Web Version Unsubscribe

Welcome to Issue 135 of Phaser World

It seems strange to be saying it 7 days in, but Happy New Year! This is the first issue of 2019 and I trust you all had a good festive time off. I played far too many board games and enjoyed a nice, if short, break. Although, to be honest, I was raring to get coding again towards the end! This issue we've a brand new Dev Log, some fun new games and of course tutorials!

Be sure to read it on the web so you don't miss anything.

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.

Fresh batch of Phaser Sticker Packs arrived

Due to their popularity, I've had another batch of the Phaser stickers printed, so there's now plenty in stock. The next 100 orders will get a **Phaser pin-badge** and a new **pixel-art sticker** included as well, for no extra charge.

Click here to order a sticker pack.

The Latest Games

Game of the Week

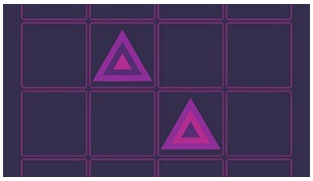
Parking Fury 3

In this furiously addictive game can you park your car on an empty parking spot without hitting other cars or obstacles?

Staff Pick

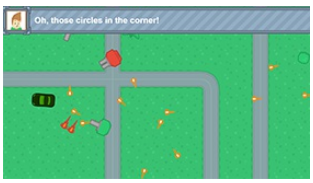
Protect the Dojo

Protect the dojo from the waves of attackers in this addictive upgrade / fighting game.



Geometric Memory

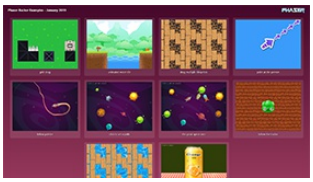
Match the tiles based on the differing geometric shapes within them.



Time Stream

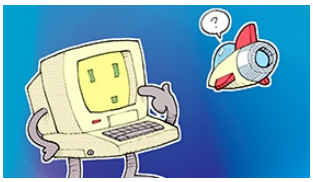
Jump between the time streams to align the signals, but don't take too long!

What's New?



Phaser Backer Examples January 2019

The January Phaser Backer Examples are now available including animated Tile Sprites and Pointer followers.

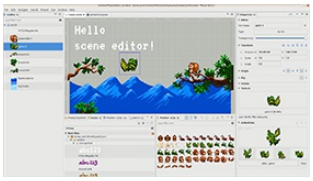


A Year in Review: Phaser in 2018 and beyond

A look back at the news and numbers of Phaser in 2018 and what the New Year holds.

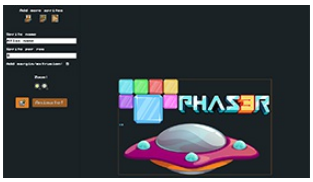
Student Success

How Lucas Knight stepped out of the box when designing his latest game and found Emojis!



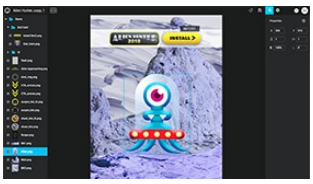
Phaser Editor v2 Progress Report

Read about the development progress of Phaser Editor 2 including the powerful new Scene Editor.



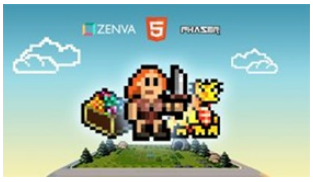
Atlas Packer Phaser

A free web based texture atlas creator for Phaser 3.



Quest.AI

Build complete Phaser games, interactions and product pages directly from PSD files, using a comprehensive visual front end to apply effects and publish to multiple platforms.



