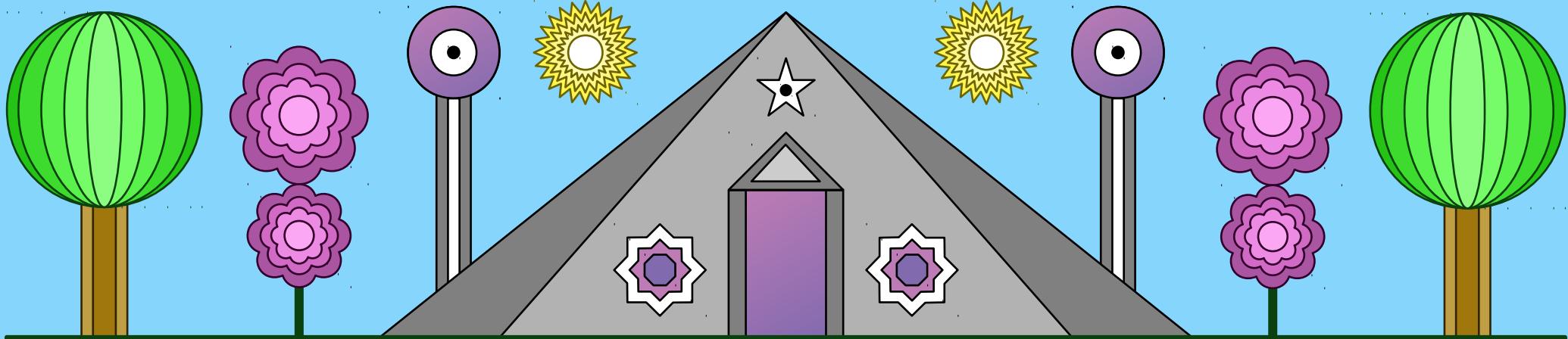


# Desenvolvimento de Games 2D



**Introdução à Programação  
Linguagem JavaScript  
Gabriel Felippe**

# Sumário

- Introdução
- Tecnologias Usadas
- Por que JavaScript Puro, Canvas e HTML?
- Conhecendo o Conteúdo do Curso
- Entendendo o Canvas
- Sistema de Coordenadas do Canvas
- Funcionamento do Canvas
- Ambiente de Desenvolvimento
- Desenhando Objetos no Canvas
- Animações
- Eventos
- Game Loop
- Relógio
- Games Desenvolvidos
- Considerações Finais

# Introdução

Olá viajante, seja muito bem-vindo ao curso introdutório sobre como criar games usando apenas **JavaScript Puro, Canvas e HTML**.

Neste curso, você aprenderá os fundamentos essenciais para criar seus próprios games empolgantes e interativos diretamente no navegador, sem a necessidade de frameworks ou bibliotecas externas.

Vamos explorar desde os conceitos básicos do **Canvas** até técnicas de animação e lógica.

# Tecnologias

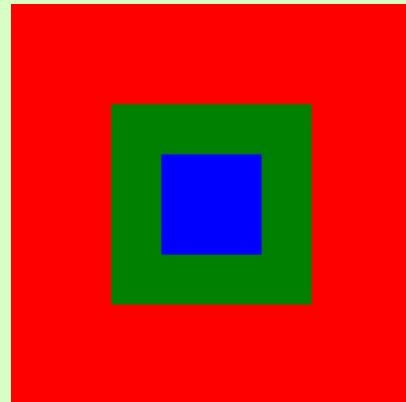
O desenvolvimento de games em JavaScript oferece uma abordagem **flexível e acessível** para criar games multiplataforma diretamente no **navegador**.



# Tecnologias

Com o uso do **Canvas HTML**, podemos desenhar gráficos e animações de forma eficiente, proporcionando uma experiência de jogo imersiva para os usuários.

```
<canvas id="myCanvas" width="200" height="200" />
<script type="text/javascript">
    var canvas = document.getElementById("myCanvas");
    var ctx = canvas.getContext("2d");
    ctx.fillStyle = "red";
    ctx.fillRect(0, 0, 200, 200);
    ctx.fillStyle = "green";
    ctx.fillRect(50, 50, 100, 100);
    ctx.fillStyle = "blue";
    ctx.fillRect(75, 75, 50, 50);
</script>
```



# Por que JavaScript Puro, Canvas e HTML?

**Acessibilidade e Ubiquidade:** JavaScript é uma linguagem de programação amplamente suportada por todos os principais navegadores da web. Isso significa que os games desenvolvidos em JavaScript podem ser facilmente acessados e jogados em uma ampla variedade de dispositivos e plataformas, incluindo computadores desktop, tablets e smartphones.



MOBILE



TABLET



DESKTOP

# Por que JavaScript Puro, Canvas e HTML?

**Flexibilidade e Controle:** Usar JavaScript Puro nos dá total controle sobre o código do game. Não estamos limitados pelas restrições de uma biblioteca ou framework específico, o que nos permite personalizar cada aspecto do nosso game de acordo com nossas necessidades específicas.



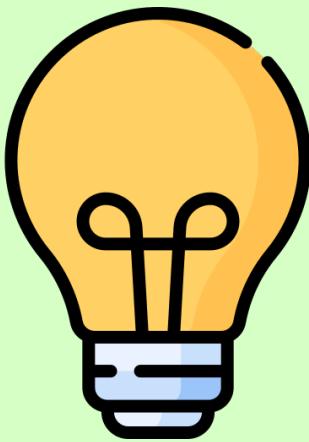
# Por que JavaScript Puro, Canvas e HTML?

**Performance:** O Canvas HTML oferece uma maneira eficiente de renderizar gráficos 2D em tempo real diretamente no navegador. Isso nos permite criar games visualmente impressionantes e altamente responsivos, mesmo em dispositivos com recursos limitados.



