Web Version Unsubscribe

Welcome to Issue 136 of Phaser World

When I write about games, in order to add them to the Phaser web site, I, of course, have to play them first. Which is always interesting, because I love seeing what people have made with Phaser. But, when you have such a superb game as Unikitty Saves the Kingdom, it's easy to get utterly sucked in, and before you know it, an hour has gone by! Definitely an hour well spent, though :)

There are also new tutorials and a Dev Log all about the Phaser 3 Scale Manager, so be sure to read this newsletter on the web so you don't miss anything.

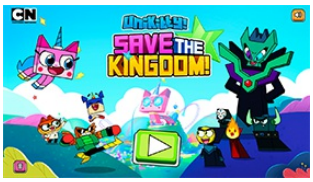
If you've a game or tutorial you'd like featured then simply reply to this email, or messaging me on Slack, Discord or Twitter. Until the next issue, keep on coding!

Fresh batch of Phaser Sticker Packs arrived

Due to their popularity, I've had another batch of the Phaser stickers printed, so there's now plenty in stock. The next 100 orders will get a **Phaser pin-badge** and a new **pixel-art sticker** included as well, for no extra charge.

Click here to order a sticker pack.

The Latest Games



Game of the Week

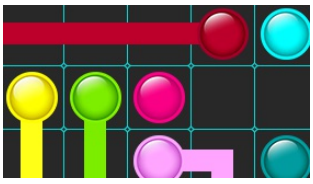
Unikitty - Save the Kingdom!

Can Unikitty and her friends save the Kingdom from the Doom Lords? in this fantastic platformer-action game.

Staff Pick

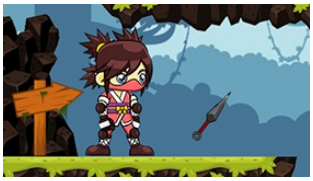
Toby's Adventures

Guide the fox to collect all the pizza, while using the special powers to smash and jump through this puzzle platform game.



Flow Mania

The problem is simple: Just connect the dots of the same color, creating a flow between them. However, they cannot overlap!



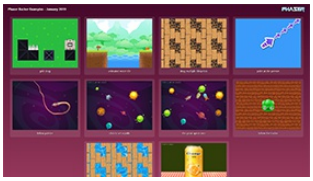
Minako vs. Zombies

Ninjas, saving the world from a zombie invasion? What more could you ask for?!

Mini-O-Stars

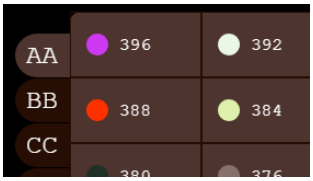
Bounce around the screen, collecting stars and avoiding the baddies in this Minions / Star Wars cross-over.

What's New?



Phaser Backer Examples January 2019

The January Phaser Backer Examples are now available including animated Tile Sprites and Pointer followers.

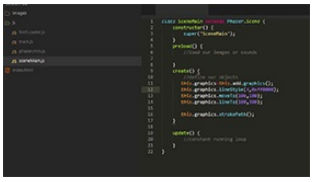


rexUI Plugins

A powerful set of UI plugins for Phaser 3 that includes Grid Tables, Textboxes, Pop-up Menus, Labels and many more.

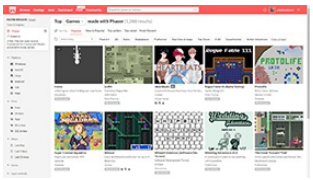
Creating a Phaser 3 Template

Part 1 of a tutorial on creating a common Phaser template that you can re-use across all of your projects.



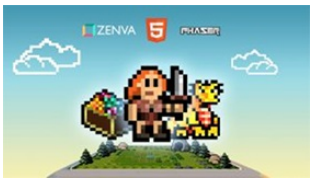
Introduction to Phaser 3 Graphics

A video introduction to the Phaser 3 Graphics Game Object and how to use it.



itch.io games made with Phaser

There are over 1280 games made with Phaser on itch.io. Here is how to add yours to the list.



