Phaser - Introduction

Lesson Plan

Overview & Purpose

Introduction to game development using the HTML 5 Phaser Framework.

Objectives

1. Learn how to set up the basic HTML code structure for a single screen Phaser game.
2. Learn how to instantiate the Phaser Game Engine using the Phaser.Game class and a configuration object.
3. Learn about Phaser.Scene class and its main functions: preload, create and update.

Prerequisites

1. Students should be familiar with HTML, CSS and JavaScript basics.

2. Students should be familiar with Computer Science core concepts that include: variables, conditionals, loops and functions.
3. Students should have some knowledge of advanced concepts like Classes and Objects in JavaScript.

Materials Needed

1. Machine [MacOS, Windows, Linux]
2. Editor [Sublime Text, MS Visual Studio]
3. Server [http-server]

Verification

Steps to check for student understanding

1. Students should be able to answer what Phaser is and list the core functionalities it offers.

2. Students should have successfully built the basic HTML code structure for Phaser.

3. Students should understand the purpose of configuration object and the preload, create and update functions belonging to Phaser.Scene class.

Introduction

Phaser5 is an HTML- 5 game development platform framework for building 2D games. It is a very fun and easy to learn framework. It has many features that make game development easy such as image rendering, sprites and animation creation, input control that includes mouse and keyboard events and three physics engines that can be used for things like collision detection, speed and acceleration settings and many more exciting things!

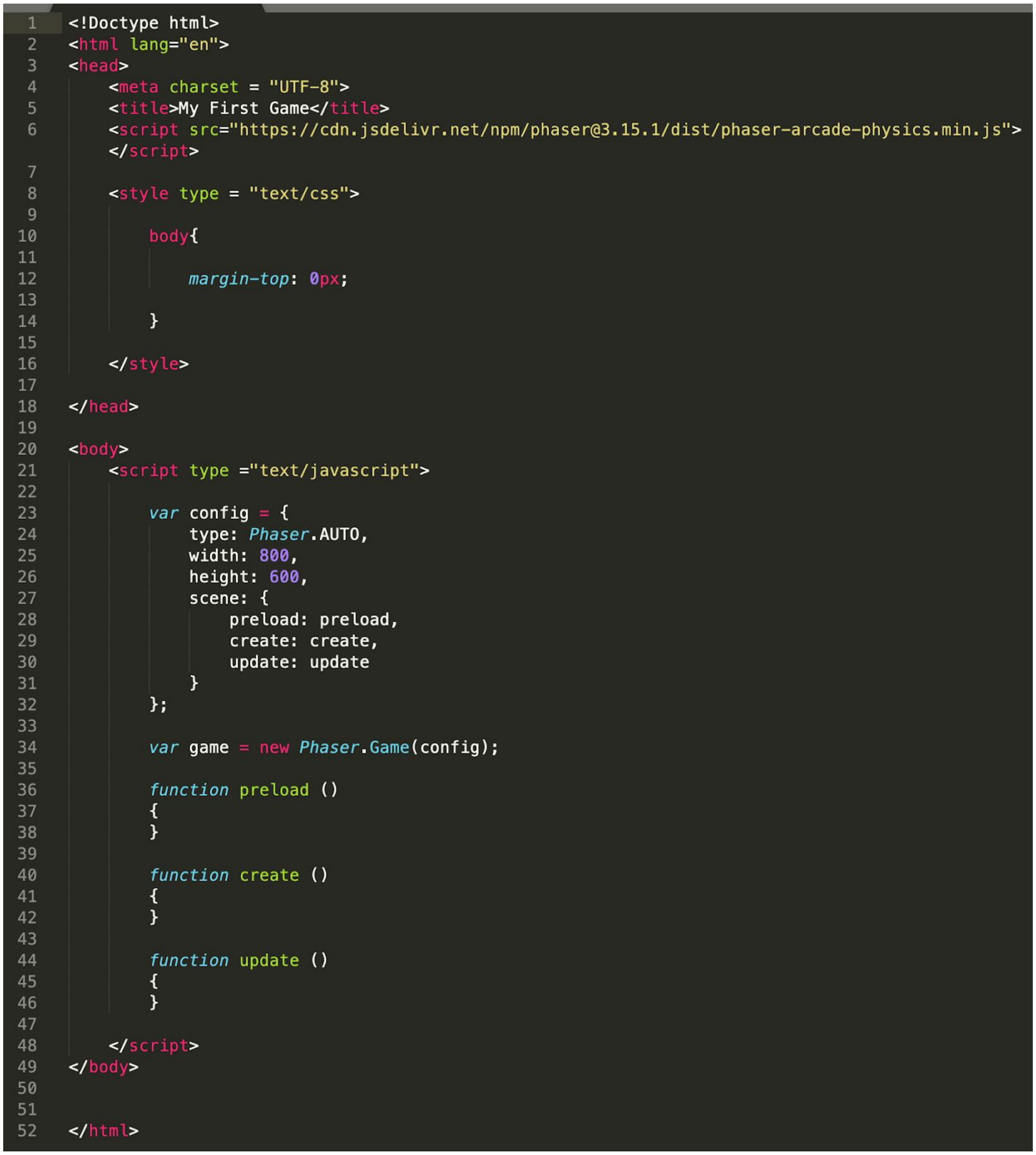
There are a number of ways of incorporating the Phaser 5 framework in your project. These lesson series will focus on being simple and straightforward by using the boilerplate code pattern available on Phaser's official website. The boilerplate code is a single HTML document. It includes the Phaser 5 framework available on jsDelivr CDN. Both CSS and JavaScript are also incorporated in this single document using HTML style and script elements.

