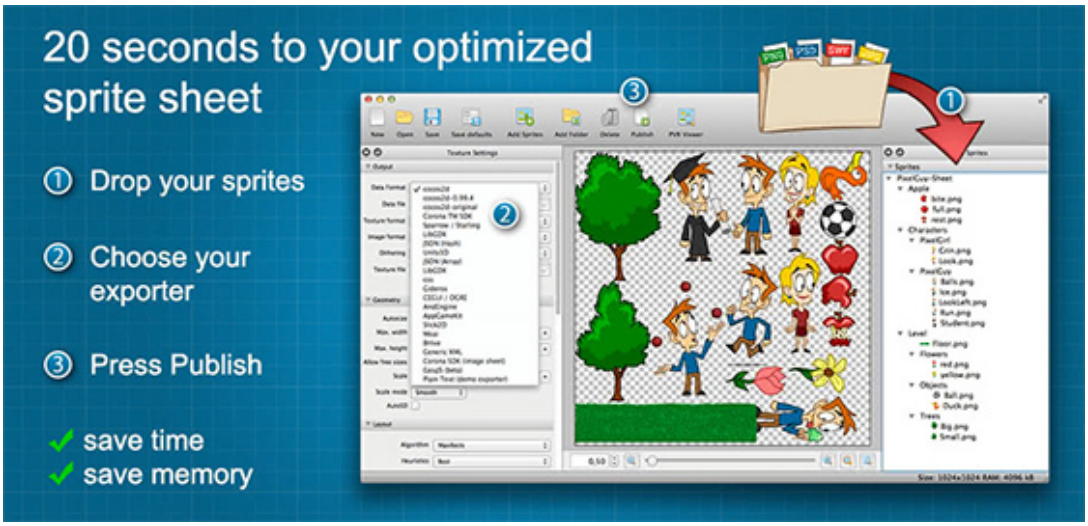
Web Version Unsubscribe

Welcome to Issue 132 of Phaser World

Happy Halloween! We've got a pretty packed issue here, including a superb special offer on Texture Packer, the Phaser stickers are now shipping, there are some awesome new games and tutorials - and not forgetting, a huge Dev Log all about Spine support, context switching and plenty more.

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.



50% Indie Discount on Texture Packer

If you're even slightly serious about getting the best performance from your games, then you should already be aware of how essential a decent texture atlas app is. Using atlases saves memory, network requests and draw calls. I cannot state enough how important they are. To create an atlas I personally have been using the app Texture Packer for many years now and consider it part of my day-to-day workflow.

Which is why I wanted to mention that for the next couple of days you can get a **50% Indie Developer Discount on Texture Packer**. The discount also covers the apps Sprite Illuminator and Physics Editor, which all work great together. The license includes a year of free updates too.

Click here to get a 50% Discount on Texture Packer

Phaser Sticker Packs are Shipping!

I'm pleased to say that I've taken final delivery of all the Phaser stickers and will start mailing them out on Monday. I printed enough to cover all pre-orders and then an extra 100 units. Once those are gone, they're gone. Every pack sold helps contribute a little towards the development of Phaser.

You'll get 10 high-quality durable vinyl stickers, perfect for a laptop, or plastering all over your den. You also get a Phaser magnet and finally a nice double-sided glossy pixel art print.

Click here to order a sticker pack.

If you support Phaser on Patreon check your personal messages for a custom discounted link.

The Latest Games

Game of the Week

Yatzy Halloween

Why are these dice looking so scary and tiny pumpkins are everywhere? What strange event is going to hit Yatzy for Halloween?

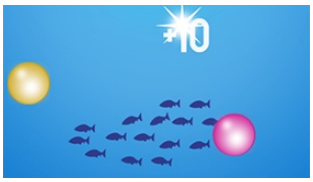
Staff Pick

MudWars.io

Tunneling fun game! Compete with other players in a huge arena to tunnel through the mud, destroy the gold nuggets and power-up your tank.

Madcap Mahjong

Match the tiles and clear the board in this crazy take on solitaire mahjong.



Blob 'n Pop!

Pop the bubbles, avoiding the forbidden color. Pop color combos to reach the highest score.