Phaser 3 Game Development Course

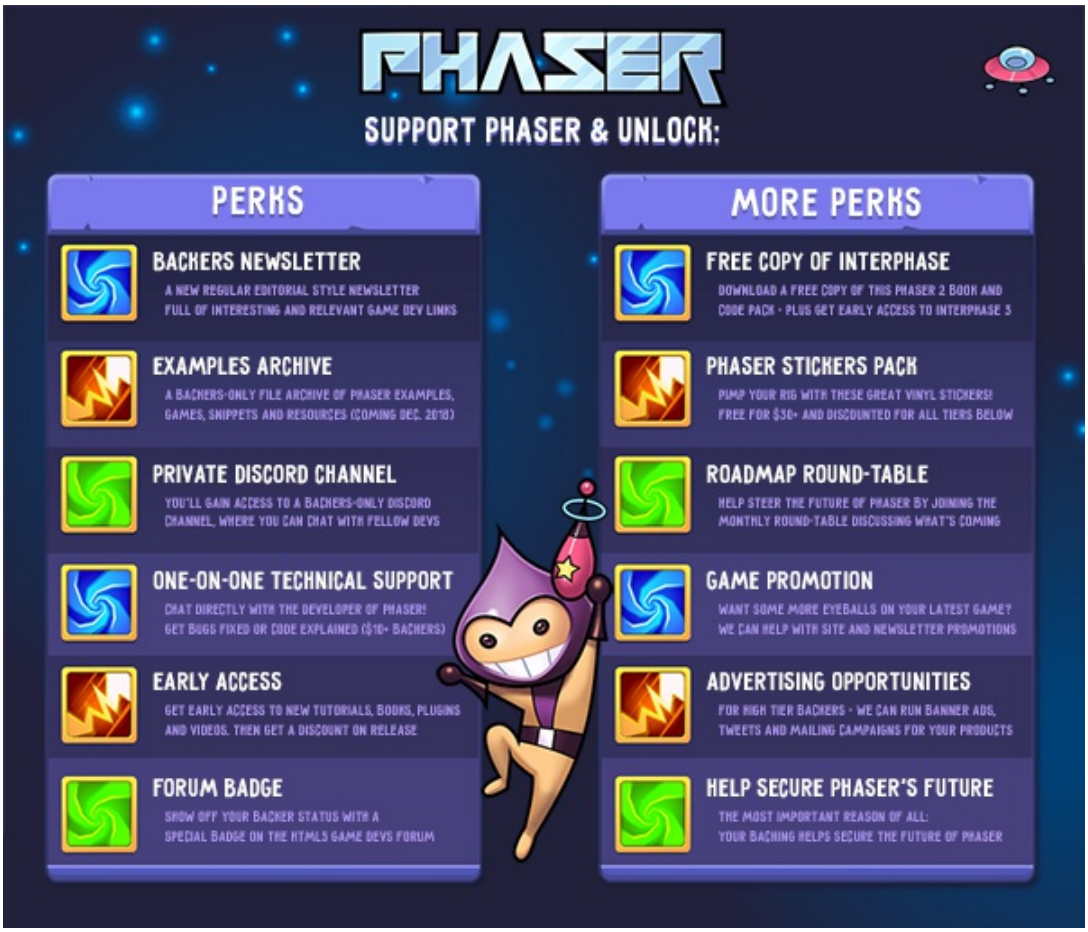
A complete Phaser 3 and JavaScript Game Development package. 9 courses, 119 lessons and over 15 hours of video content. Learn to code and create a huge portfolio of cross platform games.

Help support Phaser

Because Phaser is an open source project, we cannot charge for it in the same way as traditional retail software. What's more, we don't ever want to. After all, it's built on, and was born from, open web standards. The core framework will always be free, even if you use it commercially.

You may not realize it, but because of this, we rely 100% on community backing to fund development.

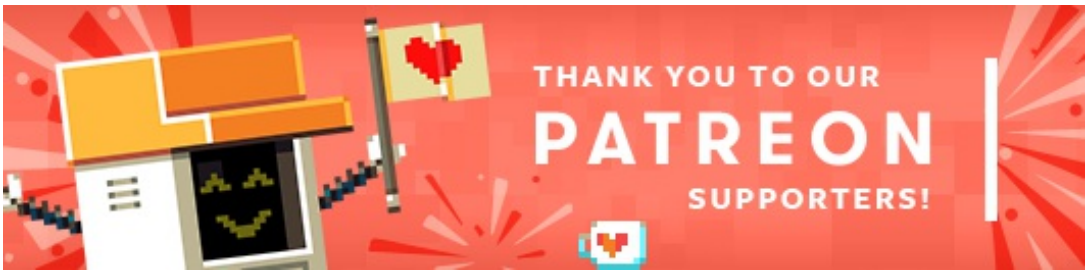
Your support helps secure a constant cycle of updates, fixes, new features and planning for the future. There are other benefits to backing Phaser, too:



Click to see the full list of backer perks

I use Patreon to manage the backing and **you can support Phaser from $1 per month.** The amount you pledge is entirely up to you and can be changed as often as you like.

Please help support Phaser on Patreon



Thank you to these awesome Phaser Patrons who joined us recently:

Alex Berlovan
Alexander Lyabah
Andrea
Anna Kalampokeleurou
Can Delibas
Colby Williams
Daniel Dombrowsky
Евгений Тройнов
Halil ÇAKAR
Jeff Blubaugh
Joakim Walldén
Joseph DeStefanis
Justin Stahlman
Lane Fowler
Nathan Bean
Oktay Acikalin
Robert Podgorski
Sebastian Hendowski
seeward
slhtyosoro
Stacy Read
Sébastien Martin
Tony Colley

Also, thank you to **Vladislav Forsh** for increasing their pledge.

