Matrix Adivinhation

Amanda Lima Bezerra

Engenharia da Computação - Universidade Estadual de Feira de Santana (UEFS) Av. Transnordestina, s/n, Novo Horizonte Feira de Santana – BA, Brasil – 44036-900

limaa9000@gmail.com

Resumo. Após despertar, Neo enfrentou vários desafios para derrotar o seu maior adversário, o agente Smith. Com a ajuda dos alunos do primeiro semestre de Engenharia da Computação da Universidade Estadual de Feira de Santana, Neo vencerá a sua última adversidade, a construção do jogo das somas. O programa foi construído usando a linguagem de programação Python através do editor de código-fonte Visual Studio Code no sistema operacional Windows 11.

1. Introdução

Matrix, filme de 1999 [The Matrix, 2022], se passa em um futuro distópico, onde Neo, personagem principal, se percebe vivendo em uma realidade manipulada, ou seja, uma matrix. Após despertar para a realidade, uma simulação, Neo tem grandes desafios pela frente. É nessa contextualização que foi proposto o problema para os alunos do curso de engenharia da computação. O jogo das somas é o último desafio que Neo deve enfrentar para vencer o seu grande adversário o agente Smith.

O programa trata-se de um jogo para dois jogadores, jogado em um ou dois tabuleiros no qual o objetivo é tentar adivinhar a soma das linhas e das colunas da matriz. A pontuação é dada em cada rodada ao jogador que mais se aproxima do resultado da soma em questão. Ao final, o jogador vencedor é o que possui a maior quantidade de pontos.

Em relação à questão pedagógica, o objetivo principal do problema é estimular os alunos do primeiro semestre de Engenharia da Computação a aprender e trabalhar com manipulação de matrizes na programação com a linguagem Python. Ademais praticar a modularização dos programas, ou seja, definir funções que executam passo a passo o jogo.

Em suma, o trabalho foi realizado ao longo de cinco sessões da disciplina MI - Algoritmos, na qual foi discutido como produzir o programa. Ao longo deste relatório será detalhado como o projeto foi executado e quais direcionamentos, pessoais e coletivos, moldaram a construção do jogo.

2. Metodologia

O programa é um jogo de tabuleiro para duas pessoas no qual existem três níveis: fácil, médio e difícil, e dois modos de jogo, com um ou dois tabuleiros e com número de rodadas definido ou até preencher todo o tabuleiro. Para facilitar a construção do jogo, o código foi desenvolvido com várias funções no qual são usadas diversas vezes ao longo da partida.

Partindo desse pressuposto, o produto foi desenvolvido em vinte e sete funções, que serão detalhadas neste relatório. Um grande auxiliar na construção desse projeto foi um caderno

de rascunho, onde foram anotadas as principais ideias e ordenação das funções, visto que a primeira versão do jogo ficou confusa e de difícil entendimento.

2.1 Organização

A modularização do código foi uma regra preestabelecida, logo o programa foi feito dentro de várias funções. A organização das mesmas está na seguinte ordem: receber os principais dados dos jogadores, funções responsáveis por montar os tabuleiros, modos de jogo, funções responsáveis pela jogabilidade e finalmente as funções de validações.

Além do mais a construção do jogo requereu o uso de duas bibliotecas, logo foram chamadas nas primeiras linhas do código. A utilidade delas é a geração de números aleatórios e a cópia profunda de alguns itens ao longo do jogo. A cópia profunda, diferente da cópia superficial, permite a alteração do item copiado sem alterações no item original.

De início é necessário coletar os dados dos participantes, dados esses como nome dos jogadores, nível do jogo e os modos. Todas essas informações são coletadas pela função menu, Figura 1.

```python
def menu():
    print('\t\tJOGO DA SOMA\nInicio')
    nome_1 = input('Nome do jogador 1 -----> ')
    nome_2 = input('Nome do jogador 2 -----> ')
    nome_1, nome_2 = validacao_nomes(nome_1, nome_2)

    quant_tabuleiro = input('Quantidade de tabuleiros: \n1. Um tabuleiro\n2. Dois tabuleiros\n-----> ')
    quant_tabuleiro = validacao_1(quant_tabuleiro)

    nivel = input('Nível do jogo: \n1. Fácil\n2. Médio\n3. Difícil\n-----> ')
    nivel = validacao_2(nivel)

    # Coletando as informações sobre o tamanho do tabuleiro.
    linhas, colunas, limite, quant_numeros = tamanho_tabuleiro(nivel)

    termino = input('Encerramento da partida\n1. Número de rodadas\n2. Tabuleiro completamente revelado\n-----> ')
    termino = validacao_1(termino)
    if termino == 1:
        rodadas = input('Qual o números de rodadas?\n-----> ')
        rodadas = validacao_3(rodadas)
        # Nessa parte do código será chamada as funções de cada tipo de jogo. Foi dividido desse jeito pois ficou mais fácil a organização
        # do código.
        if quant_tabuleiro == 1:
            tabuleiro1_rodadas(nome_1, nome_2, quant_tabuleiro, linhas, colunas, limite, quant_numeros, rodadas)
        else:
            tabuleiro2_rodadas(nome_1, nome_2, quant_tabuleiro, linhas, colunas, limite, quant_numeros, rodadas)
    else:
        # Nessa parte do código será chamada as funções de cada tipo de jogo.
        if quant_tabuleiro == 1:
            tabuleiro1_rodadas0(nome_1, nome_2, quant_tabuleiro, linhas, colunas, limite, quant_numeros)
        else:
            tabuleiro2_rodadas0(nome_1, nome_2, quant_tabuleiro, linhas, colunas, limite, quant_numeros)
```

