

Universidade de São Paulo  
Escola de Engenharia de São Carlos

Engenharia Elétrica com ênfase em Sistemas de Energia e Automação



**Introdução à Programação para Engenharias**  
**Jogo da Memória em Python**

Felipe Henrique Carvalho Silva – 14746252  
Ulisses Oliveira Basile – 14749255

25 de Maio, 2023  
São Carlos, SP



## 1 – Objetivos

O objetivo deste relatório é apresentar o desenvolvimento de um jogo da memória em Python 3, desenvolvido no **Visual Studio Code**, que é simples, interativo e possui uma interface intuitiva para garantir uma experiência sem dificuldades ao usuário.

Para atender às exigências estabelecidas, utilizamos estruturas de repetição, estruturas condicionais, vetores e/ou matrizes, além de implementar tratamento de erros e exceções em todo o código.

O jogo da memória é uma atividade lúdica que estimula o raciocínio, a concentração e a memória dos jogadores. Nossa versão do jogo foi desenvolvida em Python 3, aproveitando-se das funcionalidades dessa linguagem de programação para criar uma experiência agradável e desafiadora.

Ao longo do desenvolvimento, utilizamos estruturas de repetição, como loops, para criar iterações e permitir o funcionamento adequado do jogo. Além disso, implementamos estruturas condicionais para controlar o fluxo do programa, garantindo que as ações do jogador sejam tratadas corretamente.

Para armazenar as informações das cartas, utilizamos matrizes, permitindo a criação de um tabuleiro virtual onde as cartas são distribuídas e manipuladas durante o jogo. Essa abordagem nos permitiu criar uma estrutura organizada e eficiente para o armazenamento dos dados.

Também demos ênfase ao tratamento de erros e exceções em todo o código, a fim de evitar falhas inesperadas e melhorar a robustez do jogo. Implementamos mecanismos para lidar com situações de entrada inválida, erros de manipulação de dados e outros problemas que possam surgir durante a execução.

No geral, o desenvolvimento do jogo da memória em Python 3 atendeu aos requisitos estabelecidos, resultando em um programa funcional, interativo e fácil de usar. A combinação de estruturas de repetição, estruturas condicionais, vetores e/ou matrizes, juntamente com o tratamento adequado de erros e exceções, proporcionou uma base sólida para a criação de um jogo divertido e desafiador.

## 2 – Detalhes do projeto

### 2.1 – Estrutura de dados

Para o desenvolvimento do nosso jogo da memória, consideramos que a utilização de matrizes para armazenar e gerenciar as informações era a abordagem mais adequada. Optamos por trabalhar com duas matrizes distintas, cada uma desempenhando um papel específico no jogo.

A primeira matriz é responsável por armazenar os elementos a serem descobertos, ou seja, as cartas do jogo. Cada posição da matriz representa uma carta, contendo o valor correspondente a essa carta. Essa matriz nos permite definir os elementos que estarão presentes no jogo e garantir a diversidade das cartas.

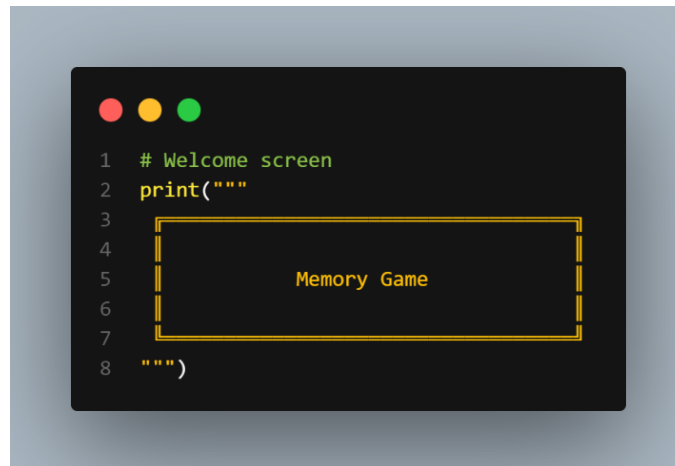
A segunda matriz é utilizada para controlar a posição e o estado das cartas. Cada posição da matriz armazena um valor booleano, indicando se o elemento correspondente já foi descoberto ou não. O valor True indica que a carta foi descoberta e deve ser mostrada, enquanto o valor False indica que a carta ainda não foi descoberta e deve permanecer oculta.

