Welcome to Issue 113 of Phaser World

You know that feeling when you're so deeply in a project that you struggle to even tell where the line between real-life and work lays? We're there. Right now!

Phaser 3 is imminent and it's taking its toll (see the Dev Log for details). On the plus side, there are some amazing games and tutorials this issue :) I really had fun coding in Golden Quest and there's a new release of Phaser CE too - woo!

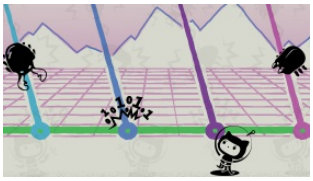
So, until the next issue, keep on coding. Drop me a line if you've got any news you'd like featured by simply replying to this email, messaging me on slack or Twitter.

The Latest Games

Game of the Week

Golden Quest

Take to the high seas, visit islands and code your robot to dig for buried treasure - but watch out, you're not alone!



Staff Pick

Commit4

The bug invaders are coming to destroy master branch. Your mission is to prevent bugs conquering the production.

BADGUN

Take to the road, weaving in and out of traffic, as you collect money to pay off your speeding fines in this retro inspired arcade title.

Brick Break

Clear the levels by shooting colored bricks in from the sides in this great addictive puzzler.

Truck Reign

Drive your monster struck across the levels, collecting steering wheels as you go in this challenging physics game.

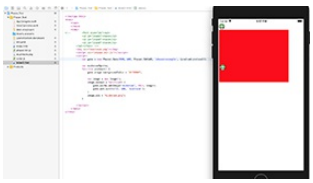
What's New?

Official Phaser T-Shirts

A new range of Phaser T-Shirts are now available from Amazon in the US. Available in 5 different colors with male and female fits, look the part at your next game jam or meet-up!

Phaser CE v2.10.0 Released

The first release of Phaser CE in 2018 is a great one, with new features, updates and fixes across the API.



Phaser to iOS without PhoneGap or Cordova

This Gamasutra featured tutorial covers how to publish a Phaser game to iOS without using the most common wrappers.

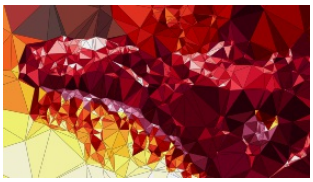


Image Vectorizer

Converts an image into a series of vector triangles and exports as SVG or PNG, including source code.

Making DOOM 3D in Phaser - Part 1

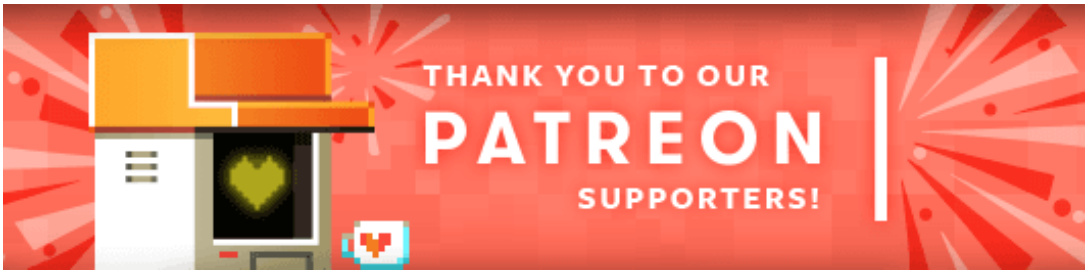
WiLD carries on re-creating the classic game Doom in Phaser, this time looking at setting up the Game State.

Stack the Crates Tutorial

The Tipsy Tower prototype evolves into a full game, with orientation and responsive support, including source code.

Dispatching Events with Signals Tutorial

Learn how to create and use Phaser Signals in your own games.



Thank you to our awesome new Phaser Patrons **Jeff Damon** and **Brad Harms**.

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 #113

Hold onto yer pants, this is a huge Dev Log packed full of demos and cool updates! If you've been peeking at our commit log you'll notice that things have been insane recently. Literally hundreds of commits, loads of changes and things getting finished off all over the place as we gear up for release. It has been intense, to say the least. I'm literally living and breathing Phaser 3 non-stop right now. It's at the point where I'm even dreaming about coding on it, to the point where I have to sanity check the commit log in the morning to work out what I actually did, and what I dreamt :)

So let's dig in. I simply cannot feature all the changes we have put in this week, but here are some highlights. As usual, there is a new beta: Phaser 3.0 Beta 19 which is now on npm and GitHub. All examples this issue were built against it. The master branch is changing hourly, so if you pull from master please understand the carpet might literally be pulled out from under your feet at a moments notice. Hold it together as we march towards 3.0! First up, what's new in the land of audio?

