Welcome to Issue 93 of Phaser World

If you've released a game or tutorial recently then please let me know about it. I'll happily feature it in the newsletter and on the web site. Just send me a link, either

on twitter, slack, email or post it to the forum (just be sure to tag it with Phaser on there).

Summer appears to be creeping quickly to an end, although it's hard to tell given the sheer amount of work going on around here at the moment. There's a bunch of Phaser 3 updates in this issues Dev Log, some great new games thanks to LD39 and the usual raft of quality tutorials.

So, until the next issue, keep on coding. Drop me a line if you've got any news you'd like featured (you can just reply to this email) or grab me on the Phaser Slack or Discord channels.

The Latest Games

Defend Your Homes >>> **Game of the Week**
An 80s inspired shoot-em-up created for Homes.com, one of the top real estate sites in the US.

Silly ways to die: Differences 2 >>> **Staff Pick**
Spot the differences in the scenes as the cute characters get themselves into all kinds of problems!

Power Grid Manager

Sat in front of an ancient computer terminal can you work out how to power the town?

Universe Hopper

Can you grapple from planet to planet in this highly addictive LD39 entry?

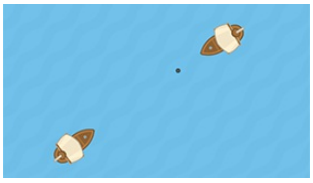
Tap Jet

Fly in space among the endless barriers and see how far you can get.

What's New?

RobotlegsJS Phaser Extension

A package that allows you to use RobotlegsJS with Phaser CE.



Creating a multiplayer Pirate Shooter Game

A comprehensive tutorial on remixing a multiplayer Phaser game on Glitch.

Faking a Swipe in Phaser

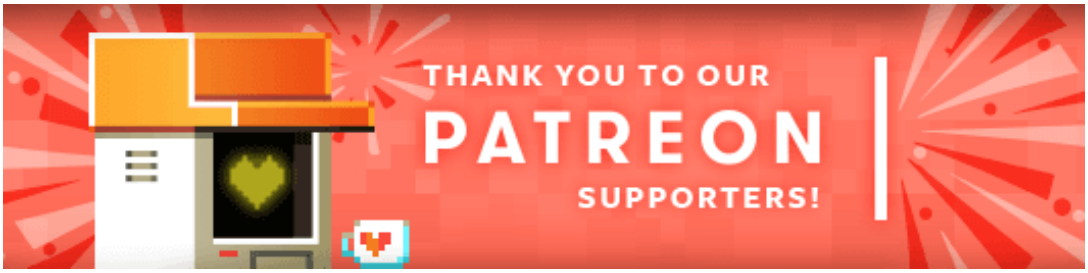
A short tutorial on faking the swipe gesture in Phaser games.

Flipping Legend Prototype Tutorial

A tutorial on creating a prototype based on the hit mobile game Flipping Legend.

Slither.io Tutorial Part 5

In part 5 of the tutorial series eyes are added to the snakes.



Welcome and a massive thank you to the new Phaser Patreons who joined us

this week: **Enzo Barbaguelatta Diaz**, **Xander Hayward** and **Richard Healy**. Also, thank you to **Ric Hoddinott** for his awesome donation.

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 #93

It was yet another busy week of development on Phaser. It started out with me focusing on more feedback from the Alpha release. A few of you had spotted bugs that I fixed, which is always a nice feeling. I also spent a lot of time in the first few days experimenting with more ways of generating the Phaser 3 documentation. I tried all kinds of packages, lots of which didn't work, some of which worked in part but didn't give me the results I needed.

It appears there are two main parts to API documentation: The format of the docs themselves, within the source code, and the parsing of them. JSDoc is the standard way to document JavaScript code and although jsdoc itself doesn't generate the best-looking docs, the actual tags used are well defined and generally sensible. What's more, lots of IDEs recognize them automatically too. So the conclusion is that I will continue to use the JSDoc format within the source code itself, just as v2 does.

