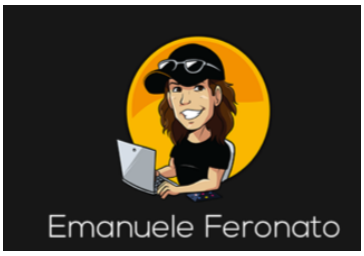
Week 17

HTML 5 Game Development using JavaScript and Gaming Development Platform - This week we will continue learning about Phasor Game Development Platform

01/24/2024

Author:



What is a cross-platform game and why should I make cross-platform games?

With the great interest in mobile games, capable of running on modern portable devices such as smart phones and tablets, there's a lot of talking about “cross- platform” term these days.
Although we are talking about modern devices, the cross-platform concept comes from an older computer age, before smart phones and tablets, probably before any kind of portable device smaller than a mid-sized suitcase.

In its original context, cross-platform is an attribute conferred to computer software or computing methods and concepts that are implemented and inter- operate on multiple computer platforms. Such software and methods are also called “platform independent”. To tell you the short story, a cross-platform software will run on any platform without special adaptation, or with a minimum special adaptation.

A good example of a cross-platform language is Java: a compiled Java program runs on all platforms under Java Virtual Machine, which you can find in all major operating systems, including Windows, Mac OS and Linux.

What is HTML5?

HTML5 is the latest version of HTML, or Hypertext Markup Language, the code web pages are mostly built with.
We can basically split each web page in three kinds of code: HTML, which provides the structure; Cascading Style Sheets (CSS), which take care of presentation; and JavaScript, which makes things happen.
Unlike other tools like Flash or Java, HTML5 works without requiring any additional software like browser plugins or extensions, and it's capable of doing everything from animation to apps, can play music or movies, and can also be used to build incredibly complicated applications – or in this case, games – that run on almost any browser.
Moreover, HTML5 isn't proprietary, so you don't need to pay fees, subscriptions or royalties to use it. It's also cross-platform, which means it doesn't care whether you're using a tablet or a smart phone, or a laptop, or a smart TV: if your device browser supports HTML5 – and most of them do – your app will work flawlessly.
At this time you may think you have to learn to code HTML5 to build

a HTML5 game.

That's not true. Actually, you won't be writing more than a couple of lines of HTML5, since your game will be written in JavaScript.

JavaScript is a scripting language that allows you to create dynamically updating content, control multimedia, animate images, and pretty much everything you need to build a game, especially if combined with JavaScript frameworks created with game development in mind.

What is Phaser?

Phaser is a free HTML5 game framework which aims to help developers make powerful, cross browser HTML5 games really quickly using JavaScript.

JavaScript, being a familiar and intuitive language, is one of the most common languages so even if you didn't already developed JavaScript applications you will find a lot of books and tutorials around the web to get you started.

Anyway, you don't need anything else than this book to build your first complete game, so let's start having some fun.

Ok, I am lost. HTML5, CSS, JavaScript,

Phaser... too much stuff

Actually, it's way easier than you may think. Your final game will be a HTML5 game, but actually the only HTML5 element you will be using in your game is the <canvas> element, which is only a container for the graphics.

You will use JavaScript to draw and update the graphics, as well as to define the game design.

Phaser is the JavaScript framework which you will use to handle sprites, sound effects, graphic effects, explosions, screen updates and basically everything capable of making your game look nice, move smooth and sound good.

CSS will be used for minor adjustments on the page which will host your game, mostly to define where <canvas> element should be placed.

Choosing a free text editor

In order to start coding games, you'll first need a software to write code. There are a lot of free offers:

Brackets (http://brackets.io/) is an open-source editor written in HTML, CSS, and JavaScript with a primary focus on web development and available for OS X, Windows, and Linux machines.

Other free software you may need

Games basically are a collection of images and sounds which are moved and played accordingly to player actions and scripting logic, so during the development of the game you will also need to create and edit both images and sounds.

Audacity (https://www.audacityteam.org/) is a great free software to work with sounds available for OS X, Windows and Linux.

oceanaudio (http://www.ocenaudio.com/) is a cross-platform, easy to use, fast and functional audio editor. It is the ideal software for people who need to edit and analyze audio files without complications.

GIMP (http://www.gimp.org/) is a powerful image editor available for OS X, Windows and Linux..

Krita (https://krita.org/en/) is a great open source painting software available for OS X, Windows and Linux. You may find it really useful in the creation of textures, concept art, illustrations and backgrounds.