Sound Manager Updates

Pavle writes: Last week I made a lot of progress with HTML5 Audio implementation of the Sound API. I spent a lot of time testing on mobile devices and came across the good old issue of mobile devices preventing audio from loading and playing without previous explicit user interaction.

What this means is that devices simply ignore any attempt to load or play element before user taps on the screen.

In order to make your life easier when dealing with locked audio on mobile devices, I've added two very useful features to the sound manager. First one is the SoundManager#locked property that lets you query if audio is locked on a device or not. Second one is SoundManager#unlocked event that gets emitted when audio gets unlocked.

Together they enable you to do something like this:

```
if (this.sound.locked) {

  this.sound.once('unlocked', function (soundManager) {

    // Do stuff

  }, this);

}
else {
  // Do the same stuff
}
```

One side effect of this restriction is that on mobile devices HTML5 Audio Sound objects will initially have its duration properties set to 0 since we cannot read the duration of a sound that has not yet loaded.

I updated a few examples to make use of these newly added features, give them a go on your mobile devices:

In most cases you won't need to do anything different than usual since all actions performed on sound objects, such as calling a method or assigning a new value to any of the properties, will be queued and as soon as audio gets unlocked they will be performed immediately in chronological order.

This means that if you start playing background music as soon as game loads, on desktop it will work as expected, but on mobile devices user will need to tap on the screen before music can start playing automatically.

You can check updated Basic Playback and Events example, and AudioSprite example on your mobile device to see that they work just the same, except for the initial locking, without any changes to the example code:

When it comes to Web Audio API thing are a bit better since only iOS devices impose this restriction and as we load audio files using XHR, only audio playback is locked until explicit user interaction, so we can initialize sound objects' duration properties to correct values.

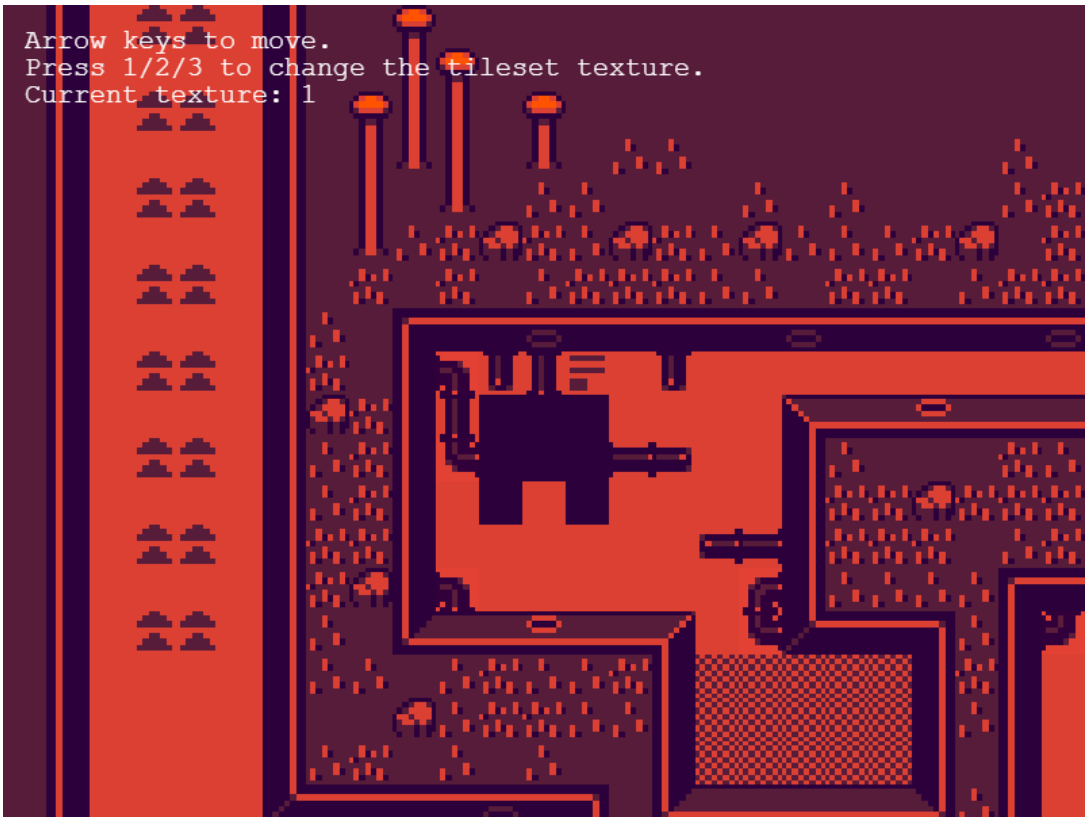
I've posted an article where you can read more about how Web Audio API locking is handled in v3.

