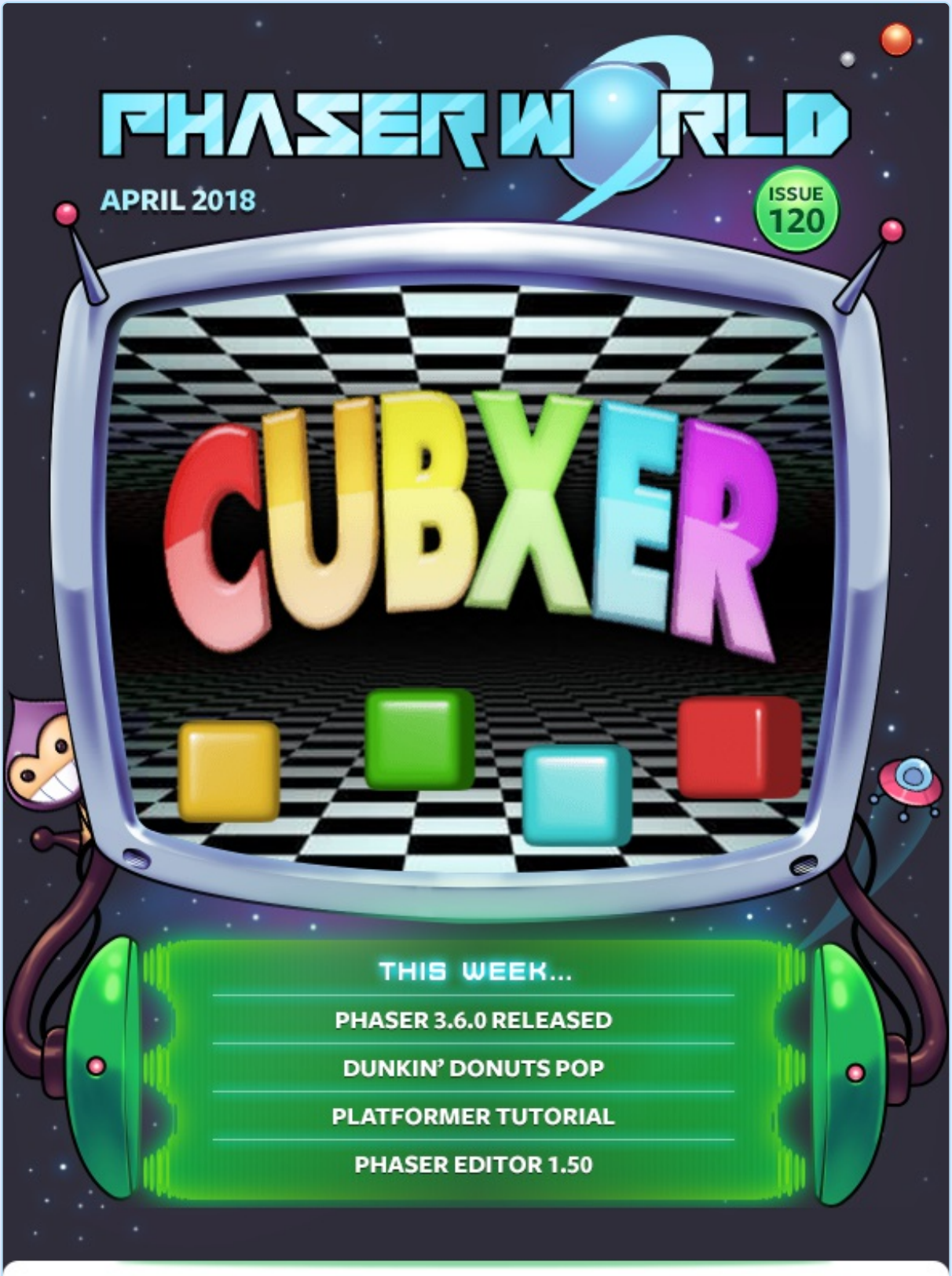


Welcome to Issue 120 of Phaser World

In this issue, we've got a huge and slightly overdue Dev Log, some superb new games (I had real fun playing Cubxer), loads of tutorials, a Kickstarter to back

and a double-dose of news. I know we're publishing more erratically at the moment, you can read the Dev Log to find out why, but we'll definitely be back with issue 121 when it's ready.

Until the next issue, keep on coding. Drop me a line if you've got any news you'd like featured by simply replying to this email, messaging me on Slack, Discord or Twitter.

The Latest Games

Game of the Week
Cubxer
A lovely take on the classic game Columns, with great 3D effects, sounds and speech.

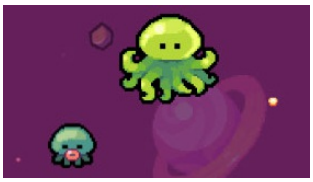
Staff Pick
Dunkin' Donuts Donut Pop
Match 3 donuts in this cute and mouth-watering game from Dunkin' Donuts.