Choosing a free web server

To test your Phaser games, and more in general to test most web applications, you will need to install a web server on your computer to override browser security limits when running your project locally.

WAMP (http://

WAMP (http://www.wampserver.com/) is a complete Windows web development environment which allows you to create web applications with Apache2, PHP and a MySQL database.
MAMP (https://www.mamp.info/en/) runs on Mac and Windows, works pretty much the same way WAMP does, and also features a paid PRO version with more options.
Fenix Web Server (http://fenixwebserver.com/) is the web server I currently use because it's really simple, with no extra stuff, and open source. It's available for both Windows and Mac.

Note: The latest Mac Operating Systems do not allow a 32 bit program to work, so we need to use Terminal -
(Terminal Commands : ls = "List Structures" , cd = "Change Dir")

--> use terminal, ls (list structures files), cd change dir, get to specific (dir) with the index.html cd .. (to go back 1 dir)

$ python -m SimpleHTTPServer 8001

->> then open browser to localhost:8001

REALLY choosing a web server, …

I know at this time most of you may think “come on, it's just JavaScript, what's this server stuff, I quit!”.
This is the same thing I said when I first had to install and configure a web server just to run a JavaScript page.
Let me explain why you should really choose a web server, rather than quit reading: browsers do not simply allow you to properly display web pages and HTML5 games. They also take care of your security.
When you load a page locally in your browser, you won't have problems as long as it's just a static HTML web page.

problems as long as it's just a static HTML web page.
But when you launch more complex scripts which load and handle resources from your hard disk such as images, audio files and every other kind of data, to prevent malicious scripts to access to virtually any file on your computer, browsers have a series of security measures which stop files to be accessed and – unfortunately but necessarily – this causes your games not to work.
The most frequent error you will get if you run a Phaser game directly in your browser is something like “Cross origin requests are only supported for protocol schemes: http, data, chrome, chrome-extension, https.”
With a web server, browsers will know they are running in a small, safe environment where only some files – the ones you placed in a given project folder – can be accessed, and they will give your scripts green light to work properly.
Believe me, it's necessary and way easier than you may think.

The structure of your first Phaser project

As said, every HTML5 game is a web page with some magic in it, so we are going to create a new folder which will contain the entire game.
The folder will contain the web page itself, the Phaser framework we just downloaded, and every other game file we will need as soon as the game gets more complex and needs more resources like sprites, backgrounds, sound effects and so

more resources like sprites, backgrounds, sound effects and so on.

Done with the creation of the folder?

Create an index.html file which will be the web page you will call to launch the game, and you'll have all you need to start writing the first lines of code of your Phaser game.

If you followed all instructions, the folder containing your game should look like this:

Now let's edit index.html:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript" src="phaser.min.js"></script>
        <script type="text/javascript" src="game.js"></script>
    </head>
    <body>
    </body>
</html>
```

As you can see, it's just an empty web page calling two JavaScript files:

`phaser.min.js` is the file we just downloaded, and `game.js` will contain our game script.

Now create a new file, call it `game.js` and write this code:

```
window.onload = function() {
    var game = new Phaser.Game();
}
```

Congratulations, you just created your first Phaser script.

And this is how your game folder should look like now:

This is in the (subfolder)
..\Week_11\example_0

We will write the entire game code into game.js file.
Writing the whole game code into a single file may seem a malpractice, because the code would be way more understandable if organized in more files, such as a file for the configuration, a file handling player input, and so on.

The drawback is most game sponsors want the entire game to fit in just one file and probably won't accept the submission of your game if your code is scattered through a dozen files.

Since finding a sponsor – which means finding someone willing to pay for our hard work – is a top priority, that's why we are going to write the entire code in one single file.
Back to our code, let's see what these lines mean:

The onload event for the window fires after all objects in the DOM hierarchy (images, scripts, frames, and so on) have finished loading and the document object has been built up.

It checks for everything to be loaded, then calls the callback function.

> A JavaScript **function** is a block of code designed to perform a particular task, executed when "something" invokes it. When you make something to execute a function, we say you **call** the function.
>
> A function can contain any number of lines, and we use **curly brackets**, { and } to define the start and the end of a function, or more in general, the start and the end of a block of code.

Inside the function, there's only one line of code which initializes game variable as

a new `Phaser.Game` instance:

