Este artigo foi escrito por Alvin Ourrad e Richard Davey. Recentemente atualizado para Phaser 2.0! Bem-vindo ao nosso primeiro tutorial sobre Como fazer um jogo com a Phaser. Aqui vamos aprender a criar um pequeno jogo envolvendo um jogador correndo e pulando em torno de plataformas coletando estrelas. Ao passar por este processo, vamos explicar alguns dos principais recursos do framework.

O que é Phaser?

A Phaser é uma estrutura de jogo HTML5 que tem como objetivo ajudar os desenvolvedores a fazerem os poderosos jogos HTML5 com vários navegadores muito rapidamente e, ao contrário de outros, foi construído apenas para trabalhar com os navegadores móveis. O único requisito do navegador é o suporte da tag de tela. Também empresta muito de Flixel.

Requisitos

- Você precisa ter um conhecimento muito, muito básico de JavaScript.
- Certifique-se também de que você passa pelo Guia de Introdução, ele mostrará como fazer o download da estrutura, configurar um ambiente de desenvolvimento local e dar-lhe um vislumbre da estrutura de um projeto Phaser e suas funções principais.

Se você já passou pelo Guia de Introdução, terá baixado o Phaser e terá tudo configurado e pronto para codificar. Baixe os recursos para este tutorial a partir daqui e descompacte-o em sua raiz da web. Abra a página part1.html em seu editor de escolha e vamos ter um olhar mais atento sobre o código. Depois de um pouco HTML predefinido que inclui Phaser a estrutura de código se parece com isto:

```
Var = novo Phaser.Game (800, 600, Phaser.AUTO, '', {preload: preload, create: create, update: update});

Função pré-carga () {
}

Function create () {
}

Function update () {
}
```

Linha 1 é onde você traz Phaser à vida, criando uma instância de um objeto Phaser.Game e atribuindo-o a uma variável local chamada 'game'. Chamá-lo de "jogo" é uma prática comum, mas não uma exigência, e é isso que você encontrará nos exemplos da Phaser. Os dois primeiros parâmetros são a largura ea altura do elemento de tela que a Phaser irá criar. Neste caso 800 x 600 pixels. Seu mundo de jogo pode ser qualquer tamanho que você gosta, mas esta é a resolução do jogo será exibido polegadas O terceiro parâmetro pode ser Phaser.CANVAS, Phaser.WEBGL ou Phaser.AUTO. Esse é o contexto de renderização que você deseja usar. O parâmetro recomendado é Phaser.AUTO que tenta automaticamente usar o WebGL, mas se o navegador ou o dispositivo não o suportar, ele voltará para o Canvas. O quarto parâmetro é uma string vazia, Este é o id do elemento DOM no qual você gostaria de inserir o elemento de tela que a Phaser cria. Como deixamos em branco, ele será simplesmente anexado ao corpo. O parâmetro final é um objeto contendo quatro referências a funções essenciais de Phasers. Seu uso é completamente explicado aqui . Observe que este objeto não é necessário - Phaser suporta um sistema de estado completo permitindo que você quebre seu código em objetos muito mais limpos. Mas para um guia de introdução simples como este, usaremos essa abordagem, pois permite uma prototipagem mais rápida. Seu uso é completamente explicado aqui . Observe que este objeto não é necessário - Phaser suporta um sistema de estado completo permitindo que você quebre seu código em objetos muito mais limpos. Mas para um guia de introdução simples como este, usaremos essa abordagem, pois permite uma prototipagem mais rápida. Seu uso é completamente explicado aqui . Observe que este objeto não é necessário - Phaser suporta um sistema de estado completo permitindo que você quebre seu código em objetos muito mais limpos. Mas para um guia de introdução simples como este, usaremos essa abordagem, pois permite uma prototipagem mais rápida.

Carregar ativos

Vamos carregar os ativos que precisamos para o nosso jogo. Você faz isso colocando chamadas para game.load dentro de uma função chamada preload. A Phaser procurará automaticamente esta função quando iniciar e carregar qualquer coisa definida dentro dela. Atualmente, a função de pré-carga está vazia. Altere-o para:

```
Função pré-carga () {

    Game.load.image ('sky', 'assets / sky.png');
    Game.load.image ('terra', 'assets / platform.png');
    Game.load.image ('estrela', 'assets / star.png');
    Game.load.spritesheet ('dude', 'assets / dude.png', 32, 48);

}
```

Isso carregará em 4 ativos: 3 imagens e uma folha de sprite. Pode parecer óbvio para alguns de vocês, mas gostaria de salientar o primeiro parâmetro, também conhecido como a chave de imobilizado. Esta seqüência é um link para o recurso carregado e é o que você usará em seu código ao criar sprites. Você é livre para usar qualquer seqüência de caracteres JavaScript válida como a chave.

