Jogo tipo Breakout usando o Phaser (passo-a-passo)

Fase 1

Vale lembrar que como o Phaser usa HTML, CSS e JavaSCript para a criação de seus jogos, então podemos fazer ele basicamente em qualquer sistema operacional.

Uma grande vantagem de trabalhar com um framework como o Phaser é sua popularidade. Se fizer uma busca por algo do Phaser encontrará muita coisa. Isso é importante, especialmente para quem está começando.

Apesar de não contar ainda com uma interface gráfica eu gosto muito do phaser. Ele tem um framework cujas bibliotecas são muito organizadas e facilitam bastante a criação de jogos. Para isso veja quantas linhas de código tem alguns exemplos. Realmente facilita o trabalho de criação de jogos.

Aqui estarei traduzindo, resumindo e comentando este ótimo e detalhado exemplo.

No site da MDN (Mozilla Developer Network)
https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser

Estarei traduzindo e mostrando cada uma das 16 fases do mesmo.

O autor afirma que quando você concluir as 16 fases estará apto a criar seus próprios jogos simples usando o Phaser.
Ele também oferece o código fonte para download.

Aqui ele entrega todo o pacote, inclusive os fontes do jogo separado por fases, na pasta demos:
https://github.com/end3r/Gamedev-Phaser-Content-Kit

A pasta demos contem duas pastas: img e js e os 16 arquivos .html de cada fase além também do index.html. Podem ser vistos online:
https://github.com/end3r/Gamedev-Phaser-Content-Kit/tree/gh-pages/demos

Criando a Fase 1 - Inicialização do framework

Original em inglês -
https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser/Initialize_the_framework

Quando trabalhando com jogos mobile é recomendado usar um servidor web. O Xampp para windows é meu favorito, mas existem outros como o Wampp, o EasyPHP, etc. Este jogo funciona bem sem um servidor web, mas fica o aviso: sempre que encontrar problemas ao executar um jogo, use um servidor web.

Criar a estrutura de pastas abaixo:

\breakout
    \img
    \js
    fase1.html

Faça o download do framework phaser.min.js daqui
https://github.com/photonstorm/phaser/releases/download/v2.4.6/phaser.min.js
Ou daqui:
https://github.com/end3r/Gamedev-Phaser-Content-Kit/blob/gh-pages/demos/js/phaser.2.4.2.min.js
Veja, que existem diferenças nas versões e isso pode dar problema. Idealmente use a versão que o autor do jogo tá usando.
Copie este arquivo para a pasta fase1/js

Copie os arquivos da pasta img para nossa pasta fase1\img
https://github.com/end3r/Gamedev-Phaser-Content-Kit/tree/gh-pages/demos/img

Cria o arquivo breakout\fase1.html, com a seguinte estrutura:

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8" />
   <title>Breakout em Phaser - fase 1: Inicialização do framework</title>
   <style>* { padding: 0; margin: 0; }</style>
   <script src="js/phaser.2.4.2.min.js"></script>
</head>
<body>
<script>
var game = new Phaser.Game(480, 320, Phaser.AUTO, null, {preload: preload, create: create,
update: update});

function preload() {}
function create() {}
function update() {}
</script>
</body>
</html>
```

Alerta: aqui comigo só funcionou bem com o Firefox. O Chrome não mostrou o jogo, apenas a área do canvas. Mas quando copio a pasta para o servidor web (c:\xampp\htdocs\breakout) então funciona também no Chrome. Isso parece indicar que o Chrome tem mais preocupações com a segurança, restringindo mais.

Quando estiver pronto, abra o fase1.html no Firefox ou outro navegador.

Comentando o código do fase1.html
Esta primeira fase apenas inicializa o framework, com algumas informações para começarmos a trabalhar.

Comecemos com a primeira linha: <!DOCTYPE html>. Ela diz ao navegador que nosso código é HTML5.
O <stile> diz ao navegador que não guarde margens nem padding no canvas.
A linha <script> indica onde encontrar o código do framework Phaser.
O segundo <script> é que realmente traz o código do jogo.
A linha de definição da variável game = new Phaser.Game ... é a mais importante aqui. Aqui damos vida ao Phaser, criando uma instância do objeto Phaser. Game e atribuindo esta instância à variavel 'game'. O canvas será gerado automaticamente pelo framework com 480 pixels de largura por 320 pixels de altura. Chamar esta variável de 'game' não é obrigatório mas é uma boa prática, comumente encontrada nos exemplos.

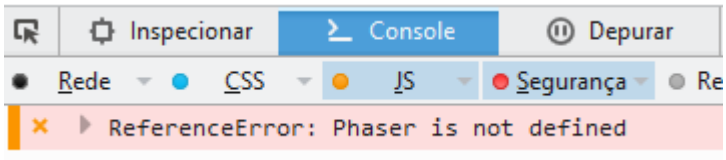
O terceiro parâmetro é Phaser.AUTO e pode ser também Phaser.CANVAS ou Phaser.WEBGL. É o tipo de renderização do canvas e Phaser.AUTO é o recomendado.
O parâmetro null é o ID do elemento DOM do canvas, que é opcional.
Finalmente temos um objeto contendo quatro referências para funções exxenciais do Phaser: preload(), create() e update().

preload - faz uma carga antecipada dos assets
create - executa uma única vez após tudo ser carregado e pronto
update - executada a cada frame

Executando
Nesta fase praticamente não veremos nada no navegador. Apenas uma área escura no canto, a limitação da área do canvas, com 480 x 320 que indicamos. Nesta região é onde nosso jogo será completamente renderizado e somente aí.

Debug
Experimente comentar a linha que faz o include do phaser.min.js e abra novamente no navegador.

Após abrir no navegador clique na tela com obotão direito - Inspecionar elemento, depois em Console e verá:



Ele reclama que Phaser não foi definida. Dessa forma podemos corrigir vários erros em nosso código, tendo em vista que geralmente não recebemos mensagens de erro no navegador, mas use o Inspetor para descobrir sempre que tiver algum.

Pronto. Agora estamos preparados para começar nosso jogo.

Exemplo de Jogo tipo Breakout usando o Framework Phaser - Fase 2

Original em inglês -
https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser/Scaling

Esta fase trata da Escala

Definição de escala - é a relação entre as dimensões de um desenho e o objeto por ele representado. Exemplo: quero desenhar um carro, sendo o meu desenho todo proporcional ao original. Precisarei que meu desenho seja tantas vezes menor que o carro. Supomos que cada centímetro no desenho corresponde a 100 centímetros do carro. Então teremos uma escala de 1 para 100.

Aqui escala se refere a aplicar escala ao canvas do game em diferentes tamanhos de tela. Nós podemos ajustas a escala do game para caber em qualquer tamanho de tela automaticamente durante o estágio do preload, de forma que não tenhamos que nos preocupar com isso depois.

O Phaser Object Scale
Está disponível no Phaser um objeto especial **scale** com alguns métodos e propriedades.
Adicionar as 3 linhas abaixo à função preload():

```
function preload() {
    game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
    game.scale.pageAlignHorizontally = true;
    game.scale.pageAlignVertically = true;
}
```

Na primeira linha temos scaleMode. scaleMode tem algumas diferentes opções disponíveis para a forma como o canvas pode ser escalado:

- NO_SCALE
- EXACT_FIT
- SHOW_ALL
- RESIZE
- USER_SCALE

As duas últimas linhas na função preload() são responsáveis por alinhar o elemento canvas horizontal e verticalmente para que fique sempre centralizado na tela independente do tamanho da mesma.

Adicionando Cor Personalizada ao Fundo do Canvas
Por padrão a cor do fundo do canvas é preta.
O objeto stage tem uma propriedade backgroundColor para esta finalidade, que nós podemos setar usando a definição de cor da sintaxe do CSS. Adicione a seguinte linha abaixo para as outras 3 na função preload():

```
game.stage.backgroundColor = '#eee';
```

O código do nosso game até agora deve estar assim:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Gamedev Phaser Workshop - lesson 02: Scaling</title>
    <style>* { padding: 0; margin: 0; }</style>
    <script src="js/phaser.2.4.2.min.js"></script>
</head>
<body>
```

```
<script>
var game = new Phaser.Game(480, 320, Phaser.AUTO, null, {preload: preload, create: create,
update: update});

function preload() {
    game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
    game.scale.pageAlignHorizontally = true;
    game.scale.pageAlignVertically = true;
    game.stage.backgroundColor = '#eee';
}
function create() {}
function update() {}
</script>
</body>
</html>
```

Exemplo de Jogo tipo Breakout usando o Framework Phaser - Fase 3

Original - https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser/Load_the_assets_and_print_them_on_screen

Esta fase trata de carregar assets e mostrar na tela

Nosso jogo, depois de pronto, se comportará assim: Uma bolinha rolando na tela, rebatendo na raquete do jogador e destruindo os tijolinhos acima para ganhar pontos.

