Hangman Game

Deividas Ovsianikovas, 2nd year BSc Computer Science

19358376

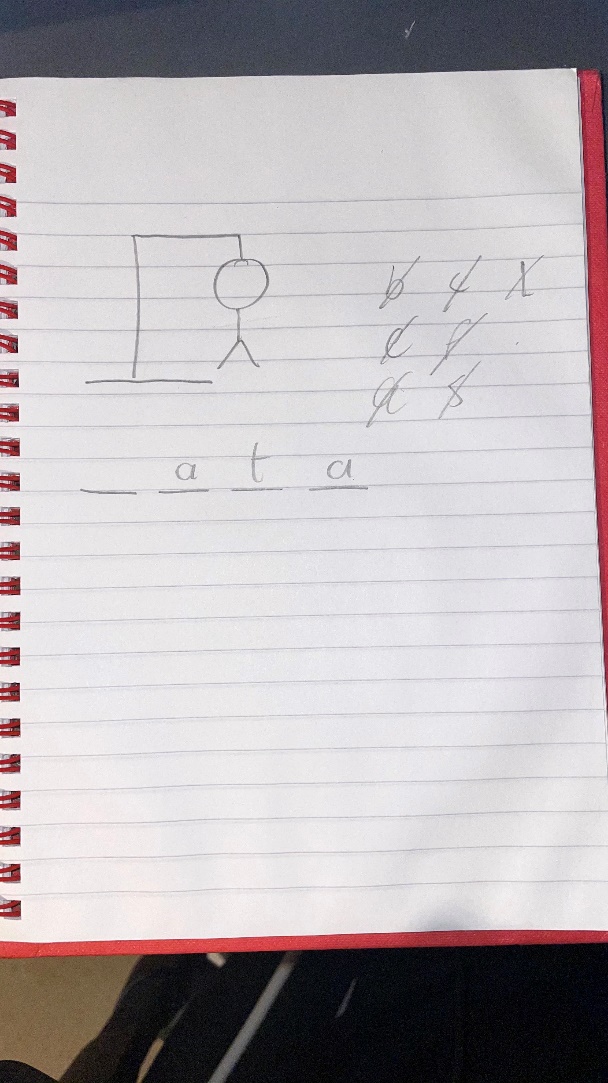
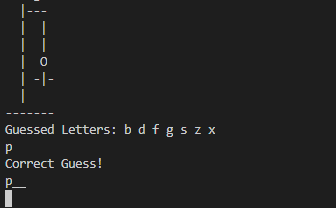
Maynooth University

CS210

**Overview of game**

While its origins are unknown, the original Hangman game has been enjoyed by millions around the globe for decades. In the classic game of Hangman, one player thinks of a [word](https://en.wikipedia.org/wiki/Word), [phrase](https://en.wikipedia.org/wiki/Phrase) or [sentence](https://en.wikipedia.org/wiki/Sentence_(linguistics)) and the other player(s) tries to guess it by suggesting [letters](https://en.wikipedia.org/wiki/Letter_(alphabet)) within a certain number of guesses. For every incorrect guess, an extra dash is added to progress the sketch of a Hangman stick-figure.

In this project I wanted to challenge myself and develop Hangman into a version playable from a Java file making it as interactive as possible within a Java Command Prompt. In my version, the player must figure out a word that has been chosen randomly by the computer based on the selected difficulty by guessing it letter by letter. The player has 8 lives and for every incorrect guess they make they lose a life. The player can win by guessing all the correct letters of the word but lose if they run out of lives. The player’s current health is shown by an ascii representation of the Hangman stick-figure.



**How to play**

1. Run the “Hangman.Java” file in your Integrated Development Environment (IDE).
2. Select whether you would like to see the rules or not by inputting “Y” or “N”.
3. Choose difficulty by inputting the correlating word (Easy, Medium, Hard).
4. The AI then chooses a word to guess based on selected difficulty and you will be presented with the length of the word and the state of your current guessed word which will be represented with the “\_” character representing all characters of the original word.
5. You can now guess the letters contained in the word by typing in your chosen character and hitting enter (Only alphabetical characters are allowed). If the character is incorrect, you lose a life and are presented with a Hangman stick-figure step representing your health. If you guess correctly, your current guessed word gets updated with the guessed character and you can continue guessing more characters.
6. If all characters of the word are guessed correctly you win. If not, you will be presented with the full Hangman and lose.
7. Once complete, you are given the choice to either retry or quit by typing in the respective word. If quit is chosen, then the program terminates. If retry is chosen, the program restarts from the difficulty select step.

