



Desenvolvimento de jogos usando a tecnologia web

Seminfo 2019

Desenvolva jogos com
**HTML5 Canvas
e JavaScript**



 Casa do
Código

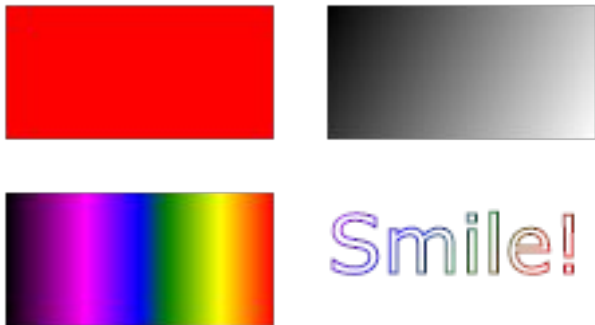
ÉDERSON CÁSSIO

Material baseado neste livro

Canvas HTML5

O canvas é um elemento HTML usado para desenhar gráficos em uma página web. Em outras palavras, o elemento serve para a realização de desenhos gráficos em uma página HTML de forma dinâmica usando Javascript.






O gráfico abaixo foi criado com o canvas. Ele mostra 4 elementos: um retângulo vermelho, um retângulo gradiente, um retângulo multicolorido e um texto multicolorido.



Canvas HTML5

O canvas é apenas um contêiner para gráficos. Você deve usar o Javascript para desenhar os gráficos. Ele possui vários métodos para desenhar caminhos, caixas, círculos, texto e adicionar imagens.

Os navegadores que suportam o elemento estão descritos abaixo:

Element					
<canvas>	4.0	9.0	2.0	3.1	9.0

Canvas HTML5

Um canvas é uma área retangular em uma página HTML. Por padrão, não possui borda nem conteúdo:

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

Nota: Sempre especifique um id para conseguir identificar o elemento via Javascript. Os atributos width e height (largura e altura) podem ser definidos posteriormente ou inseridos na marcação do elemento.

Canvas HTML5

Métodos para desenhar elementos no canvas:

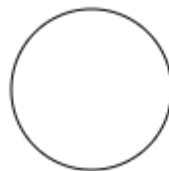
Desenhar uma linha

```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
ctx.moveTo(0, 0);  
ctx.lineTo(200, 100);  
ctx.stroke();
```



Desenhar um círculo

```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
ctx.beginPath();  
ctx.arc(95, 50, 40, 0, 2 * Math.PI);  
ctx.stroke();
```



Canvas HTML5

Métodos para desenhar elementos no canvas:

Desenhar um texto

```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
ctx.font = "30px Arial";  
ctx.fillText("Hello World", 10, 50);
```

Hello World

Desenhar um texto vazado

```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
ctx.font = "30px Arial";  
ctx.strokeText("Hello World", 10, 50);
```

Hello World

Canvas HTML5

Métodos para desenhar elementos no canvas:

Desenhar um gradiente linear

```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
// Create gradient  
var grd = ctx.createLinearGradient(0, 0, 200, 0);  
grd.addColorStop(0, "red");  
grd.addColorStop(1, "white");  
// Fill with gradient  
ctx.fillStyle = grd;  
ctx.fillRect(10, 10, 150, 80);
```



Canvas HTML5

Métodos para desenhar elementos no canvas:

Desenhar um gradiente circular

```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
// Create gradient  
var grd = ctx.createRadialGradient(75, 50, 5, 90, 60, 100);  
grd.addColorStop(0, "red");  
grd.addColorStop(1, "white");  
// Fill with gradient  
ctx.fillStyle = grd;  
ctx.fillRect(10, 10, 150, 80);
```



Canvas HTML5

Métodos para desenhar elementos no canvas:

Desenhar uma imagem

```
var c = document.getElementById("myCanvas");  
var ctx = c.getContext("2d");  
//a classe Image é uma das formas de exibir imagens via JS  
var img = new Image();  
//depois de instanciar, tem que se passar o caminho  
//da imagem  
img.src = "img/logo.png";  
ctx.drawImage(img, 10, 10);
```

Javascript

Javascript (ou ECMAScript) é uma linguagem de programação utilizada em praticamente todos os browsers existentes no mercado. Com ela, é possível implementar funcionalidades que requeiram algum tipo de lógica ou função mais sofisticada. No nosso mini curso vamos trabalhar com os seguintes aspectos na linguagem:

- Orientação a Objetos;
- Tratamento de Eventos de teclado;
- Renderização de Imagens;
- Reprodução de Áudios.