Figura 1 – Função menu.

O próximo bloco de funções, Figura 2, são as funções responsáveis por construir as matrizes, no caso do jogo foram usadas duas: um tabuleiro e um tabuleiro de resposta. Cada tabuleiro no jogo tem o seu próprio tabuleiro resposta que irá acompanhar ao longo de todo o programa. No caso do tabuleiro resposta ele nunca é mostrado ao usuário, mas ele é essencial para fazer todas as comparações necessárias para o bom funcionamento do jogo.

```
  # Função que determina o tamanho do tabuleiro.
  # Em todos os casos o tabuleiro terá duas linhas e
  # colunas a mais para armazenar o resultado da soma.

  # Fácil = 3x3
  # Médio 4x4
  # Díficil 5x5

  # O limite é para definir o intervalo dos números aleatórios e
  # a quant_numeros é para definir quantos números aleatórios irá precisar.
> def tamanho_tabuleiro(nivel):···

  # Função que organiza a criação do/dos tabuleiro(s) do jogo.
> def montando_tabuleiro(linhas, colunas, limite, quant_numeros):···

  # Funções que montam o/os tabuleiro(s).
> def criar_matriz(linhas, colunas):···

> def montar_tabuleiro(tabuleiro, linhas, colunas):···

> def colocar_numeros(tabuleiro_resp, limite, quant_numeros, linhas, colunas):···

> def mostrar_matriz(matriz, linhas, colunas):···
```

Figura 2 – Funções para criar os tabuleiros.

Logo em seguida, Figura 3, estão as funções responsáveis pela jogabilidade, logo é a parte principal do código.

```
  # Função para um único tabuleiro.
> def achar_soma(tabuleiro_resp, dicionario_jogadas, nome_1, nome_2, linhas, colunas):···

  # Função para dois tabuleiros.
> def achar_soma2(tabuleiro_resp1, tabuleiro_resp2, dicionario_jogadas, nome_1, nome_2, linhas, colunas):···

> def comparar(nome_1, nome_2, soma1, soma2, dicionario_jogadas):···

> def ganhador(dicionario_jogadas, nome_1, nome_2, soma1, soma2, pontuacao1, pontuacao2):···

> def mostrar_vencedor(lista_ganhador):···

> def historico(copia_historico, nome_1, nome_2, historico1, historico2):···

> def mostrar_historico(nome_1, nome_2, historico1, historico2):···

> def achar_posicao(lista_ganhador, dicionario_jogadas, tabuleiro_resp, linhas, colunas):···

> def trocar_numeros(dicionario_revelacao, tabuleiro, tabuleiro_resp, linhas, colunas):···
```

Figura 3 – Jogabilidade.

Por último, Figura 4, as funções responsáveis pelas validações das jogadas, cada uma para determinada parte do código.

