Web Version Unsubscribe

Welcome to Issue 125 of Phaser World

And it's another packed newsletter :) There has been a lot going on recently, with Phaser 3.12 Beta 2 released, loads of new games and tutorials and ever more people chatting in the Slack and Discord channels. It's great to see Phaser 3 being used for high profile Disney ad games.

I'm going to be taking a short holiday, so we'll be back on August 27th with issue 126, which should also be very close to the final 3.12 release as well.

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.

Emanuele Feronato is easily the most prolific author of Phaser tutorials on the planet. He has published over 250 of them in the past few years and started writing Phaser 3 tutorials before most others. As the framework has evolved, his tutorials have kept pace. So it's great to see that he's been busy writing a book and it's now available.

It's 155 pages long, with 28 source code examples, taking you through the process of creating a full cross platform game. What's more, buying a copy actively contributes towards funding my work on Phaser too!

The Latest Games

Game of the Week

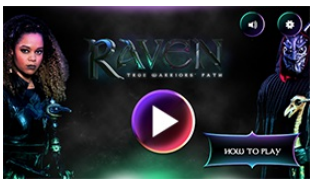
Footchinko Russia 18

Discover this unique version of a soccer classic that mixes pachinko and pinball elements. Choose your favorite team and join the World Cup!

Staff Pick

Incredibles 2 Cookie Catch

Control Jack-Jack and catch as many cookies as possible, while avoiding the milk, in this film tie-in game.



Staff Pick

Raven: True Warriors' Path

Demons scourer this land, to defeat them you must use the magic you have learned as a true warrior.

Online Mahjong Solitaire

The classic tile matching game given a lovely visual feel, with new levels added every day.

Plow Games Retro Tour

Pick a character and go on a tour of the Plow offices in this 16-bit pixel art homage.

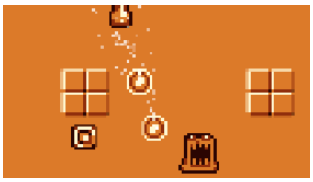
What's New?

Spanish Phaser Book

A great 130 page book with full source code and assets covering each step of creating games in Phaser and Impact.

Converting a Flash Game into HTML5

A comprehensive tutorial on converting Flash games to Phaser and how to get the best out of it.

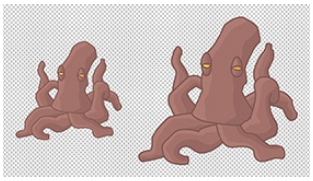


Phaser 3 Tilemap & File Pack Project Template

A complete template package to let you make overhead 2D tilemap games quickly, including a great RPG style demo game.

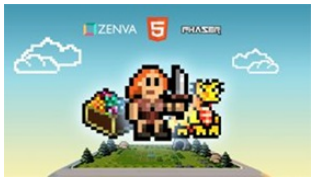
Palette Swapping Example

A source code example and explanation about using palette swapping on a Sprite Sheet in Phaser 3.



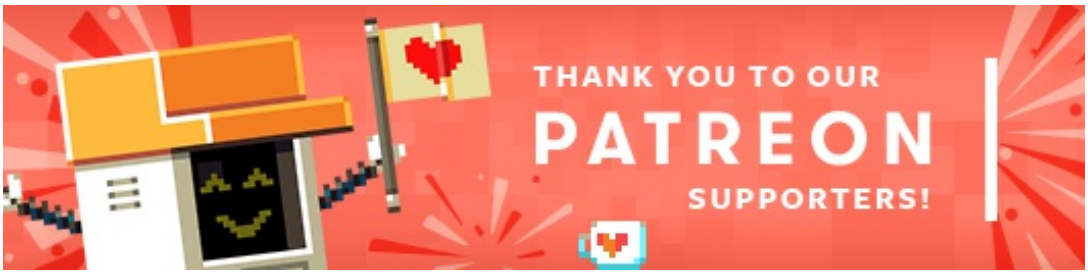
Retina Tutorial

A tutorial on supporting retina images in your Phaser 3 game, with demo and sample 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.