Crossword Scape

Think, search, drag and connect all the letters to form a word in this cross between Word Cookies and a Crossword.

Snakes and Ladders

Fight a huge monster or play a nice board game instead? Find out in this RPG snakes and ladders game.



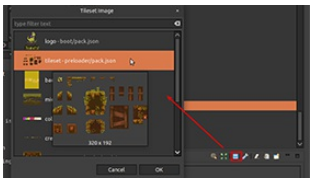
Omicronian

A side-scrolling shooter game template to buy and reskin.

What's New?

Phaser 3.6.0 Released

This release includes the final version of Containers, physics improvements, Group updates and more documentation completed.

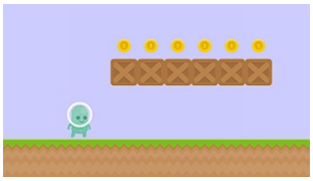


Phaser Editor 1.5

The latest version of the best Phaser IDE is out and it comes with a new free license and lots of great features!

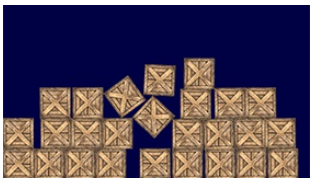
Game Dev Kickstarter

Learn to code and make impressive games with JavaScript and Phaser. Create a huge portfolio of both mobile and desktop games.



Phaser 3 Platformer Tutorial

How to Make a Mario-style Platformer with Phaser 3.

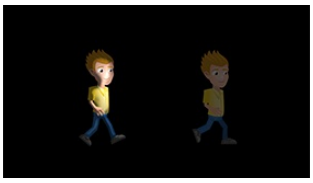


Matter Physics Example

A mini tutorial from Emanuele Feronato about creating Matter Phyics Bodies.

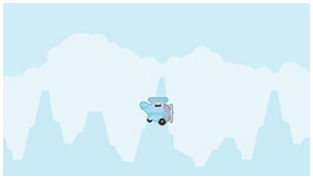
Coding Train Tutorial

Learn Phaser 3 Game Development with Cat Small in this Coding Train video.



Light Effects Tutorial

A tutorial on creating and using normal maps for light effects in Phaser 3.

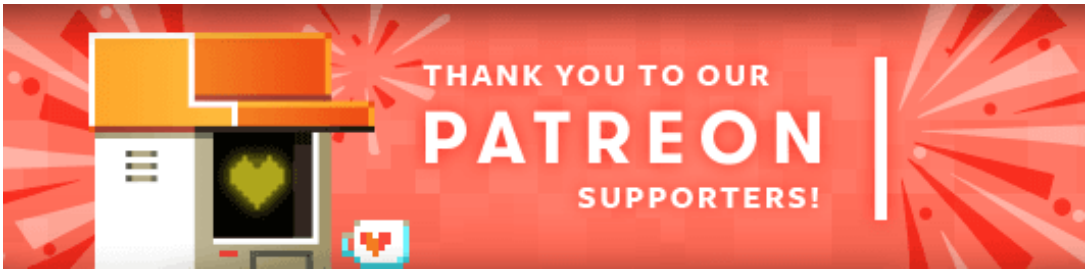


Mobile Games Tutorial

A new tutorial all about creating mobile games with Phaser 3 and Cordova.

Magick Prototype Tutorial

A tutorial containing the complete source to re-create a prototype of the classic iPad game.



Thank you to our awesome new Phaser Patrons who joined us recently:

Mike Carter
Niklas Bichinger
Rob Muhlestein
tweed li
James Tease
Nikita Borisov
Robin Gersabeck

Patreon is a way to contribute towards the Phaser project on a monthly basis. This money is used *entirely* to fund development costs and is the only reason we're able to invest so much time into our work. You can also donate via PayPal.

Backers get forum badges, discounts on plugins and books, and are entitled to free monthly coding support via Slack or Skype.

Dev Log #120

It's been 3 weeks since the last issue and things haven't let up one bit. The sheer volume of work going on with Phaser 3 at the moment is extraordinary. Since #119 we have released Phaser 3.4.0, 3.5.0, 3.5.1, 3.6.0 and are planning to release 3.7.0 this week. These are not minor versions either. Each one carries with it significant new features and hundreds of updates and fixes. I'm going to cover the most important of these in more detail now, but as always please do read the Change Log. I know it can be quite overwhelming but we're careful to log as much as we can, so you can see what has changed and where.

Containers have arrived

Truth be told, containers were a lot more work than expected. It took nearly 2 months to develop and integrate, ending up touching multiple areas of the API as a result of their complexity. They were merged in Phaser 3.4.0 but under a beta flag and then set free in Phaser 3.6.0. As with most things in Phaser using them is as straight forward as you'd expect:

```
var container = this.add.container(400, 300);

var sprite = this.add.sprite(0, 0, 'lemming');

container.add(sprite);
```

The container has its own transform, meaning a position, rotation, and scale. Any changes made to the container flow down to its children. In the case of the above code, you've got one Sprite, which is positioned at 0 x 0 *within the Container*, meaning it'll render at 400 x 300 on-screen in this example because that is where the Container was created.

