Hangman 项目报告

17341097 廖永滨

一、实验内容

用 python 编写“hangman”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、用户与电脑之间的交互需求及游戏规则等实现游戏的主体。
3.在 2 的基础上，完成几个提示用户猜词的函数，并利用这些函数在游戏主体中加入提示功能。

三、实验流程

Start.

这一部分，题目要求完成基本的 Hangman 游戏功能。

1. 首先，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。

```
WORDLIST_FILENAME = (os.path.abspath(os.path.dirname(__file__))+"\\words.txt")
#要求单词表文件 words.txt与.py在同一个目录


def load_words():

    print("Loading word list from file...")
    # inFile: file
    inFile = open(WORDLIST_FILENAME, 'r')
    # line: string
    line = inFile.readline()
    # wordlist: list of strings
    wordlist = line.split()
    print("  ", len(wordlist), "words loaded.")
    return wordlist
```

这里出现了一个小问题，由于开发环境的不同，或者说是文档结构的不同，python 相对路径的定位经常出现问题，有时候会定位到别的地方去。这里我采用了 os 库里的 os.path.abspath()方法获取了 py 脚本的绝对路径，解决的这个问题。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。效果等同于相对路径，只是方式有点像绝对路径。

2. 其次，需要实现基本的 input，print 操作。

先是初始化一些关键的变量。

guesses_remaining 表示机会数， letters_guessed 表示猜过的字母列表，

Flag_win 表示胜利与否，ok，开始循环判断

按题目要求步骤进行循环：

1)已经胜利？没有次数？是则将 Flag_win 置为 True 后跳出循环

2)输入猜测并加入猜测列表

3)将猜测结果告诉玩家。结果失败则猜测次数减 1

4)判断是否所有字母都被猜出来，是则置胜利标志 Flag_win 为 True。

5)继续循环操作

循环结束后，通过 Flag_win 判断胜负并告知玩家。

Part1.

这一部分，题目要求增设 Hangman 游戏功能。

1. 首先，增加 is_word_guessed(secret_word, letters_guessed) 判断是否猜中，此部分内容事实上在 Part1 就已经完成（不然不存在胜负概念），设计方法很简单，只要选择的单词中的每个字母都出现在猜测列表中就返回 True

```
def is_word_guessed(secret_word, letters_guessed):
    #对于选择的单词，如果每个字母都已经被猜中则返回True
    for letter in secret_word:
        if letter not in letters_guessed:
            return False
    return True
```

2. 获取被屏蔽掉部分信息的单词用以提示用户，设计函数：

get_guessed_word(secret_word, letters_guessed)

写法就是拷贝选择的单词生成一个字符串，对于猜测列表里没出现过的字母就用‘_’屏蔽掉（可以用 replace 函数操作）

```
def get_guessed_word(secret_word, letters_guessed):
    #先创建一个字符串并赋值为选择的词
    guessed_word = secret_word
    for i in guessed_word :
        if i not in letters_guessed:
            guessed_word = guessed_word.replace(i,"_")
            #对于未被猜中的字母则用_替代
    return guessed_word
```

3. 获取合法的输入字母防止错误输入，设计函数：

get_available_letters(letters_guessed)

先创建一个从 a 到 z 字母表，然后剔除被猜过的字母形成的字符串就是结果了（为了防止输入错误我还加设了一个判断输入长度的判断语句，不为

1 也认为输入错误）

```
def get_available_letters(letters_guessed):
    #创建一个字母表
    available_letters = string.ascii_lowercase
    for i in available_letters:
        if i in letters_guessed:
            available_letters = available_letters.replace(i,'')
            #已经被猜测过的字母在表中剔除
    return available_letters
```

Part2.