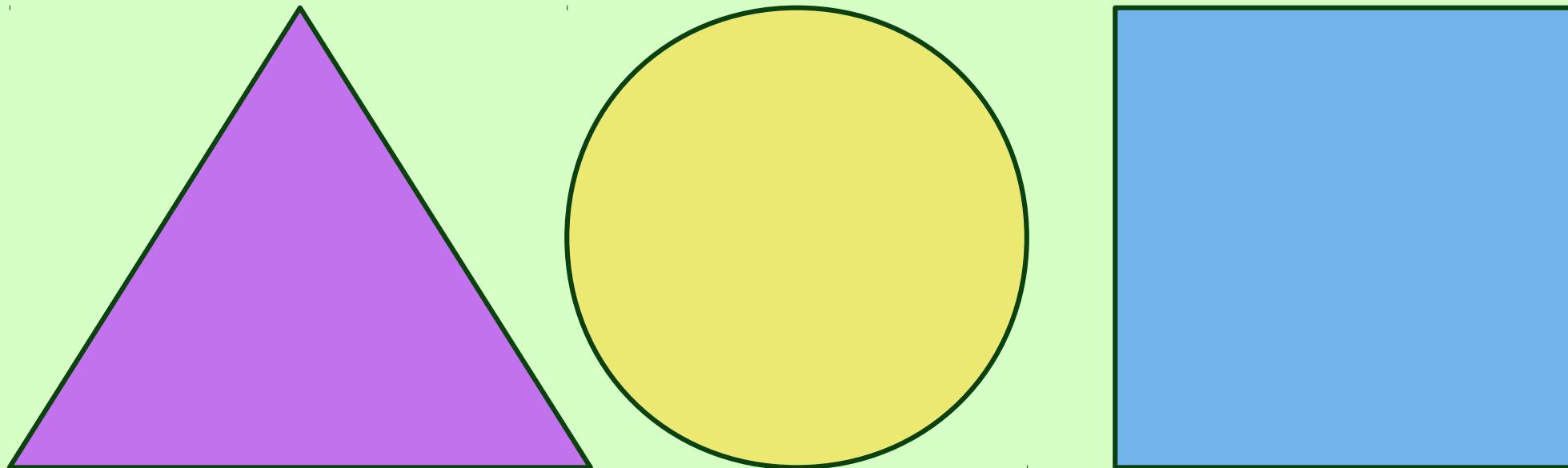
# Por que JavaScript Puro, Canvas e HTML?

**Aprendizado Valioso:** Desenvolver games com JavaScript Puro, Canvas e HTML é uma excelente maneira de aprender os fundamentos da programação e dos gráficos de computador. Você ganhará uma compreensão mais profunda dos conceitos subjacentes enquanto cria games divertidos e interativos.



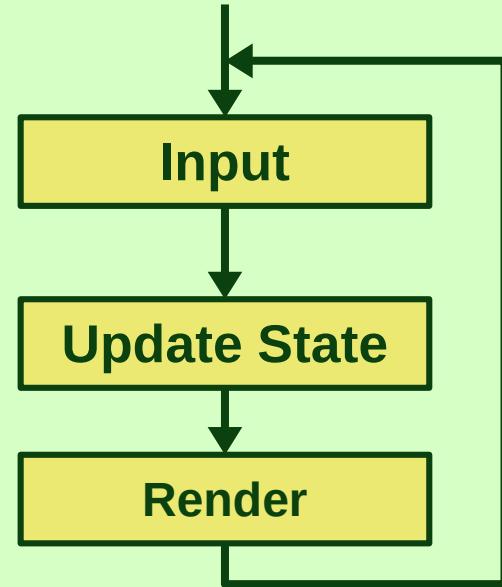
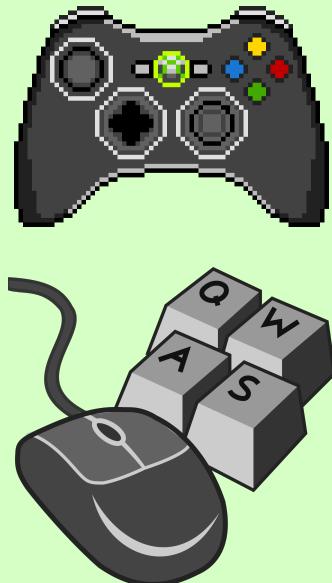
# Conteúdo

**Desenho de objetos no Canvas:** Você aprenderá como desenhar formas básicas, como retângulos e círculos, e como estilizá-los para criar uma experiência visual envolvente.



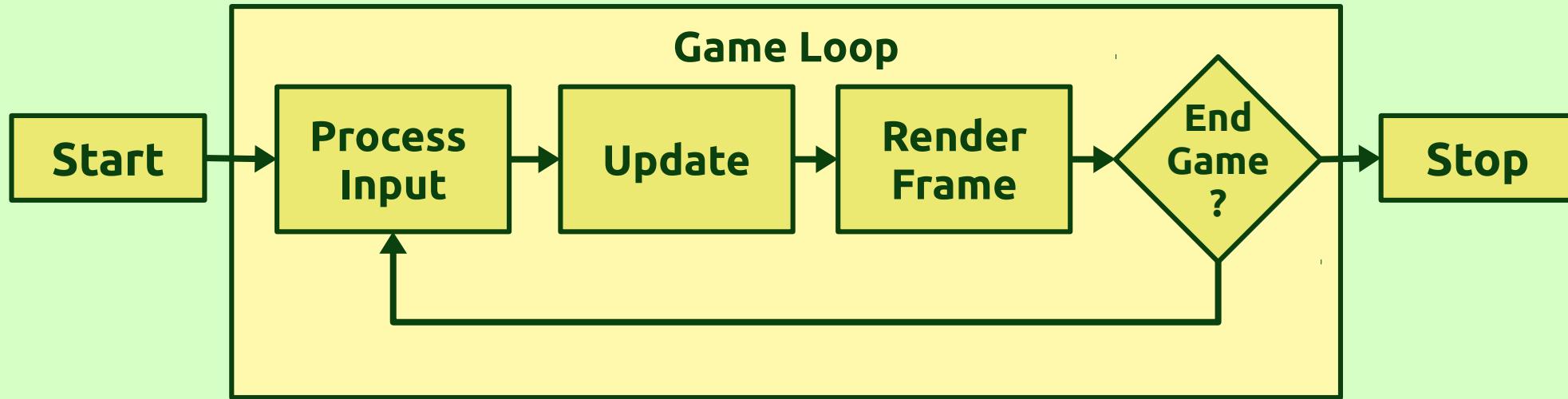
# Conteúdo

**Animação e interação do usuário:** Vamos explorar como animar objetos no Canvas e como responder às entradas do usuário, como teclado e mouse, para criar jogos interativos.



