Introducing Phaser
We're getting a tutor

- Sessions are tentatively scheduled for Wednesday afternoons and Friday mornings
  - Starting time depends on when Summer Session gets back to me, so stay tuned for more details

We may be moving to a closer classroom in the future

- But I want to make sure to give you advance notice before it happens
  - At least 24 hours in advance
  - Watch the announcements on Canvas
Get out your laptop!
(if you have one)
Framework

A “foundation” or “support structure” around which you build an application.

Frameworks provide a relatively static, generic structure on and in which you build something specific and unique.

You build with a framework

- Phaser
- Three.js
- P5.js
- ImpactJS
- LÖVE
- MonoGame

Engine

Full-feature toolsets that handle core logic and implementation details upon which you build an application.

Engines provide a self-contained but externally-controllable piece of code that encapsulates powerful logic designed to perform a specific kind of work.

You build on an engine

- Unity
- Unreal
- CRYENGINE
- Construct2
- GameMaker
- Godot
- Frostbite
How do you choose a framework, anyway?
First, get your head straight

It's not the tools
I like this example...

Carlos Ruiz @Kairuiz_ · 20 Apr 2017
So let’s say I was doing environment sketches just for fun
I like this example...
Some Helpful Questions

➔ What works best for you and/or your team?
➔ What fits your skill level?
➔ What workflow makes sense to you?
➔ What creates the least amount of friction?
➔ What fits your budget?
➔ What are your target platforms?
➔ What scales to multiple games or genres?
➔ Can I get help when I need it?
➔ What quality of help will I receive?
➔ What compromises am I comfortable making?
➔ Will I be able to achieve my target goal?
Phaser *(let's get started!)*
1. **DOWNLOAD PHASER**

2. **GETTING STARTED**

You can code your Phaser games in JavaScript or TypeScript and we've Getting Started guides for both. If you're coming from Flash / AS3 then we'd recommend giving TypeScript a go, otherwise stick with JavaScript.

- JavaScript Guide
- TypeScript Guide
- Cloud-based IDE Guide
Why yes, we are looking for Phaser CE!
Please Use

Phaser CE 2.13.1

(As of June 2019)
Phaser CE 2.12.0

Version: 2.12.0
Released: 6th February 2019
Previous Version: 2.11.1

Download

Download this version of Phaser CE from npm. Need more details? Follow this tutorial

```
npm install phaser-ce@2.12.0
```

Change Log

Version 2.12.0 - 2 October 2018

If you’re using the `leadAnchors` argument in the PhaserCreature constructor, you’ll have to change your code.

New Features / API Changes

- BitmapText has a new property `letterSpacing` which accepts a positive or negative number to add or reduce spacing between characters.
- Camera now has new properties `centerX` and `centerY` to get the center of the camera’s current viewport.
Minification is a bandwidth optimization technique that reduces the size of code transmitted over the web.
1. **DOWNLOAD PHASER**

2. **GETTING STARTED**

   You can code your Phaser games in JavaScript or TypeScript and we’ve Getting Started guides for both. If you’re coming from Flash / AS3 then we’d recommend giving TypeScript a go, otherwise stick with JavaScript.

   - JavaScript Guide
   - TypeScript Guide
   - Cloud-based IDE Guide
Getting Started with Phaser

Choose your own adventure

By Richard Davey on 14th February 2018  @photonstorm

Please select the guide for the version of Phaser you are using:

Getting Started with Phaser 3

or

Getting Started with Phaser 2 / Phaser CE
Part 1 - Introduction

By Richard Davey on 14th February 2018 @photonstorm

This guide is for Phaser 2 / Phaser CE. If you’re using Phaser 3 then please go here.

In this tutorial we’re going to cover setting-up a development environment with which you can build your Phaser games. This will include running a local web server, picking an IDE, getting the latest version of Phaser and checking it all works together.

If you trust us that you do need a local web server for development, then you can skip the explanation below and head directly to part 2.

If you’d like to know the reasoning why, please read on...

A web server? But I want to make games!

"Why do I need a local web server? Can’t I just drag the html files onto my browser?"

A. SANE, DEVELOPER

Not these days, no. I appreciate that it's a bit confusing, even contradictory at times, but it all boils down to browser security. If you're making a static html web page then you can happily drag this file into your browser and see the end results. You can also “Save As” entire web pages locally and re-open them with most the contents intact. If both of these things work why can’t you drag an HTML5 game into a browser and run it?