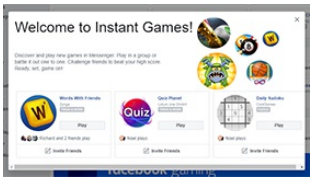
Poo Jumper

You're a runner, out for a run - can you avoid all of the dog poo in order to get the highest score possible?!

What's New?

Phaser 3.15.0 Released

This important release contains several high priority fixes for a performance regression and iOS input locking bug, as well as several new features.

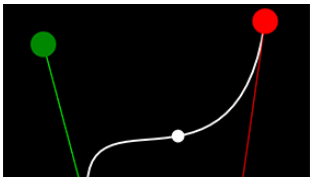


Facebook Instant Games Phaser Tutorial

The first in a series of long-form tutorials covering the process of creating a Facebook Instant Game with Phaser 3.

Managing Big Maps in Phaser 3

A tutorial on splitting your game world up into chunks and progressively handling it as the player moves around.

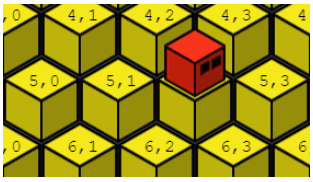


Tweens and Curves Tutorial

Learn how to use Tweens with curves and Bezier Curves with full source code.

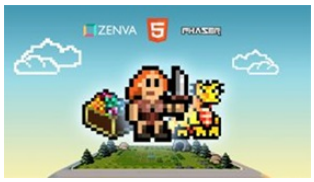
UI Blocks

A lightweight alternative to Containers for Phaser 3 Games.



Down the Mountain Tutorial

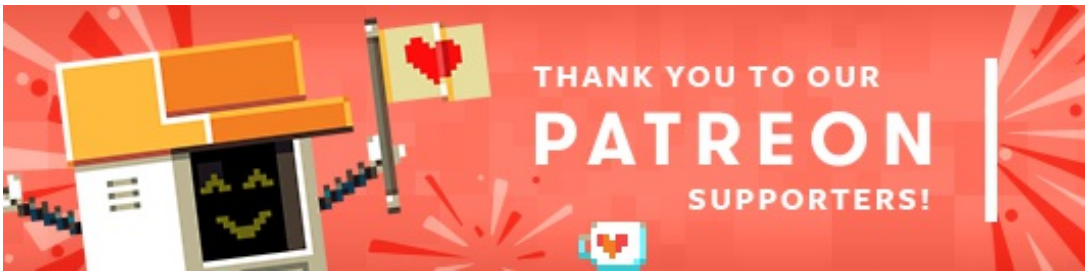
In this tutorial Emanuele updates his “Down the Mountain” prototype to Phaser 3 with full source code.



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.

Try this 4 course sample for free!



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

Andreas Löw (Texture Packer)
Gaelen Hadlett

Also, thank you to **Ashley Williams** for increasing their pledge.

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 #132

Welcome to Dev Log 132. I needed a break from the newsletter last week, so this one encompasses two weeks worth of development updates. Recent work has centered around adding Spine support in Phaser 3. It's been a really interesting journey and is the main topic for this Dev Log.

Phaser 3 Doc Jam is nearly over!

I announced the Phaser Doc Jam 3 issues ago and since then we have collectively completed the documentation for 2,472 items. This is an incredible amount for such a short space of time. It means there are just 980 items left

before Phaser 3 has 100% API documentation coverage. Could you help get that number to zero?

All you need do is point your browser at docjam.phaser.io. When the page loads it will randomly pick one of the descriptions that need writing. Look at the surrounding code and see if you can infer the meaning of what's going on. If so, please enter a description.

Everyone who contributes a description, which I then approve, will be automatically entered into a prize draw. You get one entry into the draw for every description that is approved.

At the end of November, I will pick 6 winners. The first two will win a **$100 Amazon gift voucher** each. The remaining 4 will each win a $50 gift voucher. The vouchers will be sent digitally, in time for you to spend on Christmas presents :) This is entirely optional. If you'd like to just help for the sake of helping, then you don't need to give your email address at all. However, I feel it should make what is quite an arduous task at least a little more fun.

Spine Development - Skeletons Ahoy

Spine is an app by Esoteric Software specifically for creating 2D skeletal animation. Animation in Spine is handled by attaching images to bones, then animating the bones. This is called skeletal or cutout animation and has numerous benefits over traditional, frame-by-frame animation:

Smaller size - Traditional animation requires an image for each frame of animation. Spine animations store only the bone data, which is very small, allowing you to pack your game full of unique animations.

