Web Version Unsubscribe

Welcome to Issue 131 of Phaser World

It's not even Halloween yet and we've already got zombies as the cover game :) It *is* a great game though! There's also a really nice platform making tool, some new UI components, lots of tutorials and videos. The Dev Log is quite long this issue, covering iOS bug hunting, a Matter.js tutorial and 3.15 progress updates. Dive on in and enjoy!

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.

Phaser Sticker Packs (Upgraded)

I'm pleased to announce that the Phaser sticker packs are now available to pre-order. In the pack, 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 glossy pixel art print.

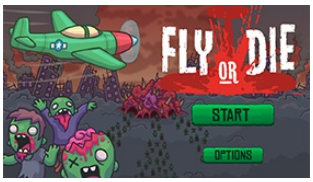
While talking to my printing company it turns out that I can upgrade the art print so it's double-sided for no extra cost. This means it's now on thicker 400gsm coated paper with the Phaser 3 art on one side and Phaser 2 art on the other - so you can choose which side to display. Everyone gets this double-sided print as standard now.

Click here to order your stickers.

Pre-orders run until the end of October. Please order early if you'd like them as

once the stock has run out, that's it. If you support Phaser on Patreon then check your personal messages as I have sent you all a custom discounted ordering link.

The Latest Games



Game of the Week

Fly or Die

The world has been overrun by zombies! Take to the skies in your constantly upgrading plane and blow them away, while trying to clear the map.

Staff Pick

Infinity Land

Run and jump your way through lots of community created levels, from easy to insane. When you've had enough, use the brilliant tools to build your own.

The Last Battery

Hold on to the battery for as long as possible! Whoever has it the longest, wins the round.

Fields and Flags

Very similar to minesweeper, except mistakes cost you money. How far can you get while keeping all your funds?

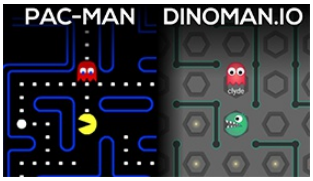
Neon Dunk

Shoot the neon basketball through a series of hoops and prevent it from slamming into the spikes on the screen.

What's New?

SunnyLands Wood Pack

A super-cute and free pack of pixel art for a woodland level platform game.



Coding Multiplayer Pacman in One Week

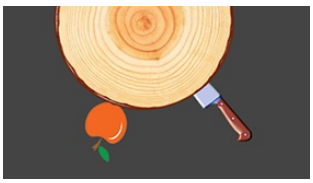
A video showing the creation of an IO version of Pacman in just a week.

Wheel of Fortune Tutorial Part 3

In the next part of the tutorial series learn how change the size of each slice of the wheel.

Wheel of Fortune Tutorial Part 4

In the final part of the series learn how to add icons to the wheel using Containers.



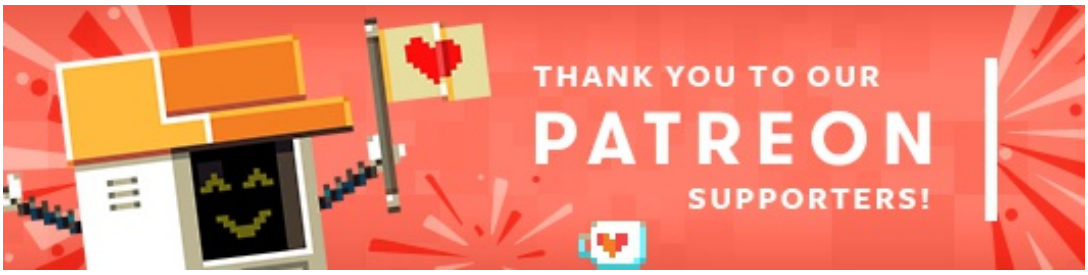
Knife Hit Tutorial Part 4

Learn how to add apples which you can slice apart in to your re-creation of the Knife Hit game in this part of the tutorial.

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:

Taylor Dragoo
Dan Thomas
Paul Schwarz

Also, thank you to **Vasileios Stodis** for their donation towards the project.

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 #131

Welcome to Dev Log 131. As we creep through October there are several development strands going on simultaneously. I'm working on both Spine integration and the new Scale Manager, as they will be released together in 3.15. So let's take a look at progress on each front.