Atualmente, a linguagem se encontra na versão ES10, no entanto as versões mais utilizadas são a ES6/7.

Orientação a Objetos

Javascript não é uma linguagem que suporta de forma nativa a Orientação a Objetos, mas existem abordagens que auxiliam na hora de se programar utilizando tal paradigma. A seguir são mostradas três formas de desenvolvimento:

```
class Classe1{
  constructor(attr){
    this.attr = attr;
  }
  getAttr(){
    return this.attr;
  }
  setAttr(attr){
    this.attr = attr;
  }

  // ...
}
```

```
function Classe2(attr){
  this.attr = attr;
}

Classe2.prototype = {
  getAttr: function(){
    return this.attr;
  },
  setAttr: function(attr){
    this.attr = attr;
  }
}

// ...
};
```

```
function Classe3(attr){
  this.attr = attr;

  this.getAttr = function(){
    return this.attr;
  };
  this.setAttr = function(attr){
    this.attr = attr;
  };

  // ...
}
```

Orientação a Objetos

A instância das classes é feita da forma convencional, fazendo uso da palavra reservada **new**, e a utilização dos métodos é feita chamando-os através do <nome_do_objeto>.<nome_do_metodo>(...):

```
var c1 = new Classe1(10);  
var c2 = new Classe2(20);  
var c3 = new Classe3(30);  
  
var num = c1.getAttr();  
console.log(num);
```

Tratamento de Eventos de Teclado



O tratamento de eventos de teclado em Javascript é feito através da função **addEventListener**, onde o primeiro parâmetro indica o tipo de evento e o segundo é uma função de callback que executa alguma lógica ou comando.

Dentre os eventos de teclado, existem:

- **keyup**: determinada tecla foi solta;
- **keydown**: determinada tecla **foi** pressionada;
- **keypress**: determinada tecla **está** sendo pressionada.

Tratamento de Eventos de Teclado

```
document.addEventListener('keydown', function(event){
    //imprime o código da tecla que foi pressionada
    console.log(event.keyCode);
});

document.addEventListener('keyup', function(event){
    //imprime o código da tecla que foi solta
    console.log(event.keyCode);
});

document.addEventListener('keypress', function(event){
    //imprime o código da tecla que está pressionada
    console.log(event.keyCode);
});
```

Cada tecla do teclado possui um código, que auxilia na identificação de qual tecla foi/está sendo usada.

Renderização de Imagens

Para a renderização de imagens, usamos a classe Image, que instancia uma imagem pronta para exibição na tela:

```
var imagem = new Image();  
//após instanciar o objeto, precisamos definir  
//o caminho da imagem  
imagem.src = "imagens/pe-de-pano.png";  
//a classe suporta arquivos PNG e JPEG  
//talvez outros também
```


Reprodução de Áudios

Para a reprodução de áudios, usamos a classe Audio:

```
var audio = new Audio();  
//caminho do arquivo de audio  
audio.src = "sons/gemidao-do-zap.mp3";  
//posicionando o cursor de audio  
//no tempo referido  
audio.currentTime = 0;  
//volume do audio reproduzido  
audio.volume = 0.5;  
//carregando o arquivo  
audio.load();  
//reproduzindo  
audio.play();
```

A classe suporta arquivos MP3, M4A, OGG, WAV, talvez outros...

