Contents

1 Basic Test Results

```
Sun Nov 22 15:05:12 IST 2020
Sun Nov 22 15:05:12 IST 2020
Archive:  /tmp/bodek.KRNToq/intro2cs1/ex4/tsviel/final/submission
  inflating: src/hangman.py
4 passed tests out of 4 in test set named 'presubmit'.
result_code    presubmit    4    1
--> BEGIN TEST INFORMATION
Test name: main_double
Module tested: hangman
Function call: main()
Expected return value: "F\nR\n('___', [], 2)\n('a__', [], 2)\n('a_c', [], 2)\n('abc', [], 2)\nP\nR\n('___', [], 2)\n('___',
More test options: {'setup': 'double'}
--> END TEST INFORMATION
**********************************************************************
********************     There is a problem:
********************     The test named 'main_double' failed.
**********************************************************************
Wrong result, input: []:
expected: "F\nR\n('___', [], 2)\n('a__', [], 2)\n('a_c', [], 2)\n('abc', [], 2)\nP\nR\n('___', [], 2)\n('___', ['a'], 1)\n('
actual:   "F\nR\n('___', [], 2)\n('a__', [], 2)\n('a_c', [], 2)\n('abc', [], 2)\nP\nR\n('___', [], 2)\n('___', ['a'], 1)\n('
result_code    main_double    wrong    1
4 passed tests out of 5 in test set named 'main'.
result_code    main    4    1
9 passed tests out of 9 in test set named 'single'.
result_code    single    9    1
9 passed tests out of 9 in test set named 'hints'.
result_code    hints    9    1
--> BEGIN TEST INFORMATION
Test name: update_1
Module tested: hangman
Function call: update_word_pattern('apple','___l_','p')
Expected return value: '_ppl_'
More test options: {}
--> END TEST INFORMATION
**********************************************************************
********************     There is a problem:
********************     The test named 'update_1' failed.
**********************************************************************
Wrong result, input: ['apple', '___l_', 'p']:
expected: '_ppl_'
actual:   '___l_'
result_code    update_1    wrong    1
--> BEGIN TEST INFORMATION
Test name: update_3
Module tested: hangman
Function call: update_word_pattern('banana','b_n_n_','a')
Expected return value: 'banana'
More test options: {}
--> END TEST INFORMATION
**********************************************************************
********************     There is a problem:
********************     The test named 'update_3' failed.
**********************************************************************
Wrong result, input: ['banana', 'b_n_n_', 'a']:
expected: 'banana'
actual:   'b_n_n_'
result_code    update_3    wrong    1
--> BEGIN TEST INFORMATION
Test name: update_4
```

```
Module tested: hangman
Function call: update_word_pattern('banana','______','b')
Expected return value: 'b_____'
More test options: {}
--> END TEST INFORMATION
**********************************************************************
********************     There is a problem:
********************     The test named 'update_4' failed.
**********************************************************************
Wrong result, input: ['banana', '______', 'b']:
expected: 'b_____'
actual:   '______'
result_code    update_4    wrong    1
--> BEGIN TEST INFORMATION
Test name: update_8
Module tested: hangman
Function call: update_word_pattern('zzzz','____','z')
Expected return value: 'zzzz'
More test options: {}
--> END TEST INFORMATION
**********************************************************************
********************     There is a problem:
********************     The test named 'update_8' failed.
**********************************************************************
Wrong result, input: ['zzzz', '____', 'z']:
expected: 'zzzz'
actual:   '____'
result_code    update_8    wrong    1
4 passed tests out of 8 in test set named 'update'.
result_code    update    4    1
6 passed tests out of 6 in test set named 'filter'.
result_code    filter    6    1
TESTING COMPLETED
```

2 hangman.py

```python
#################################################################
# FILE : hangman.py
# WRITER : TSVIEL ZAIKMAN , Tsviel , 208241133
# EXERCISE : intro2cs2 ex4 2020
# DESCRIPTION: A simple game of Hangman
# NOTES: There are some additional supporting functions
# I had to write for the main functions to work
#################################################################

from hangman_helper import *

# Welcome Message before the single game
WELCOME = "Welcome to Hangman"
# Message handler for invalid letter case
INVALID_LETTER = "Invalid letter"
# Kept for legacy purposes
NOT_SUPPORTED = "not supported"
# Used letter message handler
USED_LETTER = "You had already made this guess mate. try another letter"
GOOD_JOB = "You are right, this letter is indeed part of the word, now try " \
           "another letter"
WRONG_GUESS = "Wrong Guess, please try another guess"
WIN = "Game Over, You Win this round"
LOSE = "Game Over, You lose"
BLANK = "_"
TURN = "Its your turn mate. Live or die, make your choice"


def split(string):
    """
    This helper function splits a string into a list of letters
    :param string: any string
    :return: an ordered list of letters consisting from the letters of the
    string
    """
    return [letter for letter in string]


def merge(lst):
    """
    This helper function recieves a list of letter and conjoin tham to string
    :param lst: a lst of letters
    :return: a string consisting of the lst (a word)
    """
    return "".join(lst)


def generate_new_pattern(word):
    """
    :param word: a given word represented by a string
    :return: a string of n BLANKS when n == len(word)
    """
    return merge([BLANK for i in range(len(split(word)))])


def is_input_valid(string):
    """
    :param string: a string given by the user's input
    :return: True if the input is Valid, False if not
    """
    if len(string) > 1 or not string.isalpha() or string.isupper():
        return False
    return True


def update_word_pattern(letter, pattern, word):
    """The function updates a received pattern with the letter if it exist in
    the word of the current single game"""
    word_lst = split(word)
    pattern_lst = split(pattern)
    for i in range(len(word_lst)):
        if word_lst[i] == letter:
            pattern_lst[i] = letter
    return merge(pattern_lst)