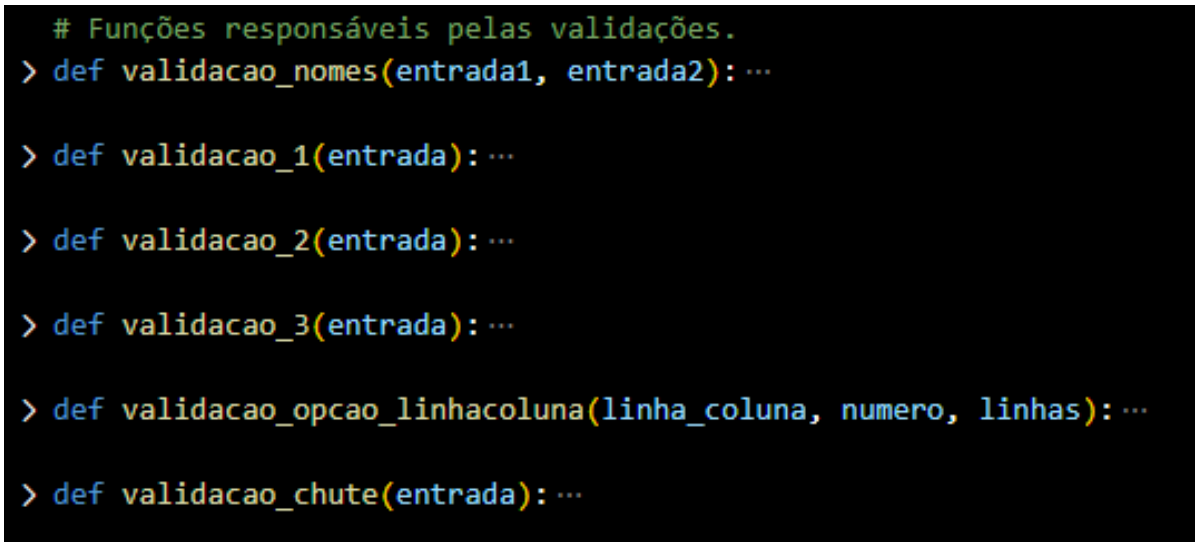

Figura 4 – Validações.

2.2 Regras do jogo

O objetivo principal do jogo, do ponto de vista do usuário, é tentar adivinhar a soma das linhas ou das colunas do tabuleiro. Para o desenvolvedor as regras estabelecidas foram construir tabuleiros de acordo com o nível escolhido pelo usuário e preencher esse tabuleiro com números aleatórios sem repetição. Além de tudo, a célula do tabuleiro revelada depende do valor colocado pelo jogador; caso o palpite seja maior que o resultado da soma o número revelado será o maior, no caso contrário o número revelado será o menor por último em caso de acerto do valor o jogo deve revelar todas as casas da linha ou coluna acertada.

2.3 Uso de matriz

Em Python uma matriz é formada a partir de uma lista de listas e o objetivo principal do projeto é aprender a manipular matrizes na computação. A construção do jogo é toda baseada em matrizes, já que o tabuleiro é uma matriz por si só.

Além de construir o tabuleiro o desenvolvedor deve saber manipular os valores contidos nela, no caso específico do jogo saber percorrer a matriz para realizar as somas, achar os menores ou maiores valores das linhas ou colunas, revelar uma ou mais células da matriz.

No código do jogo todas as manipulações das matrizes foram feitas através de estruturas de repetição encadeadas, ou seja, *for coluna in range(colunas)* dentro de um *for linha in range(linhas)*, Figura 5, em que as devidas verificações eram feitas a fim de encontrar o valor desejado na matriz. No exemplo citado na Figura 5 a matriz foi percorrida com o objetivo de achar o valor que deveria ser trocado no tabuleiro que aparece para o jogador.

```python
def trocar_numeros(dicionario_revelacao, tabuleiro, tabuleiro_resp, linhas, colunas):
    # Trocando os números no tabuleiro.

    # Estrutura do dicionario_revelacao = Nome: [numero_revelado, linha, coluna]
    for lista_informacoes in dicionario_revelacao.values():
        for linha in range(1, linhas):
            for coluna in range(1, colunas):
                # Acertou somente um número.
                if lista_informacoes[1] == linha and lista_informacoes[2] == coluna and tabuleiro_resp[linha][coluna] != 0:
                    tabuleiro[linha][coluna] = tabuleiro_resp[linha][coluna]
                    tabuleiro_resp[linha][coluna] = 0
                # Acertou a soma da linha.
                elif lista_informacoes[1] != 0 and lista_informacoes[2] == 0 and linha == lista_informacoes[1] \
                    and tabuleiro_resp[linha][coluna] != 0:
                    tabuleiro[lista_informacoes[1]][coluna] = tabuleiro_resp[lista_informacoes[1]][coluna]
                    tabuleiro_resp[lista_informacoes[1]][coluna] = 0
                # Acertou a soma da coluna.
                elif lista_informacoes[1] == 0 and lista_informacoes[2] != 0 and coluna == lista_informacoes[2] \
                    and tabuleiro_resp[linha][coluna] != 0:
                    tabuleiro[linha][lista_informacoes[2]] = tabuleiro_resp[linha][lista_informacoes[2]]
                    tabuleiro_resp[linha][lista_informacoes[2]] = 0
    return tabuleiro, tabuleiro_resp
```

Figura 5 – *For* encadeado.

