Para executar, descompacte e abra main.html

---

https://github.com/nluizsoliveira/Pokemon-Like-Digimon-Game

Ferramentas utilizadas:

- HTML
- CSS
- JavaScript
- **Typescript**

Javascript originalmente não é tipado ou possui suporte completo à orientação a objetos (É possível realizar todos estes, mas de maneira não usual e pouco intuitiva). Também não é compilado, o que dificulta a depuração e escalabilidade.

Typescript é um wrapper/superconjunto de Javascript desenvolvido pela Microsoft. Wrapper na medida em que é totalmente compatível com Javascript (É possível escrever escrever JS puro em um programa TS), mas, ainda assim, introduz conceitos **opcionais** como **tipagem forte, genéricos, classes, atributos privados.**. e, principalmente, um processo obrigatório de **compilação**, o que combinado com a forte compatibilidade com o editor VsCode, também da Microsoft, facilita a escrita e principalmente depuração.

A saída da compilação de um programa .ts é um programa .js. Por isso, muitas pessoas chamam o processo de transpilação. No final das contas, **é o código .js que possui compatibilidade com o navegador**. O código main.js da pasta não foi produzido por mim, e sim pelo transpilador. **Typescript é apenas uma ferramenta que dá robustez e facilita a produção de programas em javascript.**

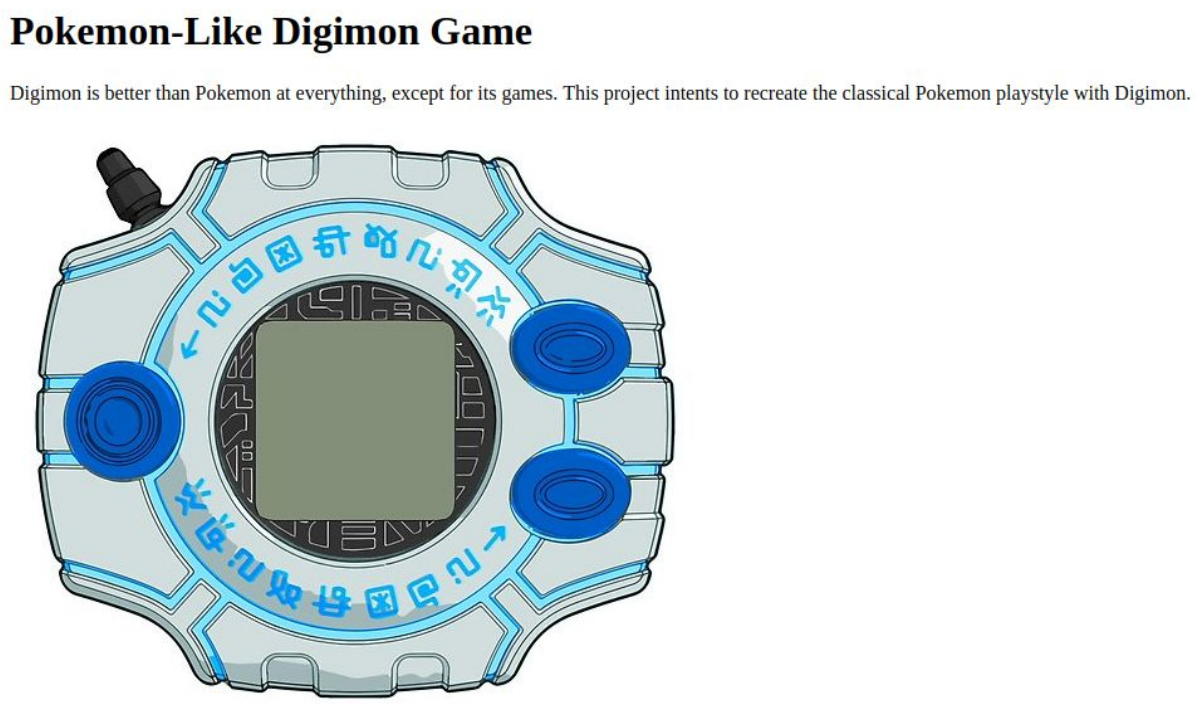
O projeto:

Trata-se de um jogo com personagens do universo Digimon, mas que simula a jogabilidade clássica dos jogos da franquia Digimon. Com exceção das imagens e gifs, **tudo foi manualmente produzido em HTML/CSS (Telas, botões, diagramação) sem auxílio de biblioteca externa.** Além de POO como objetivo de trabalho, houve como objetivo pessoal a tentativa de em **desenvolvimento web e de jogos**, áreas em que não possuía conhecimento prévio. Durante a execução, o paradigma de **programação orientada a eventos** se mostrou uma dificuldade não esperada, mas que agregou muito em termos de conhecimento.