Atividade proposta

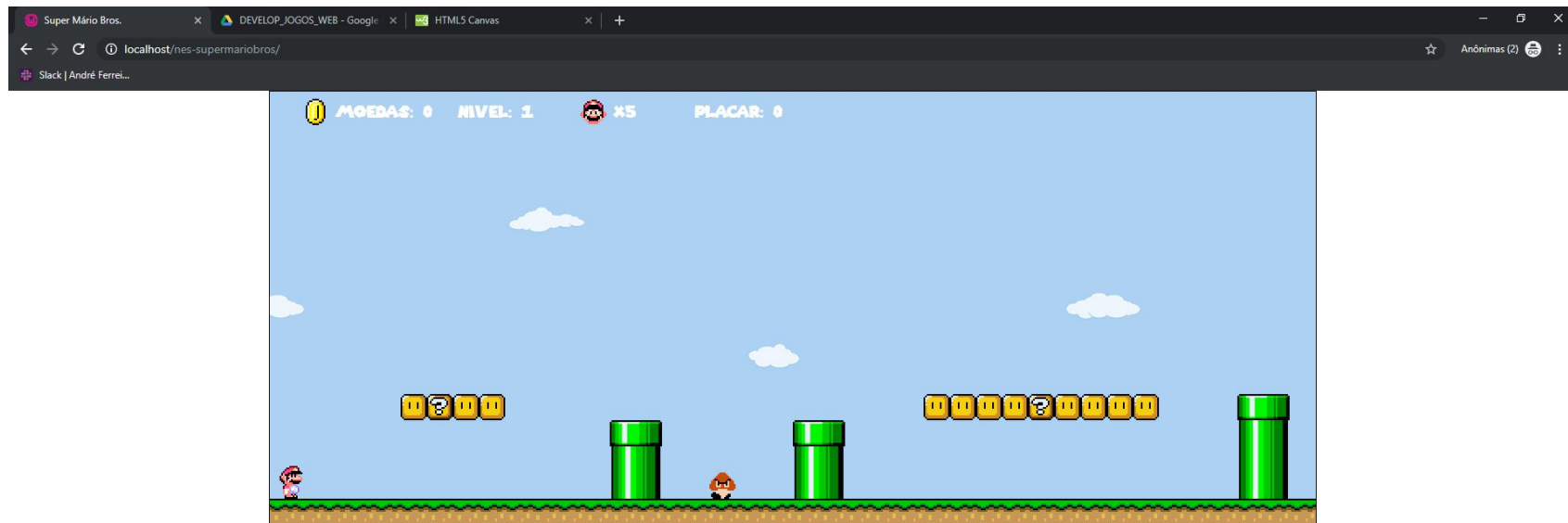
A ideia do mini curso é desenvolver um jogo usando o Canvas para as animações e Javascript para controle de elementos e lógica, então o jogo escolhido foi o Super Mario Bros. Para esta implementação seremos capazes de:

- Desenhar cenários usando os elementos básicos presentes (blocos de moedas, de elementos power up, tubos);
- Desenhar e controlar o Mario pelo cenário usando eventos de teclado;
- Passar entre as fases desenhadas de modos diferentes;
- Desenhar os goomas, um dos inimigos que tentam atrapalhar a jornada do Mario;
- Os power ups a serem desenvolvidos são:
 - Aumentar de tamanho;
 - Soltar foguinho



Atividade proposta

Ao final, espera-se que o seu jogo esteja com esta aparência:



Atividade proposta

Para adiantarmos algumas coisas, gostaria de pedir que baixasse o seguinte repositório pré-pronto:

<https://github.com/linnik-maciel/supermariobros.git>

Neste repositório, temos:

- arquivos CSS iniciais;
- fontes usadas para mensagens;
- sprites do jogo;
- arquivos de áudio para os sons característicos do jogo;
- arquivos de ferramentas iniciais, na pasta ferramentas/;
- gerenciamento do jogo, no arquivo MarioGame.js

Vamos aqui a uma breve explicação dos arquivos JS enviados inicialmente:



CarregarCoisas.js

A finalidade desta classe é carregar em memória as imagens utilizadas no jogo, além de fazer um pré-carregamento do mapa atual no jogo. Ela apresenta algumas variáveis globais para controle de elementos HTML que indicam o andamento do carregamento, além das imagens:

```
this.iniciar = function(){
    percentual = visao.create("div");
    visao.addClass(percentual, "percentual");
    visao.setHTML(percentual, "0%");
    visao.appendToBody(percentual);

    imagens = {
        1: 'imagens/bg.png',
        2: 'imagens/bullet.png',
        3: 'imagens/coin.png',
        4: 'imagens/elements.png',
        5: 'imagens/enemies.png',
        6: 'imagens/flag-pole.png',
        7: 'imagens/flag.png',
        8: 'imagens/mario-head.png',
        9: 'imagens/mario-sprites.png',
        10: 'imagens/powerups.png',
    };

    that.carregarImagens(imagens);
};
```

```
this.carregarImagens = function(imagens){
    var imagensOK = {};
    var carregadas = 0;
    var totalImagens = 0;

    for(var i in imagens){
        totalImagens++;
    }

    for(var i in imagens){
        imagensOK[i] = new Image();
        imagensOK[i].src = imagens[i];
        imagensOK[i].onload = function(){
            carregadas++;
            porcentagem = Math.floor(carregadas*100/totalImagens);
            visao.setHTML(percentual, porcentagem + "%");

            if(carregadas==totalImagens){
                visao.removeFromBody(percentual);
                that.iniciarMapa();
            }
        };
    }
};
```

```
this.iniciarMapa = function(){
    var config = Configuracoes.getInstancia();
    config.iniciar();
};

window.onload = function(){
    var carregador = new Carregador();
    carregador.iniciar();
};
```



