Jogo 2048

Sayumi Mizogami Santana

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

Departamento de Tecnologia

mizogami15@gmail.com

Resumo. *Este relatório descreve a solução desenvolvida para o jogo 2048 em linguagem Python. O sistema utiliza uma estrutura de repetição para manter o jogo em um loop contínuo até que o usuário escolha encerrá-lo e também para a manipulação da matriz, que é um dos pontos mais importantes para o jogo. Além disso, faz uso das estruturas condicionais para verificar as condições de derrota ou vitória, bem como para executar os movimentos necessários. Ao final da execução, o programa apresenta os resultados ao usuário e o histórico dos jogos anteriores, cumprindo assim o objetivo do problema proposto de forma eficaz.*

1. Introdução

Com o intuito de proporcionar um melhor aprendizado, os estudantes do primeiro semestre de Engenharia de Computação da UEFS foram desafiados a desenvolver o código do jogo 2048, com a inclusão das regras, pontuação e contagem da quantidade de jogadas válidas realizadas, permitindo também a repetição do jogo até que o usuário deseje encerrá-lo.

O jogo 2048 foi criado pelo italiano Gabriele Cirulli e apresenta uma mecânica simples, mas desafiadora. O jogo começa com dois números dentro de um quadro, podendo ser eles 2 ou 4. O jogador escolhe os movimentos a serem realizados, com o intuito de realizar as somas dos pares de números iguais. Vence o jogo quando, a partir dessas somas, surge o número 2048. No momento em que o jogador inserir cada movimento, surge um novo número, também podendo ser 2 ou 4. A derrota ocorre quando não é mais possível realizar movimentos em nenhuma direção, devido à ausência de números idênticos um ao lado do outro e pela falta de espaço para novos movimentos.

Com base no contexto apresentado, foi desenvolvido um programa em linguagem Python. Durante o processo de desenvolvimento do sistema, foram identificados os seguintes pontos a serem implementados: a realização dos movimentos dentro de uma matriz, a identificação dos números iguais, a soma deles e o sorteio dos números que deveriam aparecer dentro da matriz para cada movimento realizado.

O objetivo deste relatório é descrever a solução desenvolvida para o funcionamento do programa. A seguir, serão apresentadas as estratégias utilizadas, juntamente com a descrição dos meios empregados para concretizar essa solução.

2. Desenvolvimento

Inicialmente, foi necessário gerar uma matriz com 16 espaços para formar o quadro onde os números serão exibidos na tela. Uma representação comum para um conjunto de dados bidimensionais em Python é uma lista de listas. Em particular, podemos representar uma matriz bidimensional como uma lista de linhas, com cada linha contendo uma lista de valores [Goodrich et al. 2013].

No código em questão, foram utilizadas listas para criar a matriz necessária para o funcionamento do jogo e também para realizar os movimentos necessários. As listas, são um tipo de variável que permite o armazenamento de vários valores, acessados por um índice [Menezes 2014].

Além das listas, foi necessário o uso de funções para uma melhor organização e funcionamento do programa. Essas funções incluem a impressão das instruções e da matriz do jogo, a cópia da matriz, os movimentos, o sorteio dos números, a soma dos elementos iguais e a verificação das igualdades. A seguir, serão apresentados os passos essenciais para a elaboração do código, acompanhados de uma explicação detalhada sobre a utilidade de cada função desenvolvida.

Como realizar as repetições do jogo? Em jogos, é comum permitir que o jogador reinicie do começo após sofrer uma derrota. No código em questão, foram inseridos dois loops *while* no início, um para poder recomeçar a cada fim de jogo e outro para continuar as jogadas. A instrução *while* permite especificar a execução repetitiva de um conjunto de instruções enquanto uma determinada expressão do tipo lógico tiver o valor verdadeiro [Martins 2015]. Isso possibilitou a repetição de todas as ações dentro do programa.

Como armazenar as letras inseridas pelo usuário ao realizar os movimentos e impedir que ele digite uma informação errada? Para realizar esse objetivo foi necessário utilizar o comando de entrada *input*. A função input é utilizada para solicitar dados do usuário. Ela recebe um parâmetro, que é a mensagem a ser exibida, e retorna o valor digitado pelo usuário [Menezes 2014].

Após a inserção dos dados, a letra escolhida pelo jogador é submetida a uma estrutura condicional *if* para verificar se está de acordo com as opções pré-estabelecidas pelo sistema. As condições, servem para selecionar quando uma parte do programa deve ser ativada e quando deve ser simplesmente ignorada [Menezes 2014]. Caso seja inserido um caractere diferente do permitido, a solicitação de entrada de dados é repetida por meio de um loop *while* até que uma informação correta seja adicionada, evitando assim que o jogador digite uma letra errada.