To understand the framework, we will start building a simple game called Shoot the Stars. As we build, we will learn along all the way. Let's look at the HTML code structure in detail.

Glossary

A **platform framework** is a collection of code,tools and runtime environment that provides a building block or predefined structure for creating another software. This makes it easier for other developers to reuse and extend the framework functionalities by building their applications on top of it and customizing the functionalities according to their software requirements.

Boilerplate HTML that includes Phaser, CSS and JavaScript



```html
<!Doctype html>
<html lang="en">
<head>
    <meta charset = "UTF-8">
    <title>My First Game</title>
    <script src="https://cdn.jsdelivr.net/npm/phaser@3.15.1/dist/phaser-arcade-physics.min.js">
    </script>

    <style type = "text/css">

        body{

            margin-top: 0px;

        }

    </style>

</head>

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

        var config = {
            type: Phaser.AUTO,
            width: 800,
            height: 600,
            scene: {
                preload: preload,
                create: create,
                update: update
            }
        };

        var game = new Phaser.Game(config);

        function preload ()
        {
        }

        function create ()
        {
        }

        function update ()
        {
        }

    </script>
</body>


</html>
```

Brief overview of the HTML document

The HTML

1. The HTML document starts with a document type declaration followed by the <html> element. The <html> element is the root container in an HTML document and holds different elements of the document. It consists mainly of the <head> and <body> elements.
2. The <head> element defines the meta-data of the document such as the character encoding, title of the document, links to additional scripts and style element for writing CSS.
3. The <body> element is the document body which is the visible part of the HTML document. The JavaScript goes inside of it under the <script> element.
4. Within the HTML <head> element, a <script> element is used to acquire the Phaser script from the jsDelivr CDN . It is done by setting the src attribute of the <script> element to the specified url as shown below:

```
<script src =
"https://cdn.jsdelivr.net/npm/phaser@3.15.1/di
st/phaser-arcade-physics.min.js"></script>
```

The CSS

5. The <style> element uses the body selector and sets the margin-top property to a value of 0 pixels.

```
<style type = “text/css”>

body {

        margin-top: 0px;

}

</style>
```

The JavaScript

6. The <body> element includes the <script> element. This is where all the JavaScript will go.

The Configuration Object

7. The <script> element has a predefined variable called config of type object { }. It is the configuration object. It is It has a couple of key:value pairs. These key:value pairs are some basic properties that can be passed as an argument to the Phaser.Game constructor in order to instantiate a Phaser.Game object.

The Phaser.Game Object

8. Phaser.Game object is the main controller of the Phaser framework. It does all the initial setup for the game which includes booting, parsing the configuration object that has been passed to it, creating a renderer and making the sound and input controls available. After that it calls the Scene Manager which is responsible for starting the first scene in the game. The Phaser game is made up of scenes which has the actual game logic. For more details you can browse through the official Phaser documentation website

   https://photonstorm.github.io/phaser3-docs/Phaser.Game.html .

9. The configuration object contains all the values for what type of renderer to be used, the height and width of the game screen and the scenes that are part of the game.

The Phaser.Scene Object

10. In the above configuration object, there is only a single scene and its predefined methods which are to be overridden by our code are declared: preload, create and update. Each of these methods has a specific purpose.

    preload():

    In this predefined function, we can use the Loader object to load external assets into our game. Our assets can be sprites, images,sounds, texture atlases and data files. A Scene object has a unique Loader object associated with it. However, whatever assets the Loader of any Scene loads, it gets stored inside of the global game-level caches and hence are accessible to other Scene objects in the game.

    Example:

Let’s find an image and download it to our machines at the location where your html file is stored.

Inside of your preload function type:

this.load(‘game-object’, *path*);

create():

This function is where you can actually create your loaded content, display them, give them properties and use them according to your game requirements. It is automatically called after the preload() function.

Example:

Now let’s display our image. Type in the following code:

var image = this.add.image(300,200,’game-object’);

update():

This function is called 60 frames every second. It is a function that loops as long as your game Scene is running. It updates and redraws the game objects. This property of the function allows us to add event handling logics such as collision detections, input events and many more.

Example:

Now let’s try moving our image from left to right.

this.image.x +=100;