2.4 Uso de função

O código foi dividido em várias funções, no qual o objetivo era cada uma delas ter apenas um único uso. Algumas funções foram usadas para organizar a chamada de outras funções, a exemplo da Figura 6 que organiza a chamada das funções para criar o tabuleiro e o tabuleiro resposta. Ao todo o produto final tem vinte e sete funções, que serão explicadas mais a fundo nesse tópico.

```python
# Função que organiza a criação do/dos tabuleiro(s) do jogo.
def montando_tabuleiro(linhas, colunas, limite, quant_numeros):
    tabuleiro_vazio = criar_matriz(linhas, colunas)
    tabuleiro = montar_tabuleiro(tabuleiro_vazio, linhas, colunas)
    # Cópia profunda do tabuleiro pois eu tenho que alterar o tabuleiro
    # resposta ao longo do jogo sem que o tabuleiro seja alterado.
    copia = copy.deepcopy(tabuleiro)
    tabuleiro_resp = colocar_numeros(copia, limite, quant_numeros, linhas, colunas)
    return tabuleiro, tabuleiro_resp
```

Figura 6 – Organizar a chamada das funções.

As funções mais básicas são as do menu, junto com as que montam os tabuleiros e por último as que fazem a validação das entradas. O menu apresenta o jogo e colhe os dados necessários para o funcionamento do jogo, a montagem do tabuleiro é dividido em: criar uma matriz com números zeros, o preenchimento do mesmo com indicações de linhas e colunas, a criação do tabuleiro resposta e a apresentação do tabuleiro e finalmente, o último bloco de funções notáveis, as validações que verificam cada uma das entradas e das jogadas para saber se são válidas ou não. Ao total são treze funções que compõe esse bloco.

As funções responsáveis por organizar a jogabilidade são quatro, como mostrada na Figura 7, dentro delas existem duas variáveis iniciadas com zero para a contagem da pontuação, pontuacaoo1 e pontuacao2, e duas listas vazias que acumulam os históricos dos jogadores, historico1 e historico2.

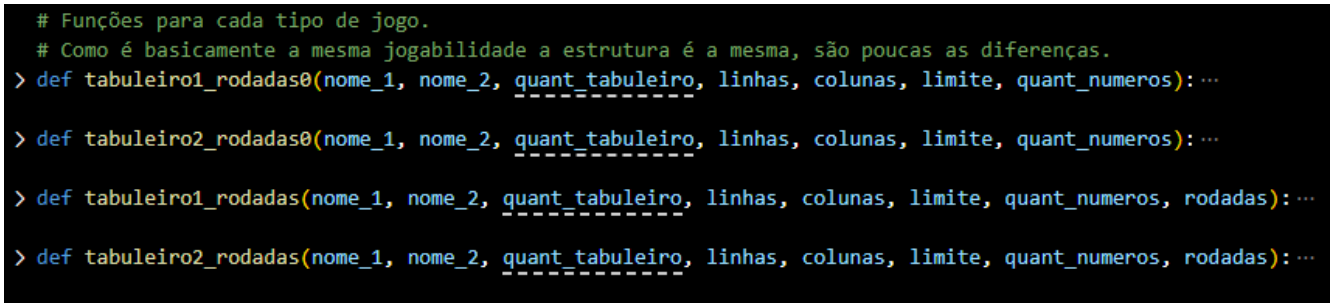

Figura 7 – Modos do jogo.

A divisão do jogo em quatro modos foi feita com o objetivo de organizar uma melhor escrita do código, visto que a primeira versão não finalizada do jogo ficou confusa. A diferença entre esses modos de jogo está justamente no loop que faz o jogo continuar ou não e na chamada das funções que fazem o jogo funcionar, nos casos com dois tabuleiros as funções são chamadas duas vezes. Quando o jogo acaba com a tabela toda preenchida, o loop faz uma verificação se ainda existem células do tabuleiro a serem preenchidas Figura 8.

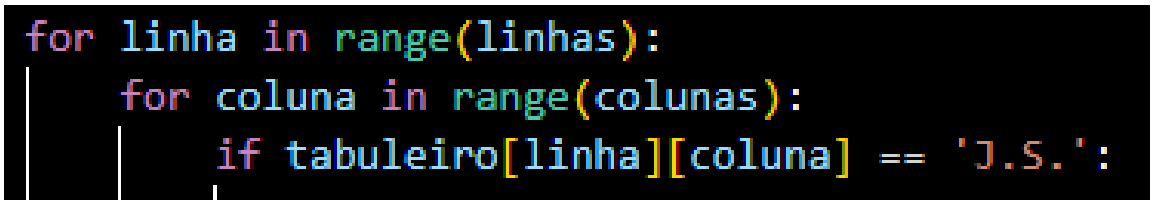

Figura 8 – Código de verificação.