**Code structure**

1. **Imports**

Importing Java.utils to use the “Random” and “Scanner” objects, and importing java.io to use the “File”,”StringBuffer”, and “BufferedReader” classes.

1. **Public Class Hangman**
2. **Static variables**

Private and public variables used throughout the program.

1. **void Main()**

The initial method called by the Java program which calls the Start() method.

1. **void Start()**

This method calls the ResetVars() method, introduces the player, allows them to choose a difficulty where GameSetup() is called depending on it, allows the player to guess until the game is over by calling the Guess() method, displays the end game result to the player and also calls the RetrySetup() method.

1. **void GameSetup(int difficulty)**

This sorts the words in the words.txt file based on their length, smallest to largest and sets the position of where the present easy, medium, and hard length words start. It then chooses a word from the sorted array based on the selected difficulty and displays to the user the length of the word and the state of their current guessed word (which will now just be “\_” characters to hide the actual characters of the word)

1. **void Guess(String guessedChar)**

This handles the guessing aspect of the game allowing the input of characters for guessing and returns a congratulatory message if the correct character is guessed with the new state of their guessed word, the state of their hangman if an incorrect guess is made or an error message to the player if an invalid input is given. Also checks if the game is over and manages the players health.

1. **void ShowLives(int health)**

This prints out the current Hangman stick-figure step the player is at based on their health.

1. **void RetrySetup()**

This asks the player if they would like to restart the game or quit. If they’d like to restart, the Start() method is called and the process begins again. If they’d like to quit, the method returns, and the program ends.

1. **void ResetVars()**

This method resets selected variables to their default state.

1. **boolean YesNoAnswer()**

I wrote this method so I would not have to write the same if statements that check if the player player inputted yes or no. It returns true if the player had inputted “yes”, false if “no”, and an error prompt if the input is invalid.

1. **void MergeAlgo(int p, int q, int r)**

This is the Merge Sort algorithm that sorts the words array from smallest to largest.

1. **void Sort(int p, in r)**

This accommodates the MergeAlgo() sorting algorithm splitting the array into 2 and sorting each half until both are sorted.

1. **Class Dictionary**
2. **Public variable**

Declares the input[] array which will hold all of the words in the “words.txt” file.

1. **Dictionary()**

The constructor method where the load() method is called with a link to the “words.txt” directory.

1. **Int getSize()**

This method returns the amount of words contained in the input[] array.

1. **String getWord(int n)**

This returns the word located at position n in the input[] array.

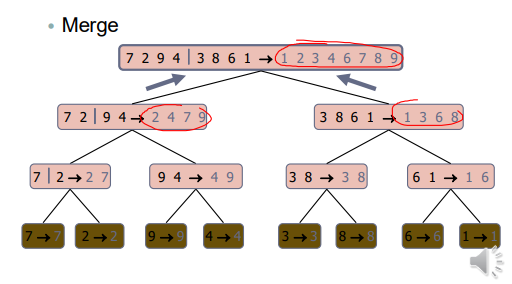
1. **String[] load(String file)**

This method handles extracting all of the data in the “words.txt” file and returning it to the input[] array. It handles any error exceptions that may occur while iterating through the data and also makes sure to only add words that are of only alphabetical characters. (i.e. no numbers, hyphens, dots, etc.)

1. **End of file**

**Data structures and algorithms used**

* The Merge Sort algorithm is used to sort all the words by length smallest to largest. Chosen as it had the best Big O notation (Big o = nlogn) of the previous learned algorithms.



* 2-Dimensional Array for containing the Hangman stick-figure.
* 2 1-Dimensional String arrays to hold all available words before and after sorting.
* Nested Loop to print out the Hangman stick-figure.
* For loops, While Loops for various iterative tasks.
* Random Method to select a random word in given range.

**Learning outcomes**

The main learning outcome of this project was to apply the knowledge that I learned during CS210 Semester 1 and first-year Computer Science. It allowed me to gain a better grasp of how to use the Algorithms & Data Structures and Data Structuring techniques that I learned in real world applications.

This project let me apply the knowledge I gained from Software Testing to test and debug the object-oriented Java program to ensure there was no presence of bugs, how to work with multiple classes in a Java project, how to import objects and use them in a Java file, and how to use Regex as a conditional.

I also learned of some efficiencies that can be built with helper methods which can be re-used numerous times without the need of having to write them out constantly. I learned this after realizing that I would have to re-write terms checking if the player had typed “y” or “n” for numerous statements without this method and it helped me cut down on around 30 lines of code.

ALLWORDS NOT HOLDING PREVIOUS VALUE

**Appendix**

Github: https://github.com/DE0S/Hangman