Tech Basics I: Hangman

Major Digital Media

Louis Rohrbach
louis.rohrbach@stud.leuphana.de
Leuphana University Lüneburg
Matriculation Number: 3037215

Course: Tech Basics I Stream A
SoSe 2019
Helena Lingor

Table of Contents

Introduction

Motivation

This essay will be the description of how I programmed the game Hangman from scratch. I wanted to program a game because of the fun factor in playing it. I also thought it was challenging to implement certain rules and work directly with user input, to give out an answer that the user can see. I also promote the instant feedback of a game, telling the user if he lost or won. Since there is a traditionally manual way to play the game, because it already existed, the aim of the program is to imitate it as accurately as possible. The traditional game needs another person that draws and thinks of random words. I wanted to create the game, with the computer putting in the letters, and also drawing the hanging stickman, while choosing words at random.

Outline and Description

The interaction starts with getting to know the user and then starting the game. The Hangman game is programmed, so that it works in in the IDE terminal directly. Hangman is a pretty simple game, where there are secret words that have to be guessed. The goal of the game is to guess the word correctly, by putting in single letters to find out the word. There are rules to be followed while guessing the letters. At the same time, there are a limited amount of tries the user has before the game is lost. The maximum guess count set in this program is eight, drawing a line per wrong guess. The game itself is pretty fast and can be repeated multiple times, with different words that can be chosen easily. The words are stored in a .txt file which is read by the program every time the game is started. This .txt and the words can be easily interchangeable, and since .txt is a simple format, can be replaced by other wordlists.

If the user guesses all letters of the word or the word itself correctly without running out of guesses, the user wins. If they run out of guesses, the game is lost. The program reveals the word at the end in case of loss, and indicates the outcome of the game.

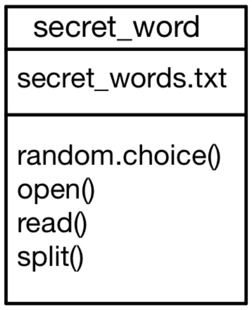
Main Structure and Functionalities

The main structure of the program begins with a user input regarding the name of the player. This is done to personalize the game. The variable “name” is set with the user input after the prompt “Hey, what’s your name?”. While the name is entered in the beginning, it gets used throughout the application and at the end to call the user. This is a basic input that gets assigned to a variable. For design-purposes, a lot of “\n”’s were used throughout the code, separating lines to make it more visually pleasing when playing.

Since the game is based on rules, the second function asks for another user input, this time a “yes” or “no”, regarding if the user knows the rules of the game hangman. The variable “knows_rules” is assigned with the answer. An if-function is introduced to check the answer of the user input. If “YES” is answered, meaning the user knows the rules, the game begins. “NO” prompts a short explanation of the rules after an elif-function checks the answer, and then starts the game. If the user answers a different thing than “yes” or “no” the game quits, saying the rules have to be followed, with an else function that checks the initial answer. Since there is multiple ways to write “yes” like “Yes” and “YES”, a method called upper() which translates the “knows_rules” variable into all uppercase letters is brought in. The if-function is set to check for an uppercase “YES” which, no matter how the user actually writes “yes”, is given. The same applies for the elif-function asking for a “NO”. This is the introductory sequence that asks for the users name and checks for the knowledge of the ruleset.

Hangman begins now, starting with another variable. The variable “secret_word” is the word that has to be guessed throughout the game. This word is pulled from a .txt file, the word database. It is important, that only one word is chosen and that the chosen word is random from the list. Otherwise the chosen word would always be the first one from the list. The .txt file contains the words in uppercase letters, with a new word in each line. To set the variable the methods “secret_word = (random.choice(open(“secret_words.txt”).read().split()))"[1] are used. In the beginning of the code, the “random” module has to be imported with “import random”, so that the program knows to pick a random word, using “random.choice” (see Figure 1). The code has to be telling the program to open the file with “open()”, to actually read the file with “read()” and to split the list of strings in the .txt with “split()”. Setting this up allows the program to choose a word from a file, that has a split list of strings, at random. Since it is desirable to use this secret word chosen out of the list in the upcoming code, it is important to assign another variable. Otherwise, every time the variable “secret_word” would be used, the random word generation would occur, ultimately changing the word and confusing the game.

Figure 1



This is why “answer = secret_word” is chosen as a variable that carries the answer word per individual game started.