Tilemap and Physics

Here's Michael on what's new with the Tilemap API:

Arcade, Impact & Matter Physis are all now wired-up to the Tilemap API. If you are itching to play with physics and are pulling directly from the master branch, keep in mind that the API will be in flux until the v3 release. With that warning out of the way, on to the update.

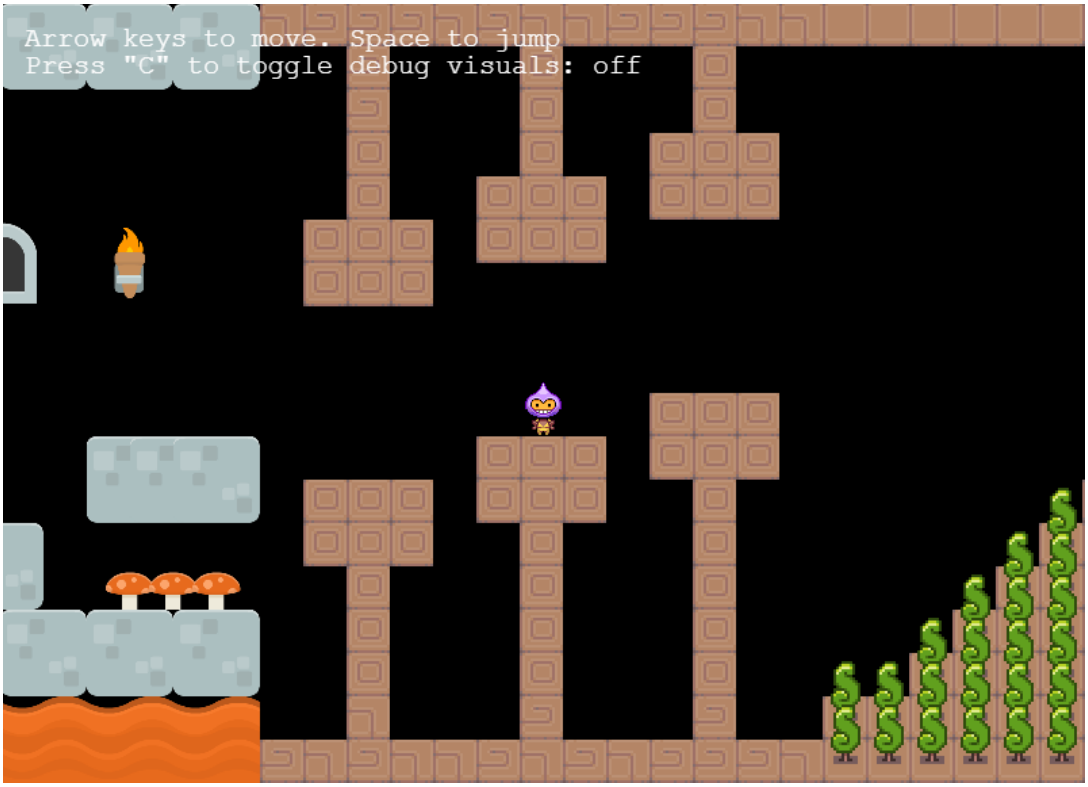
First up, a non-physics update. Tilesets are now treated as the single source of truth for which texture to use when rendering tiles, making it easy to do things like dynamically change the look of all tiles in a map at once. Note: this only really makes sense to do if the textures are variations on the same tiles, e.g. in this case, the same tiles in different color palettes:



Click on the screenshot to dynamically swap between red, green and blue tile color palettes.

Also, this.load.tilemapJSON(…) has become this.load.tilemapTiledJSON(…) to make room for the Weltmeister JSON tilemap format (more on that below).

Time for some physics updates. Arcade physics has been upgraded this week to handle collisions when a map has layers with different tile sizes:



Click on the screenshot to hop around a triptych world with different tile sizes.

Impact physics and the Weltmeister level editor format are now integrated with the Tilemap API. The editor has pretty nice features, like the ability to define a collision map where you can assign slopes to tiles. If you used Ninja physics in v2, you are probably familiar with that concept.

