How this game works

Thank you for your curiosity into how this game works! “Hatman” is very similar to the popular game “Hangman”, except in this version, no one gets hurt :)

I created this game to introduce R in a fun way to those who are interested in coding and also practice my skills in regular expression and text analysis. It is designed to be easy to play for R beginners while also allowing more advanced players to go into the source code directly to see specifically how this program runs. This game touches upon the concepts of regular expressions, text parsing, conditional statements, and plotting simple shapes. Keep in mind there are many ways one could create this game in R; here, I present what made the most sense to me!

This document broadly outlines the structure of the R code and the logic R follows to play Hatman. To understand the details of the game, you can read the R source code (in the “code” folder) or contacting me (chris.j.blackford@gmail.com or github.com/Christopher-Blackford). You may use this code for any projects you like but please contact me if you do so I know it has been useful!

General steps taken

The game executes the following steps to play:

1. Create a word list by either a) choose a word from a predefined word list or b) parse a pdf given by the user, then choose a word from that word list.
2. Plot the game board based on the number of letters in the chosen word.
3. Defines a “guess” function that will take the players input and search if that letter exists anywhere in the word.
4. Player guesses a letter. If the letter exists in the word, plot that letter on the game board. If it doesn’t exist, add to a mistake counter and plot part of Hatman.
5. Repeat step 4 until the player guesses all letters correctly (wins) or guessed wrong seven times (lose)

See below for more details on each of these steps. When discussing R code, all sub-files the Hatman.R file calls on can be found in the “./code” directory.

Parsing a pdf to create a word list

The code relevant to this section is: “Generating_Word_List.R”. This was the most interesting part of the game to code.

When playing Hatman, the user can choose to pick from a word list that was pre-generated or upload a pdf to the "./word_files/pdf" folder in which case R will break down that pdf into individual words and draw from that list.

The pre-generated word list was generated by parsing a large free online dictionary (titled "easier_english_student_dictionary_upper-int.pdf" found at: https://www.easypacelearning.com/english-books/english-books-for-download-pdf/category/33-3-dictionaries-to-download-in-pdf)

Parsing the pdf took fewer lines to code than expected. The code is commented in detailed but to summarize, it separates the pdf into words, then goes through each word and removes words that have non letter characters (symbols, numbers etc.) in them. At the end of the code, R outputs a word list text file meaning that if you want to re-use that document, it skips this code and loads the word list text file instead (thus saving a lot of time.)

At the end of this process, the "Selecting_Word.R" file chooses one word out of the entire word list for you to guess. In the Selecting_Word.R code, notice how use set.seed() and the sample() function to avoid storing that word in the global environment and thus making it harder for cheaters ;)

Note: Another way to create the word list would be to download a list of English words online. However, this would still require some word processing to remove words with numbers in them (lots of chemicals) or hyphenated words, both of which aren't usually accepted in Hatman/Hangman. Since I wanted the user to be able to choose words based on a pdf they upload, I decided to just use this code to generate the default words as well.

Visualizing Hatman

The code responsible for the visualizing Hatman can be found in the "Hatman_Game_Board.R", and "Hatman_Letters.R" files.

I decided to use the plotting window to visualize the game board as well as the eponymous "Hatman". I felt this allowed for a prettier visualization than showing all the output in the console. I used the *sp* and *rgeos* packages to create simple shapes for each letter of the alphabet, the game board, and Hatman. Essentially, the plotting window becomes a virtual graph paper for plotting the game.

The number of blank lines on the game board depends on the number of letters in the word R chooses for the user to guess. Each letter was designed as a spatial lines object as if it would occupy the first blank line space (i.e the first letter of the word). If the letter exists in another location, the x-axis coordinates that define the letter get transformed so the letter moves to a subsequent blank line. Deciding where a letter gets put is all accomplished through mathematical transformations and "if" statements.

Guessing function

The code responsible for guessing each letter can be found in the "Guessing_Function.R" code.

If the letter guessed is present in the word, R runs the "Hangman_Letters.R" file to place the letter. R will also store that you got a letter right.

If the letter guess is not present in the word, R plots part of Hatman. R will also store that you got a letter wrong.

The game ends if you guessed all the letters correctly (you win) or if you guess wrong seven times (you lose).

Note: I hid a "reveal" function in the Guessing_Function.R code that allows the user to reveal the word if you end up losing but want to still see the word. I wanted to reward people who either read documentation or go into the sub-code.

To use the reveal function, simply run the code:

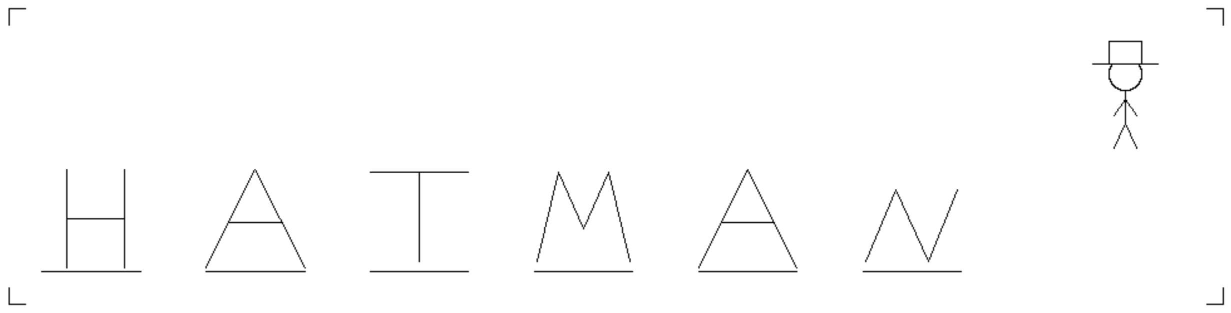
reveal()

Some caveats

1. This game only works for English words since it doesn't handle non-English characters or symbols well (e.g. ê, ö). It'd be a really neat expansion of the game if someone was able to have it recognize non-English characters.

2. The pdf parsing isn't perfect. In testing, I was able to get around 96% accuracy (i.e. the code identified a "word" that wasn't actually a real word 4% of the time) with the dictionary word list but there will be some cases where non-words will slip through the cracks.

3. Words in the margin of the user supplied pdf document are likely to be removed entirely.



Have fun!