UNIVERSIDADE ESTADUAL DE FEIRA DE SANTANA

VIVIAN MARTINS MOURA

RELATÓRIO PBL

FEIRA DE SANTANA

2025

1. INTRODUÇÃO

1.1 Resumo do problema

O "Jogo da Vida", criado por John Conway, é uma simulação onde células vivem, morrem ou se multiplicam em uma grade, seguindo algumas regrinhas simples. Apesar de parecer um jogo, ele não tem jogador, tudo acontece automaticamente, o que torna os resultados ainda mais interessantes. A proposta é representar o comportamento de seres vivos e como eles mudam ao longo do tempo. O mais curioso é ver como padrões bem complexos podem surgir a partir de configurações bem simples. Isso chama atenção de várias áreas, como biologia, matemática e ciência da computação. Resolver esse problema ajuda a entender melhor como a organização e a evolução funcionam, mesmo em sistemas artificiais. É como observar uma "vida" digital surgindo diante dos nossos olhos. A ideia de implementar esse jogo é justamente explorar essa dinâmica e ver até onde ela pode ir.

1.2 Descrição breve da solução

Então implementei esse jogo utilizando um código em python na versão mais atual, a versão 3.13.2, no Visual Studio Code e implementei tudo que foi solicitado, que detalharei melhor na metodologia.

2. METODOLOGIA

2.1 Processo de construção de conhecimento nas sessões tutoriais

Na primeira sessão mesmo lendo o problema eu não havia entendido muito bem como o jogo funcionava, porém, meus colegas decidiram fazer uma versão menor do jogo no quadro apenas para entendermos melhor o seu funcionamento e isso me ajudou muito a entender suas regras e como ele acontecia, o que foi muito útil para mim. Além disso, as sessões no geral me ajudaram a perceber que havia várias maneiras diferentes de criar o jogo e aplicar suas regras, pois muitos dos meus colegas me explicaram sua lógica e como aplicaram isso e todos fizeram dar certo de uma maneira diferente, o que eu achei muito interessante.

2.2 Definição de requisitos

A proposta é representar a evolução de uma sociedade em um tabuleiro 10x10, onde cada célula pode estar viva ou morta. O sistema deve permitir que o usuário insira um padrão inicial, definindo as células vivas, e as que sobrarem serão as mortas. Os símbolos para representar células vivas e mortas devem ser escolhidos pelo próprio desenvolvedor. A lógica de evolução segue regras específicas baseadas nos vizinhos (as 8 células ao redor):

- Solidão: célula viva com menos de 2 vizinhos vivos morre.
- Sobrevivência: célula viva com 2 ou 3 vizinhos vivos continua viva.
- Superpopulação: célula viva com mais de 3 vizinhos vivos morre.
- Reprodução: célula morta com exatamente 3 vizinhos vivos se torna viva.

A simulação deve terminar automaticamente quando todas as células estiverem mortas. Além disso, o programa deve ser desenvolvido com modularização, ou seja, dividido em funções que organizem melhor as tarefas do sistema.

2.3 Descrição de alto nível

O programa desenvolvido simula o "Jogo da Vida", utilizando a linguagem Python e aplicando o conceito de modularização. Ele é dividido em dois arquivos principais: um contendo a lógica principal da execução e outro com variáveis e funções auxiliares. Essa separação ajuda a organizar melhor o código. De forma geral, o algoritmo pode ser dividido em três grandes partes: (1) inicialização do ambiente e criação do tabuleiro, (2) entrada de dados para definição do estado inicial, e (3) execução da simulação conforme as regras do jogo até que todas as células estejam mortas.

Na etapa de **inicialização**, o programa cria uma matriz 10x10 onde todas as células estão inicialmente mortas. Também são definidos os símbolos que representam células vivas e mortas, bem como outras variáveis úteis para o funcionamento do jogo, como o tempo de espera entre as gerações.