Visao.js

A finalidade desta classe é gerenciar a adição/remoção de elementos HTML na página:

```
this.getContainer = function(){  
    var elemento = document.getElementsByClassName("main-wrapper")[0];  
  
    return elemento;  
};  
  
this.create = function(elemento){  
    var elemento = document.createElement(elemento);  
  
    return elemento;  
};  
  
this.addClass = function(elemento, classe){  
    elemento.className = classe;  
};
```



Visao.js

```
this.append = function(elementoPai, elementoFilho){
    if(elementoFilho.className=="score-wrapper"){
        elementoPai.insertBefore(elementoFilho, elementoPai.firstChild);
    }else if(elementoPai.lastChild&&elementoPai.lastChild.className=="score-wrapper"){
        elementoPai.insertBefore(elementoFilho, elementoPai.lastChild);
    }else{
        elementoPai.appendChild(elementoFilho);
    }
};

this.appendToBody = function(elementoFilho){
    document.body.appendChild(elementoFilho);
};

this.remove = function(elementoPai, elementoFilho){
    elementoPai.removeChild(elementoFilho);
};

this.removeFromBody = function(elementoFilho) {
    document.body.removeChild(elementoFilho);
};
```



Visao.js

```
this.style = function(elemento, styles){
    for(var property in styles){
        elemento.style[property] = styles[property];
    }
};

this.setHTML = function(elemento, conteudo){
    elemento.innerHTML = conteudo;
};
}

return {
    getInstancia: function(){
        if(instancia==null){
            instancia = new Visao();
        }

        return instancia;
    }
};
})();
```



InterfaceUI.js

A finalidade desta classe é gerenciar as ações de manipulação do Canvas, como desenhar imagens, limpar a área, definição de dimensões do elemento etc:

```
var canvas = document.getElementById("jogo");  
var contexto = canvas.getContext("2d");  
var that = this;
```

```
this.getWidth = function(){  
    return canvas.width;  
};  
  
this.getHeight = function(){  
    return canvas.height;  
};
```

```
this.setWidth = function(width){  
    canvas.width = width;  
};  
  
this.setHeight = function(height){  
    canvas.height = height;  
};
```

```
this.getCanvas = function(){  
    return canvas;  
};  
  
this.exibirCanvas = function(){  
    canvas.style.display = "block";  
};  
  
this.esconderCanvas = function(){  
    canvas.style.display = "none";  
};
```



InterfaceUI.js

```
this.limparTela = function(x, y, width, height){
    contexto.clearRect(x, y, width, height);
};

this.rolarAnimacao = function(x, y){
    contexto.translate(x, y);
};
```

```
return {
    getInstancia: function(){
        if(instancia==null){
            instancia = new Interface();
        }

        return instancia;
    }
};
```

```
this.desenharCaixa = function(x, y, width, height){
    contexto.rect(x, y, width, height);
    contexto.fillStyle = "black";
    contexto.fill();
};

this.escrever = function(text, x, y){
    contexto.font = "20px SuperMario256";
    contexto.fillStyle = "white";
    contexto.fillText(text, x, y);
};
```

```
this.desenhar = function(imagem, sx, sy, width, height, x, y, width, height){
    contexto.drawImage(imagem, sx, sy, width, height, x, y, width, height);
};
```



Configuracoes.js

A finalidade desta classe é gerenciar o início do jogo e carregamento dos cenários. Aqui a nossa abstração dos cenários os considerou como um array, onde cada posição indica qual elemento será disposto naquela região:

```
this.iniciarJogo = function(mapas){  
    marioGame.limparInstancias();  
    marioGame.iniciar(mapas, 1);  
};
```

```
return {  
    getInstancia: function(){  
        if(instancia==null){  
            instancia = new Configuracoes();  
        }  
  
        return instancia;  
    }  
};
```

```
this.iniciar = function(){  
    marioGame = new MarioGame();  
  
    mapas = that.carregarMapas();  
    that.iniciarJogo(mapas);  
};
```



[illegible]

MarioGame.js

É aqui que tá toda a violência do jogo. Neste arquivo fazemos a instância dos objetos principais do jogo, os ajustes na área de desenho, instanciamos todos os elementos como goombas e power up's, capturamos os eventos de teclado responsáveis por controlar o Mario, renderizamos os elementos do cenário e tratamos das colisões entre elementos e de elementos com o cenário.