```
var game = new Phaser.Game();
```

Translating what we done in English, it means "once the entire document has been loaded, create a new `Game` object and assign it to a variable called `game`."

> A JavaScript **variable** is a container for storing data values. When you create a variable we say you **declare** a variable, using the `var` keyword. You assign a value to a variable with `=` operator.
>
> A JavaScript **object** is a particular kind of variable which can contain many values.

Well, this is our first Phaser game, so let's run it and see what happens.

Index.html

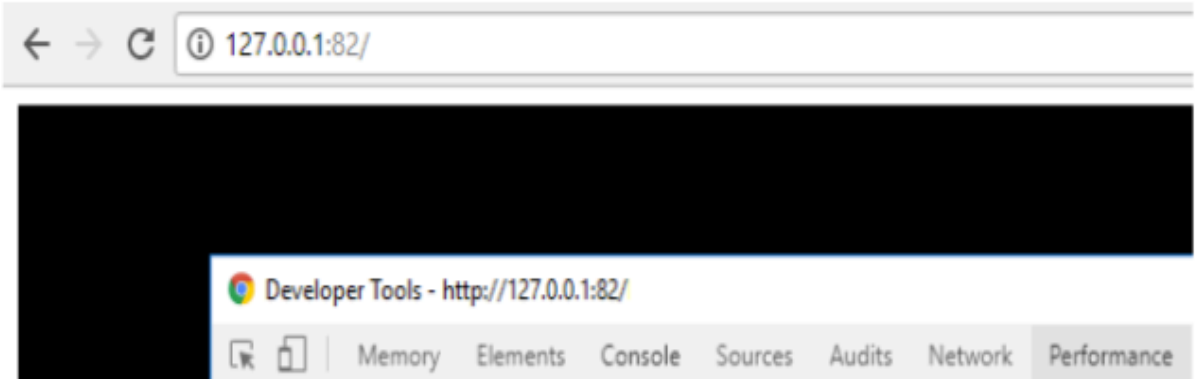
```
<!DOCTYPE html>
<html>
  <head>
    <title>(Your Name) Awesome Game</title>
    <meta name = "viewport" content = "width = device-width, initial-scale = 1.0, maximum-scale = 1.0, minimum-scale = 1.0, user-scalable = 0, minimal-ui" />
    <style type = "text/css">
      body{
        padding: 0px;
```

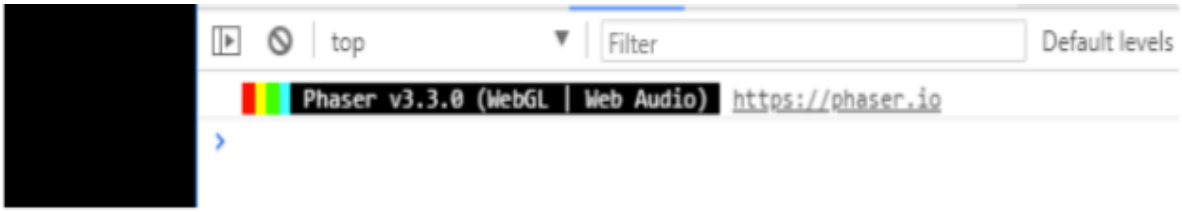
```
            margin: 0px;
        }
        canvas{
            display:block;
            margin: 0;
            position: absolute;
            top: 50%;
            left: 50%;
            transform: translate(-50%, -50%);
        }
    </style>
    <script type="text/javascript" src="phaser.min.js"></script>
    <script type="text/javascript" src="game.js"></script>
  </head>
  <body>
  </body>
</html>
```

Running your Phaser game

To run the game on your local web server, first setup the web server of your choice according to your preferences and the folder where you saved the game.

Then simply point your browser to `index.html` file in your game folder which in most cases will be something like http://localhost/yourphasergamefolder/ and this is what you should get if running it on your Google Chrome browser:





The black square on the left is your game.

By default, Phaser creates a game on a black background.

You can also see which Phaser version you are using by opening the Console window pressing F12.

Vehical.firecannon(How_High, how_far_LeftRight)

Creating a Phaser Game configuration object

There are several options which can be configured when you are about to create a `Phaser.Game` instance, and you will need to place them inside an object which is passed as argument when you create the game.

> A variable passed as parameter inside a function is called **argument**.
>
> Arguments are passed to a function by placing them between the parentheses.
>
> Functions can have any number of arguments, separated by commas.

Add these lines to `game.js`:

```
window.onload = function(){
    var gameConfig = {
        width: 480,
```

```
        height: 640,
        backgroundColor: 0xff0000
    }
    var game = new Phaser.Game(gameConfig);
}
```

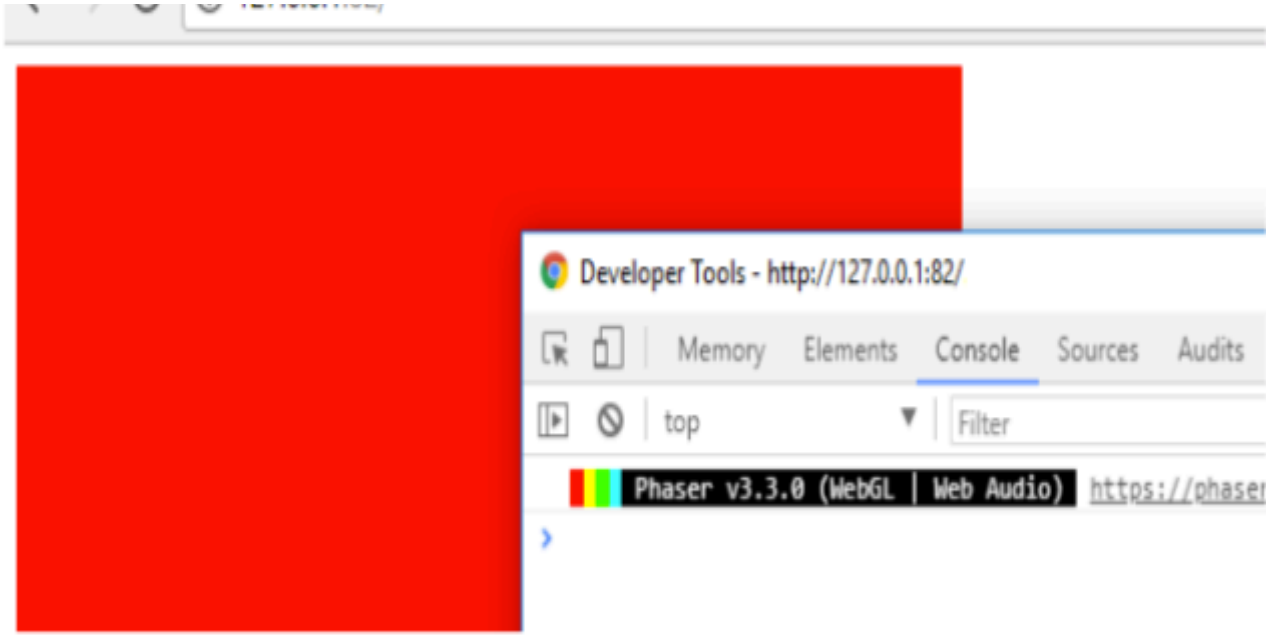
We created a new variable called `gameConfig` which is an object containing some game settings such as game width, height and background color.

The object is then passed as argument to `Phaser.Game` instance.

Now run the game, inspect the console and see what changed:

```
<html>
    <head>
        <script type="text/javascript" src="phaser.min.js"></script>
        <script type="text/javascript" src="game.js"></script>
    </head>
    <body style="overflow: hidden;">
        <canvas width="480" height="640"></canvas>
    </body>
</html>
```

Now the canvas has the same size we specified in `gameConfig` object and the background color of the game is red, and this is what you should see:



Anyway, having the game in the top left corner of the page does not make it look that good. Let's center it in the page.

Adjusting CSS to make the game run in the center of the page

To make the game run in the center of the page, there's nothing of JavaScript involved.

We'll handle `<canvas>` element just like any ordinary web page element we want to center in its container.

Add these lines to `index.html`:

```
<!DOCTYPE html>
```

```
<html>
    <head>
        <title>My Awesome Game</title>
        <meta name = "viewport" content = "width = device-width, initial-scale =
              1.0, maximum-scale = 1.0, minimum-scale = 1.0, user-scalable = 0,
              minimal-ui" />
        <style type = "text/css">
            body{
                padding: 0px;
                margin: 0px;
            }
            canvas{
                display:block;
                margin: 0;
                position: absolute;
                top: 50%;
                left: 50%;
                transform: translate(-50%, -50%);
            }
        </style>
        <script type="text/javascript" src="phaser.min.js"></script>
        <script type="text/javascript" src="game.js"></script>
    </head>
    <body>
    </body>
</html>
```