Loading a Weltmeister map is almost exactly the same as loading a JSON Tiled map:

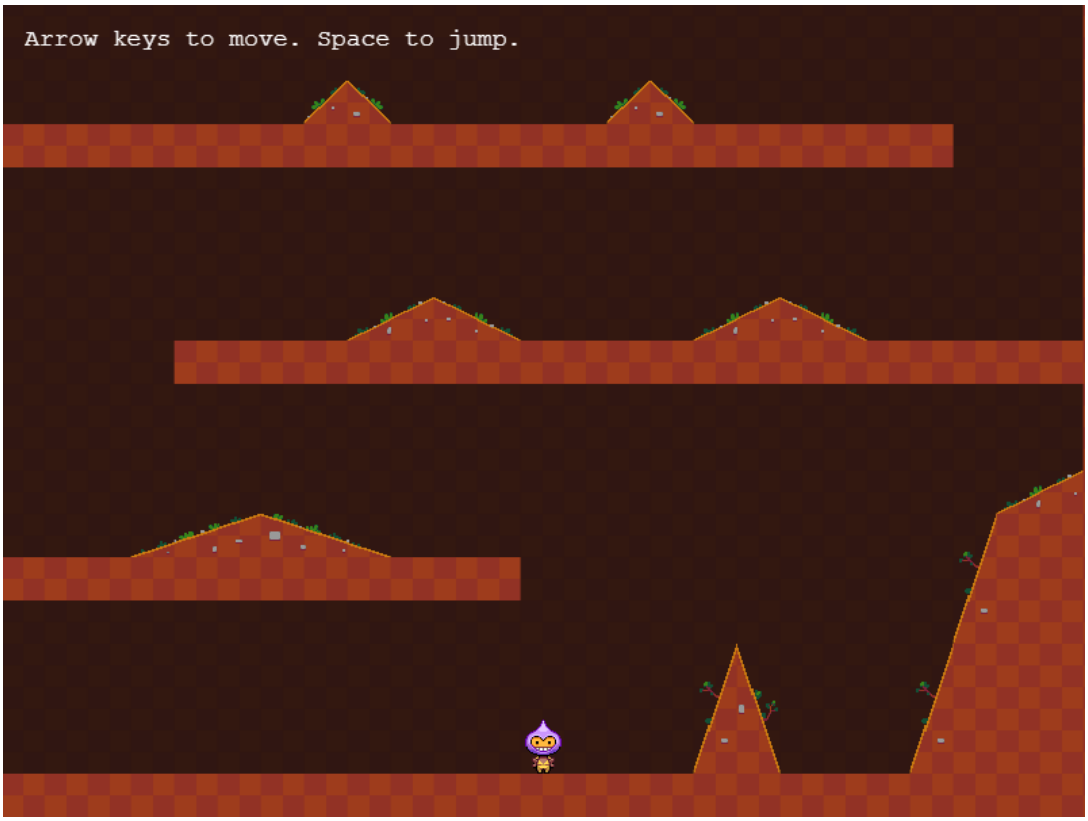
```
function preload ()
{
  // A standard Weltmeister map with two layers: "map" & "collision"
  this.load.tilemapWeltmeister('map', 'assets/tilemaps/maps/impact3.json');
}

function create ()
{
  var map = this.make.tilemap({ key: 'map' });

  // Name of tileset from Weltmeister map, name of image in Phaser cache
  var tileset = map.addTilesetImage('media/tiles.png', 'tiles');

  // Name of layer from Weltmeister, tileset, x, y
  var layer = map.createStaticLayer('map', tileset, 0, 0);

  // This will pull in the "collision" layer from the associated map
  this.impact.world.setCollisionMap('map');
}
```

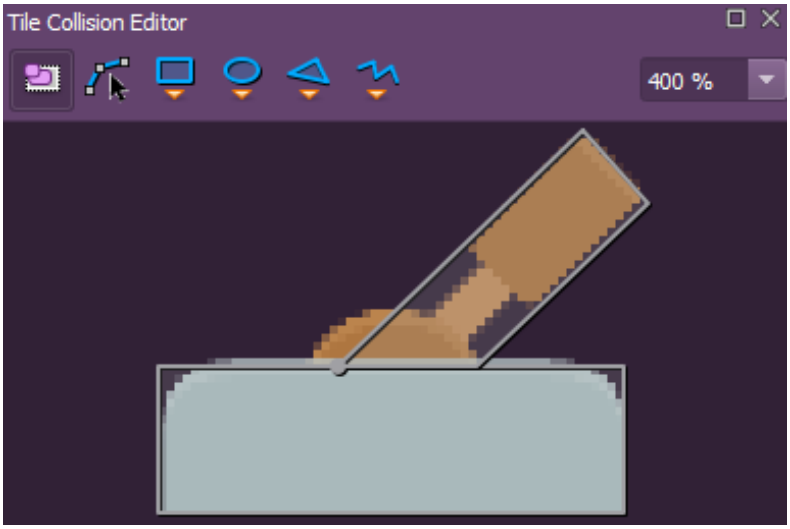


Click on the screenshot to jump & slide around a simple Weltmeister level.

