Lab 9 - Hangman

Jamie Coyle

October 23, 2015

Synopsis of Program

This is a one player game of hangman. The words are randomly picked from a list of preset words. There is a graphic to let the player know how many lives that they have left.

Input

The letters that the player guesses.

Output

A won or lost game.
If won, a 'You Won' message is displayed.
If lost, a graphic of a hanged man is displayed with a message saying 'You Loose'.

Assumptions

- The algorithm assumes a one player game.
- Only one game can be played at a time.
- The algorithm assumes that there is a text file called dictionary.txt in the same directory as the python file.
- The algorithm only allows single letters guessed at a time.

Steps

1. Generate a random word from a preset list of words, from a text file.
2. Put this word into a list for checking.

3. Create an initial amount of starting lives.
4. Display the gallows graphic.
5. Display the rules.
6. Create a string of stars, to let the player know how close they are to guessing the answer.
7. If the player has some lives remaining, ask them for a letter.
8. Check if the letter has not been guessed before.
9. If it has been guessed, display all incorrect guesses in alphabetical order, and let the player know that they have not lost a life.
10. If the letter has not been guessed, and it is in the list of correct letters, inform them of this and show them what they have guessed. Replace the corresponding star, or stars, with the guessed letter.
11. If a letter was guessed correctly, allow user to guess the full word if they want.
12. If the letter has not been guessed, and it is not in the list of correct letters, inform them of this, let them know they have lost a life, and display the gallows graphic. Put the letter into a list of bad guesses.
13. Repeat steps 7-11 until the player has guessed the word, or is out of lives.
14. In the lives are exhausted, print the gallows graphic, and inform the player that they have lost.

Functions

- *generate_word*, used in Step 1, returns a random word in a from a file. It takes the words out of the file, puts them into a list, counting the number of them as they go in. A random number is generated to pick a word with more then 5 letters from the list.
- *generate_answer_list(word)*, used in Step 2, takes the randomly generated word, and returns a list, where each element of the list, is a letter of the input word.
- *graphic(lives)* inputs the current number of lives. It prints out the gallows graphic, depending on how many lives are remaining. It is called in Step 4, and is always called after the *wrong_guess* function is called.
- *rules()*, called in Step 5, and displays the rules of the game.

- *initial_guess_state(x)*, used is Step 6, inputs the length of the word, and returns a string of stars, with the length the same as the length of the word.
- *user_guess*, used in Step 7, asks the player for a letter, and returns the lower case counterpart of the guessed letter.
- *repeat_guess(letter, guess_state, bad_guesses*, used in Step 9, inputs the guessed letter, the current guess state, and the list of bad guesses. It inform the player that they have already guessed the letter, and informs them of what the current guess state is and the list of incorrect guesses. It reassures them that they did not loose a life.
- *right_guess(letter, guess_state, answer_list)*, used in Step 10, takes the input of the guessed letter, the string of stars and the list of correct letters. It informs the player that they were correct, and then loops through the list of correct letters, and when the guessed letter matches with the index of the correct letter list, replaces the corresponding star with the letter. It prints out the current iteration of the guess state, and returns to the main algorithm with this guess state.
- *wrong_guess(lives, letter, bad_guesses)*, used in Step 11, takes the input of the player's lives, the guessed letter, and a list of incorrect guesses. The function reduces the number of lives by one, appends the letter to the list of bad guesses, and prints out an alphabetized list of the wrong guesses. It returns to the main algorithm with the current number of lives.
- *guess_full_word(answer)*, used in Step 12, asks the user for a word, and checks it against the function parameter, answer. If correct, the game ends. If not, the game continues without the loss of a life.