Criar um Sprite

Para adicionar um sprite ao nosso jogo, coloque o seguinte código na função create:

```
Game.add.sprite (0, 0, 'estrela');
```

Se você abrir a página em um navegador, agora deverá ver uma tela de jogo preta com um único sprite estrela no canto superior esquerdo:

A ordem na qual os itens são exibidos na tela corresponde à ordem em que os criou. Então, se você deseja colocar um fundo atrás do sprite estrelas que você precisa para garantir que ele foi adicionado como um sprite primeiro, antes da estrela.

Edifício Mundial

Sob o capô game.add.sprite está criando um novo objeto Phaser.Sprite e adicionando o sprite ao "mundo do jogo". Este mundo é onde todos os seus objetos vivem, ele pode ser comparado ao Palco em Actionscript3. **Nota:** O mundo do jogo não tem tamanho fixo e se estende infinitamente em todas as direções, com 0,0 sendo o centro dele. Por conveniência Phaser coloca 0,0 na parte superior esquerda do seu jogo para você, mas usando a câmera embutida você pode se movimentar conforme necessário. A classe mundial pode ser acessada via game.world e vem com um monte de métodos úteis e propriedades para ajudá-lo a distribuir seus objetos dentro do mundo. Ele inclui algumas propriedades simples como game.world.height, mas também alguns mais avançados que usaremos em outro tutorial. Por enquanto, S acumular a cena, adicionando um fundo e plataformas. Aqui está a função de criação atualizada:

```
Var plataformas;

Function create () {

    // Vamos usar física, então habilite o sistema Arcade Physics
```

```
    Game.physics.startSystem (Phaser.Physics.ARCADE);

    // Um fundo simples para o nosso jogo
    Game.add.sprite (0, 0, 'céu');

    // O grupo de plataformas contém o solo e as 2 saliências que podemos saltar sobre
    Platform = game.add.group ();

    // Vamos habilitar a física para qualquer objeto que é criado neste grupo
    Platforms.enableBody = true;

    // Aqui criamos o solo.
    Var ground = platforms.create (0, game.world.height - 64, 'terra');

    // Dimensioná-lo para ajustar a largura do jogo (o sprite original é 400x32 em tamanho)
    Ground.scale.setTo (2, 2);

    // Isso impede que ele caia quando você pula nele
    Ground.body.immovable = true;

    // Agora vamos criar duas bordas
    Var ledge = platforms.create (400, 400, 'terra');

    Ledge.body.immovable = true;

    Ledge = platforms.create (-150, 250, 'terra');

    Ledge.body.immovable = true;

}
```

Se você executar este, que você encontrará como part4.html no arquivo zip tutorial, você deve ver uma cena muito mais parecida com o jogo:

A primeira parte é o mesmo que o sprite estrela que tínhamos antes, só que em vez disso, mudou o Chave para 'céu' e ele exibiu nosso fundo de céu em vez disso. Este é um PNG 800x600 que preenche a tela do jogo.

Grupos

Os grupos são realmente poderosos. Como seu nome indica, permitem agrupar objetos semelhantes e controlá-los como uma única unidade. Você também pode verificar a colisão entre grupos, e para este jogo vamos usar dois grupos diferentes, um dos quais é criado no código acima para as plataformas.

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

Como com os sprites game.add cria nosso objeto Group. Nós o atribuímos a uma nova variável local chamada plataformas. Agora criados, podemos adicionar objetos a ele. Primeiro é o chão. Isso é posicionado na parte inferior do jogo e usa a imagem 'ground' carregada anteriormente. O solo é dimensionado para preencher a largura do jogo. Finalmente, definimos sua propriedade 'immovable' como true. Se não tivéssemos feito isso, o chão se moveria quando o jogador colidisse com ele (mais sobre isso na seção de Física). Com o chão no lugar, criamos duas bordas menores para saltar sobre a usar a mesma técnica exata que para o solo.