Dev Log #136

Welcome to another new Dev Log! This week has been a constant stream of progress, which is always a good thing. There have been several days where I've gotten so into the coding zone that I didn't even realize many hours had passed. So, what has been taking all my attention? Let's dive in ...

Scale Manager Progress

I've been working almost exclusively on the new Scale Manager and am really happy with where things now stand. I have rewritten it from scratch, compared to the Scale Manager found in Phaser 2. My original plan was to port it over to V3 and use it 'as is', just tidying things up a bit as I went. But, it soon became apparent that the way it was built wasn't really relevant for the web as we know it today. It also had a lot of weird features that, honestly, even I struggled to understand what they did. In my mind, starting from a fresh slate made more sense.

So, I dumped all the old V2 code and APIs, and re-thought how a more modern scaling system should operate. The first thing to do was to decide upon what scaling modes would be offered. This was, in part, actually influenced by Unity. Although I'm not a Unity developer (I've never actually coded a single thing in it), I was aware it had something called the Aspect Ratio Fitter, which is a UI component that is responsible for simply controlling the dimensions of whatever you attached it to, based on an aspect ratio and a selected aspect mode.

At the end of the day, this is all scaling is: the management of the dimensions of an object, based on an aspect mode and that of its parent. The difference is that for the Phaser Scale Manager it needed to apply its constraints to the game canvas, rather than an arbitrary UI component.

This got me thinking. Rather than building all of this functionality into the Scale Manager, why not create a new component that could manage the lions share of the work, but in such a generic way that it could be re-used both through-out Phaser and utilized directly in your own games as well. To that end, the Size component was created.

Super Size Me

You can create a Size component directly by instantiating a Size object. The constructor arguments allow you to pass in a width, height, aspect mode and a

parent. All of these are optional and can be set at any point via class methods. There are other limitations you can place on the Size too, including a minimum width and height, a maximum width and height, a 'snapping' value, which means the dimensions are snapped to the given grid size, and finally a parent element.

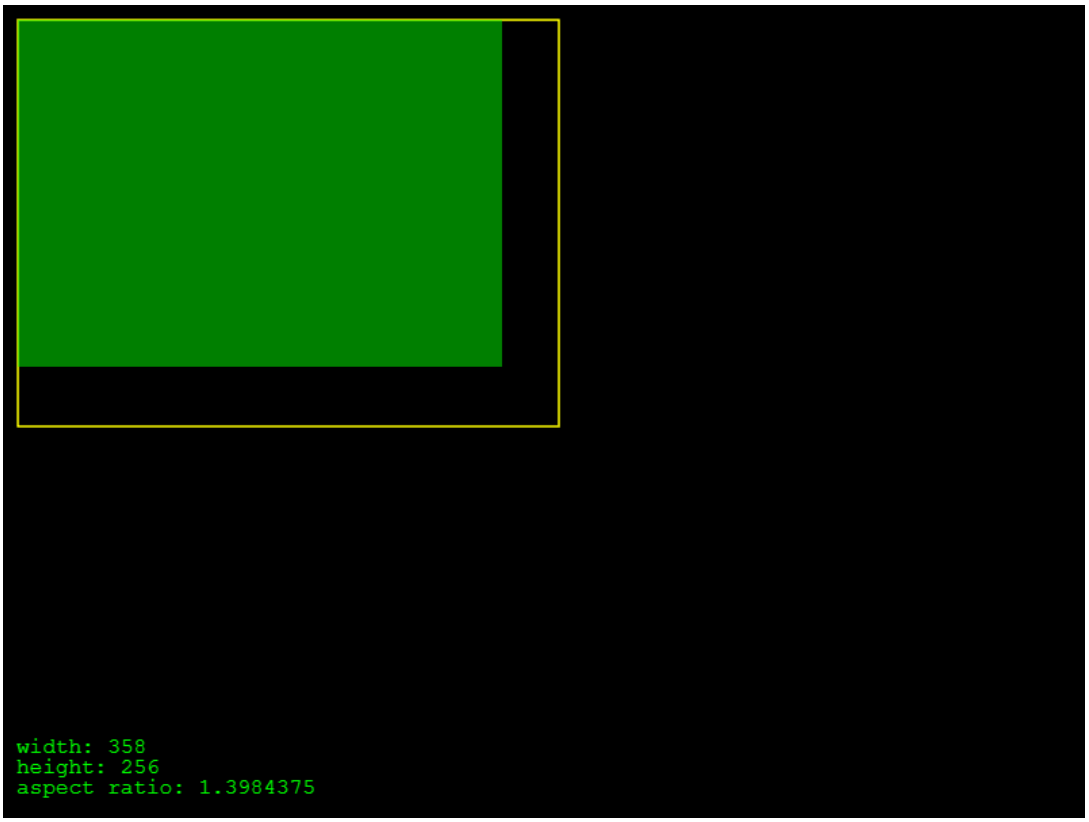
Let's create a Size object that is 400 x 200, with the default aspect mode and no parent:

```
var box = new Phaser.Structs.Size(400, 200);

console.log(box.aspectRatio);

// Outputs: 2
```