Nesta fase aprenderemos a adicionar sprites ao game.

Criando uma Bola

Vamos iniciar criando uma variável em JavaScript que representará a bola. Adicione a seguinte linha entre a inicialização do game (var game...) e a função preload():

```
var ball;
```

Nota: para este tutorial usaremos variáveis globais, mas fica o alerta para que se evite variáveis globais, preferindo as locais.

Carragando o Sprite da Bola

Carregar imagens e mostrar na tela é mais fácil usando o Phaser do que usando o JavaScript puro. Para carregar assets nós devemos usar o objeto game criado pelo Phaser, executando seu método load.image(). Adicione a seguinte linha à função preload() justamente abaixo das linhas existentes:

```
game.load.image('ball', 'img/ball.png');
```

O primeiro parâmetro do método acima é o nome que desejamos para o assets. Uma boa prática é usar o mesmo nome da imagem, para ficar mais fácil de recordar.

Download da imagem

https://github.com/end3r/Gamedev-Phaser-Content-Kit/blob/gh-pages/demos/img/ball.png

Crie a masta img dentro da nossa pasta breakout e copie a imagem baixada para ela, assim: \breakout\img\ball.png

Agora, para mostrar na tela, nós devemos usar outro método do Phaser chamado de add.sprite(). Adicione mais esta linha, mas agora na função create():

```
function create() {
    ball = game.add.sprite(50, 50, 'ball');
}
```

Isso deve adicionar a bola para o nosso game e renderizá-la na tela. Os dois primeiros parâmetros são as coordenadas x e y do canvas e o terceiro é o nome do assets que nós definimos anteriormente.

Definição de Canvas
Para quem ainda não conhece, o canvas é um elemento HTML usado para desenhar gráficos, textos e animações diretamente no browser do usuário. Podemos definir o canvas como um elemento que aceita desenhos dentro dele, igual a um **quadro branco** de pintura. Para desenhar qualquer coisa no canvas nós utilizamos a linguagem de programação **JavaScript**. Crédito - http://blog.triadworks.com.br/html5-desenhando-graficos-no-browser-com-canvas

O código do nosso game até agora deve estar assim:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Gamedev Phaser Workshop - lesson 02: Scaling</title>
    <style>* { padding: 0; margin: 0; }</style>
    <script src="js/phaser.2.4.2.min.js"></script>
</head>
<body>
<script>
var game = new Phaser.Game(480, 320, Phaser.AUTO, null, {preload: preload, create: create, update: update});

var ball;

function preload() {
    game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
    game.scale.pageAlignHorizontally = true;
    game.scale.pageAlignVertically = true;
    game.stage.backgroundColor = '#eee';
    game.load.image('ball', 'img/ball.png');
}
function create() {
    ball = game.add.sprite(50, 50, 'ball');
}

function update() {}
```

```
</script>
</body>
</html>
```

Exemplo de Jogo tipo Breakout usando o Framework Phaser - Fase 4

Original em inglês - https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser/Move_the_ball

Esta fase trata do movimento da bola

Nós temos a nossa bola azul na tela, mas ela está parada e não faz nada. Vamos fazer ela se movimentar. Esta fase mostra como fazer isso.

Atualizando a posição da bola em cada frame

O código na função update() é executado sempre a cada frame, sendo que este é o lugar adequado para colocar o código que deve atualizar a posição da bola na tela. Adicione as duas linhas a seguir dentro da função update(), como mostrado:

```
function update() {
    ball.x += 1;
    ball.y += 1;
}
```

O código acima adiciona 1 para as propriedades x e y representando as coordenadas da bola no canvas, em cada frame. Caso tenha criado o código como citado acima e até agora, atualize a página do navegador e veja que agora a bola se movimenta. Veja que a bola passa direto no fundo da página, mas isso ficará para próximas fases.

Agora é uma boa hora para falar sobre o sistema de coordenadas usada aqui.

A origem dos eixos fica acima e à esquerda. Isso é importante saber para efetuar movimentos. Observe que a bola vai descendo e para a direita. Aumenta para a direita e para baixo, como nas setas acima.

O código do nosso game até agora deve estar assim:

```
<!DOCTYPE html>
<html>
<head>
```

```
    <meta charset="utf-8" />
    <title>Gamedev Phaser Workshop - lesson 02: Scaling</title>
    <style>* { padding: 0; margin: 0; }</style>
    <script src="js/phaser.2.4.2.min.js"></script>
</head>
<body>
<script>
var game = new Phaser.Game(480, 320, Phaser.AUTO, null, {preload: preload, create: create,
update: update});

var ball;

function preload() {
    game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
    game.scale.pageAlignHorizontally = true;
    game.scale.pageAlignVertically = true;
    game.stage.backgroundColor = '#eee';
    game.load.image('ball', 'img/ball.png');
}
function create() {
    ball = game.add.sprite(50, 50, 'ball');
}
```

```
function update() {
    ball.x += 1;
    ball.y += 1;
}
</script>
</body>
</html>
```

Próximos Passos

O próximo passo é adicionar uma básica detecção de colisão, de forma que nossa bola possa rebater na parede. Isso geralmente acarreta muitas linhas de código, mas felizmente o Phaser facilita bastante as coisas. Antes de tudo precisamos conhecer sobre a física do framework Phaser, que veremos na próxima fase.

Exemplo de Jogo tipo Breakout usando o Framework Phaser - Fase 5

Original em inglês - https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser/Physics

Esta fase trata da Física do Jogo

Para uma detecção apropriada da colisão entre objetos em nosso jogo nós temos necessidade de usar física no jogo. Este artigo introduz o assunto no Phaser e demonstra um exemplo simples de configuração.

Adicionar Física

O Phaser já vem empacotado com 3 engines diferentes de física - Arcade Physics, P2 e Ninja Physics - com uma quarta alternativa, o Box2D, que começa a ficar disponível como um plugin comercial. Para um simples game como o nosso, nós podemos usar a engine Arcade Physics. Nós não necessitamos de nenhum cálculo geométrico pesado. - afinal de contas temos apenas uma bola rebatendo nas paredes e nos tijolos.

Primeiro, deixe me innicializar a engine Arcade Physics em nosso game. Adicione o método physics.startSystem() para o início da função create(), faça que seja a primeira linha da função, como mostro abaixo:

```
function create() {
    game.physics.startSystem(Phaser.Physics.ARCADE);
    ball = game.add.sprite(50, 50, 'ball');
}
```

Depois, nós precisamos habilitar nossa bola para o sistema de física. A física não é habilitada por padrão nos objetos do Phaser. Adicione a seguinte linha no início da função create(), assim:

```
function create() {
    game.physics.startSystem(Phaser.Physics.ARCADE);
    ball = game.add.sprite(50, 50, 'ball');
```

```
    game.physics.enable(ball, Phaser.Physics.ARCADE);
}
```

Depois, se nós desejamos mover nossa bola pela tela, nós podemos configurar a velocidade em seu corpo. Nossa função create() deve ficar assim:

```
function create() {
    game.physics.startSystem(Phaser.Physics.ARCADE);
    ball = game.add.sprite(50, 50, 'ball');
    game.physics.enable(ball, Phaser.Physics.ARCADE);
    ball.body.velocity.set(150, 150);
}
```

Remover conteúdo da função ()
Vamos remover as duas linhas que adicionamos à função update:

```
function update() {
}
```

Divirta-se com a Física
Você pode fazer muito mais com a física. Por exemplo, usando
ball.body.gravity.y = 100;
Você seta a gravidade vertical da bola. Como resultado ela será lançado para cima, mas depois cairá devido aos efeitos da gravidade puxando-o para baixo.

Outras referências sobre física:
http://phaser.io/docs/2.4.6/index#physics
http://phaser.io/examples/v2/category/arcade-physics

O código do nosso game até agora deve estar assim:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Gamedev Phaser Workshop - lesson 02: Scaling</title>
    <style>* { padding: 0; margin: 0; }</style>
    <script src="js/phaser.2.4.2.min.js"></script>
</head>
<body>
<script>
var game = new Phaser.Game(480, 320, Phaser.AUTO, null, {preload: preload, create: create,
update: update});

var ball;

function preload() {
    game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
```

```
    game.scale.pageAlignHorizontally = true;
    game.scale.pageAlignVertically = true;
    game.stage.backgroundColor = '#eee';
    game.load.image('ball', 'img/ball.png');
}

function create() {
    game.physics.startSystem(Phaser.Physics.ARCADE);
    ball = game.add.sprite(50, 50, 'ball');
    game.physics.enable(ball, Phaser.Physics.ARCADE);
    ball.body.velocity.set(150, 150);
}

function update() {
}
</script>
</body>
</html>
```

Exemplo de Jogo tipo Breakout usando o Framework Phaser - Fase 6

Original em inglês -
https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser/Bounce_off_the_walls

Esta fase trata da detecção de colisão

Evitar que a bola atravesse as fronteiras do mundo
A maneira mais fácil de fazer a bola rebater nas paredes é dizer ao framework que nós desejamos tratar as fronteiras do elemento canvas como paredes e não deixar a bola mover-se através delas. No Phaser isso pode facilmente ser conseguido usando collideWorldsBound. Adicione a linha abaixo logo após game.physics.enabled(...):

```
ball.body.collideWorldBounds = true;
```

Rebater nas fronteiras

Agora, a bola vai parar na borda do canvas, em vez de desaparecer, mas não rebaterá. Para fazer com que isso ocorra temos que definir o seu rebatimento. Adicione a seguinte linha abaixo da anterior:

```
ball.body.bounce.set(1);
```

Tente recarregar a página do jogo agora. Verá a bolinha movendo-se e rebatendo nas paredes.

