PHASER

A GUIDE TO THE SCALE MANAGER

Richard Davey

A Guide to the Phaser Scale Manager

Richard Davey

This book is for sale at http://leanpub.com/phaserscalemanager

This version was published on 2014-12-04

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Richard Davey by spreading the word about this book on Twitter!

The suggested tweet for this book is:

A Guide to the Phaser Scale Manager is out now

The suggested hashtag for this book is #phaserjs.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#phaserjs

Contents

Welcome

Modern HTML5 games need to be able to scale to the resolution of the device on which they are running. In order to do this Phaser has a built in Scale Manager. This guide covers the various different aspects of the Scale Manager, its properties and the various scaling modes on offer.

Phaser 2.2

Please note that this guide only applies to Phaser 2.2.0 and above.

Picking a resolution for your game

No matter which scale mode you use. No matter if you opt to create a fully responsive game and align everything to the very last possible pixel; you still need to decide upon a base resolution.

This is the resolution at which you should create all of your art assets. It's the resolution at which you should create visuals, plan out the alignment of screens and buttons and base everything around.

Picking the correct resolution isn't easy. There is no hard and fast rule because it depends a lot on the type of game you are making and the target audience. Here are some guidelines you can use to base your decision on:

Performance

How concerned are you about performance? Obviously everyone wants their game to run fast, but different types of game require different approaches. Essentially it's all about the quantity of pixels you need to shift every frame.

If you're creating a match-3 style game then for the majority of the time the game screen doesn't really change. Yes you may have falling gems, spot animation and particles going on - but in terms of the quantity of pixels actually moving at any one time it's probably quite low.

Contrast this to an endless runner game where you've got a constantly scrolling backdrop - maybe over parallax layers - with sprites and characters on top. The comparison here with the match-3 isn't the complexity of the game, but the graphical bandwidth required to render it. The more complex it is and the higher the base resolution, then the move power the device will need in order to meet your frame rate.

Using a lower resolution results in a faster game. The less pixels being drawn to the canvas, the faster it can draw them. The trade-off of course is that a lower resolution game, when scaled-up to a larger screen, will look worse as the graphics become anti-aliased.

Target Audience

Who is your target audience? This can often be an important issue when scaling games. For example if you're building a game aimed at small children then you need large, clear and well defined hit areas and buttons. Older and more 'accurate' players will be able to deal with smaller hit areas, or those closer together.

Cross Platform?

Which devices do you need to support? If you're building a game that you know will only be played on desktops then you can worry less about what screen size it may appear on, and instead focus on nice high res assets. The opposite is of course true.

Budgets

Depending on your art budget you could opt to create all of your assets at two different sizes: low res and high res (often called SD and HD, or 'mobile' and 'desktop'). When the game boots you can detect the available screen space and choose which set of assets to load as a result. Of course this isn't an option for everyone. Not all games have the budget, or need to be able to do this. But it's defintely an approach you should consider if the situation allows.

Bandwidth

With great resolutions comes great responsibility to your player's bandwidth. While it may be possible to create lovely retina graphics at the highest resolution possible they still have to be downloaded. This is all good and well on high-speed desktops, but mobile is usually a different matter. Animation rich games can suffer in this area too, as multi-frame sequences take time to pull down. Asset management is vital in any game, but it's especially important if you're trying to hit a wide variety of devices. Again, let your target audience dictate this one.

Portrait or Landscape?

Don't automatically assume your game has to run in a landscape orientation. Lots of them do, but there is no hard and fast rule here. Use whichever orientation best fits your game (or your client!).

After taking all factors into consideration you should have an approximate idea of what you should be aiming for. I'm not going to recommend any base resolutions to you. You should pick one after doing your research and analysing requirements. Build your prototype at that size and then test it. Do you still get the performance you need? Could you push the size higher or lower with minimal impact? Test until you're sure.

With this said, let's explore the different ways you can scale your Phaser game.

Scaling Concepts

It's important to get a few core concepts explained before we dive into any code. Phaser uses the concept of "the Game", "the World" and "the Parent" when dealing with scaling.

Game Size

Fixed Dimensions

Phaser has what we refer to as a "Game Size". This is the size of the game in pixels, as defined in the Game constructor. For example the following code sets the game to be `640 x 480` pixels in size:

```
var game = new Phaser.Game(640, 480, Phaser.AUTO, 'container');
```

The dimensions here control the initial size of the Canvas object that Phaser will create.

Percentage based Dimensions

As well as providing integers you can also specify a percentage:

```
var game = new Phaser.Game('100%', '100%', Phaser.AUTO, 'container');
```

Here the Phaser game will be set to 100% of the available width and height of the `container` element (known as the Game Parent). If you don't provide a parent element then it will be set to whatever the browser window dimensions are.

It's important to understand that specifying a percentage for the game size is only used when the game first starts up. It takes the dimensions of the container and calculates the size of the Canvas that will be created based on the percentage you gave. It doesn't automatically track the container size in the default scale mode, so if the container changes size (even the browser window) the game will not.

World Size

The Game World is where all Phaser game objects live. When you create a new Sprite it is added to the World and exists in world space. The World can be any size you need and the Phaser Camera is used to view into a "Game Sized" section of this world.

By default the World size is the same as the Game Size. You can change it:

```
game.world.resize(2000, 1000);
```

This will set the Game World to be `2000 x 1000` pixels in size. If we assume that the game was created with a size of `640 x 480` then by default the camera will be looking at the top-left corner of the World. It can be moved around by changing `camera.x` and `camera.y` respectively.

