

Sessão 02

Pedra, papel, tesoura e lógica

Professor Thiago Goveia

Objetivos da sessão:

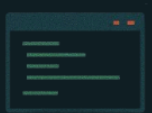
- Representar graficamente a lógica do jogo "Pedra, Papel e Tesoura" para auxiliar o desenvolvimento do algoritmo
- Revisar os recursos básicos da linguagem C: entrada e saída, estruturas condicionais, laços, vetores e matrizes
- Utilizar números aleatórios para introduzir a ideia de "sorte" no jogo.
- Implementar diferentes versões do jogo levando em consideração diferentes estratégias e recursos, considerando as vantagens e desvantagens de cada uma.

Materiais:

- [Repositório](#) do projeto criado na Sessão 01 no GitHub.
- Ferramenta de fluxograma (opcional, pode ser papel e caneta).

Sumário:

1. Relembrando e Modelando o Jogo	2
1.1 As Regras Clássicas	2
1.2 Modelando com Fluxograma	2
2. A Caixa de Ferramentas em C para o Jogo	3
2.1 Entrada e Saída (printf e scanf)	3
2.2 Geração de Números Aleatórios (rand e srand)	3
2.3 Estruturas Condicionais (if/else if/else)	4
2.4 Laços de Repetição (while ou do-while)	4
2.5 Vetor e Matriz ([] e [][])	4
3. Mão na massa! Implementando Pedra, Papel, Tesoura	4
3.1 Versão 01 - Utilizando apenas ifs	4
3.2 Versão 02 - Utilizando apenas ifs não aninhados	5
3.3 Versão 03 - Inclusão do laço de repetição	5
4. Mão na Massa! Implementando Pedra, Papel, Tesoura, Lagarto Spock	6
4.1 Versão 04 - Utilizando apenas ifs (aninhados ou não)	6
4.2 Versão 05 - Utilizando vetores	7
4.4 Versão 06 - Utilizando matrizes	7
5. Indo além! (Para casa)	8
5.1 Pedra, Papel, Tesoura, Lagarto Spock com matrizes	8
5.2 Pedra, Papel, Tesoura, Lagarto Spock com vetores	8



Conteúdo da sessão:

1. Relembrando e Modelando o Jogo

1.1 As Regras Clássicas

Antes de programar, precisamos ter certeza de que entendemos a lógica. O jogo "Pedra, Papel e Tesoura" possui três regras de vitória:

- Pedra ganha de Tesoura (amassando-a).
- Tesoura ganha de Papel (cortando-o).
- Papel ganha de Pedra (embrulhando-a).
- Se ambos os jogadores escolherem a mesma opção, ocorre um **empate**.

Jogue com a sua equipe para definir a ordem da "posse" do computador. Façam uma "melhor de 3" com todos os integrantes da equipe.

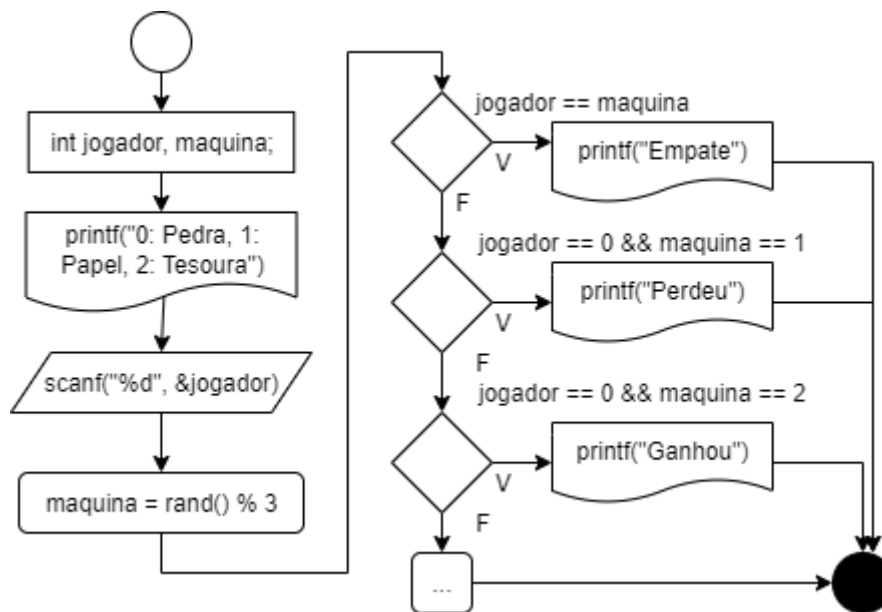
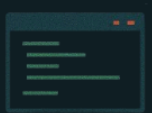
1.2 Modelando com Fluxograma

