Web Version Unsubscribe

Welcome to Issue 138 of Phaser World

This week a beautiful pixelart platformer graces the cover and an equally great pixelart tower defense game is the Staff Pick! They both offer very different styles of gameplay but are really lovely in their execution and old-school console feel.

Check out the Dev Log for details about Phaser 3.16 RC1, new snapshot features, how to use fullscreen mode and more. As usual, be sure to read this newsletter on the web so you don't miss anything.

Got a game or tutorial you'd like featured? Simply reply to this email, or message me on Slack, Discord or Twitter. Until the next issue, keep on coding!

Phaser Sticker Packs + badges + freebies

The Phaser sticker packs continue to prove popular :) and each sale helps contribute towards the development of Phaser, so it's win-win! All orders placed this month will get a free **Phaser pin-badge** and a new **transparent sticker** included as well.

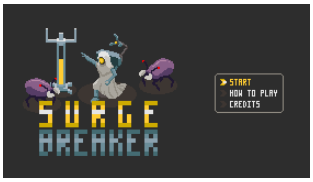
Click here to order a sticker pack.

The Latest Games

Game of the Week

Goodnight

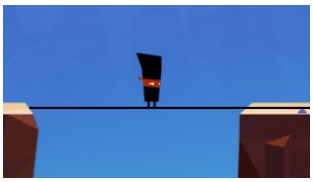
Team up with your unconscious mind to find your way out of a dream that trapped you in. Will you ever wake up?



Staff Pick

Surge Breaker

Zap the crawlers, grab the cells and try to power-up the energy bomb before you get overwhelmed, in this action-strategy game.

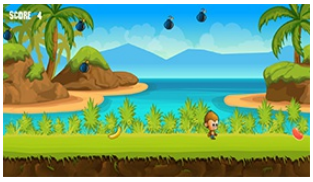


Stick Freak

Can you make it from one mountain to another by carefully stretching out the stick so you land in the right place?

The Secret Cave

You are Stefano, a trainee alchemist with a secret you need to share. But who do you choose to trust?



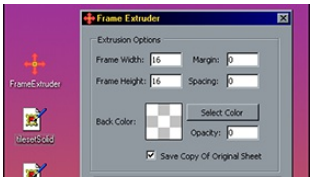
Monkey Boom

Run, dash and fly to collect the bananas and avoid the bombs.

What's New?

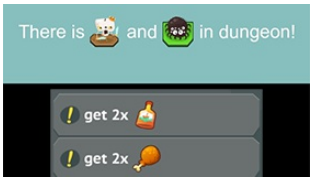
Criando um jogo completo com Phaser

A comprehensive series of videos that take you through the process of creating a complete game, from start to finish (Portuguese)



Frame Extruder

A Windows based app designed for extruding tilesets to avoid the bleeding pixels problem present in WebGL rendering.



Add Sprites into a Font as characters

How to mix Sprites into your Bitmap Font, so you can easily inject them into your game text.

Circular Endless Runner Tutorial Part 2

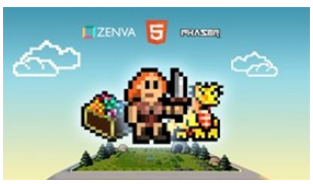
In this part of the series endless obstacles are added into the mix.

Circular Endless Runner Tutorial Part 3

In the next part, collision detection with the spikes is added using line collision instead of physics.

Generator Phaser Plus

A Yeoman generator that makes starting up Phaser 3 game projects dead simple.