However, the parsing of the docs and generation of the resulting documentation is still undecided. I have some quite specific requirements from the v3 docs, which don't appear to be readily served by the packages out there. I need the docs to generate a single JSON file, which is something JSDoc can do. I need them to pull in external files written in markdown and merge those with the output, so I can include extended examples in the source without having to embed them all in the code files. I also want to be able to output the docs as single markdown files, one per class. Lots of the more modern packages appear to favor outputting

just one single, massive, HTML or markdown file, that you jump around via anchors. This is no good to me due to the sheer scale of the Phaser 3 docs. I would also very much like the ability to output the docs into publishing ready formats like PDF or ePub. I need the doc generator to recognize only changed files, and not have to republish the whole lot again just because I edit one source file. Finally, I am pretty sure I'm going to need to have my own custom jsdoc tags to help define TypeScript definitions. Because there is no way we want to be doing this by hand again. I need to be able to generate the TS defs from source.

The conclusion I'm coming to is that no package out there provides all of this. I will spend a bit more time investigating but I'm not hopeful. It may well transpire that I need to create my own documentation generator. I can still use lots of the tools that already exist, like esprima, to do the grunt work, but I was really hoping there would be something I could use off the shelf. If you know of anything please drop me a line (rdavey@gmail.com).

Tween Manager Completion

As well as docs I also spent the week completing the Tween Manager. This was a brand new system for Phaser 3. I had already worked on it a lot but the manager itself was missing some final features in order for me to be able to mark it off as 100% complete. I also created some performance tests for it. Literally throwing tens of thousands of tweens around and checking it didn't blow-up the profiler or leak memory anywhere.

Timescale

Each Scene in your game has its own Tween Manager, and each Tween Manager has its own global time scale. By setting the time scale on a global basis all tweens being processed by the manager are scaled immediately. This allows you to create 'slow-mo' or 'speed-up' effects for a specific Scene, without impacting any other scene that may be running in parallel:

Use the arrows to control the global time scale

The above example has a bunch of independent tweens running, but all are impacted by the change in global time scale initiated by the buttons.

You can also change the timescale of an individual tween

Change the tween timescale

Here we have a single tween updating 5 different sprites. The arrows control the time scale of the tween itself, allowing you to speed it up or slow it down. This example also demonstrates another new feature. The ability for a tween to flip a Game Object when it yo-yo's or repeats. If you look at the example you'll notice the fish flip horizontally as they reach the edge of the screen. This is caused by a new optional config setting:

```
var tween = this.tweens.add({
    targets: [ image1, image2, image3, image4, image5 ],
    x: 700,
    duration: 4000,
    ease: 'Sine.easeInOut',
    flipX: true,
    yoyo: true,
    repeat: -1,
    delay: function (i, total, target) {
        return i * 1000;
    }
});
```

If you set **flipX** (or **flipY**) in the config then it'll flip the sprite for you automatically. There are cases when you will need more complex logic that controls when a

sprite flips or not, but for simple tweens like this one it works really well and avoids you having to define callbacks.

Tween Callbacks

I've also implemented all of the tween callbacks. There are 6 in total covering the following events: onStart, onUpdate, onYoyo, onRepeat, onLoop and onComplete. These can be set either in the tween configuration or via the Tween.setCallback method. The Tween Manager uses callbacks rather than events because in all likely hood you are going to be creating hundreds of tweens that never need any kind of callback or event listener. Yet in an event based system, those events get dispatched anyway, regardless if there is anything listening to them or not. But with a callback, it is never invoked unless there is an active listener. So by using callbacks, we avoid un-necessary event creation and you get the same flexibility as before. It also keeps our tween system in-line with others like TweenMax. There are lots of callback examples in the labs if you'd like to take a look at how they are structured.

Tween Manager Helpers

I had a list of functions I wanted to add to the Tween Manager before I would be happy it had all that was needed, and these are now in place. The new features are all to do with tween inspection and control and include:

- getAllTweens
- getTweensOf
- isTweening
- killAll
- killTweensOf
- pauseAll
- resumeAll

Lots of the above take either a single Game Object as their argument, or an array of them. You can easily pause or resume all running tweens, kill off certain tweens, or just get tweens from a given object. I also completed the shutdown and destroy methods, rounding the Tween Manager off nicely.