And if you don't have Weltmeister, no problem – you can still take advantage of Impact physics. this.impact.setCollisionMap(…) also accepts a 2D array, so you can generate your map by looping over your tiles and building an array. Check out the corresponding example. One important note: the Impact collision map is not updated automatically when you modify tiles with the Tilemap API, so if you remove a colliding tile, this.impact.collisionMap needs to be updated.

And last but not least, Matter! Like with p2 & Ninja in v2, a tilemap layer needs to be "converted" so that physics bodies can be created for the colliding tiles. There are a number of ways to do this that are built into v3 (more on that next week), but for now, I want to demo the Tiled integration.

The tileset collision editor in Tiled allows you to visually map out which areas of a tile should collide by drawing basic shapes. Rectangles, polygons, and ellipses (which are treated as circles) are supported within Phasers Matter.



Here I used the collision editor to map out this lever tile with a rectangle and a polygon.

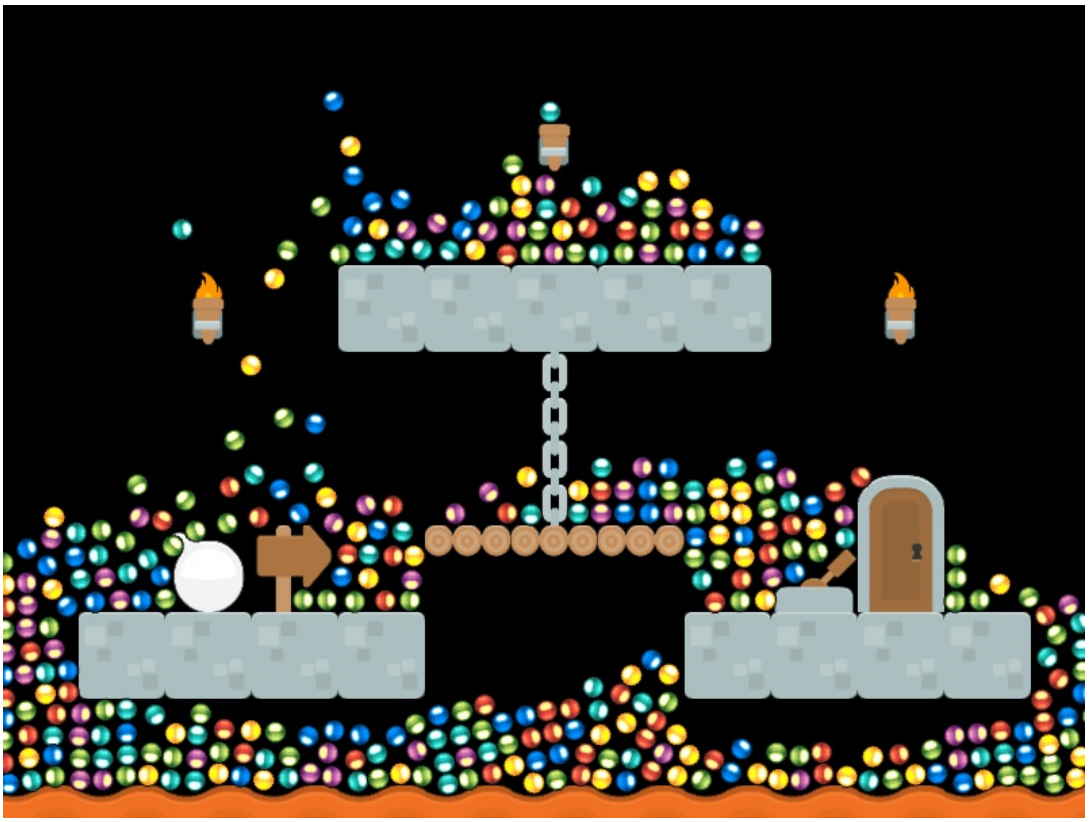
Once you've got those shapes defined, you can load them into Matter bodies:

```
var layer = map.createDynamicLayer(0, tileset, 0, 0);

// Mark the appropriate tiles as colliding
layer.setCollisionBetween(5, 12);

// Convert the layer. Any colliding tiles will be given a Matter body.
// If a tile has collision shapes from Tiled, these will be loaded. If
// not, a default rectangle body will be used.
// The body will be accessible via tile.physics.matterBody.
this.matter.world.convertTilemapLayer(layer);
```