Sprites and other game objects can exist anywhere within this `2000 x 1000` area. As the camera moves over where they are located, or they move into the cameras view, they will be rendered.

For the sake of scaling, what is important to understand is that the world size is a virtual dimension and entirely independant of the displayed size of the game.

Parent Container

As Phaser games live inside a web browser they always have a parent container. This is optionally specified in the game constructor:

```
var game = new Phaser.Game(640, 480, Phaser.AUTO, 'myDiv');
```

In the code above when the game is started it will create a Canvas DOM element inside of the `myDiv` container. The Canvas will be `640 x 480` pixels in size in this instance.

Any CSS applied to `myDiv` controls what then happens to the layout. For example if `myDiv` only has a width of 500 pixels and `overflow: none` then you're not going to see the right-most 140 pixels of your game, as they'll be cut off. Equally if you've not specified any CSS then `myDiv` will expand to fit the Canvas - but again this is all controlled by your CSS and not Phaser.

Should `myDiv` not exist for any reason, or not be a valid DOM node, Phaser will make its parent the browser window.

If you don't specify a parent, i.e.:

```
var game = new Phaser.Game(640, 480, Phaser.AUTO, '');
```

Then Phaser will use the browser window as its parent and the Canvas it creates will be immediately added to the DOM. If there is already content in the page above the point where you create your Phaser game then it will appear after that.

The Game Parent, be it the browser window or a containing DOM element, is used to position the Phaser Canvas and manage all scaling.

From this point on in the rest of the guide we will refer to the game's parent container as simply "the parent".

Setting a Phaser Scale Mode

You can set the scale mode that your game uses at any point in your code, however we would strongly suggest you do it once at the start and then not change it again. The best place to do this is in the `init` method. In this simple example the scale mode has been set to one of the Phaser defaults and the game has been told to vertically and horizontally align itself within the parent container:

```
function init() {

        game.scale.scaleMode = Phaser.ScaleManager.NO_SCALE;
        game.scale.pageAlignVertically = true;
        game.scale.pageAlignHorizontally = true;

}
```

All scale operations take place directly on the Phaser Scale Manager. If you have `game` as a global variable then you can access this via `game.scale` as in the code above.

If you are using Phaser States for a multi-state game then this should go inside your Boot state file.

```
init: function () {

        this.scale.scaleMode = Phaser.ScaleManager.NO_SCALE;
        this.scale.pageAlignVertically = true;
        this.scale.pageAlignHorizontally = true;

}
```

The property `scale` is always available from a Phaser State and saves having to reference it directly via `game`.

We will use the "global game var" approach for the rest of the code in this guide. But you can easily swap it out for the style above for the same results.

Scale Mode 1 - EXACT_FIT

```
game.scale.scaleMode = Phaser.ScaleManager.EXACT_FIT;
```

The Exact Fit scale mode is a special-case scale mode that will take your game and resize it to fit into the exact dimensions of the parent. It does not maintain aspect ratio in this scale mode.

This means that if the parent isn't the same size as your game it will be stretched to accomodate, as seen in the image below:

Exact Fit 1

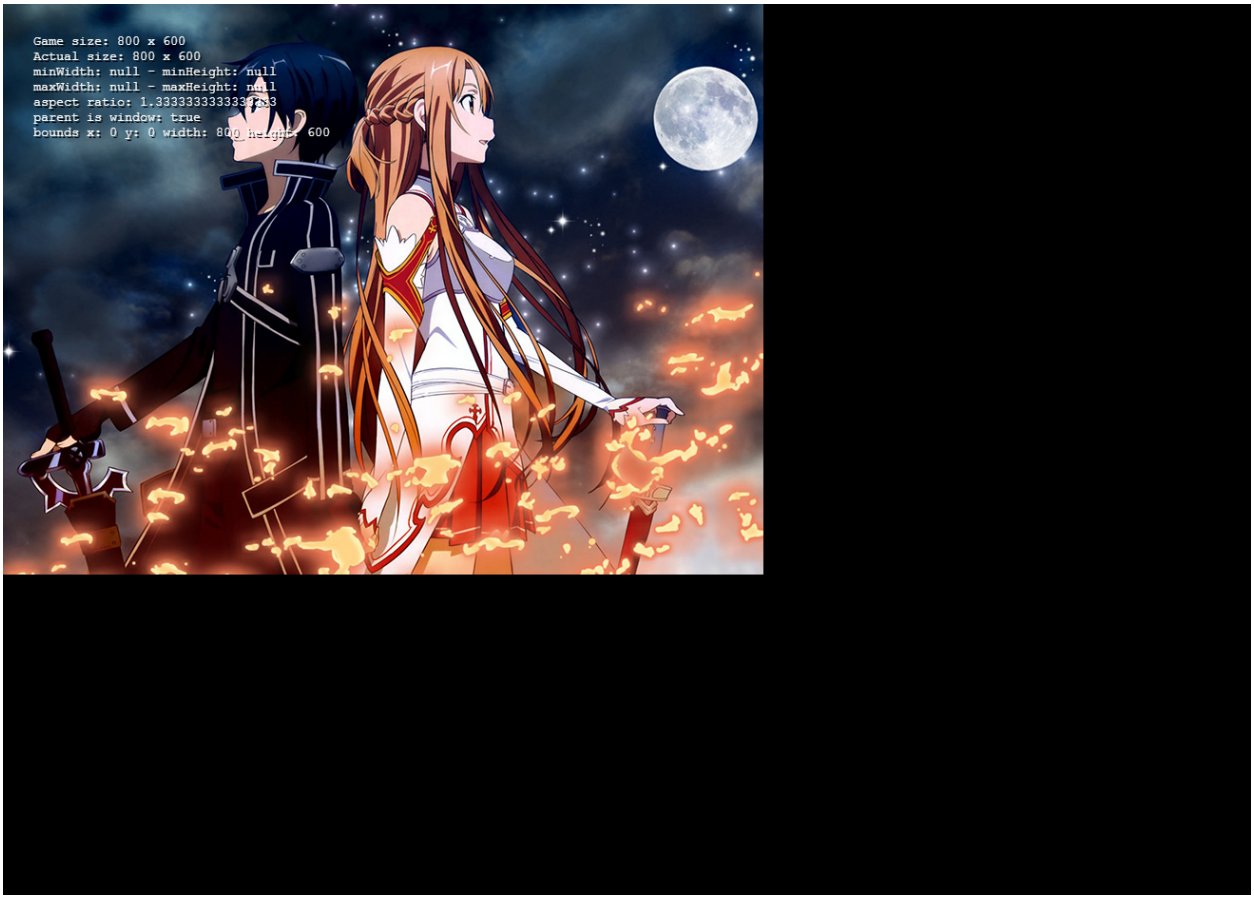
You can see that the image is quite distorted as the browser window was very wide and short.