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

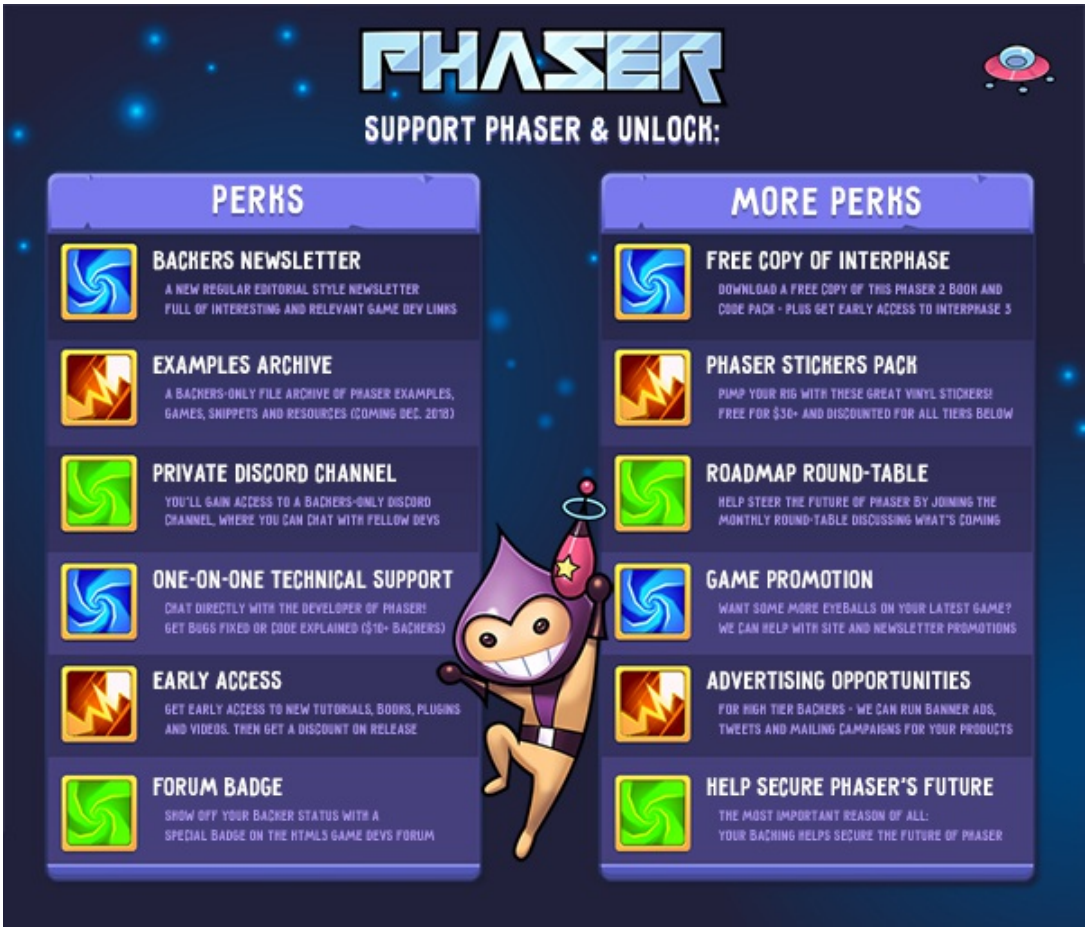
Because Phaser is an open source project, we cannot charge for it in the same

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. It's part of our manifesto that the core framework will always be free, even if you use it commercially, as many of you do.

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

Those funds allow Phaser to improve, and when it improves, everyone involved benefits. 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:

Ajay Sewradj
Alexander Higgins
Blaine Motsinger
Dan Wilson
David
Gidenilson Santiago
Jose Vicente Pons
Kevin Indreland
Lucian Niculai
Marwan Fikrat
Max Berman
Mike Thomas
Nathaniel Foldan
Nicholas
Nicolas Challeil
Philouze Simon
Ryan Cannon
Sergio Banderas
Tatiana Mizerova
Vladimir Odnoshivkin

Also, thank you to **Paul Schwarz** and **Yaroslav Mykhalchuk** for increasing their pledges.

Dev Log #135

Welcome to the first Dev Log of 2019. I had only been back in the office 2 days at the time of writing this, so it's a bit shorter than usual and half of it is taken-up with a mini tutorial.

My priority for the coming weeks is to finish and release Phaser 3.16. There are a few changes to wrap-up in the Spine Plugin, and the Scale Manager to finish, but both of these are progressing well and I'm increasingly happy with the state of them. The plan is to take a deeper look at the new Scale Manager in the next Dev Log.

