Universidade Federal do Ceará

Aluno: Evandro Rodrigues de Paula Junior
Matricula: 402324
Curso: Ciência da Computação
Disciplina: Linguagens de Programação

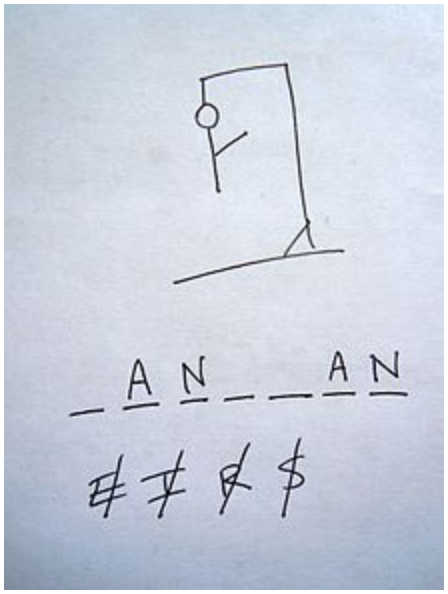
Introdução

No trabalho final da disciplina, resolvi escolher o paradigma Funcional pois já me era familiar. Após pensar em várias coisas pra fazer, decidi implementar um jogo da forca (também conhecido como Hangman Game), me baseando em algumas funções do Haskell já existentes e algumas dicas da literatura pela internet.

Ideia

Como dito anteriormente, o projeto escolhido foi um jogo da forca. O conceito do jogo, embora clássico, consiste da seguinte forma:

- O jogo começa com com uma palavra escolhida aleatoriamente e é expressa na forma de uma sequência de "_" onde cada repetição do mesmo é a quantidade de letras da palavra.
- O jogador ou jogadores começam a dar palpites em letras do que poderia estar na palavra e a medida que acertam (a letra pertence a palavra) a letra deve ser mostrada na posição, teoricamente, facilitando o desenvolver do jogo.
- **Por exemplo**: a letra "A" aparece três vezes em _A_A_A (BANANA)



Funções Usadas

hangmanMatrix: função que retorna uma matriz onde é armazenada o "desenho" do personagem usado no jogo da forca.

render: função que recebe um inteiro qualquer como parâmetro, que no caso é a quantidade de tentativas e retorna o personagem usando a função anterior.

maxErrors: função usada para calcular a quantidade de tentativas que o usuário pode ter, baseado no tamanho da matriz transposta (**hangmanMatrix**), ou seja, sempre será 6 tentativas.

showWord: função responsável por dar a dificuldade do jogo. Transforma caracteres de uma determinada palavra, dada por entrada, em "_". Por exemplo: "casa" -> "_ _ _ _".

tryLetter: função que recebe a palavra escolhida aleatoriamente, uma letra e o número de tentativas atual e verifica se a letra pertence a palavra, caso sim, a letra é revelada caso contrário o número de tentativas diminui em -1.

sortWord: função responsável por dar a aleatoriedade ao jogo. Aqui, a função recebe um número como parâmetro e através deste, seleciona um dicionário com palavras de um mesmo tema, em seguida, escolhe, aleatoriamente, uma palavra deste dicionário. Por exemplo, o dicionário "países" contém a palavra França, que pode ser selecionada aleatoriamente, dentre as outras opções.

game: função responsável pelo "game loop". A função recebe como parâmetro a palavra aleatória, escolhida de algum dos dicionários (que também é escolhido pelo usuário) e recebe também o número máximo de erros, que por padrão é 6.

main: função principal responsável por exibir o menu principal do jogo. Durante sua execução, ela recebe um número de 1 à 3 e seleciona o dicionário correspondente, em seguida, seleciona aleatoriamente uma palavra do mesmo e inicia a função de "game loop" juntamente com a quantidade máxima de erros -- **game** --.

ALGORITMO:

```
import System.IO
import System.Process
import System.Random (randomIO)
import Data.Char
import Data.List
import Data.List (transpose)
import DictPaths

hangmanMatrix :: [[String]]
hangmanMatrix =
   transpose
   [["   ", " O ", " O ", " O ", " O " , " O " , " O " ]
   ,["   ", "   ", " | ", " | ", " | " , "/| " , "/|\\" ]
   ,["   ", "   ", "   ", "/  ", "/ \\", "/ \\", "/ \\"]]

render :: Int -> [String]
render index =
   "----------" :
   "|    |" :
   map ("|   " ++) image
   where image = hangmanMatrix !! index

maxErrors :: Int
maxErrors = length hangmanMatrix - 1

showWord :: String -> String
showWord word = intersperse ' ' [if a `elem` ['a'..'z'] then '_'
                                 else a | a <- word]

tryLetter :: String -> Char -> Int -> IO()
tryLetter word letter tries
   | letter `elem` word = game [if letter == a then toUpper letter else a | a <- word] tries
   | otherwise = game word (tries - 1)

sortWord :: [Char] -> IO[Char]
sortWord selection = do
   dictionary <- readFile (dictWords selection)
   let words = filter validWord $ lines dictionary
   let numWords = length words
   randomNumber <- randomIO
   let randomWord = words !! (randomNumber `mod` numWords)
```

```
    return $ randomWord
    where
        validWord word =
            '\" `notElem` word &&
            map toLower word == word

game :: String -> Int -> IO()
game word tries
    | word == map toUpper word = do
        system("clear")
        putStrLn $ showWord word
        putStrLn "\nVOCE GANHOOOUUU! :DDD"
    | tries == 0 = do
        system("clear")
        putStrLn $ showWord word
        putStrLn $ unlines $ render(maxErrors - tries)
        putStrLn "\nvocê perdeu :( ..."
    | otherwise = do
        system("clear")
        putStrLn $ unlines $ render(maxErrors - tries)
        putStrLn $ "Você tem " ++ show tries ++ " tentativa(s) restantes."
        putStrLn $ showWord word
        putStrLn "Digite uma letra: "
        letter <- getLine
        tryLetter word (head letter) tries

main :: IO()
main = do
    system("clear")
    putStrLn "--------------------------------"
    putStrLn "|  Bem-Vindo ao Hangman Game!  |"
    putStrLn "--------------------------------"
    putStrLn "| Escolha um dicionário:       |"
    putStrLn "| 1. UFC                       |"
    putStrLn "| 2. Frutas                    |"
    putStrLn "| 3. Países                     |"
    putStrLn "--------------------------------"
    putStr ">>> "
    selection <- getLine
    word <- sortWord selection
    system("clear")
    game (map toLower word) maxErrors
    putStrLn "\nObrigado por jogar! Espero que tenha se divertido! ;D"
```