Bem-vindos à Guilda dos Desenvolvedores: Git & GitHub!

Hoje vamos desbloquear duas ferramentas PODEROSAS que todo criador de jogos (e programador) usa! Por que isso AGORA? Para vocês salvarem seu progresso como PROs (chega de ProjetoFinal_V2_AgoraVai.zip!), entregarem suas atividades de forma organizada e começarem a construir seu portfólio de projetos!

Nossas ferramentas: **Git** e **GitHub**! Prepare-se para uma jornada de descobertas que vai transformar completamente a maneira como você gerencia seus projetos de programação. Com estas duas ferramentas essenciais, você estará dando um passo gigante em direção a se tornar um desenvolvedor profissional.

por Mayouma Studio

Git: Sua Máquina do Tempo Pessoal para Código!

O que é Git?

É um **Sistema de Controle de Versão (VCS)**. Pense nele como um sistema de "save points" superavançado para seus projetos de código. Ele roda no SEU computador, sem necessidade de internet após instalado.

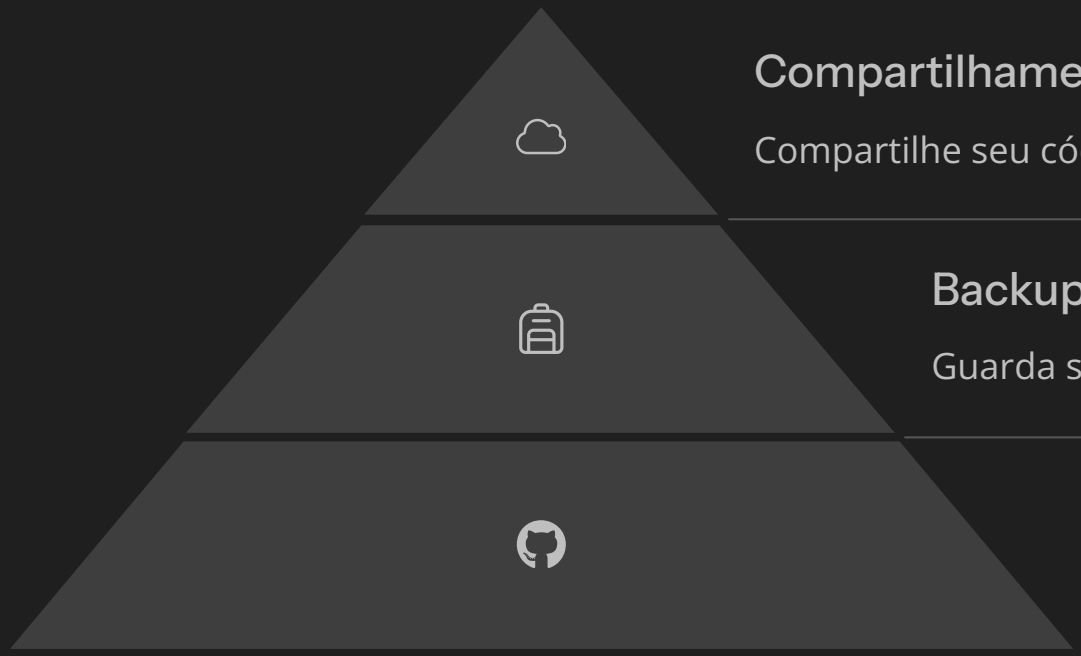
O que ele faz?

Registra o histórico de todas as alterações que você faz no seu código. Permite que você **volte para versões anteriores** se algo der errado (tipo dar "load" no último save).

Por que usar?

Ajuda a organizar o trabalho, mesmo que você seja o único "player" no projeto por enquanto. É como ter um diário detalhado de cada linha de código que você escreve, apaga ou modifica.

GitHub: Sua Fortaleza na Nuvem (e Ponto de Encontro)!



Compartilhamento

Compartilhe seu código com o mundo

Backup na Nuvem

Guarda seus projetos Git com segurança

Plataforma Online

Um site que hospeda seus repositórios Git

O GitHub é como um "Google Drive" ou "Dropbox" turbinado para seus projetos de código. É onde milhões de desenvolvedores colaboram em projetos incríveis, de jogos indie a softwares gigantes. Será nosso "portal de entrega" das atividades!