The program prints a phrase to introduce the word. Since the game’s purpose is to create a hidden word with the user having to guess it, it is necessary to encrypt the randomly chosen word, to show the amount of letters. It would be possible to just print out the number of letters in a word, to simplify the code, but this would not be visually pleasing nor would it be effective in later parts of the game. The user will also randomly guess letters, which could be in any spot of the secret word. In result of that, the word and its letters have to be visualized, to show the correct positions of the already-guessed letters. A list element called “underscores” is necessary. This list is empty at first, but is extended with “underscores.extend(answer)” to contain the secret word. It has to be created empty, so that the word, after being randomly chosen, can always be added to the list. This prevents the variable “secret_word” of staying the same word for every game and allows the variable “answer”, which is used throughout the code, to be assigned.

[1] This source code line, in this version was found accessing: https://stackoverflow.com/questions/32773275/pulling-a-random-word-string-from-a-line-in-a-text-file-in-python

Since the word is a different word every time the program runs, the actual underscores that will be displayed to the player vary in length, hence it's required to check the length of the word every time the program runs from the beginning. The code "for i in range(len(underscores)):
underscores[i] = "_"" checks for the amount of letters, in this case, the length of the "underscores" list. The range of the list turns the list elements into an integer and the "for i in" command uses the number of letters, and uses that number to determine the amount of actual underscores needed for the display, that the user eventually sees. To clean it up a bit and make it more visual, the "join()" function is used to add a space after each underscore, ultimately printing it out for the player.

In a nutshell, the code up until now introduces the game, asks for a name, and chooses a random word out of a .txt file to turn it into underscores and display it to the user. The point "underscores" in Figure 2 is reached.

Furthermore, the actual ruleset has to be applied to turn the now-displayed underscores into a game that can be played. The ruleset gives a set amount of tries or wrong guesses to be made, while also counting the correct guesses. To achieve this, the program has to have set variables that would start at 0 in the beginning of the game. Chosen variables are "correct_letters", "number_of_tries" and "wrong_guesses", all being set to 0 to start with. The ruleset itself consists out of a constant while-loop with if-functions embedded. The while-loop is necessary to check if the player has already guessed the word after a few tries. It checks if the number of correct letters is lower than the number of the length of the answer (number of letters) with "correct_letters < len(answer):". Since the variable "correct_letters" has previously been set to start at 0, the while loop initiates the next functions. The secret word that is stored in the variable "answer" has to be more than 0 letters long, which makes sense. The program prompts a user input to store as a first variable: "guess"; the first letter can be guessed. While the words in the secret_words.txt are all stored in uppercase letters, the user input of the letter has to be translated to an uppercase letter with the same function used in the beginning of the code, ".upper()". Every time the user makes a guess, the variable “number_of_tries" increases by one. This is useful for the end to show the user the amount of tries used for the guessing of the word.

The first function inside the while loop is an if-function that determines the amount of wrong guesses, which is crucial to setting a limit of tries to the game. When the game is started, the wrong guesses are set to 0, because nothing has been guessed wrongly. The function checks if the input that is stored in variable “guess” is not contained in the set variable “answer”, to increase the number of wrong guesses by one. This only applies when the input is not part of the answer. Provided that the “answer” variable is a random string, the next function checks every letter of the answer corresponding to the guess. Using “for i in range(len(answer)):” determines the amount of times, which is the number of letters in the answer, the next if-function is carried out. The code checks if the guess is identical to an element in the answer variable, and if it is, replaces that exact element in the list of “underscores”. Since the underscores are displayed and changed, the list of underscores joined by spaces is now altered with a correctly guessed letter. At the end of the while-loop, the “underscores” list, which is possibly altered by a correct letter, is printed out, showing the progress of the overall guessing-game. This also increases the variable “correct_letters” by one. The while-function, as previously mentioned, starts again, checking if the number that is assigned to the variable “correct_letters” is smaller than the length of the answer. When all the letters of the words are guessed correctly, the while-function does not initiate anymore, because of the internal function that increases the “correct_letters”, until it is the same length of the number of letters in the answer.
A second if-function inside the while-loop, which controls the duration of the game, is implemented for guessing the whole word. If, at any point, the user is ready to correctly guess the entire word, this if-function checks if it is identical to the variable “secret_word” and prints out a winning statement, when correct. This also shows the number of tries needed to finish the game to the player and quits the application.

