Welcome to Issue 116 of Phaser World

Oh wow. It's been quite intense here since the last issue! The big news, of course, is that Phaser 3 shipped. Yes, finally! As a result, this is a special issue

of Phaser World dedicated entirely to Phaser 3. Even the tutorials and games covered are Phaser 3 only.

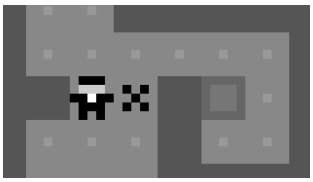
We'll return to normal in issue 117, on March 5th, because you lot are still producing amazing Phaser 2 content :) but for just this issue please allow me to indulge in the 3.0 release and tell you all about it.

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 or Twitter.

What's New?

Phaser 3 Bootstrap Platformer

A great ES6 + Phaser 3 Bootstrap Template and a platform game example to learn from as well!

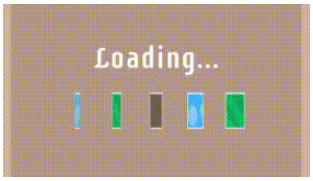


Phaser 3 Sokoban Tutorial

A tutorial and source code for converting the classic sokoban game from Phaser 2 to Phaser 3.

Lessons and Code Part 2

In part 2 learn about passing data between scenes, configuring tweens and more in Phaser 3.



Lessons and Code Part 1

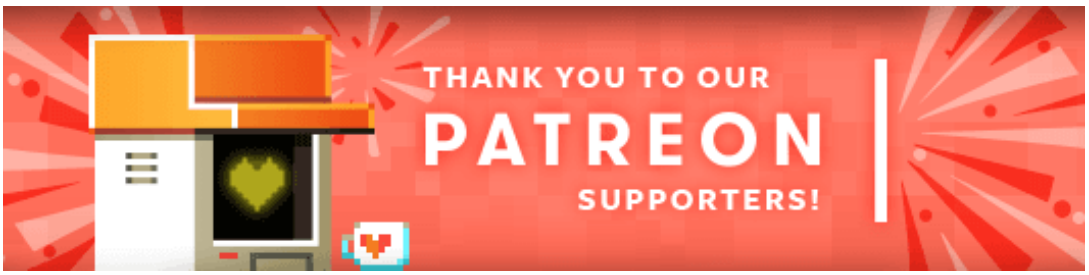
A tutorial written as the developer journeyed through Phaser 3 to build his new game.

Pixel Memory

A nice little matching pairs game with bonus cards and difficulty levels.

Phaser CE v2.10.1 Released

We can't ignore v3s older brother :) another great CE release is out.



Thank you to all of the new and awesome Phaser Patrons who joined us in the past couple of weeks:

Elliott Wallace

First Studio
Anthony De Gorlof
Ryan Malm
Slash
Pablo Lopez Soriano
jestershark
Thomas Hunter II
Shawn Taylor
Kosmoon Studio
Greg
Douglas Lapsley

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 #116

My goodness, we finally did it! After 18+ months of work, numerous rewrites and several false starts, Phaser 3 is officially out. There were points when I didn't think we'd ever make it - when giving up would have been the easy option, but after an utterly insane period leading up to February 13th, I prepared the final build and pushed it to GitHub and npm.

That's right, 3.0.0 actually launched 10 days ago. If you follow me on Twitter or in Slack you will have likely heard about it. I wasn't ready to publish a newsletter at the time because there were things I wanted to get in place first - and honestly, I was utterly exhausted! I had worked myself into the ground and it took a few days to recover.

During this time bug reports started to come in, as we knew they would, and we wanted to prioritize fixing those as quickly as possible over and above *everything*

else. It just didn't feel right to spend time on things like this newsletter, when there were bugs that needed our attention.

So with a laser focus, we worked on them solidly, pushing out 3.1.0, 3.1.1 and 3.1.2 releases. With these versions under our belt, things settled down a little. We had time to take-stock, work on updating the website, write new tutorials and finally publish this newsletter. With that out of the way, let's dive right in!