When you add a child to a Container it is removed from the display list, if it was on it, and added exclusively to the container's own internal display list. A child cannot exist in more than one Container at once (well, it can, but we'll cover that later! for now the default is that it cannot)

Positions are relative, not absolute. Have a look at the following code:

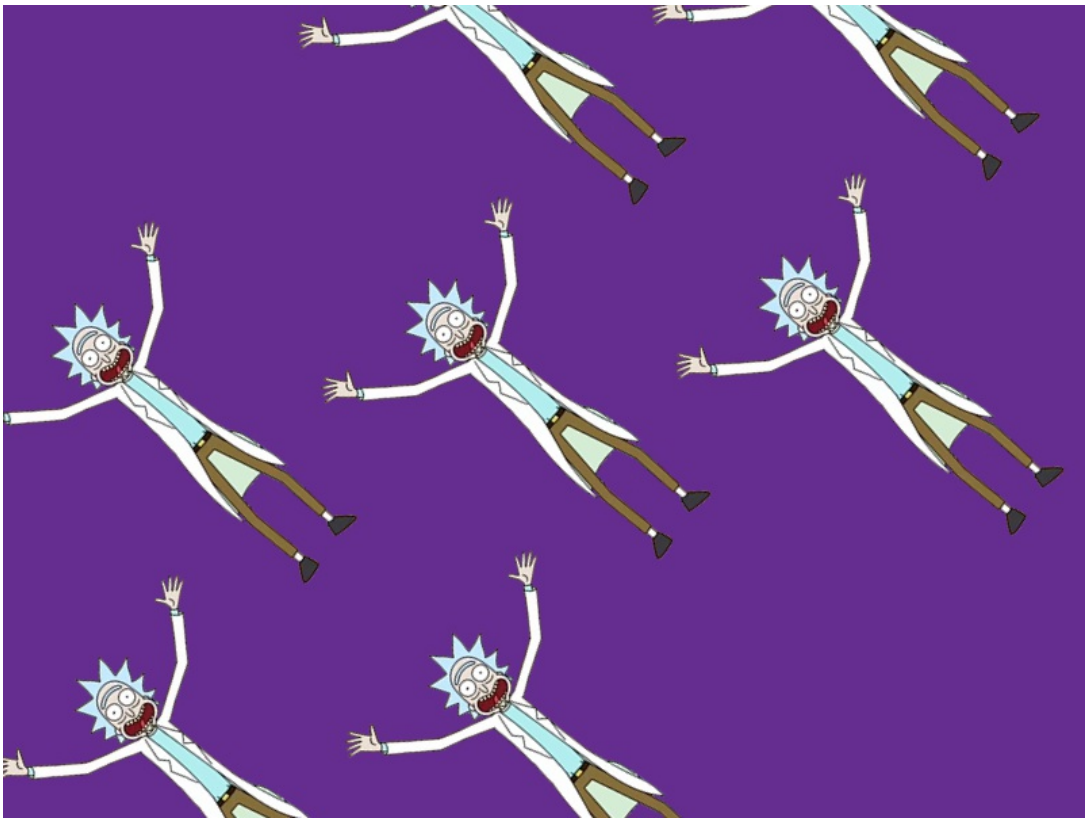
```
var container = this.add.container(400, 300);

var sprite0 = this.add.sprite(-400, 0, 'rick');
var sprite1 = this.add.sprite(0, 0, 'rick');
var sprite2 = this.add.sprite(400, 0, 'rick');
var sprite3 = this.add.sprite(-200, -200, 'rick');
var sprite4 = this.add.sprite(200, -200, 'rick');
var sprite5 = this.add.sprite(200, 200, 'rick');
var sprite6 = this.add.sprite(-200, 200, 'rick');

container.add([ sprite0, sprite1, sprite2, sprite3, sprite4, sprite5, sprite6 ]);
```

Here we're creating 7 sprites and adding them all to the Container by passing an array of them. What's important though are the positions - you can see positions like "200, -200" which are relative to the position of the Container itself, where 0 x 0 is the Containers center.