Dentre os eventos de teclado possíveis, temos:

- setas direcionais para controlar o personagem;
- tecla ctrl dispara os foguinhos;
- tecla shift faz o personagem aumentar a velocidade durante a caminhada.

Já para as colisões possíveis, temos:

- verificação do personagem no cenário;
- verificação dos power up's no cenário;
- verificação dos goombas no cenário;
- verificação do foguinho no cenário;



MarioGame.js

Já para as colisões possíveis, temos:

- verificação do personagem no cenário;
- verificação dos power up's no cenário;
- verificação dos goombas no cenário;
- verificação do foguinho no cenário;
- verificação do personagem com o power up;
- verificação do personagem com o goomba;
- verificação do foguinho com o goomba;
- verificação se o personagem ainda está no cenário ou caiu.

Outros métodos de controle foram implementados, como atualização da posição do personagem, verificação da posição, indicação do final do cenário, pausa e reinício do jogo.



Classes a serem implementadas

Os arquivos que serão implementados serão:

- para renderização do personagem principal;
- para renderização dos cenários;
- para renderização dos inimigos;
- para renderização de outros elementos;
- montagem do placar;
- controle dos sons reproduzidos no jogo;



Mario.js

Aqui implementamos a renderização da animação do personagem. A classe possui os seguintes atributos:

- estado atual do personagem: pequeno, grande, pegando fogo;
- coordenadas X e Y da imagem;
- largura e altura dos frames da imagem;
- velocidade de locomoção, e velocidades de cada coordenada;
- estados: pulando, no chão, invulnerável;
- velocidade das sprites em cada coordenada;
- controle dos frames.

Os métodos implementados são:

- iniciar;
- desenhar;
- checarTipoMario;
- mudarPosicao.



Mario.js

```
this.iniciar = function(){  
    that.x = 10;  
    that.y = interface.getHeight() - 40 - 40;  
  
    marioSprite = new Image();  
    marioSprite.src = "imagens/mario-sprites.png";  
};
```

```
this.desenhar = function(){  
    that.sX = that.width * that.frame;  
    interface.desenhar(marioSprite, that.sX, that.sY, that.width, that.height, that.x, that.y, that.width, that.height);  
};
```



Mario.js

```
this.checarTipoMario = function(){
  if(that.tipo=="grande"){
    that.height = 60;

    if(that.invulneravel){
      that.sY = 276;
    }else{
      that.sY = 90;
    }
  }else if(that.tipo=="pequeno"){
    that.height = 44;

    if(that.invulneravel){
      that.sY = 222;
    }else{
      that.sY = 4;
    }
  }else if(that.tipo=="pegando-fogo"){
    that.height = 60;
    that.sY = 150;
  }
};
```

```
this.mudarPosicao = function(){
  that.x = canvas.width/10;
  that.y = canvas.height - 40;
  that.frame = 0;
};
```



Inimigo.js

Aqui implementamos a renderização da animação dos goomas. A classe possui os seguintes atributos:

- estado atual do personagem: morto, morto pelo foguinho ou de boas;
- coordenadas X e Y da imagem;
- largura e altura dos frames da imagem;
- velocidades de cada coordenada;
- estados: no chão;
- velocidade das sprites em cada coordenada;
- controle dos frames.
- além deles, há dois atributos que controlam a aparição/desaparecimento dos goomas no cenário;

Os métodos implementados são:

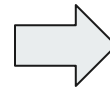
- gooma;
- desenhar;
- atualizar;



Inimigo.js

```
this.goomba = function(){  
  this.tipo = 20;  
  that.sX = 0;  
};
```

```
this.atualizar = function(){  
  var gravidade = 0.2;  
  
  if(that.noChao){  
    that.velY = 0;  
  }  
  
  if(that.estado=="morto"){  
    that.frame = 2;  
    tickCounter++;  
    if(tickCounter>=60){  
      that.frame = 4;  
    }  
  }  
}
```



```
    }else if(that.estado=="mortoPeloFoguinho"){  
      that.frame = 3;  
      that.velY += gravidade;  
      that.y += that.velY;  
    }else{  
      that.velY += gravidade;  
      that.x += that.velX;  
      that.y += that.velY;  
  
      tickCounter++;  
      if(tickCounter>maxTick){  
        tickCounter = 0;  
        if(that.frame==0){  
          that.frame = 1;  
        }else{  
          that.frame = 0;  
        }  
      }  
    }  
  }  
};
```

```
this.desenhar = function(){  
  that.sX = that.width * that.frame;  
  interface.desenhar(elemento, that.sX, that.sY, that.width, that.height, that.x, that.y, that.width, that.height);  
};
```