Por outro lado, a versão com número de rodadas pré-definida existe um loop com uma variável auxiliar que é sempre comparada com o número de rodadas definida pelos jogadores Figura 9.

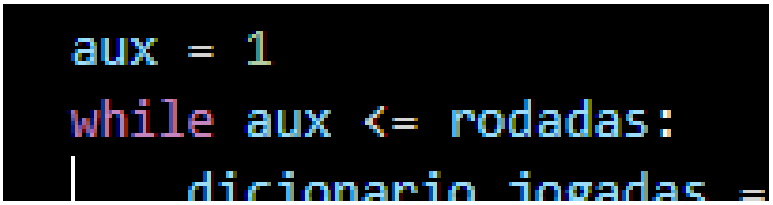

Figura 9 – Loop de comparação.

Ao total o número de funções responsáveis pelo funcionamento do jogo foram dez, funções essas mais elaboradas. A primeira delas é a escolha das jogadas, onde o programa guarda a indicação da escolha da linha ou coluna e o número chutado em um dicionário no qual a chave é o nome do jogador e o valor é uma lista que contém todas as informações. Logo em seguida existem duas funções achar_soma e achar_soma2, essas são, respectivamente, responsáveis por achar a soma da linha ou coluna escolhida pelos usuários que estão jogando em uma única tabela e de duas tabelas. A escolha de separar em duas funções veio justamente por parecer uma opção mais rápida de construir o código, já que o prazo para a finalização do projeto estava curto. No final a diferença entre elas está somente na chamada de um ou dois tabuleiros resposta e o local de procura dessas somas.

Ainda analisando as funções da jogabilidade, as funções comparar, ganhador e mostrar_ganhador analisam a diferença entre os valores chutados pelos jogadores com as somas das respectivas linhas ou colunas, atribuem a um dicionário se o valor foi maior, menor

ou igual a soma e apresentam o ganhador. Logo em seguida são as funções responsáveis pela adição das jogadas no histórico e apresentar esse histórico para os jogadores. Todas essas funções estão presentes na Figura 10.

```
  # Função para um único tabuleiro.
> def achar_soma(tabuleiro_resp, dicionario_jogadas, nome_1, nome_2, linhas, colunas):···

  # Função para dois tabuleiros.
> def achar_soma2(tabuleiro_resp1, tabuleiro_resp2, dicionario_jogadas, nome_1, nome_2, linhas, colunas):···

> def comparar(nome_1, nome_2, soma1, soma2, dicionario_jogadas):···

> def ganhador(dicionario_jogadas, nome_1, nome_2, soma1, soma2, pontuacao1, pontuacao2):···

> def mostrar_vencedor(lista_ganhador):···

> def historico(copia_historico, nome_1, nome_2, historico1, historico2):···

> def mostrar_historico(nome_1, nome_2, historico1, historico2):···
```

Figura 10 – Jogabilidade.

As funções que demandaram mais tempo e análise para serem construídas, foram as funções achar_posição e trocar_numero, com utilidade de, respectivamente, achar a posição do número a ser trocado e efetuar essa troca. A primeira ideia era de fazer essas duas funções em uma única função, porém no momento da construção do código a função ficou muito grande e confusa. Visando facilitar o melhor entendimento possível do código surgiu a ideia de dividir essa função em duas, uma que acha somente a posição que será trocada e outra que de fato efetua essa troca. Os maiores problemas do código surgiram nessas funções, problemas esses que serão discutidos mais a fundo no tópico de testes.

3. Resultado e discussões

As discussões a respeito do bom funcionamento do jogo e dos seus erros e melhorias serão falados nos próximos tópicos.

3.1 Manual de uso

O usuário que deseja jogar o jogo das somas deve primeiramente ter o arquivo do programa e a linguagem Python instalada no computador. O jogo pode ser jogado no terminal do computador sem problemas ou em qualquer IDE desejada. Ao abrir o arquivo basta executar o programa, mas para jogar novamente o programa deve ser reaberto, visto que o programa não executa de novo o jogo.

3.2 Testes

Os testes do jogo foram inicialmente feitos mostrando na tela a tabela resposta e todos os dicionários usados, com o propósito de verificar como estava acontecendo a leitura do código e suas variáveis, a exemplo da Figura 11 e Figura 12. Através dos testes realizados foi detectado o primeiro erro da função achar_posicao, dentro de algumas jogadas as coordenadas da posição

do número a ser trocado estavam erradas. Analisando a função o erro estava relacionado com a função de trocar as posições, já que após a troca o número era substituído por zero. Após isso, o menor número na comparação era o zero. O erro foi corrigido colocando uma condicional que indicava que o número só poderia ser comparado se fosse diferente de zero.