Performance Tests

It was important to me that Phaser 3 tweens performed *far better* than v2 but also were powerful enough that you didn't need to think about using an alternative. I'm

happy to have achieved this now and created a few performance tests to demonstrate it (to myself as much as anyone else!).

Hold mouse down to add tweens

The example above demonstrates the creation of unique endlessly running tweens. It's using a Blitter object for drawing the sprites, but the real test is in the running of thousands of tweens in parallel. I could comfortably push it into the tens of thousands before even noticing it on the performance chart, and at that point, you're into the realms of the rendering causing issues as well.

In the next test, I monitored what would happen if I created and destroyed thousands of tweens over a long period:

10,000 tweens

I always wanted the Phaser 3 tween system to be highly disposable. By that I mean I wanted you to be able to just fire off tweens left, right and center without really caring about it, and that Phaser would gracefully manage their creation and destruction, without any negative impact on your game.

This is what the test above is for. It creates a brand new tween every 10ms that has a random duration of between 500ms to 1000ms and then dies. Using a timer this is repeated 10,000 times. This means there are approximately 90 active tweens at any one time, but as the timer progresses thousands upon thousands of them are destroyed. So it's a good test of impact on gc and memory. Thankfully both cope very well. I left Chrome Profiler running on the above test for 1 minute and over all of that time Major GC accounted for only 0.3% (9.4ms total). The biggest activity was, of course, the Animation Frame Fired, taking 74% time, but out of that Minor GC was so tiny it didn't even register, at 0.0%. This lead to a JS Heap fluctuation of just 1.4 MB, and remember that includes all of Phaser (not just the Tweens), all the work the browser is doing, and all the work the Profile is doing as well. In short, I'm very happy that things are clearing up properly and we can all use tweens in our games with reckless abandon (to a degree, anyway!)

Deferred Lighting Lands

Last issue I showed you the new Light Layer that Felipe had added into v3. This used a Forward Rendering technique, which limits you to how many lights you can have in a scene before it really starts to kick performance in the nuts. Last week he implemented the deferred lighting system.

Deferred works differently to Forward in that it dramatically reduces the fragment count by performing the lighting calculations on the pixels on the screen, rather than each visible fragment. It requires the browser to support the WebGL Draw Buffers extension, which not all do, so the Light Layer will create the shader it requires based on browser support. You can also check the maximum number of lights possible by calling LightLayer.getMaxLights.

Click to add lights

The maximum light constraints are enforced via a Consts file that is built into Phaser. We kept this separate from the source to allow you to easily change the values and then rebuild Phaser without needing to touch the code itself. Because of the way the LightLayer objects are created this isn't something you can set in the Game Config I'm afraid. The defaults are sensible best-guesses but are there to be changed as required. Lights are powerful things, creating a whole different look over the scene with very little code, but they come at a cost. Especially when shadows are introduced.

That's all for this week. There will be another shorter Dev Log in issue 94 next

week, and then we're taking a week off for a well earned holiday. On our return, we'll be heading for the first Beta release.

Phaser 3 Labs

Visit the Phaser 3 Labs to view the new API structure in depth, read the FAQ, previous Developer Logs and contribution guides. You can also join the Phaser 3 Google Group. The group is for anyone who wishes to discuss what the Phaser 3 API will contain.

A brilliant read about the Atari Star Wars arcade game and the process involved in the making of it.

Kongregate's John Cooney attempts to encapsulate the immense history of Flash games and how it has shaped the current game industry in his GDC Talk.

This is an awesome making-of The Loot Train Attack sequence from Game of

Thrones, looking at the SFX and stunts.

Further Reading ...

Phaser Facebook Group
GameDev.js Weekly Newsletter
HTML5 Game Development
Lostcast

Phaser Releases

The current version of Phaser CE is 2.8.3 released on July 24th 2017.

The current version of Phaser is 3.0.0 Alpha released on July 31st 2017.

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.