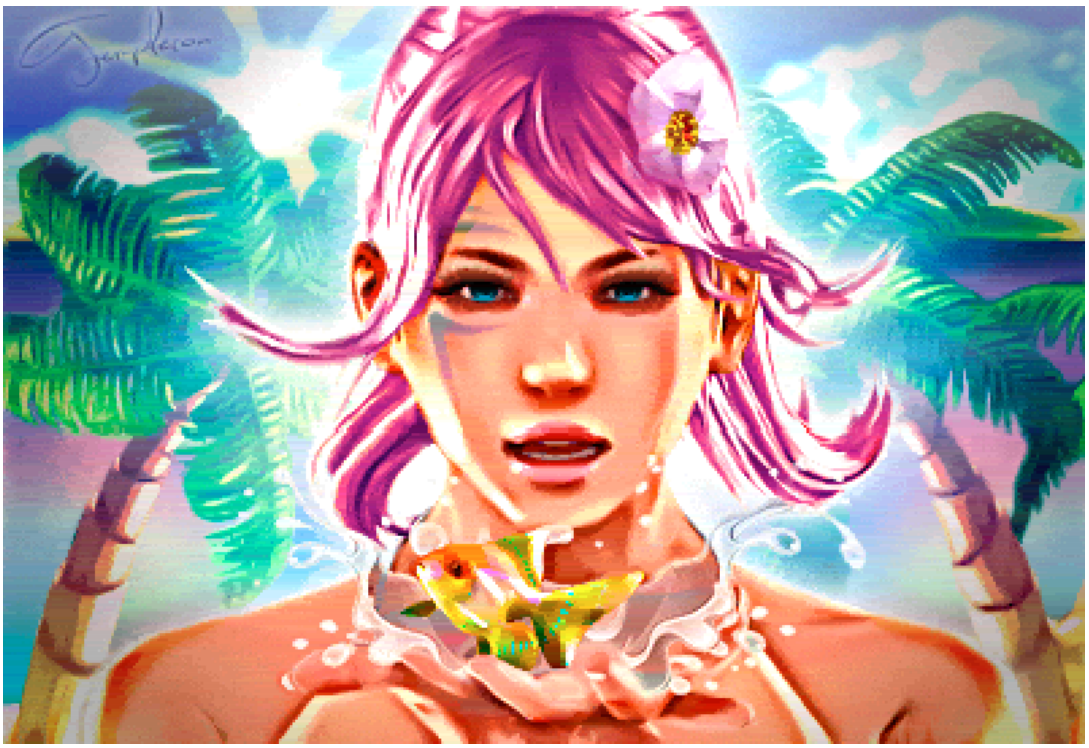
Thank you to these awesome Phaser Patrons who joined us recently:

Anton

also, thank you **Andrew Ozdion** 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.

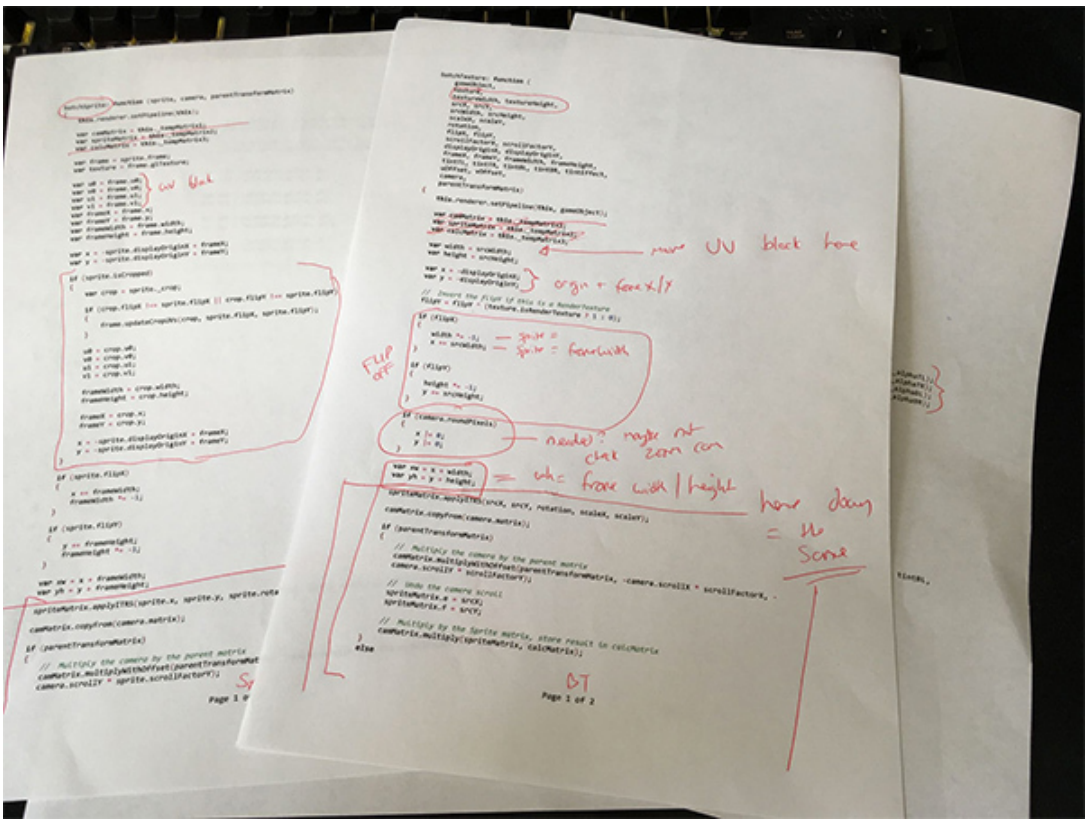
Backers get forum badges, discounts on plugins and books, and are entitled to free monthly coding support via Slack or Skype.