And then have some fun:

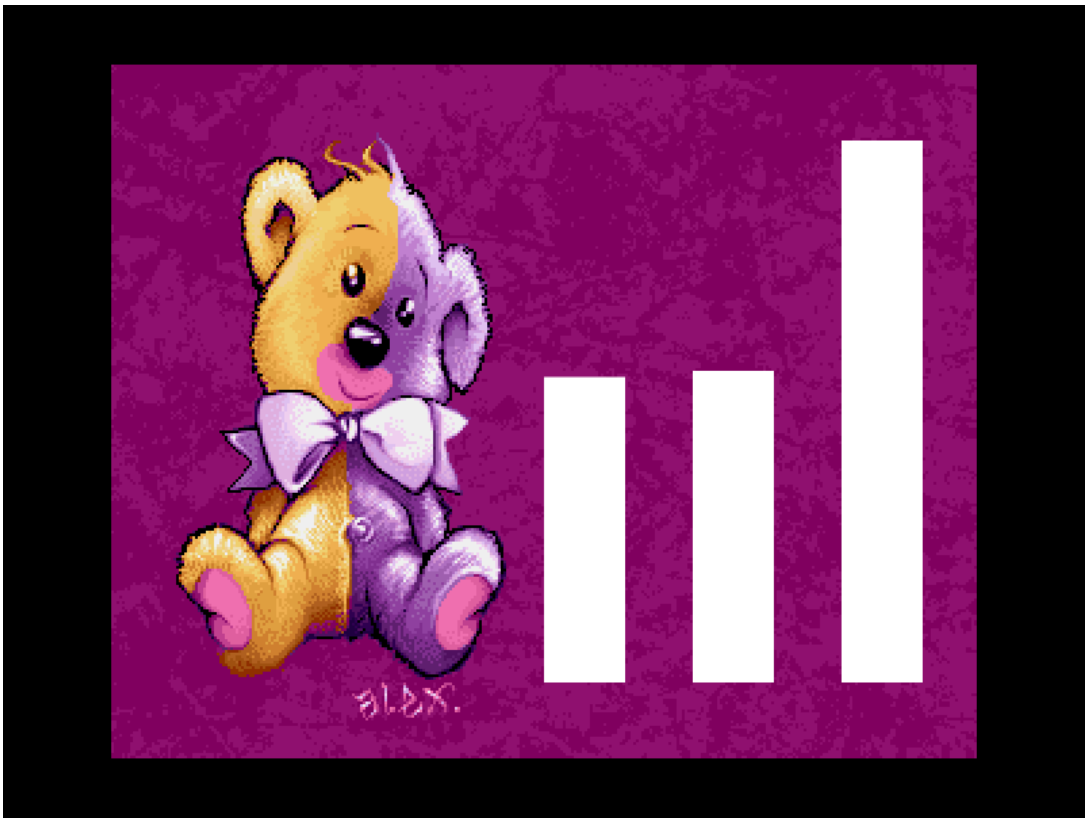


Click on the screenshot to drop some balls around a map.

More on this next week!

"Remember: short, controlled bursts."

I updated huge parts of the API last week. This wasn't a case of just new features being added, but rather putting all the final pieces into place. The first to get worked on was the Loader. After fixing a few issues with older browsers I exposed the progress events up. You can now listen for progress events from the Loader itself, or from individual files. If a file is loading via XHR (which all of them will in modern browsers) then you get byte-by-byte progress events *per file*. This is really handy if your game contains a number of large files, like large texture atlases or audio files. In Phaser 2 the progress event would only update per file, but you now have the option of displaying the load progress of the big files too. Here's a demo showing 3 loading bars, one per audio track:



Because of the way Scenes are handled in the game loop now you can render a progress bar even as images in the same scene are loading. The following code shows one really quick way of using a graphic for a progress bar as several hundred images load in!

```
function preload ()
{
    var progress = this.add.graphics();

    this.load.on('progress', function (value) {

        progress.clear();
        progress.fillStyle(0xffffff, 1);
        progress.fillRect(0, 270, 800 * value, 60);

    });

    this.load.on('complete', function () {

        progress.destroy();

    });

    //  Now let's load a huge stack of files!
}
```

Another task on the Loader todo list that was ticked-off is the ability for you to specify local JSON objects in the Loader. So for any loader call that can use a

json file, such as loading a tilemap or texture atlas, instead of giving the path to the json you can give it a JS Object instead. This is really handy if the json has already been loaded, is generated dynamically, or you want to bundle all of your json data together into a single file. You can see how it works by viewing the source of this example.