Para executar, descompacte e abra main.html

A tarefa de diagramação se mostrou peculiarmente trabalhosa. O que é simples em editores de texto como o word/docs precisa ser manualmente configurado tratando-se de páginas web (Ao menos, sem utilizar frameworks/bibliotecas de desenvolvimento frontend).

A tela inicial tem o seguinte design com seu HTML cru.



Para o ajuste do título, por exemplo, é necessário o seguinte código em css:

```
#titulo  {
   text-align: center;
   font-family: 'Raleway', sans-serif;
   font-size: 23px;
   font-weight: 800;
   border-bottom: 3px solid #e46124;
   text-align: center;
   border-width: 2px;
   width: 40%;
   align-content: center;
   float: center;
   margin-left: auto;
   margin-right: auto;
   width: 30%;  }
```

Para executar, descompacte e abra main.html

Após os ajustes necessários, as telas ganham um visual mais atrativo:

Ao total, são 9 telas:

- 1- Página inicial com Digivice
- 2- Transição de boas vindas com gif
- 3- Tela de escolha do Digiovo
- 4- Tela de Chocamento do Digiovo
- 5- Tela que mostra o novo Digimon
- 6- Tela que indica que o Digimon foi atacado
- 7 - Tela da primeira Batalha
- 8 - Tela da segunda Batalha
- 9 - Tela da terceira Batalha

Sendo as últimas 5 últimas ajustáveis às decisões do jogador, e as 3 geradas por uma função atualiza_tela() que dinamicamente aloca os elementos de acordo com o estágio atual do digimon

O jogo permite jogar com 9 diferentes personagens. Escolhe-se entre 3 “digiovos” iniciais, e cada digimon nascido deste ovo possui 3 estágios (Level 0, 1 e 2, ou Base, Rookie e Mega). Evoluir, além de melhorar aparência e atributos, desbloqueia golpes novos.

