**Criando seu primeiro projeto PhaserJS**

Neste tutorial darei uma noção de como funciona o Phaser JS. Precisamente como funciona a adição de textos e imagens.

Se você não sabe o que é o Phaser JS, recomendo dar uma pesquisada: **<https://phaser.io>** . Mas só para dar uma luz, o Phaser é um framework JavaScript para desenvolvimento de jogos 2D com HTML5 para desktop e dispositivos móveis.

Em linhas gerais, é interessante haver uma familiaridade com a linguagem JavaScript para melhor compreensão.

Sem mais delonga, comecemos...

* **Criando seu projeto**

Se você já tem um projeto Phaser, pode pular para a próxima etapa.

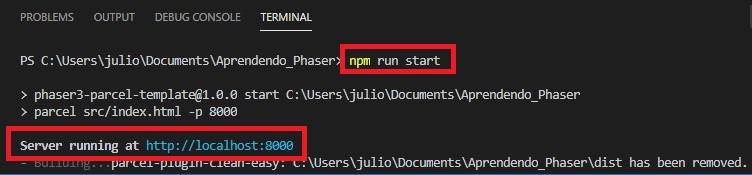
A primeira coisa que recomendo é baixar esse arquivo disponibilizado neste link: **<https://github.com/ourcade/phaser3-parcel-template>** .

É um template que facilita o uso do Phaser com a metodologia moderna para programar com JavaScript. Baseado nele você vai desenvolver o seu jogo. Existem outras maneira, mas essa é uma das mais simples. Recomendo acompanhar esse tutorial junto à este arquivo. Abra-o em um editor de sua preferência. Abra o terminal/prompt de comando e localize a pasta com o comando ***cd “caminho\_da\_pasta”*** (sem aspas) e dentro dela utilize o comandos:

***npm install*** *<-- instalar dependências necessárias*

***npm run start*** *<-- executa servidor para rodar no navegador*

No seu navegador digite o endereço **localhost:8000**. Referente a porta, você identifica após executar o segundo comando acima.



A partir de agora, toda vez que você atualizar e salvar seu código ele será compilado e executado automaticamente no navegador.

Vale lembrar que independente do template que você usar como base ou se não usar nenhum, não interfere na lógica do código do PhaserJS.

Caso apareça isso:

Significa que funcionou com sucesso!

Basicamente, o que ocorre é que dentro da pasta principal existe a pasta **src**. Lá contém 2 arquivos: **index.html** e **main.js** e uma pasta **scenes** que contém o arquivo **HelloWorldScene.js**. Ao chamar o **localhost:8000**, o **index.html,** que é o arquivo base, na sua estrutura chama o arquivo **main.js**

**...**

**<body>**

**<script src="main.js"></script>**

**</body>**

**...**

No **main.js**, configura-se como será apresentada a tela (resolução de tela, renderização de game etc). Em seguida ele chama o arquivo **HelloWorldScene.js**  que é a primeira “cena” de fato do jogo, ou seja, a primeira instância visível para o usuário, que nesse caso será como a imagem acima. (Deem uma olhada nos arquivos para melhor compreensão).

Mas como fazer aparecer estas imagens?

* **Carregar e instanciar assets**

Quando falamos de assets, falamos sobre arquivos de multimídia: imagens, áudios e até vídeos que são carregados e criados em cenas para dar ou não interação com o usuário durante aquela cena.

Para carregar e mostrá-las é bastante simples. Mas explicarei algo antes. O que é uma cena?

Entenda aqui que uma cena, de maneira mais grosseira, é, em Phaser, uma superclasse (**Phaser.Scene**) que possui dois métodos bases:

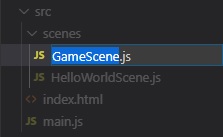
O método **preload** e o método **create.** Existe um terceiro método que é o **update,** mas que não abordaremos aqui.

Todas as classes que são cenas do jogo são filhas desta classe Phaser.Scene: **class Nome\_da\_classe extends Phaser.Scene{...}**.

No **preload**, como o nome indica é o momento onde você carrega o arquivo de mídia, deixando-o disponível para o uso. Não necessariamente ele vai aparecer. Mas estará apto para isso.

Vamos fazer passo a passo:

Dentro da dentro da pasta **src** crie um novo arquivo chamado **GameScene.js**:



Após isso dentro deste arquivo, copie este código:

//Não é imagem pode copiar o texto

import Phaser from 'phaser'

export default class GameScene extends Phaser.Scene{

    constructor(){

        super('game-scene')

}

preload(){

}

create(){

    }

}

Nesse momento você criou uma nova cena mas a mesma ainda não é visível, pois não foi inicializada.