Para testar o código online use o link referido no tutorial original:
https://jsfiddle.net/end3r/dcw36opz/?utm_source=website&utm_medium=embed&utm_campaign=dcw36opz

O código do nosso game até agora deve estar assim:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Gamedev Phaser Workshop - lesson 02: Scaling</title>
    <style>* { padding: 0; margin: 0; }</style>
    <script src="js/phaser.2.4.2.min.js"></script>
</head>
<body>
<script>
var game = new Phaser.Game(480, 320, Phaser.AUTO, null, {preload: preload, create: create,
update: update});

var ball;

function preload() {
    game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
    game.scale.pageAlignHorizontally = true;
    game.scale.pageAlignVertically = true;
    game.stage.backgroundColor = '#eee';
    game.load.image('ball', 'img/ball.png');
}

function create() {
    game.physics.startSystem(Phaser.Physics.ARCADE);
    ball = game.add.sprite(50, 50, 'ball');
    game.physics.enable(ball, Phaser.Physics.ARCADE);
    ball.body.velocity.set(150, 150);
    ball.body.collideWorldBounds = true;
    ball.body.bounce.set(1);
}

function update() {
}
</script>
</body>
</html>
```

Exemplo de Jogo tipo Breakout usando o Framework Phaser - Fase 7

Original em inglês - https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser/Player_paddle_and_controls

Esta fase trata da raquete do jogador e dos controles

Nós temos a bolinha movendo pela tela e rebatendo nas paredes, mas isso rapidamente fica chato pois não tem nenhuma interação. Neste artigo nós introduziremos um jogador (paddle) como uma raquete que se move na horizontal abaixo e rebate a bola.

Download do paddle
https://github.com/end3r/Gamedev-Phaser-Content-Kit/blob/gh-pages/demos/img/paddle.png

Baixe e copie para a sub pasta \img do nosso jogo

Renderizando o paddle
Do ponto de vista do framework o paddle é semelhante à bola - nós precisamos também de uma variável para representá-lo, carregar o respectivo assets e então fazer a mágica.

Carregar o Paddle
Primeiro crie a variável paddle logo abaixo da variável ball.

```
var ball;
var paddle;
```

Na função preload() carregue o paddle como fizemos com a ball.

```
function preload() {
    // ...
    game.load.image('ball', 'img/ball.png');
    game.load.image('paddle', 'img/paddle.png');
}
```

Renderizando o Paddle com Física
Agora nós devemos inicializar o paddle adicionando a chamada add.sprite() para a função create(). Adicione a seguinte linha na parte de baixo da função:

```
paddle = game.add.sprite(game.world.width*0.5, game.world.height-5, 'paddle');
```

Nós podemos usar o world.width e world.height para posicionar o paddle exatamente nós desejamos ele: game.world.width*0.5 deve ser o meio da tela. Em nosso caso o world é o mesmo que o canvas, mas para outros tipos de jogos, como o side-scrollers por exemplo, o world deve ser maior e você pode brincar com este para conseguir interessantes efeitos.

Como você deve ter notado se você recarregar a index.html a este ponto, o paddle fica não exatamente no meio. Que houve? Por causa de a âncora da posição ser calculada sempre iniciando no topo à esquerda na beira do objeto. Nós podemos mudar isto para ter a âncora do paddle para a largura e para a parte de baixo da sua altura,

As you'll notice if you reload your `index.html` at this point, the paddle is currently not exactly in the middle. Why? Because the anchor from which the position is calculated always starts from the top left edge of the object. We can change that to have the anchor in the middle of the paddle's width and at the bottom of it's height, por isso é mais fácil posicioná-lo contra a borda inferior. Adicione a seguinte linha abaixo da anterior:

```
paddle.anchor.set(0.5,1);
```

O paddle agora foi posicionado corretamente onde nós desejamos que fosse. Agora, para fazer com que ele cilida com a bola, nós precisamos habilitar a física para o paddle. Continue e adicione a linha abaixo para o final da função create():

game.physics.enable(paddle, Phaser.Physics.ARCADE);

Agora a mágica pode começar a acontecer - o framework pode checar a detecção de colisão a cada frame.

Detecção de Colisão entre a Bola e o Paddle
Para habilitar a detecção de colisão entre o paddle e a bola, adicione o método collide() para a função update() como abaixo:

```
function update() {
    game.physics.arcade.collide(ball, paddle);
}
```

O primeiro parâmetro é um dos objetos em que estamos interessados nele, em nosso caso a bola e o segundo é o ouro - o paddle. Isto funciona mas não sai como esperávamos - quando a bola toca no paddle o paddle cai na tela!

Imobilizando a raquete/paddle

Tudo o que queremos é a bola rebatendo no paddle e o paddle ficando no mesmo lugar. Podemos definir o corpo da raquete/paddle para que seja imóvel, para que a bola não vá movê-lo quando o atinge. Para fazer isso, adicione a linha a seguir na parte inferior da função create ():

paddle.body.immovable = true;

Agora funciona como esperávamos.

Controlando a Raquete/Entrada du Usuário
Nosso próximo problema é que não podemos mover a raquete. Para isso, podemos usar a entrada padrão do sistema (mouse ou toque, dependendo da plataforma) e definir a posição da raquete para onde a entrada do usuário (input) posicionar ela. Adicione o seguinte nova linha para a função update (), como mostrado:

```
function update() {
    game.physics.arcade.collide(ball, paddle);
    paddle.x = game.input.x;
}
```

Agora a cada novo frame a posição x do paddle deve ser ajustada para a posição x do input. mas quando nós iniciamos o game, a posição do paddle não está no meio. Isto é por que a posição do input ainda não foi definida.

Centralizando o Paddle na Horizontal
Para corrigir isso e deixar o paddle no centro nós podemos configurar a posição default (caso uma posição input ainda não tenha sido definida) para o meio da tela. Atualize a linha anterior para esta: paddle.x = game.input.x || game.world.width*0.5;

Recarregue o jogo e teste agora.

Posicionando a Bola

A raquete funciona como esperado, então vamos posicionar a bola nela. É muito semelhante ao posicionamento da raquete - precisamos tê-la colocado no meio da tela horizontal e verticalmente na parte inferior com um pouco de deslocamento da parte inferior. Para colocá-la exatamente como queremos que vamos definir a âncora para o meio exato da bola. Encontrar a linha existente da bola

ball = game.add.sprite (...)

e substituí-lo com as duas linhas seguintes:

```
ball = game.add.sprite(game.world.width*0.5, game.world.height-25, 'ball');
ball.anchor.set(0.5);
```

Então a velocidade permanece a mesma - nós estamos apenas mudando o valor do segundo parâmetro de 150 para -150, sendo que a bola deve iniciar o game movendo para cima ao invés de para baixo. Procure a linha existente

ball.ody.velocity.set(...) e altere para a linha seguinte:

```
ball.body.velocity.set(150, -150);
```

Agora a bola deve iniciar corretamente, bem no meio da raquete.

Para testar esta fase online - https://jsfiddle.net/end3r/ogqza0ye/?utm_source=website&utm_medium=embed&utm_campaign=ogqza0ye

O código do nosso game até agora deve estar assim:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Gamedev Phaser Workshop - lesson 02: Scaling</title>
    <style>* { padding: 0; margin: 0; }</style>
    <script src="js/phaser.2.4.2.min.js"></script>
</head>
<body>
<script>
var game = new Phaser.Game(480, 320, Phaser.AUTO, null, {preload: preload, create: create, update: update});

var ball;
var paddle;

function preload() {
    game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
    game.scale.pageAlignHorizontally = true;
    game.scale.pageAlignVertically = true;
    game.stage.backgroundColor = '#eee';
    game.load.image('ball', 'img/ball.png');
```

```
    game.load.image('paddle', 'img/paddle.png');
}

function create() {
    game.physics.startSystem(Phaser.Physics.ARCADE);
    ball = game.add.sprite(game.world.width*0.5, game.world.height-25, 'ball');
    ball.anchor.set(0.5);
    game.physics.enable(ball, Phaser.Physics.ARCADE);
    ball.body.velocity.set(150, -150);
    ball.body.collideWorldBounds = true;
    ball.body.bounce.set(1);

    paddle = game.add.sprite(game.world.width*0.5, game.world.height-5, 'paddle');
    paddle.anchor.set(0.5,1);
    game.physics.enable(paddle, Phaser.Physics.ARCADE);
    paddle.body.immovable = true;
}

function update() {
    game.physics.arcade.collide(ball, paddle);
    paddle.x = game.input.x || game.world.width*0.5;
}
</script>
</body>
</html>
```

Exemplo de Jogo tipo Breakout usando o Framework Phaser - Fase 8

Original em inglês - https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser/Game_over

Esta fase trata do Game Over

Para tornar o jogo mais interessante nós devemos introduzir a possibilidade de perda - caso o jogador não toque a bola antes ela alcança a borda inferior da tela e então será o Game Over.

Como Perder

Para provê a habilidade de perda nós devemos desabilitar a colisão da bola com a borda inferior da tela. Adicione o código abaixo na função create(); justamente após sua definição dos atributos da bola:

```
game.physics.arcade.checkCollision.down = false;
```

Isto deve fazer com que as 3 paredes (de cima, esquerda e direita) devem rebater a bola de volta, mas a quarta parede (de baixo) deve desaparecer, deixando a bola cair na tela caso a raquete não a acerte. Nós precisamos de uma maneira de detectar isto e agir de acordo. Adicionar as seguintes linhas justamente abaixo da linha anterior, assim:

```
ball.checkWorldBounds = true;
ball.events.onOutOfBounds.add(function(){
    alert('Game over!');
```

```
    location.reload();
}, this);
```

Adicionando estas linhas deve fazer com que a bola cheque os limites do mundo (em nosso caso o canvas) e execute a função bound para o evento onOutOfBounds. Quando você clica no alert('Game Over') a página deve ser recarregada e o jogo começar novamente.

Execução online desta fase, no jsfiddle: https://jsfiddle.net/end3r/436bckb7/?utm_source=website&utm_medium=embed&utm_campaign=436bckb7

O código do nosso game até agora deve estar assim:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Gamedev Phaser Workshop - lesson 02: Scaling</title>
    <style>* { padding: 0; margin: 0; }</style>
    <script src="js/phaser.2.4.2.min.js"></script>
</head>
<body>
<script>
var game = new Phaser.Game(480, 320, Phaser.AUTO, null, {preload: preload, create: create, update: update});

var ball;
var paddle;

function preload() {
    game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
    game.scale.pageAlignHorizontally = true;
    game.scale.pageAlignVertically = true;
    game.stage.backgroundColor = '#eee';
    game.load.image('ball', 'img/ball.png');
    game.load.image('paddle', 'img/paddle.png');
}

function create() {
    game.physics.startSystem(Phaser.Physics.ARCADE);
    ball = game.add.sprite(game.world.width*0.5, game.world.height-25, 'ball');
    ball.anchor.set(0.5);
    game.physics.enable(ball, Phaser.Physics.ARCADE);
    ball.body.velocity.set(150, -150);
    ball.body.collideWorldBounds = true;
    ball.body.bounce.set(1);

    ball.checkWorldBounds = true;
```

```
    ball.events.onOutOfBounds.add(function(){
        alert('Game over!');
        location.reload();
    }, this);

    paddle = game.add.sprite(game.world.width*0.5, game.world.height-5, 'paddle');
    paddle.anchor.set(0.5,1);
    game.physics.enable(paddle, Phaser.Physics.ARCADE);
    paddle.body.immovable = true;
}

function update() {
    game.physics.arcade.collide(ball, paddle);
    paddle.x = game.input.x || game.world.width*0.5;
}
</script>
</body>
</html>
```

Exemplo de Jogo tipo Breakout usando o Framework Phaser - Fase 9

Original em inglês -
https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser/Build_the_brick_field

Esta fase trata de Construir o bloco de Tijolinhos

Criar um bloco de tijolinhos é um pouco mais complicado do que criar apenas um. Deixe mostrar como criar um grupo de tijolos e mostrar na tela usando um laço.

Definindo as novas variáveis
Primeiro, deixe-me definir as novas variáveis necessárias - adicione o seguinte para a região da nossa definição de variáveis:

```
var bricks;
var newBrick;
var brickInfo;
```

Avariável **bricks** acima deve ser usada para criar um grupo de tijolinhos, a **newBrick** será usada para ser um novo objeto adicionado ao grupo em todas as iterações do laço e **brickInfo** deverá armazenar todas as informações de que precisamos.

Renderizando a imagem do tijolo
Efetuando o download da imagem -
https://github.com/end3r/Gamedev-Phaser-Content-Kit/blob/gh-pages/demos/img/brick.png
Salve a imagem no subdiretório \img.
Agora vamos carregar a imagem do tijolo, logo abaixo das demais:

```
function preload() {
    // ...
```

```
    game.load.image('brick', 'img/brick.png');
}
```

Desenhando os Tijolos

Nós devemos colocar todo o código para desenhar os tijolos dentro da função initBricks() para guardar ele separado do resto do código. Adicionar uma chamada para a função initBricks() ao final da função create():

```
function create(){
    // ...
    initBricks();
}
```

Agora a função em si. Adicione a função initBrick() para o final do código do nosso game, justamente antes da tag de fechamento </script> como abaixo. Para começar

Now onto the function itself. Add the `initBricks()` function at the end of our games code, just before the closing </script> tag, as shown below:

```
function initBricks() {
    brickInfo = {
        width: 50,
        height: 20,
        count: {
            row: 3,
            col: 7
        },
        offset: {
            top: 50,
            left: 60
        },
        padding: 10
    }
}
```

Este objeto brickInfo irá armazenar todas as informações de que precisamos: a largura e altura de um único tijolo, o número de linhas e colunas de tijolos que vamos ver na tela, a parte superior e deslocamento à esquerda (eft offset) (o local na tela em que deve começar a desenhar os tijolos) e do preenchimento entre cada linha e coluna de tijolos (padding).

Agora, vamos começar a criar os próprios tijolos - adicionar um grupo vazio primeiro para conter os tijolos, adicionando a seguinte linha na parte inferior da função initBricks():

```
bricks = game.add.group();
```

Nós podemos efetuar um laço pelas linhas e colunas para criar um novo tijolo em cada iteração - adicione a linha seguinte abaixo para nosso código anterior:

```
for(c=0; c<brickInfo.count.col; c++) {
    for(r=0; r<brickInfo.count.row; r++) {
        // create new brick and add it to the group
```

```
    }
}
```

Agora, vamos começar a criar os próprios tijolos - adicionar um grupo vazio primeiro para conter os tijolos, adicionando a seguinte linha na parte inferior da função initBricks (): Desta forma, vamos criar o número exato de tijolos que precisamos e tê-los todos contido num grupo. Agora precisamos adicionar algum código dentro da estrutura de loop aninhado para desenhar cada tijolo. Preencha o conteúdo como mostrado abaixo:

```
for(c=0; c<brickInfo.count.col; c++) {
    for(r=0; r<brickInfo.count.row; r++) {
        var brickX = 0;
        var brickY = 0;
        newBrick = game.add.sprite(brickX, brickY, 'brick');
        game.physics.enable(newBrick, Phaser.Physics.ARCADE);
        newBrick.body.immovable = true;
        newBrick.anchor.set(0.5);
        bricks.add(newBrick);
    }
}
```

Aqui nós estamos no loop através das linhas e colunas para criar os novos tijolos e colocá-los na tela. O tijolo recém-criado está habilitado para o motor de física Arcade, seu corpo está definido para ser fixo (por isso não se moverá quando atingido pela bola), e também está definida a âncora para estar no meio e adicionando o tijolo para o grupo.

O problema é atualmente que estamos pintando todos os tijolos em um só lugar, com as coordenadas (0,0). O que temos de fazer é desenhar cada tijolo em sua própria posição x e y. Atualizar as linhas brickX e bricky da seguinte forma:

```
var brickX = (r*(brickInfo.width+brickInfo.padding))+brickInfo.offset.left;
var brickY = (c*(brickInfo.height+brickInfo.padding))+brickInfo.offset.top;
```

Cada posição do brickX é trabalhada como brickInfo.width mais bkickInfo.padding, multiplicado pelo número da linha, r, mais o brickInfo.offset.left; a lógica para a bricky é idêntica, exceto que ele usa os valores para o número de coluna, c, brickInfo.height e brickInfo.offset.top. Agora, cada único tijolo pode ser colocado em seu lugar correto, com estofamento/padding entre cada tijolo, e desenhado em um deslocamento das bordas de lona esquerda e superior.

Checando o Código da função brickInfo()
Aqui está o código completo da função brickInfo():

```
function initBricks() {
    brickInfo = {
        width: 50,
        height: 20,
        count: {
            row: 3,
            col: 7
        },
```

```
        offset: {
            top: 50,
            left: 60
        },
        padding: 10
    }
    bricks = game.add.group();
    for(c=0; c<brickInfo.count.col; c++) {
        for(r=0; r<brickInfo.count.row; r++) {
            var brickX = (r*(brickInfo.width+brickInfo.padding))+brickInfo.offset.left;
            var brickY = (c*(brickInfo.height+brickInfo.padding))+brickInfo.offset.top;
            newBrick = game.add.sprite(brickX, brickY, 'brick');
            game.physics.enable(newBrick, Phaser.Physics.ARCADE);
            newBrick.body.immovable = true;
            newBrick.anchor.set(0.5);
            bricks.add(newBrick);
        }
    }
}
```

Se recarregar a tela do navegador agora verá os tijolos na tela, todos a uma certa distância um do outro.

Execução online desta fase, no jsfiddle: https://jsfiddle.net/end3r/cck2b9e8/?utm_source=website&utm_medium=embed&utm_campaign=cck2b9e8

O código do nosso game até agora deve estar assim:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Gamedev Phaser Workshop - lesson 02: Scaling</title>
    <style>* { padding: 0; margin: 0; }</style>
    <script src="js/phaser.2.4.2.min.js"></script>
</head>
<body>
<script>
var game = new Phaser.Game(480, 320, Phaser.AUTO, null, {preload: preload, create: create, update: update});

var ball;
var paddle;
var bricks;
var newBrick;
var brickInfo;
```

```
function preload() {
    game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
    game.scale.pageAlignHorizontally = true;
    game.scale.pageAlignVertically = true;
    game.stage.backgroundColor = '#eee';
    game.load.image('ball', 'img/ball.png');
    game.load.image('paddle', 'img/paddle.png');
    game.load.image('brick', 'img/brick.png');
}

function create() {
    game.physics.startSystem(Phaser.Physics.ARCADE);
    ball = game.add.sprite(game.world.width*0.5, game.world.height-25, 'ball');
    ball.anchor.set(0.5);
    game.physics.enable(ball, Phaser.Physics.ARCADE);
    ball.body.velocity.set(150, -150);
    ball.body.collideWorldBounds = true;
    ball.body.bounce.set(1);

    ball.checkWorldBounds = true;
    ball.events.onOutOfBounds.add(function(){
        alert('Game over!');
        location.reload();
    }, this);

    paddle = game.add.sprite(game.world.width*0.5, game.world.height-5, 'paddle');
    paddle.anchor.set(0.5,1);
    game.physics.enable(paddle, Phaser.Physics.ARCADE);
    paddle.body.immovable = true;

    initBricks();
}

function update() {
    game.physics.arcade.collide(ball, paddle);
    paddle.x = game.input.x || game.world.width*0.5;
}

function initBricks() {
    brickInfo = {
        width: 50,
        height: 20,
        count: {
            row: 7,
            col: 3
        },
        offset: {
```

```
            top: 50,
            left: 60
        },
        padding: 10
    }
    bricks = game.add.group();
    for(c=0; c<brickInfo.count.col; c++) {
        for(r=0; r<brickInfo.count.row; r++) {
            var brickX = (r*(brickInfo.width+brickInfo.padding))+brickInfo.offset.left;
            var brickY = (c*(brickInfo.height+brickInfo.padding))+brickInfo.offset.top;
            newBrick = game.add.sprite(brickX, brickY, 'brick');
            game.physics.enable(newBrick, Phaser.Physics.ARCADE);
            newBrick.body.immovable = true;
            newBrick.anchor.set(0.5);
            bricks.add(newBrick);
        }
    }
}
</script>
</body>
</html>
```

Exemplo de Jogo tipo Breakout usando o Framework Phaser - Fase 10

Original em inglês - https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser/Collision_detection

Esta fase trata da Detecção de Colisão

Agora nosso próximo desafio - a detecção de colisão entre os tijolos e a bola. Ainda bem que podemos usar o engine de física para detectar não somente entre objetos simples (como a bola e o paddle) mas também entre um objeto e um grupo.

Detecção de Colisão entre Tijolos e Bola

O motor de física torna tudo muito mais fácil - só precisamos adicionar duas peças simples de código. Primeiro, adicione uma nova linha dentro da função update() que verifica a detecção de colisão entre bola e tijolos, como mostrado abaixo:

```
function update() {
    game.physics.arcade.collide(ball, paddle);
    game.physics.arcade.collide(ball, bricks, ballHitBrick);
    paddle.x = game.input.x || game.world.width*0.5;
}
```

A posição da bola é calculada pelas posições de todos os tijolos no grupo. O terceiro parâmetro, opcional é a função executada quando ocorre uma colisão - ballHitBricks(). Criar esta nova função na a parte inferior do código, imediatamente antes da tag de fechamento </script>, como segue:

```
function ballHitBrick(ball, brick) {
    brick.kill();
}
```

E é isso! Atualize o código e você deve ver a nova detecção de colisão trabalhando tão necessáriamente.

Graças a Phaser existem dois parâmetros passados para a função - o primeiro é a bola, que foi explicitamente definido no método de colisão, e o segundo é o único tijolo do grupo de tijolos que a bola está a colidir com ele. Dentro da função nós removemos o tijolo em questão da tela simplesmente executando o método kill () nele.

Você esperaria ter que escrever muito mais cálculos de sua própria conta para implementar a detecção de colisão quando se usa JavaScript puro. Essa é a beleza do uso do framework - você pode deixar um monte de código chato para o Phaser, e se concentrar nas partes mais divertidas e interessantes de fazer um jogo.

Testando
Experimente novamente e lembre que posso ter cometido algum erro. Caso perceba algum erro verifique o original em inglês e por favor me avise nos comentários.

Execução online desta fase, no jsfiddle: https://jsfiddle.net/end3r/wwneakwf/?utm_source=website&utm_medium=embed&utm_campaign=wwneakwf

O código do nosso game até agora deve estar assim:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Gamedev Phaser Workshop - lesson 02: Scaling</title>
    <style>* { padding: 0; margin: 0; }</style>
    <script src="js/phaser.2.4.2.min.js"></script>
</head>
<body>
<script>
var game = new Phaser.Game(480, 320, Phaser.AUTO, null, {preload: preload, create: create, update: update});

var ball;
var paddle;
var bricks;
var newBrick;
var brickInfo;

function preload() {
```

```
    game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
    game.scale.pageAlignHorizontally = true;
    game.scale.pageAlignVertically = true;
    game.stage.backgroundColor = '#eee';
    game.load.image('ball', 'img/ball.png');
    game.load.image('paddle', 'img/paddle.png');
    game.load.image('brick', 'img/brick.png');
}

function create() {
    game.physics.startSystem(Phaser.Physics.ARCADE);
    ball = game.add.sprite(game.world.width*0.5, game.world.height-25, 'ball');
    ball.anchor.set(0.5);
    game.physics.enable(ball, Phaser.Physics.ARCADE);
    ball.body.velocity.set(150, -150);
    ball.body.collideWorldBounds = true;
    ball.body.bounce.set(1);

    ball.checkWorldBounds = true;
    ball.events.onOutOfBounds.add(function(){
        alert('Game over!');
        location.reload();
    }, this);

    paddle = game.add.sprite(game.world.width*0.5, game.world.height-5, 'paddle');
    paddle.anchor.set(0.5,1);
    game.physics.enable(paddle, Phaser.Physics.ARCADE);
    paddle.body.immovable = true;

    initBricks();
}

function update() {
    game.physics.arcade.collide(ball, paddle);
    game.physics.arcade.collide(ball, bricks, ballHitBrick);
    paddle.x = game.input.x || game.world.width*0.5;
}

function initBricks() {
    brickInfo = {
        width: 50,
        height: 20,
        count: {
            row: 7,
            col: 3
        },
        offset: {
            top: 50,
```