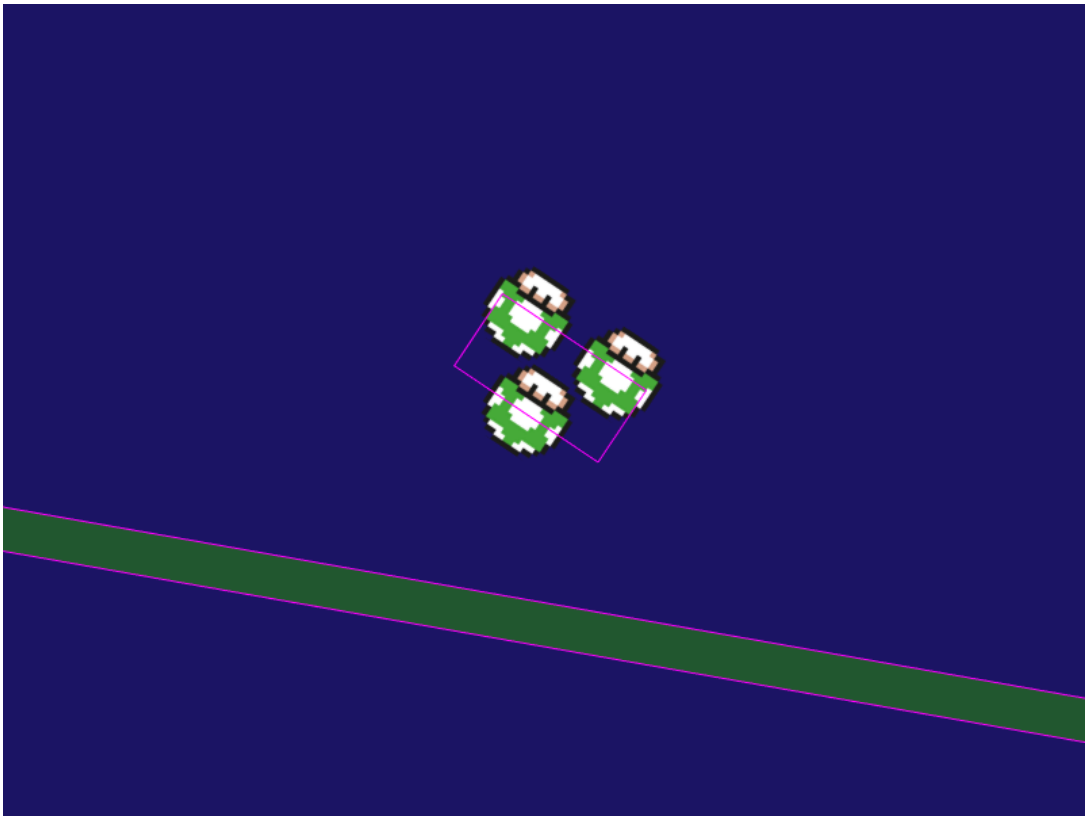
You can then manipulate the Container to create some interesting effects. Here's the full version of the above code, with a simple tween added to it (click the screenshot to run it):



Get schwifty!

Containers can be nested. This means you can insert one Container inside another and branch children off of that. We do not recommend this as the deeper the chain goes, the more expensive every single look-up becomes, as each child traverses the tree back to the root every time it renders. But, it can be done, and in some situations, or for low-intensity areas of a game, is perfectly acceptable.

Containers can also have physics bodies. There are 3 examples in the labs showing how to add Arcade Physics, Impact Physics and Matter Physics to a Container. Here's the Matter body example:



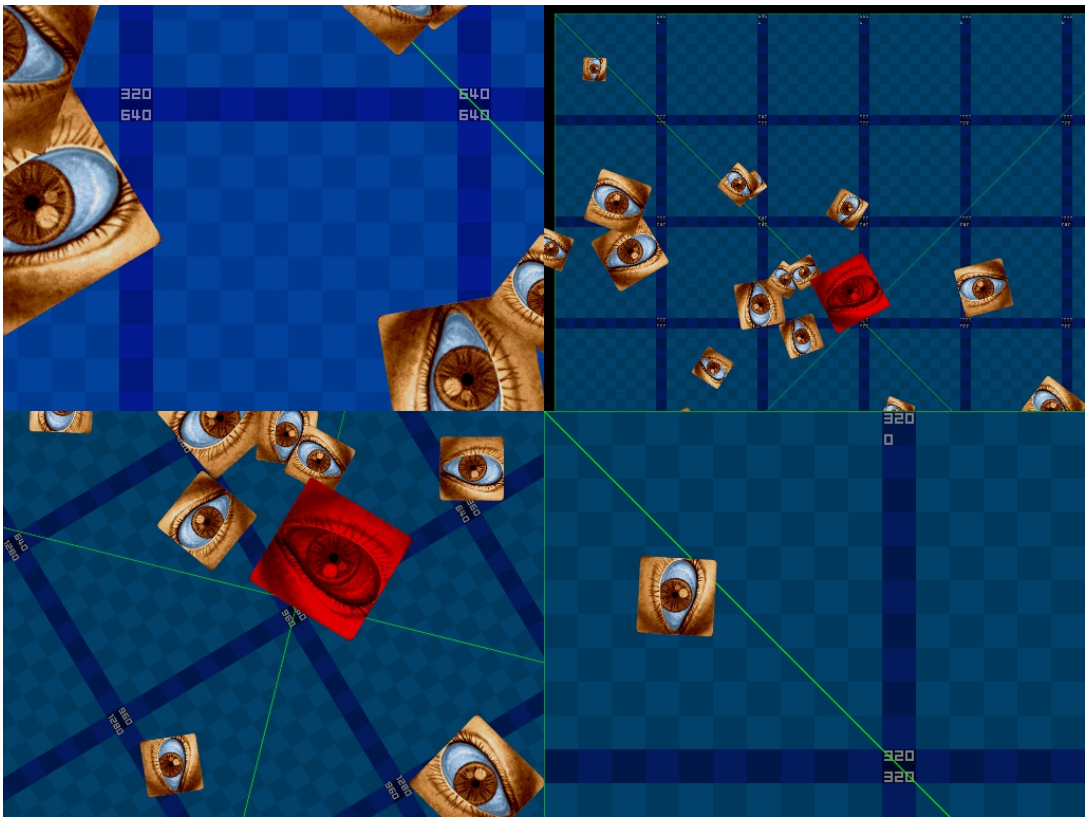
It's important to understand the limitations of physics and Containers. While you can add a body to the Container itself, if any of the children have physics bodies then things are not going to align properly (unless the Container is positioned at 0x0 in the Scene.) We were adamant at the start of adding Containers that their introduction wouldn't harm the normal performance of Phaser. In order to support Container children having physics bodies, we would have had to add in multiple extra steps and branching to factor in the influence of the Container hierarchy on the physics enabled children. This wasn't something we were willing to sacrifice at this time. The only exception to this is Arcade Physics. Because of the way it internally works it was easy for us to support the feature - but not so with Impact or Matter. It's something we may revisit at a later date but at now you ought to be aware of it.