Para isso basta você ir no **main.js.** Importe o arquivo que você criou para dentro do **main.js**.

import GameScene from './scenes/GameScene'

Dentro da constante **config** altere as cenas que vão ser chamadas:

De:

scene: [HelloWorldScene]

Para:

scene: [GameScene]

Repare que ficou a tela toda preta ao atualizar, pois você não carregou nada no arquivo GameScene.js. :)

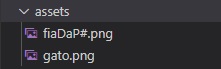
Crie uma pasta com nome **public** destro deste template, no mesmo nível de src. Será utilizada para os assets.

img-4

Após isso crie uma pasta com nome **assets** dentro da pasta **public** já criada:

img-4.5

Pegue algum arquivo de imagem e coloque dentro desta pasta. Dê preferência pelo uso de arquivos **.png** por uma questão de qualidade de imagem.



As imagens estarão disponibilizadas no repositório deste tutorial.

Feito isso, agora voltemos para o **GameScene.js**.

No método preload, digite:

this.load.image('cachorro', 'assets/fiaDaPu.png')

this -> referente ao escopo maior que é a própria classe (ou cena), nesse caso.

load -> método para disponibilizar os arquivos.

image -> método identificador do tipo de objeto a ser carregado ou adicionado, nesse caso uma imagem.

Dentro de **image**, passa-se 2 parâmetros, o primeiro é uma chave única para identificar a imagem a ser carregada e o segundo é o caminho onde está a imagem.

Ok, mas ainda não aparece a imagem. Pois, como eu havia dito antes ela só fica disponibilizada, mas não necessariamente inicializada. Para isso, vamos até o outro método, o **create**.

O **create** serve para criar as imagens, deixá-las visíveis, interativas ao usuário, etc. É aqui que preferencialmente se aplica todo o comportamento dos objetos instanciados, desde movimentações, eventos, até controle deles.

Digite o seguinte comando dentro do **create**:

this.add.image(400, 300, 'cachorro')

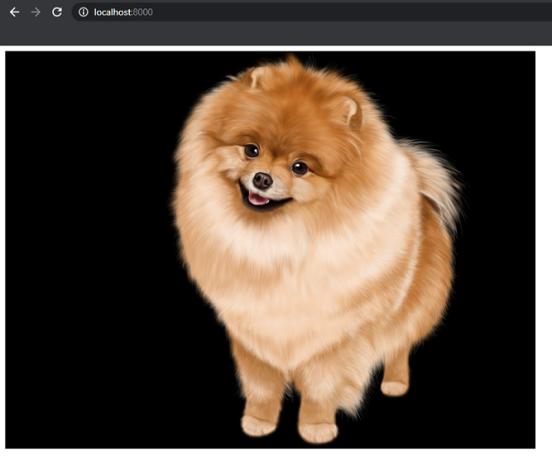
this -> referente ao escopo maior que é a própria classe (ou cena), nesse caso.

add-> método para adicionar objeto à cena

image -> método identificador do tipo de objeto a ser carregado ou adicionado, nesse caso uma imagem.

Essencialmente , são 3 parâmetro em **image**, nesse contexto: os dois primeiros referem-se as coordenadas: a primeira, o ponto das abcissas (ou eixo do x) e a segundo, as ordenadas (ou eixo do y). O terceiro parâmetro é a chave criada anteriormente no **preload**.

Se aparecer alguma imagem:



Dimensão da Tela(parte preta):

**800** de largura x **600** de altura.

Definida na const config em **main.js**

Então funcionou com sucesso!

**OBS**: é imprescindível a criação da pasta **public** entre seus arquivoscaso a mesma não apareça. Caso contrário não será possível carregar os arquivos.

Vamos outra imagem, mas antes diminuiremos e reposicionaremos esta imagem.

Para diminuir a imagem sem precisar redimensioná-la originalmente basta usar o método **setScale**, desta forma.