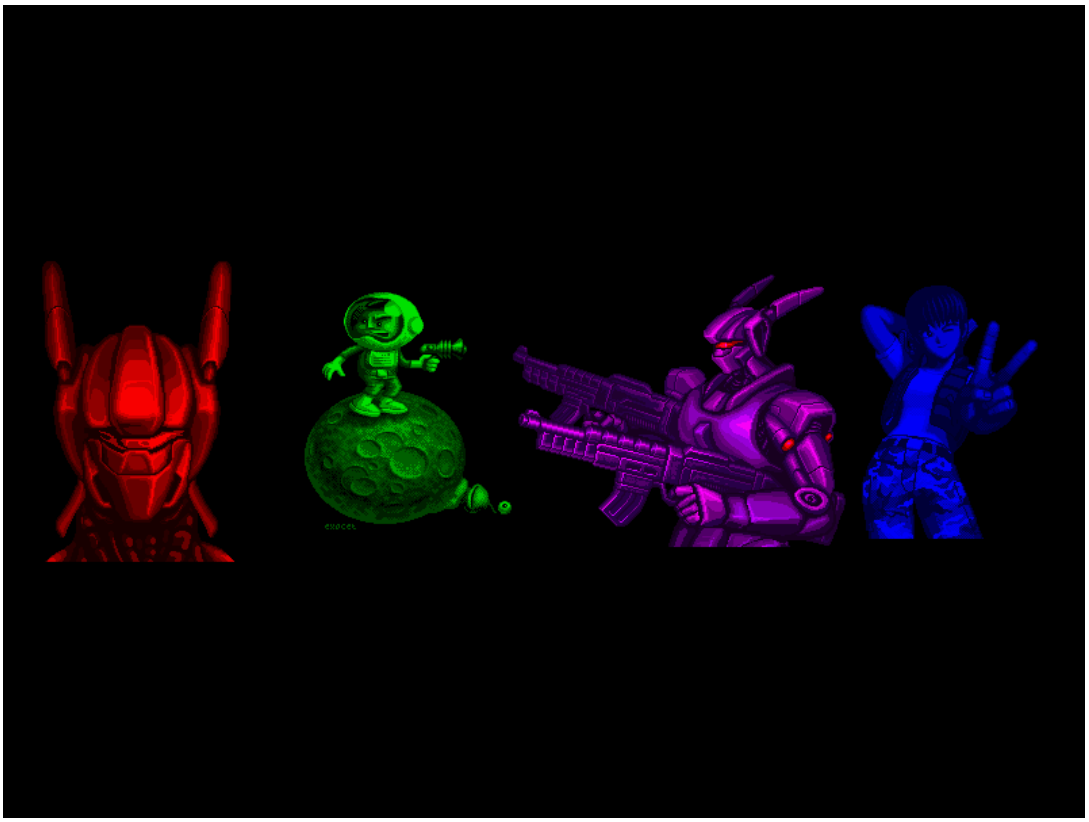
Another thing you can now do is get access to any **meta data** that was stored in a Texture Atlas json file. When the atlas parser inspects the json it will look for the frames objects and extract all the texture frames from it. But now it will also copy any extra data it finds too, which you can easily get to via the Texture Manager. This is really handy if you want to store build data in your json, compare versions, extract RGBA data and so on.

In a similar vein, you can also now store your own custom data per frame in your texture atlas json. For example, have a look at the following JSON and the code that reads from it:

```
{
    "filename": "contra2",
    "frame": {"x":2,"y":316,"w":142,"h":222},
    "rotated": false,
    "trimmed": false,
    "spriteSourceSize": {"x":0,"y":0,"w":142,"h":222},
    "sourceSize": {"w":142,"h":222},
    "pivot": {"x":0.5,"y":0.5},
    "tint": 0×ff0000
}

var img = this.add.image(x, y, 'megaset', 'contra2');

img.setTint(img.frame.customData.tint);
```



View the source to see how this one works

This opens up all kinds of opportunities for editors and build tools.

As well as events and custom data, the Loader is now fully modular. Every File Type that the Loader can handle is its own separate entity, that registers itself with the Loader. Don't need to load in any Tilemaps? Then you can exclude the tilemap file types in your own custom build. Equally, the opposite is true - want to extend the loader? You can just create a new File Type for it and it'll automatically register itself and expose it's via the Loader in your game, all without changing the actual Loader source code at all. You'll see this pattern used a lot through-out V3, especially in the next part ...

