hangman

This project is based on the famous HangMan Game

You will need to create a private repository with the name `hangman-classic`

Notions

- Golang Documentation: ioutil
- Golang Documentation: rand
- Reading File Example

Instructions

Create a program `hangman` that will take a file as parameter. Create a file `words.txt` which contains a bunch of words with which the program will play. Each word is separated with a newline `\n`. Or you can find differents dictionnaries here

PART 1 —

You will have 10 attempts to complete the game.

- First, the program will randomly choose a word in the file.
- The programm will reveal `n` random letters in the word, where `n` is the `len(word) / 2 - 1`
- The program will read the standard input to suggest a letter.
- If the letter is not present, it will print an error message and the number of attempts decreases (10->9->...0)
- If the letter is present, it will reveal all the letters corresponding in the word.
- The program continues until the word is either found, or the numbers of attempts is 0.

PART 2 —

Let's name José the poor man that will be hanging to this rope if you lose.

You will be given a file named hangman.txt that contains all the position of José. This file contains 10 positions corresponding to the 10 attempts needed to complete the game.

You will need to parse this file and display the appropriate position of José as the count of attemps decreases.

Each position has :

- A height of 7 lines, ending with a new line (so 8)
- A length of 9 character, ending with a new line (so 10)

Here is an example of the seventh position of José :

```
  +---+
  |   |
  O   |
 /|   |
      |
      |
=========
```

Help

If you don't know how to handle the data of your program you can always try using a structure, and pointer to structure inside the parameters of your future functions.

This is just an example, you are free not to use it, to modify it or to create your own.

It is as you please and feel free to use what you are comfortable with to make this project.

```
type HangManData struct {
    Word             string // Word composed of '_', ex: H_ll_
    ToFind           string // Final word chosen by the program at the beginning. It is the word to fi
    Attempts         int // Number of attempts left
    HangmanPositions [10]string // It can be the array where the positions parsed in "hangman.txt" are
}
```

Good luck !

Allowed packages

- Only the standard go packages are allowed

Usage

```
 $> ./hangman words.txt
Good Luck, you have 10 attempts.
_ _ _ _ O

Choose: E
_ E _ _ O

Choose: A
Not present in the word, 9 attempts remaining









=========



Choose: L
_ E L L O

Choose: B
Not present in the word, 8 attempts remaining

      |
      |
      |
      |
      |
=========

Choose: Z
Not present in the word, 7 attempts remaining
  +---+
      |
      |
      |
      |
      |
=========

Choose: H
H E L L O

Congrats !
```

hangman-advanced-features

Objectives

You must follow the same principles as the first subject.

In hangman-advanced-features you can implement the following things:

- The player can suggest either a word or a letter. (A word is a string of at least two character long)
    - If the word is found the game stops
    - If not, the counter of attempts decrease by 2
- Stock the letters that are suggested by the player so the player can't propose the same letter twice.
    - You can display error message when this case happens.

Allowed packages

- Only the standard go packages are allowed

Instructions

- The code must respect the **good practices**.
- We suggest you to search for the principles of a good website design.