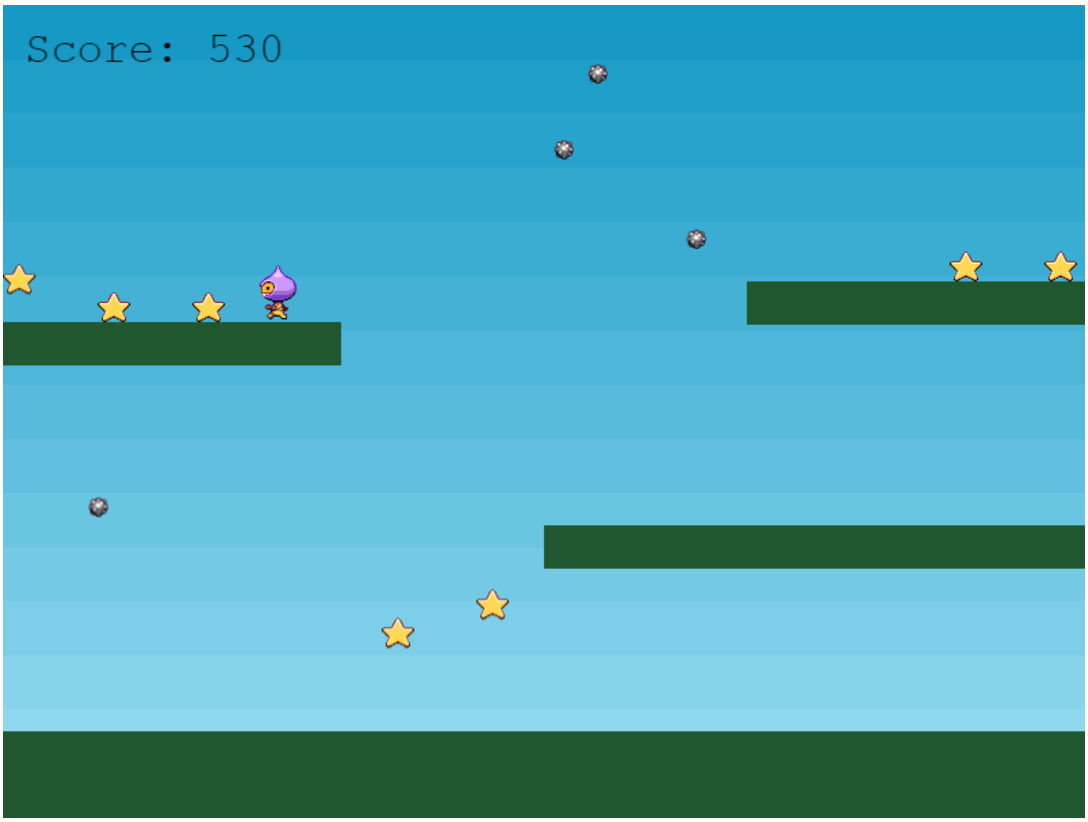
Download Phaser 3

On February 13th we released 3.0.0 and quickly followed with new versions. The current release is 3.1.2 which you can get from GitHub, npm and the jsDelivr CDN. Full details are available here.

Getting Started

We have updated our Getting Started Guide, so it now covers Phaser 2 and 3. If you are brand new to Phaser this is a good place to start.

Familiar with Phaser already? Then we recommend the new Making your first Phaser 3 Game tutorial. This has been rewritten and updated for Phaser 3, covers some of the new features and you get a fun little game at the end of it :)



Phaser 3 Forum

I've shuffled around the Phaser forums, removed the 'beta' one and we now have two forums to pick from: The official Phaser 3 Forum and the Phaser 2 Forum. The v2 forum is basically the old Phaser forum renamed. There is also a 'Demos and Projects' sub-forum within Phaser 3, so you can show-off the things you have made or are working on. Please use it!

Moving from Phaser 2 to Phaser 3 (Part 1)

This is Part 1 of a guide on the differences between Phaser 2 and 3. It's not an exhaustive migration guide, although it will form the basis of one over time. It should help ease the transition, or at the very least stop you going 'Where has that gone?' - Part 2 will follow next week with another 10 changes.