This is what you'd expect, because the aspect ratio is the width divided by the height, and 400 / 200 is 2. So far, so normal. Where the Size component comes into play, however, is when you set one of the five available aspect modes. The modes are as follows:

NONE

This is the default aspect mode. It essentially means "don't do anything". It won't try to enforce the ratio between the width and height at all. If you adjust the dimensions of the Size component while in this mode, they will just change and the aspect ratio itself will be updated. Click the screenshots for interactive examples:



It will, however, still respect the Parent Size object, if set. It will also factor in the optional minimum and maximum dimensions and snapping values.

It's a useful default, but the other modes are where it starts to get interesting.

WIDTH_CONTROLS_HEIGHT

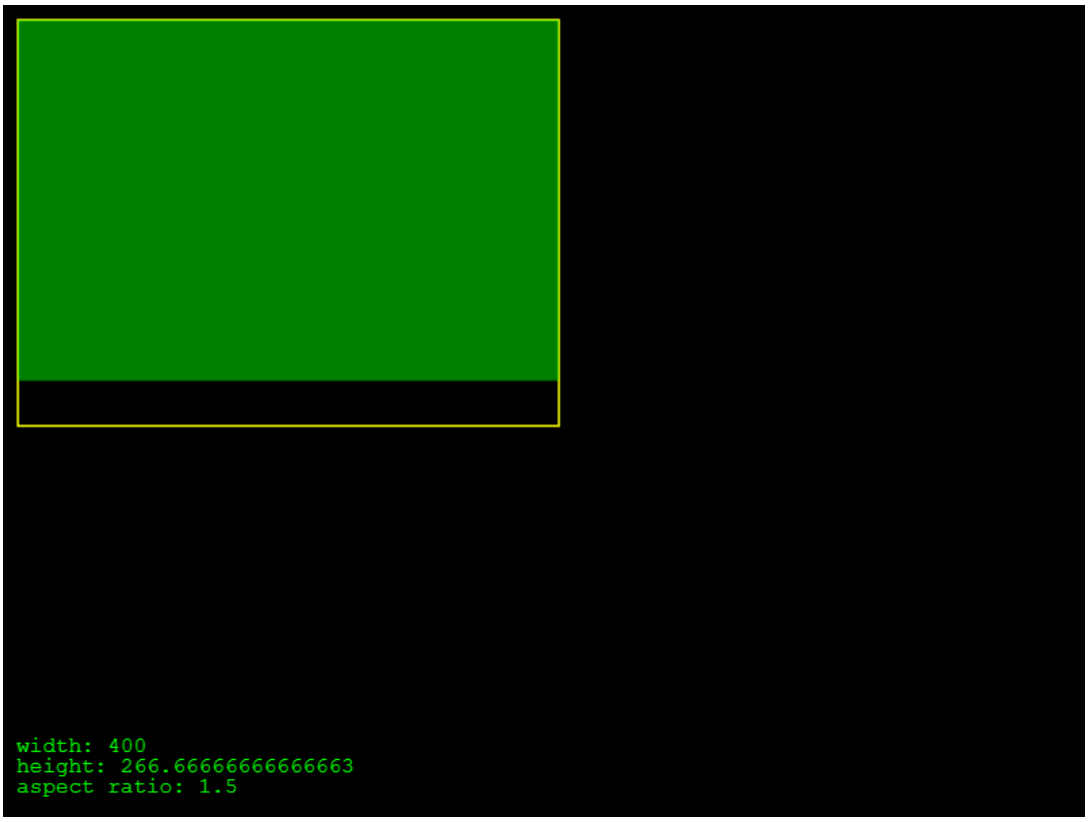
This aspect mode locks the aspect ratio between the width and the height. If you then adjust the dimensions of the Size component the `width` will be changed to whatever you requested, but the `height` will be calculated based on the ratio and the new width.

For example:

```
var box = new Phaser.Structs.Size(320, 240, Phaser.Structs.Size.WIDTH_CONTROLS_HEIGHT);

console.log(box.toString());

//  [{ Size (width=320 height=240 aspectRatio=1.3333333333 aspectMode=1) }]

box.setWidth(700);

console.log(box.toString());

//  [{ Size (width=700 height=525 aspectRatio=1.3333333333 aspectMode=1) }]
```

Notice how in the code above the aspect ratio has remained the same. The new

width has been set to 700 as requested, but the height has changed to 525. This is because that's the new height based on the width and ratio, because in this mode, the width controls the height.



