



UNIVERSIDADE CATÓLICA DE ANGOLA

FACULDADE DE ENGENHARIA

Interação Homem Máquina

*JavaScript: Origem, Evolução e Aplicações, Uma Análise
Comparativa e Demonstração Prática de Desenvolvimento de
Um Mini-Jogo*

Luis Domingos Marques – 32700

Luanda, 2024

ÍNDICE

ÍNDICE DE FIGURAS	1
1. ORIGEM E EVOLUÇÃO	2
1.2. Aplicações em Desenvolvimento Web e Mobile.....	2
1.3. Var, Let e Const	2
1.4. Promises	3
1.5. Funções Assíncronas, Async e Await.....	4
1.6. Funções Anônimas	5
1.7. Arrow Function.....	6
1.8. Callbacks.....	6
2. NA WEB	7
2.1. Árvore DOM e sua Importância	7
2.2. Hierarchy da DOM.....	7
2.3. API's da DOM	8
2.3.1. API de Seleção de Elementos:	8
2.3.2. API de Manipulação de Elementos:	9
2.3.3. API de Manipulação de Conteúdo:.....	9
2.3.4. API de Navegação e Histórico:	10
2.3.5. API de Armazenamento de Dados:	10
3. DESENVOLVIMENTO DE UM JOGO WEB USANDO CONCEITOS DE POO, PROGRAMAÇÃO FUNCIONAL E DOM COM JAVASCRIPT	11
3.1. Personagens do Jogo e Estrutura de Código	11

Personagem Principal e Animações Manuais via JS	11
Código Js do Player e Suas Configurações para Animações	12
Animação Manual do Player Usando Array Function	13
3.2. Animações com Chamadas de Instruções CSS e a Função setInterval.....	14
Chamadas de Css.....	14
Classe Zombie e Suas Instancializações	15
Instancias em Posições Aleatórias e Referencia via DOM.....	16
Método Construtor do Zombie	17
Método de Inserção dos Zombies na View	17
Função para Instancia de Array de Zombies	17
Um Dos Usos do setInterval	18
Controle de Animação Usando Eventos do Teclado	18
3.3. Importante	19
4. COMPARAÇÃO COM JAVA E C# EM POO E TIPAGEM	20
4.1. Vantagens do JavaScript em relação ao Java	20
4.2. Desvantagens do JavaScript em relação ao Java	21
CONCLUSÃO	22

ÍNDICE DE FIGURAS

Figura 1. Uso de Promise (Fonte: Autor).....	4
Figura 2. Async em declaração de funções assíncronas (Fonte:Autor).....	4
Figura 3. Chamadas de funções Assíncronas (Fonte:Autor)	5
Figura 4 Função anónima com parâmetros (Fonte:Autor).....	5
Figura 5. Função Arrow (Fonte:Autor)	6
Figura 6. CallBack chamadas e declaração (Fonte:Autor).....	6
Figura 7. Manipulação da DOM (Fonte:Autor).....	7
Figura 8. Cenário do Jogo (Fonte:Autor)	11
Figura 9. Sprites da animação de corrida do player (Fonte: Assets story).....	11
Figura 10. Estrutura de arquivos do projecto (Fonte:Autor).....	12
Figura 11. Configuração das animações do player (Fonte:Autor).....	13
Figura 12. Função de corrida do player (Fonte:Autor).....	14
Figura 13. Controle da animação do cenário movimentado (Fonte:Autor).....	15
Figura 14. Classe Zombie (Fonte:Autor)	16
Figura 15. Construtor da classe Zombie (Fonte:Autor)	17
Figura 16. Método da classe Zombie para desenhar zombie na tela (Fonte:Autor).....	17
Figura 17. Gerador Aleatório de instâncias da classe Zombie (Fonte:Autor).....	18
Figura 18. Controlador de período de criação de Zombies (Fonte:Autor).....	18
Figura 19. Controle de evento do Teclado (Fonte:Autor)	19
Figura 20. Inclusão de códigos js no html (Fonte:Autor)	19

1. ORIGEM E EVOLUÇÃO

JavaScript foi criado em 1995 por Brendan Eich, enquanto trabalhava na Netscape Communications Corporation. Inicialmente, foi concebido para adicionar interatividade a páginas web estáticas. Sua popularidade cresceu rapidamente devido à sua capacidade de ser executado diretamente no navegador do usuário, permitindo a criação de páginas dinâmicas.

Ao longo dos anos, JavaScript evoluiu significativamente. A padronização do ECMAScript trouxe melhorias na linguagem e estabilidade para desenvolvedores. O surgimento de bibliotecas e frameworks como jQuery, Angular, React e Vue expandiu suas capacidades, possibilitando a construção de aplicações web complexas.

1.2. Aplicações em Desenvolvimento Web e Mobile

JavaScript é uma das linguagens mais utilizadas no desenvolvimento web. Ele é essencial para a criação de sites dinâmicos, aplicações web de uma única página (SPA), aplicações de mensagens instantâneas, jogos online e muito mais. Além disso, com o advento do Node.js, JavaScript também é usado no desenvolvimento de servidores, permitindo a construção de aplicações web completas no lado do servidor.

No desenvolvimento mobile, frameworks como React Native e Ionic permitem que os desenvolvedores usem JavaScript para criar aplicativos móveis nativos para iOS e Android.