Um fluxograma é um desenho que representa os passos de um algoritmo. Ele nos ajuda a organizar as ideias antes de escrever o código. Para o nosso jogo, o fluxo é :

1. **Início**
2. Apresentar as opções ao jogador.
3. Ler a escolha do jogador.
4. Gerar a escolha aleatória do computador.
5. Comparar as duas escolhas para definir o resultado (Vitória, Derrota ou Empate).
6. Exibir o resultado.
7. Perguntar se o jogador quer jogar novamente.
8. Se sim, voltar ao passo 2. Se não, **Fim**.

A seguir está o esboço de um fluxograma que não considera a repetição.

Com sua equipe, complete o [fluxograma](#) incluindo o laço de repetição.



2. A Caixa de Ferramentas em C para o Jogo



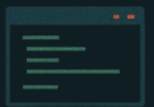
Para construir nosso jogo, usaremos cinco ferramentas essenciais da linguagem C :

2.1 Entrada e Saída (**printf** e **scanf**)

- **printf()**: Exibe mensagens na tela para o usuário (ex: "Escolha sua jogada:").
- **scanf()**: Lê um valor digitado pelo usuário e o armazena em uma variável.

2.2 Geração de Números Aleatórios (**rand** e **srand**)

- **rand()**: Gera um número inteiro pseudo-aleatório. É incluída pela biblioteca **stdlib.h**.
- Para limitar o resultado (ex: entre 0 e 2), usamos o operador de módulo (%): **rand() % 3**.
- **srand(time(NULL))**: "Alimenta" a função **rand()** com uma semente baseada no tempo atual, garantindo que os números sejam diferentes a cada execução do programa. Deve ser usada apenas uma vez, no início do código. A função **time** é incluída pela biblioteca **time.h**.



2.3 Estruturas Condicionais (**if/else** **if/else**)

É o cérebro do nosso jogo. Usamos essa estrutura para comparar a jogada do jogador com a do computador e aplicar as regras para determinar o vencedor.

2.4 Laços de Repetição (**while** ou **do-while**)

Para que o jogo não termine após uma única rodada, colocamos toda a lógica dentro de um laço que continua executando enquanto o jogador desejar continuar.

2.5 Vetor e Matriz (**[]** e **[][]**)

Utilizamos estas estruturas de dados para separar a **lógica do jogo** dos **dados do jogo**. Ao utilizar vetor e matriz, aprimoramos o código, deixando-o mais limpo, mais fácil de manter e escalável/genérico.

3. Mão na massa! Implementando Pedra, Papel, Tesoura

Crie um novo projeto no VSCode ou OnlineGDB e dentro dele o arquivo **main.c**. Comece incluindo as bibliotecas necessárias e declarando as variáveis que armazenarão as jogadas e o controle do laço.

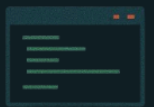
Atenção! Lembre-se de criar um novo commit pelo menos para cada nova versão implementada!

3.1 Versão 01 - Utilizando apenas ifs

Nesta primeira versão, vamos traduzir as regras da forma mais direta possível. Usaremos **ifs** aninhados (um dentro do outro). A lógica é: primeiro, verificamos a escolha do jogador. Depois, dentro de cada verificação, analisamos todas as possibilidades para a escolha do computador.

Instruções:

1. Crie a estrutura inicial do programa com as bibliotecas e a função **main**.
2. Declare as variáveis para a jogada do jogador e do computador.



3. Peça para o jogador digitar sua escolha (1 para Pedra, 2 para Papel, 3 para Tesoura).
4. Gere a jogada do computador.
5. Implemente a lógica de decisão com `ifs` aninhados.

3.2 Versão 02 - Utilizando apenas ifs não aninhados

A versão com `ifs` aninhados funciona, mas pode ficar longa e repetitiva. Agora, vamos refatorar o código para reduzir o número de estruturas `if/else` a partir do uso de operadores lógicos (`&&` para "E", `||` para "OU"). A ideia é agrupar todas as condições de vitória do jogador em uma única expressão.

Instruções:

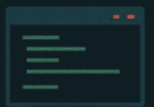
1. Comece verificando a condição mais simples: o empate.
2. Em seguida, crie um `else if` que contenha todas as três condições em que o jogador vence, unidas pelo operador `||` (OU).
3. Se não for empate e o jogador não venceu, a única possibilidade restante é a vitória do computador. Use um `else` para tratar esse caso.