HEIGHT_CONTROLS_WIDTH

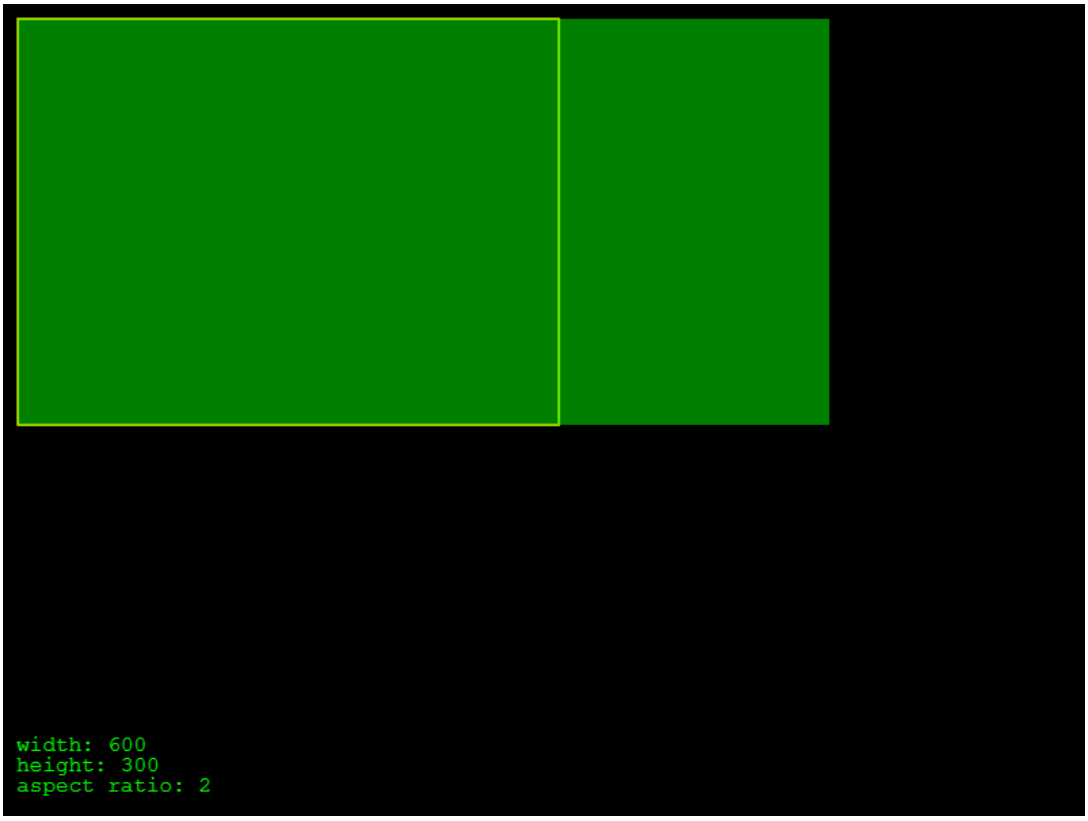
As you may expect, this is the opposite of the previous mode. The ratio is again locked and if you adjust the dimensions of the Size component the `height` will be changed to whatever you requested, but the `width` will be calculated based on the ratio and the new height.

For example:

```
var box = new Phaser.Structs.Size(320, 240, Phaser.Structs.Size.HEIGHT_CONTROLS_WIDTH);

console.log(box.toString());

//  [{ Size (width=320 height=240 aspectRatio=1.3333333333 aspectMode=2) }]

box.setHeight(500);

console.log(box.toString());

//  [{ Size (width=666.6666666 height=500 aspectRatio=1.3333333333 aspectMode=2) }]
```

Again, notice how in the code above the aspect ratio has remained the same.

The new height has been set to 500 as requested, but the width has changed to 666. This is because that's the new width based on the height and ratio, because in this mode, the height controls the width.



FIT

Nothing to do with broken New Year's resolutions, this aspect mode is where things start to get interesting. As before, the initial dimensions given to the Size component set the aspect ratio. When you then call `setSize`, or the `fitTo` method, or change its width or height properties, it recalculate the new dimensions based on its aspect ratio.

```
var box = new Phaser.Structs.Size(320, 240, Phaser.Structs.Size.FIT);

console.log(box.toString());

//  [{ Size (width=320 height=240 aspectRatio=1.3333333333 aspectMode=3) }]

box.fitTo(550, 400);

console.log(box.toString());

//  [{ Size (width=533.3333333 height=400 aspectRatio=1.3333333333 aspectMode=3) }]
```

Let's assume our game has a resolution of 320 x 240. I know, that's tiny, but work with me here for a moment. Then the browser resizes to 550 x 400 in size.

We want our game to still fit perfectly, with nothing obscured or cut off. The Size component performs its calculations and we get a new size of 533 x 400. The aspect ratio is still exactly the same, so our game hasn't been 'distorted' at all and will maintain its appearance. But it now fits as snuggly as it can into the new 550 x 400 size, while maintaining its ratio.