1.3. Var, Let e Const

JavaScript é uma linguagem de programação de alto nível, leve e interpretada. Alguns dos comandos mais comuns e sua sintaxe incluem:

1. **Var:** **var** era a forma mais antiga de declarar variáveis em JavaScript. Ela tem escopo de função, o que significa que a variável declarada com **var** é visível em toda a função em que foi declarada.

```
var numero = 10;
```

```
function mostrarNumero() {  
    var outroNumero = 20;  
    console.log(numero); // 10  
}  
  
console.log(outroNumero); // Erro: outroNumero is not defined
```

2. **Let:** let foi introduzido no ECMAScript 6 (ES6) e possui escopo de bloco, o que significa que a variável é visível apenas dentro do bloco em que foi declarada.

Exemplo:

```
let numero = 10;  
if (true) {  
    let outroNumero = 20;  
    console.log(numero); // 10  
}  
  
console.log(outroNumero); // Erro: outroNumero is not defined
```

3. **Const:** const também foi introduzido no ES6 e é usado para declarar constantes. O valor de uma constante não pode ser alterado após a sua atribuição inicial, e ela também tem escopo de bloco.

Exemplo:

```
const PI = 3.14;  
  
PI = 3; // Erro: Assignment to constant variable
```

1.4.Promises

Promises são objetos usados para representar a eventual conclusão ou falha de uma operação assíncrona. Elas são usadas para lidar com operações assíncronas de forma mais limpa e legível, evitando o chamado "callback hell".



```

1 function fazerRequisicao() {
2   return new Promise((resolve, reject) => {
3     setTimeout(() => {
4       if (sucesso) {
5         resolve("Dados da requisição");
6       } else {
7         reject("Erro na requisição");
8       }
9     }, 2000);
10  });
11 }
12
13 fazerRequisicao()
14   .then(dados => console.log(dados))
15   .catch(erro => console.error(erro));

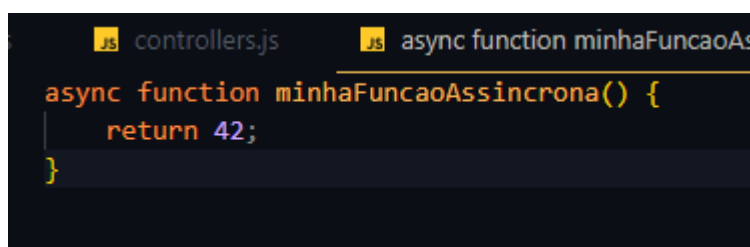
```

Figura 1. Uso de Promise (Fonte: Autor)

1.5. Funções Assíncronas, Async e Await

1. **Funções Assíncronas:** Funções assíncronas são funções que operam de forma assíncrona, o que significa que elas podem executar outras tarefas enquanto aguardam o término de uma operação assíncrona, como uma requisição de rede ou leitura de arquivo.
2. **Async:** A palavra-chave **async** é usada para declarar uma função assíncrona. Ela permite que a função retorne uma Promise, que resolve com o valor retornado pela função assíncrona, ou rejeita com o erro lançado pela função.

Exemplo:



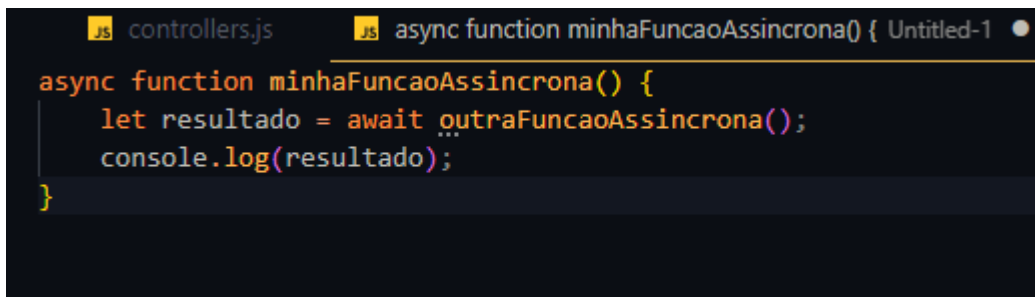
```

1 async function minhaFuncaoAssincrona() {
2   return 42;
3 }

```

Figura 2. Async em declaração de funções assíncronas (Fonte: Autor)

Await: A palavra-chave await é usada para esperar que uma Promise seja resolvida antes de continuar a execução do código. Ela só pode ser usada dentro de uma função assíncrona.



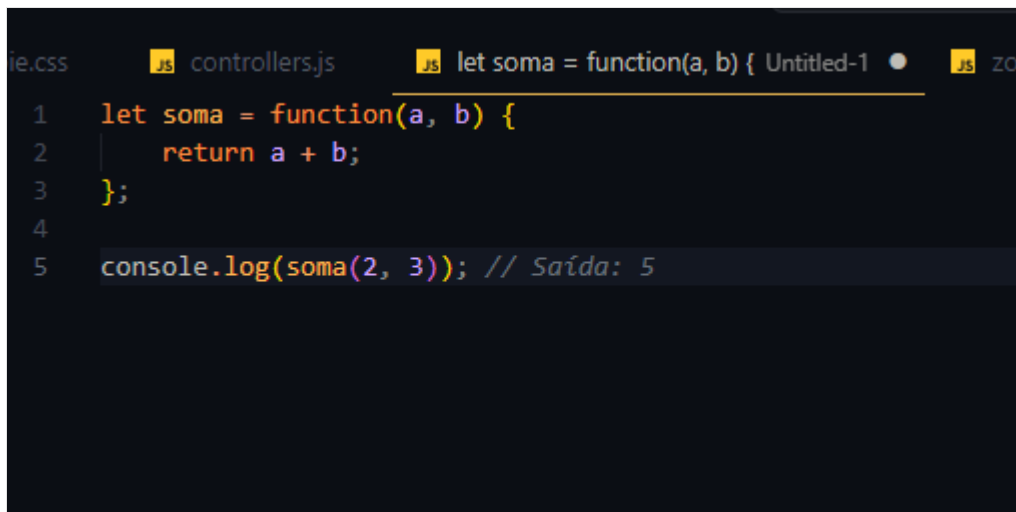
```
controllers.js  async function minhaFuncaoAssincrona() { Untitled-1 ●
async function minhaFuncaoAssincrona() {
  let resultado = await outraFuncaoAssincrona();
  console.log(resultado);
}
```