## 2.2 – Lógica principal do Código

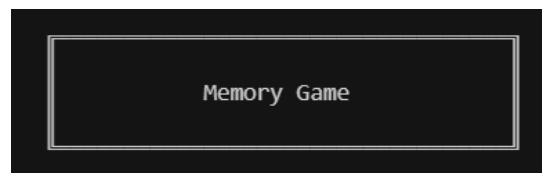
O código implementa um jogo da memória em Python 3, onde o usuário interage selecionando pares de cartas em um tabuleiro. O objetivo é encontrar todos os pares corretos. O jogo utiliza matrizes para armazenar as informações das cartas e verifica se os pares selecionados são iguais. Após encontrar todos os pares, uma mensagem de parabéns é exibida.

## 3 – Execução do projeto

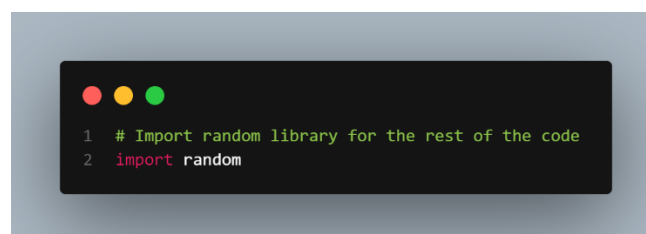
Inicialmente, é imprimida uma tela de boas-vindas para o usuário.



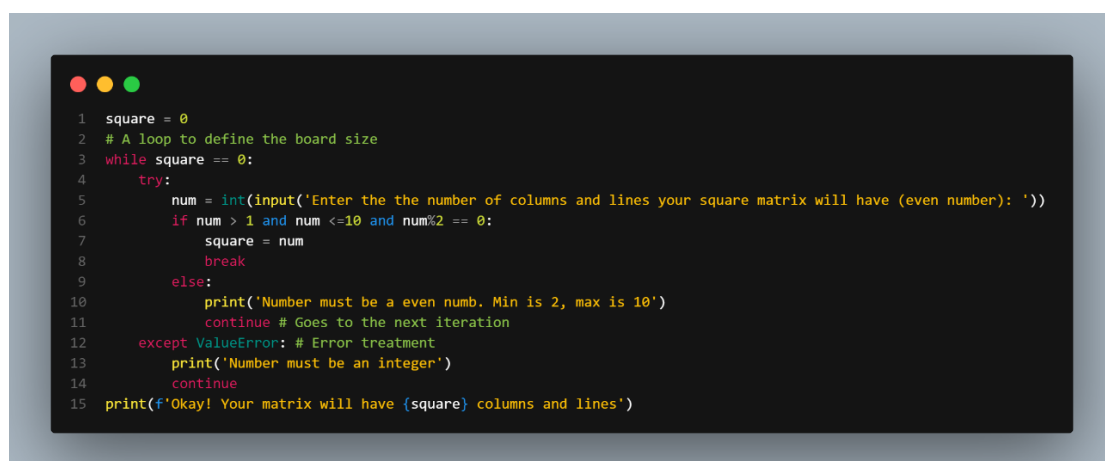
```
1 # Welcome screen
2 print("""
3
4     Memory Game
5
6 """)
7
8 """
```



Logo após, é importada a biblioteca random para ser usada no restante do código e um loop é feito para definir o tamanho do tabuleiro que será usado para o jogo da memória.



```
1 # Import random library for the rest of the code
2 import random
```



```
1 square = 0
2 # A loop to define the board size
3 while square == 0:
4     try:
5         num = int(input('Enter the the number of columns and lines your square matrix will have (even number): '))
6         if num > 1 and num <= 10 and num%2 == 0:
7             square = num
8             break
9         else:
10            print('Number must be a even numb. Min is 2, max is 10')
11            continue # Goes to the next iteration
12    except ValueError: # Error treatment
13        print('Number must be an integer')
14        continue
15 print(f'Okay! Your matrix will have {square} columns and lines')
```

```
Enter the the number of columns and lines your square matrix will have (even number): 2
Okay! Your matrix will have 2 columns and lines
(1,1) (1,2)
(2,1) (2,2)
```

```
Enter the the number of columns and lines your square matrix will have (even number): 10
Okay! Your matrix will have 10 columns and lines
(1,1) (1,2) (1,3) (1,4) (1,5) (1,6) (1,7) (1,8) (1,9) (1,10)
(2,1) (2,2) (2,3) (2,4) (2,5) (2,6) (2,7) (2,8) (2,9) (2,10)
(3,1) (3,2) (3,3) (3,4) (3,5) (3,6) (3,7) (3,8) (3,9) (3,10)
(4,1) (4,2) (4,3) (4,4) (4,5) (4,6) (4,7) (4,8) (4,9) (4,10)
(5,1) (5,2) (5,3) (5,4) (5,5) (5,6) (5,7) (5,8) (5,9) (5,10)
(6,1) (6,2) (6,3) (6,4) (6,5) (6,6) (6,7) (6,8) (6,9) (6,10)
(7,1) (7,2) (7,3) (7,4) (7,5) (7,6) (7,7) (7,8) (7,9) (7,10)
(8,1) (8,2) (8,3) (8,4) (8,5) (8,6) (8,7) (8,8) (8,9) (8,10)
(9,1) (9,2) (9,3) (9,4) (9,5) (9,6) (9,7) (9,8) (9,9) (9,10)
(10,1) (10,2) (10,3) (10,4) (10,5) (10,6) (10,7) (10,8) (10,9) (10,10)
```