这一部分，题目极大的提高了游戏性并完善了游戏规则。

A. 首先，游戏开始前先告知玩家目标单词的长度（用 len 函数轻松解决）
每次猜测前都告诉玩家合法的输入（Part2 已经解决）

```
print("I am thinking the of a word that is "+str(len(secret_word))+" letters long.")
```

B. 每次猜测前都告诉玩家还剩多少次猜测机会（print 出 guesses_remaining）
每次猜测后都告诉玩家当前的进度（Part2 中的 get_guessed_word()已解决，将函数返回的字符串 print 就可以了）
每次猜测流程结束都用“————”分割：在 while 循环语句第一句 print 出来“————”就可以了

```
        print("-----------------------------------")
        print("You have "+str(guesses_remaining)+" guesses left")
```

C. 玩家只有 3 次非法输入的 warning 次数，初始化 warnings_remaining = 3，
每次猜错就警告并扣除 1 次次数，没有次数了就扣除 guesses_remaining 1 次次数，然后重新进入循环。

```
if len(letter_guessed)!=1 or letter_guessed not in get_available_letters(letters_guessed):
#如果出现非法输入则：（如果不是一个字符也认为是非法输入）
    if warnings_remaining > 0:
        warnings_remaining -= 1
        if len(letter_guessed)!=1:
            print("Oops! You need to input only a letter. You now have "+str(warnings_remaining)+" warnings: ")
        else:
            print("Oops! You've already guessed that letter. You now have "+str(warnings_remaining)+" warnings: ")
        print(get_guessed_word(secret_word, letters_guessed))
        #扣减警告次数
    else:
        print("Oops! You've already guessed that letter. You have no warnings left")
        guesses_remaining -= 1
        print("so you lose one guess: "+get_guessed_word(secret_word, letters_guessed))
        #扣减猜测次数
    continue
```

D. 完善游戏规则

首先是错误判定，只要是错误的字符或是重复猜测都会 warning 并执行处罚（同 Part2 的所说的判断就可以了，只要判断输入是否在合法表即可）

猜错惩罚修改：

猜错部分元音字母（a e i o）就扣除 2 次机会，否则只扣 1 次机会

```
    if letter_guessed in ['a','e','i','o']:
        guesses_remaining -= 1 #部分元音字母多扣1次机会
    guesses_remaining -= 1
```

E. 分数机制

Total score = guesses_remaining* number unique letters in secret_word

其中 guesses_remaining 为剩下的机会，与所猜词中不同字母数相乘即为结果。

我实现的方式就是初始化 unique_letters = 0，每当猜中一次就+1，猜完后如果胜利，猜中的次数必然就是所猜词的不同字母数，用它就可以进行分数计算了。如果没赢，虽然该值与所猜词的不同字母数不同，但是不要求分数结算，所以也就不再重要。

Part3.

这一部分，题目给予用户与游戏更大的活动空间。分别设计了三个互相关联的函数，目的在于解决*输入操作。

1. match_with_gaps(my_word, other_word) 函数，实现很简单，先对 my_word 继续必要的去除空格操作，然后进行字符串比较，my_word 中_视为万能字符，比较时自动跳过。匹配则返回 True。

```
def match_with_gaps(my_word, other_word):
    myword_not_b = my_word.replace(' ','')
    #将被_屏蔽的字符串中的空格全部剔除
    if(len(myword_not_b) != len(other_word)):
        #如果两个词长度不一样就一定不同
        return False
    for i in range(len(myword_not_b)):
        if myword_not_b[i] != '_' and myword_not_b[i] != other_word[i]:
        #出现除_外不同的字母时说明两个词不匹配
            return False
    return True
```

2. show_possible_matches(my_word) 函数，在字典中挨个找词汇。先初始化匹配成功 flag 标志为 False。调用 1 中的匹配函数，匹配则输出并置成功 flag 为 True，负责继续找。最后，如果 flag 仍为 False，则打印匹配失败字样。

此处为了达到题目所显示的每个单词空一格的效果，对 print 语句进行参数设置：print(word,end='')

```
def show_possible_matches(my_word):
    matches_flag = False
    for word in wordlist:
        if match_with_gaps(my_word,word):
        #根据匹配函数在字典中找到匹配的单词打印出来
            print(word,end='')
            print(" ",end='')
            matches_flag = True
    if not matches_flag:
        print("No matches found")
    else:
        print("")
```

3. Hangman with hints()函数的编写，事实上和 Hangman()基本一样，只是对输入语句进行了一个额外的判断，如果输入语句是’*’，则调用函数显示所有相匹配的词。

```
if letter_guessed == '*':
# * 判断，如果文件输入了*，则展示所有匹配的单词
    print("Possible word matches are: ")
    show_possible_matches(get_guessed_word(secret_word, letters_guessed))
    continue
```

四、实验总结

1、Python 掌握得更加熟练了
2、对 Python 一些操作有了更深入的了解，比如 in，for 等等语法
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