def chosen_before(letter, wrong_guess_lst, pattern):
    """
    :param letter: The letter we are running check on
    :param wrong_guess_lst: The list of wrong letters
    :param pattern: The current word pattern
    :return: True if had been chosen before, False if not
    """
    if letter in wrong_guess_lst:
        return True
    if letter in pattern:
        return True
    return False


def do_letter(letter, wrong_guess_lst, word, pattern):
    """
    :param letter: The letter we are checking
    :param wrong_guess_lst: a list of wrong guesses of letters
    :param word: a string representing the word of the current round
    :param pattern: a string representing the pattern
    :return: a tuple consisting of the new wrong_guess_lst,
    new pattern and the score we wish to add to the participant
    """
    if letter in word:
        pattern = update_word_pattern(letter, pattern, word)
        n = word.count(letter)
        score = (n * (n + 1)) // 2
    else:
        wrong_guess_lst.append(letter)
        score = 0
    return wrong_guess_lst, pattern, score


def do_word(word_guess, word, pattern):
    """
    :param word_guess: A user's guess for the word represented by a string
    :param word: the real word represented by a string
    :param pattern: the current pattern represented by a string
    :return: the score we wish to add to the player
    """
    if word_guess == word:
        n = pattern.count(BLANK)
        score = (n * (n + 1)) // 2
        pattern = word_guess
    else:
        score = 0  # Add a Neutral number to the score if word is wrong
    return pattern, score


def similar_pattern(word, pattern):
    """
    Filter words with exposed letters in different indexes than the origin pat
    Also Filter words that the exposed letters are different
    :param word: A string representing a word
    :param pattern: A string representing the current pattern
    :return: True if the pattern and word are similar, False if not
    """
    for i in range(len(pattern)):
        if pattern[i] == BLANK:
            continue

    exposed_letters = []

    for i in range(len(pattern)):
        if pattern[i] == BLANK:
            continue

        exposed_letters.append(pattern[i])

        if pattern[i] != word[i]:
            return False

    for i in range(len(pattern)):
        if pattern[i] != BLANK:
            continue

        if word[i] in exposed_letters:
            return False

    return True


def intersects_wrong_guess_lst(word, wrong_guess_lst):
    """
    Checks if a given word has letter that is already in the wrong guess list
    :param word: a string representing a letter
    :param wrong_guess_lst: a list of wrong letters
    :return: True if word has letter in the wrong guess list, false if not
    """
    for letter in word:
        if letter in wrong_guess_lst:
            return True
    return False


def filter_words_list(words, pattern, wrong_guess_lst):
    """Returns a list that consists from words that may fit the hidden word"""
    output = []
    for word in words:
        if len(word) != len(pattern):
            continue
        if not similar_pattern(word, pattern):
            continue
        if intersects_wrong_guess_lst(word, wrong_guess_lst):
            continue

        output.append(word)

    return output


def do_hint(words, pattern, wrong_guess_lst, score):
    """
    :return:
    """
    hint_lst = filter_words_list(words, pattern, wrong_guess_lst)
    if len(hint_lst) > HINT_LENGTH:
        hint_lst_s = [hint_lst[i] for i in range(len(hint_lst))]
        hint_lst = hint_lst_s
        filtered_list = []  # Sliced filtered list
        for i in range(HINT_LENGTH):
            index = i * len(hint_lst) // HINT_LENGTH
            filtered_list.append(hint_lst[index])
        hint_lst = filtered_list
    show_suggestions(hint_lst)
    display_state(pattern, wrong_guess_lst, score, "")


def run_single_game(words_list, score):
    """Function running a single game of hangman"""
    word = get_random_word(words_list)  # Generates new random word
    pattern = generate_new_pattern(word)  # Generates new pattern for the word
    wrong_guess_lst = []  # A list holding wrong guesses of letters
    display_state(pattern, wrong_guess_lst, score, WELCOME)
    while True:
        if score <= 0 or BLANK not in pattern:
            break
        action = get_input()  # Get users input
        input_value = action[1]
        if action[0] == LETTER:  # Letter Menu Option
            if not is_input_valid(input_value):
                display_state(pattern, wrong_guess_lst, score, INVALID_LETTER)
                continue
            if chosen_before(input_value, wrong_guess_lst, pattern):
                display_state(pattern, wrong_guess_lst, score, USED_LETTER)
                continue
            score -= 1
            letter_res = do_letter(input_value, wrong_guess_lst, word, pattern)
            wrong_guess_lst, pattern = letter_res[0], letter_res[1]
            score += letter_res[2]
            display_state(pattern, wrong_guess_lst, score, "")
        elif action[0] == WORD:  # Word Menu Option
            score -= 1  # Take 1 point from the player in any case
            word_res = do_word(input_value, word, pattern)
            pattern = word_res[0]
            score += word_res[1]  # Update the score
            display_state(pattern, wrong_guess_lst, score, "")
        elif action[0] == HINT:  # Initiate Hint
            score -= 1
            do_hint(words_list, pattern, wrong_guess_lst, score)
    return score


def main():
    words_lst = load_words()
    game_counter = 0
    score = run_single_game(words_lst, POINTS_INITIAL)
    game_counter += 1
    while score > 0:
        msg = "So far you played " + str(game_counter) + " Games "
        msg += "and earned " + str(score) + " points!"
        msg += "\n Do you want to play another one?"
        if not play_again(msg):
            break
        score = run_single_game(words_lst, score)
        game_counter += 1
    else:
        msg = "You survived" + str(game_counter) + "Games."
        msg += "\n Do you want to play another one?"
        play_again(msg)


if __name__ == '__main__':
    main()
```