1 - Flat Display List

Phaser 2 used a display tree structure. There was a root display object and then everything else in your game descended from that. Groups had children, which in turn could have more children, each with their own children, and so on. Phaser 3 does not do this. We have an entirely linear structure. Game Objects are not able to contain other objects and Groups no longer have any position or attributes within the World. This allows us to do ...

2 - setDepth

You can now fully control the depth (or z-index) of all Game Objects within a Scene by simply calling setDepth(), or by setting the depth property directly. Please note that Game Objects also have a z property. This is reserved for future 3D use and will not adjust the depth in a 2D Scene. Give the function a value and it can change the position it is rendered at without adjusting its position within the display list itself. Extremely powerful for isometric games.

3 - Cameras

Cameras have been completely rebuilt from scratch in Phaser 3. They were extremely limited in v2 and if you tried to do something like scale one it would cause all kinds of problems. In v3 they are built into everything from the core. You can change the position and size of a camera, you can zoom them, scroll them, shake, flash and fade them. They can also follow Game Objects and maintain their own 'ignore' lists, allowing you to tell them to skip rendering specific Game Objects. Most important of all, you can have multiple cameras per Scene. These cameras can be positioned next to each other, or within each other (i.e. a minimap style camera) - in short, they are everything I ever wanted cameras to be in v2 :)

4 - scrollFactor

In Phaser 2 the function **fixedToCamera** would prevent a Game Object from moving as the camera scrolled. In Phaser 3 this is replaced with the **setScrollFactor** function. Give it a value of zero and the object will stop following the camera entirely. Give it a value of 1 and it's a 1:1 movement with the camera, but you can also set any other value, for example, 0.5 will scroll the object at half the speed the camera is moving. Very handy for creating quick parallax style effects.

5 - Farewell Pixi

When I merged Pixi in with Phaser all those years ago it was simply the best possible thing I could have done at the time. It added a lot of powerful features and for many years we fed back into Pixi and bought their updates down into Phaser. My utmost thanks extend to Matt and the rest of the Pixi team. For Phaser 3 we had our own specific requirements, so worked on creating a renderer suited to those needs. It took a lot of hard work and Felipe has done a fantastic job. The v3 renderer was built specifically for how Phaser works and will be expanded over time. Already significantly faster than v2 there's still a lot we can do, and performance and compatibility will be areas we'll focus on in the coming year.

6 - Anchor / Origin

The anchor property must easily be one of the most misunderstood properties to ever exist in Pixi :) (with pivot being a close second!) - what everyone thought it did (set a registration point) vs. what it was actually for (set a texture offset) were often worlds apart. Needless to say, anchor is gone from v3. To adjust the offset

between an objects position and where it eventually ends up on the screen you should now use the origin properties: **originX** and **originY** or the method **setOrigin(x, y)**.

7 - Center Aligned by default

In Phaser 3 Game Objects are now positioned based on their center by default. This is the v2 equivalent of setting anchor(0.5). If you wish to position an object based on its top-left use **setOrigin(0)**.

8 - Flip not Scale

In Phaser 2 if you wanted to horizontally or vertically flip a Game Object you would need to give it a negative scale. Sprite.scale.x = -1, for example, would flip a sprite horizontally. In Phaser 3 you don't need to do this anymore. Instead, you can use the **flipX** and **flipY** properties to achieve the same thing, without changing the scale at all. Flipping always takes place from the center of the Game Object, regardless of origin.

9 - Pixel Art

In v3 you can add a property to the Game Configuration object: **pixelArt: true**. This will tell the WebGL renderer to automatically create gl textures using a linear filter mode, in short, it'll make your pixel art remain crispy. This is a global setting, so is applied to every single texture created by the renderer. If you want to achieve the same result selectively, on a texture by texture basis, you can call **Texture.setFilter**.