Given that this game is called hangman, and not a regular word guessing game, a hangman figure is drawn by the program to create a visual feedback of the progress of the game.

Inside the while-loop are eight if-loops that monitor the amount of wrong guesses. Mentioned above, the while loop increases the variable “wrong_guesses” by one every time a guess input does not match an element in the “answer” variable. For each different scenario of increasing wrong guesses, the stickman hanging is drawn one by one. One part, e.g. the head or an arm, is drawn for each wrong guess. Because this game is inside the IDE terminal, there was no real GUI that could be used to actually use png or jpeg files to create a hanging stickman. Therefore the stickman is drawn from special symbols, spaces and new lines. The last function, which triggers when the amount of maximum guesses, eight in this case, is reached is responsible for indicating the loss. It prints out the number of guesses used, reveals the secret word and declares, that the player has lost the game, before quitting it completely.

When the number of correctly guessed letters is the same as the length of the word, a last if-function outside of the while-loop prints out the winning message, also indicating the needed tries and the final word. Meaning that if all the letters in the word have been correctly guessed, the game is won.

Figure 2

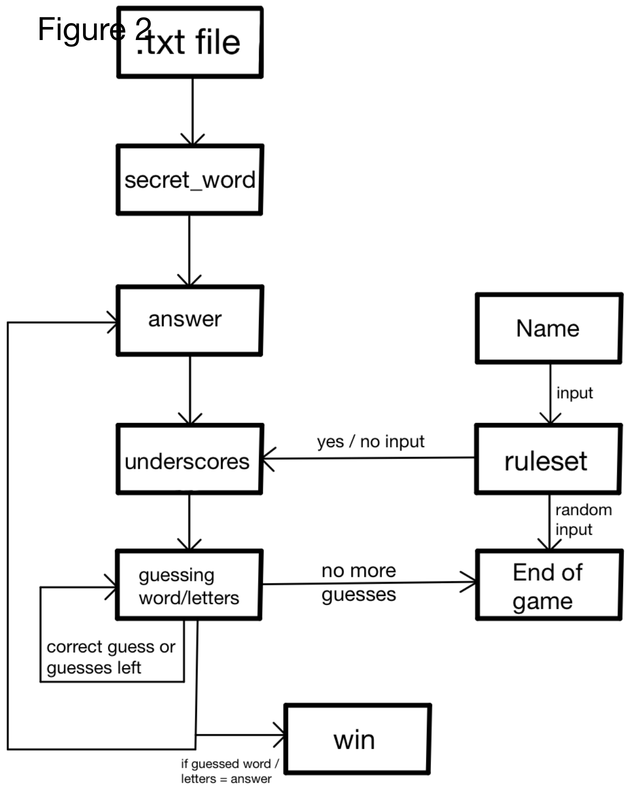


Figure 2 shows the complete process of the program in an activity diagram, starting with extracting the answer word from the secret_word that was randomly chosen from the .txt file, while simultaneously asking for the first two inputs of the user about the name and the ruleset.

Feasibility Report

This project was the final assessment for the module Tech Basics I. The task at hand was to ideate and conceptualize a small application in Python. The project chosen was a small IDE terminal based game, called hangman. The game is already existent in other cases. The goal of this project was to find out how the game operates, from the perspective of a developer, not a user. The game is fully playable and offers an easily modifiable database in form of a .txt file. The main skeleton of the program is also shown in Figure 3. Once the game has finished, the program has to be started again to play the game repeatedly.

The presentation of the game was heavily influenced by the traditional manual way to play the game with a pen and a paper. It was aimed to duplicate the same experience a user might have, but without needing a second person thinking of random words.

Since this task was only 100 lines of code, there were a few functions that were left out, that were desirable to have. A counter of the letters that were previously input by the user that would show with the hangman drawing would be one of those functions.

After troubleshooting and bug searching, one bug has presented itself. This bug contains the counting method of the winning scenario. While the function looks for the variable that represents the amount of correct letters guessed it determines whether or not the game is won or not. Since there is no mechanism built in that prevents the user from using duplicate entries, the game counts each individual entry. If a duplicate entry is not part of the answer, it will still increase the "wrong_guesses" counter by one, even though previously used. This applies to the correct guesses as well, meaning that if the word is four letters long and two of them are an "E", the game would finish after guessing an "E" again, because it only counts the number of correct guesses. The first guess would raise this number by two, and so would the second. This would indicate four correct guesses, which would trigger the ending scenario, because the amount of correct guesses represents the amount of letters in the word.