Figura 3. Chamadas de funções Assincronas (Fonte:Autor)

1.6.Funções Anônimas

Uma função anônima é uma função que não possui um nome associado a ela. Elas são úteis em situações onde você precisa passar uma função como argumento para outra função, ou quando você precisa de uma função temporária que não será reutilizada em outros lugares do código.

Exemplo:

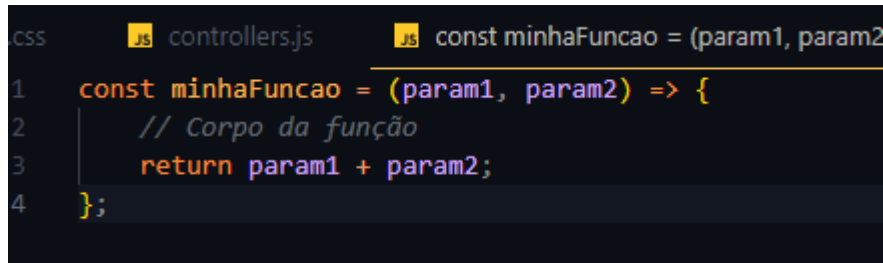


```
ie.css  controllers.js  let soma = function(a, b) { Untitled-1 ●  zo
1  let soma = function(a, b) {
2    return a + b;
3  };
4
5  console.log(soma(2, 3)); // Saída: 5
```

Figura 4 Função anônima com parâmetros (Fonte:Autor)

1.7.Arrow Function

As arrow functions são uma forma mais concisa de escrever funções em JavaScript. Elas foram introduzidas no ECMAScript 6 (ES6) e oferecem algumas vantagens em relação às funções tradicionais, principalmente em relação à sintaxe mais enxuta e ao comportamento léxico do **this**.

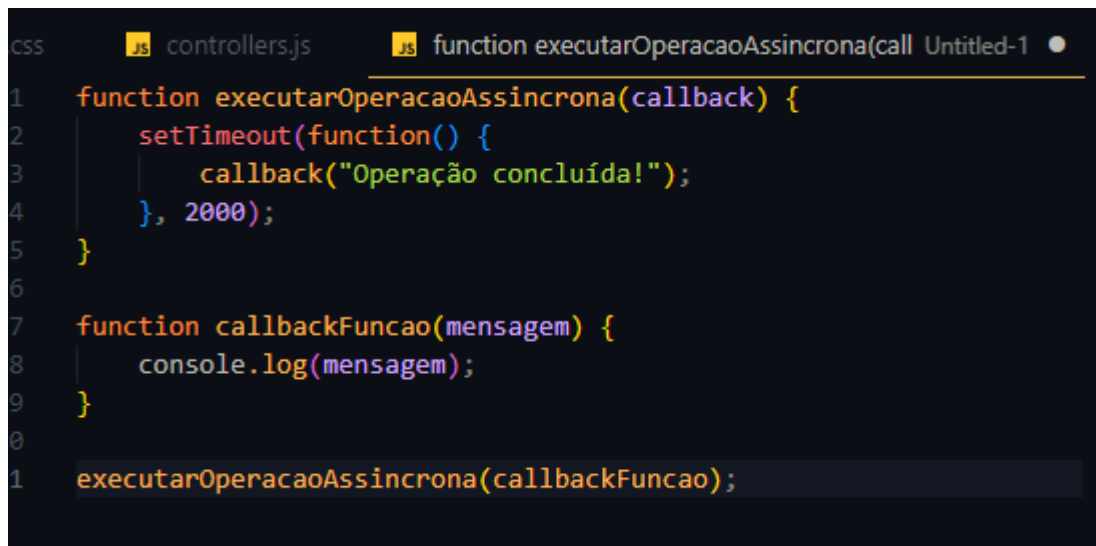


```
css      JS controllers.js      JS const minhaFuncao = (param1, param2
1  const minhaFuncao = (param1, param2) => {
2      // Corpo da função
3      return param1 + param2;
4  };
```

Figura 5. Função Arrow (Fonte:Autor)

1.8.Callbacks

Um callback é uma função que é passada como argumento para outra função e é executada após a conclusão de uma operação assíncrona ou de alguma tarefa específica, comumente usa-se muito a sintaxe de funções anônimas ou arrow function em declarações de callbacks



```
css      JS controllers.js      JS function executarOperacaoAssincrona(call Untitled-1 ●
1  function executarOperacaoAssincrona(callback) {
2      setTimeout(function() {
3          callback("Operação concluída!");
4      }, 2000);
5  }
6
7  function callbackFuncao(mensagem) {
8      console.log(mensagem);
9  }
10
11  executarOperacaoAssincrona(callbackFuncao);
```

Figura 6. CallBack chamadas e declaração (Fonte:Autor)

2. NA WEB

No desenvolvimento web, que é aqui o foco, o javascript é a linguagem mais usada a nível do mundo, sendo um recurso que disponibiliza diversas opções para manipulação e interação com o usuário, tem-se as seguintes características.