Scale Mode 2 - NO_SCALE

```
game.scale.scaleMode = Phaser.ScaleManager.NO_SCALE;
```

This is the default scale mode. As you can appreciate from its name it doesn't actually do anything to scale your game at all. The Game Size you provide on instantiation is the size that will be used through-out, and nothing will change it.

The way this differs from EXACT_FIT is that this scale mode won't distort your game, no matter what happens to the parent.



No Scale 1

The large black area is the rest of the web page. The game is `800 x 600` and sits in the top left corner. Here is what happens if the parent becomes too small for your game size:

No Scale 2

Note: You can still change the Canvas size via normal CSS, but Phaser itself won't touch it while in this scale mode.

Scale Mode 3 - SHOW_ALL

```
game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
```

This scale mode works by resizing your game to fit the size of the parent, but maintains the game proportions as it does so.

Unlike with EXACT_FIT this scale mode won't distort your game. It works by calculating the aspect ratio of the Game Size dimensions. Then as the parent scales it resizes the Canvas automatically keeping the same aspect ratio.

Depending on the size of the parent this can sometimes lead to what is known as a "letter boxing" effect:

Show All 1

In the image above the browser window was `705 x 670` pixels in size. The game was `800 x 600`. In order to respect the original aspect ratio Phaser resized the game canvas to `705 x 529` pixels. The black area at the bottom is the web page background.

The Scale Manager has the ability to align the game within the parent, which can help improve the effect of letter-boxing. This is covered in a later section.

Scale Mode 4 - RESIZE

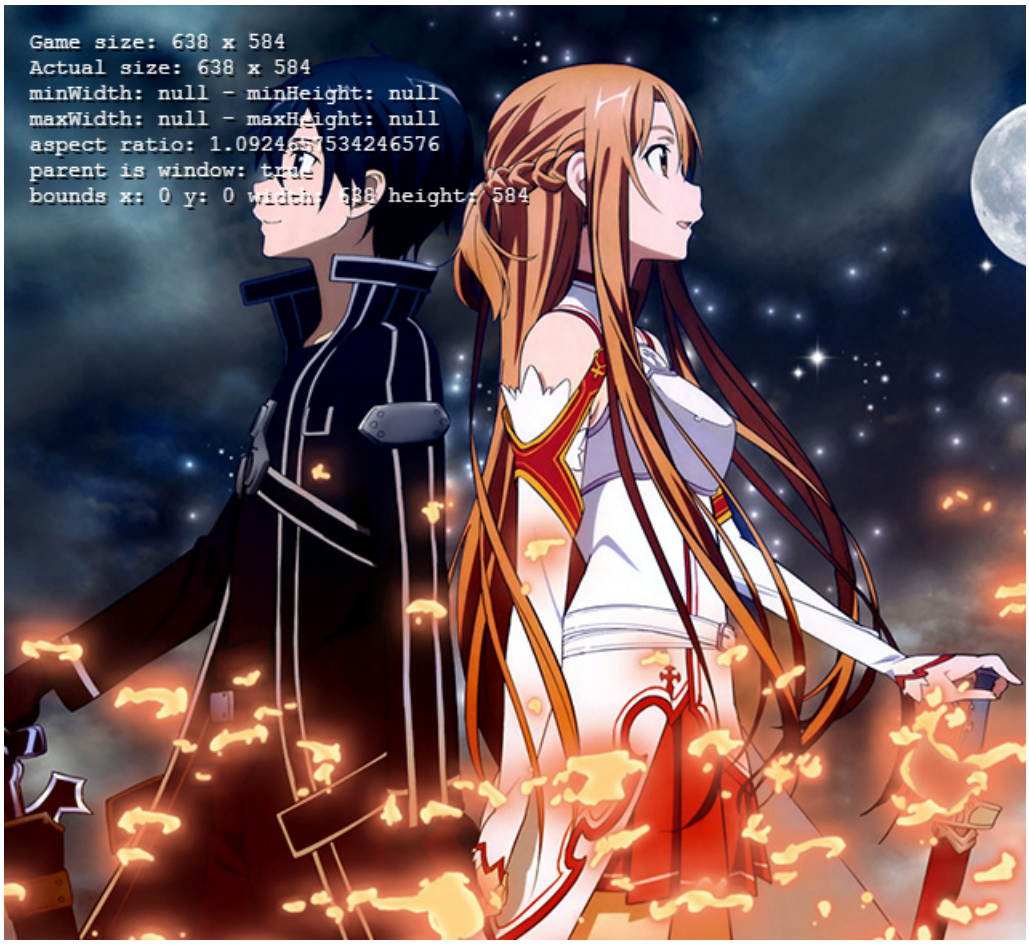
```
game.scale.scaleMode = Phaser.ScaleManager.RESIZE;
```