This bug only appears if the user guesses an already correctly guessed letter. If the user follows the rules explained in the beginning of the game, this bug can be negated. Otherwise, the program fully runs, is effective and practical.

All in all, the projects propose was to create a concept of a program in Python, which was successfully done. The game truly represents a version of the manual, traditional game which is played on pen and paper, with another person. If the rules stated in the game are followed, the game uses visual clues to elude the user of their progress and uses a GUI and database.

Bibliography, Figures and Necessary Data

The module “import random” is used to utilize the function “random.choice()”

1 This source code line, in this version was found accessing: https://stackoverflow.com/questions/32773275/pulling-a-random-word-string-from-a-line-in-a-text-file-in-python

The .txt file database that was used to import the random word can be modified freely. The one used for testing the game contained:

```
GIRAFFE
NECK
MONKEY
TABLE
ZEBRA
HAIR
OCTOPUS
ANNIHILATION
JUDGEMENT
BEER
MICROFIBER
ACRYLIC
LAPTOP
ASSESSMENT
CODE
TROMBONE
MASSAGE
NINTENDO
```

It is important for the code to work, that the letters are written beneath each other in uppercase letters and that the file is named secret_words.txt

The Figures are drawn by myself.

Figure 1:

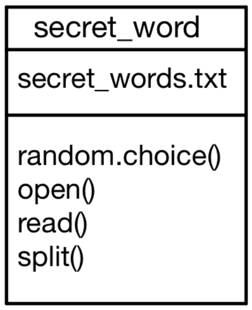


Figure 2:

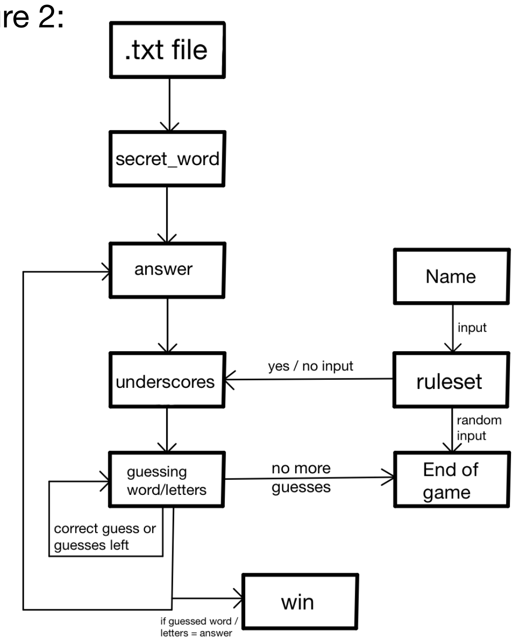
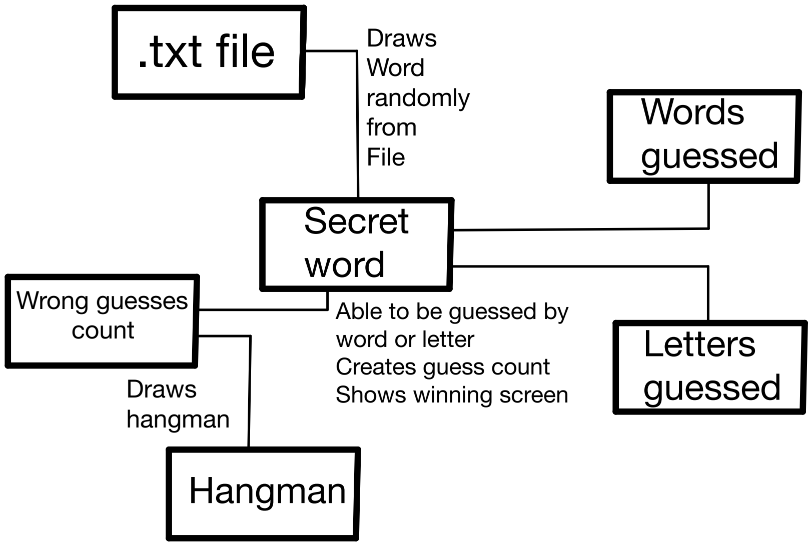


Figure 3:



Statutory Decalaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

[illegible]

Louis Rohrbach