Contents

1 Basic Test Results

```
Starting tests...
Mon Mar 27 21:41:27 IDT 2017
0debe91b1a25d4e71d9a9f54223d4d3f67eeffec  -


Archive:  /tmp/bodek.OUKC6s/intro2cs2/ex4/amit.baskin/presubmission/submission
  inflating: src/hangman.py
  inflating: src/README

Testing README...
Done testing README...

Running presubmit tests...
6 passed tests out of 6
result_code    ex4    6    1
Done running presubmit tests

Tests completed

Additional notes:

There will be additional tests which will not be published in advance.
```

2 README

```
amit.baskin
312259013
Amit Baskin


=
======================

  no special comments  =
======================



I did not discuss the excercise with anyone


##############################
=   README for ex4   =
##############################


##############################
Description: ex4.py:
a program that executes the game "hangman"
##############################


the functions that are used in the program, are the following:


update_word_pattern(word, pattern, letter) -
the function gets a word, the current pattern and a letter as parameters
and returns an updated pattern that contains the letter
    :param word: the word
    :param pattern: the current pattern
    :param letter: the letter
    :return: updated pattern


original_pattern(word)-
the function returns a blank note ("_"), multiplied by the length of the given word
    :param word: the given word
    :return: a blank note ("_"), multiplied by the length of the given word


run_single_game(words_list) -
the function gets a list of words from a file and runs the game itself
    :param words_list: a list of words from a file
    :return: graphic messages in context with the game progress


run_multiple_games(words_list) -
the function ensures that the game will not be exited while another game shall be played
    :param words_list: a list of words
    :return: a beginning of another game or not if the user chooses not to


character_in_word(word, pattern) -
```

```
the function checks whether or not every letter that in the pattern is in the word exactly in the same place
    :param word: the word to be guessed
    :param pattern: the pattern to be shown
    :return: True if the check is positive and False if negative


letter_in_guess_list(word, wrong_guess_list) -
the function checks whether the chosen letter is in the wrong guesses list
    :param word: the word to be guessed
    :param wrong_guess_list: a list of previous wrong guesses
    :return: True if the check is positive and False if negative


filter_words_list(words, pattern, wrong_guess_lst) -
the function filters the list of words according to a few conditions
    :param words: the words to be filtered
    :param pattern: the given pattern
    :param wrong_guess_lst: the list of previous wrong guesses
    :return: the filtered list of words


max_char_count(words, pattern) -
the functions tells which is the most popular letter
    :param words: the given words
    :param pattern: the given pattern
    :return: the most popular letter


letter_to_index(letter) -
the function returns the index of the given letter in an alphabet list
        :param letter: the letter to be checked
        :return: the index of the given letter in an alphabet list


index_to_letter(index) -
thee function returns the letter corresponding to the given index
        :param index: the given index
        :return: the letter corresponding to the given index


choose_letter(words, pattern) -
the function chooses the letter according to a few conditions
    :param words: a list of words
    :param pattern: the given pattern
    :return: the chosen letter


main() -
the function runs the game itself
    :return: the game running




################################
=   No Special Comments   =
################################




A question:
What would you need to change in your program in order to play the game with a list
of words in hebrew and with hebrew letters?

Answer:
```

```
128  The conditions for the non valid msg would be different.
129  only letters within the ascii values of the hebrew letters should be accepted.
130  The same conditions should be written to functions of the letter to index
131  and index to letter so the letter in the function choose letter
132  will be chosen correctly.
```

3 hangman.py

```
1  ##################################################################
2  # FILE: hangman.py
3  # WRITER: Amit Baskin , amit.baskin , 312259013
4  # EXERCISE : intro2cs ex4 2016-2017
5  # DESCRIPTION: A program that executes the game "hangman".