Phaser 3 Doc Jam - Write Docs, Win Prizes

I announced the Phaser 3 Doc Jam two weeks ago and since then we have collectively completed the documentation for 1,518 items - and at the time of writing this newsletter, there are 531 new entries waiting for me to approve them! This is an incredible amount for such a short space of time. It means (after approval) there are just 1,466 items left before Phaser 3 has 100% API documentation coverage. Could you help get that number even closer to zero?

All you need to do to take part is point your browser to docjam.phaser.io. When the page loads it will randomly pick one of the descriptions that needs writing.

The source code is displayed and the description line highlighted. Take a look at it. Can you figure out what should be written? Maybe it's a property that needs describing? Or perhaps a return value from a method? Look at the surrounding code and see if you can infer the meaning of what's going on. There's a link to open the file on GitHub if you'd like a better look at it.

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.

Scale Manager Updates

The week started with me returning to the Phaser 2 Scale Manager code and re-evaluating it. I took the whole v2 code and ported it over to v3, hooking it into the game configuration and getting it running. It didn't take long but the further I got the more convoluted I realized it was. I finally wound-up with it working, as it does in v2, but it just didn't feel right. There are so many better ways to handling canvas scaling in browsers these days. The v2 Scale Manager is all about using a constant game-loop to check the canvas (or dom parent) size and then adjust the game accordingly. It'll listen for resize events but will also monitor parent bounds and resize the canvas as needed.

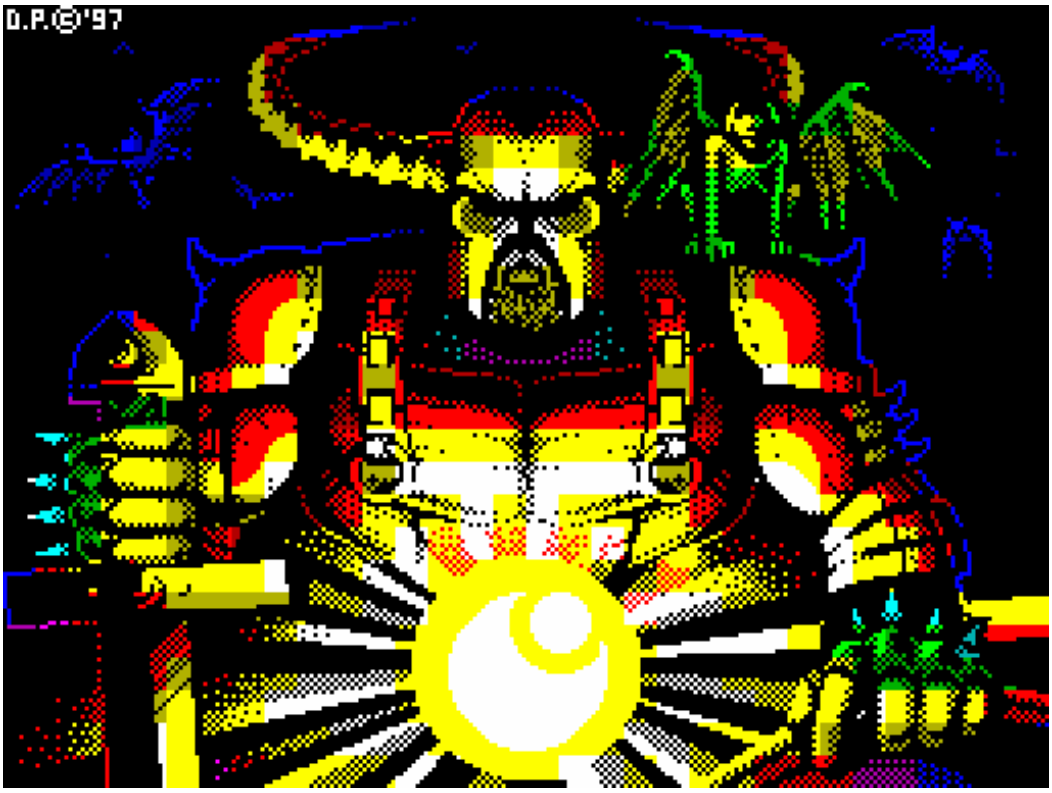
It works but it's a bit clunky, to say the least. I also struggled to wrap my head around what was going on with the USER_SCALE mode and some other internal settings, and *I was the one who coded them!*. I think we can all agree that it's a good sign something needs redoing, if when you look at it, even you don't understand why it does things the way it does. What's more, as I said browsers have evolved dramatically since v2 was created. So I spent the rest of my time on the Scale Manager removing every trace what came before and instead recoding it to run using newer scaling methods such as the 'contains' CSS property.