2.1 Funções

A função é um dispositivo que agrupa um conjunto de instruções, de modo que elas possam ser executadas mais de uma vez em um programa [Lutz and Ascher 2007]. No programa elaborado, foram utilizados treze funções, cada uma com uma finalidade.

Função de impressão

Para uma melhor a organização do código, foi preciso criar duas funções de impressão: uma destinada às instruções do jogo e outra para exibir a matriz. Para isso, foi utilizado

o comando de saída, *print*, em conjunto com a estrutura de repetição *for*, utilizada para percorrer todos os elementos da matriz. O loop *for* é um iterador de sequências genéricas no Python: ele pode percorrer os itens de qualquer objeto sequencial ordenado [Lutz and Ascher 2007]. Além disso, empregou-se uma instrução condicional *if*, para garantir a formatação adequada dos números com os espaçamentos necessários.

Função para sortear

Como o jogo requer que comece com dois valores dentro da matriz e que um valor seja adicionado a cada movimento realizado, foi fundamental o uso de módulos da biblioteca do Python. Esses módulos são pacotes de nomes que normalmente correspondem aos arquivos-fonte e servem como bibliotecas de ferramentas para uso em outros arquivos e programas [Lutz and Ascher 2007]. No código, foi utilizado o módulo *random* para sortear os números e as posições e a estrutura *while* para inseri-los na matriz.

Função para copiar a matriz

Para que o jogo funcione corretamente, é necessário que a cada movimento inserido pelo usuário, um número seja adicionado ao quadro. No entanto, se o usuário fizer um movimento inválido que não resulte em mudanças, o número não deve ser acrescentado. Para garantir essa lógica do jogo, utilizou-se uma função para copiar a matriz. Para realizar essa duplicação na função, fez-se uso da estrutura de repetição *for* para reescrever todos os números da matriz original para uma matriz temporária. Em seguida, comparou-se a cópia com a original. Caso forem iguais, nenhum número é acrescentado, se forem diferentes, um número é adicionado.

Função de verificação de espaços

Na verificação da derrota no jogo, é necessário examinar a matriz para determinar a condição de perda do jogador. Para que a derrota ocorra, é fundamental que não haja mais espaços vazios nem números idênticos na matriz. Para identificar espaços vazios, foi utilizado uma estrutura de repetição *for* e uma estrutura condicional *if* a fim de verificar a presença de espaços vazios. Quando um espaço vazio é encontrado, a função retorna o valor booleano *True*, caso contrário, retorna *False*.

Função de arrumação

Foram necessárias duas funções para organizar as matrizes de acordo com o movimento escolhido pelo usuário: a função arrumação crescente e a decrescente. Inicialmente, foi preciso verificar cada linha e coluna. Para isso, utilizou-se uma lista temporária que pudesse conter quatro números em sequência, permitindo realizar as somas dos números iguais e executar os movimentos necessários.

A cada ciclo do *loop*, comparou-se o elemento com o próximo. Para esse fim, implementou-se a estrutura *for* juntamente com o *if* para percorrer a lista enquanto se realizavam as somas e os movimentos. Contudo, nos casos dos movimentos para baixo e para direita, foi utilizado a arrumação decrescente, pois era necessário inserir os números na lista temporária na ordem inversa para que fossem impressos corretamente na matriz. Para isso, utilizou-se o *for* com a opção de incremento negativo da função *range*, que permite percorrer os números em ordem decrescente.

Como contar a pontuação do jogo? A pontuação do jogo aumenta a cada vez que o jogador conseguir realizar a soma dos números iguais. Como as somas foram feitas na função de arrumação, a contagem das pontuações ocorreram em conjunto. Para isso foi utilizado uma variável no escopo global, que é um espaço de nome onde ficam as variáveis criadas no nível superior de um arquivo de módulo [Lutz and Ascher 2007]. A cada soma realizada, é acrescentado essa soma nas pontuações, sendo apresentadas na tela no momento que o usuário estiver inserindo as letras dos movimentos.

Função dos movimentos

Para permitir que o usuário escolha o movimento, foi aplicada a estrutura condicional *if*. Com isso, foi possível, por meio de quatro funções, selecionar as letras "W" para subir, "S" para descer, "A" e "D" para mover para os lados. As funções correspondentes são chamadas com base na escolha do usuário. Dentro de cada função de movimentação, foi incluída a função de arrumação, que tem como objetivo ordenar e somar os elementos da matriz. As movimentações seguem um mesmo padrão com apenas alguns detalhes diferentes que serão expostos em seguida.