10 - Scenes replace States

In Phaser 2 there was a State Manager and each part of your game took place in a single isolated State. Phaser would run only one State at a time and all of the Game level systems were automatically mapped into this State. Which is how when you access 'this.input' you're really just accessing a reference to Game.input. In Phaser 3 there's no such thing as States any longer. They are now called Scenes, which is a more accurate term and avoids any confusion with 'state machines', a common paradigm employed in games. Scenes can be thought of as isolated 'Worlds'. They have their own camera system, display list, update step, event emitter, physics systems and more. A Scene can be set to sleep and awoken by another Scene and unlike States they can run in parallel,

rendering on-top of each other (the render order can also be controlled.) - Scenes have their own configuration object which allows you to control exactly which systems are created within them. They also have what's known as an Injection Map, which allows you to control which properties are injected into the local Scene object. It's too much to cover in this short entry but they are well worth exploring and looking at the examples for, as there's an awful lot you can do with them.

Phaser 3 Roadmap

With v3 released we can take a nice long holiday, right?! Yeah... dream on :) The release being out in the wild means we're actually now busier than ever, albeit in a good and slightly less stressed way.

Here is what we're currently working on, plus a list of suggestions for future features, because you get to vote on them.

Current v3 Work

1 - Documentation

I spent weeks adding all of the thousands and thousands of jsdoc blocks needed into v3 prior to release. It was a monumental effort and I knew it would require 3

stages. The first was to get the docblocks in and the data-types defined for all of the methods and properties. Even with the tools I built for the task, there was still a lot of manual processes involved, which lead to some errors that I've been slowly fixing. *Once all of the data-types are in and correct we can finally generate the TypeScript definitions!*

Stage 2 is going back through the docs and swapping all of the placeholder '[description]' tags with some actual documentation. There are thousands of these to do, so it will take a while, but my thinking here is that it's something we can chip away at, week by week. It's also something the community could get involved in too. More details on that in a future newsletter.

Finally, Stage 3 of the process is to go back through every method and property, of every class, and ensure each one has an example in the Phaser 3 Examples repo linked with it. Again, this is a massive undertaking, but in the long-run, I feel it will be an invaluable way for you to learn the API. We have always been big on creating examples, but the onus was on you to figure out which one of them may show you what you wanted to learn. By linking from the docs to specific examples we can cut-out that step.

So, that's the long-term docs plan. Of course, they need publishing to the Phaser website as well, but this is a separate task that I'm cracking on with alongside the rest as it's not something anyone else can help with.

2 - Scale Manager

This is the next system to be built. We've got most of the internal work in place for this already, but there's no public-facing API to expose it to you or make it easy to use. So this will be the first new system to be added.