As it stands, I need to provide a fallback for browsers that don't support this (hello IE and Edge, I'm looking at you) but thankfully it's quite easy to test for. I also need to re-integrate the full-screen API support. Especially as this is going to become essential soon. Modern Android supports full-screen in Chrome and iOS now has it available too, but it's hidden under an Advanced Settings toggle (along with WebGL2) at the moment. This does, however, mean it's very likely to surface for everyone in the next version of iOS. When it does we'll no longer need any hacky ways to make things go 'full screen' in Safari, which, right now, is impossible to do properly.

One thing I did realize as I was working on the Scale Manager is that the Phaser

3 Labs uses the dev build of Phaser by default when you're editing an example instead of viewing it. This is a problem because my work on the Scale Manager rightly breaks lots of things internally, so if you 'edit' an example in the Labs it looks like they've broken too. I'll have to make sure edit mode in the Labs uses the latest stable build instead. Another task for the to-do list.

Either way, I'm very happy with the progress and hopefully, in the next Dev Log I can show-off some scaling demos.



Spine Updates

Progress on Spine integration advances in lock-step with the Scale Manager. I didn't expect them to be joined at all, to be honest. A week ago they felt entirely independent of each other. Yet after digging in, because of the way Spine handles the scaling of skeleton data it's actually really important that the Scale Manager is working alongside Spine.

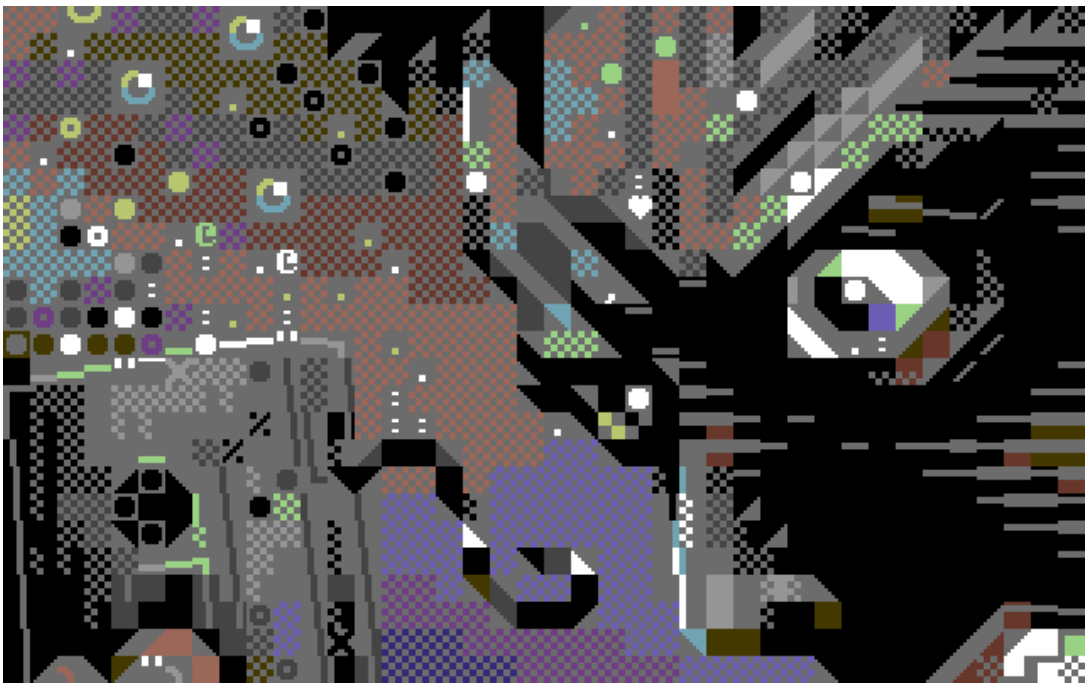
So far I have finished integration of both the Canvas and WebGL Spine Renderers and asset loading. I can now easily swap out skins on skeletons and position them within the game world. The next task to do this week is to create a new Game Object that encapsulates all of this. I'm not yet decided if you would literally create an instance of a new 'Spine' Game Object, or if you'd just create a Sprite like usual but apply a Spine skeleton to it. This option would be cleaner from an API perspective, but equally it opens-up a can of worms. For example,

input detection and physics bodies will need different handling for Spine than for a Sprite. Game Object components such as origin, crop and flip are redundant too, so I need to make sure they're not surfaced in the API at all.