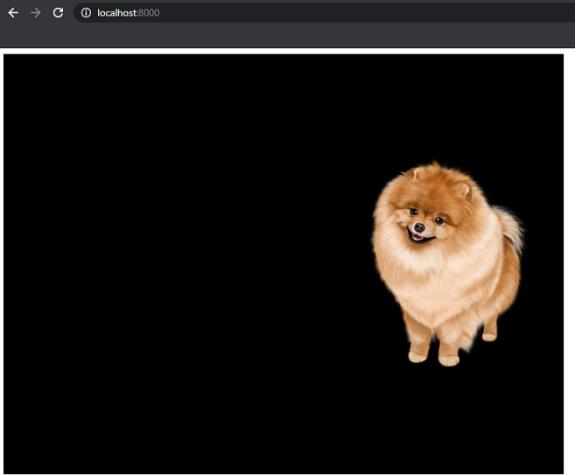
this.add.image(400, 300, 'cachorro').setScale(0.5)

Por padrão o tamanho em escala é 1, usando 0.5 então a imagem terá metade das suas dimensões.

Para reposicionar imagem basta alterar os valores dos primeiros parâmetros

this.add.image(600, 300, 'cachorro').setScale(0.5)

Então:



Agora adicionamos a outra imagem:

**preload**:

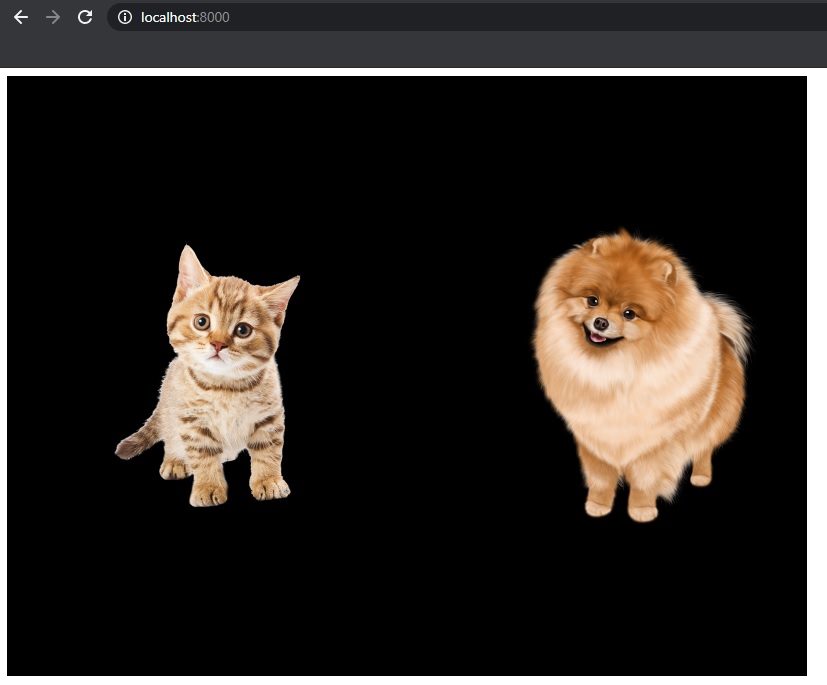
...

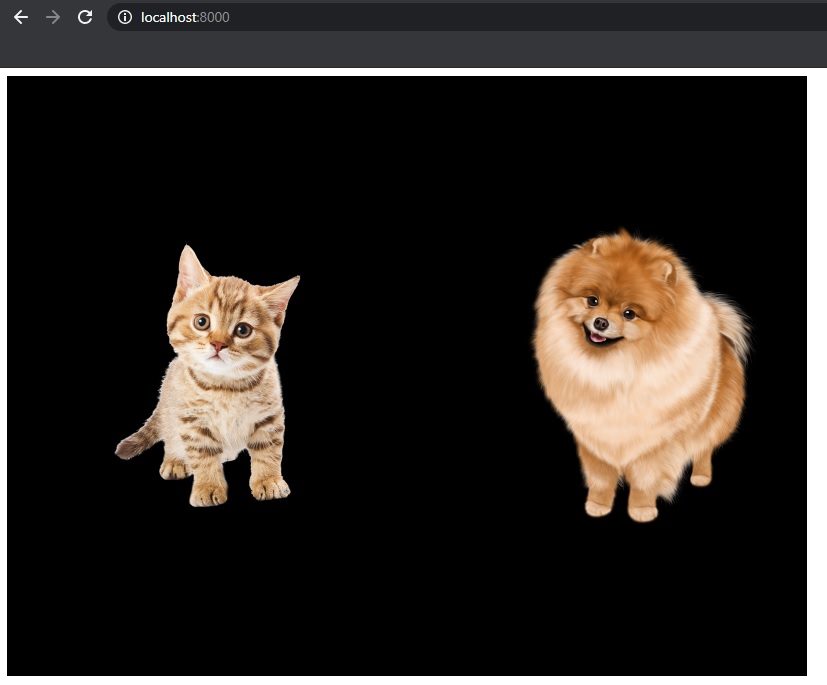
this.load.image('gato', 'assets/gato.png')

**create:**

...

this.add.image(200, 300, 'gato').setScale(0.5)

Então, agora o cachorrinho não está mas só :)

**OBS:** O ponto de ancoragem da imagem, por padrão, é definido no meio da imagem. Vamos usar, a imagem do cachorrinho que está nesse momento no ponto **600, 300** para compreender melhor.