The RESIZE scale mode will create a Canvas element for your game that is the same size as the parent container. If the parent is `900 x 1200` then your game will be `900 x 1200` pixels in size.

In this scale mode Phaser will also track the parent's dimensions. If they change then Phaser will update your game size to fit the new parent dimensions.

The difference between this and SHOW_ALL is that in RESIZE the canvas element is resized so its physical properties match the parent. You game is never actually scaled, it has a 1:1 mapping to the display, it just has a bigger or smaller area to render to based on the parent.

You can see this in the screen shots below. In the first the browser size is `638 x 584`.



Resize 1

Here is the same game with the browser window resized to `855 x 584`:

Resize 2

As you can see in the shot above you can see a lot more of the background picture. As the browser resizes so more of the image becomes visible.

This is the scale mode you need for a truly responsive game. It's up to your game code to take advantage of the extra space and how it should use it, but Phaser does have some experimental features in the `FlexGrid` class to assist with this. This isn't covered in this guide as it's still in beta stage.

Resize callback

The RESIZE scale mode has one extra feature that the others don't. If your State has a `resize` method then it will be called whenever the parent changes size. It is passed two properties: width and height. You can then use this to adjust the position of any specially aligned game objects.

For example here we'll create two sprites:

```
kirito = game.add.sprite(0, game.world.bounds.bottom, 'kirito');
kirito.anchor.y = 1;

asuna = game.add.sprite(game.world.bounds.right, 0, 'asuna');
asuna.anchor.x = 1;
```

These have been positioned in the bottom-left and top-right of the game respectively. By using the anchor property we don't need to worry about subtracting the sprite's width from the game dimensions.

When the game starts it looks like this:

Resize 3

In the State we have a resize method:

```
function resize(width, height) {

        kirito.y = height;
        asuna.x = width;

}
```

As the browser is resized this function is called repeatedly. The parameters given can be used to adjust the location of the two sprites in the game, so they remain pinned to the bottom-left and top-right of the game, regardless of scale.

Here is what it looks like after the browser has been resized to be larger:

Resize 4

You can see that although the browser is now too large even for the background image, the sprites remain pinned to their respective positions.

This is the basis for all responsive game design.

Scale Mode 5 - USER_SCALE

```
game.scale.scaleMode = Phaser.ScaleManager.USER_SCALE;
```

The final scale mode is USER_SCALE. This mode was introduced in Phaser 2.2 and allows for custom dynamic scaling.

It works in combination with the `setUserScale` method:

```
game.scale.setUserScale(hScale, vScale, hTrim, vTrim);
```

The `hScale` and `vScale` parameters control how your game is scaled. A value of `1` means no scaling. `0.5` means half size, `2` double the size and so on, just as if you were scaling anything in Phaser.

The `hTrim` and `vTrim` are integers and define the amount that is removed from the horizontal or vertical size of the canvas after scaling.

The final scale is calculated as such:

```
canvas.width = (game.width * hScale) - hTrim
canvas.height = (game.height * vScale) - vTrim
```

You can call `setUserScale` at any point in your game, but most often you'll want to do it in a resize callback. There are two options for this, the `setResizeCallback` method, or using the `onSizeChange` signal:

`game.scale.setResizeCallback(callback, context);` `game.scale.onSizeChange.add(callback, context);`

The callback is invoked before sizing calculations. This is the appropriate place to call `setUserScale` for custom dynamic scaling.

The callback is supplied with two arguments `scale` and `parentBounds` where `scale` is a reference to the ScaleManager and `parentBounds`, a Phaser.Rectangle, is the size of the Parent element.

This callback:

- May be invoked even though the parent container or canvas sizes have not changed.
- Unlike `onSizeChange`, it runs *before* the canvas is guaranteed to be updated.
- Will be invoked from `preUpdate` *even when* the game is paused.

Game Alignment

The Scale Manager has the ability to vertically and horizonally align your game within its parent container:

```
game.scale.pageAlignVertically = true;
game.scale.pageAlignHorizontally = true;
```

This works with any scale mode. Depending on the dimensions of the parent, the game size and everything else on the page will depend what actually happens. However for simple alignment of the canvas within its parent it works well.

Here is a game aligned within a web page:

Align 1

The green area is a `<div>` with a fixed size of `1024 x 800` pixels and a 1px light green border. The game is `800 x 600` with both horizontal and vertical alignment enabled. As you can see it appears in the middle of the parent.

Should the parent not be big enough to contain the game then the visual result will depend entirely on what CSS you've set.

Scaling on Mobile

Scaling your game across mobile, tablet and desktop screens is notoriously difficult. There are advantages and disadvantages to each of the scale modes Phaser offers. Picking one will depend entirely on your game, the destination site on which it will live (if you even control that) and how you want to handle scaling.