Lembre-se sempre: **Git é a ferramenta, GitHub é o lugar** onde você guarda e compartilha o trabalho feito com Git. Esta combinação poderosa é usada por todos os desenvolvedores profissionais.

Destravando o Inventário: Variáveis e Tipos de Tesouro!

string - Textos

Para guardar nomes, descrições, mensagens. Exemplos: "HeróiValente", "Poção de Cura", "Fase 1".

int - Números Inteiros

Para contar, definir níveis, pontuação. Exemplos: 100, -5, 0, número de vidas, quantidade de moedas.

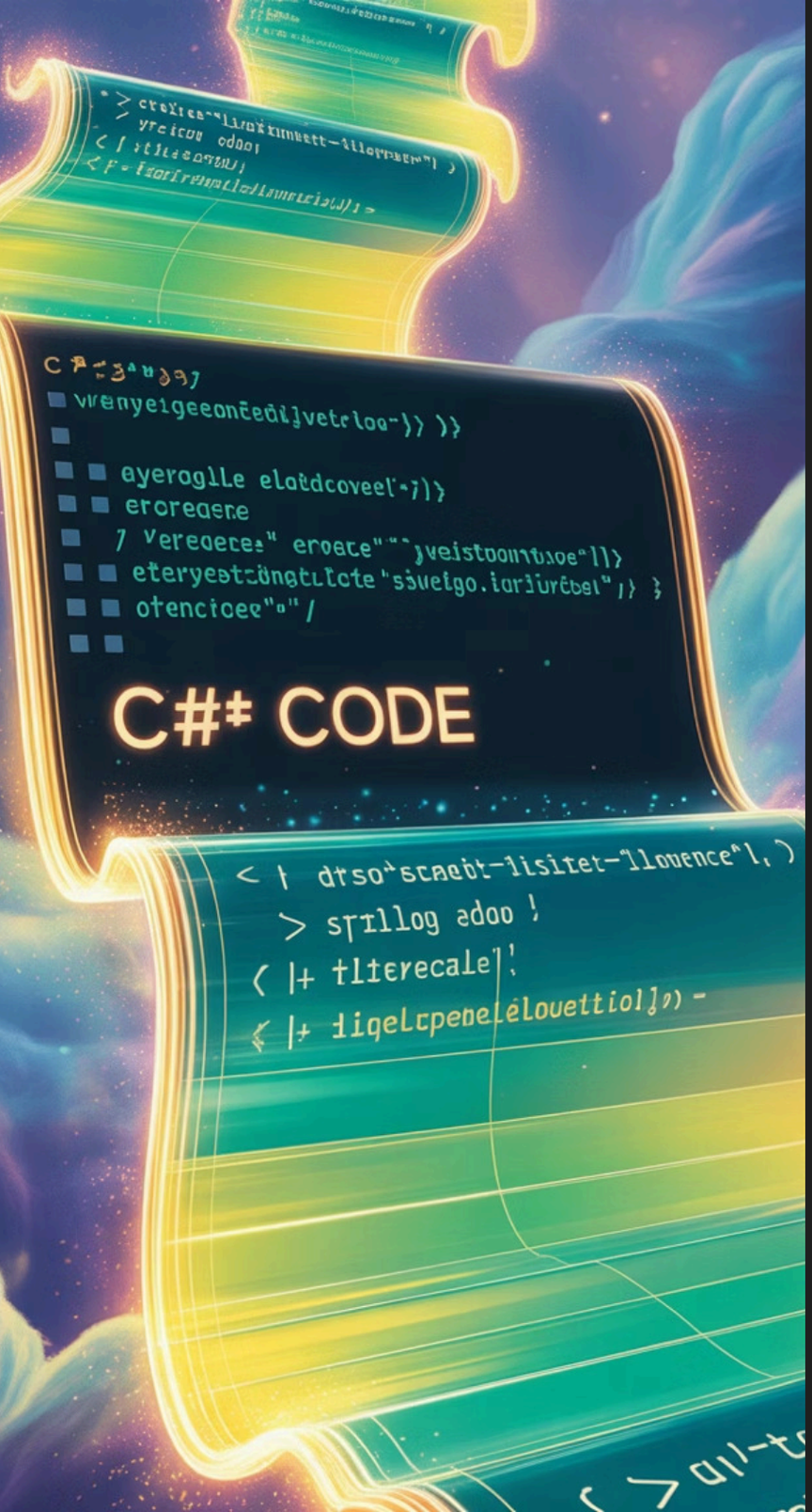
float - Números Decimais

Para valores precisos como velocidade, dano. Exemplos: 2.5f, 99.9f, 0.01f (lembre do f!).

bool - Verdadeiro/Falso

Para estados, condições, verificações. Exemplos: true, false, jogadorTemChave, estáVivo.

Variáveis são como "caixas" na memória para guardar dados. Cada variável tem um **Nome** (como você vai chamá-la), um **Tipo** (que tipo de dado ela guarda) e um **Valor** (o conteúdo atual). Use a nomenclatura camelCase (ex: vidaAtual, nomeJogador) para manter seu código organizado.



Guardando e Exibindo seu Loot!

1

Declarar

Criar a variável com seu tipo

2

Inicializar

Atribuir um valor inicial

3

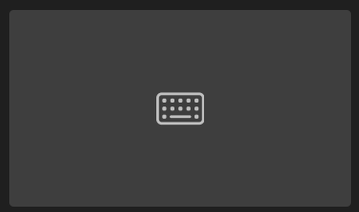
Exibir

Mostrar o valor na tela

A forma mais comum de trabalhar com variáveis é declarar e inicializar ao mesmo tempo: **tipo nomeVariavel = valorInicial;** Por exemplo: **int moedas = 100;** ou **string missaoAtual = "Salvar a Princesa";**

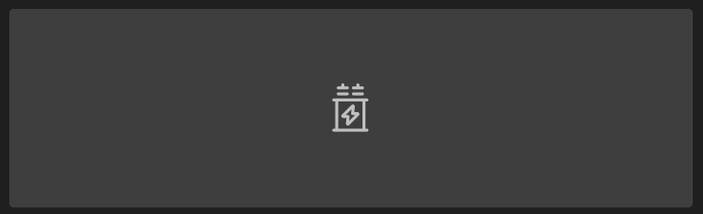