Art requirements - Spine animations require much fewer art assets, freeing up time and money better spent on the game.

Smoothness - Spine animations use interpolation so animation is always as smooth as the frame rate. Animations can be played in slow motion with no loss in quality.

Attachments - Images attached to bones can be swapped to outfit a character with different items and effects. Animations can be reused for characters that look different, saving countless hours.

Mixing - Animations can be blended together. For example, a character could play a shoot animation while also playing a walk, run or swim animation. Changing from one animation to another can be smoothly crossfaded.

Procedural animation - Bones can be manipulated through code, allowing for effects like shooting toward the mouse position, looking toward nearby enemies, or leaning forward when running up hill.

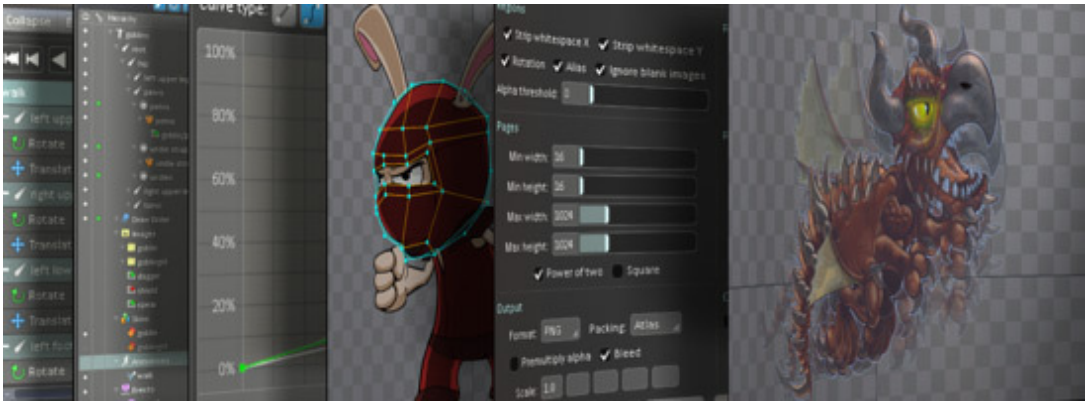
The software has many features, such as a dopesheet, graph editor, meshes, free-form deformation, weights, inverse kinematics and more. You can read all about them on the Spine web site. From this point on, I'll assume you understand what the software does and are just interested in how you can use it in your Phaser games.

Spine Plugin

The authors of Spine provide runtimes, so you can integrate Spine Animations into your games. When it comes to HTML5 they provide both a Canvas and a

WebGL runtime. The original runtimes are written in TypeScript and compiled down to ES5 JavaScript. This has the advantage of TypeScript definition files being available from the start, something I really didn't want to have to create manually!

When it came to adding Spine support to Phaser, I was determined from the outset to use their runtimes and not re-inventing the wheel by coding a custom renderer. There are several benefits doing it this way around. First, when Esoteric release new runtimes, such as when they add new features to Spine, it will be hopefully quite trivial to add these into the Phaser plugin. In theory, we should be able to just swap-out the runtimes and they'll just work. It depends, of course, on how much the runtimes have changed. Although it's unlikely that the core API will alter much, you never know. Even so, I feel this future protects us in a much better way than any other approach.



The other obvious benefit of using their runtimes is that by *not* converting them, I'm not accidentally introducing any bugs either. If the core runtime has an issue, then it's the Spine developers who are responsible for fixing it. Equally, I'm not second-guessing their intentions by trying to translate their code into anything else. This is especially true because the runtime files are fully un-documented. Documentation is provided on their web site, but the source itself doesn't have a single line of it. Which made the creation of the plugin slightly more challenging than usual, but not impossible.

There is one down-side to using the runtimes, however. They provide a runtime for Canvas and another for WebGL, but there is no single runtime that supports them both. The Canvas runtime is 247 KB (unminified) and the WebGL one is 343 KB (again, unminified). Yet, both runtimes share a huge amount of code. I would hazard a guess that around 80% of each runtime is identical. There are also classes within each runtime that I simply don't use. The Asset Loader, for example, is skipped entirely in favor of letting Phaser's own Loader handle all of this. The WebGL Runtime also includes a few special effects built-in, like the jitter

and twirl filters, even if you don't need them. They're not large, but they do add to the size.