2.1. Árvore DOM e sua Importância

O DOM (Document Object Model) é uma interface de programação para documentos HTML e XML. Ele representa a estrutura de uma página web como uma árvore de objetos, onde cada nó representa um elemento da página (tag). A manipulação do DOM é essencial para criar páginas interativas e dinâmicas. Aqui está um exemplo de como manipular o DOM:

A screenshot of a code editor with a dark theme. The editor shows a file named 'Untitled-1' with a mix of HTML and JavaScript code. The HTML part defines a document structure with a title 'Exemplo de DOM' and a body containing a div with id 'meuElemento' and some initial content. The JavaScript part, enclosed in a script tag, uses 'document.getElementById' to find the element and then changes its 'innerHTML' to 'Novo conteúdo' and its 'style.color' to 'red'. The code is line-numbered from 1 to 15.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Exemplo de DOM</title>
5 </head>
6 <body>
7   <div id="meuElemento">Conteúdo Original</div>
8   <script>
9     // Manipulação do DOM
10    const elemento = document.getElementById("meuElemento");
11    elemento.innerHTML = "Novo conteúdo";
12    elemento.style.color = "red";
13  </script>
14 </body>
15 </html>
```

Figura 7. Manipulação da DOM (Fonte:Autor)

2.2. Hierarquia da DOM

A hierarquia de objetos do DOM (Document Object Model) é uma estrutura em forma de árvore que representa todos os elementos HTML de uma página web, bem como suas relações de pai-filho. Isso significa que cada elemento HTML é representado por um objeto no DOM, e esses objetos são organizados de acordo com a estrutura da página.

1. **Document:** O objeto **Document** é o nó raiz da hierarquia do DOM. Ele representa toda a página HTML e fornece métodos e propriedades para acessar e manipular os elementos HTML dentro dela.
2. **Element:** Os objetos **Element** representam os elementos HTML na página, como `<div>`, `<p>`, `<a>`, entre outros. Cada elemento é um nó filho do objeto **Document** ou de outro elemento HTML, criando assim a estrutura em árvore.
3. **TextNode:** Os objetos **TextNode** representam o conteúdo de texto dentro de um elemento HTML. Eles são nós folha na árvore do DOM e contêm apenas texto, sem tags HTML.
4. **Attribute:** Os objetos **Attribute** representam os atributos de um elemento HTML, como **id**, **class**, **src**, etc. Eles estão associados aos elementos HTML e podem ser acessados e manipulados através de métodos específicos.

Tem-se como objecto pai o objecto window, que faz referência a janela do navegador, contendo métodos de interação com o usuário, sendo como um container de todos os elementos da DOM, de realçar que variáveis sem var, const ou let, são atributos do window.

2.3. API's da DOM

APIs do DOM (Document Object Model) são conjuntos de interfaces e métodos fornecidos pelo navegador para interagir com os elementos HTML de uma página web. Essas APIs permitem que os desenvolvedores manipulem dinamicamente o conteúdo, a estrutura e o estilo da página, além de lidar com eventos e realizar outras operações úteis. Aqui estão algumas das APIs do DOM mais comuns:

2.3.1. API de Seleção de Elementos:

- **getElementById():** Retorna uma referência para o elemento com o ID especificado.
- **getElementsByClassName():** Retorna uma coleção de elementos que possuem a classe especificada.

- **getElementsByTagName():** Retorna uma coleção de elementos com o nome da tag HTML especificada.
- **querySelector():** Retorna o primeiro elemento que corresponde ao seletor CSS especificado.
- **querySelectorAll():** Retorna todos os elementos que correspondem ao seletor CSS especificado.

2.3.2. API de Manipulação de Elementos:

- **createElement():** Cria um novo elemento HTML.