# Conteúdo

**Game Loop e Lógica do Game:** Vamos entender o conceito de Game Loop e como usá-lo para criar uma jogabilidade suave e responsiva, além de implementar a Lógica do Game, incluindo detecção de colisões e gerenciamento de estados.



# Conteúdo

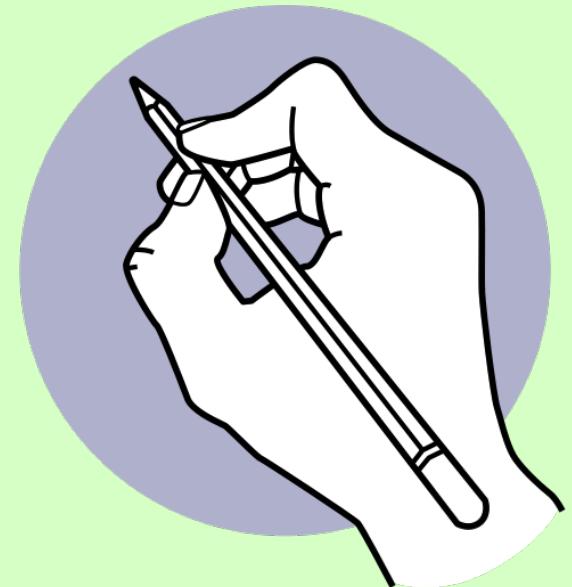
**Recursos avançados:** Por fim, vamos explorar recursos avançados, como implementação de câmeras para seguir o jogador e adição de efeitos especiais para tornar seu jogo ainda mais impressionante.



# Canvas

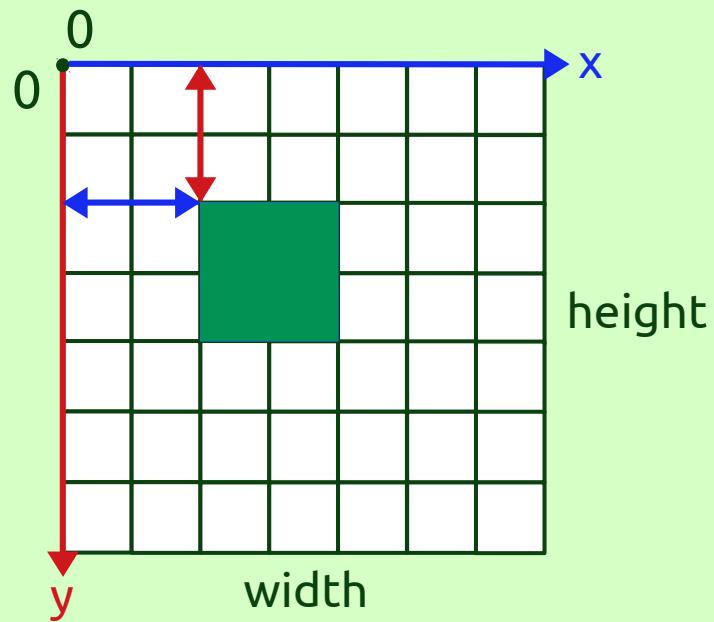
O Canvas é uma **poderosa ferramenta HTML** que nos permite desenhar gráficos, animações e outras visualizações diretamente no navegador da web usando JavaScript.

Ele fornece uma **área de desenho** dinâmica que podemos **manipular** programaticamente para criar uma variedade de **elementos visuais**, como formas geométricas, imagens e textos.



# Sistema de Coordenadas do Canvas

O Canvas possui um **sistema de coordenadas** onde o **ponto  $(0, 0)$**  está no canto superior esquerdo do canvas. As coordenadas aumentam para a direita e para baixo.



# Funcionamento do Canvas

- Criamos um elemento **<canvas>** em nosso HTML, especificando sua largura (**width**) e altura (**height**).
- Usamos JavaScript para acessar o contexto de desenho (**context**) do Canvas.
- Utilizamos os **métodos e propriedades** do contexto para desenhar e manipular elementos no Canvas.

```
<canvas id="meuCanvas" width="350" height="350"></canvas>
<script type="text/javascript" src="game.js"></script>
```

index.html

```
var canvas = document.getElementById("meuCanvas");
var context = canvas.getContext("2d");

context.fillStyle = "red";
context.fillRect(50, 50, 100, 100);
```

