
WORK in
PROGRESS

Laboratório de Programação para Dispositivos Móveis e Sem Fio

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Profª. Drª.
Michelle Larissa Luciano Carvalho

INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA
Baiano

Análise e Desenvolvimento de Sistemas

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

- **Fundamentos:** Canvas e noções de orientação a objetos em JavaScript
- **Loop de animação:** controlar animação e gerenciar sprites
- **Interação com o jogador:** captura de eventos do teclado
- **Folhas de sprites (sprite sheets):** animar o jogo utilizando imagens que contém todos os quadros de animação
- **Detecção de colisão:** detectar quando elementos se cruzam e executar ações (herói versus inimigo)
- **Incorporar animações, sons, pausa e vidas extras ao jogo**

Michelle Karissa Luciano Carvalho

INSTITUTO FEDERAL
Baiano
Campus Catu

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

- W3Schools: http://www.w3schools.com/tags/ref_canvas.asp
- Core HTML5 Canvas: http://corehtml5canvas.com
- HTML5 2D game development: Introducing Snail Bait: http://goo.gl/Ojvi9T - URL encurtada
- Comando do teclado: https://keycode.info/
- Ambiente 3D: http://www.tidbits.com.br/ambientes-3d-usando-html5-e-javascript-com-canvas
- OpenGL: https://www.khronos.org/webgl/

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Fundamentos

- **Canvas** é uma área retangular em uma página web onde podemos criar desenhos programaticamente, usando *JavaScript* (linguagem de programação normal das páginas *HTML*).
  - Com essa tecnologia, podemos criar trabalhos artísticos, animações e jogos.

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Fundamentos

- Para criar **Canvas** em uma página *HTML*, utilizamos a tag <canvas>.Os atributos width e height informam a *largura* e *altura*, respectivamente, da área de desenho.
- É importante informar um id para podermos trabalhar com ele no código *JavaScript*.

```
<canvas id="nome_canvas" width="largura" height="altura">
</canvas>
```

Michelle Karissa Luciano Carvalho

➔ Entre as tags de abertura e fechamento, podemos colocar alguma mensagem indicando que o browser não suporta essa tecnologia. Caso o browser a suporte, esse conteúdo é ignorado:

```
<canvas id="meu_canvas" width="300" height="300">
   Seu navegador não suporta o Canvas do HTML5. <br>
   Procure atualizá-lo.
</canvas>
```

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Fundamentos

Renderização

- Os atributos width e height da tag <canvas> são **obrigatórios**, pois são os valores usados na geração imagem.
- O **Canvas** pode receber dimensões diferentes via *CSS*, no entanto seu processamento, sempre será feito usando as dimensões usadas na tag.
  - Se as dimensões no *CSS* forem diferentes, o browser amplia ou reduz a imagem gerada para deixá-la de acordo com a folha de estilo.
  - Dado um **Canvas** com dimensões **100x100** pixels:

```
<canvas id="meu_canvas" width="100" height="100"></canvas>
```

  - A seguinte formatação *CSS* fará a imagem ser ampliada:

```
#meu_canvas {
    width: 200px;
    height: 200px;
}
```

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Contexto gráfico

- Para desenhar o **Canvas** , é preciso executar um **script** após ele ter sido carregado.
  - Nesse **script**, obteremos o **contexto gráfico**, que é o objeto que realiza de fato as tarefas de desenho no Canvas.
- Uma maneira é criar a tag <script> após a tag <canvas>:

```
<!DOCTYPE html>
<html>

<head>
   <title>Processando o Canvas após a tag</title>
</head>

<body>
   <canvas id="meu_canvas" width="200" height="200"></canvas>
   <script>
      // Aqui obteremos o contexto gráfico e trabalharemos com o
      // Canvas
   </script>
</body>

</html>
```

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Contexto gráfico

➔ Também é muito comum desenharmos em *eventos* que ocorrem após a página ter sido carregada.