Para executar, descompacte e abra main.html









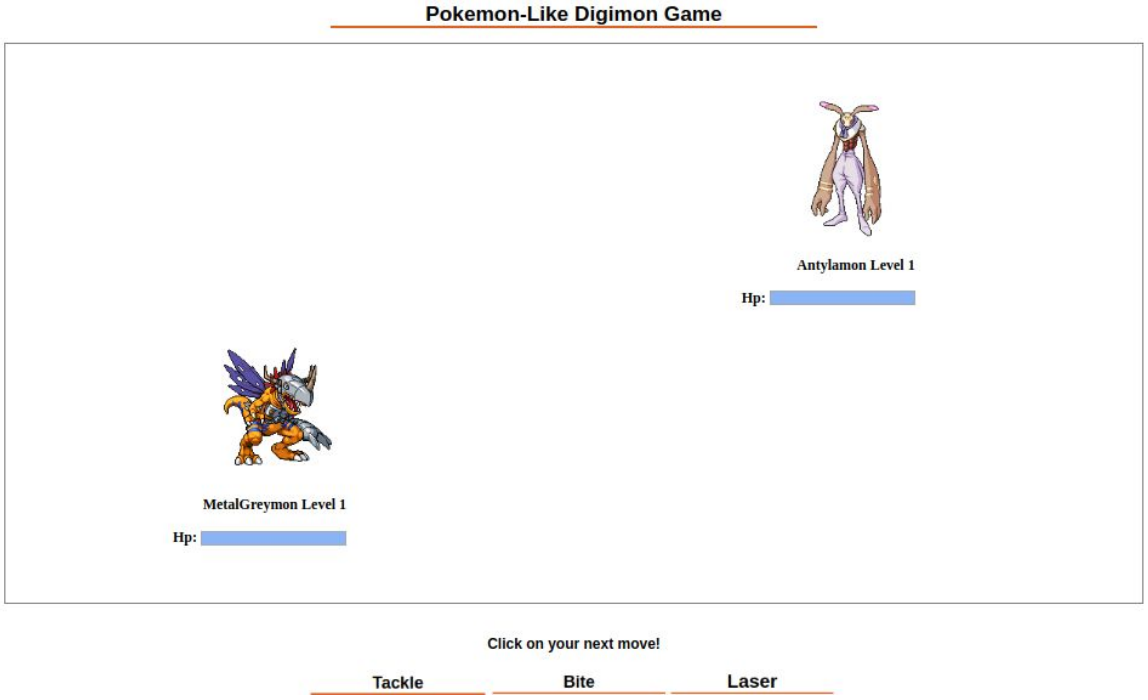






Para executar, descompacte e abra main.html

O sistema de batalhas é simples. O digimon mais rápido ataca primeiro, o digimon inimigo utiliza um golpe aleatório, e golpes desbloqueados por evolução são mais fortes que os golpes já disponíveis. Neste sentido, já está pre-implementado um sistema de stamina, que limite o uso de golpes fortes e force o jogador a tomar decisões estratégicas. Mas para que este funcione, é necessário também implementar funçoẽs de “auto_cura” e “renegera_stamina”, o que pode ser feito em etapas posteriores do projeto.



Não foi desenvolvida uma tela de endgame, possibilidade de ter mais de um digimon, ou de encontrar oponentes diferente. O jogo acaba quando se derrota o último digimon nível Mega.

POO no projeto

A classe Rookie estende Digimon, e a classe Mega, Rookie.

Digimon Base **Roookie** **Mega**



```
class Digimon {
   private choice: number;
   private level: number;
   private hp: number;
   private atk: number;
   private speed: number;
   private current_hp: number;
   private current_stamina : number;
   private stamina: number;
   private names: string[];
   private digi_img_urls: string[];
   private name: string;
   private url: string;
}
```

A implementação da escolha do digimon inicial foi feita em cima da variável choice. Com choice, é possível acessar posições de names[] e digi_img_url[] através de cálculos, sendo, a partir disto, settar o nome e url de imagem individual do digimon (Isto é feito no construtor).

Falando-se de métodos, fora os getters e setters, digimons podem **atacar**, **subir de nível, atacar aleatoriamente** e utilizar **tackle.**

Inicialmente, digimons podem utilizar apenas o método tackle()

Para executar, descompacte e abra main.html

Evoluir é na verdade deixar de ser um digimon normal, ser criado um new rookie ou mega e ter suas informações prévias guardadas neste novo digimon. Evoluir desbloqueia golpes novos e atualiza as funções de ataque e random_attack, ajustando-as às possibilidades de utilizar bite() ou laser(). **Pela necessidade de se instanciar novos digimon, evolve() não é método de digimon, e sim de página!**

Tanto attack(Enemy_Digimon: Digimon), quanto bite(Enemy_Digimon: Digimon) e random_attack(Enemy_DIgimon: Digimon) **não são apenas herdados, como também polimórficos,** afinal pode-se atacar tanto bases, quanto Rookies, quanto Megas. Getters e setters também são herdados. O que não significa que não recebam override. Random_attack e attack devem ser ajustadas às novas possibilidades abertas ao evoluir.

A classe **page** é quem de fato controla o fluxo do programa. A maioria de seus atributos são do tipo **`(HTMLElement | null)`** No contexto de ts, indica que são possíveis elementos html, mas que não necessariamente existem na página ainda. O real assignment é feito em

```
this.description = document.getElementById("description");
```

É importante notar que **pode não haver elemento com tal id** no HTML da pagina, abrindo margem a elementos nulos. Por isso, na utilização destes elementos, utiliza-se o **"!"** para explicitar que o elemento pode ser nulo.

```
this.description!.innerText = "Welcome to DigiWorld! “
```

Foi a opção escolhida em lugar de

```
if (this.description != null ){
        this.description.innerText = "Welcome to DigiWorld! “
}
```

Que também compila, mas é muito verbosa. **Page() possui diversos métodos polimórficos em relação a Digimon**. Dentre eles, destaca-se **atualiza_arena**, que gerencia toda a modificação do corpo HTML da arena quando um digimon ataca, sofre dano, evolui, etc.