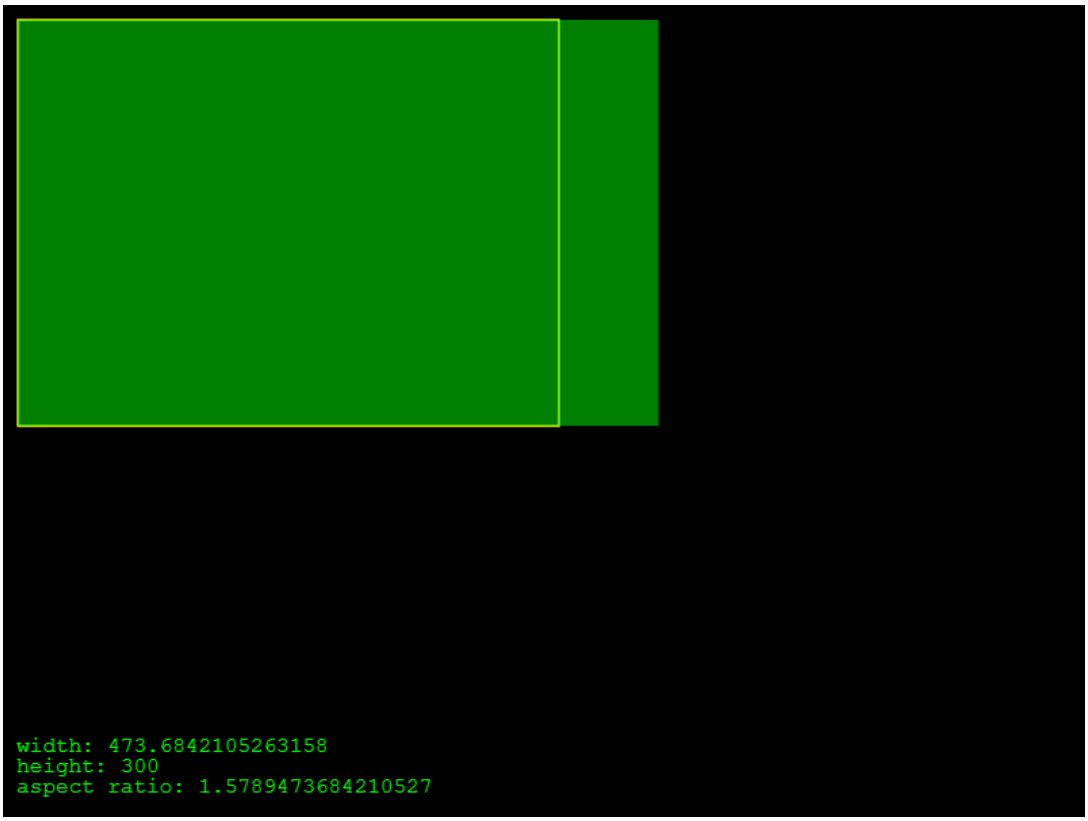
That is what the FIT mode does. It ensures your game fits the given space as tightly as possible without any change in ratio at all, and without cutting any of your game off. This means that, depending on the size you're trying to get it in, there may be some space *inside* the target that is not covered.

ENVELOP

The final aspect mode is 'envelop'. Not to be confused with 'envelope', this mode performs in a similar way to FIT. The important difference is that the dimensions are changed so that they fully envelop the new width and height, while maintaining aspect ratio.

```
var box = new Phaser.Structs.Size(320, 240, Phaser.Structs.Size.ENVELOP);

console.log(box.toString());

//  [{ Size (width=320 height=240 aspectRatio=1.3333333333 aspectMode=4) }]

box.envelop(550, 400);

console.log(box.toString());

//  [{ Size (width=550 height=412.5 aspectRatio=1.3333333333 aspectMode=4) }]
```

Again, our game has a tiny resolution of 320 x 240. The browser resizes to 550 x 400. But we want our game to fully envelop the available space, so we use the envelop aspect mode. The new size after the change is 550 x 412, meaning we've got an extra 12 vertical pixels that won't be visible within the browser window. However, there is literally no space not filled. Depending on your game this may be a preferable, visually, to any other mode.