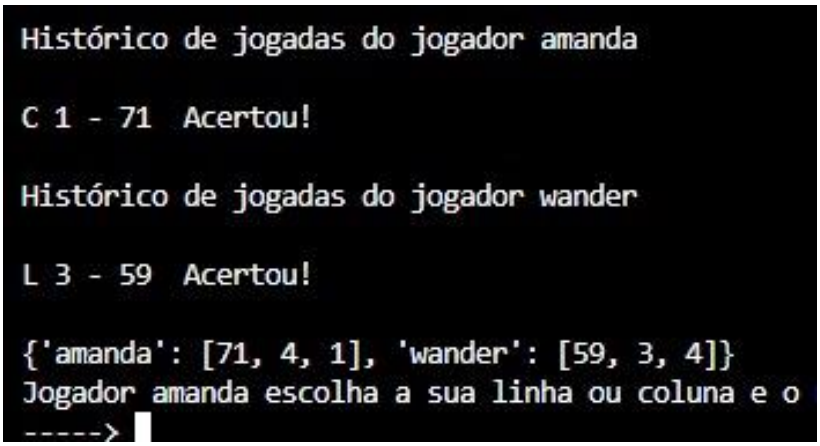

Figura 11 – Testes de variáveis dicionário.

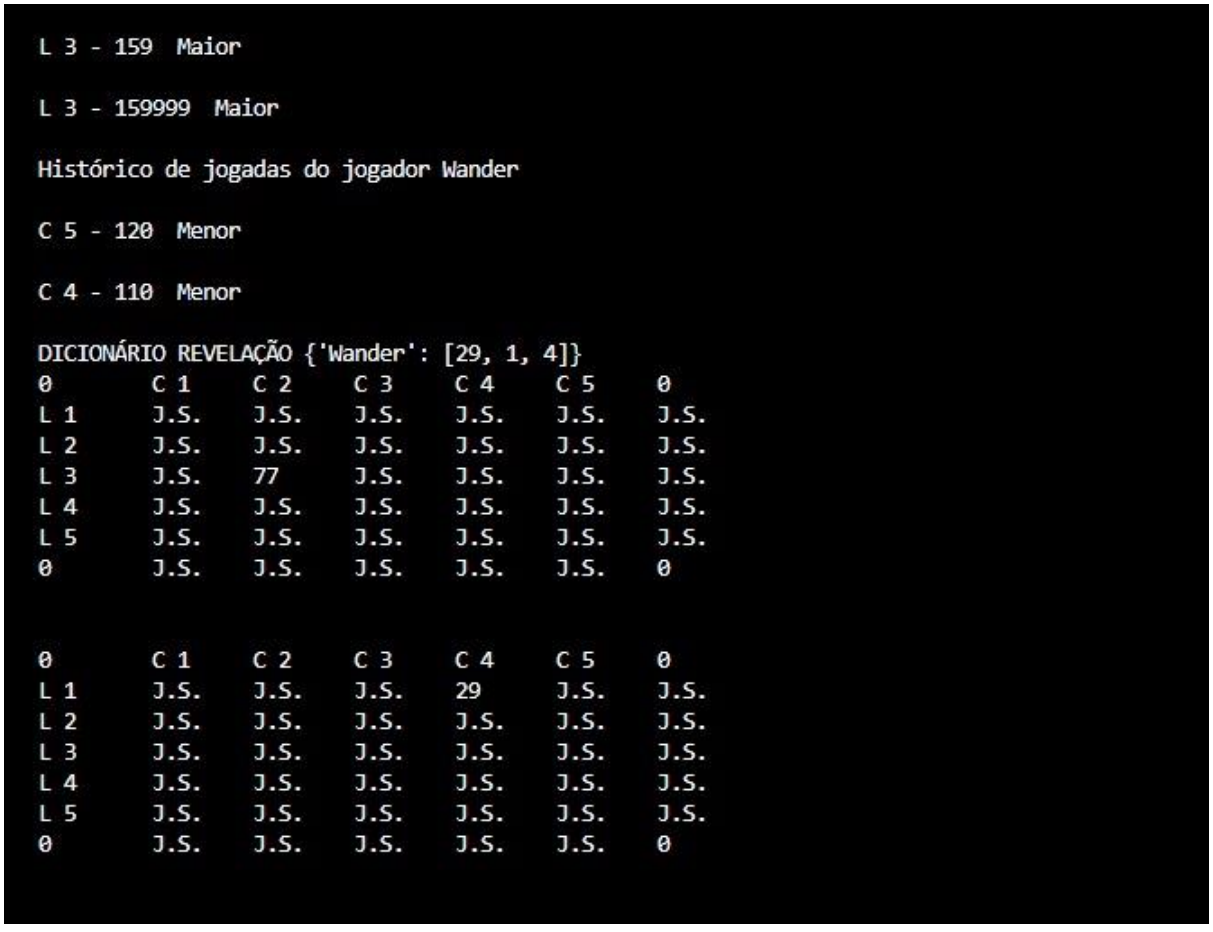

Figura 12 – Testes de variáveis dicionário.