Containers and Input

By far the most complex aspect of adding Containers was updating the Input system to cope with them. Input in Phaser 3 is already quite complex because of the ability to support multiple cameras and cameras within cameras. Add in multi-level Container transforms to this and it became a whole new level of pain! It took Felipe and I several weeks to figure out this element alone.

The first complication was that by default Input works by taking the 'top most' Game Object on the display list when you interact with it and then ignoring the rest (by default), we did this via a nice simple quick sort of the interactive objects

based on their depth property. Factor in Containers though and it's no longer a simple single array sort. Plus I wanted for Containers themselves to be able to be interacted with, so have their own optional hit zones and input events. Drop this back into a multi-camera, multi-depth scenerio and you can see why it took a while to resolve. But, resolve it we did, as you can see in this example:



As mentioned above, Containers themselves can be interactive too. To enable this you need to provide a shape it will use as the hit area. This is because Containers don't have textures, so we cannot automatically create a hit area from a texture dimension. Doing so is pretty easy though:

```
var hitArea = new Phaser.Geom.Circle(0, 0, 60);

container.setInteractive(hitArea, Phaser.Geom.Circle.Contains);
```

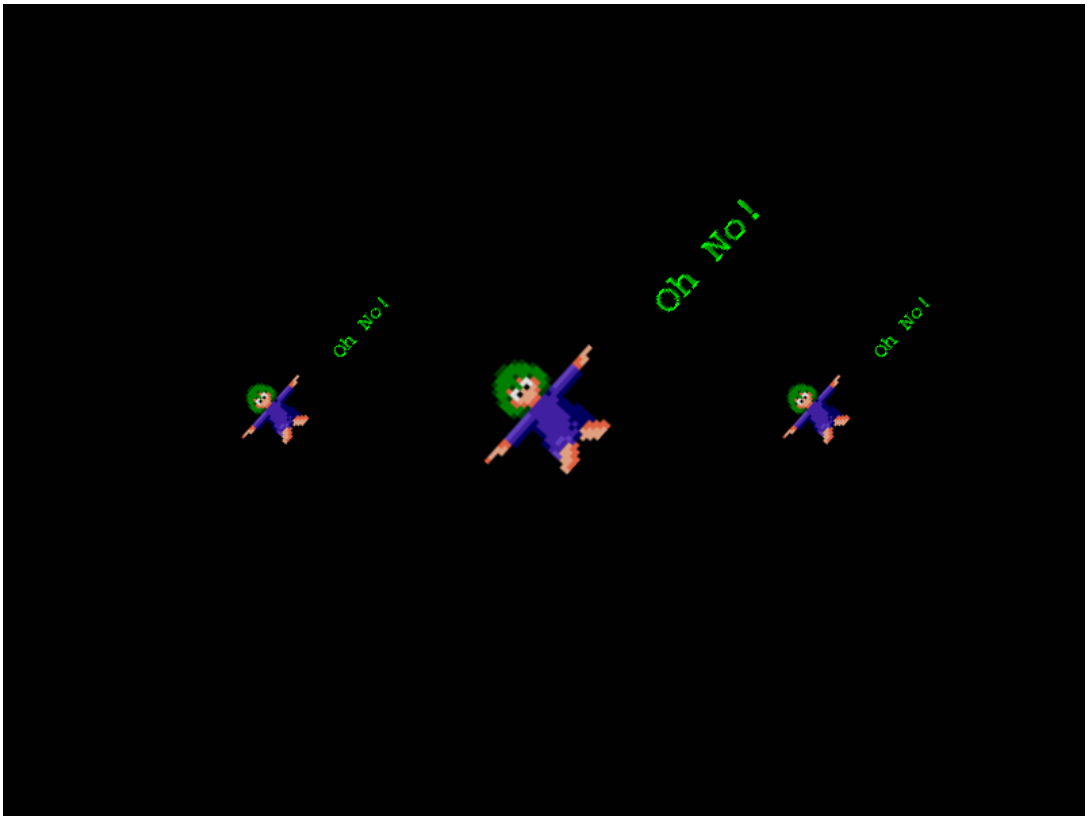
Just pass in a shape and a function to use that determines if an x/y coordinate is inside it or not. In this case the Container has a circle hit area, which you can see in this example where we render the hit area to a Graphics object so you can see it:

The **setInteractive** method allows you to define *any* object and callback as the hit area, it doesn't have to be a Phaser Geometry object, it could literally be anything. As long as the 'contains' callback function returns a boolean that is good enough for the Input system. This gives you an extreme amount of flexibility for all interactive objects, not just Containers.

Non Exclusive Containers

There's another feature of Containers that may not be immediately obvious from the name, so is worth explaining in more detail. A traditional Container implements its own internal display list. You can move children up and down the list, swap positions, and so on. But the children can only exist in one Container at once, and if you add them to another Container they'll remove their reference from the first. So far, so standard.

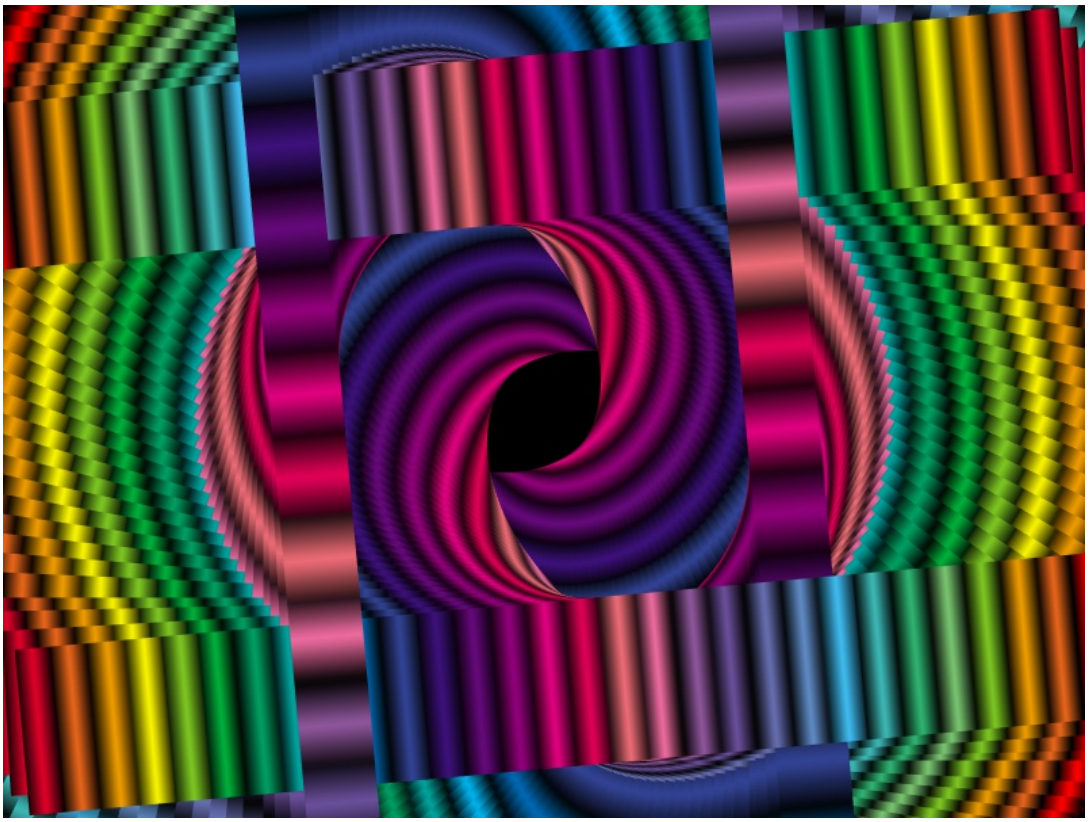
However, you can also create non-exclusive Containers. As the name implies the children of a non-exclusive Container are not bound by such restrictions. They can exist in multiple Containers, as well as the core Scene Display List, all at the same time. In the following example there's only 1 Sprite and 1 Text object, but we've 3 copies of them rendered:



It works by creating a Game Object, just like you would normally. Consider this as being the original 'true source' if you will. Any time you add this object to a non-exclusive Container it creates a temporal clone that exists purely for rendering only. For example, if you had an animated Sprite, then added the Sprite to 10 other non-exclusive Containers, they would all render it based on its original properties and whatever influence their own transform has upon it.