Multiple Scene Drag

Just before I finished for Christmas there was a really interesting bug posted to GitHub Issues. Raised by @probt, issue #4249 detailed how if you had multiple Scenes running in parallel, with draggable Game Objects in them, that only Game Objects in the top Scene could be properly dragged. Those in any Scene lower down the Scene list would start dragging but never complete, getting stuck in a locked state.

I set about creating a minimal test case. Once I'd done that, I could see what was happening for myself. Although a test had been provided (thank you @probt!) it had a bit too much going on to be useful for debugging. It did, however, demonstrate the bug clearly, so I knew something was going awry. My own test case proved it as well.

It transpired that the issue was caused by a Pointers `dragState` getting locked. When a Pointer interacts with a draggable Game Object it maintains a 'drag state'. By default, this state is zero, meaning that the Pointer isn't dragging anything. During the life-cycle of the Input Plugin, it will run through a series of checks, changing the drag state as it goes. For example, there is a specific state for when a Pointer *was* actively dragging a Game Object in the previous frame, but has since been released, allowing the object to dispatch its `dragend` event.

There are 6 drag states like this in total that a Pointer can go through in its lifecycle. The issue is that this state was being stored on the Pointer instance itself, in its `dragState` property. Ordinarily, this would be fine, and indeed it worked fine if you only had a single active Scene, or only one Scene with draggable objects within it. But when you have more than one Scene the problem arises, because Pointer instances are global.

When a Scene's Input Plugin runs, it operates on Pointer instances that belong to the global Input Manager. This means that if SceneA sets the drag state of a

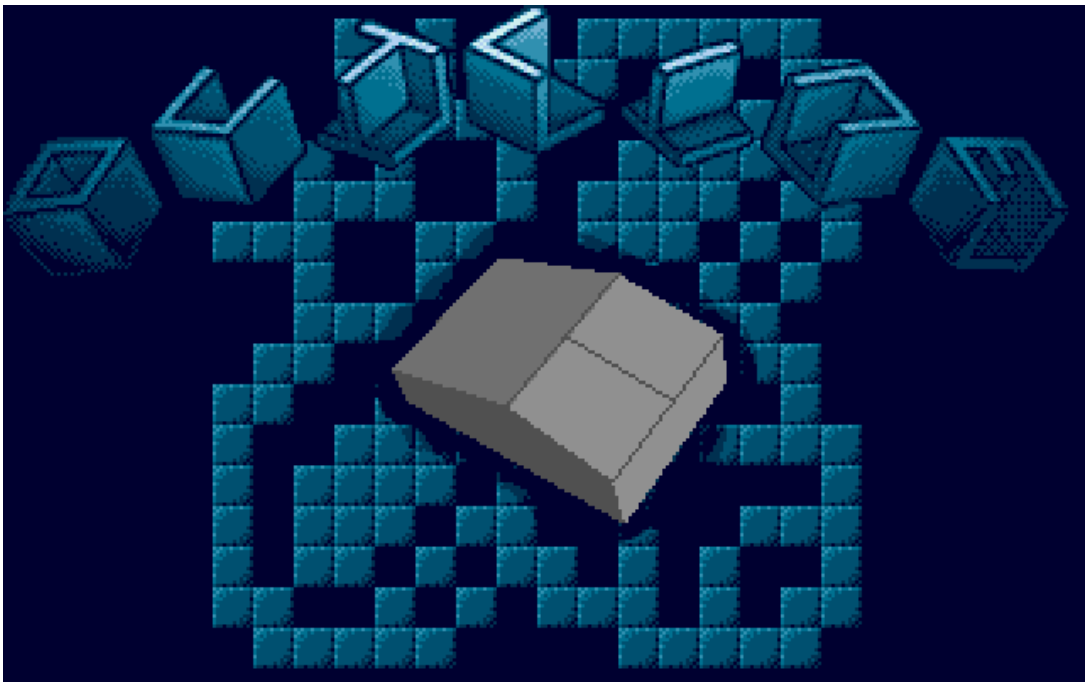
Pointer, then when it came around to SceneB checking the drag state, the state had already been mutated by SceneA and was now holding an invalid value. What's more, you can't just reset the drag state between the Scenes, because it needs to be remembered across potentially many frames.