Assuming you have decided upon your base resolution the next choice is on which scale mode to use.

For mobile you should only pick from SHOW_ALL, RESIZE or USER_SCALE. Please see the relevant sections of this guide for details about how each of them work.

Here is an `800 x 600` sized game scaled with SHOW_ALL on an iPad Mini running Safari:

Safari 1

In this scale mode your game will proportionally fit to the screen, but often with a letter-boxing effect to the sides. As the game scales up you may also see anti-aliasing issues in items that need clarity, such as text.

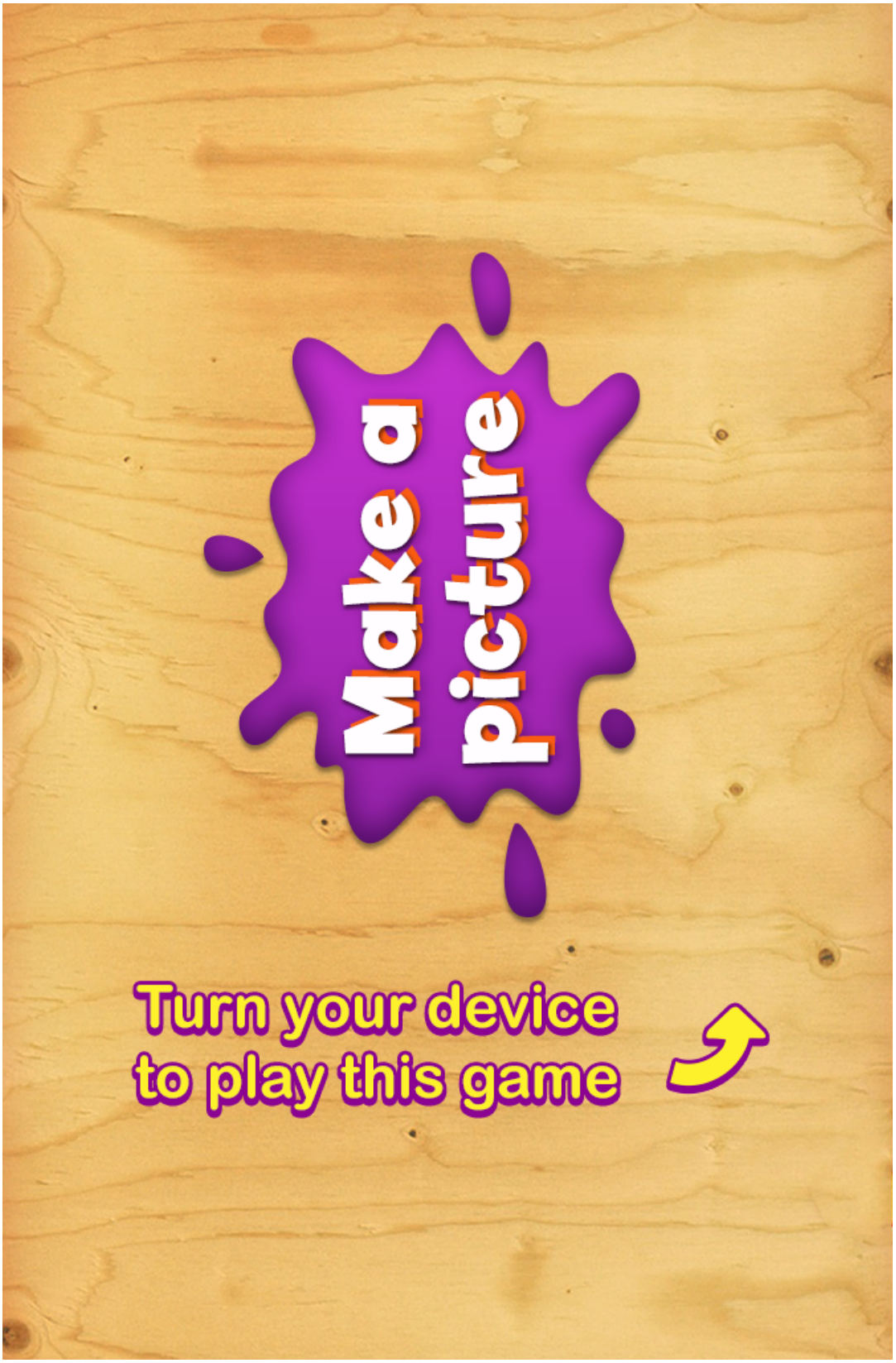
As you can see there are black borders on the left and right. Equally the text and pixel-art sprites are slightly blurred. However all of the game fits.

Forcing Orientation

You can't 'force' an orientation. Yet.

At the time of writing (December 2014) there is currently no way to consistently *force* a mobile browser into a specific orientation. Native apps *can* enforce an orientation however.

With that in mind the second best thing we can do is pause our games and display a screen saying "Please rotate your device". Once the player turns their device around the game can carry on again.



Rotate 1

Phaser has built-in support for this activated by the following method:

```
game.scale.forceOrientation(forceLandscape, forcePortrait);
```

Where the `forceLandscape` and `forcePortrait` parameters are booleans. If you wish your game to run in landscape you'd set `forceLandscape` to `true`. Naturally you need to know when the device is in the wrong orientation. For this Phaser provides three signals which you can listen to like this:

```
game.scale.enterIncorrectOrientation.add(callback, context);
game.scale.leaveIncorrectOrientation.add(callback, context);
game.scale.onOrientationChange.add(callback, context);
```

The first two are plain signals that don't pass any parameters to your callback. It's up to you what you do in these callbacks. You can pause the game, display an image, hide the game canvas and display a DOM element, whatever you need.