game.js

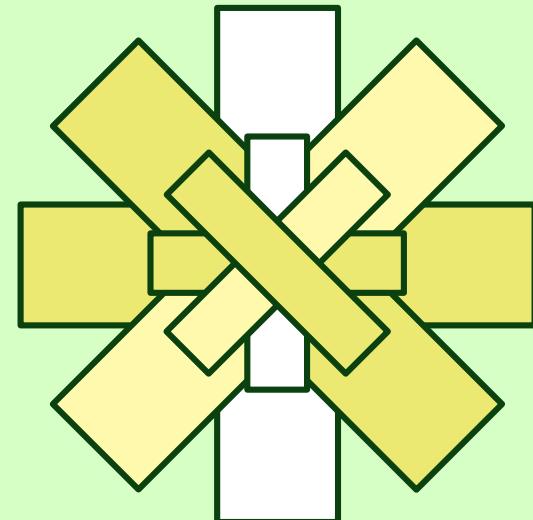
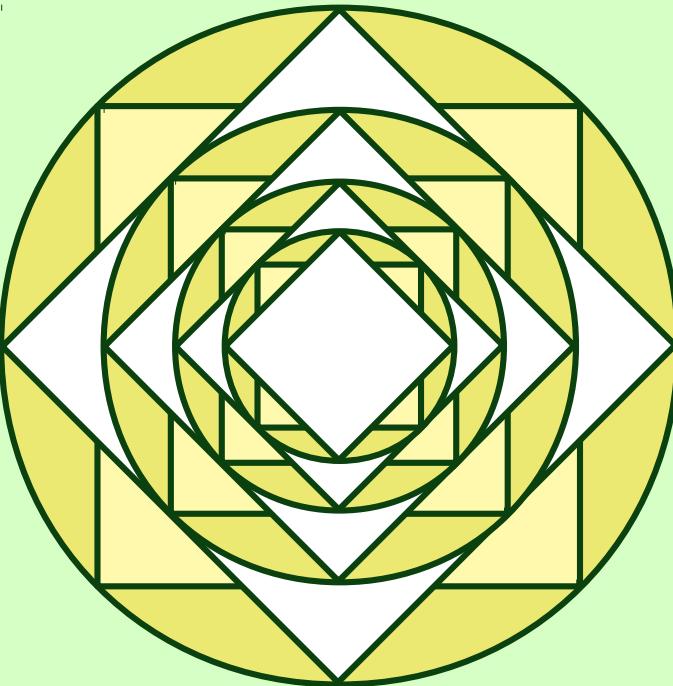
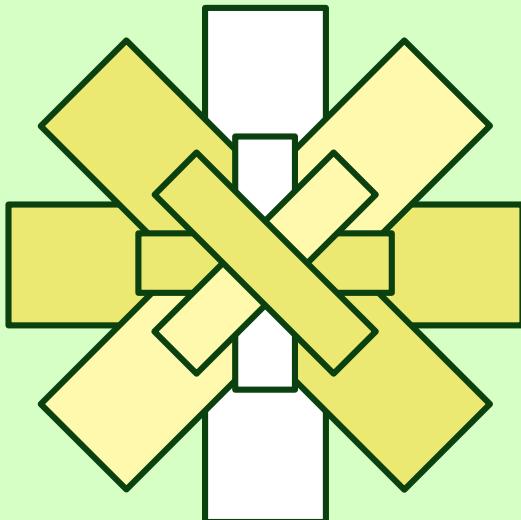
# Ambiente de Desenvolvimento

Para configurar o ambiente de desenvolvimento para criar games com JavaScript e Canvas, você só precisa de um **editor de código** e um **navegador da web**. Aqui está um passo a passo básico:

- 1. Escolha um editor de código:** Você pode usar qualquer editor de código que preferir. Alguns dos populares são Visual Studio Code e Sublime Text.
- 2. Crie um arquivo HTML:** Crie um arquivo HTML para seu jogo e adicione um elemento `<canvas>` para começar a desenhar. Você também pode incluir seu arquivo JavaScript neste HTML.
- 3. Comece a programar:** Escreva o código JavaScript para interagir com o Canvas e criar elementos visuais.
- 4. Teste no navegador:** Abra seu arquivo HTML em um navegador da web para visualizar seu jogo. Você pode usar o Chrome, Firefox, ou qualquer outro navegador moderno.

# Desenhando Objetos no Canvas

O Canvas HTML oferece várias maneiras de desenhar formas básicas, como retângulos, círculos e linhas.



# Retângulos

Para desenhar um retângulo no Canvas, podemos usar o método **fillRect()** ou **strokeRect()**.

O **fillRect()** preenche o retângulo com uma cor sólida, enquanto o **strokeRect()** apenas desenha a borda do retângulo.

**exemplo.js**

```
// Define a cor de preenchimento  
ctx.fillStyle = "blue";  
  
// Desenha um retângulo preenchido  
// Nas coordenadas (50, 50) com largura 100 e altura 75  
ctx.fillRect(50, 50, 100, 75);
```

# Círculos

Para desenhar um círculo, usamos o método **arc()** do contexto de desenho. Este método cria um arco de círculo.

## exemplo.js

```
// Inicia o caminho do desenho
ctx.beginPath();
// Desenha um círculo com raio 50 e centro nas coordenadas (150, 150)
ctx.arc(150, 150, 50, 0, Math.PI * 2);
// Define a cor de preenchimento
ctx.fillStyle = "red";
// Preenche o círculo
ctx.fill();
```

# Linhas

Para desenhar linhas, usamos os métodos **moveTo()** para definir o ponto inicial e **lineTo()** para definir o ponto final.

## exemplo.js

```
// Inicia o caminho do desenho
ctx.beginPath();
// Move para as coordenadas (200, 200)
ctx.moveTo(200, 200);
// Desenha uma linha até as coordenadas (300, 200)
ctx.lineTo(300, 200);
// Define a cor da linha
ctx.strokeStyle = "green";
// Desenha a linha
ctx.stroke();
```

# Gradientes

Uma das maneiras de estilizar objetos no Canvas é definindo cores de preenchimento.

Podemos usar **cores sólidas** ou **gradientes** para preencher formas.

```
var gradient = ctx.createLinearGradient(50, 50, 150, 150);
gradient.addColorStop(0, "red");
gradient.addColorStop(1, "blue");
ctx.fillStyle = gradient;
ctx.fillRect(50, 50, 100, 100);
```