Porém novamente ocorreu um erro nessa mesma função, erro esse que quebrava o código e aparecia a mensagem da Figura 13. Acontece que a forma como os números estavam sendo comparados era pegando o primeiro número da linha ou coluna e usando como referência para comparar com os outros, mas em alguns casos entrava em conflito com a verificação se era diferente de zero. Caso o primeiro número fosse zero o código não ia ter nenhum número como referência para comparar. Esse erro foi solucionado com duas variáveis que são usadas para comparar quando o número é maior ou menor. No caso dos maiores números um número extremamente baixo era usado para comparar, nesse caso o numero_revelado_maior recebeu -10000 e o numero_revelado_menor recebeu 10000.

```
-----> 123
LISTA DE GANHADOR FUNÇÃO GANHADOR ['a']

O(s) ganhador(es) da rodada:

Parabéns a, ganhador da rodada!

L 3 - 21  Menor

Histórico de jogadas do jogador w

L 3 - 123  Maior

C 1 - 123  Maior

Traceback (most recent call last):
  File "d:\Visual Studio Code\MI Algoritmos - Jogo da Soma.py", line 544, in <module>
    menu()
  File "d:\Visual Studio Code\MI Algoritmos - Jogo da Soma.py", line 36, in menu
    tabuleiro1_rodadas0(nome_1, nome_2, quant_tabuleiro, linhas, colunas, limite, quant_numeros)
  File "d:\Visual Studio Code\MI Algoritmos - Jogo da Soma.py", line 194, in tabuleiro1_rodadas0
    dicionario_posicao = achar_posicao(lista_ganhador, dicionario_jogadas, tabuleiro_resp, linhas, colunas)
  File "d:\Visual Studio Code\MI Algoritmos - Jogo da Soma.py", line 394, in achar_posicao
    elif numero_revelado > tabuleiro_resp[linha_coluna_numero][coluna]:
UnboundLocalError: local variable 'numero_revelado' referenced before assignment
PS D:\Visual Studio Code>
```

Figura 13 – Erro do código.

3.3 Erros e melhorias

A primeira melhoria que se deve fazer no programa é colocar uma verificação melhor na entrada da indicação da linha ou coluna que o jogador quer escolher. A execução do código sem erros depende de o usuário dar um espaço entre as opções, caso contrário o código exibe uma mensagem de erro como na Figura 14. Mesmo que a verificação das entradas de linhas e colunas tenha suas falhas, ainda assim existe uma verificação para caso o jogador coloque algo diferente de ‘L’ ou ‘C’, representando linha e coluna respectivamente, ou algo diferente de números para a indicação de qual linha ou coluna ele se refere.