It's to do with the protocol used to access the files. When you request anything over the web you're using http, and the server level security is enough to ensure you can only access files you're meant to. But when you drag a file in it's loaded via the local file system (technically file://) and that is massively restricted, for obvious reasons. Under file:// there's no concept of domains, no server level security, just a raw file system.
A basic Phaser "bootstrap" folder
index.html is our "central hub"
(and the web page where your game will appear)
framework

phaser

projectName

[local server]

index.html

js
main.js

assets

audio

img

js

styles.js

css

phaser.min.js
My naming convention is not the only way, but merely a way

It is OK to (sensibly) rename things

It matters more to have any naming convention than any of the details of that naming convention
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>[Your Game Title Here]</title>
  <script type="text/javascript" src="../framework/phaser.min.js"></script>
  <script type="text/javascript" src="js/main.js"></script>
  <link rel="stylesheet" type="text/css" href="css/styles.css" />
</head>
<body>
  <!-- We don't need anything in the <body> -->
  <!-- Phaser will make the <canvas> element for us 👍 -->
</body>
</html>

Be careful with file paths
File paths:

From index.html:

To access a file/folder relative to our location, type its name: `src="css"`

To access a file/folder within that relative path, add a slash: `src="css/style.css"`

To access a parent file/folder relative to our location, add ",../": `src="../framework/phaser.min.js"

Understanding these relationships helps keep everything organized.
Our Local Server
A web server? But I want to make games!

"Why do I need a local web server? Can I just drag the html files onto my browser?"

A. SANE, DEVELOPER

Not these days, no. I appreciate that it’s a bit confusing, even contradictory at times, but it all boils down to browser security. If you’re making a static HTML web page then you can happily drag this file into your browser and see the end results. You can also “Save As” entire web pages locally and re-open them with most the contents intact. If both of these things work why can’t you drag an HTML5 game into a browser and run it?

It’s to do with the protocol used to access the files. When you request anything over the web you’re using http, and the server level security is enough to ensure you can only access files you’re meant to. But when you drag a file in it’s loaded via the local file system (technically file://) and that is massively restricted, for obvious reasons. Under file:// there’s no concept of domains, no server level security, just a raw file system.

Ask yourself this: do you really want JavaScript to have the ability to load files from anywhere on your local file system?

Your immediate answer should of course be “not in a million years!”. If JavaScript had free reign while operating under file:// there would be nothing stopping it loading pretty much any file, and sending it off who knows where.

Because this is so dangerous browsers lock themselves down tighter than Alcatraz when running under file://. Every single page becomes treated as a unique local domain. That is why “Save Web page” works, to a degree. It opens under the same cross-site restrictions as if they were on a live server.

DO YOU REALLY WANT JAVASCRIPT TO HAVE THE ABILITY TO LOAD FILES FROM ANYWHERE IN YOUR FILE SYSTEM?
What's the whole point of “localhost”, hosts and ports at all?

10 Answers

In computer networking, localhost (meaning "this computer") is the standard hostname given to the address of the loopback network interface.

41

Localhost always translates to the loopback IP address 127.0.0.1 in IPv4.

It is also used instead of the hostname of a computer. For example, directing a web browser installed on a system running an HTTP server to http://localhost will display the home page of the local web site.


The :80 part is the TCP port. You can consider these ports as communications endpoints on a particular IP address (in the case of localhost - 127.0.0.1). The [IANA](https://www.iana.org) is responsible for maintaining the official assignments of standard port numbers for specific services. Port 80 happens to be the standard port for [HTTP](https://en.wikipedia.org/wiki/HTTP).
On Mac: In Terminal, navigate to your phaser folder and type:

**python 3:** `python3 -m http.server`

**python 2:** `python -m SimpleHTTPServer 8000`

Then, in your browser, go to: `http://localhost:8000`
On Windows: use cmd.exe or PowerShell