exemplo.js

# Animação

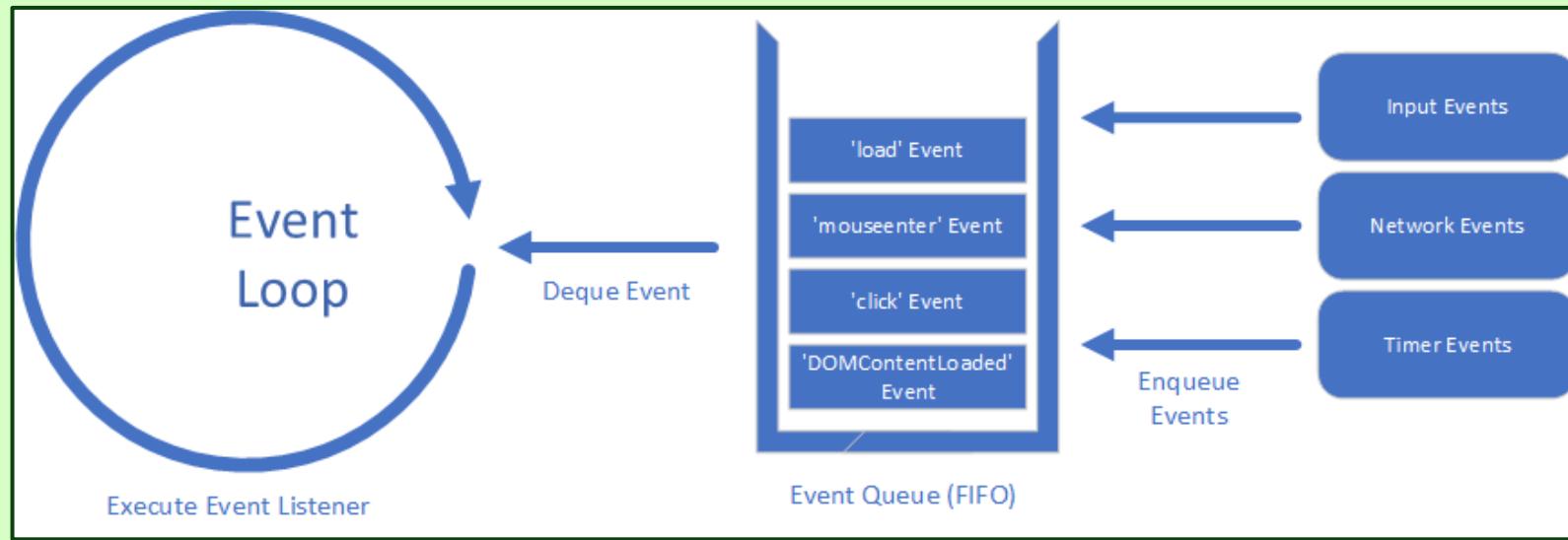
**requestAnimationFrame** é um método que executa uma função de renderização na próxima atualização de animação do navegador. Ele é frequentemente usado para criar animações suaves no Canvas.

```
var x = 0;
var dx = 1;
function animateBox() {
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    ctx.fillStyle = "red";
    ctx.fillRect(x, 50, 50, 50);
    x += dx;
    if (x <= 0 || x >= canvas.width - 50) {
        dx = -dx;
    }
    requestAnimationFrame(animateBox);
}
animateBox();
```

exemplo.js

# Eventos

Para lidar com eventos de **teclado** e **mouse**, podemos usar os eventos fornecidos pelo navegador, como **keydown**, **keyup**, **mousemove**, **mousedown**, **mouseup**, entre outros. Em seguida, podemos adicionar ouvintes de eventos para responder a essas interações do usuário.



# Exemplo de Eventos

A seguir temos um exemplo de como lidar com eventos de teclado (movendo um objeto com as teclas direcionais).

1. Começamos definindo o nosso objeto (**player**).

```
var player = {  
    x: 50,  
    y: 50,  
    width: 50,  
    height: 50,  
    speed: 5  
};
```

# Exemplo de Eventos

2. Podemos atualizar as coordenadas de um objeto (ou player) em resposta às teclas direcionais pressionadas pelo usuário. Isso nos permite mover objetos na tela de acordo com a entrada do usuário.

```
document.addEventListener("keydown", function(event) {  
    if (event.key === "ArrowRight" && player.x + player.width + player.speed <= canvas.width) {  
        player.x += player.speed;  
    } else if (event.key === "ArrowLeft" && player.x - player.speed >= 0) {  
        player.x -= player.speed;  
    } else if (event.key === "ArrowDown" && player.y + player.height + player.speed <= canvas.height) {  
        player.y += player.speed;  
    } else if (event.key === "ArrowUp" && player.y - player.speed >= 0) {  
        player.y -= player.speed;  
    }  
});
```

# Exemplo de Eventos

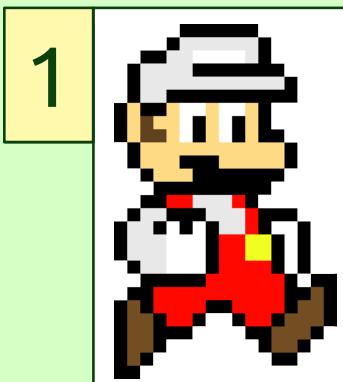
3. Por fim, podemos desenhar o objeto (player) na tela.

```
function drawPlayer() {  
    ctx.clearRect(0, 0, canvas.width, canvas.height);  
    ctx.fillStyle = "blue";  
    ctx.fillRect(player.x, player.y, player.width, player.height);  
    requestAnimationFrame(drawPlayer);  
}  
  
drawPlayer();
```

# Animação de Sprites

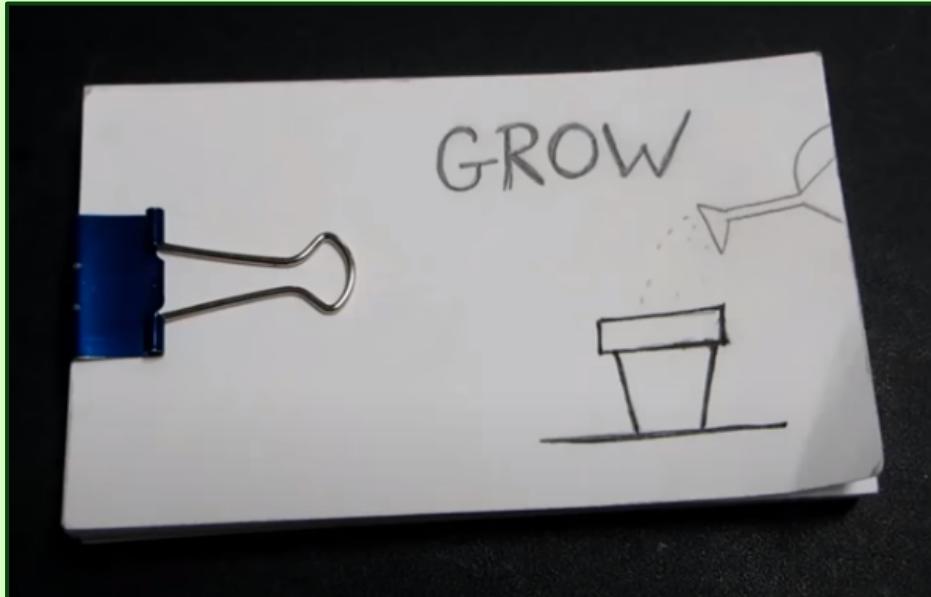
No contexto de animação de imagens usando JavaScript e Canvas, um "**frame**" se refere a uma única imagem que compõe uma sequência de animação.