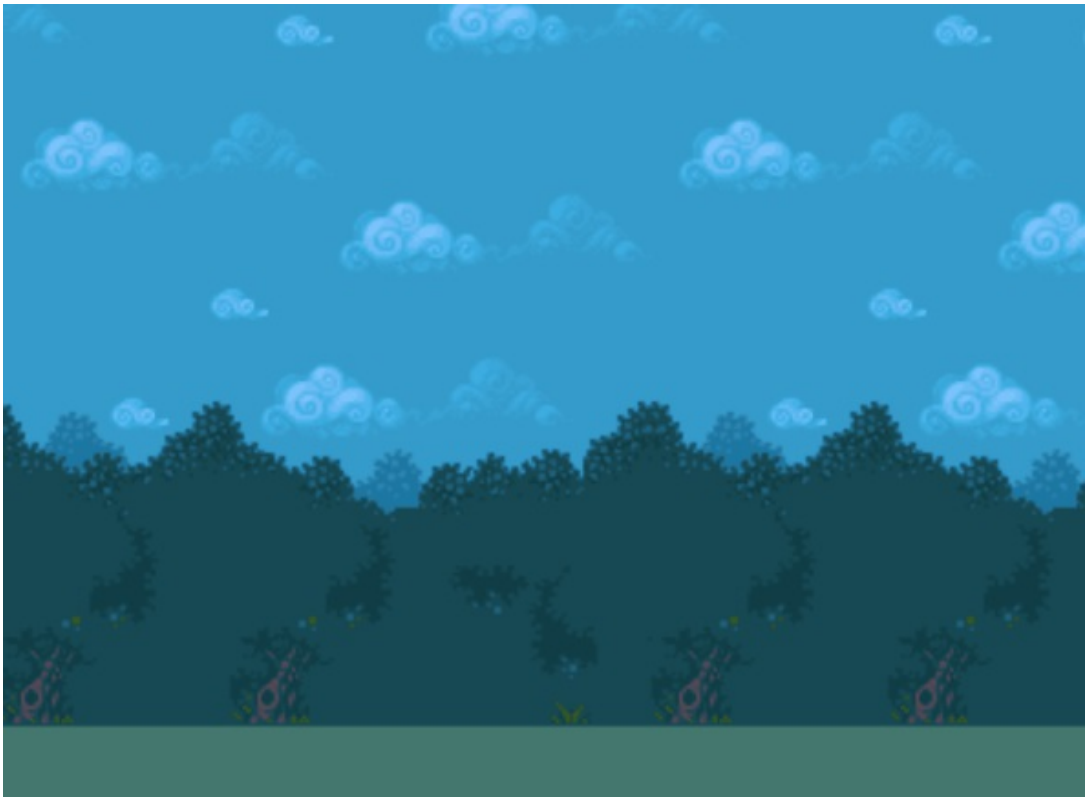
3 - Webpack 4

The latest version of Webpack was released this weekend, so we'll spend some time upgrading to take advantage of its hot new features (including dramatically reduced build times!)

4 - Render Textures

This is actually already in the master branch if you'd like to have a poke around. Felipe is building the ability for you to dynamically draw anything onto a texture. You can draw images anywhere on the texture, as fast as you like. The Render Texture itself can then be displayed in your game, or for maximum power you can use it as a Bitmap Mask! This will allow you to do some really cool effects,

such as revealing or erasing images, like so:

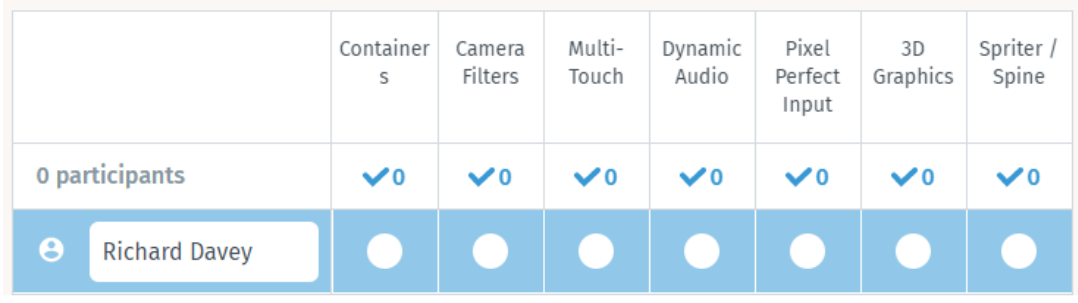


As well as this we will keep an eye on issues reported on GitHub and keep jumping on those where possible.

Vote on new features

What features should come next? We've a long list of ideas, so we're going to allow you to vote on them - then we'll use the votes as guidance when picking what to work on after the current tasks.

You can vote by clicking the image below or going here. Votes are private, which means only I can see them, no-one else. You can optionally leave a note if you'd like to suggest something or comment on what you voted for.



The vote options are:

Containers - A container would allow you to add children to it (a child being any form of Game Object) and then have transforms applied to all children. Much like how 'addChild' worked in Phaser 2.

Camera Filters - Allow you to define your own shader to be applied as a post-processing step for a Camera.

Multi-Touch - Currently v3 is set to work with single touch events and doesn't support more than this, which means you can't do gestures like pinch and zoom yet.

Dynamic Audio - The ability to generate audio effects dynamically via an API similar to sfxr.

Pixel Perfect Input - Expand the Input hit detection so you can tell it to only include clicks on non-transparent pixels in the image (if using a texture based input.)