It's not all bad, though.

Minified, the Canvas runtime is 51.1% smaller at 121 KB, which is 33 KB over the wire (gzipped). The WebGL runtime is 48.9% smaller at 175 KB minified, or 46 KB gzipped. That does mean an extra 79 KB to your payload if you wish to support both runtimes in production. To me, this feels like a fair price to pay, considering all the animation power you get. Especially if you can target a single renderer. The Phaser plugin adds a bit to the weight, but not much if you include the plugin into your build process, as it'll share lots of resources with the main library.

License Requirements

There's one final, very important point I need to mention before we dive in to the plugin. You are legally not allowed to use the Spine runtimes, and by extension the Phaser Spine plugin, *unless* you own a copy of the software. The runtime license allows you to use it for testing purposes, but if you want to use them in a released game, *even if that game is free*, you need the full license from Esoteric Software, which is what you get when you buy Spine. Given that Spine is an incredible tool, and that prices start from $69 at the time of writing, it's a worthy investment if you're serious about animation.

Right, with all of that out of the way, let's jump in!

Using the Spine Plugin

For the sake of ease-of-use, I'm going to be loading the Spine plugin at run-time in these examples. You don't have to do this for your games. You could easily bundle the plugin in to your build. For testing, however, it was just faster for me using this method. This means in all the example code you'll see the plugin included in the Game Config, like so:

```
var config = {
    type: Phaser.CANVAS,
    parent: 'phaser-example',
    scene: {
        preload: preload,
        create: create,
        pack: {
            files: [
                {
                    type: 'scenePlugin',
                    key: 'SpineCanvasPlugin',
                    url: 'plugins/SpineCanvasPlugin.js',
                    sceneKey: 'spine'
                }
            ]
        }
    }
};
```

This is using the Pack Loader feature of Phaser to grab the plugin and load it before the Scene is started. The reason for this is that the plugin creates both a new File Type and a new Game Object, and we want to be able to access these in our `preload` method. With the plugin loaded, we can now load some Spine data:

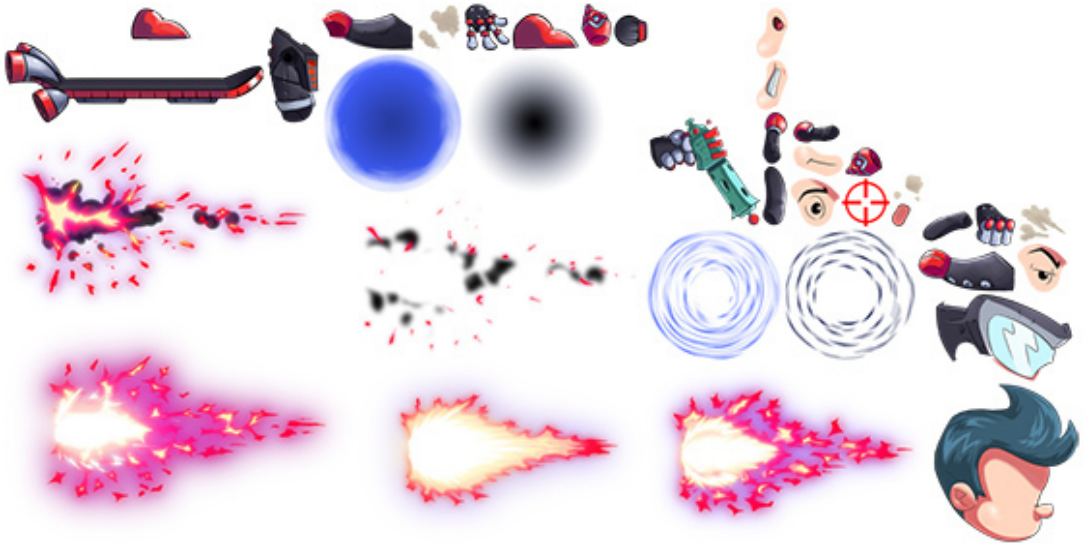
```
function preload ()
{
    this.load.setPath('assets/animations/spine/');

    this.load.spine('boy', 'spineboy.json', 'spineboy.atlas');
}
```

Here we invoke `load.spine`, which is the new File Type handler that the plugin adds to Phaser. It takes three arguments: A key, like all assets loaded in to Phaser, with which you'll reference this asset by later. A JSON file and an Atlas file. When you export an animation from Spine it creates 3 files at a minimum: The JSON file holds all of the data, such as the skeletons, animations, bones, slots, skins and events. This data is 'global', in that it's not tied to specific textures. This is what allows you to use the same animation across multiple objects by sharing the bone structure.