All in all, based on the notes I've been making I think it's very likely that Spine support will be a stand-alone plugin, just like Camera 3D support is, that you choose to package into your build (or load at runtime), and the act of doing this adds in a new Game Object and set of helper functions to Phaser. After all, I can't have it enabled by default because the runtimes alone are several hundred KB in size. So it has to be something you opt-in to using, not out of.

A few people have asked me what features (from Spine) the Phaser Spine plugin will support. The answer is that it'll support *everything the official runtimes support*. I'm not coding my own version of their runtimes, or anything of the sort. I'm literally taking their runtime code and using it as-is, minus the asset loaders which are redundant for Phaser, and building an interface over the top between them and Phaser. This means that any feature they support officially, the plugin will support too. It also means that when they update the runtimes we can swap to the new versions without too much hassle, as it hopefully won't require any changes to the plugin.

Keep your eye on the Phaser 3 Labs for new Spine demos later this week, and I'll feature them in an upcoming Dev Log as soon as I can.



It's a Bug Hunt

On GitHub several people had reported that v3.14 performance on iOS was particularly bad. It took a lot of back and forth, but with the help of other community members, we finally narrowed down the cause to being Canvas backed Game Objects, such as Text, TileSprite or non-gl Render Textures.

It all started with this bug reported back in September. The issue was that random characters in an open type font just weren't rendering when the font size or content was changed. They'd suddenly appear black. This only happened in WebGL, so pointed at an issue with the way in which canvas backed textures were managed. I spent a lot of time working through the issue and concluded that if the underlying gl texture was deleted and then re-created, the bug went away.

It took a while for it to surface that this change, while it had fixed the bug, had caused an issue especially on mobile. The change was fine for things like TileSprites, because the fill texture is rarely modified. But, for Text objects, where the contents of the text can change every frame in some cases, it caused a real performance issue. Yes, the bug was solved, but at the expense of mobiles in particularly.

On Friday, with more iOS reports surfacing, I took the time to investigate this further. I talked to Ivan Popelyshev about it and showed him the samples. We concluded that deleting the gl texture and creating a new one each each time was the cause and also wasn't required, because you could recreate the gl texture from the canvas pixel data without the need to delete it. However, changing that back to how it was before made the original 'black texture' bug re-surface. Performance was restored, at the cost of un-fixing a different issue.

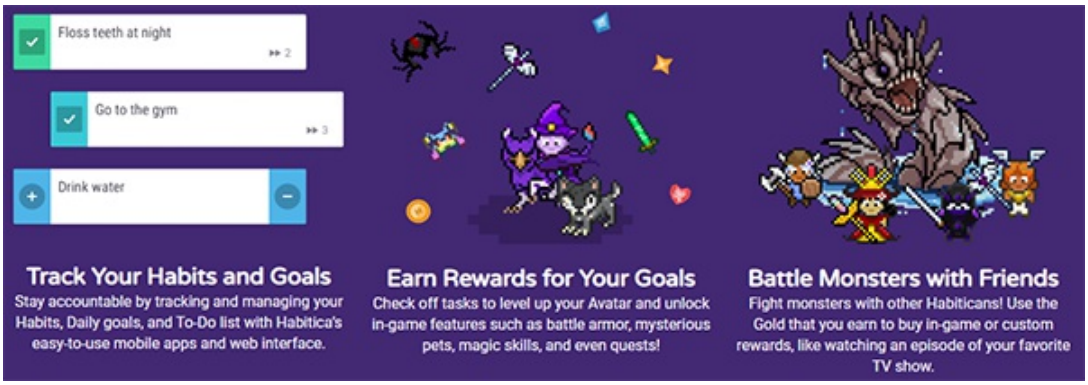
More investigation took place until we finally struck upon the root cause. Within Phaser, if a canvas backed texture has a power of two dimension, it is given the `gl.REPEAT` filter mode. If it's not a power-of-two, it's given the `gl.CLAMP_TO_EDGE` mode instead. TileSprites, in particular, take advantage of the REPEAT mode, so it was required for those. Text objects, however, don't need it. After much debugging it transpired that what was happening was that the Text objects internal canvas was being created with a `CLAMP` mode initially, but then due to a change either in the content of the Text object, or the font size, or similar factor, the Text object, quite by chance, now had an internal canvas size that was a power of two. When passed over to the WebGL Renderer it noticed this and swapped from the CLAMP filter mode to REPEAT. Doing this on an *existing* texture causes it to fail, rendering black in-game.