Dev Log #125

Last week was one of those weeks where a seemingly easy trip in to fixing one Game Object lead to a refactor that was as deep as it was wide. Making it an intense week of development for sure. I find that coding often feels like juggling. Depending on the task you can be mentally managing either just a few 'balls' in the air, or a whole stack. It's safe to say that week I could barely see the sky for them.

At one stage it got to the point where I had to actually print out two functions to compare their code. I can't remember the last time I had to print out some source. Can you? After dusting off the printer, and being amazed it had some ink in it, I realized that you can't actually print anything from VS Code! It's so rare you actually need an extension for it :)



Once I'd wrestled with the printer I was pleased I had perservered. It was a really helpful process. I could line-up the functions on each sheet of paper next to each other and draw from one path to another, circling sections, crossing others out. It was a nice and visual way to refactor the code and when it came to making the changes they worked first time. Don't get me wrong, I usually just set VS Code to fill my ultra-wide monitor so I can have 3 editor panels side-by-side, but sometimes it helps to plot things out in a physical way that's just not possible within your IDE. Paper to the rescue yet again.

Render to a Texture with a Render Texture

So, what has all the work been? It actually covers a lot of areas but it all started with the **Render Textures**. The idea behind a Render Texture is that you can draw to it and whatever you draw remains there on the texture. For example, if you had a visually complex, yet static, backdrop in your game then you could draw all the pieces to the Render Texture and use that as the backdrop instead. If you've got a lot of elements then you save a whole stack of draw calls, as once it has been created the renderer only needs to draw the Render Texture itself, rather than every individual thing on it.

That's the theory, at least.

It's safe to say that Render Textures in v3 were not as powerful as in v2. You

could only draw a texture frame to them and if you wanted to draw the frame slightly rotated, or scaled, then you had to use transform operations on the Render Texture itself. I.e. if you wanted to draw a bunch of textures in a fan-like pattern then you'd need to transform the Render Texture, then draw the texture frame, then transform it again, and so on until the effect was complete. There was also no ability for you to draw a Game Object directly to a Render Texture either, meaning you couldn't say "draw this Sprite here". More complex objects like Graphics, Containers or Bitmap Text were completely out of the question without a huge chunk of custom code.

While they mostly worked, they weren't very flexible and also suffered from a number of bugs. It was while going through the GitHub Issues I saw a couple of Render Texture related issues raised by Tom Atom / SBC Games. The issues seemed relatively easy to solve, just things like error warnings in Firefox and incorrect viewport sizing. I set about fixing these and quickly had them working a lot better. Even so, I was frustrated because in v2 you could draw a Sprite to a Render Texture and I wanted that to be possible in v3. It didn't take long to refactor it to support this. I was pleased because mission accomplished, yes? Then I tried to draw a Graphics object to a Render Texture. And all hell broke loose. I set about fixing the draw method to enable this as well and that's when I started to realize there was a core underlying problem through-out the Canvas Renderer and all Game Objects, as well as problems with texture binding in WebGL to cap it off.

Internally a Render Texture uses its own Canvas, or it's own WebGL Texture and Frame Buffer Object (FBO) under WebGL. The idea is that you tell the renderer "don't use the Game Canvas, use this one instead" and then you just call the standard `renderCanvas` or `renderWebGL` functions that every Game Object has. In theory, they shouldn't actually care about the destination and should just render themselves to whatever the current active context is. In practice, of course, that isn't what happened.

I set about fixing every Game Objects' 'renderCanvas' function. As I worked through this I saw the same thing I had been seeing previously and discussed last issue: masses of duplicated code, all doing the same thing in slightly different ways. I had already spent a good amount of time consolidating the WebGL side of things, now it was the Canvas renderer's turn. I created a common `batchSprite` method within the renderer that was flexible enough to handle virtually all types of Game Object. This has allowed me to shave off hundreds of lines of code. Text objects, for example, just make one single call to this function now. Where-as before they used 74 lines of code to render themselves. The same savings apply across the board.