Pronto Jogador Um

Crie uma nova variável local chamada 'player' e adicione o seguinte código para a função create. Você pode ver isso em part5.html:

```
    // O player e suas configurações
    Player = game.add.sprite (32, game.world.height - 150, 'dude');

    // Precisamos ativar física no player
    Game.physics.arcade.enable (jogador);

    // Propriedades da física do jogador. Dê ao rapaz um ligeiro salto.
    Player.body.bounce.y = 0,2;
    Player.body.gravity.y = 300;
    Player.body.collideWorldBounds = true;

    // Nossas duas animações, andando para a esquerda e para a direita.
    Player.animations.add ('esquerda', [0, 1, 2, 3], 10, true);
    Player.animations.add ('direito', [5, 6, 7, 8], 10, verdadeiro);
```

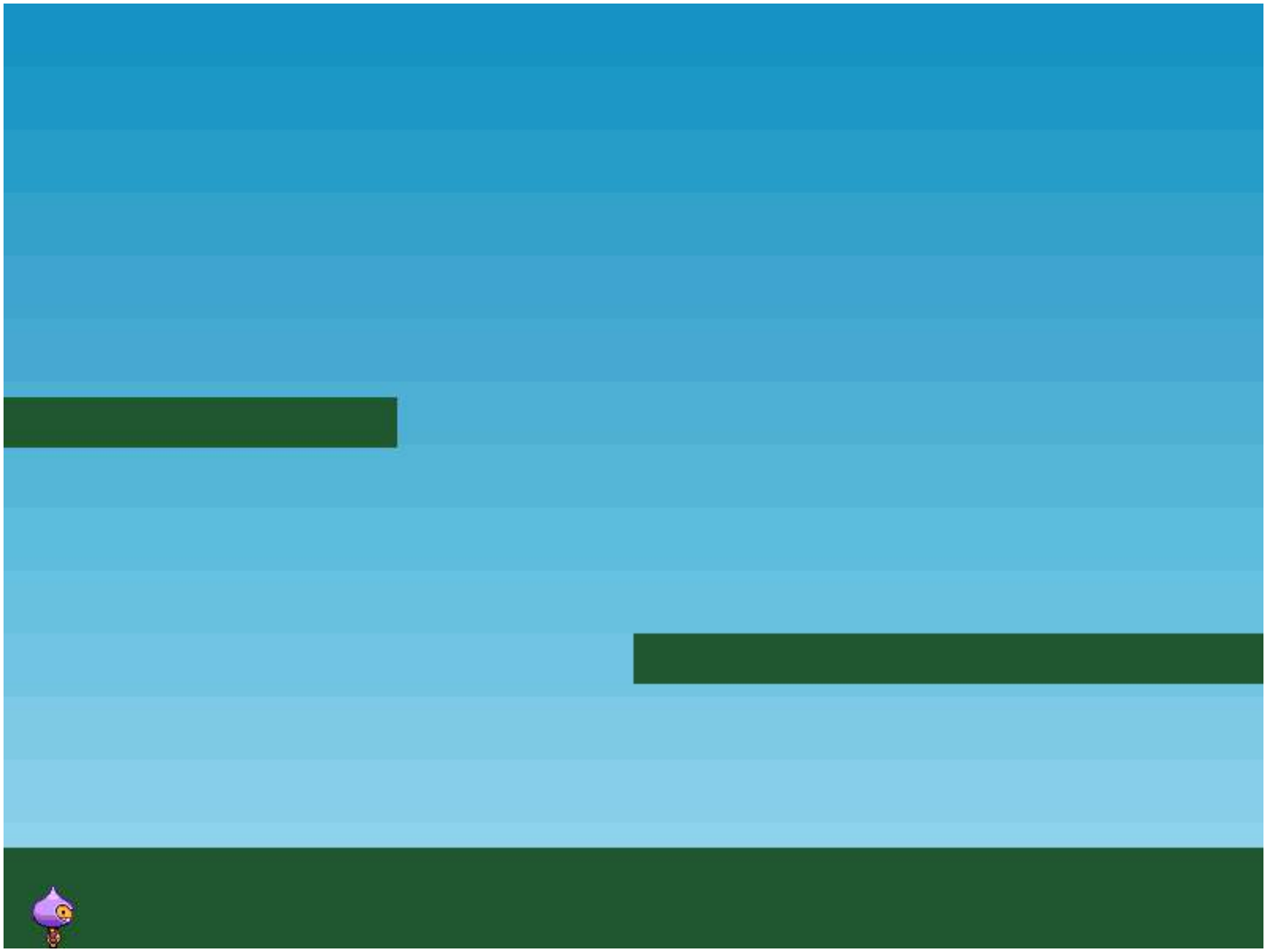
Isso cria um novo sprite chamado "player", posicionado em 32 pixels por 150 pixels da parte inferior do jogo. Estamos dizendo a ele para usar o 'gajo' ativo anteriormente carregado. Se você olhar para trás para a função de pré-carregamento você verá que 'dude' foi carregado como uma folha de sprite, não uma imagem. Isso é porque ele contém quadros de animação. Isto é o que a folha cheia do sprite olha como:  Você pode ver 9 frames no total, 4 para funcionar para a esquerda, 1 para enfrentar a câmera e 4 para funcionar para a direita. Nota: A Phaser suporta lançar sprites para salvar em quadros de animação, mas por causa deste tutorial vamos mantê-lo na velha escola. Definimos duas animações chamadas 'esquerda' e 'direita'. A animação 'esquerda' usa os quadros 0, 1, 2 e 3 e é executada em 10 quadros por segundo. A verdade' Parâmetro diz a animação para loop. Este é o nosso ciclo de corrida padrão e repetimos-o para correr na direção oposta. Com o conjunto de animações criamos algumas propriedades físicas.