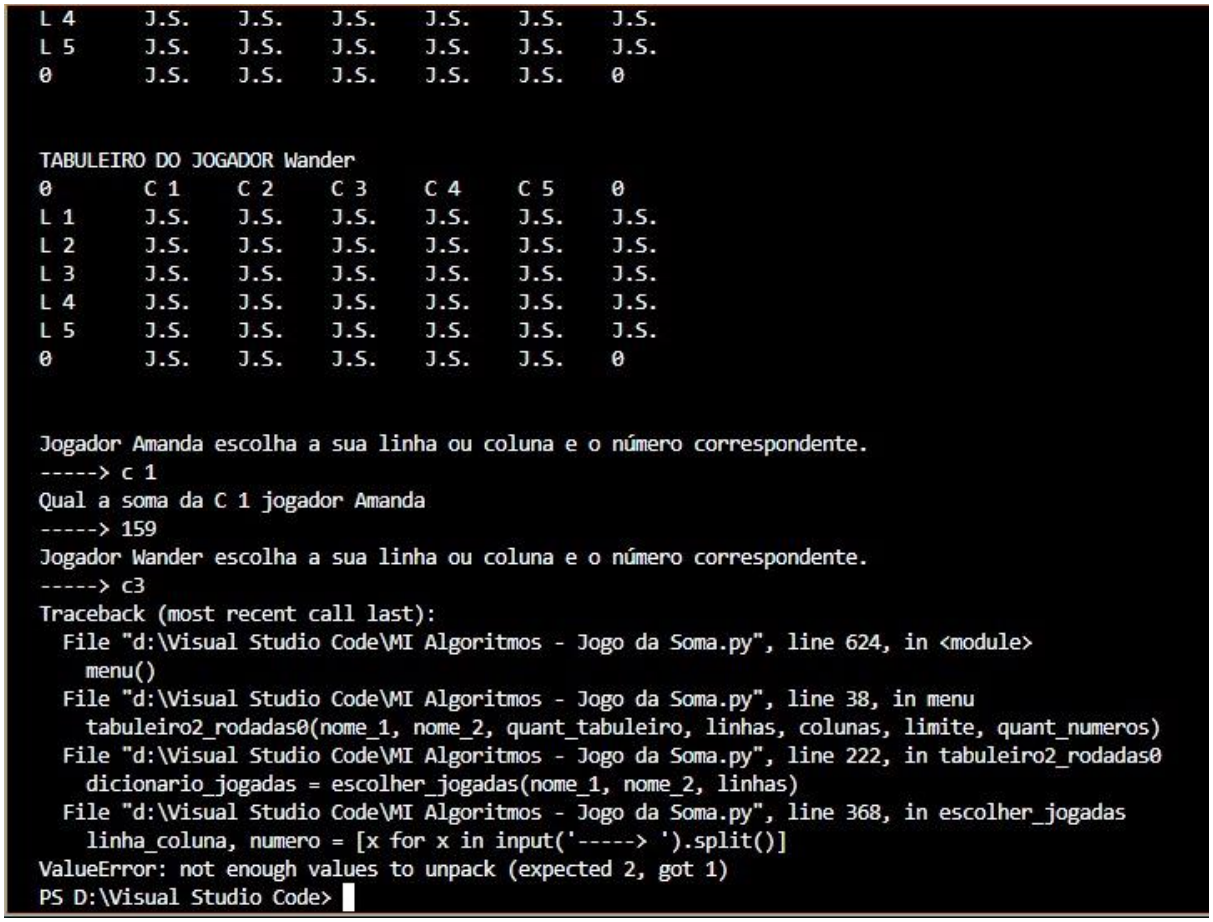

Figura 14 – Quebra do código.

Outra melhoria importante no programa é a adição de uma lista de linhas ou colunas já preenchidas para impedir que qualquer jogador escolha alguma célula já completa. Essa verificação vem da necessidade de impedir os jogadores de obter pontos facilmente.

Por último, a aparência do jogo poderia ser melhorada. Existem diversos recursos visuais que poderiam ter sido usados na produção do jogo, mas por uma questão de prazo acabou sendo deixado de lado pois a prioridade foi a produção do próprio jogo.

4. Conclusão

O objetivo principal do programa foi treinar o uso de matrizes e funções através da construção de um jogo. Esse objetivo foi cumprido, visto que o produto final, mesmo com defeitos, o jogo exerce o que foi pedido. Com isso, Neo conseguiu finalmente derrotar seu maior adversário, o agente Smith.

O produto final pode sim ser melhorado, já que existem falhas, mas ao final foi possível colocar em pratica o uso das estruturas estudadas em sala de aula.

5. Referências

Gatto, Elaine (2017) “Funções e procedimentos - Modularização” https://www.embarcados.com.br/funcoes-e-procedimentos-modularizacao/#:~:text=Modulariza%C3%A7%C3%A3o%20pode%20ser%20entendida%20como,conversam%20umas%20com%20as%20outras, Acessado em 19 de Maio de 2022.

Porto Editora (2016) “The Matrix (trilogia)” https://www.infopedia.pt/apoio/artigos/$the-matrix-(trilogia), Acessado em 16 de Maio de 2022.

Orestes, Yan (2018) “Como fazer uma cópia de uma lista no Python” https://www.alura.com.br/artigos/como-fazer-copia-de-lista-python?gclid=Cj0KCQjw-JyUBhCuARIsANUqQ_JzHaqAM06CleOFNk0SPtMrGSN0MDeO9Yi1zQm0Ba1TyuKL2Ih0vRcaAhItEALw_wcB, Acessado em 14 de Maio de 2022.

GeeksforGeeks (2018) “Python | random.sample() function” https://www.geeksforgeeks.org/python-random-sample-function/, Acessado em 12 de Maio de 2022.

Stack Overflow (2017) “Gerar números aleatórios em Python sem repetir” https://pt.stackoverflow.com/questions/195422/gerar-n%C3%BAmeros-aleat%C3%B3rios-em-python-sem-repetir, Acessado em 12 de Maio de 2022.