É criada a matriz que guardará as posições dos elementos e onde será salvo se eles são True or False, para fazer uma verificação posterior se esses elementos já foram descobertos ou não.

```
1 matrix = [] # Creating the matrix of the position of the elements and where the True or False check will be made
2 for i in range(square):
3     row = []
4     for j in range(square):
5         row.append(0)
6     matrix.append(row)
7
8 for i in range(square): # Saving the element positions
9     for j in range(square):
10        matrix[i][j] = int(f'{i+1}{j+1}')
```

São sorteados os elementos a serem descobertos em outra matriz.

```
1 # Randomizing the elements that will appear on the board
2 elements = ['AA', 'AA', 'BB', 'BB', 'CC', 'CC', 'DD', 'DD', 'EE', 'EE', 'FF', 'FF', 'GG', 'GG', 'HH', ...]
3 size = square**2
4 m_elements = elements[0:size]
5 random.shuffle(m_elements) # Doing a shuffle to make sure that the elements are randomly arranged in the matrix
6
7 elements_matrix = []
8 for i in range(square):
9     row = []
10    for j in range(square):
11        row.append(m_elements[i*square+j])
12    elements_matrix.append(row)
```

Inicialmente, toda a matriz da posição dos elementos recebe False, ou seja, isso significa que no início do jogo nenhum elemento foi descoberto ainda.

```
1 # At the beginning of the game, no elements were discovered
2 for i in range(square):
3     for j in range(square):
4         matrix[i][j] = False
```

Várias funções foram definidas para facilitar o código e diminuir sua extensão:

- Uma função para verificar se o jogo acabou ou deve continuar

```
1 # A function to check if the game is over or should continue
2 def check_false(lang):
3     for i in range(len(lang)):
4         for j in range(len(lang)):
5             if lang[i][j] == False:
6                 return True
7     return False
```

- Uma função complementar para mostrar o tabuleiro do jogo pela primeira vez

```
1 # An extra function to print the game board on the first move only
2 def first_display():
3     board = ""
4     for i in range(square):
5         for j in range(square):
6             board += f'({i+1},{j+1})'
7             board += ' '
8         board += '\n'
9     print(board)
```

- Uma função para mostrar o tabuleiro novamente depois que o usuário digita valores inválidos para as posições no tabuleiro

```
1 # A function to show the matrix after the user give invalid inputs for the positions
2 def display_playboard_middle_play(a,b):
3     board = ""
4     for i in range(square):
5         for j in range(square):
6             if matrix[i][j] is True or (matrix[i][j] is False and (i == a and j == b)):
7                 board += f'{elements_matrix[i][j]}'
8             else:
9                 board += f'({i+1},{j+1})'
10            board += ' '
11        board += '\n'
12    print(board)
```

- Uma função para mostrar o tabuleiro depois que o usuário não consegue acertar dois elementos

```

1 # A function to print out the matrix after the user's wrong choices for a match
2 def display_playboard_after_chooses(a,b,c,d):
3     board = ""
4     for i in range(square):
5         for j in range(square):
6             if matrix[i][j] is True or (matrix[i][j] is False and ((i == a and j == b) or (i == c and j == d))):
7                 board += f'{elements_matrix[i][j]}'
8             else:
9                 board += f'({i+1},{j+1})'
10            board += ' '
11        board += '\n'
12    print(board)

```

- Uma função para o tabuleiro ser mostrado depois das escolhas de posições pelo usuário

```

1 # A function to show the matrix after the user choices
2 def display_playboard_while_or_after_play():
3     board = ""
4     for i in range(square):
5         for j in range(square):
6             if matrix[i][j] is True:
7                 board += f'{elements_matrix[i][j]}'
8             else:
9                 board += f'({i+1},{j+1})'
10            board += ' '
11        board += '\n'
12    print(board)