So there we have it. Hours of debugging and investigation, comparing texture settings and browser types and all manner of gl internals, and it came down to a case of the fact that you cannot change the filter mode of an already created gl texture. The fix was put in place, a simple boolean check, and voila. No more randomly vanishing textures. Because textures were just being updated instead of deleted and re-created, combined with a small modification of when textures were bound during the batch flush, and performance returned to the previously high levels. It's surprising how much mental effort edge-cases like this can consume. Thankfully, the day ended with a win. The performance was restored and the original bug remained solved.

This leads me to an interesting but related point. The days don't always end with wins like this. In fact, very often during development, you may hit upon an issue that genuinely stumps you. And no matter how many hours you put in, how late into the night you work, it remains unsolved. Perhaps it's something rooted in logic, i.e. your game just isn't doing what you expected it to do in certain situations. Perhaps it's struggling with understanding how Phaser does something, or using a feature incorrectly and not understanding why it causes problems elsewhere. Sometimes it doesn't even have to be becoming stuck on a problem, maybe you just feel like it's taking far, far longer to progress with your game than it should be. For every step you take forward, the game feels two steps further behind. These are just some of the common traits developers suffer from.

With World Mental Health Day last week, my social streams were full of more comments relating to this than usual. I consider myself fortunate that I do not suffer from mental or anxiety related issues. I know plenty of people who do and the impact it has on their lives, and those around them is dramatic. That doesn't mean I don't ever hit a point where I've just had enough. Stress, in its various guises, gets to us all, one way or another. And I'm no more immune to it than anyone else. Preventing it doing any real damage is the key and you can only do that if you recognize it's happening in the first place.

I'm not going to give any 'advice' here. It would feel disingenuous of me to even try. After all, everyone is different. I'll say what works for me in the hope that it resonates with a few of you. It is by no means a 'fix all' though. I will say that in all cases my stress with programming can be solved through two fundamental things: time and talking. I personally can tell when I'm at a breaking point. When things just aren't working, no matter how hard I try. Very often I just need to step away to solve it. Either from the computer entirely, or just to a different and more low-intensity task. I maintain to-do lists for all the various Phaser things going on. From tasks to do in the API, to planning the newsletter and project for the future. Some of these are big sweeping tasks, yet I always try to make lists of things I consider so easy they're almost trivial too. These things are usually not creatively rewarding to me, but need to be done anyway and don't tax my brain while doing them. A way to recharge the gray cells without feeling like I'm 'getting nothing done'.



A few friends use the site Habitica to manage their tasks. It plays like an RPG game, where you assign tasks and go on quests, level-up and spend gold on rewards. The more you put in, the more you'll get out. I personally don't use it but different people respond to different incentives and it may be what you're looking for.

Sometimes, though, even easy tasks aren't enough. In these cases, I have to step away completely and do something entirely different. I may read, watch a bit of anime, play some games, play with my kids, or just get an early nights sleep.

Anything that stops my mind whirring away. I appreciate I'm only able to do this because I work for myself. You can't usually just walk out of an office, for example, because things aren't going how you'd like. For my sake though, sometimes I just need to. And I feel justified in doing it because I'm extremely passionate about the work I do with Phaser. I may need to switch off sometimes, but it's never for long, and when I return I get an awful lot done while in hyperfocus.

The other thing I find that helps if an issue is taking too long is just talking to other developers. I don't have any around me in the office because I work on my own. So I turn to Slack and Twitter instead. What's important is that I rarely go there seeking an *answer* to a problem. I go there seeking inspiration. Other developers will nearly always have a different approach to mine. While it fundamentally may not be the solution I was after, it very often helps get me there by making me think about something I'd never considered before. It reminds me of a great quote by Andy J Pizza: "Every idea is just an iteration towards a better idea."