**python 2:** C:\pathToIndexfile\python -m SimpleHTTPServer  
**python 3:** C:\pathToIndexfile\py -m http.server

Then, in your browser, go to: http://localhost:8000
Which version of Python do I have?

```
Last login: Wed Jun 27 17:50:50 on ttys000
NateBook-Pro:~ nathanaltice$ python --version
Python 2.7.11
NateBook-Pro:~ nathanaltice$ which -a python python2 python2.7 python3 python3.6
/usr/local/bin/python
/usr/bin/python
/usr/local/bin/python2
/usr/local/bin/python2.7
/usr/bin/python2.7
NateBook-Pro:~ nathanaltice$ 
```
The easiest way to run a local server is with Python

Python 2:
```
python -m SimpleHTTPServer
```

Python 3:
```
python -m http.server
```

If you don't have Python on your machine, install it.

https://www.python.org/downloads/
https://www.anaconda.com/distribution/

The easiest way to run a local server is with Python
Do **not** double-click the `index.html` file.

You need to open it with `http://` instead of `file://`. 
Use the local server!

Type localhost:portnumber into the browser address bar
protocol://subdomain.127.0.0.1:port-number/path?parameters
Though it is a fully functioning web server

https://ipchicken.com/
Now you can navigate to index.html
We’re in the right path

In the console, Phaser reports that it is running

And we even have the canvas added for us!
sad file means something went wrong
Please *don't* use XAMP, MAMP, etc.

The Phaser tutorial mentions them as options, but they're overkill. Use the Python server.
// let's keep our code tidy with strict mode 👊
"use strict";

// initialize game object
var game = new Phaser.Game(640, 880, Phaser.AUTO, '', { preload: preload, create: create, update: update });

function preload() {
    // preload our assets
}

function create() {
    // place your assets
}

function update() {
    // run game loop
}
Any plain-text file editor will work. (Remember what I said about tools?)

But I'm going to edit the file directly in the browser.
Let's check out main.js

Define new game object instance

```javascript
// initialize game object
var game = new Phaser.Game(640, 880, Phaser.AUTO, '', { preload: preload, create: create, update: update });

function preload() {
    // preload our assets
}

function create() {
    // place your assets
}

function update() {
    // run game loop
}
```
Objects are arbitrary collections of properties.

A property name can be any valid JS string.
A Phaser.Game object is a special type of object that Phaser provides

```javascript
// initialize game object
var game = new Phaser.Game(640, 880, Phaser.AUTO, '', {
  preload: preload, create: create, update: update });

function preload() {
  // preload our assets
}

function create() {
  // place your assets
}

function update() {
  // run game loop
}
```
Phaser CE Documentation:
https://photonstorm.github.io/phaser-ce/index.html
(bookmark this!)
Calling Phaser.Game object constructor

```
// initialize game object
var game = new Phaser.Game(640, 880, Phaser.AUTO, '', { preload: preload, create: create, update: update });

function preload() {
    // preload our assets
}

function create() {
    // place your assets
}

function update() {
    // run game loop
}
```
// initialize game object
var game = new Phaser.Game(640, 880, Phaser.AUTO, '', { preload: preload, create: create, update: update });

function preload() {
    // preload our assets
}

function create() {
    // place your assets
}

function update() {
    // run game loop
}
The browser console lets us look inside the Phaser object!

To do so, type the object name (Phaser) into the console text field and press Enter/Return.

Being able to access all this information from the browser console is super-powerful!
To use this superpower, let's type in the game instance that we just created.

(Anyone want to guess what will happen?)

Remember this when you are debugging! You can use the console to see what is going on with any variable.
// initialize game object
var game = new Phaser.Game(640, 880, Phaser.AUTO, '', { preload: preload, create: create, update: update });

function preload() {
    // preload our assets
}

function create() {
    // place your assets
}

function update() {
    // run game loop
}
// initialize game object
var game = new Phaser.Game(640, 880, Phaser.AUTO, '', { preload: preload, create: create, update: update });

function preload() {
  // preload our assets
}

function create() {
  // place your assets
}

function update() {
  // run game loop
}
The Canvas API provides a means for drawing graphics via JavaScript and the HTML `<canvas>` element. Among other things, it can be used for animation, game graphics, data visualization, photo manipulation, and real-time video processing.

The Canvas API largely focuses on 2D graphics. The WebGL API, which also uses the `<canvas>` element, draws hardware-accelerated 2D and 3D graphics.