```

- Uma função que existe enquanto o jogo não acaba e que resulta em suas jogadas até que o jogo acabe

```

1 # A function to show the matrix after the user choices
2 def display_playboard_while_or_after_play():
3     board = ""
4     for i in range(square):
5         for j in range(square):
6             if matrix[i][j] is True:
7                 board += f'elements_matrix[i][j]'
8             else:
9                 board += f'({i+1},{j+1})'
10            board += ' '
11        board += '\n'
12    print(board)
13
14 # A function that will result in the player's moves and continue until the game is over
15 def playing():
16     a = None
17     b = None
18     c = None
19     d = None
20     while a is None or b is None or c is None or d is None:
21         try:
22             while a is None:
23                 a = int(input(f'Row: '))-1
24                 if a == -1:
25                     print("\nError: selected row can't be zero")
26                     a = None
27                     display_playboard_middle_play(a,b)
28                     continue # Continue to the next iteration if a is invalid
29                 elif a > square-1:
30                     print(f"\nError: selected row exceeds board's size! Select row again")
31                     a = None
32                     display_playboard_middle_play(a,b)
33                     continue # Continue to the next iteration if a is invalid
34             while b is None:
35                 b = int(input(f'Column: '))-1
36                 if b == -1:
37                     print("\nError: selected column can't be zero")
38                     b = None
39                     display_playboard_middle_play(a,b)
40                     continue # Continue to the next iteration if b is invalid
41                 elif b > square-1:
42                     print(f"\nError: selected column exceeds board's size! Select column again")
43                     b = None
44                     display_playboard_middle_play(a,b)
45                     continue # Continue to the next iteration if b is invalid
46             if matrix[a][b] is True:
47                 print('\nThis position has already been discovered, select again')
48                 a = None
49                 b = None
50                 display_playboard_middle_play(a,b)
51                 break
52             if a != None and b != None:
53                 display_playboard_middle_play(a,b)
54             while c is None:
55                 c = int(input(f'Row: '))-1
56                 if c == -1:
57                     print("\nError: selected row can't be zero")
58                     c = None
59                     display_playboard_middle_play(c,d)
60                     continue # Continue to the next iteration if c is invalid
61                 elif c > square-1:
62                     print(f"\nError: selected row exceeds board's size! Select row again")
63                     c = None
64                     display_playboard_middle_play(c,d)
65                     continue # Continue to the next iteration if c is invalid
66             while d is None:
67                 d = int(input(f'Column: '))-1
68                 if d == -1:
69                     print("\nError: selected column can't be zero")
70                     d = None
71                     display_playboard_middle_play(c,d)
72                     continue # Continue to the next iteration if d is invalid
73                 elif d > square-1:
74                     print(f"\nError: selected column exceeds board's size! Select column again")
75                     d = None
76                     display_playboard_middle_play(c,d)
77                     continue # Continue to the next iteration if d is invalid
78                 elif a == c and b == d:
79                     print("\nYou've selected the same element twice! Select the second once again")
80                     c = None
81                     d = None
82                     break # Continue to the next iteration if c or d is invalid
83                 if matrix[c][d] is True:
84                     print('\nThis position has already been discovered, select the second once again')
85                     display_playboard_middle_play(a,b)
86                     c = None
87                     d = None
88                     break # Continue to the next iteration if c or d is invalid
89                 if elements_matrix[a][b] == elements_matrix[c][d]:
90                     matrix[a][b] = True
91                     matrix[c][d] = True
92                     print(f'\nCongratulations, you got it!')
93                 else:
94                     print('\nThese are the cards you selected:')
95                     display_playboard_after_chooses(a,b,c,d)
96                     print(f'\nNo match, unfortunately...')
97             except ValueError:
98                 print(f'\nError: your number needs to be an integer and be between 1 to {square}! Try again')
99                 display_playboard_while_or_after_play()
100                continue
101        if a != None and b != None and c != None and d != None:
102            display_playboard_while_or_after_play()

```



- Parte do código que mostra o tabuleiro pela primeira vez no jogo e um loop e uma condição que verificam se o jogo deve continuar e quando ele acaba

```
1 # Part of the code that prints the board for the first time and a loop
2 # and a condition that checks when the game continues and when it ends
3 first_display()
4 while check_false(matrix) == True:
5     playing()
6 if check_false(matrix) == False:
7     print("
8     print(" *
9     print("
10    print("
11    print("
12    print("
13    print("
14    print("
15    print(" Congrats, you've done it! ")
16    print("
17    print("
18    print("
19    print("
20    print("
21    print("
22    print("
23    print("

```

## 4 – Rodando o código para um tabuleiro 2x2

início: usuário escolhe o tamanho da matriz (tabuleiro), nesse caso, 2x2.