- **appendChild():** Adiciona um nó filho ao final da lista de filhos de um nó pai especificado.
- **removeChild():** Remove um nó filho de um nó pai especificado.
- **setAttribute() / getAttribute():** Define ou retorna o valor de um atributo de um elemento.
- **classList:** Permite adicionar, remover ou alternar classes em um elemento.

2.3.3. API de Manipulação de Conteúdo:

- **textContent:** Propriedade que representa o conteúdo textual de um nó e de seus descendentes.
- **innerHTML:** Propriedade que representa o conteúdo HTML de um elemento.

2. API de Estilo:

- **style:** Propriedade que fornece acesso aos estilos CSS de um elemento.

3. API de Eventos:

- **addEventListener():** Adiciona um ouvinte de eventos a um elemento.
- **removeEventListener():** Remove um ouvinte de eventos de um elemento.
- **Event / MouseEvent / KeyboardEvent:** Interfaces para criar e manipular eventos.

2.3.4. API de Navegação e Histórico:

- **location:** Objeto que fornece informações sobre a URL atual da página.
- **history:** Objeto que fornece métodos para navegar no histórico do navegador.

2.3.5. API de Armazenamento de Dados:

- **localStorage:** Armazena dados persistentes no navegador, que não expiram automaticamente.
- **sessionStorage:** Armazena dados temporários na sessão do navegador, que são removidos quando a sessão é encerrada.

3. DESENVOLVIMENTO DE UM JOGO WEB USANDO CONCEITOS DE POO, PROGRAMAÇÃO FUNCIONAL E DOM COM JAVASCRIPT

Neste ponto, demonstrar-se-á algumas aplicações de conceitos do js, usando a programação orientada a objecto, construindo classes para instância de objectos do jogo, bem como a manipulação de elementos da árvore DOM por meio de chamadas de funções.



Figura 8. Cenário do Jogo (Fonte:Autor)

3.1. Personagens do Jogo e Estrutura de Código

O personagem principal, é um soldado que tem um conjunto de muitas imagens, chamadas de assets, organizados em grupos, que representam acções e cada imagem de um grupo é um certo estado da mesma acção

Personagem Principal e Animações Manuais via JS



Figura 9. Sprites da animação de corrida do player (Fonte: Assets story)

A imagem acima, é um exemplo da animação de correr que foi tratada manualmente, via javascript, onde as imagens foram recortadas em porções iguais, e salvas em uma pasta

específica do projecto, com o nome da animação.

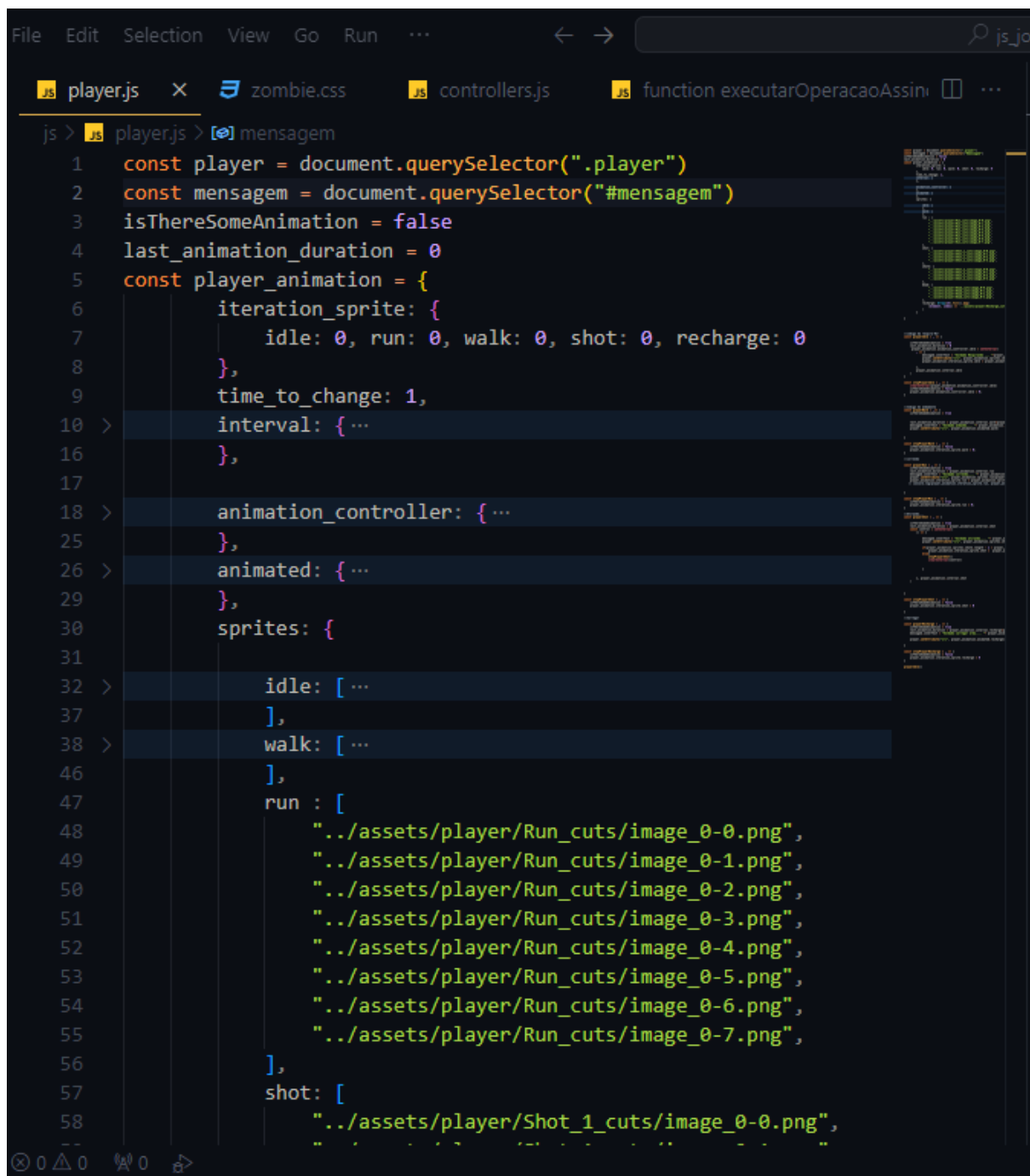


Figura 10. Estrutura de arquivos do projecto (Fonte:Autor)

Código Js do Player e Suas Configurações para Animações

No HTML, foi criado um elemento com a class player que é a tag img do personagem, via DOM foi possível pegar esse elemento e alternar as suas imagens.

A seguir o objecto html player e o objecto js de configuração da animação

A screenshot of a code editor with a dark theme. The editor shows a file named 'player.js' with JavaScript code for configuring player animations. The code includes selectors for 'player' and '#mensagem', sets initial animation state to false, and defines a 'player_animation' object. This object contains an 'iteration_sprite' with states for idle, run, walk, shot, and recharge, each with a value of 0. It also has a 'time_to_change' of 1, an 'interval' object, an 'animation_controller' object, an 'animated' object, and a 'sprites' object. The 'sprites' object has 'idle' and 'walk' arrays, and a 'run' array containing eight image paths for running animation frames. It also has a 'shot' array with one image path for a shot animation frame. The editor interface includes a menu bar (File, Edit, Selection, View, Go, Run), a toolbar with navigation icons, and a tab bar showing 'player.js', 'zombie.css', 'controllers.js', and 'function executarOperacaoAssin...'. The status bar at the bottom shows '0' errors, '0' warnings, and '0' info messages.