3.3 Versão 03 - Inclusão do laço de repetição

Para que o jogo possa ser jogado várias vezes sem precisar reiniciar o programa, vamos colocar toda a lógica dentro de um laço de repetição. O laço `do-while` é ideal aqui, pois garante que o jogo seja executado pelo menos uma vez antes de perguntar se o jogador quer continuar.

Instruções:

1. Declare uma variável do tipo `char` para controlar a continuação (ex: `char continuar = 's';`).
2. Envolver todo o código da rodada (desde a exibição do menu até a verificação do vencedor) dentro de um bloco `do {...}`.
3. Ao final do bloco, pergunte ao jogador se ele deseja jogar novamente e leia a resposta para a variável `continuar`.
4. Feche o laço com a condição `while (continuar == 's' || continuar == 'S');`.



4. Mão na Massa! Implementando Pedra, Papel, Tesoura, Lagarto e Spock

Pedra-papel-tesoura-lagarto-Spock é uma expansão do jogo clássico, incluindo outras duas armas adicionais: o lagarto (formado pela mão igual a uma boca de fantoche) e Spock (formada pela saudação dos vulcanos em Star Trek). As regras são as seguintes:



- Tesoura corta papel
- Papel cobre pedra
- Pedra esmaga lagarto
- Lagarto envenena Spock
- Spock esmaga tesoura
- Tesoura decapita lagarto
- Lagarto come papel
- Papel refuta Spock
- Spock vaporiza pedra
- Pedra amassa tesoura

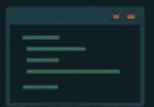


4.1 Versão 04 - Utilizando apenas ifs (aninhados ou não)

Vamos adicionar "Lagarto" e "Spock" ao jogo, seguindo as regras listadas acima.

Instruções:

1. Defina um número para cada jogada: 1-Pedra, 2-Papel, 3-Tesoura, 4-Lagarto, 5-Spock.
2. Adapte o código da Versão 02 ou Versão 03 para contemplar as novas regras.
3. Avalie as conclusões da equipe juntamente com a turma.



4.2 Versão 05 - Utilizando vetores

Vamos voltar ao jogo clássico (3 opções) e aprender uma técnica mais elegante. Em vez de escrever as regras no código, vamos armazená-las em um vetor (array).

Instruções:

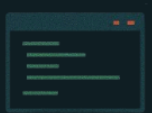
1. Vamos usar 1-Pedra, 2-Papel, 3-Tesoura.
2. A ideia é criar um vetor onde o *índice* representa uma jogada e o *valor* armazenado nesse índice é a jogada que a derrota.
3. Declare o vetor no início do seu código. Como Pedra (1) perde para Papel (2), Papel (2) perde para Tesoura (3), e Tesoura (3) perde para Pedra (1), o vetor fica assim: `int perdePara[1] = {0, 2, 3, 1};` (A posição 0 não é usada para manter a relação direta entre o número da jogada e o índice).
4. Substitua a lógica `if/else` complexa pela verificação do vetor.
5. Como poderíamos reduzir o tamanho do vetor `perdePara`?

4.4 Versão 06 - Utilizando matrizes

Para criarmos soluções mais abrangentes e elegantes, a solução mais poderosa é uma matriz de decisão, que funciona como uma "tabela de consulta" de resultados.

Instruções:

1. Crie uma matriz 3x3. As linhas serão a jogada do jogador e as colunas a do computador. O valor no cruzamento `[linha][coluna]` indicará o resultado. Usaremos 1 para vitória do jogador, -1 para derrota e 0 para empate.
2. Declare e inicialize a matriz de regras no início do seu código.
3. **Implementação:** Dentro do laço do jogo, a lógica de decisão se resume a uma consulta na matriz.



5. Indo além! (Para casa)

5.1 Pedra, Papel, Tesoura, Lagarto Spock com matrizes

Utilizando o conhecimento da **Versão 06 (matrizes)**, sua tarefa é implementar a versão completa do jogo com as 5 opções. O programa deve estar completo, com o laço de repetição para jogar várias vezes.

5.2 Pedra, Papel, Tesoura, Lagarto Spock com vetores

Avalie se é possível implementar o jogo utilizando vetores, como foi feito na **Versão 05**. Se sim. Apresente uma solução.



Próximas sessões: Pedra, Papel, Tesoura com:

Programação Web

Arduino

Machine Learning (IA)

Não percam. Até lá!