If you can, talk to other devs too. Ideally ones local to you, but online can work as well like it does for me. Just be cordial if you're talking to people you've never interacted with before. You may well be frustrated because you're stuck on something, but taking that frustration out socially, even if passive-aggressively, doesn't help anyone, least of all you. I see this quite often. People will roll-up on Phaser Slack or Discord stressed about something and kick-off, blaming the docs, or the examples, or some 'weird API choice'. Essentially trying to push the blame away from their lack of understanding to something else, when the reality is that the answers are *always* there to be found in the source. It just takes a little digging or knowledge to uncover them. And we'll happily help you do that, so long as you don't offend us in the process of asking. So please, do come and join us and chat about your games.

Accessing Matter.js Directly

The same question has come up a few times recently so I thought it warranted a mini-tutorial. In essence, the question being asked was how do you add something to a Matter.js world without using one of the Phaser objects.

Phaser provides a number of factory functions that encapsulate the creation of a Phaser Game Object, such as an Image, with an automatically generated Matter physics body. You may have seen functions such as `this.matter.add.image()` that provide this. Internally, it creates an Image, adds a special bunch of Matter components, gives it a body based on the image texture size and returns it. Which is great if that's what you need. Yet how about the times when you just

want to add bodies directly into the Matter world without binding them to any other object?

It's actually easier than you may think. You just have to know how to reference the Matter world and it's associated libs. For this example let's create a body directly in the Matter world, without using any factory functions at all. The key to this is knowing where the libs are. If you're using a runtime version of Phaser, by which I mean Phaser loads independently from your game code (like in the Phaser Examples), then you can reference all of the Matter libs via `Phaser.Physics.Matter.Matter`. Yes, that's duplicated at the end, because the first Matter is the namespace within Phaser.Physics, and the second is a reference to the entry point of the Matter library itself.

This means that anything you see listed in the Matter.js API docs, such as the Bodies, Constraint, Vertices, and so on, are available via `Phaser.Physics.Matter.Matter`.

Lots of the Matter lib functions require references to either the Matter Engine or the Matter World instances to be passed in to them. The Engine is available via `this.matter.world.engine` from within any Scene. The Matter world composite, which contains all of the simulated bodies and constraints, is available via `this.matter.world.localWorld`.

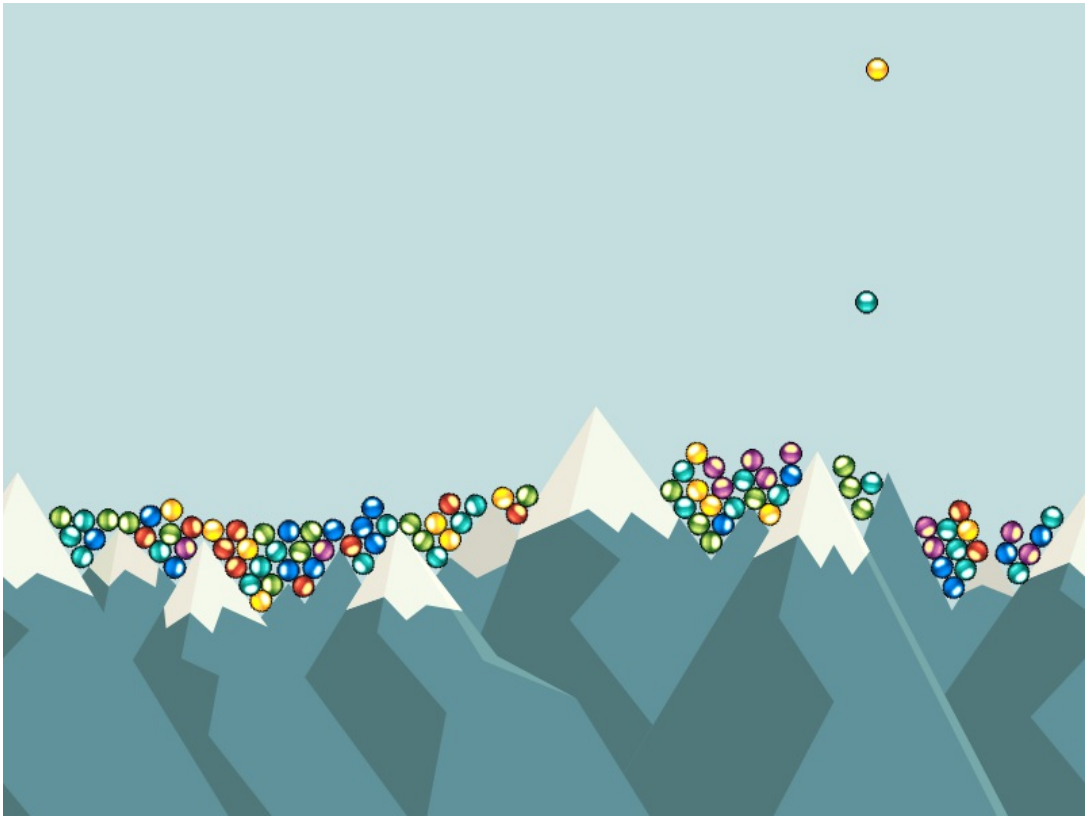
So, let's create a body using nothing more than a list of vertices and 'native' Matter functions. First, let's take some path data and pass it over to the handy `Vertices.fromPath` function:

```
var path = '0 307 0 67 8 55 12 53 57 128 86 94 128 136 148 103 190 159 210 135 222 149 248 109 267 133 293 93 321
128 361 75 381 97 439 4 523 117 551 78 563 92 569 93 603 38 637 99 654 53 701 154 729 109 750 140 800 66 800 307';

var verts = Phaser.Physics.Matter.Matter.Vertices.fromPath(path);
```