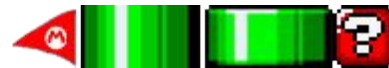
Elemento.js

Aqui implementamos a renderização dos elementos do cenário. A classe possui os seguintes atributos:

- tipo (no caso, qual elemento);
- coordenadas X e Y da imagem;
- largura e altura dos frames da imagem;
- velocidade das sprites em cada coordenada;

Os métodos implementados são a indicação de qual elemento será renderizado:

- | | |
|------------------|----------------|
| • plataforma; | • canoEsq; |
| • caixaDeMoeda; | • canoDir; |
| • caixaSurpresa; | • topoCanoEsq; |
| • caixaUsada; | • topoCanoDir; |
| • posteBandeira; | • desenhar; |
| • bandeira; | |



Elemento.js

```
this.plataforma = function(){  
    that.tipo = 1;  
    that.sX = 0;  
};  
  
this.caixaDeMoeda = function(){  
    that.tipo = 2;  
    that.sX = 1 * that.width;  
};  
  
this.caixaSurpresa = function(){  
    that.tipo = 3;  
    that.sX = 2 * that.width;  
};  
  
this.caixaUsada = function(){  
    that.tipo = 4;  
    that.sX = 3 * that.width;  
};
```

```
this.posteBandeira = function(){  
    that.tipo = 5;  
    that.sX = 4 * that.width;  
};  
  
this.bandeira = function(){  
    that.tipo = 6;  
    that.sX = 5 * that.width;  
};  
  
this.canoEsq = function(){  
    that.tipo = 7;  
    that.sX = 6 * that.width;  
};
```

```
this.canoDir = function(){  
    that.tipo = 8;  
    that.sX = 7 * that.width;  
};  
  
this.topoCanoEsq = function(){  
    that.tipo = 9;  
    that.sX = 8 * that.width;  
};  
  
this.topoCanoDir = function(){  
    that.tipo = 10;  
    that.sX = 9 * that.width;  
};
```



Elemento.js

```
this.desenhar = function(){  
  interface.desenhar(elemento, that.sX, that.sY, that.width, that.height, that.x, that.y, that.width, that.height);  
};
```



PowerUp.js

Aqui implementamos os power ups disponíveis no jogo. A classe possui os seguintes atributos:

- tipo (cogumelo ou flor);
- coordenadas X e Y da imagem;
- velocidades de cada coordenada;
- estados: no chão;
- velocidade das sprites em cada coordenada;
- largura e altura dos frames da imagem;

Os métodos implementados são a indicação de qual elemento será renderizado:

- cogumelo;
- flor;
- desenhar;
- atualizar;



PowerUp.js

```
this.cogumelo = function(x, y){  
    that.x = x;  
    that.y = y - that.height;  
    that.tipo = 30;  
    that.sX = 0;  
};
```

```
this.flor = function(x, y){  
    that.x = x;  
    that.y = y - that.height;  
    that.tipo = 31;  
    that.sX = 32;  
};
```

```
this.atualizar = function(){  
    if(that.tipo==30){  
        var gravidade = 0.2;  
  
        if(that.noChao){  
            that.velY = 0;  
        }  
  
        that.velY += gravidade;  
        that.x += that.velX;  
        that.y += that.velY;  
    }  
};
```

```
this.desenhar = function(){  
    interface.desenhar(elemento, that.sX, that.sY, that.width, that.height, that.x, that.y, that.width, that.height);  
};
```



Foguinho.js

Aqui implementamos a renderização do foguinho no jogo. A classe possui os seguintes atributos:

- coordenadas X e Y da imagem;
- velocidades de cada coordenada;
- estados: no chão;
- velocidade das sprites em cada coordenada;
- largura e altura dos frames da imagem;

Os métodos implementados são:

- iniciar;
- desenhar;
- atualizar;



Foguinho.js

```
this.iniciar = function(x, y, direcao){  
    that.velX = 8 * direcao;  
    that.velY = 0;  
    that.x = x + that.width;  
    that.y = y + 30;  
    that.tipo = 30;  
    that.sX = 0;  
};
```

```
this.atualizar = function(){  
    var gravidade = 0.2;  
  
    if(that.noChao){  
        that.velY = -4;  
        that.noChao = false;  
    }  
  
    that.velY += gravidade;  
    that.x += that.velX;  
    that.y += that.velY;  
};
```

```
this.desenhar = function(){  
    interface.desenhar(elemento, that.sX, that.sY, that.width, that.height, that.x, that.y, that.width, that.height);  
};
```



Sons.js

Aqui implementamos a reprodução dos sons no jogo. Aqui apenas objetos globais da classe Audio.