.

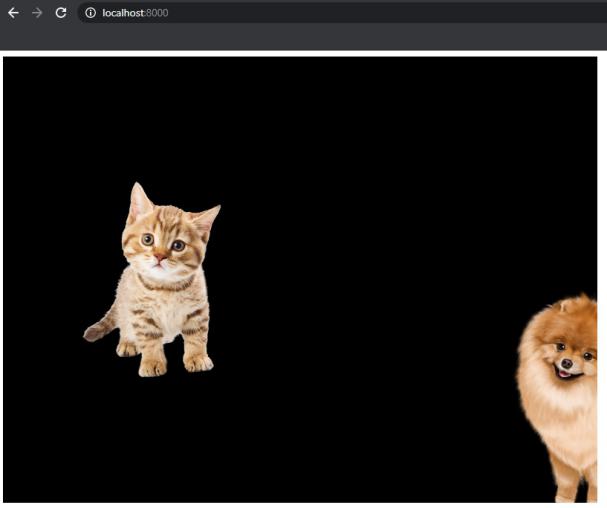
Considere o ponto verde como o ponto central da imagem tanto no eixo x como no y. Esse ponto é quem estaria na coordenada 600, 300. Ou seja, o ponto mais centralizado da imagem é quem está sendo reposicionado. Este então, é o que chamamos de ponto de ancoragem.

Não é muito comum alterar o ponto de ancoragem, mas caso seja necessário você pode utilizar o método **setOrigin**.

Exemplo:

this.add.image(600, 300, 'cachorro').setScale(0.5).setOrigin(0)

Aí...



.

A imagem não foi reposicionada, somente o ponto de ancoragem foi redefinido. Por padrão o valor deste é **setOrigin(0.5)**. Colocando **setOrigin(0)**, ele vai para o ponto inicial de x e y. E se por exemplo, **setOrigin(1)** vai para o final da imagem tanto em x como y.

* **Adicionar texto**

Adicionar texto também é bastante simples. Basta utilizar, no método **create** este código:

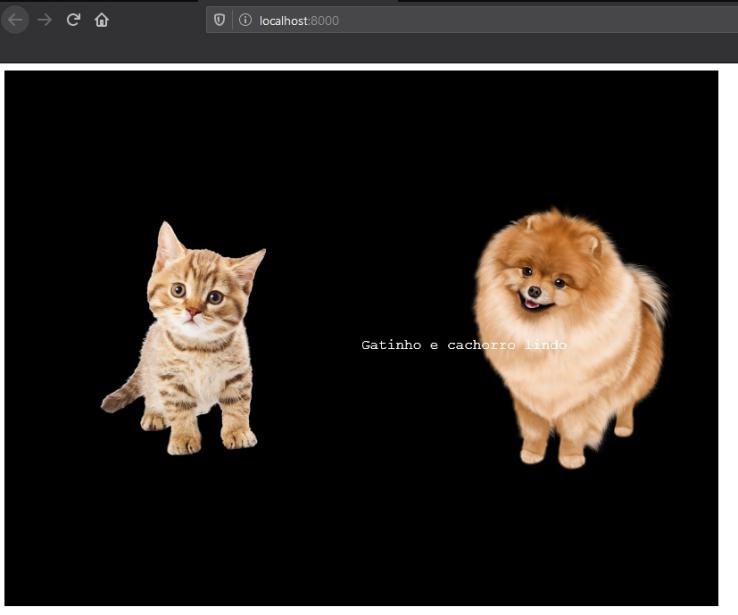
this.add.text(400, 300, 'Gatinho e cachorro lindo')

this -> referente ao escopo maior que é a própria classe (ou cena), nesse caso.