- Isto é útil caso queiramos colocar os **scripts** na seção <head> do documento *HTML*:

```
<!DOCTYPE html>
<html>

<head>
   <title>Processando o Canvas na seção HEAD</title>
   <script>
      window.onload = function() {
         // Aqui trabalharemos com o Canvas
      }
   </script>
</head>

<body>
   <canvas id="meu_canvas" width="200" height="200"></canvas>
</body>

</html>
```

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Contexto gráfico

➔ No código, nós referenciamos o **Canvas** e obtemos o **contexto gráfico**.
- O **Canvas** é referenciado como qualquer elemento em uma página;
- O **contexto** é obtido pelo método getContext do **Canvas**. Como parâmetro passamos uma *string* identificando o **contexto** desejado (ex: 2d).

```
<canvas id="meu_canvas" width="200" height="200"></canvas>
<script>
   // Referenciando o Canvas
   var canvas = document.getElementById('meu_canvas');

   // Obtendo o contexto gráfico
   var context = canvas.getContext('2d');
</script>
```

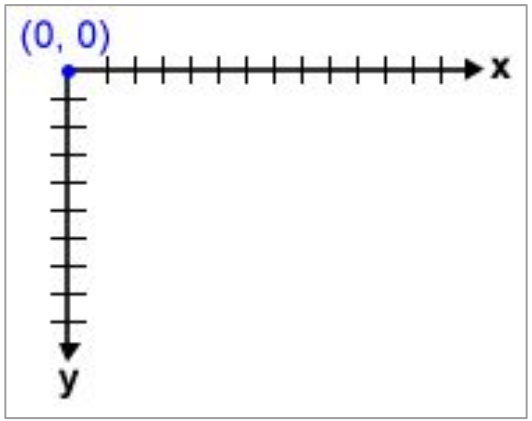
Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Sistema de coordenadas do Canvas

Eixo das abscissas e coordenadas

➔ Para posicionarmos os desenhos no **Canvas**, pensamos nele com um enorme conjunto de pontos.

- Cada ponto possui uma posição horizontal (x) e uma vertical (y).
- O ponto (0,0) (lê-se: zero em x e zero em y) corresponde ao canto superior esquerdo do **Canvas**:



Michelle Karissa Luciano Carvalho

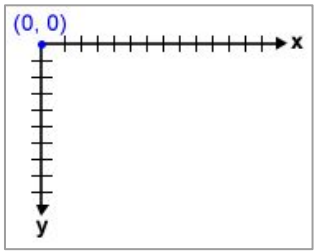
Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Começando desenhar (Exemplos de métodos)

Eixo das abscissas e coordenadas

➔ Uma vez obtido o **contexto gráfico**, podemos configurar várias propriedades e chamar nele os métodos de desenho.

- Por exemplo, para desenhar retângulos, podemos usar os métodos:
  - fillRect(x, y, largura, altura): pinta completamente uma área retangular.
  - strokeRect(x, y, largura, altura): desenha um contorno do retângulo.
- Os valores de x e y corresponderão à posição do canto superior esquerdo do retângulo.
  - A partir daí, o retângulo vai para a direita (*largura*) e para baixo (*altura*).



Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Começando desenhar (Exemplos de métodos)

➔ Exemplo de código para desenhar um retângulo no **Canvas:**

```
<!-- arquivo: retangulos-1.html -->
<!-- este código vai dentro do body -->
<canvas id="meu_canvas" width="200" height="200"></canvas>
<script>
   // Canvas e contexto
   var canvas = document.getElementById('meu_canvas');
   var context = canvas.getContext('2d');

   // Desenhar um retângulo
   context.fillRect(50, 50, 100, 100);
</script>
```

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Começando desenhar (Exemplos de métodos)

- O resultado será um simples retângulo preto (desenhado com fillRect).
  - Seu canto superior esquerdo localiza-se no ponto (50,50) e ele possui **100px** de *largura* por **100px** de *altura*:

  - Se trocarmos fillRect por strokeRect, veremos apenas o contorno:

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Começando desenhar (Exemplos de métodos)

➔ Podemos configurar algumas propriedades do **contexto**, de forma a escolher as cores e espessuras:

- fillStyle: cor do preenchimento
- strokeStyle: cor da linha
- lineWidth: espessura da linha em pixels

```
<!-- arquivo: retangulos-2.html -->
<canvas id="meu_canvas" width="200" height="200"></canvas>
<script>
   // Canvas e contexto
   var canvas = document.getElementById('meu_canvas');
   var context = canvas.getContext('2d');

   // Preenchimento vermelho
   context.fillStyle = 'red';
   context.fillRect(50, 50, 100, 100);

   // Contorno azul, com espessura de 3px
   context.lineWidth = 3;
   context.strokeStyle = 'blue';
   context.strokeRect(50, 50, 100, 100);
</script>
```

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

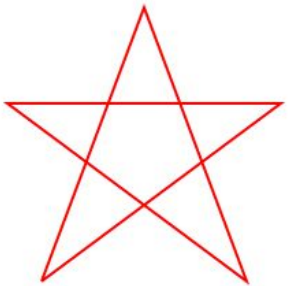
Começando desenhar (Exemplos de métodos)

➔ Podemos desenhar uma estrela usando os seguintes comandos:

- moveTo(x, y): posiciona a caneta virtual em um determinado ponto.
- lineTo(x, y): traça uma linha do ponto atual até o ponto indicado.

```
// Desenhar uma estrela
context.moveTo(75, 250);  // Ponto inicial
context.lineTo(150, 50);
context.lineTo(225, 250);
context.lineTo(50, 120);
context.lineTo(250, 120);
context.lineTo(75, 250);

// Configurar a linha
context.lineWidth = 2;
context.strokeStyle = 'red';
```



Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Desenhar imagens

➔ Para desenhar imagens pré-elaboradas em um **Canvas**:

- Primeiro temos de carregar o arquivo de imagem.
  - O objeto `Image` do *JavaScript* é equivalente a um elemento `<img>` na página, porém somente em memória.
- Após criá-lo, apontamos seu atributo `src` para o arquivo desejado:

```
// Exemplo teórico

// Carregando uma imagem programaticamente
var imagem = new Image();
imagem.src = 'minha-imagem.png';  // gif, jpg...

// Obtendo de uma imagem na página
var imagem = document.getElementById('id_da_tag_img');
```

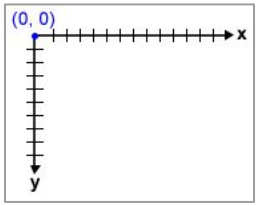
Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Desenhar imagens

➔ Convém aguardar a imagem ser carregada antes de desenhá-la.

- O objeto Image possui o *evento* onload, que será disparado automaticamente pelo browser quando carregamento estiver completo:

```
// Exemplo teórico
imagem.onload = function() {
    // Aqui trabalhamos com a imagem
}
```



- Estando carregada, a imagem pode ser desenhada através do método drawImage do context. Esse método pode ser chamado de duas formas:
  - drawImage(imagem, x, y, largura, altura): desenha a imagem inteira, na posição e tamanho especificados.
  - drawImage(imagem, xOrigem, yOrigem, larguraOrigem, alturaOrigem, xDestino, yDestino, larguraDestino, alturaDestino): desenha parte da imagem.

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Desenhar imagens

➔ Vamos ver um exemplo de carregamento da imagem ovni.png, que representa o inimigo que a nave irá enfrentar no jogo.