Os métodos implementados são:

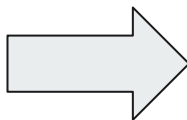
- iniciar;
- play;

```
this.iniciar = function(){  
    coin = new Audio('sons/coin.wav');  
    powerUpAppear = new Audio('sons/power-up-appear.wav');  
    powerUp = new Audio('sons/power-up.wav');  
    marioDie = new Audio('sons/mario-die.wav');  
    killEnemy = new Audio('sons/kill-enemy.wav');  
    stageClear = new Audio('sons/stage-clear.wav');  
    bullet = new Audio('sons/bullet.wav');  
    powerDown = new Audio('sons/power-down.wav');  
    jump = new Audio('sons/jump.wav');  
};
```



Sons.js

```
this.play = function(elemento){
  switch(elemento){
    case "coin":
      coin.pause();
      coin.currentTime = 0;
      coin.play();
      break;
    case "powerUpAppear":
      powerUpAppear.pause();
      powerUpAppear.currentTime = 0;
      powerUpAppear.play();
      break;
    case "powerUp":
      powerUp.pause();
      powerUp.currentTime = 0;
      powerUp.play();
      break;
    case "marioDie":
      marioDie.pause();
      marioDie.currentTime = 0;
      marioDie.play();
      break;
```



```
    case "killEnemy":
      killEnemy.pause();
      killEnemy.currentTime = 0;
      killEnemy.play();
      break;
    case "stageClear":
      stageClear.pause();
      stageClear.currentTime = 0;
      stageClear.play();
      break;
    case "bullet":
      bullet.pause();
      bullet.currentTime = 0;
      bullet.play();
      break;
    case "powerDown":
      powerDown.pause();
      powerDown.currentTime = 0;
      powerDown.play();
      break;
    case "jump":
      jump.pause();
      jump.currentTime = 0;
      jump.play();
      break;
  }
};
```



Placar.js

Aqui implementamos a renderização do placar do jogo. A classe possui os seguintes atributos:

- total de moedas coletadas;
- total de pontos marcados;
- total de vidas;

Os métodos implementados são:

- iniciar;
- atualizarMoedas;
- atualizarPlacar;
- atualizarVidas;
- atualizarNivel;
- mostrarPlacar;
- esconderPlacar;
- mostrarGameOver;



Placar.js

```
this.iniciar = function(){
    that.moedas = 0;
    that.totalPlacar = 0;
    that.vidas = 5;

    container = visao.getContainer();

    placar_container = visao.create("div");
    moeda_container = visao.create("div");
    totalPlacar_container = visao.create("div");
    vidas_container = visao.create("div");
    nivel_container = visao.create("div");

    visao.addClass(placar_container, "score-wrapper");
    visao.addClass(moeda_container, "coin-score");
    visao.addClass(totalPlacar_container, "total-score");
    visao.addClass(vidas_container, "life-count");
    visao.addClass(nivel_container, "level-num");

    visao.append(placar_container, moeda_container);
    visao.append(placar_container, nivel_container);
    visao.append(placar_container, vidas_container);
    visao.append(placar_container, totalPlacar_container);
    visao.append(container, placar_container);

    that.atualizarMoedas();
    that.atualizarPlacar();
    that.atualizarVidas();
    that.atualizarNivel(1);
};
```

```
this.atualizarMoedas = function(){
    if(that.moedas==100){
        that.moedas = 0;
        that.vidas++;
        that.atualizarVidas();
    }

    visao.setHTML(moeda_container, "Moedas: " + that.moedas);
};
```

```
this.atualizarPlacar = function(){
    visao.setHTML(totalPlacar_container, "Placar: " + that.totalPlacar);
};
```

```
this.atualizarVidas = function(){
    visao.setHTML(vidas_container, "x" + that.vidas);
};
```



Placar.js

```
this.atualizarNivel = function(nivel){  
    visao.setHTML(nivel_container, "Nivel: " + nivel);  
};
```

```
this.mostrarPlacar = function(){  
    visao.style(placar_container, { display: "block", background: "#add1f3" });  
};
```

```
this.esconderPlacar = function(){  
    visao.style(placar_container, { display: "none" });  
};
```

```
this.mostrarGameOver = function(){  
    visao.style(placar_container, { background: "black" });  
}
```



index.html

Ao final, teremos o nosso index com as seguintes marcações:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=1280" />
  <title>Super Mário Bros.</title>
  <link rel="stylesheet" type="text/css" href="css/style.css">
  <link rel="stylesheet" type="text/css" href="css/reset.css">
</head>
<body>
  <div class="main-wrapper">
    <canvas id="jogo"></canvas>
  </div>