With all objects using the same code path, it became possible for them to draw themselves to a Render Texture. The Render Texture just has to say to the renderer 'change the main canvas / frame buffer to me' and then anything that renders gets drawn to the Render Texture instead. When it's done, the renderer swaps back to using the main canvas / FBO again. This means that Render Textures now support the following Game Objects being drawn to them:

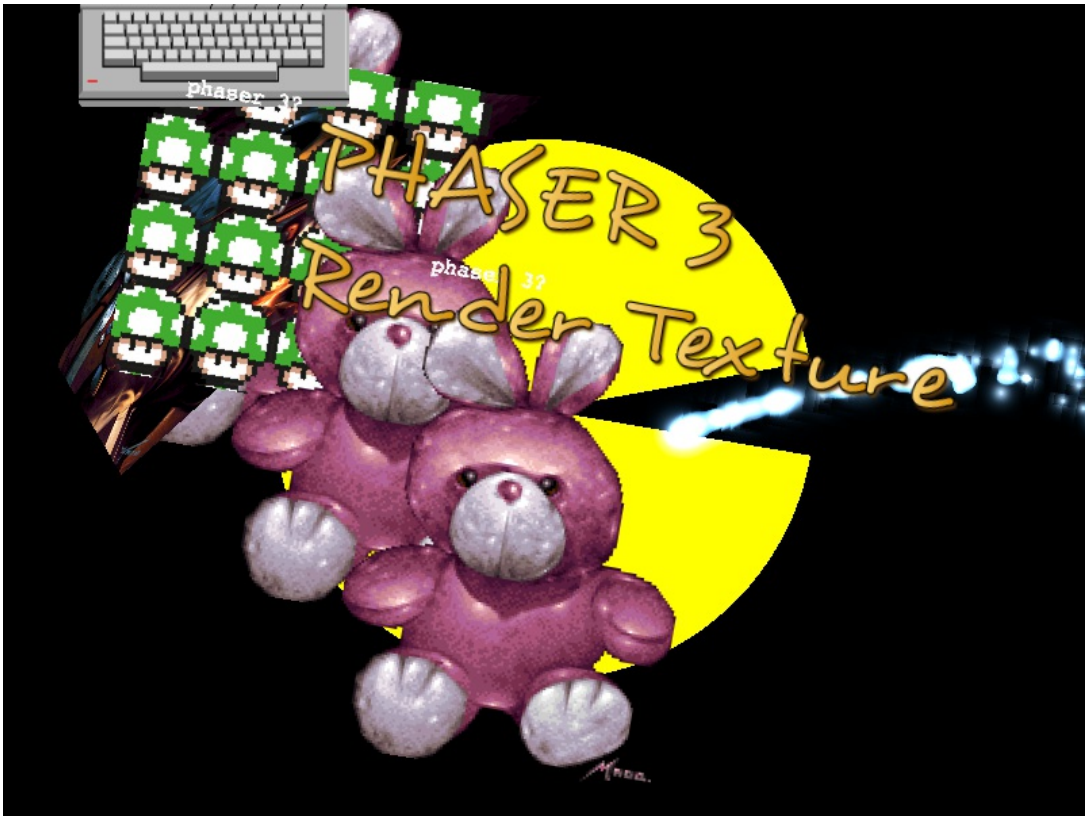
- Any renderable Game Object, such as a Sprite, Text, Graphics or TileSprite.
- Dynamic and Static Tilemap Layers.
- A Group. The contents of which will be iterated and drawn in turn.
- A Container. The contents of which will be

iterated fully, and drawn in turn.

- A Scene. Pass in `Scene.children` to draw the whole display list.
- Another Render Texture.
- A Texture Frame instance.
- A string. This is used to look-up a texture from the Texture Manager.

It can also take arrays, or nested arrays, of the above objects. They're iterated in turn so you have complete control over the order in which they're drawn. Previously, every single time you drew an item to a Render Texture it would cause a batch flush in WebGL. I also fixed this, so it only happens once, at the end.

When drawing a Game Object it will use whatever transform it has applied to it while drawing it. This gives you much more control than was possible before. Render Texture's also now have their own internal camera. You can move, zoom or rotate their own camera, just like you would any other and it'll impact anything being drawn to it. Allowing you to create a hot mess like the example below:



(this was just a test to see if I could draw nearly all types of Game Object to a Render Texture at the same time!)