The third one, `onOrientationChange` has a more complex signature. It is dispatched when the orientation changes *or* the validity of the current orientation changes. Your callback will receive three parameters:

- `scale` - A reference to the Phaser Scale Manager.
- `prevOrientation` - A string containing the previous orientation.
- `wasIncorrect` - A boolean that is `true` if the previous orientation was last determined to be incorrect.

The final part of the puzzle is the ability to check if the game is in the right orientation or not. You may wish to do this when your game first starts up. You can check the boolean: `game.scale.incorrectOrientation`.

I'd suggest wrapping all of this in a conditional so it doesn't activate on desktop browsers, where you almost certainly don't want to enforce an orientation. Here is a complete set of calls for a typical mobile set-up:

```
if (game.device.desktop === false)
{
        game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
        game.scale.pageAlignHorizontally = true;
        game.scale.pageAlignVertically = true;
        game.scale.forceOrientation(true, false);
        game.scale.enterIncorrectOrientation.add(this.enterIncorrectOrientation, this);
        game.scale.leaveIncorrectOrientation.add(this.leaveIncorrectOrientation, this);
}
```

This will scale our game to fit the device, align it within the parent, force landscape orientation and set-up two callbacks to handle the change.

screen.orientation.lock

Although this section started by saying you cannot force a mobile browser to remain in a specific orientation - that is changing. It's available (or will be) via the Screen Orientation API[1].

Currently only Chrome for Android supports it. We can only cross our fingers that it lands in iOS eventually. To activate it you simply call:

`screen.orientation.lock('landscape')` or `screen.orientation.lock('portrait')`

I would suggest checking if the property is available before doing so however.

[1] http://caniuse.com/#feat=screen-orientation

Using the Fullscreen API

Phaser has support for the Fullscreen API[2]. This API is still experimental and not available in all browsers. But where it is available it allows your game to take over the display of the entire monitor or device's screen, with no browser chrome getting in the way.

Checking if the Fullscreen API is available

To check if fullscreen mode is available:

```
if (game.scale.compatibility.supportsFullScreen) {}
```

Entering Fullscreen

If supported then you can enable Fullscreen mode by calling:

```
game.scale.startFullScreen();
```

This method ++must++ be called from either a Pointer event or a Mouse event. I.e. the player has to actually click or touch something and only in that exact callback can you start fullscreen mode. This is a security measure to prevent web sites from automatically jumping into fullscreen mode without the users consent.

Here's an example calling fullscreen mode when clicking anywhere in the game:

```
function create() {

        game.input.onDown.add(goLarge, this);

}

function goLarge() {

        game.scale.startFullScreen();

}
```

[2] https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM/Using_full_screen_mode

Of course you could also link this to a Button, Sprite or anything else you like - so long as it receives a Pointer or Mouse event to activate it.

Just calling `startFullScreen()` doesn't guarantee the browser will actually enter fullscreen mode, it just makes the request to it. You should listen to the fullscreen related signals to handle what happens.

You can leave fullscreen mode by calling `game.scale.stopFullScreen`.

Fullscreen Scale Modes

Just as you define the scale mode of your game, so you can define the scale mode used when entering full screen. They don't have to be the same scale modes either. For example:

```
function init() {

        game.scale.scaleMode = Phaser.ScaleManager.NO_SCALE;
        game.scale.fullScreenScaleMode = Phaser.ScaleManager.SHOW_ALL;

}
```

This will tell your game to not scale at all when displayed in browser. But if it should enter fullscreen mode then it will jump to SHOW_ALL and proportionally resize to fill the screen. The same scale modes as are available to the game are available in fullscreen mode and their effects are the same.

Fullscreen Signals

There are 3 fullscreen related signals you can listen for:

onFullScreenInit

`game.scale.onFullScreenInit`

This signal is dispatched when fullscreen mode is ready to be initialized but before the fullscreen request is made to the browser API.

The callback for this signal is passed two parameters: `scale` - a reference to the Phaser Scale Manager and an object in the form `{targetElement: DOMElement}`.

If you have specified a fullscreen target element (via `game.scale.fullScreenTarget`) then the `targetElement` will contain a reference to this. You can then apply custom CSS styling or resets as required in advance of the fullscreen change.

If `targetElement` is *not* the same element as the FullScreen target then a new DOM element is created and after initialization the Canvas is moved onto it for the duration of the fullscreen mode. Once quit the Canvas is restored to its original DOM location.

onFullScreenChange

`game.scale.onFullScreenChange`

This signal is dispatched when the browser enters or leaves fullscreen mode, if supported. The callback to this signal is sent a single argument: `scale` which is a reference to the Phaser Scale Manager. You can use `game.scale.isFullScreen` within this callback to determine if your game is currently running in fullscreen mode or not.

onFullScreenError

`game.scale.onFullScreenError`

This signal is dispatched when the browser fails to enter fullscreen mode; or if the device does not support fullscreen mode and `startFullScreen` is invoked. The callback to this signal is sent a single argument: `scale` which is a reference to the Phaser Scale Manager.