Eventos e fluxo de informação

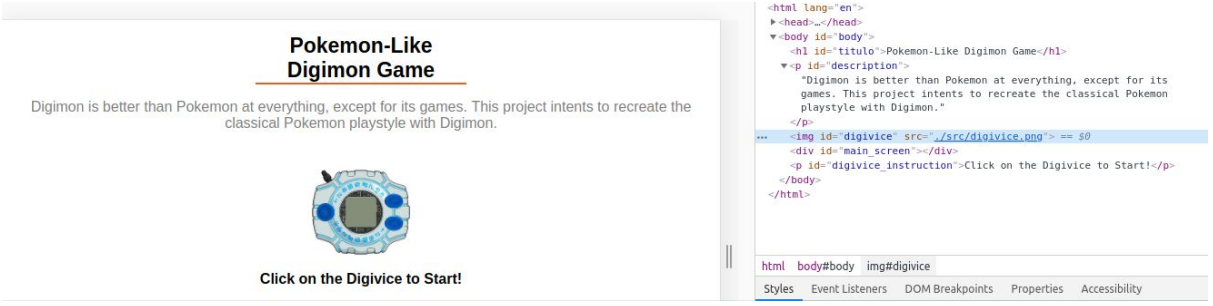
É comum em c o fluxo do programa ser majoritariamente ditado na função **main**, restando às funções apenas executar operações, chamando outras funções caso necessário. No projeto, devido à maneira como se comportam os **eventos**, isto se tornaria **impossível**.

```
window.onload = function main() {

    let pagina = new page();
    let call_event_one = ()=>{pagina.event_one()};
    pagina.digivice!.addEventListener("click", call_event_one);

}
```

Ao criar a página, em page(), já é chamado o construtor, que detecta que há uma imagem com id = “digivice” na tela e a associa ao atributo digivice de pagina, afinal, no HTML da primeira tela há de fato uma <img id = “digivice>.



<h1 id="titulo">Pokemon-Like Digimon Game</h1>

Digimon is better than Pokemon at everything, except for its games. This project intents to recreate the classical Pokemon playstyle with Digimon.

<img id="digivice" src="./src/digivice.png">

Click on the Digivice to Start!

É o conteúdo de Body no início do programa.

O mesmo não ocorre com o resto das imagens e botões gerados ao resto do programa. **Event listeners só podem ser associados a elementos não nulos.**

Não há <img> com id de digiovos nesta página para que seus event clickers sejam atribuídos. Isto só ocorre na próxima página, **após a chamada do evento inserido no digivice que faz a mudança do DOM HTML, onde é inserida a div main_screen que comporta os ovos.**


```
▼<body id="body">
    <h1 id="titulo">Pokemon-Like Digimon Game</h1>
  ▼<p id="description">
      "Digimon is better than Pokemon at everything, except for its
      games. This project intents to recreate the classical Pokemon
      playstyle with Digimon."
    </p>
    <img id="digivice" src="./src/hello.gif" style="max-width: 0px;">
  ▼<div id="main_screen">
      <img id="egg1" src="./src/choice10.png" style="max-width: 10%;">
      <img id="egg2" src="./src/choice20.png" style="max-width: 10%;">
      <img id="egg3" src="./src/choice30.png" style="max-width: 10%;">
    </div>
  ▶<p id="digivice_instruction">…</p>
  </body>
```

Existem duas soluções para tal: ***Carregar o HTML com elementos sem conteúdo, mas com ids específicas, sobrecarregando o HTML, ou deixar a chamada de eventos posteriores dentro de eventos anteriores, deixando o fluxo do programa recursivo.***

A primeira opção parece tentadora a princípio. Mas, na prática, **deixou o código muito mais confuso, pois cada elemento do HTML possui sua formatação específica no CSS, e elementos invisíveis, mesmo sem conteúdo, geram modificações na diagramação da página.**

Ou seja, frequentemente no programa seria necessário alterar configurações de design do **CSS, deixando-o sem utilidade, já que todo atributo ali escrito seria apagado e então sobrescrevido**.

Assim, decidi por tomar a segunda opção. Não tenho experiência com programação orientada a eventos, desenvolvimento de jogos ou desenvolvimento web, **posso ter tomado a decisão errada.** Tal problemática, bem como as decorrentes mudanças no projeto do fluxo do programa, foram o que me fizeram gastar mais tempo do que eu aloquei à execução do projeto quando pensei em fazer o jogo.