- **Movimentar para esquerda: “A”**

Nessa função, inicialmente, foi necessário preencher a lista temporária com zeros para não haver números anteriores. Em seguida, utilizou-se um loop *for* para percorrer os quatro números de cada linha da matriz, inserindo-os na lista temporária. Após isso, chamou-se a função de arrumação crescente que realiza as somas e os movimentos dentro da lista e, finalmente, inseriram-se os quatro números de volta na matriz. Esse processo se repete quatro vezes para cada linha na matriz.

- **Movimentar para cima: “W”**

Com a movimentação para cima, seguiu-se a mesma lógica da movimentação para a esquerda, diferindo apenas no fato de que os números foram inseridos na lista temporária a partir das colunas, ao invés das linhas da matriz.

- **Movimentar para direita “D”**

Nessa função, foi necessário inserir os números na lista temporária de forma que, quando retornassem para a matriz, estivessem organizados de maneira diferente das funções "A" e "W". Como o movimento para a direita visa trazer todos os elementos possíveis da esquerda para a direita, tornou-se necessário ordenar os números inversamente. Para isso foi utilizado a função arrumação decrescente, já citada anteriormente. Nela foi utilizado a função *range* com a implementação de passos negativos, iniciando-se o loop for pelo número 3 e terminando em -1. Os números organizados nas linhas da matriz foram inseridos na lista temporária de forma inversa e, em seguida, introduzidos de volta à matriz, garantindo que os números fossem impressos na ordem correta.

- **Movimentar para baixo “S”**

Essa função seguiu o mesmo fundamento do movimento para a direita, com a única diferença de que, em vez de inserir os números das linhas da matriz, são inseridos os números da coluna.

Função para verificação de igualdades

Foram implementadas duas funções para verificar se o jogador perdeu. A necessidade de realizar essa verificação decorreu da possibilidade de, mesmo com a matriz estando completamente preenchida, o jogador ainda poder ganhar o jogo se houvesse números idênticos um ao lado do outro. Para isso, utilizou-se um loop *for* para comparar cada número com o seu sucessor. Foi criada uma lista temporária para armazenar as linhas e colunas que precisavam ser comparadas. Os resultados dessas comparações foram retornados ao código principal através de uma variável booleana que indicava se havia números iguais ou não.

2.2 Encerrando o jogo

Nos jogos, é uma prática comum exibir um registro das pontuações alcançadas pelo jogador. Para incorporar essa funcionalidade ao código, foi utilizado uma variável do tipo lista. Ao final de cada partida foi registrado as pontuações obtidas e as jogadas válidas que ocorreram durante o jogo, e essas informações são apresentadas ao usuário ao perder ou ganhar, por meio do comando *print*, associado a estrutura de repetição *for*. Após a exibição desse histórico, uma pergunta é apresentada ao jogador, dando-lhe a escolha de prosseguir jogando caso insira o caractere "S" ou encerrar o jogo com o caractere "N". Dessa forma, o jogo é encerrado de acordo com a preferência do jogador.

3. Conclusão

Este relatório apresenta a solução criada para elaborar o jogo 2048. O programa foi desenvolvido em linguagem Python, e solucionou o problema proposto por meio da implementação de estruturas condicionais e de repetição, de funções e de bibliotecas da linguagem utilizada.

Os desafios surgidos durante a construção do código foram expostos na seção de desenvolvimento deste relatório. Cada ponto citado, foi devidamente solucionado, resultando na conclusão bem-sucedida do jogo 2048, com pontuações, movimentos e respeitando grande parte das regras existentes.

Referências

Lutz, M. and Ascher, D. (2007). Aprendendo Python, *2*.ed. Bookman.

McKinney, W. (2018) Python para Análise de Dados: Tratamento de Dados com Pandas, NumPy e IPython, 1.ed. Novatec.

Furgeri, S. (2021). Introdução à programação em Python. Senac.

Menezes, N. (2014). Introdução à Programação com Python Algoritmos e lógica de programação para iniciantes 2 ed. Novatec.

Martins, J. (2015). Programação em Python. Introdução à Programação Utilizando Múltiplos Paradigmas. 1.ed. IST Press.

Goodrich, M. T. , Tamassia, R. and Goldwasser, M. H. (2013). Data Structures and Algorithms in Python. 1.ed. Wiley Global Education.