3D Graphics - We actually did a good amount of work towards this last year, so it would be the continuation of this. For our first approach we want to expose it as a Graphics3D Game Object, to which you can render 3D objects, lights, etc in-game within an isolated container (rather than just making the whole of Phaser 3D) - we feel this will give you a good amount of power while keeping it quicker to implement.

Spriter / **Spine Support** - Take the provided native runtime libs for these bone-based animation packages and ensure they work properly in Phaser 3. This is quite a lot of work but I know how important it is to some of you, so it's on the list! If you pick this option please leave a comment saying which package you want us to support first (you can also include Dragonbones or Creature in your choices)

And that's it! I'll be keen to see what you all pick :) Remember, this just gives us a way to prioritize tasks. We hope to do them all *eventually* if funding allows.

Phaser 3 Tutorial - Input Events

Cursors to move + Drag the Sprites

I've seen this asked about a few times in the forum and Slack, so I think it's worth covering: How to respond to input events in Phaser 3.

There are 3 different ways of dealing with input in Phaser 3. The first allows you to respond to pointer events that are emitted directly from the Input system. There are 3 events you can listen for:

```
this.input.on('pointerdown', function (pointer) {} );
this.input.on('pointerup', function (pointer) {} );
this.input.on('pointermove', function (pointer) {} );
```

These events are emitted regardless if you have any interactive Game Objects or not. The function is sent two arguments: the Pointer that caused the event, and an array. The array contains all *interactive* Game Objects that were below the pointer at the time of the event. To get the position of the pointer you can check the pointer.x and pointer.y values in your callback.

I've mentioned twice now about interactive objects, so what are they? Any Game Object in Phaser 3 can be set to be interactive. You do this by calling the setInteractive method:

```
this.add.sprite(400, 300, 'akira').setInteractive();
```

Here we set a Sprite to be interactive. Once a Game Object is interactive it will be considered as 'interesting' by the Input system. There are two ways to get input events for a Game Object. The first is by listening for the events directly from the Game Object itself:

```
var sprite = this.add.sprite(400, 300, 'akira').setInteractive();

sprite.on('pointerover', function (pointer, x, y) {

  this.setTint(0xff0000);

});

sprite.on('pointerout', function (pointer) {

  this.clearTint();

});
```

Here you can see we have made a Sprite interactive and are then listening for the pointerover and pointerout events. If the pointer goes over the sprite we tint it red, and clear the tint when it leaves.

The input events a Game Object emits are: pointerdown, pointerup, pointermove, pointerover and pointerout. If it has been enabled for dragging it can also emit: dragstart, dragover, dragleave, dragenter, drag, drop and dragend.

One of the benefits of using events directly on the Game Object is that the context of the callback is automatically scoped to be the Game Object instance, which is why we're able to use 'this.setTint' in the callback above and have it work without needing to bind a context to it.

There is also a third way to receive input events on Game Objects. You can listen to the Input system for 'gameobject' events. For example:

```
this.input.on('gameobjectdown', function (pointer, gameObject) {

  gameObject.setTint(0x00ff00);

});

this.input.on('gameobjectout', function (pointer, gameObject) {

  gameObject.clearTint();

});

this.input.on('gameobjectup', function (pointer, gameObject) {

  gameObject.clearTint();

});
```

The Input system will emit these events when it encounters an interactive object. The default setting in Phaser 3 is that only the top-most Game Object will receive and emit input events.This means if you have 2 interactive objects that overlap on the display list, and you were to say click them, only events for the top-most object will dispatch.

You can change this behavior by setting **this.input.setTopOnly(false)** which tells the Input system to not favor the top-most object and instead treat them all the same.

Custom Hit Areas

The **setInteractive** method is more powerful than it may first appear. You can optionally pass it a geometry object and a custom callback. When you click within the bounds of the geometry it will invoke your callback. This needs to return a boolean to tell the Input system if it 'hit' or not.