Na etapa de **configuração inicial**, é criado um loop com ‘while True’ para fazer a verificação de entrada no input perguntando se o usuário deseja reviver alguma célula, e a resposta deve ser apenas ‘s’(sim) ou ‘n’(não), caso o usuário digite ‘s’ o ‘while True’ quebra e o programa entra no próximo loop criado. No próximo while o loop só irá parar quando o usuário digitar ‘n’, caso ele digite ‘s’ significa que ele deseja reviver alguma célula, então será perguntado qual a linha e a coluna dessa célula. Esse processo se repete até que o usuário indique que não deseja mais modificar o tabuleiro, ou seja, caso ele digite ‘n’. Após isso, o estado inicial do tabuleiro é mostrado na tela, e após isso o jogo começa.

```
while True:
    try:
        vivasresposta=input(f'Quer reviver alguma célula ? digite {sim} para sim e {nao} para não:').lower()
        assert vivasresposta in ['s','n'], 'Digite apenas s ou n'
        break
    except AssertionError as erro:
        print(f'Erro:{erro}')
while vivasresposta!=nao:
        try:
            linhareviver=int(input('Digite qual é a linha da célula que você deseja reviver, de 0 a 9:'))
            colunareviver=int(input('Digite qual é a coluna da célula que você deseja reviver, de 0 a 9:'))
            if 0 <= linhareviver < linhas and 0 <= colunareviver < colunas:
                matriz[linhareviver][colunareviver] = vivas
            else:
                print("Posição inválida.")
        except ValueError:
            print("Erro: Digite apenas números inteiros válidos.")
        try:
            vivasresposta=input(f'Quer reviver alguma célula ? digite {sim} para sim e {nao} para não:').lower()
            assert vivasresposta in ['s','n'], 'Digite apenas s ou n'
        except AssertionError as erro:
            print(f'Erro:{erro}')
            vivasresposta=input(f'Quer reviver alguma célula ? digite {sim} para sim e {nao} para não:').lower()
mostrarmatriz(matriz)
print ('                                                    ')
```

A etapa final é a **execução da simulação**. Esse trecho do código percorre cada célula da matriz e conta quantas das suas 8 vizinhas estão vivas. Para isso, utiliza dois laços internos que

varrem a vizinhança 3x3 ao redor da célula atual. A condição if (x == i and y == j) garante que a própria célula não seja contada como vizinha. Já a verificação dos limites da matriz assegura que não ocorram acessos inválidos em posições fora da grade. Se uma célula vizinha estiver viva, o contador 'vizinhas' é incrementado. Ao final da verificação, esse valor é utilizado para aplicar as regras do Jogo da Vida. Essa lógica é executada para cada célula do tabuleiro a cada iteração do jogo. Após atualizar o estado da matriz, o programa imprime a nova geração e aguarda alguns segundos antes de prosseguir. O processo continua até que todas as células estejam mortas, quando então o jogo é encerrado com uma mensagem final. As funções auxiliares, como mostrarmatriz() e todasmortas(), são responsáveis por exibir o tabuleiro e verificar o término da simulação, vendo se todas as células estão realmente mortas, respectivamente.

```
#Loop criado para o programa rodar até todas as células estarem mortas de novo, como no início
while not todasmortas(matriz):
    for i in range(linhas):
        for j in range(colunas):
            # Verifica as 8 vizinhas(se tá viva ou morta) e armazena a quantidade de células vizinhas vivas
            vizinhas = 0
            for x in range(i - 1, i + 2):
                for y in range(j - 1, j + 2):
                    if (x == i and y == j) or not (0 <= x < linhas and 0 <= y < colunas):
                        continue
                    if matriz[x][y] == vivas:
                        vizinhas += 1

            # Aplica as regras do jogo
            if matriz[i][j] == vivas:
                if vizinhas < 2 or vizinhas > 3:
                    matriz[i][j] = mortas  #morre
            else:
                if vizinhas == 3:
                    matriz[i][j] = vivas  #revive

    mostrarmatriz(matriz)
    time.sleep(tempo)
    print ('                                                    ')
print('Fim do jogo!')
```

2.4 Ordem de codificação

Realmente peguei o código para fazer apenas na segunda semana, comecei estudando mais sobre matriz e modularização, que era um conceito que eu ainda não havia ouvido falar então pesquisei para entender melhor, além de ter estudado também funções e como criá-las, pois ainda não entendia muito bem essa parte. Na terceira semana que comecei a ter ideias e colocá-las em prática para ver se funcionavam, e a minha primeira ideia não deu certo então já tive que pensar em outra maneira de fazer, e foi aí que eu tive a ideia de usar o for da maneira apresentada acima para percorrer as vizinhas e verificar se estavam vivas ou não, que era a minha maior dúvida, como fazer pra verificar as vizinhas, se estão vivas ou mortas, mas depois que tive essa ideia do for meu código fluiu praticamente todo, terminei em uma tarde. Porém, ainda ficou faltando comentar o código, adicionar a mensagem de 'não plágio' e fazer a verificação das entradas, que eu fiz apenas na última semana, assim terminando meu código.