O corpo e a velocidade: um mundo da física

A Phaser tem suporte para uma variedade de sistemas físicos diferentes. Ele é fornecido com Arcade Physics, Ninja Physics e P2.JS Full-Body Physics. Para o bem deste tutorial vamos usar o sistema Arcade Physics, que é simples e leve, perfeito para navegadores móveis. Você vai notar no código que temos de iniciar o sistema de física em execução e, em seguida, para cada sprite ou grupo que desejamos usar a física em que habilitá-los. Uma vez feito o sprites ganhar uma nova propriedade do corpo, que é uma instância de ArcadePhysics.Body . Isso representa o sprite como um corpo físico no motor Phasers Arcade Physics. O corpo objeto tem um monte de propriedades que podemos jogar. Para simular os efeitos da gravidade em um sprite, é tão simples como escrever isso:

```
Player.body.gravity.y = 300;
```

Este é um valor arbitrário, mas, logicamente, quanto maior o valor, mais pesado seu objeto se sente e mais rápido ele cai. Se você adicionar isso ao seu código ou executar part5.html você verá que o jogador cai sem parar, ignorando completamente o chão que criamos anteriormente:



A razão para isso é que ainda não estamos testando a colisão entre o solo eo jogador. Nós já dissemos Phaser que nosso chão e bordas seriam imóveis. Se não tivéssemos feito isso quando o jogador colidisse com eles pararia por um momento e então tudo teria desmoronado. Isso ocorre porque, a menos que seja dito o contrário, o sprite terrestre é um objeto físico em movimento (também conhecido como corpo dinâmico) e quando o jogador o atinge, a força resultante da colisão é aplicada ao solo, Os dois corpos trocam suas velocidades e começa a cair chão também. Assim, para permitir que o jogador colidir e tirar proveito das propriedades físicas que precisamos para introduzir uma verificação de colisão na função de atualização:

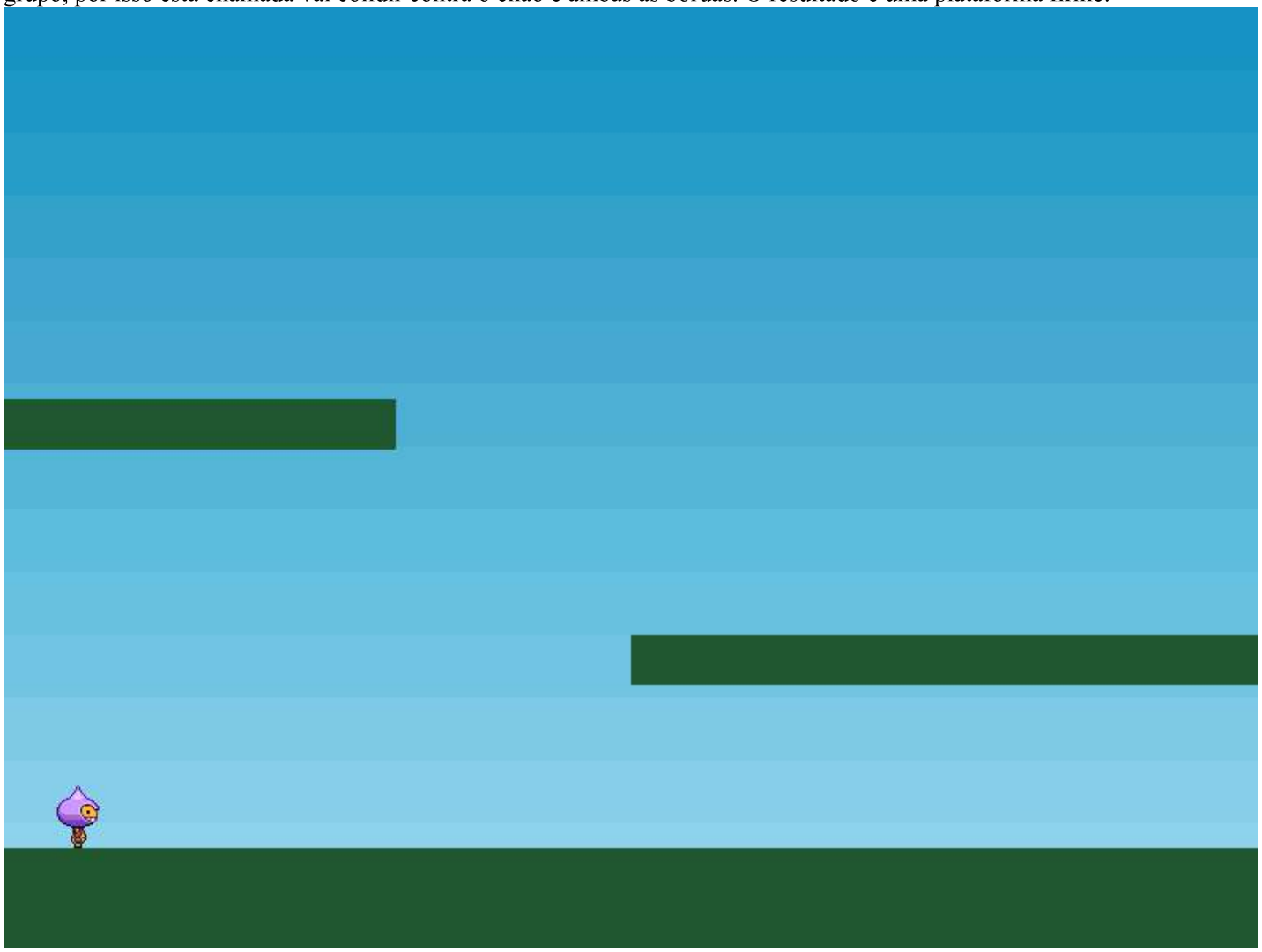
```
Function update () {

    // colidir o jogador e as estrelas com as plataformas
    Game.physics.arcade.collide (jogador, plataformas);

}
```

A função de atualização é chamada pelo loop do núcleo do jogo em cada quadro. A função Physics.collide é aquela que realiza a mágica. É preciso dois objetos e testes de colisão e realiza a separação contra eles. Neste caso, estamos dando-lhe o jogador sprite e as plataformas do Grupo. É inteligente o suficiente para executar colisão contra todos os membros do

grupo, por isso esta chamada vai colidir contra o chão e ambas as bordas. O resultado é uma plataforma firme:



Controle do player com o teclado

Colidir está tudo bem e bem, mas nós realmente precisamos que o jogador se mova. Você provavelmente pensaria em ir para a documentação e pesquisar sobre como adicionar um ouvinte de eventos, mas isso não é necessário aqui. Phaser tem um built-in Keyboard gerente e um dos benefícios de usar que é esta pequena função acessível:

```
Cursors = game.input.keyboard.createCursorKeys ();
```

Isso preenche o objeto cursores com quatro propriedades: cima, baixo, esquerda, direita, que são todas as instâncias de objetos Phaser.Key . Então tudo o que precisamos fazer é pesquisar em nosso loop de atualização:

```
    // Redefinir a velocidade dos jogadores (movimento)
    Player.body.velocity.x = 0;

    If (cursors.left.isDown)
    {
        // Move para a esquerda
        Player.body.velocity.x = -150;

        Player.animations.play ('esquerda');
    }
    Else if (cursors.right.isDown)
    {
        // Mover para a direita
        Player.body.velocity.x = 150;

        Player.animations.play ('direito');
    }
    outro
    {
        // Fique parado
        Player.animations.stop ();

        Player.frame = 4;
```

```
    }

    // Permite que o jogador salte se tocar no chão.
    If (cursors.up.isDown && player.body.touching.down)
    {
        Player.body.velocity.y = -350;
    }
```

Embora tenhamos adicionado um monte de código, tudo deve ser bastante legível. A primeira coisa que fazemos é redefinir a velocidade horizontal no sprite. Em seguida, verificamos se a tecla do cursor esquerdo é mantida pressionada. Se for nós aplicamos uma velocidade horizontal negativa e começamos a animação em execução 'esquerda'. Se eles estão segurando 'certo' em vez disso, literalmente, fazer o oposto. Desobstruindo a velocidade e ajustando-a desta maneira, cada frame, cria um estilo do "stop-start" do movimento. O sprite do jogador irá mover-se apenas quando uma tecla é mantida pressionada e parar imediatamente, eles não são. Phaser também permite que você crie movimentos mais complexos, com momentum e aceleração, mas isso nos dá o efeito que precisamos para este jogo. A parte final da verificação de teclas define a moldura para 4 se nenhum keyis for mantido pressionado.