Para exibir valores no console, use a interpolação de strings, que é o jeito mais moderno e claro: **Console.WriteLine($"Você tem {moedas} moedas para a missão: {missaoAtual}!");** Este método permite inserir variáveis diretamente dentro do texto, tornando seu código mais legível.

Falando com o Oráculo: ReadLine() e a Alquimia do Parse!



Console.ReadLine()

Captura a entrada do usuário como string



int.Parse() ou float.Parse()

Converte a string para número

Variável numérica

Armazena o valor convertido

Lembre-se que Console.ReadLine() **sempre** retorna uma string, mesmo quando o usuário digita números. Se o jogador digita '25', você recebe a STRING "25", não o número 25. Para usar como número, precisamos converter com os métodos Parse:

```
Console.Write("Digite seu nível: ");
int nivelJogador = int.Parse(Console.ReadLine());
Console.Write("Dano da sua arma: ");
float danoArma = float.Parse(Console.ReadLine());
```

Cuidado: Se o texto não for um número válido (Ex: "abc"), Parse causará um erro!

A Matemática dos Deuses: Operadores Aritméticos!

Adição (+)

int soma = 10 + 5; // soma será 15

Subtração (-)

int vidaRestante = 100 - 30; // vidaRestante será 70

Multiplicação (*)

int danoTotal = 25 * 3; // danoTotal será 75

Divisão (/)

int resultadoDivInt = 10 / 4; // resultadoDivInt será 2 (divisão inteira!) float resultadoDivFloat = 10.0f / 4.0f; // resultadoDivFloat será 2.5

Também temos o operador % (Módulo - Resto da Divisão): int resto = 10 % 3; // resto será 1 (10 dividido por 3 é 3, com resto 1). Este operador é útil para ciclos, verificar se é par/ímpar, entre outros.