  <script src="js/ferramentas/Visao.js"></script>
  <script src="js/ferramentas/InterfaceUI.js"></script>
  <script src="js/ferramentas/Configuracoes.js"></script>
  <script src="js/ferramentas/CarregarCoisas.js"></script>

  <script src="js/Sons.js"></script>
  <script src="js/Elemento.js"></script>
  <script src="js/PowerUp.js"></script>
  <script src="js/Inimigo.js"></script>
  <script src="js/Foguinho.js"></script>
  <script src="js/Mario.js"></script>
  <script src="js/Placar.js"></script>
  <script src="js/MarioGame.js"></script>
</body>
</html>
```

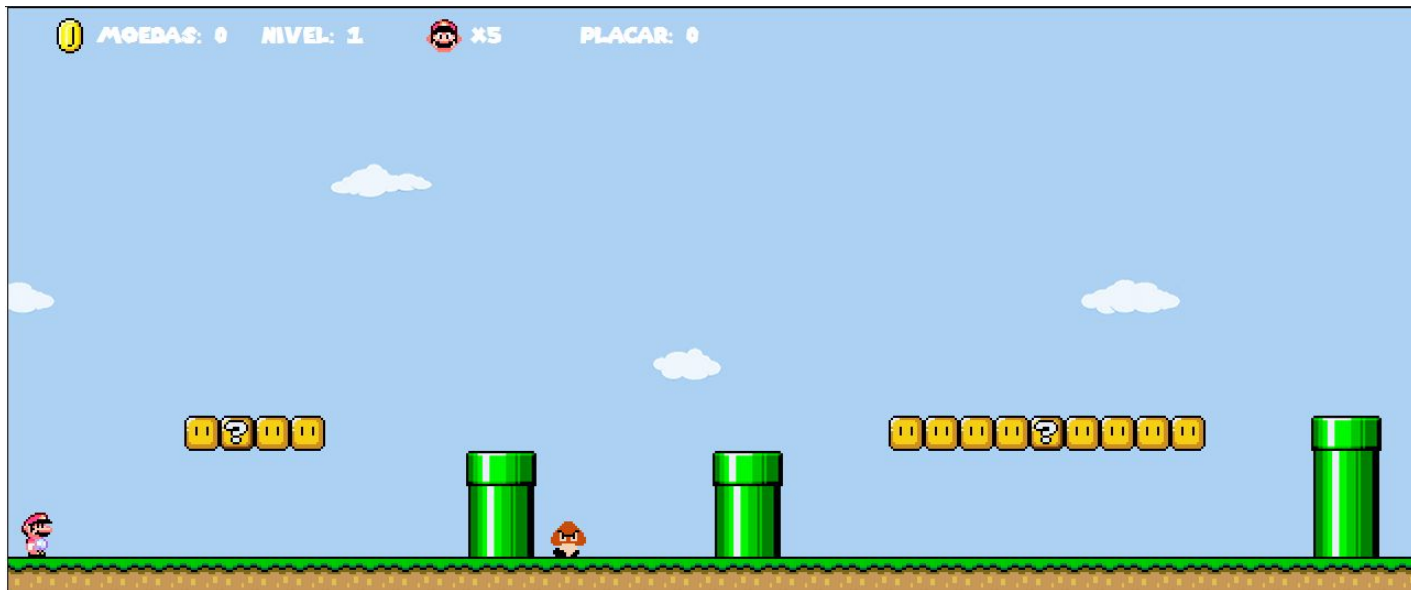
```
└─ css
   # reset.css
   # style.css
  ▸ fontes
  ▸ imagens
  └─ js
     └─ ferramentas
        JS CarregarCoisas.js
        JS Configuracoes.js
        JS InterfaceUI.js
        JS Visao.js
     JS Elemento.js
     JS Foguinho.js
     JS Inimigo.js
     JS Mario.js
     JS MarioGame.js
     JS Placar.js
     JS PowerUp.js
     JS Sons.js
  ▸ sons
  < index.html
```

Estrutura de arquivos do projeto



Conclusões

Pudemos perceber que a tecnologia web é bem versátil, nos proporcionando a experiência de desenvolver jogos interessantes usando a mesma.



Obrigado pela
participação :)