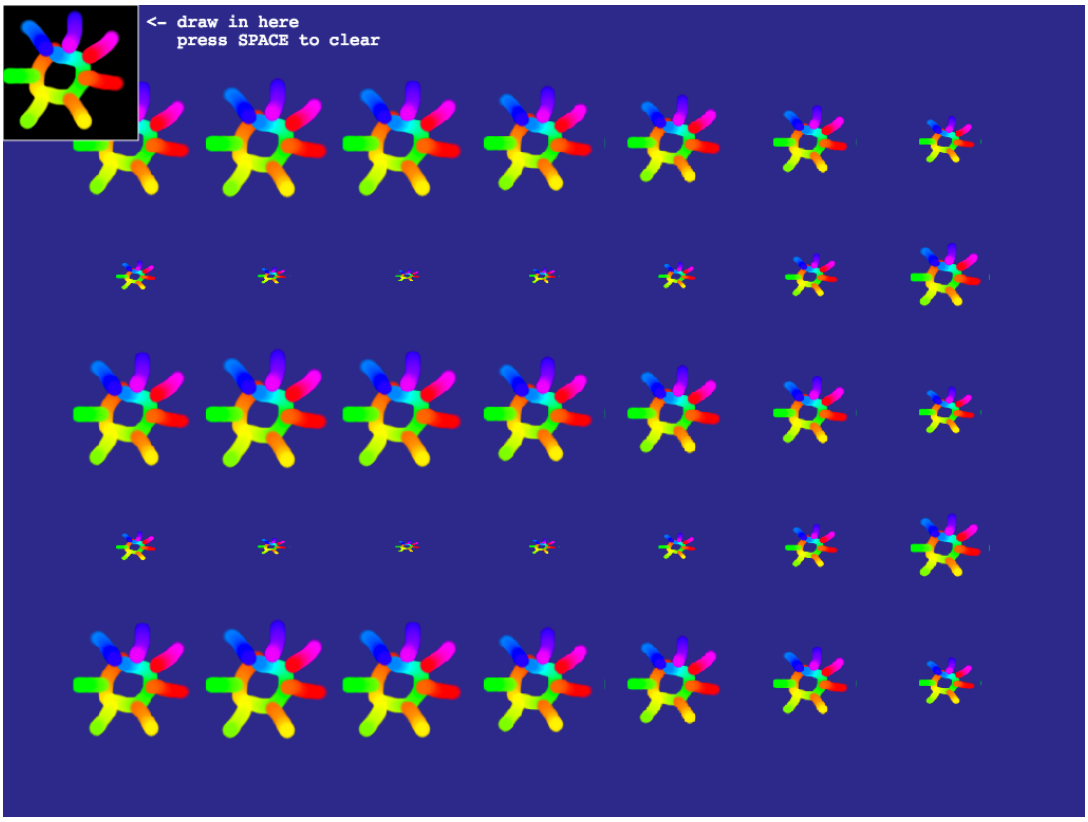
You can now also save a Render Texture to the Texture Manager. This means that all you have to do is use its key for any texture-based Game Object to be able to render with it:

```
//  Create a 128 x 128 Render Texture
var rt = this.add.renderTexture(0, 0, 128, 128);

//  Save it to the Texture Manager with the key 'doodle'
rt.saveTexture('doodle');

//  From now on, any texture based Game Object can use it:
this.add.image(x, y, 'doodle');
```

The best part is that as soon as you now draw something to the Render Texture, any Game Object using it will be instantly updated. Have a play with the example below to see:



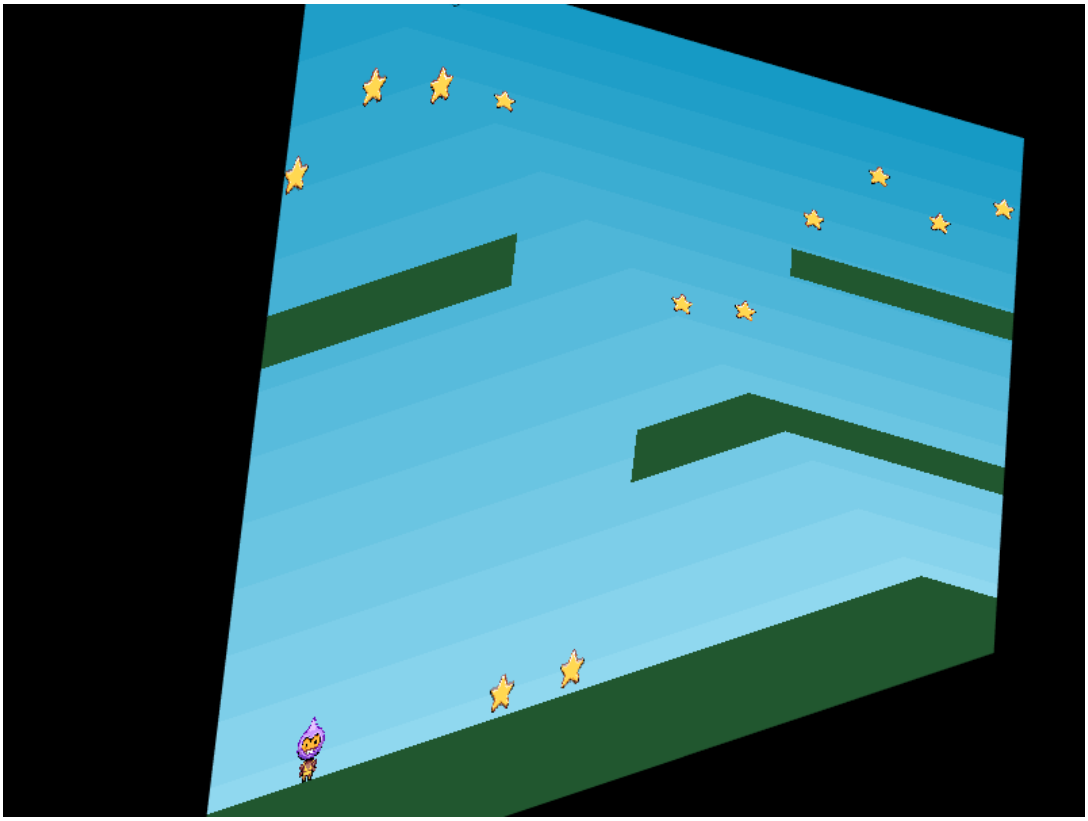
Doodle away

As you draw in the box in the top-left, the Render Texture is updated and that is immediately reflected in the scaling Group of Sprites on-screen, because each Sprite shares a reference to the same base texture. This opens up the possibility to create all kinds of cool effects, while minimising the amount of texture memory being used.

This was possible before by using a Canvas Texture, but with those every time you changed them you'd need to re-upload them to the GPU under WebGL, which is a costly operation. With Render Texture's there's no such impact, so doodle away to your heart's content!

It doesn't stop there though. You can actually draw an *entire Scene* to a Render Texture now. For example, how about when a level ends, you render how it looks to a Render Texture, then perform a neat disolve effect on it, or scale the texture off-screen to make for a fun transition? Or perhaps it's a game feature: maybe you pick-up an object that inverts the entire game for a short while. You can literally redirect the Scene to render to the texture instead and invert that until the effect wears off. This will work in both Canvas and WebGL equally. Have a play

of this game as it's drawn dynamically to a Render Texture being used by a morphing Quad:



Cursors to move and jump

Crop

Moving to a common set of functions for drawing Game Objects also allowed me to add in extra capabilities that weren't possible before. A few versions ago I added the support for cropping a texture frame. This allows you to specify a rectangular region within the frame that would be rendered. The rest would be ignored, leaving just the 'cropped' area displayed. It was an easy way to limit the visibility of an image without using a mask.

Up until now it only worked on Sprites and Images. Due to the internal changes made while fixing Render Textures, crop support now extends to Static Text, Tile Sprites and Render Texture objects too. In theory, because you can crop a Render Texture, you can now crop *any* Game Object. Also, remember how I said you can render a whole Scene to a Render Texture? If you combine that with crop you can create some pretty strange results like the this demo. Use the mouse to move the crop rect and the cursors to run and jump.

Quite neat, if a little weird :) Again, this works in both Canvas and WebGL. I'll leave it up to you to come up with some innovative uses for it!

Bug Hunt

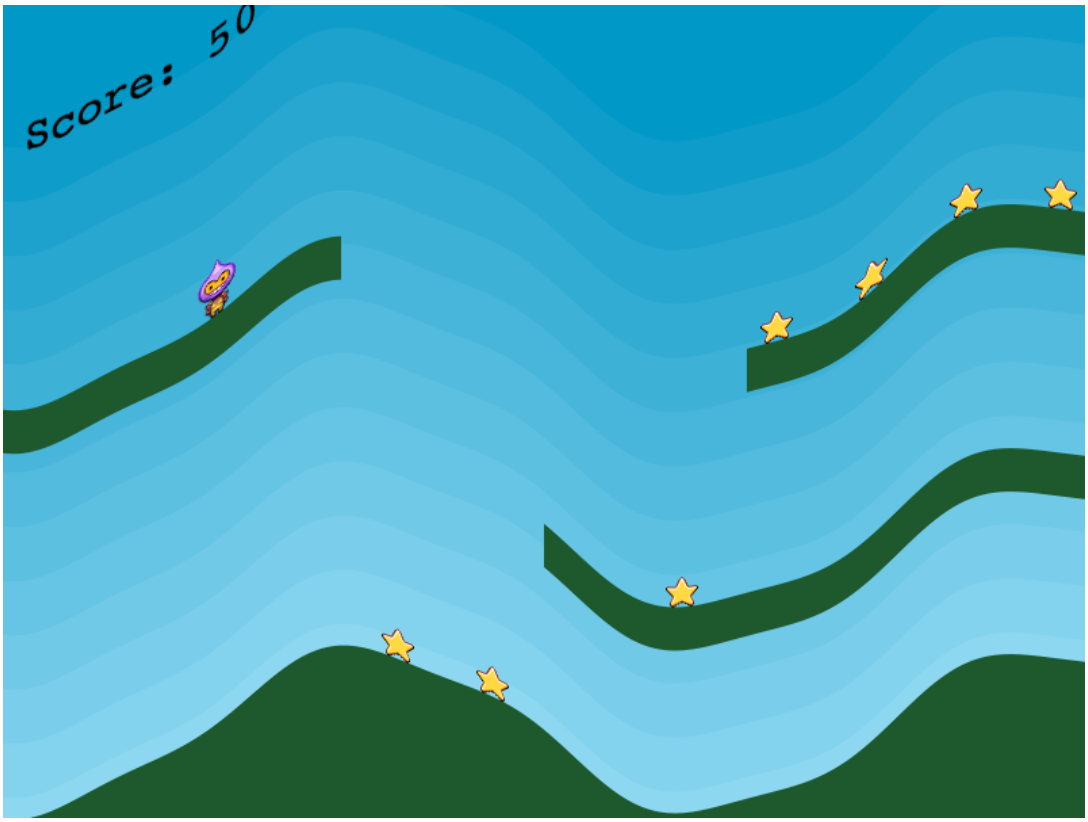
I also found a nice bug with blend modes and alpha values in the Canvas Renderer. A feature was added to the renderer so that when a blend mode was set, it would store the value in a local property. The idea being that if you had a bunch of Game Objects all using the same blend mode, it wouldn't keep setting the global context operation for all of them because it would have already been set by the first one. The problem, though, is that the cached value wasn't cleared if the context was restored to a previous state. When a context is saved and restored, the blend modes are reset too, meaning the cached value was now totally out of sync with the actual state of the context. The same was true for the alpha values.