Atenção: Na divisão, se dois números são int, o resultado será int (parte decimal é truncada). Para resultado decimal, pelo menos um deve ser float!

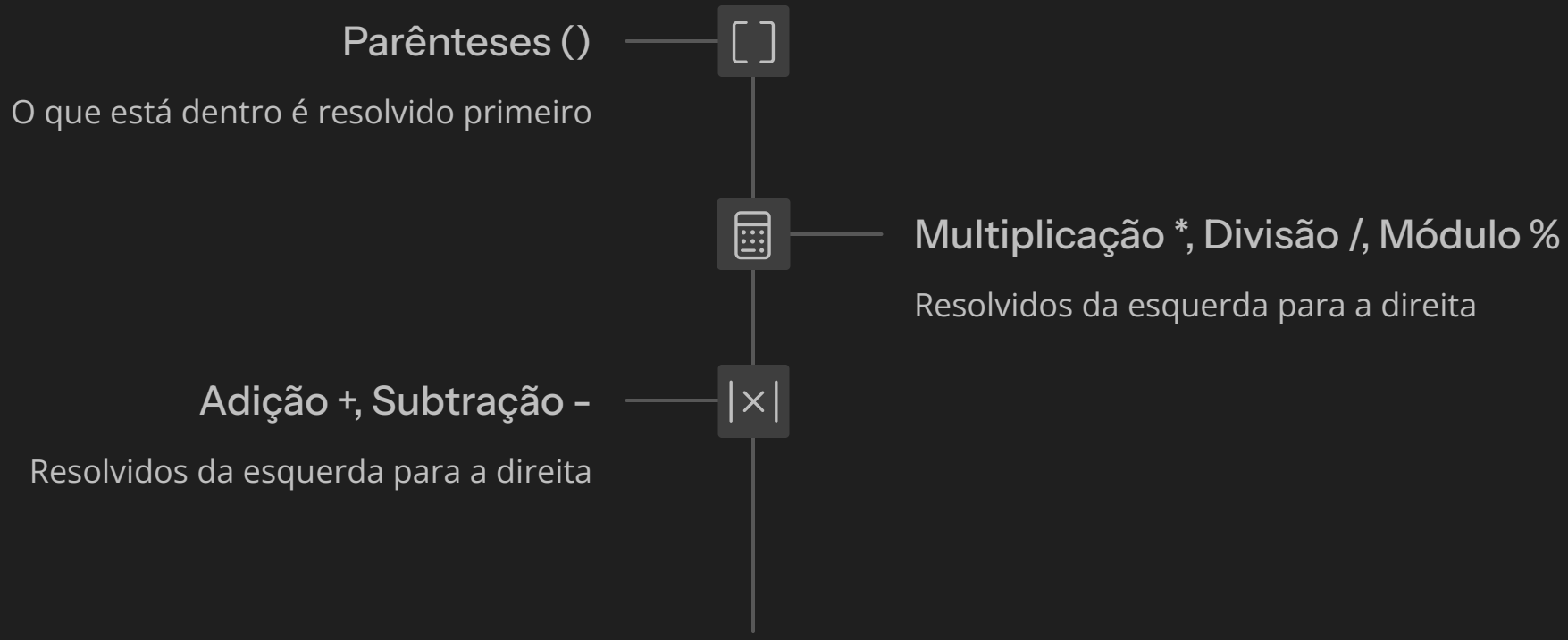
Operadores em Ação: Dano, XP e Poções!