Important Scale Manager Properties

There are a number of Scale Manager properties worth covering in detail. This is by no means a list of all of them. For that please see the API Documentation[3].

trackParentInterval

`game.scale.trackParentInterval`

This value is the maximum time (in ms) between dimension update checks of the parent. The default value is 2000. So every 2 seconds the Scale Manager checks the dimensions of the parent. If they have changed then it kicks into action, adjusting the game as defined.

You can see the effect of the 2 second delay if you quickly resize a Phaser game using a SHOW_ALL or RESIZE scale mode that uses the browser window as its parent. There will be brief periods where the game seems to be "catching up" with the parent.

You can lower the interval to whatever you like to make resizes smoother, but we'd strongly recommend keeping it at 2 seconds, or even forcing it higher if you know the platform/device your game is running on isn't going to change.

parentIsWindow

`game.scale.parentIsWindow`

A boolean value. It's set to `true` if the parent of your game is the browser window itself, otherwise this is set to `false`. Sometimes this is useful to know, especially in situations where you don't have direct control over where you game ends up being run.

scaleFactor

A Phaser.Point object that contains the current scale factor based on the game dimensions vs. the scaled dimensions. It also has an associated property `scaleFactorInversed` which contains the same value but inversed.

[3] http://docs.phaser.io

bounds

A Phaser.Rectangle object that contains the bounds of the scaled game. The `x/y` properties will match the offset of the canvas element. The `width` and `height` properties will contain the scaled width and height of the game. This is read only and shouldn't be modified directly.

Changes in Phaser 2.2

Before Phaser 2.2 a typical scaling set-up may look like this:

```
game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
game.scale.setMinMax(480, 260, 1024, 768);
game.scale.pageAlignHorizontally = true;
game.scale.pageAlignVertically = true;
game.scale.setScreenSize(true);
game.scale.refresh();
```

Here we're setting a minimum scaling size of `480 x 260` and a maximum of `1024 x 768`, aligning it and refreshing the Scale Manager.

Under Phaser 2.2 the calls to `setScreenSize` and `refresh` are no longer required, cutting the above down to:

```
game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;
game.scale.setMinMax(480, 260, 1024, 768);
game.scale.pageAlignHorizontally = true;
game.scale.pageAlignVertically = true;
```

Deprecated methods and properties

The following were all deprecated in Phaser 2.2 and should no longer be used:

```
Properties:

maxIterations
enterLandscape
enterPortrait
enterFullScreen
leaveFullScreen
fullScreenFailed
orientation

Methods:

checkResize
```

checkOrientation
setScreenSize
setSize
checkOrientationState

Source Code

Here are some complete common scaling mode examples. You can find the assets used in the Phaser Examples[4] git repository.

Optional Debug Info

If you like you can add the following code to each example. First add this to the end of the `create` function:

```
info = game.add.text(16, 16, ' ');
info.font = "Courier";
info.fontSize = 14;
info.fill = "#ffffff";
info.lineSpacing = 4;
info.setShadow(2, 2);
```

Then add this into the `update` function. This will allow you to visually see what's going on as the game resizes across device.

```
function update() {

        s = "Game size: " + game.width + " x " + game.height + "\n";
        s = s.concat("Actual size: " + game.scale.width + " x " + game.scale.height + "\n");
        s = s.concat("minWidth: " + game.scale.minWidth + " - minHeight: " + game.scale.minHeight\
 + "\n");
        s = s.concat("maxWidth: " + game.scale.maxWidth + " - maxHeight: " + game.scale.maxHeight\
 + "\n");
        s = s.concat("aspect ratio: " + game.scale.aspectRatio + "\n");
        s = s.concat("parent is window: " + game.scale.parentIsWindow + "\n");
        s = s.concat("bounds x: " + game.scale.bounds.x + " y: " + game.scale.bounds.y + " width:\
 " + game.scale.bounds.width + " height: " + game.scale.bounds.height + "\n");

        info.text = s;

}
```

[4] https://github.com/photonstorm/phaser-examples

Example 1: No Scaling, No Parent

This creates an `800 x 600` sized game added immediately to the DOM. No scaling will take place at all.

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>phaser scaling example</title>
                <script src="phaser.min.js" type="text/javascript"></script>
    </head>
    <body>
        <script type="text/javascript">

        var game = new Phaser.Game(800, 600, Phaser.CANVAS, '', { init: init, preload: pre\
load, create: create, update: update });

        function init() {

            game.scale.scaleMode = Phaser.ScaleManager.NO_SCALE;

        }

        function preload() {

            game.load.image('sao', 'assets/pics/sword_art_online.jpg');
            game.load.image('asuna', 'assets/sprites/asuna_by_vali233.png');
            game.load.image('kirito', 'assets/sprites/kirito_by_vali233.png');

        }

        var info;
        var kirito;
        var asuna;
        var s;

        function create() {

            var pic = game.add.sprite(game.world.centerX, game.world.centerY, 'sao');
            pic.anchor.set(0.5);

            kirito = game.add.sprite(0, 600, 'kirito');
            kirito.anchor.y = 1;
```

```
            asuna = game.add.sprite(800, 0, 'asuna');
            asuna.anchor.x = 1;