Plugging into Plugins

Right from the start, I was always adamant that V3 would be as modular as possible. I didn't want systems deeply referencing other systems, and I wanted you to be able to pick which systems were even active in your Scenes, as well as have the ability to create your own. As part of the tidy-up I completed this work last week and it's now in Beta 19.

Every internal system is now its own plugin. They register themselves automatically with a global Plugin Manager and it's fully under your control how they get exposed, if at all. In Phaser 2 a State would have a huge bunch of

properties injected into it, all of the short-cuts for things like 'this.physics' and 'this.sound'. They were references to the global systems running within Phaser. I wanted to keep this for Phaser 3 because honestly, it makes your life as a dev so much easier, but I also wanted it to be under your control! This work has now been completed.

You are now able to control exactly which plugins are running in each Scene. Every Scene has a small set of 'core' plugins that it must have, but beyond this, it is up to you. If you know you are not using tweens anywhere in a Scene, for example, then you can completely exclude the Tween Manager. It won't register itself in the Scene, so there are no wasted update loops running that don't need to be there.

As with the Loader File Types this opens up the ability for you to make your own plugins. In fact I even created a GitHub repo specifically for this, providing a base template to work from, especially now the Loader has the new 'load.plugin' ability too.

And remember those properties I mentioned that got auto-injected into the States in V2? You can define those yourself now as well. There is a special Injection Map which literally lets you map Scene systems to the properties they will be available as (if at all!) - perhaps you don't natively speak English and would like to map properties like 'sound', 'physics', 'input' etc to your own language? You can now do this by setting one configuration object. Or if you'd rather keep it all clean, you can hide everything, so nothing is injected at all! Keeping the Scene pristine.

Scene Manager Updates

As part of the process of finishing off the plugin system I also updated the Scene Manager a lot. It had all the right features originally, it just needed tidying up in places, so that's what I did. More importantly, though, I changed the way the Scenes flow through the game loop.

Scenes will now update flowing from top to bottom. So if you have a stack of 3 scenes, maybe a background scene, a game scene and a UI scene on the top, it will process them from the top down, so the UI scene gets updated first. To go with this the Input Manager was updated allowing you to literally stop the flow of input at any point. So if you've got a UI Scene on the top and you click a button on it, even if you've got an input enabled sprite right below that button in the game scene, it will won't receive that input event. It will be properly halted at the UI level. Naturally, this is under your control - if you'd rather events did propagate between scenes then you can enable that with a single call to the Input Manager.

When the Scenes render though, they render from bottom to top (or front to back.) which is exactly the way you'd expect them to appear visually. Of course, you can hide a scene to stop it rendering at all, send scenes to sleep, reposition them, send them to the back and so on. Much like a normal display list in fact. The more I use it, the more I get excited about just what's possible with this new set-up and I long for the chance to make some killer demos to showcase this all!

Docs and Examples

With most of the internal updates finished I'm now going through the codebase adding in the jsdoc blocks. I hope that by the middle of this week we'll have jsdoc blocks in for every property, method and class in the API. They will still need filling with descriptions, but at least all the API signatures will be visible and we can try generating some TypeScript defs too.

While I'm doing this, Antriel is plowing through the V3 examples. We're weeding out all the old and broken ones, taking screen grabs, creating new ones and generally making sure everything is in place. Because they're going to be the primary means of learning about V3 at release, as it'll take a little while for tutorials and videos to emerge.

V3 release is getting *really close* now though. We've just one Dev Log left before 3.0 ships. That is quite frightening, to be honest. I'm excited because I want this out and to see people playing with it, but it's such a huge update and follows on from such a well-loved and established predecessor that it will undoubtedly rock the apple cart. We've literally a week left, so please beam as much mental support and wishes as you can our way, because it's getting crazy around here!

Phaser 3 Labs

Phaser 3 Beta 19 is out and ready for testing.

Visit the Phaser 3 Labs to view the API structure in depth, read the FAQ, previous Developer Logs and contribution guides.

You can also join the Phaser 3 Google Group or post to the Phaser 3 Forum - we'd love to hear from you!

Sam's Journey is a new game for the Commodore 64 / Windows / Linux and it's absolutely beautiful! A really stunning platformer with great games and animation.

BitPotion by Joeb Rogers is a beautiful little font, and I mean little! Just 4x7 pixels for lower-case. A range for formats and styles are available.

Finally, the secret ska history of that weird levitating man emoji!

Phaser Releases

Phaser CE 2.10.0 released January 18th 2018.
Phaser 3 Beta 19 released January 22nd 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.