Meet the Parents

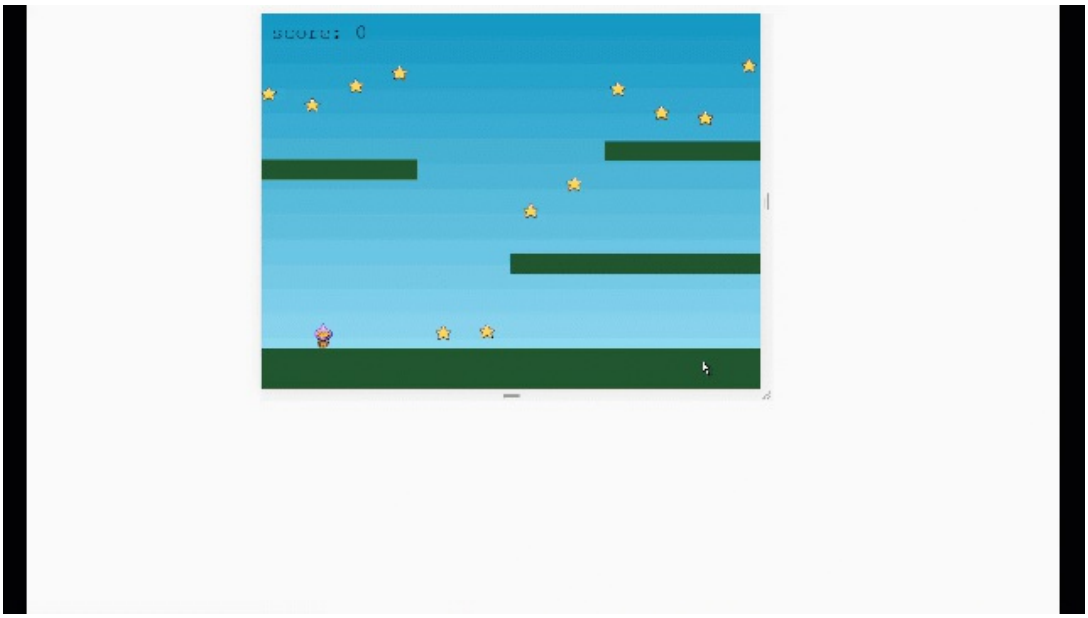
The Size component can do more than just constrain dimensions to a ratio. You can also give a component a parent. The parent can actually be any object, so long as it has publicly accessible width and height properties. So a Sprite, Rectangle, Physics Body, other Size object, etc. Most of the examples above use a Parent in them.

In the case of the Scale Manager the 'parent' is the browser window. When the browser resizes, or changes orientation, there are a bunch of internal Size components that are updated accordingly, maintaining their aspect ratios as they go. Hopefully, you can see the relationship between what the Size component offers, and what the Scale Manager does. Let's see how to apply it to a game.

All Scale Manager settings are applied via the Game Configuration. For example, here is a game set to use the FIT scale mode:

```
var config = {
    type: Phaser.AUTO,
    backgroundColor: '#2dab2d',
    scale: {
        parent: 'phaser-example',
        mode: Phaser.DOM.FIT,
        autoCenter: Phaser.DOM.CENTER_BOTH,
        width: 800,
        height: 600
    },
    scene: {
        preload: preload,
        create: create
    }
};
```

And here is what it looks like running while being resized:



You may have noticed the 'autoCenter' property in the config. This allows the game to remain centered in modes where there may be extra space available. You can also set minimum and maximum dimensions for your game. This means that even if the browser window drops below a certain size, the game won't get any smaller. The opposite is also true. Here's an example config:

```
var config = {
    type: Phaser.AUTO,
    backgroundColor: '#2dab2d',
    scale: {
        mode: Phaser.DOM.FIT,
        parent: 'phaser-example',
        width: 800,
        height: 600,
        min: {
            width: 400,
            height: 300
        },
        max: {
            width: 1600,
            height: 1200
        }
    },
    scene: {
        preload: preload,
        create: create
    }
};
```

Based on the config above the game canvas will never drop below 400x300 pixels in size, or expand beyond 1600x1200, all while still maintain its aspect ratio and fitting all available space.

SUPER RESIZE ME

Previously, Phaser 3 had a 'resize' method available on the Game instance itself. This has now moved to the Scale Manager. You can also set a 'RESIZE' scale mode and even define your size to be a percentage. The following config demonstrates:

```
var config = {
    type: Phaser.AUTO,
    backgroundColor: '#2dab2d',
    scale: {
        mode: Phaser.DOM.RESIZE,
        parent: 'phaser-example',
        width: '100%',
        height: '100%'
    },
    scene: {
        preload: preload,
        create: create
    }
};
```

Here, the canvas is created at whatever size the parent element currently is. Then as it changes, it's resized to match it as you can see in the following animated gif:

The code is as follows:

```
function create ()
{
    this.bg = this.add.tileSprite(0, 0, this.scale.width, this.scale.height, 'rain').setOrigin(0);
    this.logo = this.add.sprite(this.scale.width / 2, this.scale.height / 2, 'logo');

    this.scale.on('resize', resize, this);
}

function resize (gameSize, baseSize, displaySize, resolution)
{
    var width = gameSize.width;
    var height = gameSize.height;

    this.cameras.resize(width, height);

    this.bg.setSize(width, height);
    this.logo.setPosition(width / 2, height / 2);
}
```

The 'resize' function is sent the Size objects from the Scale Manager. Depending

on what you're doing, depends on which one you need to use in your function. As this is a RESIZE example, using 'gameSize' makes most sense. Each of them are explained in detail in the Scale Manager docs.