```
            left: 60
        },
        padding: 10
    }
    bricks = game.add.group();
    for(c=0; c<brickInfo.count.col; c++) {
        for(r=0; r<brickInfo.count.row; r++) {
            var brickX = (r*(brickInfo.width+brickInfo.padding))+brickInfo.offset.left;
            var brickY = (c*(brickInfo.height+brickInfo.padding))+brickInfo.offset.top;
            newBrick = game.add.sprite(brickX, brickY, 'brick');
            game.physics.enable(newBrick, Phaser.Physics.ARCADE);
            newBrick.body.immovable = true;
            newBrick.anchor.set(0.5);
            bricks.add(newBrick);
        }
    }
}

function ballHitBrick(ball, brick) {
    brick.kill();
}
</script>
</body>
</html>
```

Exemplo de Jogo tipo Breakout usando o Framework Phaser - Fase 11

Original em inglês - https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser/The_score

Esta fase trata dos Pontos/Score

Manter a contagem de pontos tornará o jogo mais interessante. Você pode tentar bater seu récorde ou seus amigos.

Nós devemos usar uma variável separada para armazenar os pontos e o método text() do Phaser para mostrar na tela.

Novas Variáveis
Adicione duas novas variáveis logo após as anteriores já definidas:

```
// ...
var scoreText;
var score = 0;
```

Adicionar o Texto dos Pontos para o Jogo
Adicione a linha abaixo para o final da função create():

```
scoreText = game.add.text(5, 5, 'Points: 0', { font: '18px Arial', fill: '#0095DD' });
```

O método text () pode ter quatro parâmetros:

As coordenadas x e y para desenhar o texto na tela.
O texto real que será mostrado.
O estilo da fonte para tornar o texto.

O último parâmetro é muito parecido com CSS. No nosso caso, o texto da pontuação será azul, dimensionada em 18 pixels, e usar a fonte Arial.

Atualização dos Pontos quando os Tijolos são Destruídos
Vamos aumentar o número de pontos a cada vez que a bola atingir um tijolo e atualizar o texto da pontuação para exibir a pontuação atualizada. Isso pode ser feito usando o método setText () - adicionar as duas novas linhas vistas abaixo para a função ballHitBrick():

```
function ballHitBrick(ball, brick) {
    brick.kill();
    score += 10;
    scoreText.setText('Points: '+score);
}
```

Executando
Atualize o navegador e teste o sistema de pontos.

Execução online desta fase, no jsfiddle: https://jsfiddle.net/end3r/n8o6rhrf/?utm_source=website&utm_medium=embed&utm_campaign=n8o6rhrf

O código do nosso game até agora deve estar assim:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Gamedev Phaser Workshop - lesson 02: Scaling</title>
    <style>* { padding: 0; margin: 0; }</style>
    <script src="js/phaser.2.4.2.min.js"></script>
</head>
<body>
<script>
var game = new Phaser.Game(480, 320, Phaser.AUTO, null, {preload: preload, create: create, update: update});

var ball;
var paddle;
var bricks;
var newBrick;
var brickInfo;
var scoreText;
var score = 0;
```

```
function preload() {
    game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
    game.scale.pageAlignHorizontally = true;
    game.scale.pageAlignVertically = true;
    game.stage.backgroundColor = '#eee';
    game.load.image('ball', 'img/ball.png');
    game.load.image('paddle', 'img/paddle.png');
    game.load.image('brick', 'img/brick.png');
}

function create() {
    game.physics.startSystem(Phaser.Physics.ARCADE);
    ball = game.add.sprite(game.world.width*0.5, game.world.height-25, 'ball');
    ball.anchor.set(0.5);
    game.physics.enable(ball, Phaser.Physics.ARCADE);
    ball.body.velocity.set(150, -150);
    ball.body.collideWorldBounds = true;
    ball.body.bounce.set(1);

    ball.checkWorldBounds = true;
    ball.events.onOutOfBounds.add(function(){
        alert('Game over!');
        location.reload();
    }, this);

    paddle = game.add.sprite(game.world.width*0.5, game.world.height-5, 'paddle');
    paddle.anchor.set(0.5,1);
    game.physics.enable(paddle, Phaser.Physics.ARCADE);
    paddle.body.immovable = true;

    initBricks();
    scoreText = game.add.text(5, 5, 'Points: 0', { font: '18px Arial', fill: '#0095DD' });
}

function update() {
    game.physics.arcade.collide(ball, paddle);
    game.physics.arcade.collide(ball, bricks, ballHitBrick);
    paddle.x = game.input.x || game.world.width*0.5;
}

function initBricks() {
    brickInfo = {
        width: 50,
        height: 20,
        count: {
            row: 7,
            col: 3
        },
```

```
        offset: {
            top: 50,
            left: 60
        },
        padding: 10
    }
    bricks = game.add.group();
    for(c=0; c<brickInfo.count.col; c++) {
        for(r=0; r<brickInfo.count.row; r++) {
            var brickX = (r*(brickInfo.width+brickInfo.padding))+brickInfo.offset.left;
            var brickY = (c*(brickInfo.height+brickInfo.padding))+brickInfo.offset.top;
            newBrick = game.add.sprite(brickX, brickY, 'brick');
            game.physics.enable(newBrick, Phaser.Physics.ARCADE);
            newBrick.body.immovable = true;
            newBrick.anchor.set(0.5);
            bricks.add(newBrick);
        }
    }
}

function ballHitBrick(ball, brick) {
    brick.kill();
    score += 10;
    scoreText.setText('Points: '+score);
}
</script>
</body>
</html>
```

Exemplo de Jogo tipo Breakout usando o Framework Phaser - Fase 12

Original em inglês -
https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser/Win_the_game

Esta fase trata da Vitória/Win

Implementar a vitória no jogo é bem fácil. Caso você destrua todos os tijolos então você ganhou.

Como você Ganha o Jogo?
Adicione o seguinte código para a função ballHitBrick():

```
function ballHitBrick(ball, brick) {
    brick.kill();
    score += 10;
    scoreText.setText('Points: '+score);
    if(score === brickInfo.count.row*brickInfo.count.col*10) {
        alert('You won the game, congratulations!');
        location.reload();
    }
}
```

Se a sua pontuação é a mesma que o número de tijolos inicialmente disponíveis na tela multiplicado por 10 (como você marca 10 pontos para cada tijolo), então nós mostramos uma mensagem vencedora, um alerta para reiniciar o jogo.

Executando

Atualize o navegador e teste o sistema de pontos.

Execução online desta fase, no jsfiddle

https://jsfiddle.net/end3r/u8waa4Lx/?utm_source=website&utm_medium=embed&utm_campaign=u8waa4Lx

O código do nosso game até agora deve estar assim:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Gamedev Phaser Workshop - lesson 02: Scaling</title>
    <style>* { padding: 0; margin: 0; }</style>
    <script src="js/phaser.2.4.2.min.js"></script>
</head>
<body>
<script>
var game = new Phaser.Game(480, 320, Phaser.AUTO, null, {preload: preload, create: create, update: update});
```

```
var ball;
var paddle;
var bricks;
var newBrick;
var brickInfo;
var scoreText;
var score = 0;

function preload() {
    game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
    game.scale.pageAlignHorizontally = true;
    game.scale.pageAlignVertically = true;
    game.stage.backgroundColor = '#eee';
    game.load.image('ball', 'img/ball.png');
    game.load.image('paddle', 'img/paddle.png');
    game.load.image('brick', 'img/brick.png');
}

function create() {
    game.physics.startSystem(Phaser.Physics.ARCADE);
    ball = game.add.sprite(game.world.width*0.5, game.world.height-25, 'ball');
    ball.anchor.set(0.5);
    game.physics.enable(ball, Phaser.Physics.ARCADE);
    ball.body.velocity.set(150, -150);
    ball.body.collideWorldBounds = true;
    ball.body.bounce.set(1);

    ball.checkWorldBounds = true;
    ball.events.onOutOfBounds.add(function(){
        alert('Game over!');
        location.reload();
    }, this);

    paddle = game.add.sprite(game.world.width*0.5, game.world.height-5, 'paddle');
    paddle.anchor.set(0.5,1);
    game.physics.enable(paddle, Phaser.Physics.ARCADE);
    paddle.body.immovable = true;

    initBricks();
    scoreText = game.add.text(5, 5, 'Points: 0', { font: '18px Arial', fill: '#0095DD' });
}

function update() {
    game.physics.arcade.collide(ball, paddle);
    game.physics.arcade.collide(ball, bricks, ballHitBrick);
    paddle.x = game.input.x || game.world.width*0.5;
}
```

