# The game "Soup of letters"

The game "soup of letters" is a pastime that is based on a tray of letters placed at random, where words are inscribed. The objective of the game is to find and surround the words hidden on the board. The words can be hidden vertically, horizontally or diagonally on the board, both in normal and inverted sequence.

Note the following example.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Y | L | C | Q | T | S | T |
| Y | P | O | R | T | O | O |
| M | L | I | P | X | O | X |
| Z | R | M | A | I | A | X |
| S | S | B | G | X | M | L |
| T | B | R | A | Z | V | N |
| D | F | A | R | O | F | W |
| R | F | R | B | E | A | H |

In this tutorial we will develop a simpler version, in which words are placed only horizontally.

# Game structure

This game already has some complexity, so it will be useful to use the problem decomposition to better structure the game into subproblems:

1. Create the board: in this subproblem, a matrix is constructed, where, at random, the words of a given predefined list are placed. In this case, we will only place the words horizontally, and they may be randomly placed normally or inverted. At the end of placing the words, it is necessary to fill in the vacant spaces with letters, at random.
2. Show the board: In this subproblem, the letter soup matrix is displayed.
3. Play: This subproblem will focus on the game loop, iterating over several moves until the player guesses all the words.

# Create the board

The board consists of a matrix, which is typically formed by a vector of vectors. In Python, we chose to implement it through a list of lines, each line being a string of characters, since it is a game of letters.

Function cria\_tabuleiro(linhas, colunas, lista\_palavras) has three parameters:

* **linhas**, the number of lines on the board (integer);
* **colunas**, the number of columns on the board (integer);
* **lista\_palavras**, a list of strings with the words of the game.

The algorithm consists of 3 steps:

1. Create an empty board;
2. Place the words on the board;
3. Fill in the blanks on the board with random letters.

## Create an empty board

In this step of the algorithm, an empty list is created that is filled with character strings, which in turn are filled with spaces, equal to the number of columns.

def cria\_tabuleiro(linhas, colunas, lista\_palavras):

# criar tabuleiro vazio

tabuleiro = []

for i in range(linhas):

linha = ""

for j in range (colunas):

linha += " "

tabuleiro += [linha]

|  |  |
| --- | --- |
| **Eye** | In this code the operator += does the concatenation of strings, joining spaces. It also concatenates lists, in order to add a new line in each iteration of the for loop with the variable i. |

## Place the words on the board;

In this step of the algorithm, being random, it may not be possible to put all the words on the board, as it depends on the “arrangement” of them. For this placement we will have a maximum of 50 attempts. If it is not possible to create the board, the function returns False.

# coloca palavras de forma aleatória

i = 0

tentativas = 50 # pode não haver solução possível...

# ciclo de preenchimento do tabuleiro com a lista\_palavras de palavras

while i < len(lista\_palavras):

# palavra a colocar na sopa de letras: em maiusculas e pode ser invertida

lista\_palavras[i] = lista\_palavras[i].upper()

palavra = lista\_palavras[i]

# aleatoriamente a palavra poderá ser invertida

if random.choice([True, False]):

palavra = palavra[::-1] # inverte palavra

# determina localização horizontal aleatória

l = random.randint(0, linhas-1)

linha = tabuleiro[l] # linha aleatoria

col = random.randint(0, colunas-len(palavra)) # o comprimento da palavra limita o início da coluna

# verifica se tem espaço para a palavra

if linha.count(" ", col, col+len(palavra)) == len(palavra):

# coloca palavra

tabuleiro[l] = linha[:col] + palavra + linha[col+len(palavra):]

i += 1

else:

#procura outra posicao para a palavra

if tentativas > 0:

tentativas -= 1

else:

print("Nao foi possivel construir tabuleiro")

return False

## Fill in the blanks on the board with random letters

In this final step, the entire board is covered and, if a space is found, a random letter is placed.

# preencher o resto do tabuleiro com letras aleatorias

for i in range(linhas):

for j in range(colunas):

if tabuleiro[i][j] == " ":

tabuleiro[i] = tabuleiro[i][:j] \

+ random.choice("ABCDEFGHIJKLMNOPQRSTUVWXYZ") \

+ tabuleiro[i][j+1:]

return tabuleiro

|  |  |
| --- | --- |
| **Eye** | Note that the fourth line of instructions (if) is divided into three lines using the "\" character.  This allows for greater code readability. |

# Display the board

Function mostra\_tabuleiro(tabuleiro) traverses the various lines on the board and prints the line's string.

def mostra\_tabuleiro(tabuleiro):

for linha in tabuleiro:

print(linha)

# Iterating through moves - the game loop

Function sopa\_de\_letras()implements the game loop and has the following parameters:

* linhas - the number of lines on the board;
* colunas - the number of columns on the board;
* lista\_palavras - the list with the words to put on the board;
* tentativas - o número máximo de tentativas erradas.

This function starts with the generation of the board:

def sopa\_letras(linhas, colunas, lista\_palavras, tentativas):

# criacao do tabuleiro

tabuleiro = cria\_tabuleiro(linhas, colunas, lista\_palavras)

if type(tabuleiro) != list:

print ("Nao foi possivel construir o tabuleiro...")

return

# numero de palavras a adivinhar

palavras\_adivinhar = len(lista\_palavras)

The game loop is executed through a conditional loop (while) whose condition evaluates whether the player still has words to guess, and whether still has attempts to play.

while tentativas > 0 and palavras\_adivinhar > 0:

The algorithm for each move is as follows:

1. Interaction with the player: show board and ask for a word.

*Obs.: the word is converted to capital letters so that comparison is simpler.*

# Jogada do jogador

mostra\_tabuleiro(tabuleiro)

palavra = input("Palavra: ").upper()

1. Check if the word is correct (it belongs to the list of words).

If the word is correct, it is removed from the list of words so that is not validated again in a next move.

If the word is wrong, the number of remaining attempts is decreased.

# verificar se a palavra esta correta

if palavra in lista\_palavras:

# palavra certa

palavras\_adivinhar -= 1

print("Certo! Palavras a encontrar: " + str(palavras\_adivinhar))

if palavras\_adivinhar >= 0:

#apaga palavra da lista\_palavras

for i in range (len(lista\_palavras)):

if palavra == lista\_palavras[i]:

lista\_palavras[i] = ""

else:

# palavra errada

print("Errado!")

tentativas -= 1

After the end of the game loop, it is checked whether the player won or lost...

# fim do jogo

if tentativas == 0 and palavras\_adivinhar > 0:

print ("Perdeu. Faltaram " + str(palavras\_adivinhar) + "palavras")

else:

print ("Parabens! Encontrou todas as palavras.")

|  |  |
| --- | --- |
| **Balloon animal** | Test the game with the following list:  cidades = ["Porto", "Lisboa", "Braga", "Faro", "Bragança"]  sopa\_letras(10,30, cidades, 5)  Experiment later with multiple word lists for different topics. |

# Final challenge

In the solution presented, the words were restricted to being only horizontal. Add the possibility that the board has words vertically, or even diagonally.

|  |  |
| --- | --- |
| **Share with person** | **Share** your game! |