ZOOM, ORIENTATION LOCKING, FULL SCREEN

I've also built plenty of options in to the Scale Manager. You can set a 'zoom factor' for your game. This was in V3, to a degree, in previous versions. For example, if your game was 160 x 144 in size (the size of the classic Gameboy), and you set the zoom level to 4, it would have a 'base size' of 640 x 576. You can then still apply a Scale mode and centering on the top of this, too! What's more, if you set a zoom level above 1 it will automatically enable pixel art mode for all textures too (as that's really the only time you'd ever need to use it).

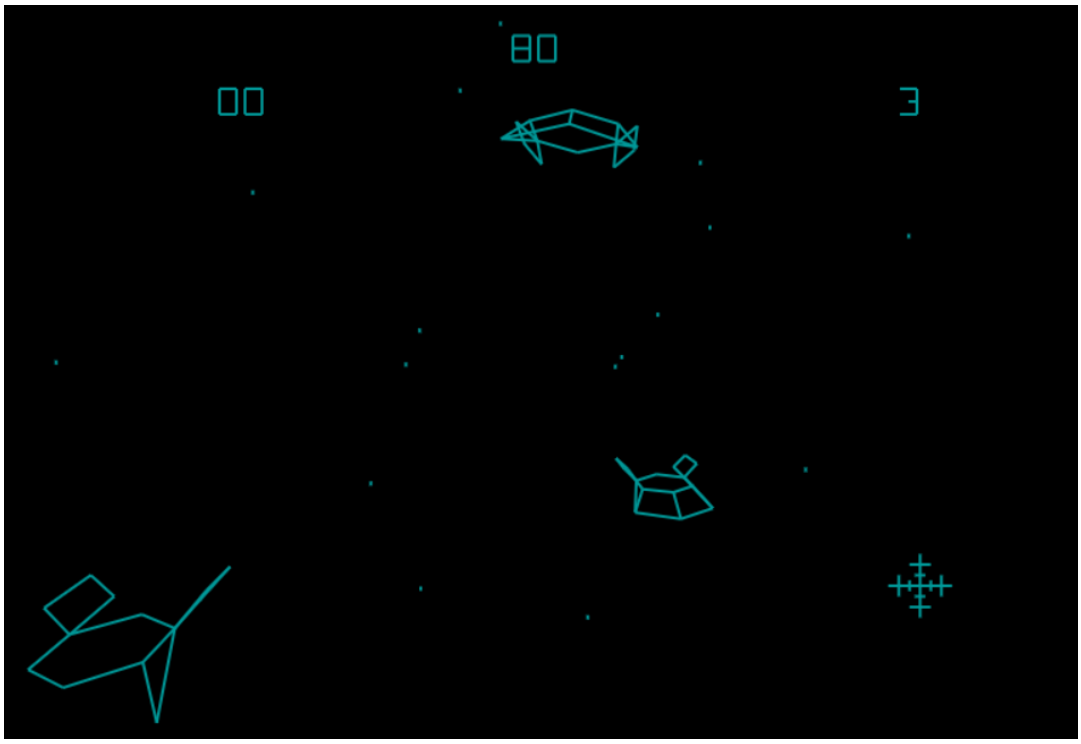
Support for orientation is also in. You can listen for orientation events from the Scale Manager and adjust your game accordingly, perhaps displaying a 'please rotate' screen. Both the browser orientation API (which only works on mobile) and a desktop calculated version are included. Orientation Locking support is also included, although again this is very browser specific.

The final main feature left to add is the Full Screen API, which I'm confident will be completed this week. I need to conduct further cross browser tests, especially on a wider variety of mobile devices, but so far it's looking great and resizing as you'd expect.

TEST BUILD

You can test it out for yourself by grabbing the V136 build from the examples repo and looking at the source code of the new Scale Manager examples. Please note that I'm actively working on it, so the examples will be changing and being added to this week.

All in all, I'm very happy with how it's working and the new structure. The Size component itself is extremely handy just for general game-use! and the fact the Scale Manager uses it for the bulk of its work is great. I've also been able to remove loads of excess code from the Input Manager and Renderers as a result of this work. I'm very hopeful we'll have a full 3.16 release before the end of January.



Tailgunner was a vector-based arcade game released in 1979. It has been faithfully ported to JavaScript via static binary translation and plays really well!

A step-by-step guide to building a simple Chess AI.

Git Command Explorer. Find the right commands you need without digging through the web.

Phaser Releases

Phaser 3.15.1 released October 16th 2018.
Phaser CE 2.11.1 released October 2nd 2018.

Please help support Phaser development

Have some news you'd like published? Email support@phaser.io or tweet us.

Missed an issue? Check out the Back Issues page.