The solution was to stop the Pointers being responsible for maintaining the drag state at all.

After some tests I added two new methods to the Input Plugin: `getDragState` and `setDragState`, both of which take a Pointer instance as an argument. The state is now stored locally in the Input Plugin, on a per Pointer basis, indexed by the Pointer ID. As the Input Plugin is owned by the Scene, it moves it away from being stored on a global level, to being stored on a Scene level.

I ran a few more tests, tidied up the code and pushed the fix. Now, you can have draggable Game Objects spread across as many parallel Scenes as you like. As with most issues, the fundamental change that was made to the API was quite small. The results, though, are significant. While it may not be the most interesting of fixes, and is likely to be hastily scrolled-past when scanning the 3.16 Change Log, it's still a really important one.

It was a bug that came about because an early design decision didn't factor in a specific use-case. Once the flaw was demonstrated, it was easy to re-evaluate the approach and replace it with a better one. I like to think it's a prime example of how the feedback loop provided by GitHub can sometimes work *really* well.



Tutorial: How to make an Arcade Physics Sprite stop at a specific position

Someone asked me on Twitter how they could make an Arcade Physics Sprite move to a specific location and then stop once reached. They wanted to use normal physics forces (i.e. velocity) to move the Sprite and not a Tween. While Arcade Physics has a built-in method `moveToObject`, all it does is set the initial velocity required. It doesn't monitor the object and stop it when it reaches the given destination. For that, we'll need some extra logic.

First, let's create a simple test case. Here we load an image, create an Arcade Physics Sprite and position it in the Scene:

```
var config = {
    type: Phaser.AUTO,
    width: 800,
    height: 600,
    parent: 'phaser-example',
    physics: {
        default: 'arcade'
    },
    scene: {
        preload: preload,
        create: create
    }
};

var source;

new Phaser.Game(config);

function preload ()
{
    this.load.image('flower', 'assets/sprites/flower-exo.png');
}

function create ()
{
    source = this.physics.add.image(100, 300, 'flower');
}
```

Next, let's add an input handler to set our 'target' destination. When you click anywhere on the Scene, it will use the pointer position as the destination and then call the `moveToObject` function. The x and y values are stored in `target`, which is a Vec2:

```
this.input.on('pointerdown', function (pointer) {

    target.x = pointer.x;
    target.y = pointer.y;

    // Move at 200 px/s:
    this.physics.moveToObject(source, target, 200);

}, this);
```

This is all good and well. If you now click, the source Sprite will start off quite merrily on its way, but it still won't stop yet. To do that, we need to monitor the Sprite in our update loop and figure out how far away it is from the destination target:

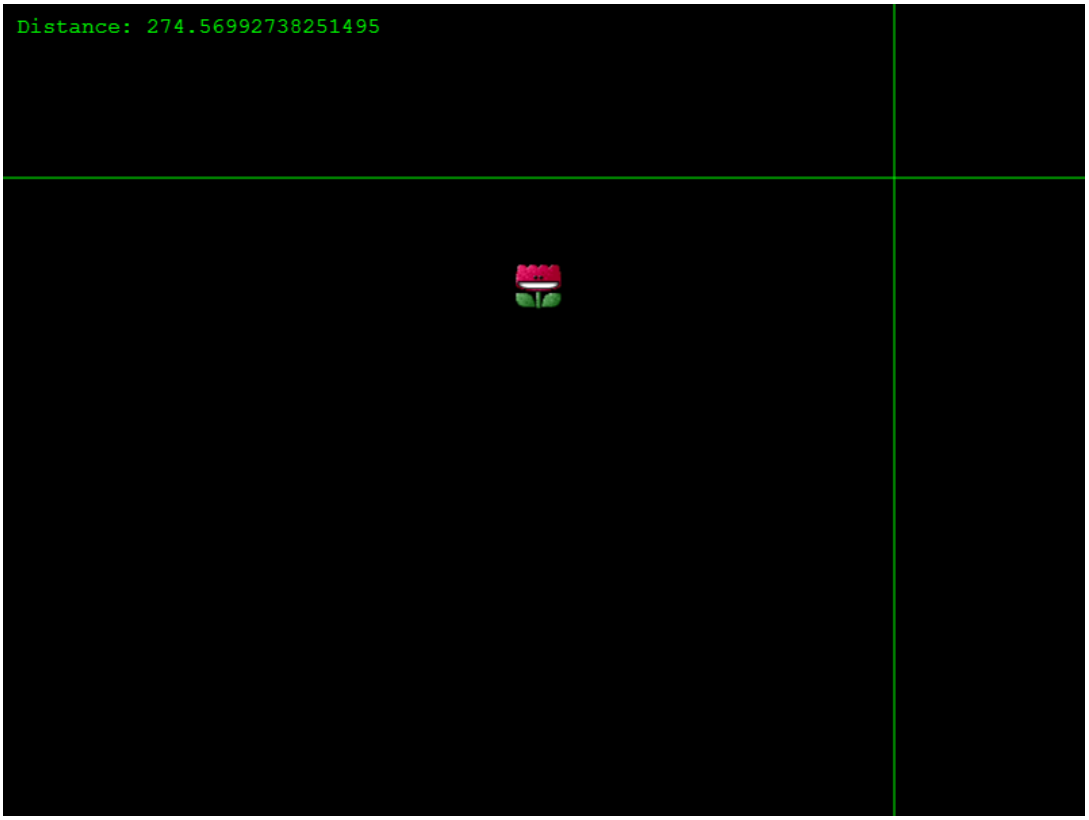
```
function update ()
{
    var distance = Phaser.Math.Distance.Between(source.x, source.y, target.x, target.y);

    if (source.body.speed > 0 && distance < 4)
    {
        source.body.reset(target.x, target.y);
    }
}
```

This uses the built-in distance function to work out how far the source is from the target. If the source is moving, which is determined by checking its speed, and if the distance is less than 4, then we consider it as having 'arrived'. The distance test is used because, due to the way physics and number rounding works in JavaScript, the Sprite is almost certainly never going to *exactly* reach the target coordinates. So instead, we check to see if the Sprite gets within an acceptable distance from the target, and if it does, bingo, the body is reset.

Calling `reset` on an Arcade Physics body cancels out all of its current velocity and also, optionally, resets the position of the body to the given x/y coordinates. Because we know our Sprite isn't going to be at *exactly* our target coordinate we take advantage of the reset arguments to force it there.

You can test it for yourself in the following example, which you can download from the Phaser Labs:



Why did we use a distance value of 4? And not a lower value? It's because of the speed the Sprite is moving at. The faster it goes, the more tolerance you need to allow for in your calculations, as the further it can 'overshoot' its target in a given step. The slower it moves, the tighter the distance check can be.

There are also other things which may prevent it from ever reaching the target. For example, if a static body is in the way. It will hit the body and possibly rebound (depending on its bounce setting), forgetting all about its target. Calling `moveToObject` simply sets a velocity that should, given a clean run, get the body to that location. Once the velocity changes, however that may happen, all bets are off. Even so, this simple distance check is often more than enough to ensure an object reaches its intended position.

That's it for this Dev Log. At the time of writing it's early Monday morning, so there's a whole week of Phaser development ahead of me - let's see what it brings!

Tiny Emus is a collection is of 8-bit emulators that run directly in the browser, such as the Amstrad CPC, ZX Spectrum, Acord Atom and more. They're all preloaded with some really lovely demos or games and if you're not careful it's easy to waste hours of time here :)

Vanilla Tilt is a smooth 3D tilt JavaScript library.

Peter Capaldi reads "The Magic Wood" by Henry Treece.

Phaser Releases

Phaser 3.15.1 released October 16th 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.

Preferences Forward