If you were to change the frame of the Sprite, by playing an animation on it or tinting it, all the other clones would instantly be updated too. Only one Sprite exists in memory, the other Containers just hold a reference to it, so it's a really cheap way of building up neat visual effects potentially displaying thousands of objects with far less memory overhead and objects created.

Here's another example of what you can do with just 4 sprites and a bunch of Containers. Click to run the demo because it's best to see it in motion!



Although you could achieve a similar result using a RenderTexture there's no texture creation involved here, it's just a pure WebGL batch operation. I had fun creating this example and came up with various different effects in the process of doing so, you can find them in the Labs Containers folder, look for the ones called Twirl 1 to Twirl 6.

Scene Transitions

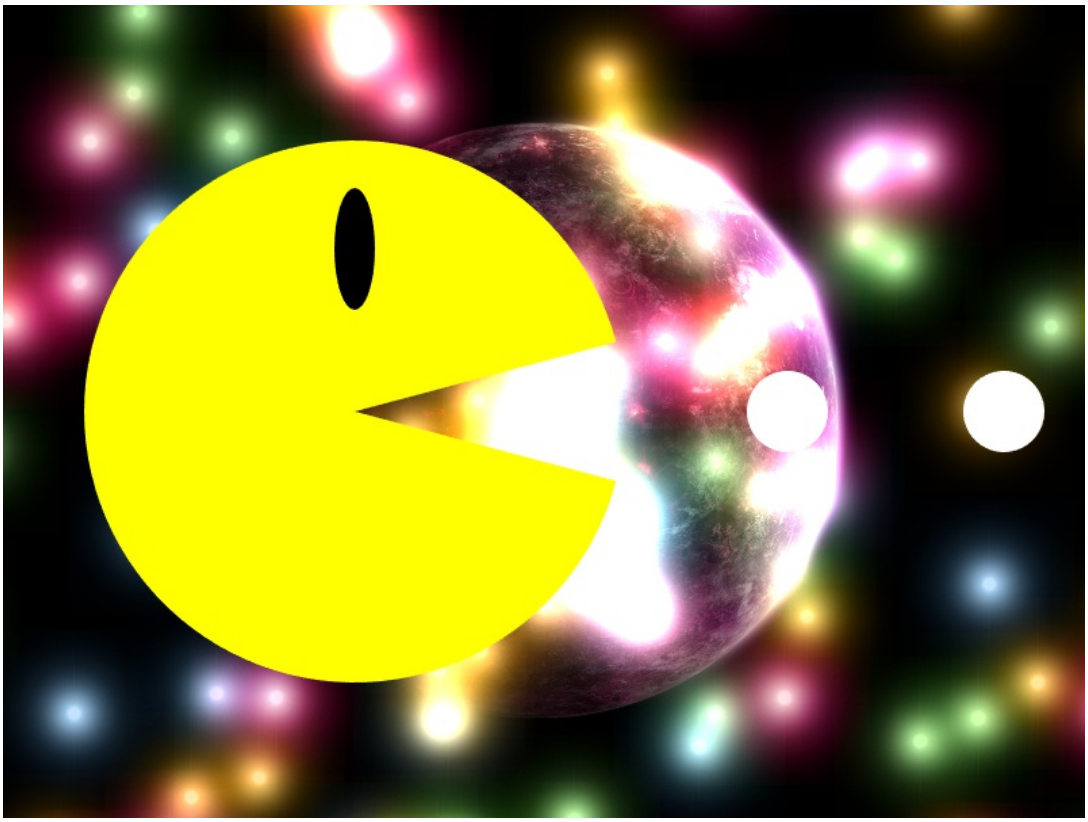
Another new feature that landed in Phaser 3.5.0 were Scene Transitions. Scenes have undergone a small internal tweak to standardize how plugins work and the boot and destruction process. This means you can now restart the same Scene as many times as you like and it'll restart properly as if from new.

Transitions were a natural evolution of the work I did in this area. You simply call **scene.transition** and pass it a configuration object. Here's an example:

```
this.scene.transition({
    target: 'sceneB',
    duration: 2000,
    moveBelow: true,
    onUpdate: this.transitionOut,
    data: { x: 400, y: 300 }
});
```

The object specifies the target Scene to transition to, the duration of the transition (2 seconds in this case) and then a range of optional properties. In the code above we're calling `moveBelow`, which means it will move the transitioning Scene below the current one in the Scene List. It defines an onUpdate callback. This callback is fired every frame for the duration of the transition and is passed a progress value (between 0 and 1) representing how far into the transition we are. Finally, the 'data' object is just a bunch of data to pass to the target Scene when it starts up.