Phaser 3 Game Development Course

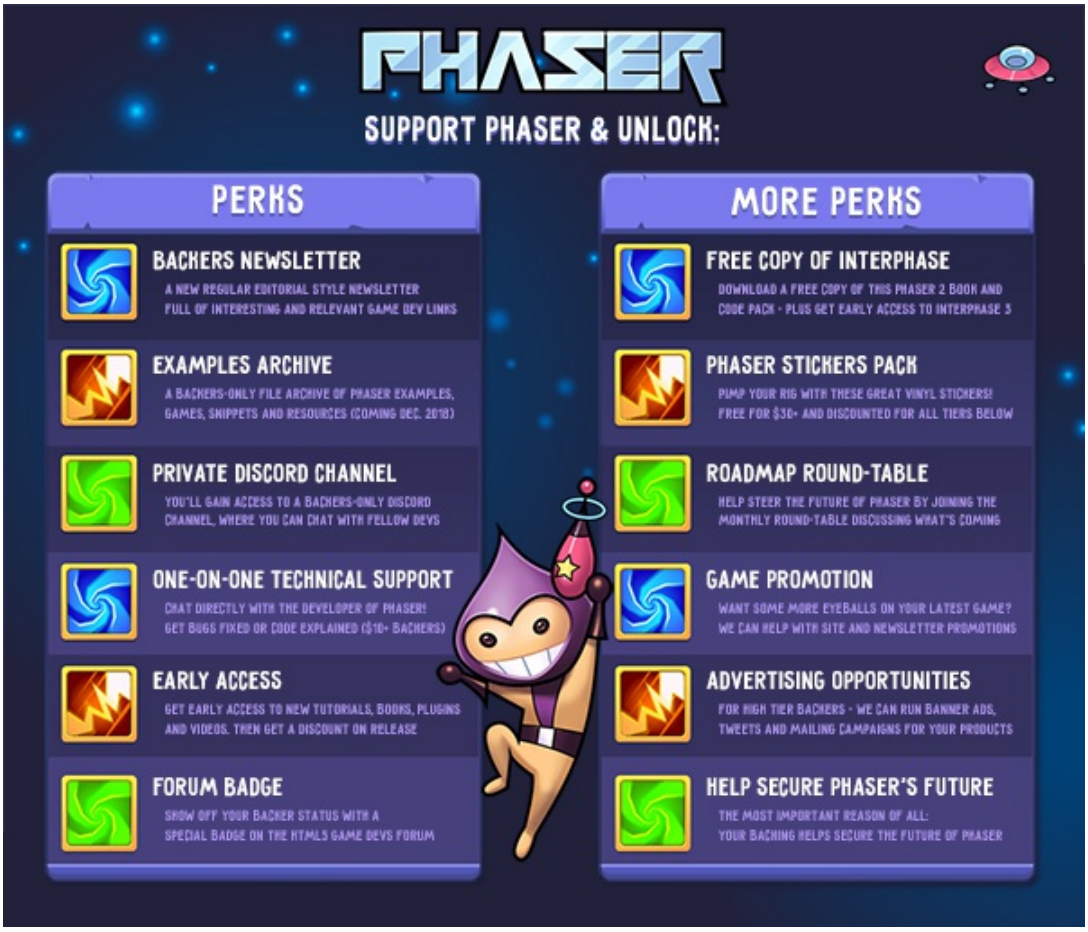
A complete Phaser 3 and JavaScript Game Development package. 9 courses, 119 lessons and over 15 hours of video content. Learn to code and create a huge portfolio of cross platform games.

Help support Phaser

Because Phaser is an open source project, we cannot charge for it in the same way as traditional retail software. What's more, we don't ever want to. After all, it's built on, and was born from, open web standards. The core framework will always be free, even if you use it commercially.

You may not realize it, but because of this, we rely 100% on community backing to fund development.

Your support helps secure a constant cycle of updates, fixes, new features and planning for the future. There are other benefits to backing Phaser, too:

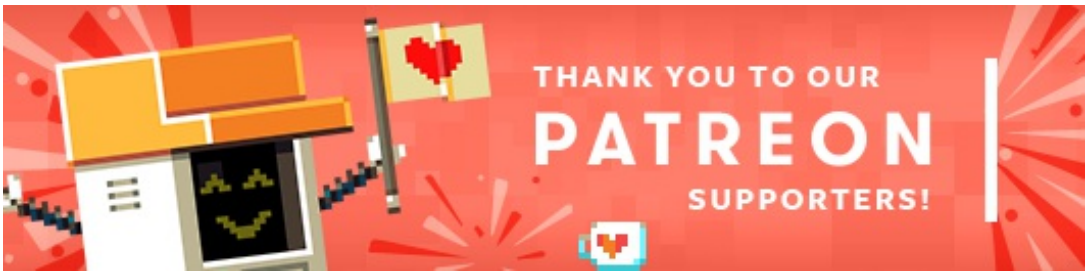


Click to see the full list of backer perks

I use Patreon to manage the backing and **you can support Phaser from $1 per**

month. The amount you pledge is entirely up to you and can be changed as often as you like.

Please help support Phaser on Patreon



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

askdaddy
Jochen Hartmann
Julián González Picatoste
Kreso Cvitanovic
Linemancer Works
Sam van Berlo
Silvan Strübi
Silviu-Ovidiu Iancu
TMoney
Tomáš Hianik

Also, thank you to **Chris Andrew** for increasing their pledge and **James Skemp** for the donation :)

Dev Log #138

This week of development has been all over the place. Not in terms of concentration, I've been fully dedicated to coding, but with regard to which parts of the API I've been working on. The work culminated on Friday with me publishing 3.16 Release Candidate 1. Let's take a quick trip through some of the

key areas covered.

Fullscreen Support

On Monday I began implementing support for the Fullscreen API into the new Scale Manager. It's one of those classic browser APIs that was a bit of a mess when first introduced, and since evolved to agree to something that looks like a standard. Except, you can't rely on the standards and instead have to perform all kinds of browser prefix filtering and handling of discrepancies between them.

Thankfully, it's a quite small API, so doesn't take too long to cover all the edge cases. And the benefit, of course, is that you can allow your games to enter fullscreen mode. It's worth pointing out that 'full screen' means literally that: the browser completely fills the screen, removing all UI, just as if you were playing a standard desktop or mobile game. Each browser shows a standard warning when you enter fullscreen for the first time, but beyond that, the experience can often be lovely due to how immersive it feels.

On desktop I would argue it's quite unusual to play web games fullscreen. Especially with wide-screen monitors, the jump out can feel a bit jarring, or even invasive. On mobile, however, it feels fantastic. We tend to multitask on mobiles a lot less, so entering fullscreen mode to play a game for a few minutes just feels right. Thankfully, both Android and iOS 12 support fullscreen mode now, and it truly looks lovely. Older versions of iOS won't do anything, so you're stuck in browser hell, with the Safari UI getting in the way of things. Older Android browsers don't support it either, but using Chrome on Android is pretty standard these days.

Thankfully, it's easy enough to check if support is there and then to get into fullscreen mode. You have to do it from a user interaction. This means your game cannot automatically go fullscreen. Just in the same way it cannot automatically play audio anymore. The user has to have 'interacted' with it at least once, but that's fine - it's quite normal to have a 'Go Fullscreen' button on a title page somewhere. And, on the plus side, the interaction that kicks you into fullscreen is also suitable for unlocking audio too.

Toggling fullscreen mode is trivial:

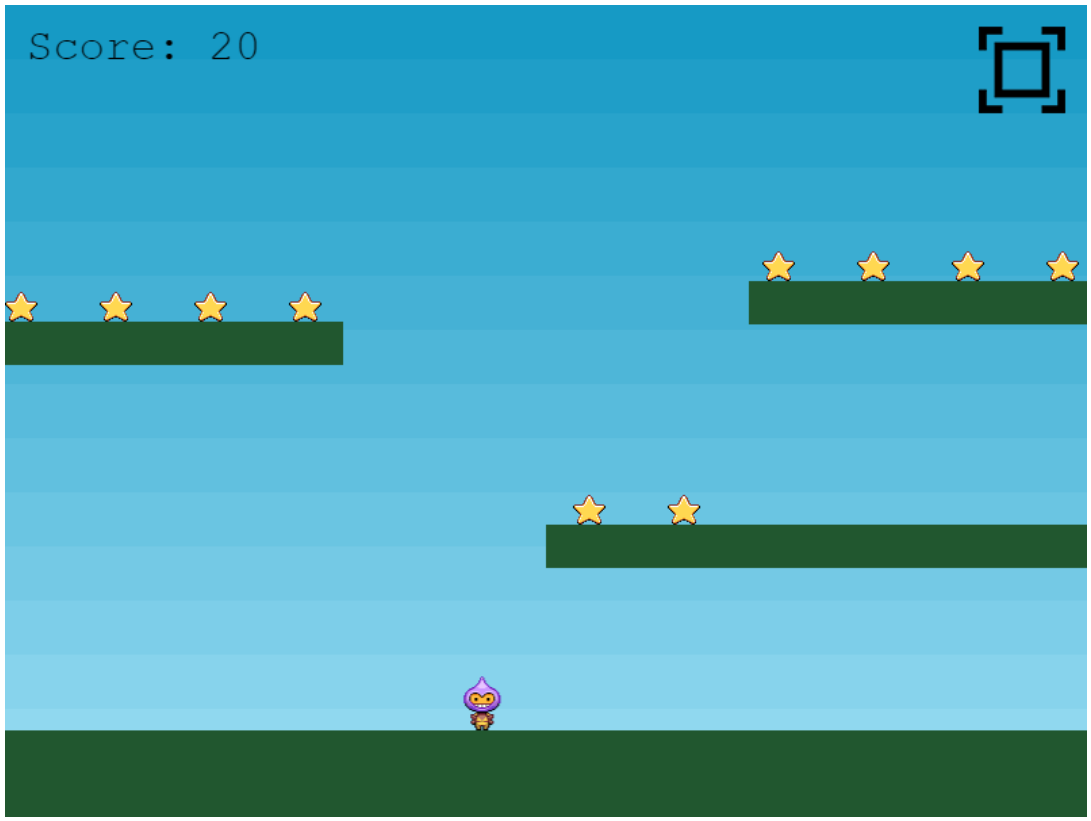
```
button.on('pointerdown', function () {

    if (this.scale.isFullscreen)
    {
        this.scale.stopFullscreen();
    }
    else
    {
        this.scale.startFullscreen();
    }

}, this);
```

Or you can even use the `toggleFullscreen` method instead.

You can try it out on the following example. Cursors to move, click the fullscreen icon to toggle fullscreen mode, as long as your browser supports it.



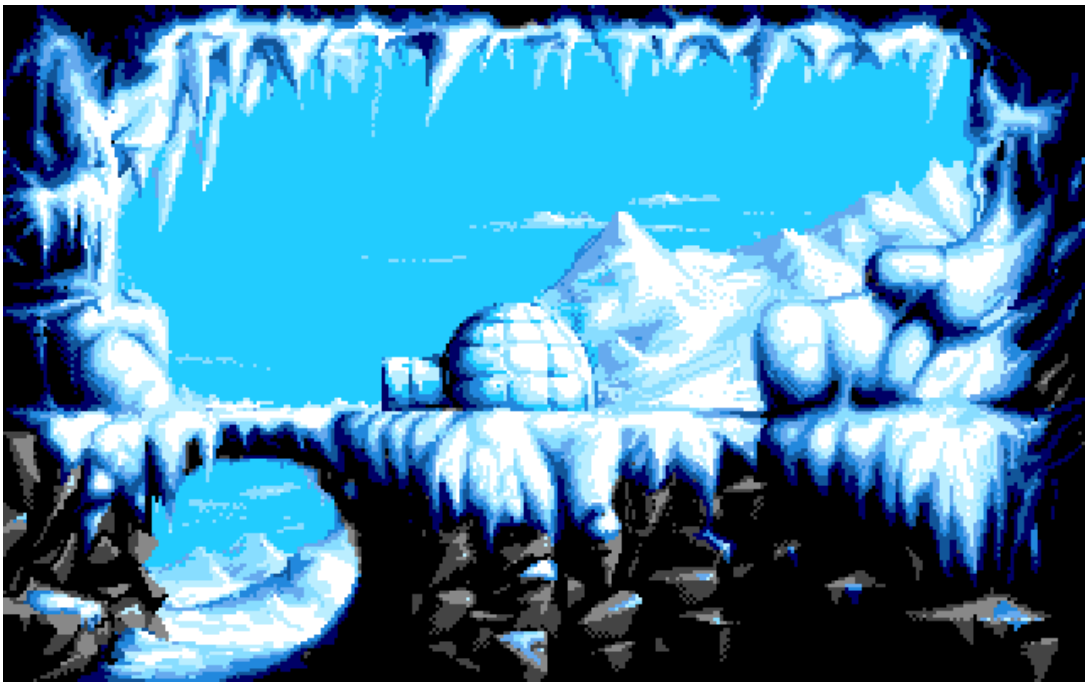
The Scale Manager will emit events should the fullscreen mode change outside of your control (i.e. the user presses ESC, or tabs to another window), so you can deal with that gracefully. I tested Fullscreen mode on Safari on macOS, iOS 12, Android, Chrome, Firefox, Edge and IE11 and it's thankfully working across the board. Somewhat unusually, Edge was the hardest to get going, where-as