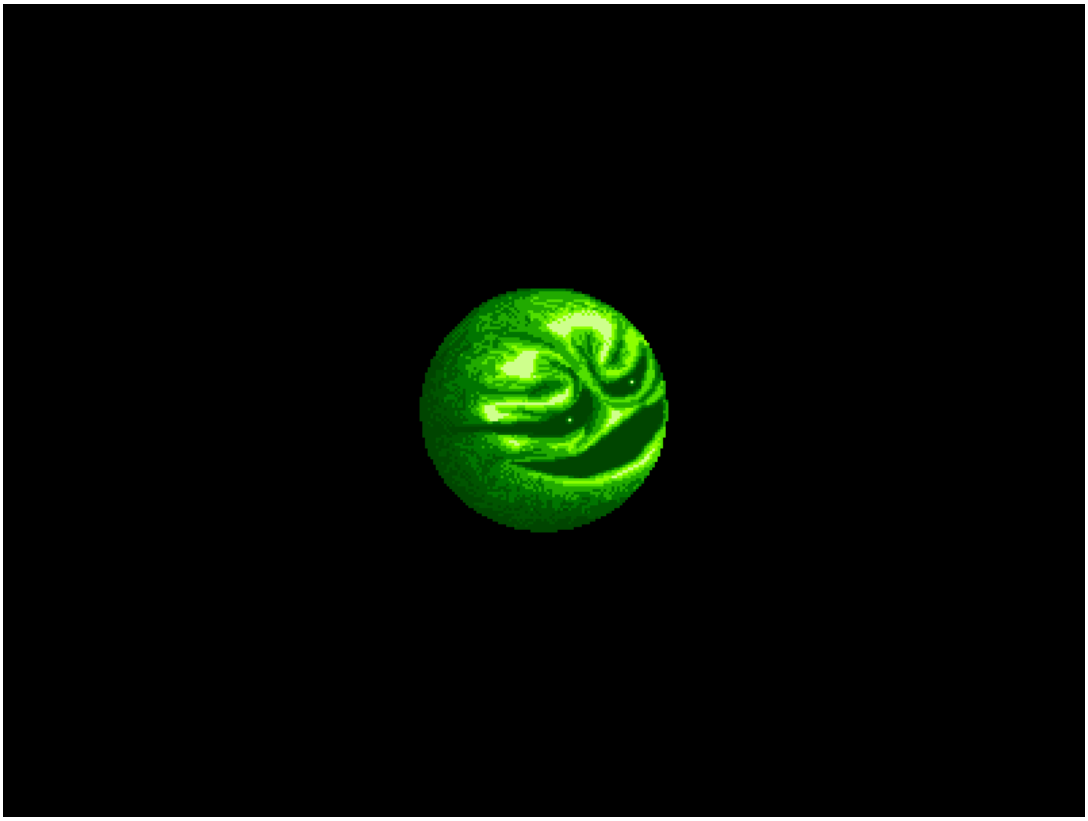
For example, if you wanted to use a Circle shape as a hit area for a Sprite instead, you could do something like this:

```
var sprite = this.add.sprite(400, 300, 'ball');

var shape = new Phaser.Geom.Circle(46, 45, 45);

sprite.setInteractive(shape, Phaser.Geom.Circle.Contains);
```

We create a Sprite using a ball image, so we want a circular hit area. We create a Circle geometry object (the values given just match the size of the ball image the Sprite uses). This shape is then passed to the setInteractive method, along with a callback. In this case we don't need any fancy processing so we can use the 'Circle.Contains' function as our callback. Have a look at the following example to see this in action:



The end result is that now only clicks within the circle shape will be considered valid. The shape applies whatever transforms the Sprite may have to itself. So if you scale or position the Sprite, it will compensate for that.

You don't have to use Geometry though. Any object can be used as the hit area shape and any callback too. This allows you to craft as complex an interaction system as you need.

Hopefully, this tutorial has given you more insight into how Input events work in Phaser 3 and how to use them in your own games.

Phaser Releases

Phaser 3.1.2 released February 23rd 2018.
Phaser CE 2.10.1 released February 18th 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.