3. RESULTADOS E DISCUSSÕES

3.1 Manual de uso

3.1.1 Como utilizar o seu programa?

Para utilizar o programa no início o usuário será perguntado se deseja reviver alguma célula. Caso sim, ele deverá inserir as coordenadas (linha e coluna) da célula que ele deseja reviver, e isso ocorre até o usuário digitar não para a pergunta. Após o usuário digitar 'n'(não), o tabuleiro inicial já estará definido, então o programa simula a evolução da matriz automaticamente, seguindo as regras do Jogo da Vida, até que todas as células estejam mortas.

3.1.2 Qual é o conjunto de dados de entrada válido para o correto funcionamento do programa?

Os dados de entrada válidos são:

- A resposta s (sim) ou n (não) à pergunta se deseja reviver alguma célula.
- Coordenadas válidas entre 0 e 9 para linha e coluna, indicando a posição das células vivas.

  Entradas inválidas, como letras ou números fora do intervalo permitido, são tratadas com mensagens de erro e novas solicitações de entrada.

3.1.3 Quais são as saídas do programa?

As saídas do programa são:

- A exibição da matriz 10x10 a cada geração, mostrando a evolução das células vivas (O) e mortas (X).
- Uma mensagem final indicando o fim do jogo quando todas as células estiverem mortas.
- Uma mensagem inicial dizendo: Seja bem-vindo ao jogo da vida!

3.1.4 Como são geradas as saídas (processamento para obtê-las)?

As saídas são geradas por meio de:

- Contagem das vizinhas vivas de cada célula.
- Aplicação das regras do Jogo da Vida (solidão, sobrevivência, superpopulação e reprodução).
- Atualização da matriz a cada iteração, com pausa temporária para visualização.

3.1.5 Quais os testes efetuados (durante e ao final do desenvolvimento) e quais os resultados obtidos? Quais erros ocorreram nos testes? Em que situação o seu programa não funcionaria?

Eu realizei vários testes, primeiro tentei imprimir a matriz inicial e consegui, depois testei o funcionamento do jogo em si, ou seja, joguei ele, coloquei as posições que eu queria reviver e depois assisti o código rodando, e nos primeiros testes estava dando erro, e mais para frente eu percebi que a causa desse erro era porque eu criei uma variável contadora para contabilizar as vizinhas vivas de cada célula analisada, porém eu esqueci de colocar para zerar a quantidade de células vizinhas vivas toda vez que mudava a célula analisada, então acabava dando superpopulação em todas e todas acabavam morrendo. Então após consertar isso meu jogo funcionou perfeitamente. Porém, uma semana depois eu abri o meu código para adicionar a verificação das entradas e tive que testar para ver se estava realmente funcionando após essas adições, e estava. Acredito que meu programa funcionaria em qualquer situação, não achei nenhuma falha nele, mas talvez exista alguma e eu não tenha percebido.

4. CONCLUSÃO

Acredito que todos os requisitos foram cumpridos e acredito que uma maneira de melhorar ainda mais esse código seria deixá-lo mais bonito visualmente para o usuário, o que eu poderia tentar implementar mais para frente, porém agora eu foquei mais no funcionamento do código do que na sua aparência final.

5. BIBLIOGRAFIA CONSULTADA

HASHTAG PROGRAMAÇÃO. *Como usar def no Python - Funções Explicadas em 10min*. YouTube, 6 nov. 2020. Disponível em: https://youtu.be/u8piwlScXT8. Acesso em: 13 abril 2025.

PYTHONANDO. *Modularização com Python | Domine a importação de módulos*. YouTube, 3 nov. 2020. Disponível em: https://youtu.be/_bZe0sh0tCs. Acesso em: 15 abril 2025