```
js > js player.js > [e] mensagem
1  const player = document.querySelector(".player")
2  const mensagem = document.querySelector("#mensagem")
3  isThereSomeAnimation = false
4  last_animation_duration = 0
5  const player_animation = {
6      iteration_sprite: {
7          idle: 0, run: 0, walk: 0, shot: 0, recharge: 0
8      },
9      time_to_change: 1,
10 > interval: { ...
16 > },
17
18 > animation_controller: { ...
25 > },
26 > animated: { ...
29 > },
30 > sprites: {
31
32 >     idle: [ ...
37 >     ],
38 >     walk: [ ...
46 >     ],
47     run : [
48         "../assets/player/Run_cuts/image_0-0.png",
49         "../assets/player/Run_cuts/image_0-1.png",
50         "../assets/player/Run_cuts/image_0-2.png",
51         "../assets/player/Run_cuts/image_0-3.png",
52         "../assets/player/Run_cuts/image_0-4.png",
53         "../assets/player/Run_cuts/image_0-5.png",
54         "../assets/player/Run_cuts/image_0-6.png",
55         "../assets/player/Run_cuts/image_0-7.png",
56     ],
57     shot: [
58         "../assets/player/Shot_1_cuts/image_0-0.png",
59         ..
```

Figura 11. Configuração das animações do player (Fonte:Autor)

Animação Manual do Player Usando Array Function

Para, animar, por exemplo o movimento de corrida, foi feita uma função que percorre o array das imagens de estado da corrida, e para cada iteração foi trocada a imagem do html element.


```
//correndo

const playerRun = _ => {
  isThereSomeAnimation = true
  last_animation_duration = player_animation.interval.run
  mensagem.innerText = "Soldado correndo.... " + player_animation.
  iteration_sprite.run
  player.setAttribute("src", player_animation.sprites.run[player_animation.
  iteration_sprite.run])
  player_animation.iteration_sprite.run = player_animation.sprites.run.
  length - 1 > player_animation.iteration_sprite.run ? player_animation.
  iteration_sprite.run + 1 : 0
  // console.log(player_animation.iteration_sprite.run, player_animation.
  sprites.run.length - 1)
}
```

Figura 12. Função de corrida do player (Fonte:Autor)

Graças ao objecto de configuração o valor do iterador percorre todas imagens possíveis, e em seguida volta à zero, caso estiver no limite, repetindo a animação.

3.2. Animações com Chamadas de Instruções CSS e a Função setInterval

As animações foram feitas, chamando algumas instruções css, graças a manipulação da dom de um dado objecto do documento, como é o caso do movimento dos zombies, e o chão do jogo.

Chamadas de Css

Para o deslocamento horizontal de um certo elemento, como o chão, foi preparado um código de animação css controlado via js, em função do clique no teclado, como é o caso à baixo:

```

JS controllers.js > [E] keyPlayerAnimation > arrowright
1  const craters = document.querySelector(".crater-area")
2
3  craters.classList.add("crater-area-animation")
4  fire_power =1
5
6  const keyPlayerAnimation = {
7    arrowright: _ => {
8      stopPlayerIdle()
9      stopPlayerShot()
0      stopPlayerWalk()
1      playerRun()
2      craters.style.animationPlayState = "running"
3      //craters.style.animationDuration = "15s"
4    },
5  },

```

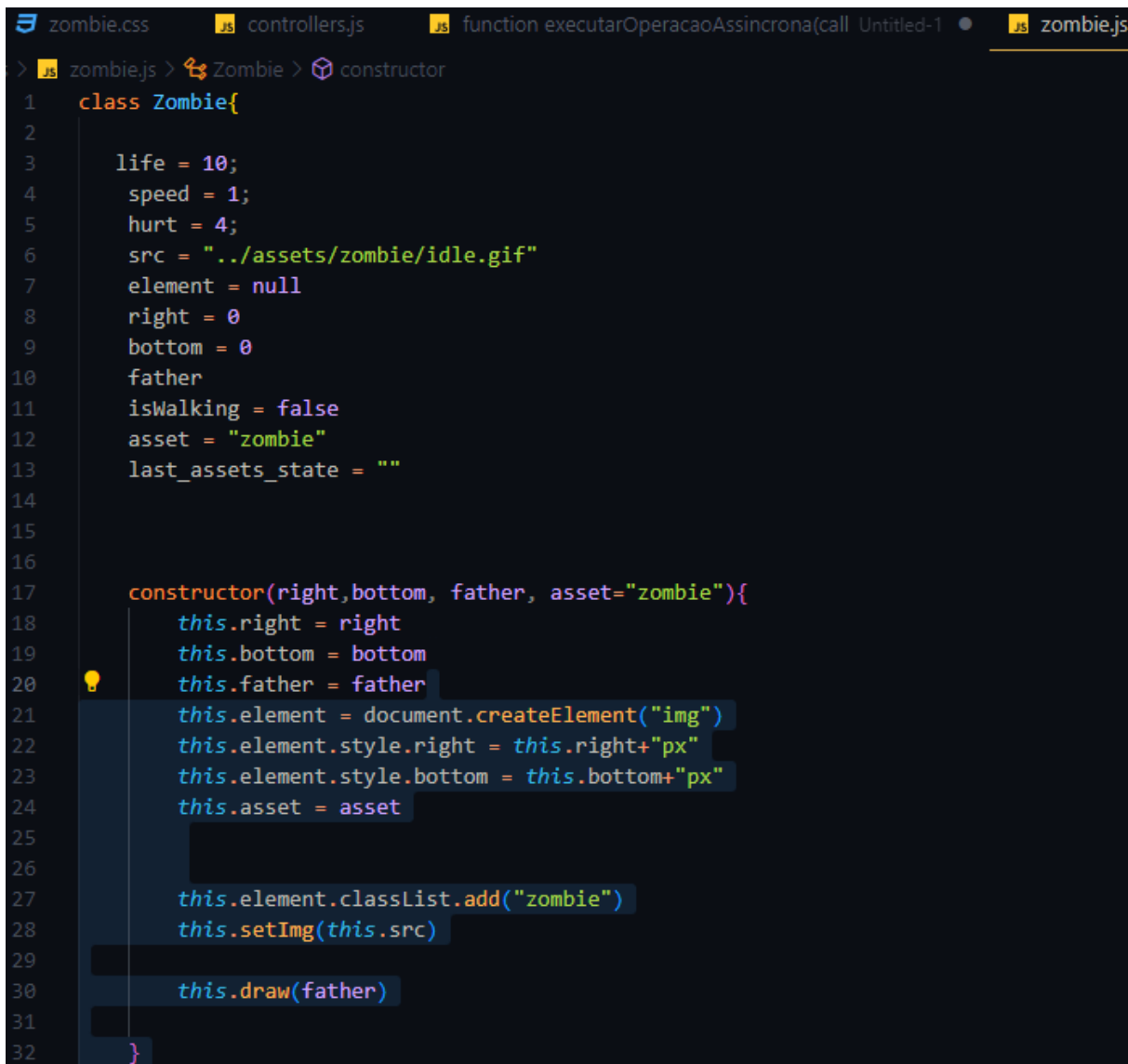
Figura 13. Controle da animação do cenário movimentado (Fonte:Autor)

Tem-se em primeiro, como exemplo, um objecto do tipo `HTMLDivElement`, que é referenciado via selector css, usando o método `querySelector` do objecto `document`, sendo este o container que tem elementos, como o inimigo e as crateras do chão.

O objecto `KeyPlayerAnimation`, contem um conjunto de campos que identificam teclas do teclado, onde o campo `arrowright` é uma função que executa a paragem de todas animações, e chama a animação de corrida do player e também activa a animação css do elemento HTML na seguinte instrução: `craters.style.animationPlayState = "running"`

Classe **Zombie** e Suas Instancializações

A classe `Zombie` foi necessária, pois o sistema precisaria gerar muitos elementos com o mesmo comportamento, o mais simples foi criar uma classe que contenham todos os comportamentos comuns dos objectos.



```

> .js zombie.js > 🐞 Zombie > 📦 constructor
1  class Zombie{
2
3      life = 10;
4      speed = 1;
5      hurt = 4;
6      src = "../assets/zombie/idle.gif"
7      element = null
8      right = 0
9      bottom = 0
10     father
11     isWalking = false
12     asset = "zombie"
13     last_assets_state = ""
14
15
16
17     constructor(right,bottom, father, asset="zombie"){
18         this.right = right
19         this.bottom = bottom
20         this.father = father
21         this.element = document.createElement("img")
22         this.element.style.right = this.right+"px"
23         this.element.style.bottom = this.bottom+"px"
24         this.asset = asset
25
26
27         this.element.classList.add("zombie")
28         this.setImg(this.src)
29
30         this.draw(father)
31
32     }

```

Figura 14. Classe Zombie (Fonte:Autor)

Para além do construtor tem-se também métodos de comportamento padrão dos zombies, como andar, correr, sofrer danos, e morrer.

Instancias em Posições Aleatórias e Referencia via DOM

Para colocar visível os zombies, na classe principal, contem o atributo que faz referencia à um objecto html, na método construtor esse atributo inicializando assim:

Método Construtor do Zombie

```
constructor(right,bottom, father, asset="zombie"){  
    this.right = right  
    this.bottom = bottom  
    this.father = father  
    this.element = document.createElement("img")  
    this.element.style.right = this.right+"px"  
    this.element.style.bottom = this.bottom+"px"  
    this.asset = asset  
  
    this.element.classList.add("zombie")  
    this.setImg(this.src)  
  
    this.draw(father)  
}
```

Figura 15. Construtor da classe Zombie (Fonte:Autor)

Método de Inserção dos Zombies na View

O elemento father, o container dos elementos, o método draw, coloca o elemento criado no construtor como filho do elemento father.

```
draw = function(father){  
    if(father){  
        father.appendChild(this.element)  
    }  
}
```

Figura 16. Método da classe Zombie para desenhar zombie na tela (Fonte:Autor)

Função para Instancia de Array de Zombies

Assim, como elemento pai é visível, o filho também o será por default, para instancia de zombies, usei uma função que gera randomicamente zombies de 5 à 14

```

5  const generatorZombies = _ => {
6
7      total = Math.round(Math.random()*10)
8
9      for( i = 0; i < total + 5; i++ ){
10         x = Math.round(Math.random()*10)*100
11         y = Math.round(Math.random()*10)*50
12
13         zombie_type = Math.random()*5
14         asset = "zombie"
15         if(zombie_type >= 3)
16             asset = "zombie2"
17         zombies[i] = new Zombie(x,200, zombies_area, asset)
18     }
19 }

```

Figura 17. Gerador Aleatório de instâncias da classe *Zombie* (Fonte:Autor)

Um Dos Usos do setInterval

Para gerar a cada unidade de tempo novos zombies, usei o `setInterval` chamando a função acima.