Por exemplo, se você tem uma animação de um personagem correndo, cada pose diferente do personagem durante a corrida seria um frame diferente.



# Animação de Sprites

A animação é criada exibindo uma série de frames em rápida sucessão para criar a ilusão de movimento. Isso é conhecido como animação por quadros, ou "sprite animation".



# Animação de Sprites

Para implementar isso no Canvas, você geralmente segue os seguintes passos:

**1. Preparação dos frames:** Você precisa ter uma série de imagens (frames) que representam os diferentes estágios da animação. Por exemplo, se você está animando um personagem correndo, você teria diferentes imagens para cada passo da corrida.

**2. Desenho dos frames no Canvas:** Você desenha cada frame da animação no Canvas em uma posição específica. Isso geralmente é feito dentro de uma função de desenho que é chamada repetidamente em intervalos regulares.

**3. Atualização dos frames:** A cada vez que a função de desenho é chamada, você avança para o próximo frame na sequência de animação. Isso cria a ilusão de movimento quando os frames são exibidos em rápida sucessão.

**4. Limpeza do Canvas:** Antes de desenhar o próximo frame, geralmente é uma boa prática limpar o Canvas para remover o frame anterior. Isso evita que os frames antigos fiquem visíveis na animação.

# Animação de Sprites

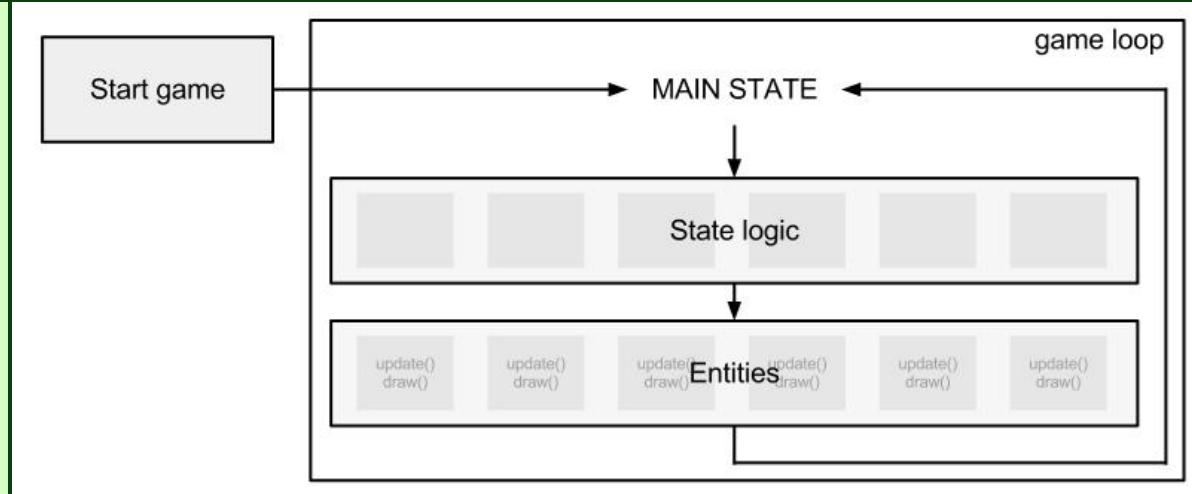
No exemplo a seguir, esses conceitos são aplicados. A cada intervalo de tempo definido pelo **setInterval**, a função **draw** é chamada para limpar o Canvas, carregar o próximo frame da sequência de animação e desenhá-lo no Canvas.

```
const marioFrames = ['mario1.png', 'mario2.png', 'mario3.png'];
let currentFrame = 0;
function draw() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  const img = new Image();
  img.onload = function() {
    ctx.drawImage(img, 50, 50, 146, 242);
  };
  img.src = marioFrames[currentFrame];
  currentFrame = (currentFrame + 1) % marioFrames.length;
}
setInterval(draw, 100);
```

# Game Loop

O game loop é um conceito fundamental em muitos jogos e aplicações interativas. Ele é responsável por controlar o fluxo de um jogo, atualizando a lógica do jogo e renderizando os gráficos em uma base contínua.

O objetivo é garantir que o jogo seja executado suavemente e de forma consistente, independentemente da velocidade do hardware ou de outras condições.



# Game Loop

O game loop típico tem três etapas principais:

**1. Processamento de Entrada:** Nesta etapa, ele captura e processa todas as entradas do usuário, como pressionamentos de teclas, cliques do mouse, toques na tela, etc.

**2. Atualização da Lógica do Jogo:** Aqui, a lógica do jogo é atualizada com base nas entradas do usuário e em qualquer outra condição que afete o estado do jogo. Por exemplo, movimento de personagens, detecção de colisões, cálculo da física do jogo, etc.

**3. Renderização Gráfica:** Na última etapa, o game loop renderiza os gráficos atualizados na tela. Isso inclui desenhar personagens, objetos, cenários e quaisquer outros elementos visuais do jogo.

Ele geralmente é executado continuamente, garantindo que o jogo seja atualizado e renderizado várias vezes por segundo para proporcionar uma experiência de jogo suave.

# Game Loop

Aqui está um exemplo simples de um Game Loop usando JavaScript:

```
function gameLoop() {  
    // Etapa 1: Processamento de Entrada  
    // Aqui você processaria qualquer entrada do usuário,  
    // como teclas pressionadas ou cliques do mouse  
    // Etapa 2: Atualização da Lógica do Jogo  
    // Aqui você atualizaria a lógica do jogo,  
    // com base nas entradas do usuário e em outras condições  
    // Etapa 3: Renderização Gráfica  
    // Aqui você renderizaria os gráficos atualizados na tela  
    // Repetir o loop chamando a função novamente  
    requestAnimationFrame(gameLoop);  
}  
  
// Iniciar o loop de jogo  
gameLoop();
```

Neste exemplo, **requestAnimationFrame** é usado para chamar a função `gameLoop` repetidamente. Esta é uma maneira eficiente de executar um game loop no navegador, pois aproveita o ciclo de atualização do navegador para sincronizar as atualizações de renderização com a taxa de atualização do monitor, proporcionando uma experiência de jogo suave e eficiente.