- A imagem possui **64 pixels** de *largura* por **32 pixels** de *altura* (64x32). Vamos criar uma página no **Canvas** e carregá-la:

```
<!-- arquivo: imagens-1.html -->
<canvas id="meu_canvas" width="500" height="100"></canvas>
<script>
   // Canvas e contexto
   var canvas = document.getElementById('meu_canvas');
   var context = canvas.getContext('2d');

   // Carregar a imagem
   var imagem = new Image();
   imagem.src = 'img/ovni.png';
   imagem.onload = function() {
      // Aqui usaremos drawImage
   }
</script>
```

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Desenhar imagens

➔ No evento onload, fazemos um *loop* para desenhar cinco OVNIS, um ao lado do outro. A variável ***x*** indica a posição de cada desenho:

```
imagem.onload = function() {
   // Começar na posição 20
   var x = 20;

   // Desenhar as imagens
   for (var i = 1; i <= 5; i++) {
      context.drawImage(imagem, x, 20, 64, 32);
      x += 70;
   }
}
```

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Desenhar imagens

➔ Se quiséssemos desenhar as imagens ampliadas ou reduzidas, bastaria modificar a *largura* e a *altura*:

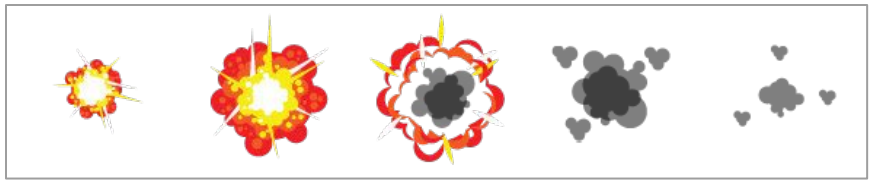
```
// Reduzindo para metade do tamanho
context.drawImage(imagem, x, 20, 32, 16);

// Ampliando para o dobro do tamanho
context.drawImage(imagem, x, 20, 128, 64);
```

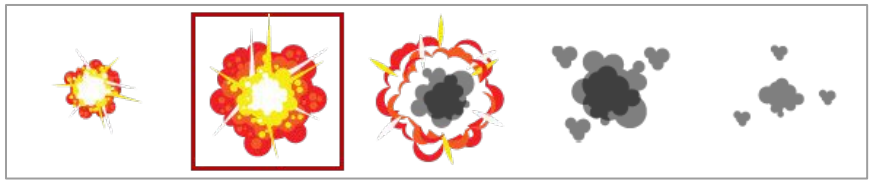
Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Desenhar imagens (Animação e Sprites)

- Vamos ver um exemplo que recebe oito valores.
  - É uma simulação de explosão utilizando a imagem explosao.png.
    - Esta imagem contém uma sequência de **animação** (*spritesheet*), da qual desenhamos uma parte por vez:



  - A técnica de *clipping* consiste em selecionar uma área da imagem original para ser desenhada:



Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Desenhar imagens (Animação e Sprites)

- Os quatro primeiros valores ao drawImage indicam o retângulo da área enquadrada;
- Os outros quatro valores representam a *posição* e o *tamanho* do desenho no **Canvas:**

```
<!-- arquivo: imagens-2.html -->
<canvas id="meu_canvas" width="300" height="300"></canvas>
<script>
   // Canvas e contexto
   var canvas = document.getElementById('meu_canvas');
   var context = canvas.getContext('2d');

   // Carregar a imagem
   var imagem = new Image();
   imagem.src = 'img/explosao.png';
   imagem.onload = function() {
      context.drawImage(
         imagem,
         80, 10, 60, 65,  // Área de recorte (clipping)
         20, 20, 60, 65   // Desenho no Canvas
      );
   }
</script>
```

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Loop de animação

➔ **Sprites** são os objetos trabalhados dentro de um *loop* de **animação**.
- É uma imagem com o fundo transparente, que pode ser **animada**.
- Dessa forma, podem ser inseridos sobre o fundo principal do cenário.