IE11 ran first time. Go figure. I need to make a few small tweaks with regard to the fullscreen container. But, they should be in by the time you're reading this.

While I was working on fullscreen support I came to a decision regarding how input events are handled. To go fullscreen you need to make the browser API request *during* an actual input event handler. If the request isn't made in that exact same handler then the browser will reject it. This was a problem, because the way Phaser handles input events is to take the native DOM event and place it into a queue. This queue is then processed by the game loop. Depending on when the native event occurs it may have to wait for one frame before being processed.

I built the event system like this, i.e. fully managed in a queue, so that I would be in absolute complete control over *when* the input events were dealt with during the game step. This is because during the development of Phaser 2 I hit so many issues with DOM events messing with the state of Game Objects and causing all kinds of weird edge-case errors, for which there is still code in there today to deal with, that I was determined not to fall into the same trap again.

There is one, massive, drawback to using an event queue though. You can't initiate any browser actions in your callback handlers that require elevated security. And these days, more and more things do, so, it was time for a change.



Input Event Changes

The alternative to putting the DOM Input events in a queue, and processing them

later, is to process them immediately. It's literally A or B, there's no other choice. And while Phaser 3 had been built entirely with route A in mind, it was clear that it needed the alternative.

This would solve two key issues: First, you can finally invoke browser actions in your callbacks that would otherwise be blocked. Such actions include: going fullscreen, playing audio for the first time, or opening a new browser window / tab.

Secondly, it also means that you could now have buttons that performed these tasks. Previously, I had added the Input methods `addUpCallback` (etc) which allowed you to invoke your own functions during the native DOM input handle. However, they were global callbacks, which meant you couldn't, for example, have a *button* in your game that would launch Twitter in a new window. Instead you'd have to add a global 'up' handler and then determine for yourself if it occurred over the button, or not. It works, but it's not exactly elegant.

Finally, although not an issue it's still important: using an event queue means there is a very slight lag between the input event itself, and it being reflected by what is happening on-screen. Having a Sprite follow a mouse pointer is a good example.

The get around all of these I refactored the Input Plugin so that they would respond immediately to the native DOM events. This means, when you get a 'pointerdown' event in your game code, it has been called as the result of an actual DOM down event. So you're free to now open windows and go fullscreen and all the other fun stuff that was more problematic to handle previously.

Although this was quite a small change in terms of the code required, it's a large change in the way the API works. I tested it as carefully as I could, but that testing doesn't factor 3rd party plugins, of which there are a growing number, or products built on-top of Phaser, such as Quest AI.