# Relógio

Um "clock" em um jogo é frequentemente usado para acompanhar o tempo decorrido desde o início do jogo, ou para medir intervalos de tempo para determinadas ações.



# Relógio

A seguir temos um exemplo de como implementar um clock em JavaScript:

```
let startTime = Date.now(); // Captura o tempo atual em milissegundos no início do jogo
function gameLoop() {
    // Calcular o tempo decorrido desde o início do jogo
    let currentTime = Date.now();
    let elapsedTime = currentTime - startTime;
    // Convertendo milissegundos para segundos
    let seconds = Math.floor(elapsedTime / 1000);
    // Mostrar o tempo decorrido no console
    console.log("Tempo decorrido: " + seconds + " segundos");
    // Repetir o loop chamando a função novamente
    requestAnimationFrame(gameLoop);
}
// Iniciar o loop de jogo
gameLoop();
```

Neste exemplo, capturamos o tempo atual em milissegundos no início do jogo usando **Date.now()** e armazenamos em **startTime**. Dentro do loop de jogo, calculamos o tempo decorrido desde o início do jogo subtraindo **startTime** do tempo atual (**currentTime**). Em seguida, convertemos o tempo decorrido de milissegundos para segundos e o exibimos no console.

# Games Desenvolvidos

Agora é hora de conhecermos os games que serão desenvolvidos no curso.



# A Jornada de Ismael

Nesta jornada aprenderemos a manipular imagens e trabalhar com animações.



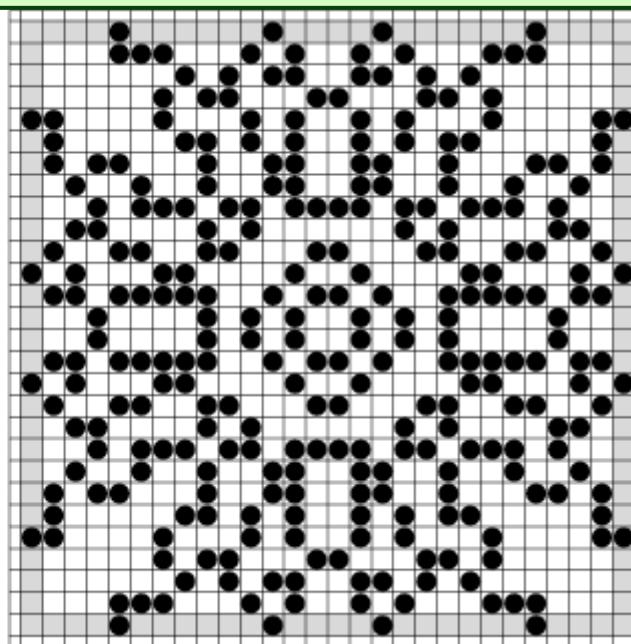
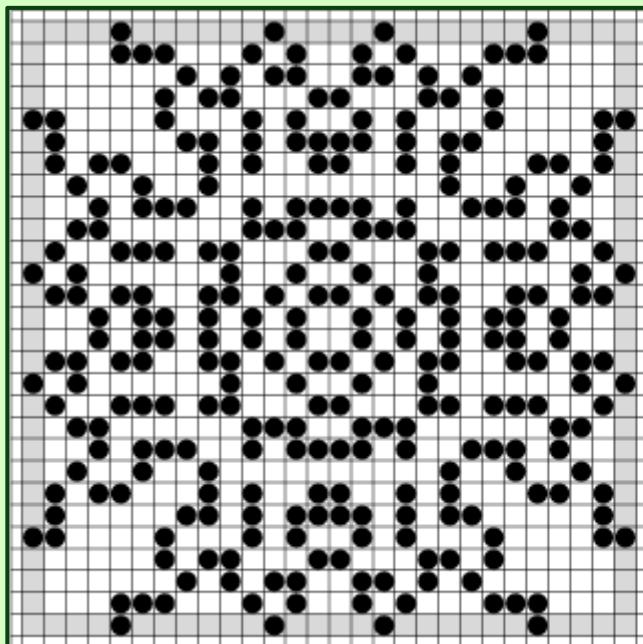
# Alpha Wings

Neste projeto vamos explorar o conceito de eventos e tempo.



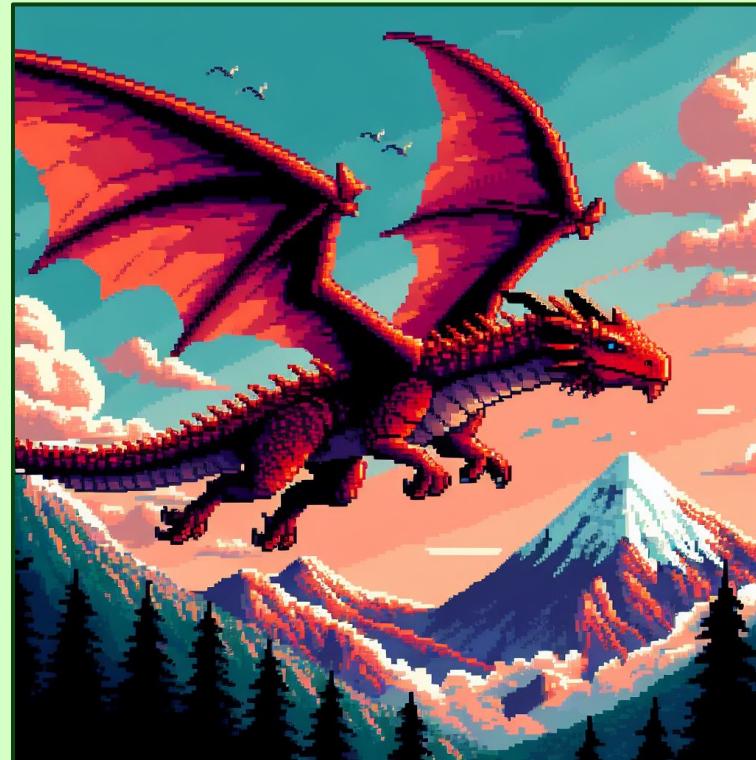
# Conway's Game of Life

Interessante experiência no mundo da matemática e lógica.