6  ##################################################################
7
8
9  import hangman_helper  # a python file contains a few functions for assistance
10
11
12 UNDER_SCORE = '_'
13
14 CHAR_A = 97
15
16 NUM_OF_LETTERS = 26
17
18
19 def update_word_pattern(word, pattern, letter):
20     """
21     the function gets a word, the current pattern and a letter as parameters
22     and returns an updated pattern that contains the letter
23     :param word: the word
24     :param pattern: the current pattern
25     :param letter: the letter
26     :return: updated pattern
27     """
28
29     word_characters_lst = list(word)  # unpack the string "word" into a list that contains
30     # the character that are in the string
31     pattern_characters_lst = list(pattern)  # unpack the string "pattern" into a list that contains
32     # the characters that are in the string
33     len_lst1 = len(word_characters_lst)
34
35     for i in range(len_lst1):
36         if word_characters_lst[i] == letter:  # if the letter is in the word:
37             pattern_characters_lst[i] = letter  # insert the letter into the pattern
38             # exactly where it is in the word
39
40     updated_pattern = ''.join(pattern_characters_lst)  # transform the list
41     # of the characters of the pattern back into a string
42     return updated_pattern
43
44
45 def original_pattern(word):
46     """
47     the function returns a blank note ("_"), multiplied by the length of the given word
48     :param word: the given word
49     :return: a blank note ("_"), multiplied by the length of the given word
50     """
51
52     len_word = len(word)  # the length of the given word
53     orig_pattern = [UNDER_SCORE] * len_word
54     orig_pattern = ''.join(orig_pattern)
55     return orig_pattern
56
57
58 def run_single_game(words_list):
59     """
```

```
    the function gets a list of words from a file and runs the game itself
    :param words_list: a list of words from a file
    :return: graphic messages in context with the game progress
    """

    error_count = 0  # the game begins with the amount of zero errors
    word = hangman_helper.get_random_word(words_list)  # pick a random
    # word from the list with the assistance of the function 'get_random_word'
    pattern = original_pattern(word)  # name 'pattern' a string of blank notes
    # by calling the function 'original_pattern'
    wrong_guess_lst = []  # the game begins with an empty list of wrong guesses
    chosen_letters = []  # the game begins with an empty list of chosen letters
    msg = hangman_helper.DEFAULT_MSG  # the game begins with a default message: ''

    while (error_count < hangman_helper.MAX_ERRORS) and (pattern != word):
        # while the amount of errors is smaller than the number of
        # maximum errors allowed and the user did not find the word,
        # hence the pattern does not equal to the word

        hangman_helper.display_state(pattern, error_count, wrong_guess_lst, msg)
        # call the function display_state which displays
        # the pattern, the amount of errors made, the list of wrong guesses,
        # and the required message

        user_input = hangman_helper.get_input()  # equals to the input given
        # by the user including the type of input and the input itself
        letter = user_input[1]  # the item in the '1' place in the tuple of the input should be the letter
        input_type = user_input[0]  # the type of the input should be
        # signified in the '0' place in the tuple of the input

        if input_type == hangman_helper.LETTER:  # if the input is a letter

            if (len(letter) != 1) or (not letter.islower()):
                # if the length of the input is different than 1 or if the input is not a letter
                msg = hangman_helper.NON_VALID_MSG
                # the msg is updated to a message that says that the input is not valid

            elif letter in chosen_letters:  # if the letter has already been chosen
                msg = hangman_helper.ALREADY_CHOSEN_MSG + letter  # the msg is updated to a message that says
                #  that the letter has already been chosen and with the letter that was chosen

            elif letter in word:  # if the letter is in the word
                chosen_letters.append(letter)  # add the letter to the list of chosen letters
                pattern = update_word_pattern(word, pattern, letter)  # the letter is to be added to the pattern
                msg = hangman_helper.DEFAULT_MSG  # the msg is updated to the default message

            else:
                chosen_letters.append(letter)  # otherwise, add the letter to the list of the chosen letters
                wrong_guess_lst.append(letter)  # add the letter to the list of wrong guesses
                error_count += 1  # the count of errors gets bigger by one
                msg = hangman_helper.DEFAULT_MSG  # the msg is updated to the default message

        elif input_type == hangman_helper.HINT:  # if the type of the input is a hint
            filtered_words_list = filter_words_list(words_list, pattern, wrong_guess_lst)
            # then the words_list will be filtered