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Loop de animação

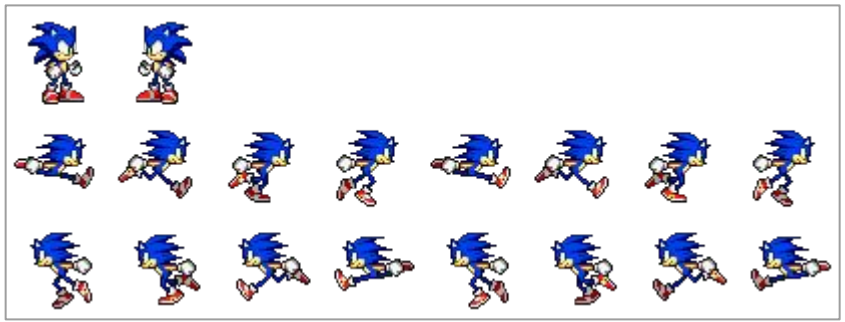
- **Sprites** são os objetos trabalhados dentro de um *loop* de **animação**.
  - É uma imagem com o fundo transparente, que pode ser **animada**.
  - Dessa forma, podem ser inseridos sobre o fundo principal do cenário.

  - **Sprites** são criados e destruídos (na memória do computador) a todo momento.
    - Em *JavaScript* anulamos as variáveis que se referem ao sprite e o deixamos livre para *garbage collector* apagá-lo da memória automaticamente.

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Folhas de sprites (SpriteSheets)

- **Sprite** é cada elemento controlado pelo *loop* de **animação** (herói, um inimigo, um bloco, ou plataforma, etc).
- Uma folha de **sprites** é uma imagem contendo várias partes de uma **animação.**
  - Essas partes são alternadas constantemente para produzir o movimento de um ou mais **sprites.**



Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Folhas de sprites (SpriteSheets)

- É uma boa prática carregar imagens maiores contendo uma porção de outras menores, em vez de cada pequena imagem em seu próprio arquivo.
- Programar **animações** com **spritesheets** pode ser tão simples ou tão trabalhoso quanto você desejar ou precisar, dependendo da abordagem utilizada.
  - No livro Core HTML5 Canvas, de David Geary, as posições de cada figurinha (x, y, largura e altura) dentro da spritesheet são chumbadas em grandes arrays:

```
var sprite = [ [0, 71, 90, 80],
               [40, 71, 80, 85], ... ]; // Continua...
```

  - Essa abordagem bastante econômica em termos do espaço ocupado pelas imagens, mas gera muito trabalho extra para coletar a posição de cada imagem e, principalmente, para enquadrá-las adequadamente.

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Folhas de sprites (SpriteSheets)

- ➔ Podemos adotar uma abordagem sistematizada:
  - cada **spritesheet** ser dividida em espaços iguais, por isso as imagens devem estar corretamente distribuídas;
  - cada linha da **spritesheet** corresponderá um estado do **sprite** (ex.: parado, correndo para a direita, correndo para a esquerda);
  - a **animação** de um estado usará somente as figuras na sua linha própria.
- ➔ Isso gera grande quantidade de espaço vazio na imagem, aumentando o tamanho do arquivo, mas certamente facilitará muito a programação.

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Folhas de sprites (SpriteSheets)

- Após carregar a imagem, é necessário fazer os recortes (**clipping**):
  - Enquadramento da parte exata que queremos.
    - Exemplo: suponha que queremos a figurinha na linha 2, coluna 7 (contados a partir de zero).
    - Na hora de programar as **animações**, ficará muito mais fácil expressar dessa maneira. No entanto, precisaremos calcular a posição (x,y) do recorte da imagem.

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Folhas de sprites (SpriteSheets)

➔ Exemplo:

```
<!-- arquivo: clipping.html -->
<!DOCTYPE html>
<html>

<head>
   <title>
      Spritesheet - recorte e enquadramento (clipping)
   </title>
</head>

<body>
   <canvas id="canvas_clipping" width="500" height="500">
   </canvas>
   <script>
      var imgSonic = new Image();
      imgSonic.src = 'spritesheet.png';
      imgSonic.onload = function() {
         // Aqui faremos o clipping!
      }
   </script>
</body>

</html>
```

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Folhas de sprites (SpriteSheets)