After looking at it, it became apparent that it was causing more problems trying to maintain this cached state than it was just setting the value each time directly. This used to be an issue 5+ years ago, when you'd try your best to reduce Canvas operations to an absolute minimum, but browsers have advanced a lot since then and are far better internally. Now, the values are set, nothing tries to be cached and it removed a whole bunch of conditional checks and invalidation bugs as a result.

Camera Filters

With Render Textures working properly now, what the fixes really mean is that you can now redirect the rendering of pretty much anything, to any valid target. If you stop and think about this for a moment it should become apparent that this means it will be easy to add in a feature where any Camera in the game can render to its own texture. Or even have any Scene render to its own texture. And when you can do that, it means you can perform per-Camera special effects.

For months and months I have wanted to add support for Camera filters into v3. It would just be so cool to be able to have a scan-line filter, or a blur fliter, or a mirror filter running on a Camera. It was kind of possible in v2, although Cameras were a lot more limited then, but with the work I finished last week it was quite trivial to add into v3. There is an experimental version of this in the 3.12 Beta 2 release. There are some basic tests in the labs, such as the following sine wave shader running over a game (move the mouse and also use cursors to play):



You can also apply shaders to Render Textures, like the following VDU shader (again, cursors to move and mouse around too):

Mouse and cursors

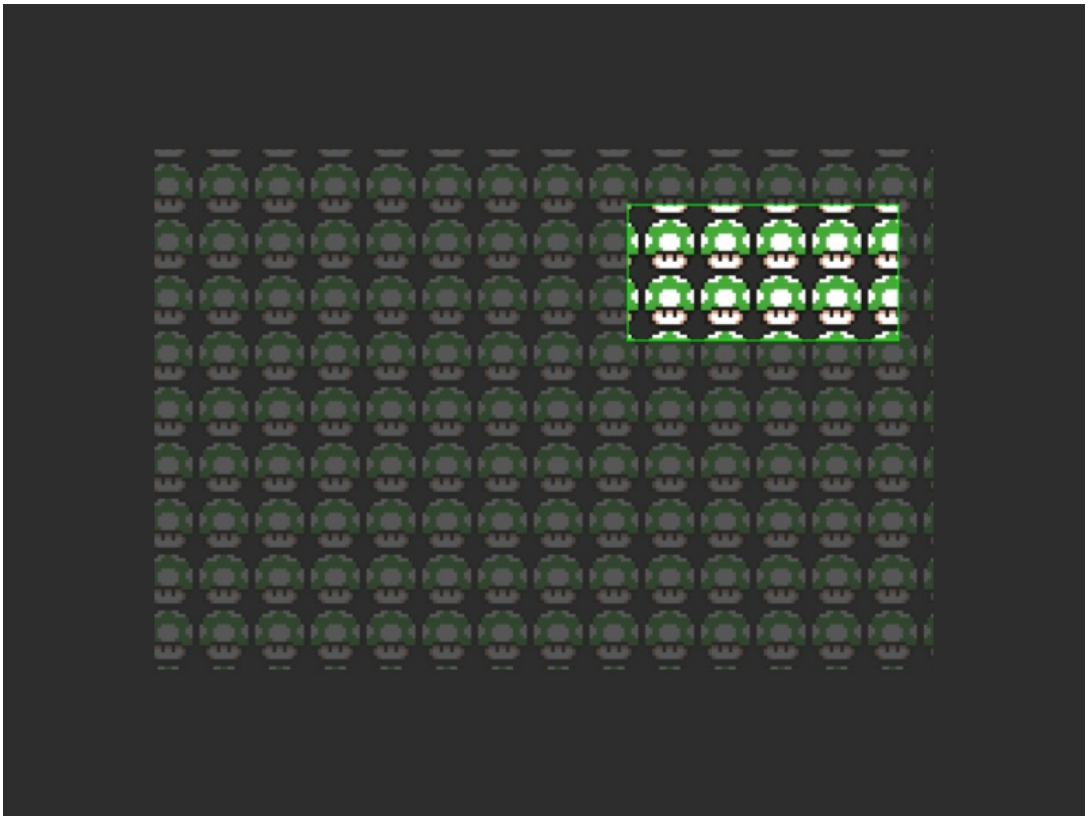
Before this lands fully I need to sort out making filter support far easier for you all to use.