The second file is an Atlas file. As the name implies, this is a custom Spine-format texture atlas file. It's raw text, so you can open it and have a look at the

structure. You'll see it's full of coordinates, offsets and rotation values for each frame of the texture. Spine creates the texture atlas files during export and those images, along with the data files, are saved when this happens. Depending on the size of the textures it may split them across multiple image files.



The Phaser Spine Loader will take the atlas file and figure out which image files it needs to load, based on the contents. It then handles this for you. So it doesn't matter if the export produces 1 or 10 texture files, you just need to point it at the Atlas and it'll pick it up from there.

The Spine Game Object

With the animation data and textures loaded, we can add a Spine Game Object in to our Scene. The plugin creates a custom Game Object for this purpose, which you can create via the Game Object Factory, as usual: (see Dev Log 131 for details about factories in Phaser):

```
function create ()
{
    this.add.spine(400, 300, 'boy', 'idle', true);
}
```

In the above code, `boy` is the key we used when loading the Spine animation data (the Spine JSON file). The following arguments are an optional animation name and loop flag. In this case, we're telling it to play the 'idle' animation, in a loop. The object will be placed at 400 x 300 in our Scene and will look like this (click the image to run the demo)

And there we have it, one smoothly moving skeletal animation! And it's a normal Game Object, too, which means you can do most of the things you'd expect with it, such as scaling, rotating, flipping, changing its alpha, setting the render depth, tweening and so on. They respond properly to the camera system, as well, so panning or zooming the camera will of course update the Spine objects. You can even add them to Containers.

There are a few key differences, though. For a start, you cannot set the origin of a Spine object in the same way as you do a normal Sprite. The origin is defined in the software itself and exported with the skeleton data. This allows you to ensure that different skins all use the same origins, so you can properly align animation sequences.

Another difference is with the angle of Spine objects. By default, Game Object's work using a clockwise rotation system, where 0 degrees points to the East, 90 points South, 180 (and minus 180) point West and -90 point North. This is a typical rotation system found in game frameworks. We adopted it in Phaser 2 when we used Pixi and kept it since then.

Spine works differently, in that rotation is counter-clockwise and 0 degrees is North. To avoid this getting really confusing in your game code the Spine plugin automatically adjusts itself, modifying the rotation system to match and compensating for the 90 degree difference between the two. This means, when

you've a Spine Game Object with an angle of zero, it'll render as you'd expect, although internally Phaser is desperate to render it rotated and pointing to the right :) All this means, from your perspective, is that if you are trying to sync-up the rotation of a Spine and non-Spine object, you'll need to allow for the 90 degrees difference in orientation.

Mixing Animations

Spine has a neat feature that allows you to mix animations together. This is explained by Esoteric: "An often felt disadvantage of 2D games is the lack of smooth animation transitions. In 3D games, transitions between animations can be calculated on-the-fly at runtime. Animations can even be blended, for example half walking and half running. In 2D games without Spine, blending is impossible and typically transitions are jarring. Artists can manually create frames for every possible transition, but even that doesn't help when animations can be interrupted mid playback.

Spine brings the benefits from the 3D world back to 2D. The Spine Runtimes smoothly and dynamically transition from one animation to the next, giving your characters natural looking fluidity. Layering allows animations to be blended on top of others, for example to play a shooting animation while your character is running or to blend walking and limping more and more as damage is taken."

Support for this is built into the plugin. You just define a mix animation, which takes two animations and controls how to blend from one to the other, and Spine will handle the transition for you.

Use cursors to move the camera

Mesh Deformation

Although this is a feature of the app, more than anything, it's also fully available via the plugin. Have a look at the awesome demo to see just what can be achieved.