### Basic example

This simple example draws a green rectangle onto a canvas.

```html
<canvas id="canvas"></canvas>
```

```javascript
const canvas = document.getElementById('canvas');
const ctx = canvas.getContext('2d');
ctx.fillStyle = 'green';
ctx.fillRect(10, 10, 150, 180);
```
```html
<!DOCTYPE html>
<html>
<head>
  <title></title>
  <style type="text/css">
    #canvas {
      border: 1px solid black;
    }
  </style>
</head>
<body>
  <canvas id="canvas" width="800" height="600"></canvas>
<script type="text/javascript">
  // wait for window to load
  window.onload = function() {
    var canvas = document.getElementById('canvas');
    var ctx = canvas.getContext('2d');
    ctx.fillStyle = '#facade';
    ctx.fillRect(10, 10, 200, 200);
  }
</script>
</body>
</html>
```

context: 2D (Canvas API)
Thanks, Phaser!
// initialize game object
var game = new Phaser.Game(640, 880, Phaser.AUTO, '', { preload: preload, create: create, update: update });

function preload() {
    // preload our assets
}

function create() {
    // place your assets
}

function update() {
    // run game loop
}
index.html

<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>[Your Game Title Here]</title>
  <script type="text/javascript" src="../framework/phaser.min.js"></script>
  <script type="text/javascript" src="js/main.js"></script>
  <link rel="stylesheet" type="text/css" href="css/styles.css" />
</head>
<body>
  <!-- We don't need anything in the <body> -->
  <!-- Phaser will make the <canvas> element for us -->
</body>
</html>

Wait a minute, there's nothing here!
double thanks Phaser!
// initialize game object
var game = new Phaser.Game(640, 880, Phaser.AUTO, '', { preload: preload, create: create, update: update });

function preload() {
    // preload our assets
}

function create() {
    // place your assets
}

function update() {
    // run game loop
}
// initialize game object
var game = new Phaser.Game(640, 880, Phaser.AUTO, '', { preload: preload, create: create, update: update });

function preload() {
    // preload our assets
}

function create() {
    // place your assets
}

function update() {
    // run game loop
}
// initialize game object
var game = new Phaser.Game(640, 880, Phaser.AUTO, '', { preload: preload, create: create, update: update });

function preload() {
    // preload our assets
}

function create() {
    // place your assets
}

function update() {
    // run game loop
}
// initialize game object
var game = new Phaser.Game(640, 880, Phaser.AUTO, '', { preload: preload, create: create, update: update });

function preload() {
    // preload our assets
}

function create() {
    // place your assets
}

function update() {
    // run game loop
}

...then your primary game loop begins!
// initialize game object
var game = new Phaser.Game(640, 880, Phaser.AUTO, '', { preload: preload, create: create, update: update });

function preload() {
    // preload our assets
}

function create() {
    // place your assets
}

function update() {
    // run game loop
}

This is a "single-state" Phaser game with three state methods
Let's make some changes!
But first, some assets...
Open-source emojis for designers, developers and everyone else!
You can reference the images by their `unicode` value
// let's keep our code tidy with strict mode 👊
"use strict";

// initialize game variables
var velocity = 4;
var emoji;

// initialize game object
var game = new Phaser.Game(640, 880, Phaser.AUTO, '', { preload: preload, create: create, update: update });

function preload() {
    // preload our assets
    game.stage.backgroundColor = '#EEE';
    game.load.image('emoji_img', 'assets/img/openmoji72x72/1F469-200D-1F3A8.png');
}

function create() {
    // place your assets
    emoji = game.add.sprite(0,0, 'emoji_img');
}

function update() {
    // run game loop
    emoji.x += velocity;
    if (emoji.x >= game.world.width - emoji.width || emoji.x <= 0) {
        velocity = -velocity;
    }
}
Your first Phaser game

And also your first programming assignment
Follow the Phaser 2 tutorial...

http://phaser.io/tutorials/making-your-first-phaser-2-game

But use the organization that we talked about in class.

And skip setting up a web server.

Your objective is to get the Phaser environment running, get the tutorial game working, and make some small tweaks.
Grading Criteria

Organization (3 points)
➔ Comment the code
➔ Assets in proper folder
➔ Code in the right place
➔ Game runs from localhost with no errors

Submit your project to Canvas as a zip with the filename
LastNameFirstNameTutorial.zip

Code Changes (7 points)
➔ Vertical resolution
➔ Add platforms
➔ New collectable object
➔ Add enemies