```
function initBricks() {
    brickInfo = {
        width: 50,
        height: 20,
        count: {
            row: 7,
            col: 3
        },
        offset: {
            top: 50,
            left: 60
        },
        padding: 10
    }
    bricks = game.add.group();
    for(c=0; c<brickInfo.count.col; c++) {
        for(r=0; r<brickInfo.count.row; r++) {
            var brickX = (r*(brickInfo.width+brickInfo.padding))+brickInfo.offset.left;
            var brickY = (c*(brickInfo.height+brickInfo.padding))+brickInfo.offset.top;
            newBrick = game.add.sprite(brickX, brickY, 'brick');
            game.physics.enable(newBrick, Phaser.Physics.ARCADE);
            newBrick.body.immovable = true;
            newBrick.anchor.set(0.5);
            bricks.add(newBrick);
        }
    }
}

function ballHitBrick(ball, brick) {
    brick.kill();
    score += 10;
    scoreText.setText('Points: '+score);
    if(score === brickInfo.count.row*brickInfo.count.col*10) {
        alert('You won the game, congratulations!');
        location.reload();
    }
}
</script>
</body>
</html>
```

Exemplo de Jogo tipo Breakout usando o Framework Phaser - Fase 13

Original em inglês - https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser/Extra_lives

Esta fase trata de Vidas Extras

Podemos tornar o jogo mais agradável adicionando vidas. Neste artigo iremos implementar o sistema de vidas, de forma que o jogador começa com 3 vidas e possa continuar jogando enquanto ele não tiver perdido as 3 vidas.

Novas Variáveis
Adicionar as variáveis a seguir para as existentes:

```
var lives = 3;
var livesText;
var lifeLostText;
```

Estas respectivamente irão armazenar o número de vidas, o rótulo de texto que exibe o número de vidas que restam, e um rótulo de texto que será mostrado na tela quando o jogador perde uma das suas vidas.

Definindo os textos parece com algo que já fizemos na lição sobre pontuação. Adicione as seguintes linhas abaixo da definição scoreText existente dentro de sua função create():

```
livesText = game.add.text(game.world.width-5, 5, 'Lives: '+lives, { font: '18px Arial', fill: '#0095DD' });
livesText.anchor.set(1,0);
lifeLostText = game.add.text(game.world.width*0.5, game.world.height*0.5, 'Life lost, click to continue', { font: '18px Arial', fill: '#0095DD' });
lifeLostText.anchor.set(0.5);
lifeLostText.visible = false;
```

Os objetos livesText e lifeLostText parecem similar para scoreText - eles definem a posição na tela, o atual texto para exibir e o estilo da fonte. O mais antigo é ancorado primeiro para a margem top right para a propriedade align na tela e o último é centralizado, ambos usando anchor.set().

O lifeLostText deve ser mostrado somente quando quando a vida for perdida, sendo que sua visibilidade é inicialmente setada para false.

Tornando nosso estilo de texto DRY
Como você deve ter notado que estamos usando o mesmo estilo para todos os três textos: CoreText, livesText e lifeLostText. Se alguma vez quiser alterar o tamanho da fonte ou a cor, vamos ter que fazer isso em vários lugares. Para tornar mais fácil e prático para nós mantermos no futuro, podemos criar uma variável separada que manterá o nosso estilo, vamos chamá-lo textStyle e colocá-lo antes das definições de texto:

```
textStyle = { font: '18px Arial', fill: '#0095DD' };
```

Nós podemos agora usar essa variável quando denominando suas etiquetas de texto - atualizar seu código para que as várias instâncias do estilo de texto são substituídos com a variável:

```
scoreText = game.add.text(5, 5, 'Points: 0', textStyle);
livesText = game.add.text(game.world.width-5, 5, 'Lives: '+lives, textStyle);
livesText.anchor.set(1,0);
lifeLostText = game.add.text(game.world.width*0.5, game.world.height*0.5, 'Life lost, click to continue', textStyle);
lifeLostText.anchor.set(0.5);
lifeLostText.visible = false;
```

Desta maneira alterando a fonte apenas em um lugar estaremos alterando em todos.

Manipulando o Código das Vidas

Para implementar vidas no nosso jogo, vamos primeiro alterar a função da bola vinculado ao evento onOutOfBounds. Em vez de executar uma função anônima e mostrando o alerta de imediato:

```
ball.events.onOutOfBounds.add(function(){
    alert('Game over!');
    location.reload();
}, this);
```

Vamos atribuir uma nova função chamada ballLeaveScreen; excluir o manipulador de eventos anterior (mostrado acima) e substituí-lo com a seguinte linha:
ball.events.onOutOfBounds.add(ballLeaveScreen, this);

Queremos diminuir o número de vidas cada vez que a bola deixa a tela. Adicione a definição da função ballLeaveScreen() no final do nosso código:

```
function ballLeaveScreen() {
    lives--;
    if(lives) {
        livesText.setText('Lives: '+lives);
        lifeLostText.visible = true;
        ball.reset(game.world.width*0.5, game.world.height-25);
        paddle.reset(game.world.width*0.5, game.world.height-5);
        game.input.onDown.addOnce(function(){
            lifeLostText.visible = false;
            ball.body.velocity.set(150, -150);
        }, this);
    }
    else {
        alert('You lost, game over!');
        location.reload();
    }
}
```

Em vez de instantaneamente mostrar o alerta quando você perde uma vida, primeiro subtrair uma vida a partir do número atual e verificar se é um valor diferente de zero. Se não, então o jogador ainda tem algumas vidas restantes e pode continuar a jogar - eles vão ver a mensagem de vida perdida, as posições da bola e da raquete será reposta na tela e na próxima entrada (clique ou toque) a mensagem será escondida e a bola vai começar a se mover novamente.

Quando o número de vidas disponíveis chega a zero, o jogo acaba e a mensagem game over será exibida.

Eventos

Você já deve ter notado a chamada add () do método addOnce () nos dois blocos de código acima e se perguntou como eles diferem. A diferença é que o método add () liga-se a função dada e faz com que ele seja executado toda vez que o evento ocorre, enquanto addOnce () é útil quando você quer

ter a função ligada executado apenas uma vez e depois não ligado para que ele não é executado mais uma vez. No nosso caso, em cada evento outOfBounds o ballLeaveScreen será executado, mas quando a bola deixa a tela só queremos remover a mensagem da tela de uma vez.

Executando

Atualize o navegador e teste o sistema de pontos.

Execução online desta fase, no jsfiddle

https://jsfiddle.net/end3r/yk1c5n0b/?utm_source=website&utm_medium=embed&utm_campaign=yk1c5n0b

Exemplo de Jogo tipo Breakout usando o Framework Phaser - Fase 14

Original em inglês - https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser/Animations_and_tweens

Esta fase trata de Animações e Interpolações

Para fazer o jogo parecer mais interessante e vivo podemos usar animações e interpolações. Isto irá resultar em uma melhor experiência, mais divertido. Vamos explorar como implementar animações e interpolações Phaser em nosso jogo.

Animações
Em Phaser, animações envolvem a tomada de uma folha de sprite de uma fonte externa e exibir os sprites sequencialmente. Como exemplo, vamos fazer a bola oscilante quando atinge algo.

Efetuar o download do spritesheet - https://github.com/end3r/Gamedev-Phaser-Content-Kit/blob/gh-pages/demos/img/wobble.png
Salve-o em seu diretório / img.
Em seguida, vamos carregar o spritesheet - colocar a seguinte linha na parte inferior de sua função preload():
game.load.spritesheet('ball', 'img/wobble.png', 20, 20);

Em vez de carregar uma única imagem da bola podemos carregar toda a spritesheet - uma coleção de imagens diferentes. Vamos mostrar os sprites sequencialmente para criar a ilusão de animação. Dois parâmetros extras do método spritesheet () determinam a largura e a altura de cada quadro/frame único determinado no arquivo spritesheet, indicando ao programa como cortá-la para obter os quadros individuais.

Carregando a Animação
Depois, no início da função create(), encontrar a linha que carrega o sprite ball e abaixo deste coloque a chamada para animation.add() assim:

```
ball = game.add.sprite(50, 250, 'ball');
ball.animations.add('wobble', [0,1,0,2,0,1,0,2,0], 24);
```

Para adicionar uma animação para o objecto que utilize o método animations.add(), que contém os seguintes parâmetros:

O nome que escolhemos para a animação

Uma matriz/array que define a ordem na qual a exibir as molduras durante a animação. Se você olhar novamente para a imagem wobble.png, você vai ver que existem três quadros. Phaser extrai estes e armazena as referências então na matriz - posições 0, 1 e 2. A matriz acima diz que estamos exibindo quadro 0, 1, então 0, etc.

A framerate, em fps. Uma vez que estamos executando a animação em 24fps e há 9 quadros, a animação irá mostrar pouco menos de três vezes por segundo.

Aplicando a Animação Quando a Bola Toca a Raquete

Na chamada do método arcade.collide() que manipula a colisão entre a bola e a raquete (a primeira linha dentro update (), veja abaixo), podemos adicionar um parâmetro adicional que especifica uma função a ser executada toda vez que a colisão acontecer. Atualize a primeira linha dentro update () como mostrado abaixo:

```
function update() {
   game.physics.arcade.collide(ball, paddle, ballHitPaddle);
   game.physics.arcade.collide(ball, bricks, ballHitBrick);
   paddle.x = game.input.x || game.world.width*0.5;
}
```

Então, podemos criar a função ballHitPaddle() (com bola e raquete como parâmetros padrões), jogando a animação oscilação quando é chamado. Adicione a seguinte função pouco antes do tag </ script>:

```
function ballHitPaddle(ball, paddle) {
   ball.animations.play('wobble');
}
```

A animação é jogada cada vez que a bola tocar na raquete. Você pode adicionar a chamada animations.play() dentro da função ballHitPaddle() também, se você sente que faria o jogo mais bonito.

Tweens/Interpolações

Considerando animações como jogar sprites externos sequencialmente, propriedades tweens/interpolam de um objeto suaviza animações no mundo do jogo, como a largura ou opacidade.

Vamos adicionar uma interpolação para o nosso jogo para fazer os tijolos sem problemas desaparecerem quando eles são atingidos pela bola. Vá para a sua ballHitBrick(), procure o seu brick.kill (); linha, e substituí-lo com o seguinte:

```
var killTween = game.add.tween(brick.scale);
killTween.to({x:0,y:0}, 200, Phaser.Easing.Linear.None);
killTween.onComplete.addOnce(function(){
```

```
    brick.kill();
}, this);
killTween.start();
```

Vamos examinar isso para que você possa ver o que está acontecendo aqui:
1 - Ao definir uma nova interpolação você tem que especificar quais propriedades serão interpolados - no nosso caso, em vez de esconder os tijolos instantaneamente quando atingido pela bola, vamos fazer a sua largura e altura escala de zero, então eles vão muito bem desaparecer. Para o fim, usamos o método add.tween (), especificando brick.scale como o argumento como este é o que nós queremos de interpolação.
2 - O método to() define o estado do objecto na extremidade da interpolação. É preciso um objeto que contém os parâmetros escolhidos desejados valores final (escala tem um valor de escala, sendo 1 100% de tamanho, sendo 0 0% do tamanho, etc.), o tempo da interpolação em milissegundos e o tipo de atenuação de usar para a interpolação.
3 - Nós também vamos adicionar o manipulador de eventos onComplete opcional, que define uma função a ser executada quando a interpolação termina.
4 - A última coisa que fazer para é começar a interpolação imediatamente usando start ().

Essa é a versão expandida da definição tween, mas também podemos usar a sintaxe abreviada:

```
game.add.tween(brick.scale).to({x:2,y:2}, 500, Phaser.Easing.Elastic.Out, true, 100);
```

Este tween vai dobrar a escala do tijolo ao meio segundo usando flexibilização Elastic, inicia-se automaticamente, e têm um atraso de 100 milissegundos.

Executando

Atualize o navegador e teste o sistema de pontos.

Execução online desta fase, no jsfiddle

https://jsfiddle.net/end3r/9o4pakrb/?utm_source=website&utm_medium=embed&utm_campaign=9o4pakrb

Exemplo de Jogo tipo Breakout usando o Framework Phaser - Fase 15

Original em inglês - https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser/Buttons

Esta fase trata de Botões

Ao invés de iniciar o jogo automaticamente sempre que abrimos a página é melhor adicionar um botão e permitir que o jogador comece quando desejar. Aqui vamos implementar isso.

Novas Variáveis
Vamos precisar de uma variável para armazenar um valor booleano representando se o jogo está

sendo jogado ou não, e outra para representar o nosso botão. Adicione estas linhas abaixo seus outros definições de variáveis:

```
var playing = false;
var startButton;
```

Carragando o Spritesheet do Botão
Efetuando o download - https://github.com/end3r/Gamedev-Phaser-Content-Kit/blob/gh-pages/demos/img/button.png

Nós podemos carregar a folha do sprite do botão da mesma forma que carregado na animação a oscilação da bola. Adicionar o seguinte para a parte inferior da função de preload():

```
game.load.spritesheet('button', 'img/button.png', 120, 40);
```

Um único frame do botão tem 120 pixels de largura e 40 pixels de altura.

Adicione o Botão para o Jogo
Adicionar o novo botão para o jogo é feito usando o método add.button. Adicione as seguintes linhas para o fundo da sua função create ():

```
startButton = game.add.button(game.world.width*0.5, game.world.height*0.5, 'button', startGame, this, 1, 0, 2);
startButton.anchor.set(0.5);
```

Os parâmetros do método botão () são os seguintes:

- x e y coordenadas do botão
- O nome do ativo gráfico a ser exibido para o botão
- Uma função callback que será executada quando o botão é pressionado
- A referência à presente para especificar o contexto de execução
- Os quadros que vão ser utilizados para a mais, para fora e para baixo eventos.

Nota: O evento over é o mesmo que o hover, é quando o ponteiro se move para fora do botão e para baixo é quando o botão é pressionado.

Agora nós vamos definir o método startGame() :

```
function startGame() {
   startButton.destroy();
   ball.body.velocity.set(150, -150);
   playing = true;
}
```

Quando o botão é pressionado, nós removemos o botão, define a velocidade inicial da bola e definir a variável de jogo para true.

Finalmente para esta seção, volte para a sua função create(), encontre a linha ball.body.velocity.set

(150, -150); e remova-o. Você só quer que a bola se mova quando o botão for pressionado, não antes!

Mantendo a raquete antes do jogo começar
Ele funciona como esperado, mas ainda podemos mover a raquete quando o jogo ainda não começou, o que parece um bocado parvo. Para acabar com isso, podemos tirar proveito da variável de jogo e fazer o movimento da raquete somente quando o jogo começar. Para fazer isso, ajustar a função update() assim:

```
function update() {
    game.physics.arcade.collide(ball, paddle, ballHitPaddle);
    game.physics.arcade.collide(ball, bricks, ballHitBrick);
    if(playing) {
        paddle.x = game.input.x || game.world.width*0.5;
    }
}
```

Executando

Atualize o navegador e teste o sistema de pontos.

Execução online desta fase, no jsfiddle

https://jsfiddle.net/end3r/1rpj71k4/?utm_source=website&utm_medium=embed&utm_campaign=1rpj71k4

Exemplo de Jogo tipo Breakout usando o Framework Phaser - Fase 16

Original em inglês - https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_breakout_game_Phaser/Randomizing_gameplay

Esta fase trata de Randomizações

Nosso jogo será finalizando nesta fase. Veja que a bola está rebatendo na raquete sempre sempre com o mesmo ângulo. Assim todas as partidas serão semelhantes. Para mudar isso e tornar o jogo mais atrativo nós mudaremos o ângulo que a bola rebate na raquete de forma randômica/aleatória. Este artigo trata disso.

Tornando os Rebotes mais Aleatórios
Nós podemos mudar a velocidade da bola, dependendo do ponto exato que bate na raquete, modificando o a velocidade x a cada vez que a função ballHitPaddle() for executada utilizando uma linha ao longo das linhas de baixo. Adicionar esta nova linha ao seu código agora, e experimentá-lo.

```
function ballHitPaddle(ball, paddle) {
    ball.animations.play('wobble');
```

```
    ball.body.velocity.x = -1*5*(paddle.x-ball.x);
}
```

É um pouco de magia - a nova velocidade é maior, quanto maior a distância entre o centro da raquete e do local onde a bola batê-la. Além disso, a direção (esquerda ou direita) é determinada por esse valor - se a bola bate no lado esquerdo da raquete ela vai saltar para a esquerda, enquanto batendo no lado direito vai saltar para a direita. Ele acabou assim por causa de um pouco de experimentação com os valores dados, você pode fazer sua própria experimentação e ver o que acontece. Não é completamente aleatório, é claro, mas faz o jogo um pouco mais imprevisível e, portanto, mais interessante.

Executando

Atualize o navegador e teste o sistema de pontos.

Execução online desta fase, no jsfiddle

https://jsfiddle.net/end3r/3yds5ege/?utm_source=website&utm_medium=embed&utm_campaign=3yds5ege