```

22
23  setInterval( _ => {
24      if(zombies.length == 0)
25          generatorZombies()
26  }, 2000)
27

```

Figura 18. Controlador de período de criação de *Zombies* (Fonte:Autor)

Controle de Animação Usando Eventos do Teclado

Criei o ficheiro controller, responsável pela interação entre o user e a aplicação que controla também as animações em função da acção do user, como é o exemplo da animação de recarregar a arma.

```

115
116 window.onkeyup = key => {
117     const pressed = key.key.toLowerCase()
118
119     if(pressed == "p"){
120         stopPlayerRun()
121         stopPlayerShot()
122         stopPlayerIdle()
123         stopPlayerWalk()
124
125         playerRecharge()
126     }
127

```

Figura 19. Controle de evento do Teclado (Fonte:Autor)

Que fecha todas as animações, e em seguida, chama o playerRecharge(), quando a tecla “p” é largada.

3.3. Importante

A precedência de chamdas de codigos js no documento html é importante, como o efeito, algumas dunções declaradas em certos arquivos foram chamados em outros arquivos.

```

51 </html>
52
53
54 <script src="../../js/player.js"></script>
55 <script src="../../js/zombie.js"></script>
56 <script src="../../js/buildWorld.js"></script>
57
58 <script src="../../js/controllers.js" ></script>
59
60

```

Figura 20. Inclusão de códigos js no html (Fonte:Autor)

3.4. Controles do sistema:

w - andar

Arrow right – correr

Enter – Atirar

P – Carregar A arma

Os zombies, morrem depois de precionar a tecla enter 10 vezes.

4. COMPARAÇÃO COM JAVA E C# EM POO E TIPAGEM

JavaScript é uma linguagem de programação orientada a objetos, porém sua abordagem difere um pouco daquelas encontradas em Java e C#. Enquanto Java e C# são fortemente tipados e possuem sistemas de herança de classes mais estruturados, JavaScript é uma linguagem de tipagem dinâmica e suporta herança baseada em protótipos.

Uma das principais vantagens do JavaScript é sua flexibilidade e simplicidade. Ele permite desenvolver rapidamente e é facilmente integrado em ambientes web. No entanto, a tipagem dinâmica pode levar a erros difíceis de depurar em grandes projetos.

4.1. Vantagens do JavaScript em relação ao Java

1. **Facilidade de Aprendizado e Uso:** JavaScript é uma linguagem mais simples e fácil de aprender do que Java, especialmente para iniciantes. Sua sintaxe é mais flexível e sua curva de aprendizado é menos íngreme.
2. **Flexibilidade e Dinamismo:** JavaScript é uma linguagem de tipagem dinâmica, o que significa que as variáveis não precisam ser declaradas com tipos específicos, proporcionando mais flexibilidade e permitindo uma programação mais rápida e dinâmica.
3. **Integração com a Web:** JavaScript é a linguagem padrão para programação do lado do cliente em páginas web. Ele é executado diretamente no navegador do usuário, o que o torna ideal para criar interatividade e dinamismo em sites e aplicações web.
4. **Ecossistema de Desenvolvimento:** JavaScript possui uma vasta gama de bibliotecas e frameworks, como React, Angular e Vue.js, que facilitam o desenvolvimento de aplicações web modernas e escaláveis.

5. **Execução Assíncrona:** JavaScript possui suporte nativo para programação assíncrona, o que é fundamental para lidar com operações de entrada/saída (I/O) e eventos em aplicações web.

4.2. Desvantagens do JavaScript em relação ao Java

1. **Gerenciamento de Memória:** JavaScript é uma linguagem de tipagem dinâmica e gerenciamento de memória automático, o que pode levar a vazamentos de memória e problemas de desempenho em aplicações mais complexas.
2. **Segurança:** Como JavaScript é executado no lado do cliente, ele pode ser vulnerável a ataques de segurança, como injeção de código e cross-site scripting (XSS), se não for devidamente protegido.
3. **Ecossistema Fragmentado:** Devido à natureza aberta e em constante evolução do ecossistema JavaScript, pode ser difícil acompanhar as mudanças e escolher as melhores ferramentas e bibliotecas para um projeto específico.
4. **Performance:** Em comparação com linguagens como Java, JavaScript pode ter desempenho inferior em determinadas situações, especialmente em aplicações que exigem processamento intensivo de CPU.
5. **Escopo Limitado:** JavaScript é principalmente uma linguagem de programação do lado do cliente, o que significa que seu escopo de aplicação é limitado a ambientes de navegador e, mais recentemente, a ambientes de servidor com o Node.js. Isso pode ser uma limitação em comparação com Java, que é usado em uma variedade de contextos, incluindo aplicações de desktop, móveis e empresariais.

CONCLUSÃO

JavaScript é uma linguagem poderosa e versátil, amplamente utilizada no desenvolvimento web e mobile. Sua evolução contínua e sua vasta comunidade de desenvolvedores contribuem para sua relevância contínua. Embora possa não ser a escolha ideal para todos os tipos de projetos, suas vantagens em termos de flexibilidade e integração com a web o tornam uma ferramenta valiosa no arsenal de qualquer desenvolvedor.