There are other properties too including the ability the block all Input during the transition, or sending the transition Scene to sleep instead of stopping it. The docs for this, and all of the Scene Plugin are now complete, so worth a read. There are also examples in the Labs, including this one. Make sure you click to advance to the next Scene:



Every different effect in the above demo is a unique Scene and you can click and then watch the transition from one to another: Hopefully, it gives you a few ideas about how you could fit transitions into your games.

As well as this I also overhauled the Camera Effects system. They're now a lot more flexible and all extend from a common effects class, which means you can now easily replace them with your own effects instead, should you wish. You'll find them fully documented now as well.

Documentation and TypeScript Defs Update

You may have noticed that I said some areas are now fully documented. That is because as I work on them, to fix bugs or add new features, I'm completing the docs as I go. Any new feature added in the past few versions is fully documented and this is a process I will carry on doing.

I receive quite a few comments about "when will the docs be finished?" which is fair enough, but let's set a few things straight: At the time of writing there are 6,385 items left to document. This includes properties, methods and parameters. I have completed all of the data-types (over 15,000 of them) but for the descriptions they cannot be automated or have tools built to handle them. I literally have to carefully describe what the thing is.

Assuming I spend just 1 minute per item, which is an extremely optimistic time scale, that still means there are over 35 days of work (given 3 solid hours per day documenting) before it's complete - and that is based on no breaks or days off. Make no mistake about it, it's a *monumental* task. There is no magical doc writing fairy that comes along in the night and waves a wand, it takes hard graft to do properly.

And it's graft that needs to be done alongside fixing bugs, replying to GitHub issues, responding on the forum, Slack, and Discord, replying to emails, writing news items for the site, creating new examples, reviewing and testing Pull Requests, writing Dev Logs and actually finding some time to sleep. Phaser is literally my full-time job right now, but please do not under-estimate the sheer quantity of effort it is taking to juggle all of these balls, because I'm doing it all on my own.

To those who have contributed towards the docs, **thank you**! You are Gods and Goddesses. If everyone reading this filled-in just one missing '[description]' block in the source we'd be complete in a single day. But I know that's not going to happen, so I'll keep plugging away at it and doing my best. Just please, let-up on

the snide remakes and comments, it's not exactly motivating.

Another element we've been working hard on are the TypeScript Defs. Antriel has done a grand job on the automated parser and between us we've fixed hundreds of docs errors and refined the parser into a good place. There are still a few tweaks left to do, specifically handling Events, but it's up, working and we generate new defs with every release. The parser source is freely available in the docs repo (it's written in TypeScript), so if anyone wants to help refine it they can. Once we're both happy the defs are solid I'll link them in from the main repo. For now, yes, you'll have to download them from the docs repo. It's a small price to pay given the weeks of hard work that has gone into them and it's only temporary.

Interestingly, I've heard a few times that Phaser 3 should never have been released as "so much" is missing compared to v2. v2 has better docs, defs, etc. There are two sides to this. First, very few of you were around when v2.0.0 was released. It was absolutely not born in the state it's in today. There were no TypeScript Defs at all, documentation was sparse, you couldn't even browse the docs online, and there were only a hundred or so examples. It took literaly *years* to build it all up. By contrast, Phaser 3 is barely 2.5 months old.

Did we release too early? From the point of view of all the extras, like docs and TS defs, we definitely did. From the point of view of the API itself, we didn't. Yes, we've been working tirelessly on it for the past few months, but most of it is as a result of actually using it in depth and getting community feedback. You guys have been amazing at finding some real obscure bugs and also suggesting great features. There's a really vibrant feedback loop going on in the community and it's something we never had during any of the beta phase. It's a classic open-source issue: People are interested but don't really have time to help while you're in beta, but as soon as you actually release and they try to use it, all of a sudden they start to help. I don't understand it, but I don't blame anyone either, as I'm the same way with other libs I use.

So from an API perspective, releasing it was the absolute right thing to do. It's gone from strength to strength purely as a result of this. It does mean that we've blown semver out of the water, with what is really a major release every week right now, but there's nothing we can do about it beyond moving to a new versioning system (which I'm tempted to consider) - in my mind while I may have been talking about Phaser 3 for years now, the actual end result is still just a newborn, finding its way in the world, and I guess like any parent I'm overly protective. There's a long way to go and a lot of work yet to complete, but that's never stopped me before.

That's it for this Dev Log. Given the amount of things I'm handling at the moment I'm not sure when the next one will be - but I'll try and keep it to a couple of weeks at most. Until then, keep on coding, opening issues and helping out if you can. It really does mean a lot.

As Rick might say: "Wubba lubba dub dub".

phX by Condense is a new Amstrad CPC demo that came 2nd place at the Revision 2018 demo competition. Considering the limitations of the 8-bit hardware on which it is running it's a stunning sight.

Chrome and Firefox will support a new standard for password-free logins.

Japanese are polishing foil balls to perfection, and the result is too satisfying.

Phaser Releases

Phaser 3.6.0 released April 19th 2018.
Phaser CE 2.10.3 released March 21st 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.