Which is why I have kept the event queue approach in the API as well, and you can toggle back to it using the game config flag `input.queue`. I'll retain both modes in there for the next few releases, which should give us all time to find those unknown edge-cases and how to deal with them, but ultimately the queue system and it's related methods and properties are now deprecated, so please try and use the new one as a default.

Snap, Crackle, Pop!

You'd be surprised how many features in Phaser are the direct result of a

conversation on Slack or Discord. Typically, it starts with a question. Then I'll explore a solution with whoever asked it, and finally, more often than not, it results in a small tweak to the API or the JSDocs.

I don't tweak the API because they asked me to. That isn't what happens. I tweak it, because when I see the problem from their angle, I see how it can be made easier for them and everyone else, usually with only minimal adjustments on my end. The change I made to the renderer snapshot feature on Friday is a perfect example of this.

The question was, on the surface, a simple one: "How do I get the pixel the player clicked on in my game?". Not the pixel from a specific Sprite, but from literally anywhere on the game canvas. They had tried various things, like using a Render Texture and wanting to draw that to a Canvas Texture in order to extract a pixel from it, but it just wasn't working as expected.

Both the Canvas and WebGL Renderers have a built-in snapshot feature. If you call the `renderer.snapshot()` method and provide a callback, then the next time the renderer updates it will take extract all the pixels from the current WebGL context, create an Image object from that and send it to your callback. Which is great, but overkill if all you need is a single pixel.

Looking at the snapshot method it became instantly obvious that it would require a minimal change to expose the area being captured. Internally, snapshot uses the WebGL readPixels function, and then builds an ImageData from that. The readPixels function allows you to specify a region. In 3.15 and before, this region was hard-coded to be the whole canvas. All it would take would be allowing you to control the area being grabbed and suddenly this feature would become very easy to offer.

So, that's exactly what I did. Both the Canvas and WebGL Renderers still have their original `snapshot` methods, untouched (although now with far better jsdocs), but they've also gained two new methods: `snapshotArea(x, y, width, height)` and `snapshotPixel(x, y)`. I recoded the internal Snapshot functions to accept the extra position and dimension parameters and that was it. If you grab a single pixel then it'll return a Color object for you, and if you grab an area (or the whole thing) it will return an Image instead, which could then be added to the Texture Manager if you wish. It's also now properly using the CanvasPool, so it won't leak memory if called too often.

You can see the results in the following demo (mouse around and click to grab an area):

I really enjoy it when chats like this help the API evolve to become a whole lot more useful, with minimal changes, just tweaking functionality that was already there.

3.16 Release Candidate 1

On Friday, right before the Global Game Jam started, I published Phaser 3.16 Release Candidate 1 to both GitHub and npm under the 'beta' tag.

This release brings together all of the work I've done over the past three months. I published the full Change Log (so far) to GitHub as well, so I would urge you to read through it, as there are some proper breaking changes in place.

If you get the chance, please help test it! You can find ready prepared build files, examples of scaling code in the labs and if you want, you can generate the JSDocs for it and check those out too. There's still more tweaks to be done, indeed I did some today before publishing this newsletter, but I'm still happy we should get a final release this week.

Thank you to everyone who has helped, especially those posting issues found with RC1 on GitHub - if you find any, first check the Change Log to see if it wasn't a breaking change, like the keydown event names, and then post a new issue so I can look into it asap!

Due to the changes with the input system, which I talked about earlier, I'd like to get as many people testing RC1 as possible, please. Feel free to grab me on Slack, Discord or Twitter if you think you've found a bug, or just post it to GitHub. Even if you try RC1 and your game *just works* then that is worth telling me, too :)

Until the next issue, have a productive week everyone and keep an eye out for 3.16!

A great article on how bullets work in games. Mostly focuses on FPS games, but still fascinating reading.

This site lets you test-drive a whole bunch of different programming fonts.

Suggestions from the commit message generator can be a little close to home sometimes! Although usually hilarious :)

Phaser Releases

Phaser 3.16.0 Release Candidate 1 released January 25th 2019.
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.