In early versions of v3 we had what was known as an 'Effects Layer'. This was a layer which could have its own shader bound to it and then any object drawn to it would be impacted by that shader. It was sadly dropped after a renderer refactor a few months before release. Currently you have to create your own pipeline and have Game Objects use it, which is ok but a lot more complex than it ought to be. In v2 every Game Object had a `filter` property that could have an array of filters set on it. They were a lot more easy to set-up and use. The downside to them is that each filter operated in screen space, as fragment shaders do, so very often people would expect the filter to be localized to just the area of that Sprite, but it would actually impact the whole screen unless the shader had been specifically coded otherwise.

I've not yet decided what form filters will take in v3. It needs to be something akin to the Effects Layer we had previously but more flexible with it. On the plus side, the hard ground work has now been done, and I know that the renderer can easily cope with it once I move onto that. If you've any thoughts about how you'd like to see shaders surfaced in v3 then please let me know (you can email me, or open an issue on GitHub)

Tile Sprite Updates

As I unified all of the Canvas Rendering code I also changed Tile Sprites to work slightly differently. Previously, they created an internal canvas to which the texture they were meant to display was drawn. A fill pattern was then created from it. During rendering, it would use the `fillRect` canvas call to draw this fill pattern to the game canvas (or whatever the current canvas was). This worked fine but it meant that every single frame it would have to call `fillRect`, even if the contents hadn't changed at all. It also meant it wasn't possible to support things like cropping Tile Sprites.



Tilesprite cropping ahoy!

After moving to the new canvas renderer I changed Tile Sprites so they now draw to their own internal texture. This texture is then drawn, via `drawImage` to the game canvas each frame. A new fill only needs to take place if you scale or re-position the tiles (or change the frame being used) and for lots of cases this doesn't happen very often. Tile Sprites are frequently used for repeating backgrounds or level assets and don't actually change their internal make-up, so it was a waste of time recalculating it and using a fill rect every frame.

It's a small change but it makes the object more consistent and less heavy for most games. Plus it means you can now crop Tile Sprites :) which could be especially useful for progress / loader bars that use repeating patterns.

Phaser 3.12 Beta 2 Release

I'm pleased to say that Phaser 3.12 Beta 2 is now available to download from GitHub and npm under the **beta** tag.

This build contains everything in the Change Log as of August 9th. All of the demos and details explained in this Dev Log (and the previous one too) will require the 3.12 beta release to work.

Please download it and try it out on your projects. Hopefully you won't have to

change much, if anything, but it depends what features you are using. If you're using Render Textures, the API calls have changed, but otherwise it's mostly business as usual on an API surface level.

There will be another beta release, Beta 3, in a couple of weeks time. This release will contain the new Scale Manager and the Facebook Instant Games Plugin as well. At the time of writing that will be August 24th. It would have been sooner, but I'm on holiday the day this newsletter comes out, so this means we're looking at a 3.12 release during the last week of August.

It also means there won't be a newsletter for 2 weeks. As much as I love you all, it's called a holiday for a reason :) So I'll see you back here on August 28th for the next Dev Log, by which point 3.12 will be imminent and we can start discussing 3.13! Have a great summer everyone and see you in a few weeks.

These officially licensed quarter size arcade cabinets are things of pure beauty! Who wouldn't want one of these sat on their desk? :)

BASIC Engine is a very low-cost single-board computer with advanced 2D color and sound capabilities and it looks great!

What's new in Chrome 69 Beta? (answer: hell of a lot! including proper offscreen canvas support in web workers!)

Phaser Releases

Phaser 3.12.0 Beta 2 released August 9th 2018.
Phaser CE 2.11.0 released June 26th 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.