add -> método para adicionar objeto à cena

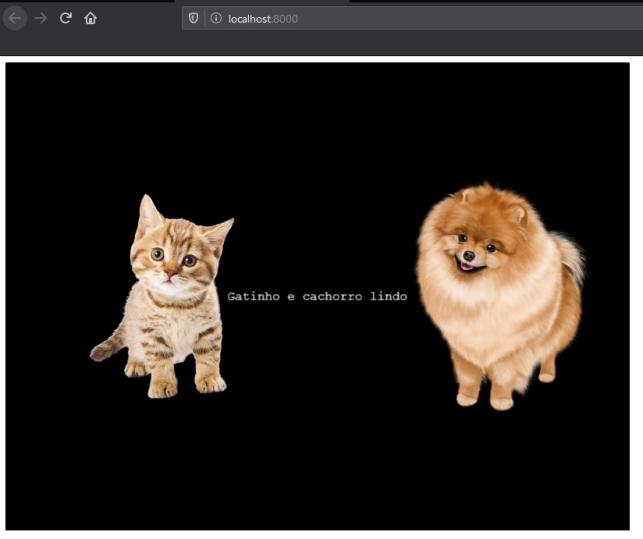
text -> método identificador do tipo de objeto a ser carregado ou adicionado, nesse caso um texto.

E isso é reproduzido em tela.



Perceba que o texto foi colocado numa posição considerada central dentro da tela. Mas o mesmo não fica centralizado. Isso porque, diferente do objeto tipo imagem, os textos possui ponto de ancoragem de acordo com o seu alinhamento padrão. Ou seja, o ponto de ancoragem vai do início do texto. Para resolver isso basta alterá-lo para 0.5:

this.add.text(400, 300, 'Gatinho e cachorro lindo').setOrigin(0.5)



O resultado é o texto alinhado

no centro da tela.

Vamos alterar a cor e fonte do texto:

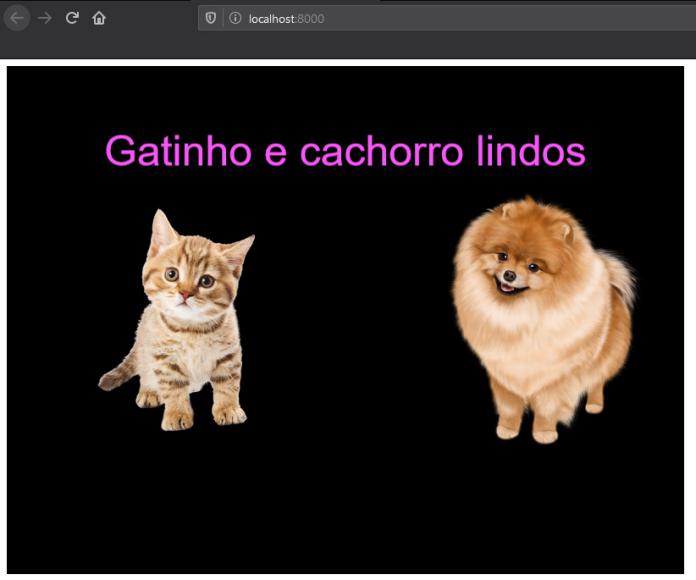
this.add.text(400, 100, 'Gatinho e cachorro lindos',{font: '50px arial', fill: '#ff55ff'}).setOrigin(0.5)

Nesse caso para formatar o texto passamos como um quarto parâmetro uma espécie de código “CSS” composto, nesse caso por 2 campos:

font: -> para formatar tamanho e estilo do texto

fill: -> para adicionar uma cor no texto, com padrão hexadecimal

E aí...



Então é isso por hoje :)

* **Considerações finais**

Bom, o Phaser é uma ferramenta bastante simples de utilizar. Esse tutorial foi só uma maneira de iniciar seu projeto com o pé direito. Com metodologia mais moderna de JavaScript e de maneira simplificada!

Segue aqui os links para utilização da ferramenta que certamente poderão lhe ajudar no seu desenvolvimento de games.

Exemplos com Phaser:

<https://phaser.io/examples>

Notas do Phaser 3: <https://rexrainbow.github.io/phaser3-rex-notes/docs/site/>

Fazendo seu primeiro jogo com Phaser:

<https://blog.ourcade.co/posts/2020/make-first-phaser-3-game-modern-javascript-part1/>

<https://blog.ourcade.co/posts/2020/make-first-phaser-3-game-modern-javascript-part2/>

<https://blog.ourcade.co/posts/2020/make-first-phaser-3-game-modern-javascript-part3/>

<https://blog.ourcade.co/posts/2020/make-first-phaser-3-game-modern-javascript-part4/>

<https://blog.ourcade.co/posts/2020/make-first-phaser-3-game-modern-javascript-part5/>

**OBRIGADO!**