by removing the letters of their inputted word
            hand = update_hand(hand, input_word)
    # Game is over (user entered '!!' or ran out of letters),
    # so tell user the total score
    if calculate_handlen(hand) <= 0:
        print("Ran out of letters")
    print("Total score for this hand:  %d point " % total_score)
    print("----------")
    # Return the total score as result of function
    return total_score
```

这一部分，要求我们实现一个单次游戏的实现，具体实现方法在给出的 py 文件中就已经有非常具体的伪代码了，将之“翻译”过来即可。

Problem 6. Playing a game

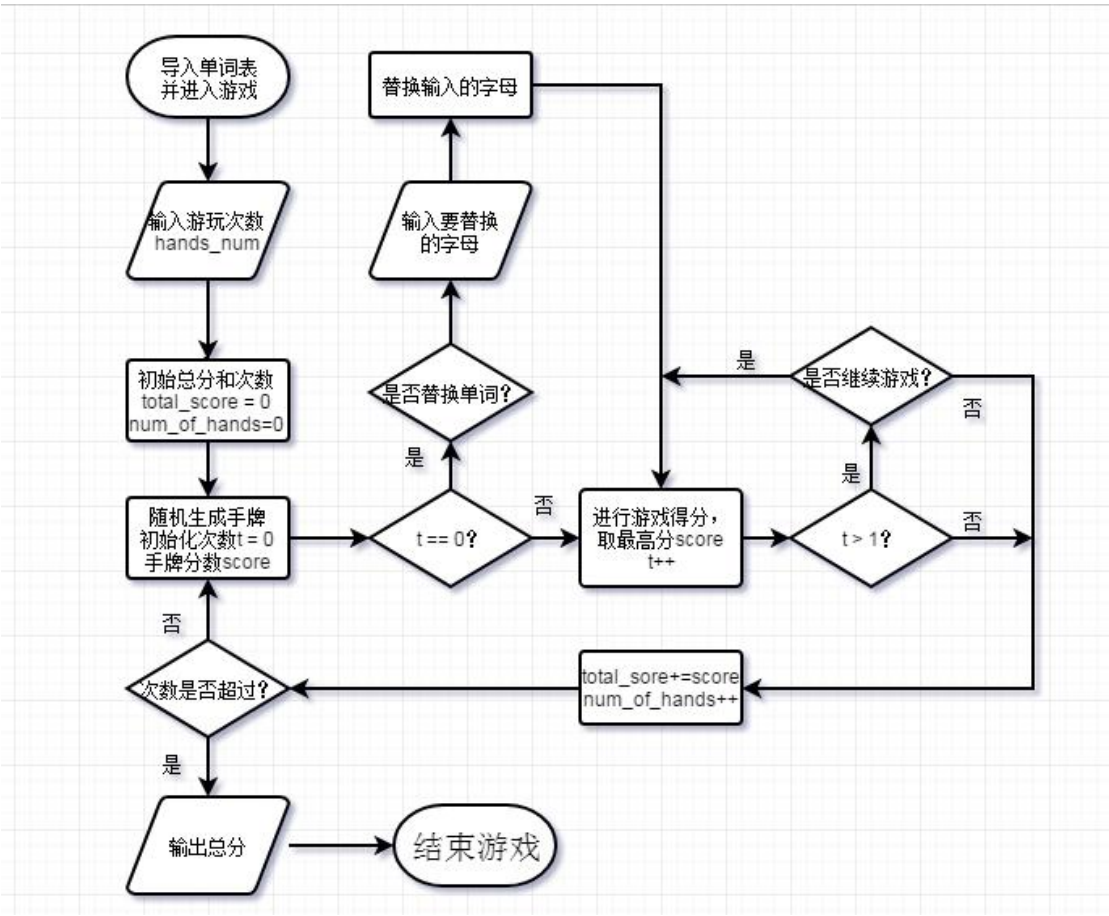
这一部分，题目提高了游戏性并进一步完善了游戏规则。要求我们实现多次游戏的统一以及单次游戏前的字母替换操作。



```
def play_game(word_list):
    hands_num = int(input("Enter total number of hands: "))
    total_score = 0
    while(hands_num > 0):
        hand = deal_hand(HAND_SIZE)
        t = 0                   #当前手牌到现在总共玩的次数
        score = 0               #当前手牌所取得的最高分
        hand_temp = deepcopy(hand)
        while(1):
            print("Current hand: ", end='')
            display_hand(hand_temp)
            if t == 0:              #只有第一次能换单词
                tmp = input("Would you like to substitute a letter?")
                if tmp.lower() == "yes":
                    replaced_letter = input("Which letter would you like to replace:")
                    substitute_hand(hand_temp, replaced_letter)
            score_new = play_hand(hand_temp, word_list)
            score = score if score > score_new else score_new   #如果更一次玩分数更高则取更高
            t += 1
            if t > 1:                   #一副手牌最多能玩的次数 <= 1
                break
            tmp = input("Would you like to replay the hand? ")
            if tmp.lower() != "yes":
                break
        total_score += score
        hands_num -= 1
    print("Total score over all hands: "+str(total_score)+" point")
```

首先，初始化总分并输入要玩的次数。对于每次游戏，先初始化手牌和对应手牌的最高分，然后进入循环（每次游戏都有多次机会，题目是有 2 次机会），先询问是否替换手牌（只有第一次游戏能替换，之后的重玩机会都将沿用这一手牌），然后开始玩，多次 play_hand 后取最高分加到 total_score 中。最后，当多次游戏结束后，total_score 就是总分。

四、程序流程图



五、实验总结

1、对 Python 一些操作有了更深入的了解，比如 in，for 等等语法
2、深入了解了深拷贝与浅拷贝的区别并应用到了程序中，程序中多处代码用到了深拷贝。
3、项目的结构思想更加深入，对模块化程序设计有了更好的认识
4、对遍历删除元素有了新认识，由于删除元素会导致迭代器终止，所以要用遍历的手法删除元素就要求多次遍历或者保存迭代信息，为了简单，我采取了前者。