            hint_letter = choose_letter(filtered_words_list, pattern)  # the hint letter will be chosen with the
            # assistance of the function choose_letter, and it will pick from the list 'filtered_words_list'

            msg = hangman_helper.HINT_MSG + hint_letter  # the msg is updated to the hint message plus the letter
            # that was chosen

    if pattern == word:  # if the pattern equals to the word, hence the word was found
        msg = hangman_helper.WIN_MSG  # the msg is updated to the 'winning message'

    else:
        msg = hangman_helper.LOSS_MSG + word  # the msg is updated to the 'loosing message'
        # + the word that was not discovered
```

```
128
129     hangman_helper.display_state(pattern, error_count, wrong_guess_lst, msg, ask_play=True)
130     # the current state is given and the question whether another game shall be played or not
131
132
133 def run_multiple_games(words_list):
134     """
135     the function ensures that the game will not be exited while another game shall be played
136     :param words_list: a list of words
137     :return: a beginning of another game or not if the user chooses not to
138     """
139
140
141     run_game = True
142
143     while run_game:
144         run_single_game(words_list)
145
146         user_input = hangman_helper.get_input()
147
148         if user_input[1]:
149             run_game = True
150
151         if not user_input[1]:
152             run_game = False
153
154
155 def character_in_word(word, pattern):
156     """
157     the function checks whether or not every letter that in the pattern is in the word exactly in the same place
158     :param word: the word to be guessed
159     :param pattern: the pattern to be shown
160     :return: True if the check is positive and False if negative
161     """
162
163     for i in range(len(word)):
164         if word[i] != pattern[i] and pattern[i] != UNDER_SCORE:
165             return True
166     return False
167
168
169 def letter_in_guess_list(word, wrong_guess_list):
170     """
171     the function checks whether the chosen letter is in the wrong guesses list
172     :param word: the word to be guessed
173     :param wrong_guess_list: a list of previous wrong guesses
174     :return: True if the check is positive and False if negative
175     """
176
177     for letter in word:
178         if letter in wrong_guess_list:
179             return True
180
181     return False
182
183
184 def filter_words_list(words, pattern, wrong_guess_lst):
185     """
186     the function filters the list of words according to a few conditions
187     :param words: the words to be filtered
188     :param pattern: the given pattern
189     :param wrong_guess_lst: the list of previous wrong guesses
190     :return: the filtered list of words
191     """
192
193     returned_words_list = []
194     for word in words:
195         if len(word) != len(pattern):
```

```
196             continue
197
198         elif character_in_word(word, pattern):
199             continue
200
201         elif letter_in_guess_list(word, wrong_guess_lst):
202             continue
203
204         else:
205             returned_words_list.append(word)
206
207     return returned_words_list
208
209
210 def max_char_count(words, pattern):
211     """
212     the functions tells which is the most popular letter
213     :param words: the given words
214     :param pattern: the given pattern
215     :return: the most popular letter
216     """
217
218     counters = [0] * NUM_OF_LETTERS
219
220
221     def letter_to_index(letter):
222         """
223         the function returns the index of the given letter in an alphabet list
224         :param letter: the letter to be checked
225         :return: the index of the given letter in an alphabet list
226         """
227         return ord(letter.lower()) - CHAR_A
228
229
230     def index_to_letter(index):
231         """
232         thee function returns the letter corresponding to the given index
233         :param index: the given index
234         :return: the letter corresponding to the given index
235         """
236         return chr(index + CHAR_A)
237
238     for word in words:
239         for letter in word:
240             if letter in pattern:
241                 continue
242             counters[letter_to_index(letter)] += 1
243
244     return index_to_letter(counters.index(max(counters)))
245
246
247 def choose_letter(words, pattern):
248     """
249     the function chooses the letter according to a few conditions
250     :param words: a list of words
251     :param pattern: the given pattern
252     :return: the chosen letter
253     """
254     letters_in_words = ''.join(words)
255     most_popular_letter = max_char_count(letters_in_words, pattern)
256     return most_popular_letter
257
258
259 def main():
260     """
261     the function runs the game itself
262     :return: the game running
263     """
```

```
264
265     words_list = hangman_helper.load_words(file='words.txt')
266     run_multiple_games(words_list)
267
268
269 if __name__ == "__main__":  # responsible to start the game
270     hangman_helper.start_gui_and_call_main(main)
271     hangman_helper.close_gui()
```