Memory Game

```
Enter the the number of columns and lines your square matrix will have (even number): 2
Okay! Your matrix will have 2 columns and lines
(1,1) (1,2)
(2,1) (2,2)
```

Depois disso, a matriz é criada apresentando as coordenadas para instruir o usuário a escolher suas "cartas".

```

(1,1) (1,2)
(2,1) (2,2)

Row: 1
Column: 1
BB (1,2)
(2,1) (2,2)

Row: 1
Column: 2

These are the cards you selected:
BB AA
(2,1) (2,2)

No match, unfortunately...

```

Após a matriz com as coordenadas ter sido mostrada, é pedido ao usuário a linha e a coluna da primeira carta e da segunda. Em seguida, é mostrada a "matriz coordenada" (a chamaremos assim) com as cartas escolhidas reveladas, caso essas não sejam similares uma mensagem relata isso ao usuário e o jogo continua.

- Exemplo de possíveis entradas errôneas: caso o usuário escolha uma coluna que não exista, coluna = 0 nesse caso, uma mensagem relata ao usuário a impossibilidade de tal caso e novamente lhe pede uma coluna. Da mesma forma, se o usuário escolhe uma coluna maior que a quantidade existente de colunas, o mesmo procedimento acontece. Observação: o erro e sua solução ocorrem tanto para linha quanto coluna.

```

No match, unfortunately...
(1,1) (1,2)
(2,1) (2,2)

Row: 1
Column: 2
(1,1) AA
(2,1) (2,2)

Row: 1
Column: 0

Error: selected column can't be zero
(1,1) (1,2)
(2,1) (2,2)

Column: █

```

```

No match, unfortunately...
(1,1) (1,2)
(2,1) (2,2)

Row: 1
Column: 1
BB (1,2)
(2,1) (2,2)

Row: 1
Column: 5

Error: selected column exceeds board's size! Select column again
(1,1) (1,2)
(2,1) (2,2)

Column: █

```

Quando o usuário seleciona duas "cartas" (elementos) similares, uma mensagem aparece relatando isso e tais "cartas" permanecem reveladas daqui para frente.

```
No match, unfortunately...
(1,1) (1,2)
(2,1) (2,2)

Row: 1
Column: 1
BB (1,2)
(2,1) (2,2)

Row: 2
Column: 1

Congratulations, you got it!
BB (1,2)
BB (2,2)

Row: 
```

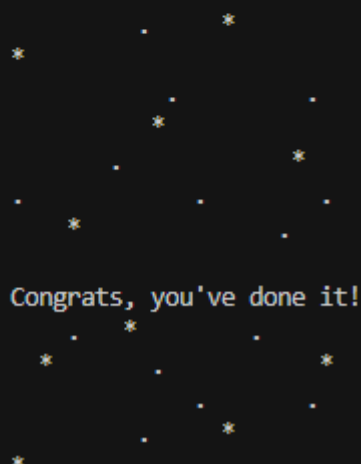
Assim que o jogador acerta o último par de cartas, o jogo acaba e uma mensagem parabenizando-o aparece. Observação: enquanto há cartas disponíveis, ainda é pedido ao usuário as coordenadas de linha e coluna.

```
BB (1,2)
BB (2,2)

Row: 1
Column: 2
BB AA
BB (2,2)

Row: 2
Column: 2

Congratulations, you got it!
BB AA
BB AA
```



Congrats, you've done it!

## 5 – Conclusões:

No contexto do desenvolvimento de um jogo da memória em Python 3, é possível concluir que o código elaborado atendeu plenamente às suas funções e apresentou um desempenho sem falhas significativas. Durante a implementação, foram utilizadas estruturas de repetição, estruturas condicionais, vetores e/ou matrizes, conforme requisitos estabelecidos. Além disso, o tratamento de erros e exceções foi aplicado de forma adequada ao longo do código, garantindo uma experiência de jogo livre de problemas. O embaralhamento aleatório dos elementos e a correspondência precisa dos pares trazem desafios e diversão ao jogo. Assim, o resultado é um jogo da memória funcional, interativo e cativante, desenvolvido com sucesso em Python 3.

## 6 – Link do executável (acesso apenas com e-mail USP)

[https://uspbr-my.sharepoint.com/:u:/g/personal/felipe\\_hcs\\_usp\\_br/EYiTBJFnQLtAuebGIZvL63UBnCuzYaXhTIDr\\_wnSUS83uA?e=a772ky](https://uspbr-my.sharepoint.com/:u:/g/personal/felipe_hcs_usp_br/EYiTBJFnQLtAuebGIZvL63UBnCuzYaXhTIDr_wnSUS83uA?e=a772ky)