Ir para ele

A parte final do código adiciona a capacidade de saltar. O cursor para cima é a nossa tecla de salto e nós testamos se isso é para baixo. No entanto, também testar se o jogador está tocando o chão, caso contrário, eles poderiam saltar, enquanto no ar. Se ambas as condições forem satisfeitas, aplicaremos uma velocidade vertical de 350 px / seg². O jogador cairá automaticamente para o solo devido ao valor de gravidade que aplicamos a ele. Com os controles no lugar, agora temos um mundo de jogo que podemos explorar. Carregar part7.html e ter um jogo. Experimente ajustar valores como o 350 para o salto para valores mais baixos e mais altos para ver o efeito que terá.

Starshine

É hora de dar nosso pequeno jogo um propósito. Vamos deixar cair uma aspersão de estrelas na cena e permitir que o jogador as colete. Para conseguir isso, criaremos um novo grupo chamado "estrelas" e o preencheremos. Em nossa função criar adicionamos o seguinte código (isso pode ser visto em part8.html):

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

    // Aqui vamos criar 12 deles uniformemente espaçados
    Para (var i = 0; i <12; i ++)
    {
        // Criar uma estrela dentro do grupo 'estrelas'
        Var star = stars.create (i * 70, 0, 'estrela');

        // Deixe a gravidade fazer a sua coisa
        Star.body.gravity.y = 6;

        // Isto apenas dá a cada estrela um valor de rejeição ligeiramente aleatório
        Star.body.bounce.y = 0,7 + Math.random () * 0,2;
    }
```

O processo é semelhante ao quando criamos o Grupo de plataformas. Usando um JavaScript 'para' loop nós dizemos para criar 12 estrelas em nosso jogo. Eles têm uma coordenada x de i * 70, o que significa que eles serão uniformemente espaçados na cena de 70 pixels de distância. Tal como acontece com o jogador, damos-lhes um valor de gravidade para que eles caem, e um valor de rejeição para que eles vão saltar um pouco quando eles batem as plataformas. Bounce é um valor entre 0 (nenhum salto em tudo) e 1 (um salto completo). A nossa vai saltar em algum lugar entre 0,7 e 0,9. Se estivéssemos a executar o código como este as estrelas iria cair através do fundo do jogo. Para parar, precisamos verificar sua colisão contra as plataformas em nosso loop de atualização:

```
Game.physics.arcade.collide (estrelas, plataformas);
```

Além de fazer isso, também verificaremos se o jogador se sobrepõe a uma estrela ou não:

```
Game.physics.arcade.overlap (jogador, estrelas, collectStar, null, this);
```

Isso informa a Phaser para verificar se há uma sobreposição entre o jogador e qualquer estrela no grupo de estrelas. Se for encontrado, passe-os para a função 'collectStar':

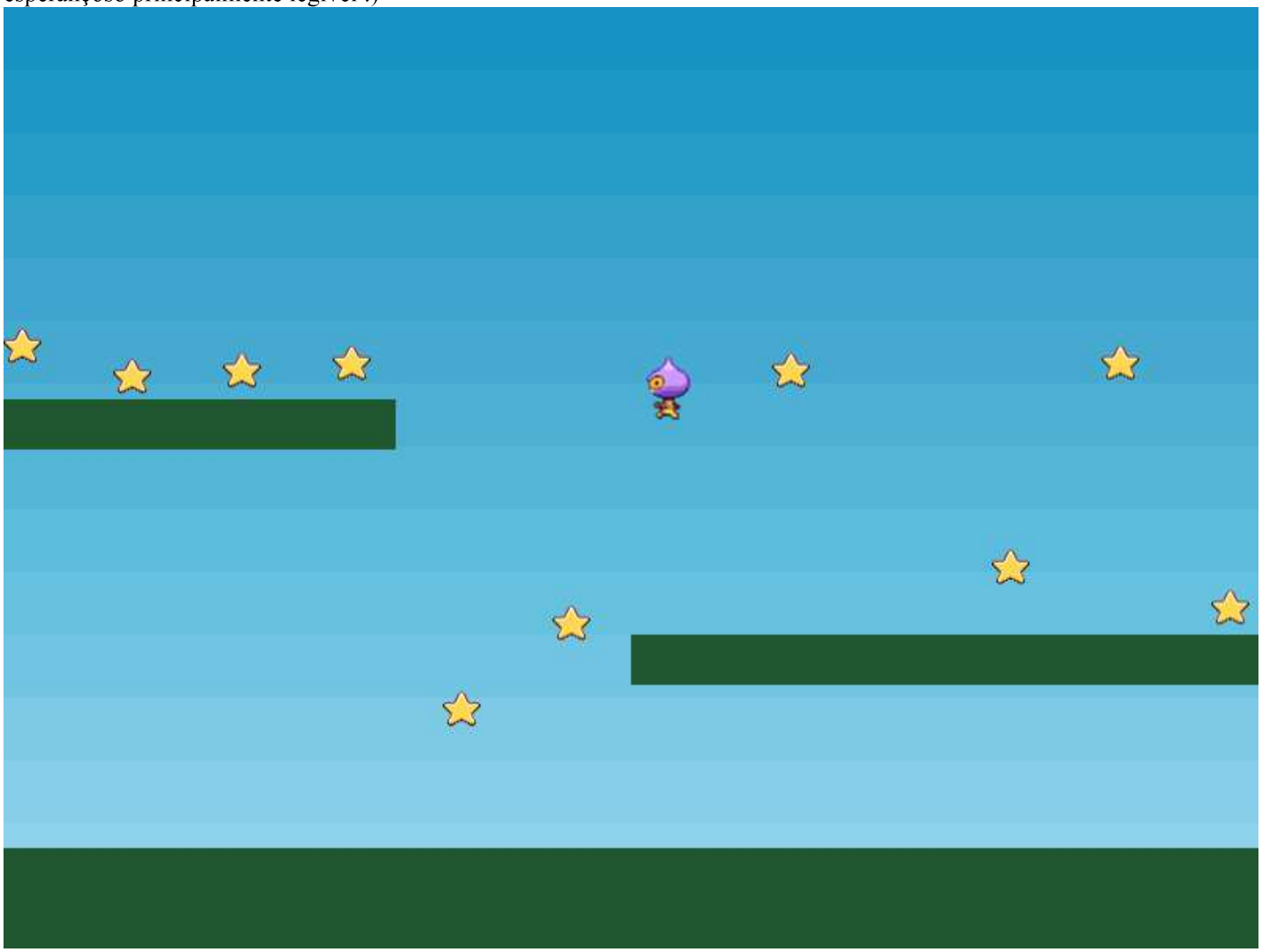
```
Função collectStar (jogador, estrela) {

    // Remove a estrela da tela
    Star.kill ();

}
```

Simplesmente a estrela é morta, o que o remove da tela. Executando o jogo agora nos dá um jogador que pode traço sobre, saltar, saltando fora das plataformas e recolher as estrelas que caem de cima. Não é ruim para algumas linhas de código

esperançoso principalmente legível :)



Toques finais

O ajuste final que faremos é adicionar uma pontuação. Para fazer isso, usaremos um objeto Phaser.Text . Aqui nós criamos duas novas variáveis, uma para manter a pontuação real eo objeto de texto em si:

```
Pontuação var = 0;
Var scoreText;
```

O scoreText está configurado na função create:

```
ScoreText = game.add.text (16, 16, 'pontuação: 0', {fontSize: '32px', fill: '# 000'});
```

16x16 é a coordenada para exibir o texto em. 'Pontuação: 0' é a seqüência padrão a ser exibida eo objeto a seguir contém um tamanho de fonte e cor de preenchimento. Ao não especificar qual fonte usaremos, o navegador será padrão, então no Windows será Arial. Em seguida, precisamos modificar a função collectStar para que quando o jogador pega uma estrela sua pontuação aumenta e o texto é atualizado para refletir isso:

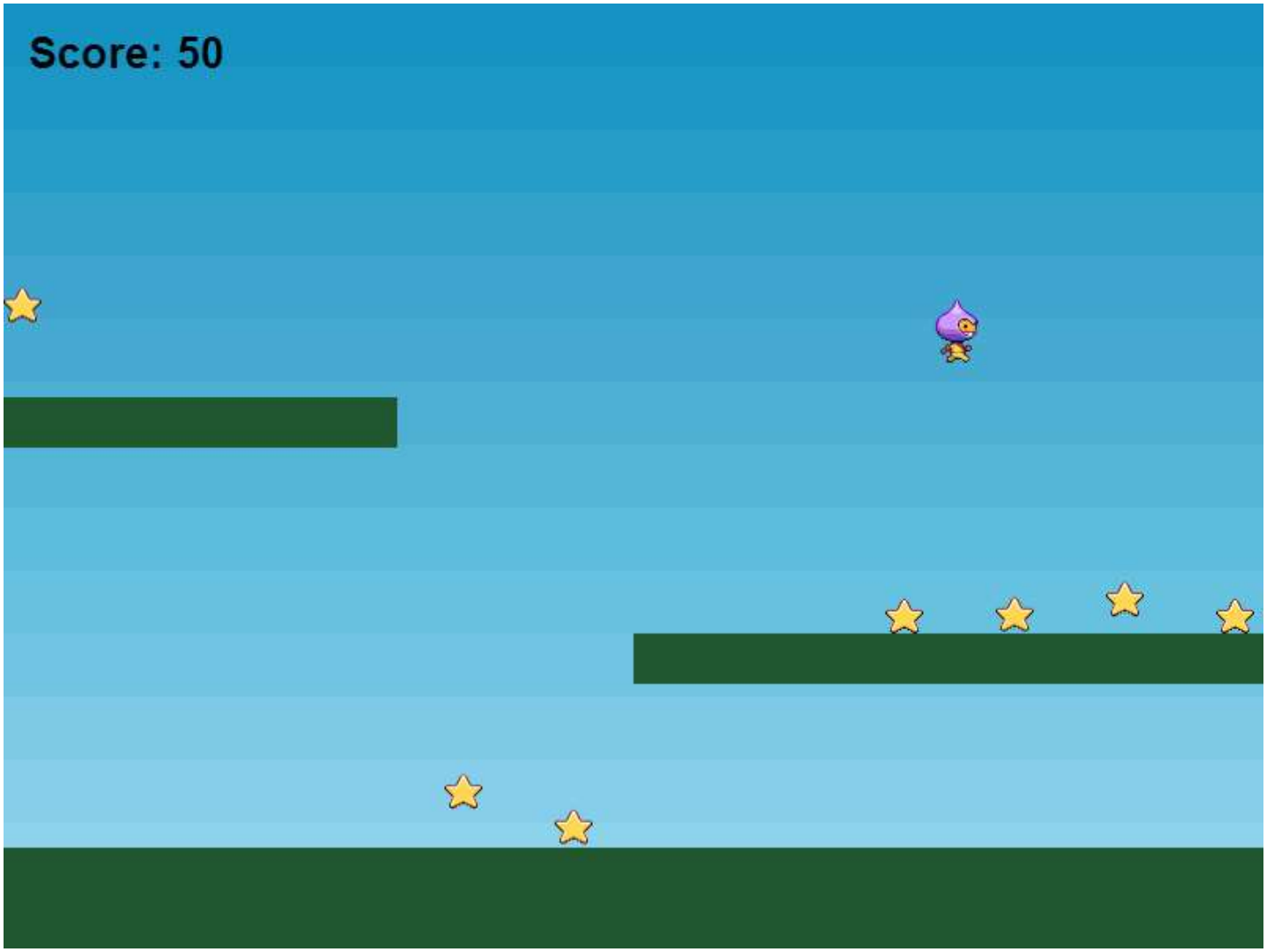
```
Função collectStar (jogador, estrela) {

    // Remove a estrela da tela
    Star.kill ();

    // Adicionar e atualizar a pontuação
    Pontuação + = 10;
    ScoreText.content = 'Pontuação:' + pontuação;

}
```

Então, 10 pontos são adicionados para cada estrela eo scoreText é atualizado para mostrar este novo total. Se você executar part9.html você verá o jogo final.



Conclusão

Você agora aprendeu a criar um sprite com propriedades físicas, para controlar seu movimento e fazê-lo interagir com outros objetos em um mundo de jogo pequeno. Há muito mais coisas que você pode fazer para melhorar isso, por exemplo, não há sentido de conclusão ou perigo ainda. Por que não adicionar alguns pontos que você deve evitar? Você pode criar um novo grupo de "picos" e verificar a colisão contra o jogador, apenas em vez de matar o sprite spike você mata o jogador em vez disso. Ou para um jogo de estilo não-violento você poderia torná-lo uma corrida de velocidade e simplesmente desafiá-los a recolher as estrelas o mais rapidamente possível. Incluímos alguns gráficos extras no arquivo zip para ajudar a inspirar você. Com a ajuda do que você aprendeu neste tutorial e os mais de 250 exemplos disponíveis para você, Você deve agora ter uma base sólida para um projeto futuro. Mas, como sempre, se você tiver dúvidas, precisar de conselhos ou quiser compartilhar o que você está trabalhando, sinta-se à vontade para pedir ajuda no fórum Phaser . Mais tutoriais seguirão, fique atento :)