Calculando Dano: int ataqueHeroi = 50; int defesaInimigo = 20; int danoCausado = ataqueHeroi - defesaInimigo; Console.WriteLine($"Você causou {danoCausado} de dano!");

Ganhando XP: int xpAtual = 1000; int xpMonstro = 150; xpAtual = xpAtual + xpMonstro; // Ou xpAtual += xpMonstro; Console.WriteLine($"XP Atualizado: {xpAtual}");

Dividindo Itens: int totalPocoes = 7; int membrosEquipe = 3; int pocoesPorMembro = totalPocoes / membrosEquipe; // Resultado: 2 int pocoesSobrantes = totalPocoes % membrosEquipe; // Resultado: 1 Console.WriteLine($"Cada um recebe {pocoesPorMembro}, sobram {pocoesSobrantes} poções.");

A Ordem Secreta: Precedência dos Operadores



Assim como na matemática da escola, C# tem uma ordem para resolver as expressões. Veja estes exemplos:

```
int resultado = 5 + 3 * 2; // resultado é 11 (3*2 primeiro, depois +5)

int resultadoComParenteses = (5 + 3) * 2; // resultado é 16 ((5+3) primeiro, depois *2)
```

Na dúvida, use parênteses para deixar claro e garantir a ordem que você quer! Isso torna seu código mais legível e evita surpresas nos resultados dos cálculos.

Prática Rápida: Sua Primeira Loja de Itens!

Poção de HP

Restaura pontos de vida. Preço: 10.5 moedas de ouro por unidade. Perfeito para aventureiros que enfrentam inimigos poderosos.

Flecha Mágica

Causa dano adicional em inimigos. Preço: 2.0 moedas de ouro por unidade. Essencial para arqueiros em missões perigosas.

Loja Completa

Nosso programa calculará o custo total da compra combinando diferentes quantidades de itens e seus respectivos preços.

Vamos criar um programa que define o preço dos itens (float precoPocao = 10.5f; float precoFlecha = 2.0f;), pergunta ao usuário quantas unidades quer comprar de cada (usando ReadLine e Parse), calcula o custo total (float custoTotal = (precoPocao * qtdPocoes) + (precoFlecha * qtdFlechas);) e exibe o resultado.

Missão Principal: Ficha de Personagem Interativa + Stats!

AB

Coletar Dados

Nome do Personagem (string), Classe (string), Força Base (int), Dano da Arma (int), Bônus de Habilidade (float)

||||

Calcular Stats

Poder de Ataque Total = Força Base + Dano da Arma Dano Crítico Estimado = Poder de Ataque Total × Bônus de Habilidade

Exibir Ficha Completa

Mostrar todos os dados coletados e calculados usando interpolação de strings

git

Salvar no Git

add, commit -m "Ficha de Personagem com Cálculos", push

Esta atividade combina tudo o que aprendemos: variáveis, entrada de dados, conversão, operações matemáticas e exibição formatada. Além disso, praticamos o fluxo de trabalho do Git para salvar nosso progresso de forma profissional!

Dicas do Sábio: Quando o Código "Quebra"

FormatException: Geralmente ocorre ao tentar Parse um texto que não é um número válido. Sempre verifique a entrada do usuário!

Nomes de Variáveis Errados: C# é case-sensitive, então vidaAtual é diferente de VidaAtual. Mantenha consistência no padrão de nomes.

Ponto e Vírgula Faltando: Um clássico! Toda instrução em C# termina com ; e a falta dele causa erros.

Divisão por Zero: int x = 10 / 0; causará DivideByZeroException. Sempre verifique denominadores.

Próxima Dungeon: Tomando Decisões no Código!

Bifurcações no Código

Assim como em RPGs, seu código precisa tomar decisões baseadas em condições. As estruturas condicionais permitem que seu programa siga caminhos diferentes.

Estruturas Condicionais

Na próxima aula, vamos aprender sobre if, else e como usar operadores relacionais e lógicos para criar condições complexas em nossos programas.

Exemplos Práticos

Se o HP do jogador for menor que zero, ENTÃO game over! Se o jogador tiver a chave E estiver na porta, ENTÃO abrir! Aplicações reais que usaremos em jogos.

Dominamos como guardar dados e fazer contas básicas! Na próxima aula, daremos mais um passo importante. Não esqueçam de praticar o ciclo do Git com os exercícios de hoje - add, commit, push. É fundamental para o curso e para sua carreira como desenvolvedor!