With a list of vertices prepared we can now create a Body. Again, Matter has a helper function called `fromVertices` available in the Bodies class. Then, we use `World.add` to add it to the simulation:

```
var body = Phaser.Physics.Matter.Matter.Bodies.fromVertices(408, 492, verts, { ignoreGravity: true }, true, 0.01,
10);

Phaser.Physics.Matter.Matter.World.add(this.matter.world.localWorld, body);
```

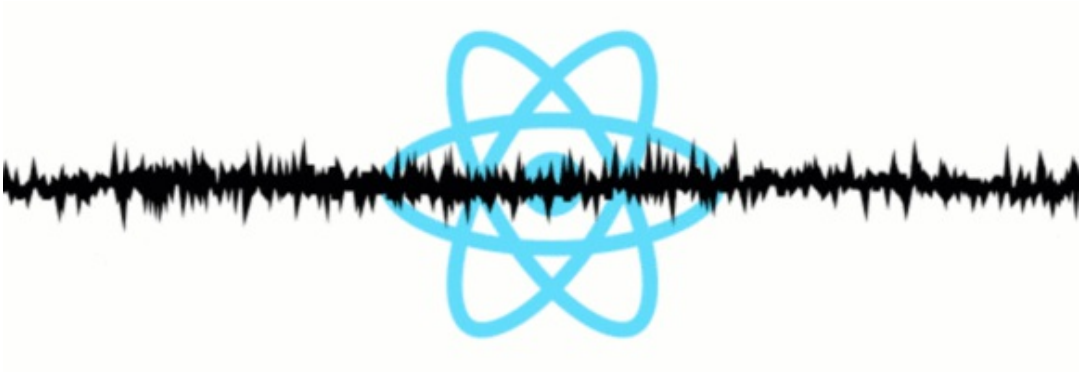
It's not much code but it's suddenly become a lot more verbose. If we run this example we can see our mountain range working and some balls bouncing off it (these are added via a timed loop):



Still, if you need direct access to Matter, this is how you can do it. For comparison here is the same code as above using Phaser factory functions:

```
var verts = this.matter.verts.fromPath(path);

this.matter.add.fromVertices(408, 492, verts, { ignoreGravity: true }, true, 0.01, 10);
```

This works because `matter.verts` is an alias for the Vertices helper functions, and `matter.add.fromVertices` is an alias for the Bodies fromVertices function, that also adds the resulting body into the world. I designed Phaser to save you time by needing less coding, and this is one of the ways I do that. Just remember that if you ever need to get right into the internals of Matter, you can do it, it's all there available to you, so long as you know how to get in.



Audio Visualisations with Web Audio and React (although it's just a plain canvas, so the code in the article is trivial to add to your Phaser game)

Goodbye JavaScript, Hello Web Assembly is a really interesting read.

A deep dive into 'this' in JavaScript: why it's critical to writing good code (well worth reading, especially if you're new to JS)

Phaser Releases

Phaser 3.14.0 released October 1st 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.