"A Spine character made up of rigid 2D images can already give excellent results, such as Spineboy in the demo above. To give your characters even more life, Spine brings more tricks from the 3D world in the form of meshes and weights. With meshes, images are no longer rigid and can bend and deform however you like. Weights bind meshes to bones, so the images deform automatically as the bones are moved.

Meshes can also improve your game's performance, reducing fill rate usage by not drawing transparent portions of your images. This is especially important on mobile devices."

Custom Context

When integrating the Spine runtimes with Phaser, one of the most interesting challenges was what to do at the point where a Spine object is about to be rendered. I needed to hand over control to Spine, let it do its thing and then take control back again at the end, continuing to render the Scene. For Canvas, this is trivial. You just give it a reference to the context, it draws on it, then you carry on your merry way. For WebGL, however, it's a bit more involved.

The first step is to cleanly clear down the WebGL context. The current pipeline needs flushing and resources tidied up. At this point, Spine can take over, binding its shaders and starting its own batcher. It then renders the skeletons, using its internal Skeleton Renderer. At the end, I need to clear down what Spine has done, resetting the blend modes, clearing out its shader and vertex buffers and restoring access to Phaser. All of these were tasks the Phaser WebGL pipeline was not originally designed to handle. Swapping between pipelines is handled, of course, but not to the extent of passing control fully over to a non-pipeline.

So, I spent a while adding new functions in to the renderer to support this. The new `clearPipeline` method tidies-up house, making sure that it doesn't matter what came before in the display list, the new system gets a clean gl context to draw to. When it's time to pass back to Phaser again, the new `rebindPipeline` method is called. This performs tasks such as clearing the depth buffer, resetting the blend mode and putting things back the way they were. In doing this work, I realized I had, almost inadvertently, opened-up quite a range of possibilities.

To test my hunch, I tried mixing the popular 3D library three.js with Phaser. After a few small tweaks, it worked great. Just to be clear, this was three.js rendering in to the exact same canvas as Phaser was using. It scanned the display list, rendered a few Phaser Images, then hit the new Extern object, which passed full control over to three.js, which rendered a full 3D scene, and then returned to Phaser, which carried on adding the rest of the images on the display list. It's a powerful combination, and not limited to three.js either, literally any 3rd party library should be able to take advantage of this.

I'll cover this in more detail in a future Dev Log, as it's a really quite exciting feature :) and a lovely side-effect of the work involved creating the Spine plugin.

Release Schedule

I'm expecting to completely finish the Spine plugin by the end of next week, which will be November 9th. This will include examples and documentation. After that, I will release a beta to npm for testing and feedback. Then, it can bed down for a couple of weeks while I finish the Scale Manager. Both systems will be included in the Phaser 3.16 release together, towards the middle of November.

After this, I need to return to GitHub and check out any important issues or pull requests. My aim, is that by the end of this year, I'll have finished off the final parts of the documentation, 3.16 will be out (or realistically, 3.17 by then) and I'll beat the issues list into submission.

I've some big plans for 2019, not least of which includes refactoring Phaser 3 in either ES6 or TypeScript, and working carefully on the build process so we can get the runtime size down as small as can be. However, it largely depends on funding. A few people have been asking on Slack and Discord about how Phaser is funded, which I feel is a worthy topic of discussion in a future Dev Log. The short answer, though, is that I'm personally working *full-time* on Phaser and am not doing any client work at all (and have not been for most of this year), because Phaser needs all of my time, and, if I'm honest, because Phaser is where my true passion lies anyway. I'm just about getting-by on the money that Patreon and other sales bring in, and it's my sole objective to make the project financially self-sustainable in 2019, *including* my salary and expenses (something it has never

been before). I'll cover this all in more depth soon. Suffice to say, November is going to be a busy month for releases and then things will calm a little in December, so I'm recharged and raring to go come 2019.

Failing to Fail: The Spiderweb Software way. If you watch one GDC talk, watch this. It's both insightful and fascinating - all about how they've been indies since 1994 and what they learned along the way.

The V8 Blog has a fascinating discussion about what's involved in sorting an array in JavaScript, from the engine-level. It's probably a lot more than you'd think!

PRNT SCRN is a podcast hosted by Dorothy R. Santos about bridging the gaps between analog, new media, and digital art practices.

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.