# Elder Dragon

Aqui iremos aperfeiçoar nossa habilidade com animações e tempo!



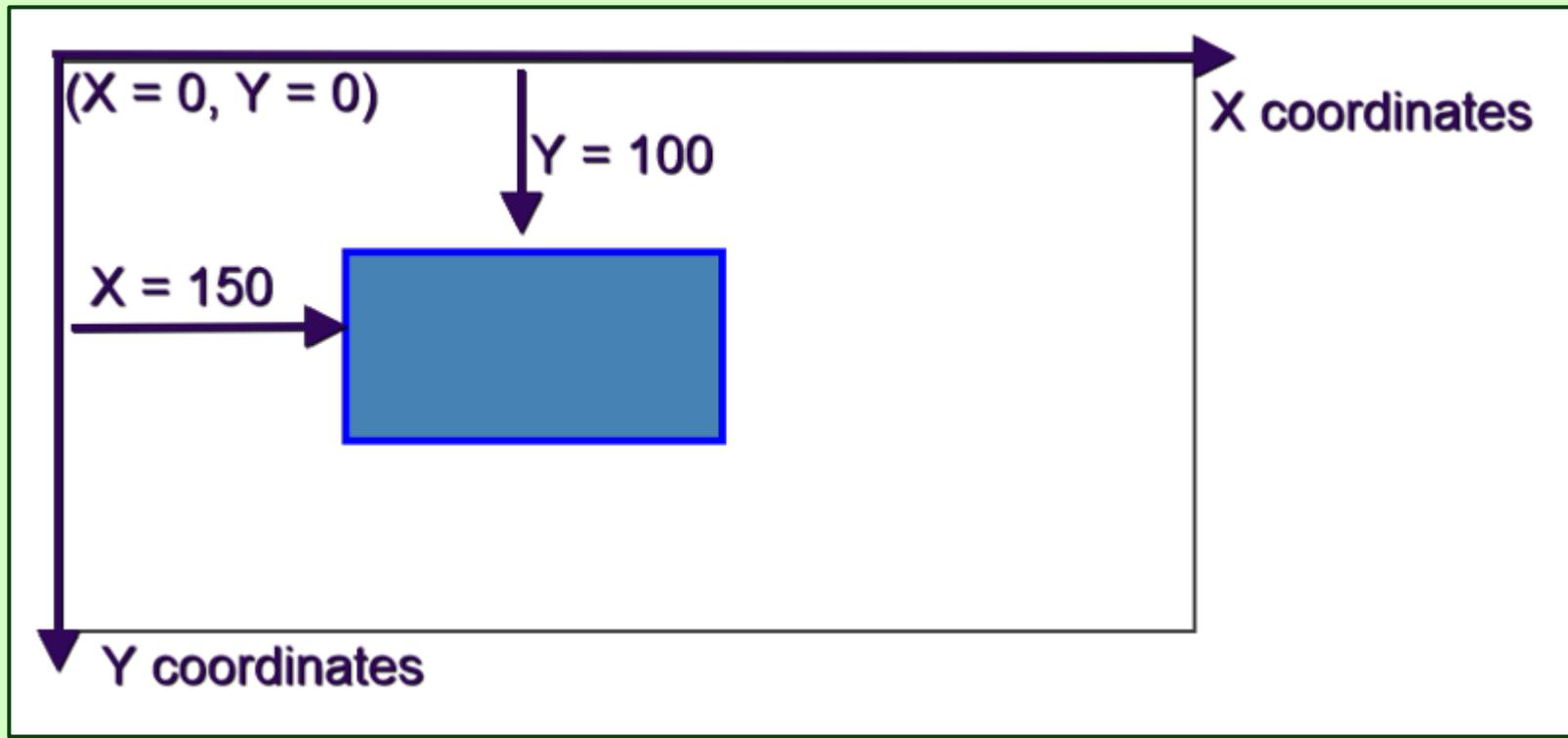
# Shadow Templar

Nosso primeiro Platformer 2D usando um sistema de grid, colisões e gravidade.



# Sistema de Coordenadas

Nesta aventura compreenderemos definitivamente o sistema de coordenadas.



# Snake

Desenvolveremos este clássico dos games primordiais.



# Considerações Finais

**Prática é Fundamental:** A prática é essencial para melhorar suas habilidades de programação. Quanto mais você pratica escrever código, mais confortável e habilidoso você se torna.

**Entenda os Fundamentos:** Antes de tentar projetos complexos, é importante entender os fundamentos da linguagem de programação que você está usando. Isso inclui conceitos como variáveis, estruturas de controle, funções e objetos.

**Mantenha o Código Limpo e Organizado:** Escrever código limpo e bem organizado é crucial para a legibilidade e manutenção do código. Use nomes descritivos de variáveis e funções, evite repetições desnecessárias e siga as convenções de estilo da linguagem.

**Teste e Depure o Código:** Sempre teste seu código e depure quaisquer erros ou bugs. A depuração é uma habilidade valiosa que pode ajudá-lo a identificar e corrigir problemas em seu código.

# Considerações Finais

**Aprenda com os Outros:** Não tenha medo de procurar exemplos de código, tutoriais e documentação online. A comunidade de programadores é vasta e há uma abundância de recursos disponíveis para ajudá-lo a aprender e aprimorar suas habilidades.

**Seja Paciente e Persistente:** A programação pode ser desafiadora e frustrante às vezes, mas é importante permanecer paciente e persistente. Continue praticando, aprendendo com seus erros e celebrando seus sucessos.



# Muito Obrigado!

"A simplicidade é a sofisticação final."  
**Leonardo da Vinci**