        }

        function update() {
        }

                </script>

    </body>
</html>
```

Example 2: Fit to Parent

This creates a game that is sized to be the same dimensions as the parent container.

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>phaser scaling example</title>
                <script src="phaser.min.js" type="text/javascript"></script>
    </head>
    <body>

        <div id="container" style="width: 800px; height: 600px"></div>

                <script type="text/javascript">

        var game = new Phaser.Game("100%", 100%", Phaser.CANVAS, 'container', { init: init\
, preload: preload, create: create, update: update });

        function init() {

            game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;

        }

        function preload() {

            game.load.image('sao', 'assets/pics/sword_art_online.jpg');
            game.load.image('asuna', 'assets/sprites/asuna_by_vali233.png');
            game.load.image('kirito', 'assets/sprites/kirito_by_vali233.png');
```

```
        }

        var info;
        var kirito;
        var asuna;
        var s;

        function create() {

            var pic = game.add.sprite(game.world.centerX, game.world.centerY, 'sao');
            pic.anchor.set(0.5);

            kirito = game.add.sprite(0, 600, 'kirito');
            kirito.anchor.y = 1;

            asuna = game.add.sprite(800, 0, 'asuna');
            asuna.anchor.x = 1;

                }

        function update() {
        }

                </script>

    </body>
</html>
```

Example 3: Scale to Browser Window

This creates a game that scales proportionally to fill the browser window. It assumes there is nothing else on the page. You should use CSS to specify the page padding, margins, etc.

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>phaser scaling example</title>
                <script src="phaser.min.js" type="text/javascript"></script>
    </head>
    <body>

                <script type="text/javascript">
```

```
        var game = new Phaser.Game(800, 600, Phaser.CANVAS, '', { init: init, preload: pre\
load, create: create, update: update });

        function init() {

            game.scale.scaleMode = Phaser.ScaleManager.SHOW_ALL;

        }

        function preload() {

            game.load.image('sao', 'assets/pics/sword_art_online.jpg');
            game.load.image('asuna', 'assets/sprites/asuna_by_vali233.png');
            game.load.image('kirito', 'assets/sprites/kirito_by_vali233.png');

        }

        var info;
        var kirito;
        var asuna;
        var s;

        function create() {

            var pic = game.add.sprite(game.world.centerX, game.world.centerY, 'sao');
            pic.anchor.set(0.5);

            kirito = game.add.sprite(0, 600, 'kirito');
            kirito.anchor.y = 1;

            asuna = game.add.sprite(800, 0, 'asuna');
            asuna.anchor.x = 1;

                }

        function update() {
        }

                </script>

    </body>
</html>
```

Example 4: Responsive Game

This creates a game that expands to fit the full available space of the parent (in this case the browser window) using the RESIZE scale mode. The resize method is used to position the sprites as the browser changes shape.

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>phaser scaling example</title>
                <script src="phaser.min.js" type="text/javascript"></script>
    </head>
    <body>

                <script type="text/javascript">

        var game = new Phaser.Game(800, 600, Phaser.CANVAS, '', { init: init, preload: pre\
load, create: create, update: update, resize: resize });

        function init() {

            game.scale.scaleMode = Phaser.ScaleManager.RESIZE;
            game.stage.backgroundColor = '#5b0c26';

        }

        function preload() {

            game.load.image('sao', 'assets/pics/sword_art_online.jpg');
            game.load.image('asuna', 'assets/sprites/asuna_by_vali233.png');
            game.load.image('kirito', 'assets/sprites/kirito_by_vali233.png');

        }

                var pic;
                var info;
        var kirito;
        var asuna;
        var s;

        function create() {

            pic = game.add.sprite(game.world.centerX, game.world.centerY, 'sao');
```

```
            pic.anchor.set(0.5);

            kirito = game.add.sprite(0, 600, 'kirito');
            kirito.anchor.y = 1;

            asuna = game.add.sprite(800, 0, 'asuna');
            asuna.anchor.x = 1;

                }

        function update() {
        }

        function resize(width, height) {

            pic.x = game.world.centerX;
            pic.y = game.world.centerY;

            kirito.y = height;
            asuna.x = width;

        }

                </script>

    </body>
</html>
```

Example 5: Full Screen

This example shows going into full screen mode.

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>phaser scaling example</title>
                <script src="phaser.min.js" type="text/javascript"></script>
    </head>
    <body>
        <script type="text/javascript">

        var game = new Phaser.Game(800, 600, Phaser.CANVAS, '', { init: init, preload: pre\
load, create: create, update: update });
```

```
function init() {

    game.scale.scaleMode = Phaser.ScaleManager.NO_SCALE;
    game.scale.fullScreenScaleMode = Phaser.ScaleManager.SHOW_ALL;

}

function preload() {

    game.load.image('sao', 'assets/pics/sword_art_online.jpg');
    game.load.image('asuna', 'assets/sprites/asuna_by_vali233.png');
    game.load.image('kirito', 'assets/sprites/kirito_by_vali233.png');

}

var info;
var kirito;
var asuna;
var s;

function create() {

    var pic = game.add.sprite(game.world.centerX, game.world.centerY, 'sao');
    pic.anchor.set(0.5);

    kirito = game.add.sprite(0, 600, 'kirito');
    kirito.anchor.y = 1;

    asuna = game.add.sprite(800, 0, 'asuna');
    asuna.anchor.x = 1;

    game.input.onDown.add(goLarge, this);

}

function goLarge() {

    game.scale.startFullScreen();

}

function update() {

}
```

```
                </script>

    </body>
</html>
```

Credits

Version 1.0 published 3rd December 2014. Version 1.1 published 4th December 2014.

Written by Richard Davey[5]

Cover image by QuasiXi[6]

Sprites by valiryn[7]

[5] http://www.photonstorm.com
[6] http://quasixi.deviantart.com
[7] http://valiryn.deviantart.com