- No evento onload da imagem, calcularemos as posições x e y do recorte.
- Como estamos considerando quadros de tamanhos iguais, basta dividir a largura total pelo número de colunas para obter a largura de um quadro.
- Depois basta multiplicar este valor pela coluna desejada, e temos a posição x onde o recorte se inicia.
- Para a posição y, o processo é análogo a partir da altura total e número de linhas:

```
imgSonic.onload = function() {
   // Passo estes valores conforme a minha spritesheet
   var linhas = 3;
   var colunas = 8;

   // Dimensão de cada quadro
   var largura = imgSonic.width / colunas;
   var altura = imgSonic.height / linhas;

   // Quadro que eu quero (expresso em linha e coluna)
   var queroLinha = 2;
   var queroColuna = 7;

   // Posição de recorte
   var x = largura * queroColuna;
   var y = altura * queroLinha;

   // Continua...
}
```

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Folhas de sprites (SpriteSheets)

- Tendo calculado x e y, largura e altura, basta lembrarmos do método do drawImage contexto.
  - Esse método pode ser chamado de duas formas:
    - Desenhando uma imagem inteira ⟹ context.drawImage(imagem, x, y, largura, altura)
    - Fazendo o **clipping** ⟹ context.drawImage(
      imagem,
      xOrigem,
      yOrigem,
      larguraOrigem,
      alturaOrigem,
      xDestino,
      yDestino,
      larguraDestino,
      alturaDestino
      );

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Folhas de sprites (SpriteSheets)

➔ No código:

```
var context =
    document.getElementById('canvas_clipping').getContext('2d');
context.drawImage(
   imgSonic,
   x,
   y,
   largura,
   altura,
   100, // Posição no canvas onde quero desenhar
   100,
   largura,
   altura
);
```

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Detecção de colisões

➔ Um dos métodos mais simples para detectar **colisões** é criar uma caixa delimitadora (*bounding box*) ao redor de cada sprite e verificar a interseção entre os retângulos:

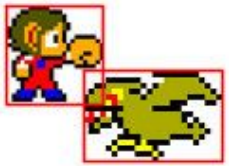
Sprites colidindo: retângulos (bounding boxes) apresentam intersecção

Sem intersecção, não há colisão

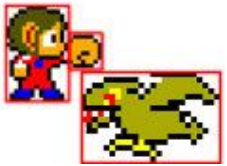
Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Detecção de colisões

➔ Um dos métodos mais simples para detectar **colisões** é criar uma caixa delimitadora (*bounding box*) ao redor de cada sprite e verificar a interseção entre os retângulos:



Retângulos intersectando sem haver a colisão real desejada.
O dragão colide com uma área vazia do sprite do herói.

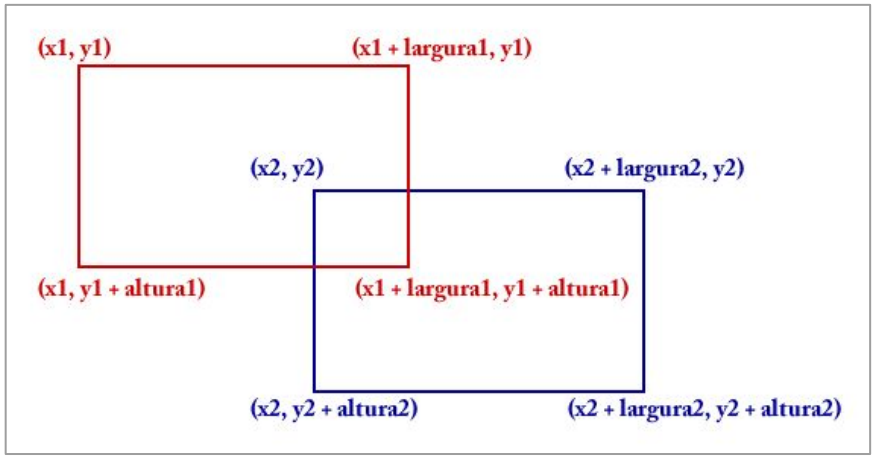


Definindo vários bounding boxes para objetos irregulares

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Detecção de colisões

➔ Definindo vários *bounding boxes* para objetos irregulares:



➔ Pela geometria, dois retângulos se intersectam se:

- $(x1$ + largura1) > $x2$ **E**
- $x1$ < ($x2$ + largura2) **E**
- ($y1$ + altura1) > $y2$ **E**
- $y1$ < ($y2$ + altura2)

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Animação de fundo com efeito Parallax

➔ O cenário do jogo terá o seguinte aspecto, em resolução de 500 por 500 pixels:

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Animação de fundo com efeito Parallax

➔ O fundo, no entanto, ser composto por 3 imagens separadas: uma para o gradiente, outra para as estrelas e outra para as nuvens. Estas duas últimas ter o fundo transparente, para podermos encaixar umas sobre as outras, como camadas:

Combinando os elementos do fundo

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Animação de fundo com efeito Parallax

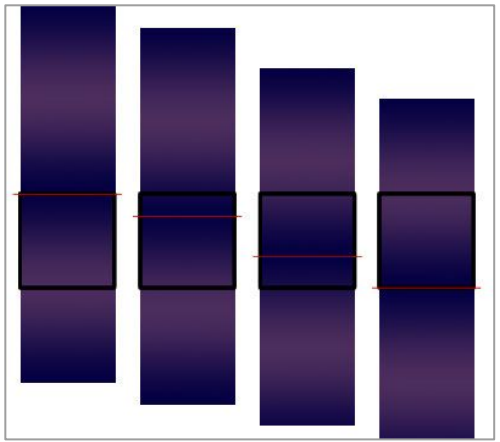
➔ Cada imagem possui 500 pixels de largura por 1000 pixels de altura (portanto, o dobro da área do jogo).
  - Elas irão rolar pelo cenário a velocidades diferentes, criando um efeito denominado **parallax** (paralaxe):

Os elementos do fundo rolam a velocidades diferentes

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Animação de fundo com efeito Parallax

- Durante a rolagem, cada imagem deverá ser desenhada no mínimo duas vezes, de forma a cobrir toda a área de desenho durante a rolagem.
- Se fosse uma imagem menor, teríamos que desenhá-la mais vezes.
- A cada ciclo, nós emendamos os desenhos a uma altura diferente (marcada com uma linha vermelha):



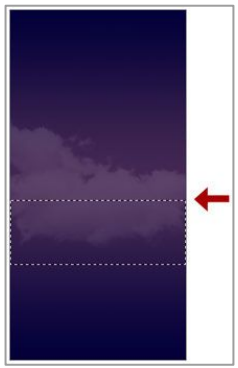
Cada imagem de fundo é desenhada quantas vezes forem necessárias para cobrir a área do jogo. Com imagens maiores que o Canvas, duas vezes são suficientes.

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

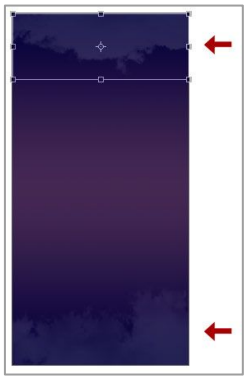
Animação de fundo com efeito Parallax

➔ Repare que os pontos inicial e final do gradiente têm de possuir a mesma cor para serem emendados (lembre-se disso ao criar as imagens em seu programa gráfico predileto).

- Da mesma forma, os extremos das outras figuras precisam encaixar-se perfeitamente! Por exemplo, para criar as nuvens, podemos proceder da maneira descrita nas figuras abaixo:



Quebramos uma nuvem pelo centro



Viramos cada metade de cabeça para baixo e encaixamos nas bordas da imagem

Desenvolvimento de Jogos com HTML5 Canvas e JavaScript

Animação de fundo com efeito Parallax

Imagens desceram muito, hora de reposicioná-las

Alguma dúvida?