Week 6: More About How Phaser.js Works

Going Deeper into Phaser

In last week's materials, we looked at the basics of the Phase game engine, exploring the game loop, game states, the world, assets, graphics, and the basics of sprites. In this week, we go deeper into the object model that the Phaser.js JavaScript library exposes to us for game building.

The Phaser Object Model

Phaser is organized as a set of classes – different kinds of objects that we need to compose a playable game.The Phaser CE documentation has details about each kind of object we can create. The most basic kind of object we create with Phaser is a Game.

```
var mygame = new Phaser.Game( 400, 500, Phaser.CANVAS, NULL, { } );
```

Each Game object we create contains a collection of properties (values) and methods (commands) we can work with in our JavaScript code. For instance, we can read the dimensions of any Game we create using the height or width properties.

```
var myheight = mygame.height;
```

We use dot notation to access the properties and methods of any objects we create in code.

Sprites

Every entity in the game world that affects or is part of gameplay is a sprite. At its most basic a sprite is a set of coordinates and an image asset – a texture if you will – that is rendered in the game world. Sprites contain properties allowing for physics motion (via Sprite.body), input handling (via Sprite.input), events (via Sprite.events), animation (via Sprite.animations), camera integration as well as other capabilities.

We can create a new sprite in our game using dot notation to access the add object property of our game and call its sprite() method – add.sprite() accepts a few arguments: horizontal placement (x coordinate), vertical placement (y coordinate), and the asset name to use for the sprite's bitmap – what it looks like.

```
var mysprite = game.add.sprite(50,75,'assetname');
```

This topic is discussed in textbook section 6.8.

Sprite Sheets

> "Texture Packer is an essential part of our daily workflow. Every bit of GPU memory helps when dealing with mobile html5 games, so intelligent packing of assets is a must. And Texture Packer has all the features we need to effortlessly create atlases for our games."
> *Richard Davey (@photonstorm) - Creator of Phaser*

With a sprite sheet, we cut down on server requests. Rather than hitting the server ten times for ten different images, only one request is sent. That one file may have a larger relative file size, but it will likely be smaller than the sum of all the individual files. Smaller overall size and fewer HTTP requests makes for better performance.

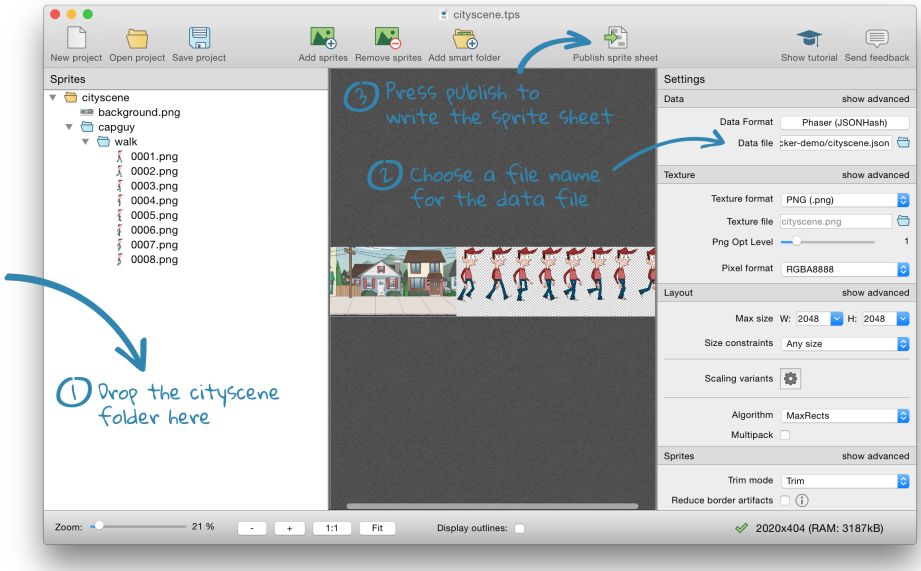
Once we have the full sprite sheet loaded into memory, we then need to display only the section of the image we need. When we load a sprite sheet asset in Phaser, we specify the dimensions of each individual sprite frame within the sheet.

```
game.load.spritesheet('myasset', 'assets/asset.png', 40, 30);
```

When we use the asset, we must let Phaser know which frame to display for the sprite.

```
mysprite = game.add.sprite(100, 300, 'myasset');
mysprite.frame = 2;
```

The rendering of sprites can be sped up dramatically by using sprite sheets. The browser-based tool known as SpriteMe, online at **http://spriteme.org/ (http://spriteme.org/)**, can help you compose your own sprite sheets. The commercial software TexturePacker provides a GUI toolset for composing and saving sprite sheets. You can download a free trial version of the software from **https://www.codeandweb.com/texturepacker/download (https://www.codeandweb.com/texturepacker/download)**. You can also compose a spritesheet in any bitmap editing software, including Adobe Photoshop, Watch this week's Screencast for a demonstration of creating one.



This topic is discussed in textbook section 6.8.1.

Fixed to Camera

If you need to keep a sprite visible within the view of the camera in your game, consider fixing the sprite to the camera's position.

```
mysprite.fixedToCamera = true;
```

This topic is discussed in textbook section 6.8.2.

Input

The Input class gives us access to all possible input devices that a game player may use to interact with the game. We can monitor the state of individual input devices as well as associate event triggers for sprite objects based on inputs. The Input object maps each type of input device to its own object value property. Each of these are discussed below.

This topic is discussed in textbook section 6.11.

Keyboard

Using JavaScript dot notation to access the current state of any keyboard key is trivial using Phaser's well-articulated Input object value. We can test keyboard key states in real-time using conditional statements.

```
if ( game.input.keyboard.isDown( Phaser.Keyboard.RIGHT ) ) {
  // do something
}
```

We can also associate a JavaScript function with a keyboard event.

```
this.spacekey = game.input.keyboard.addKey( Phaser.Keyboard.SPACEBAR );
this.spacekey.onDown.add( doSomething, this );
function doSomething( mygame ) {
  // runs when space key pressed
}
```

This topic is discussed in textbook section 6.11.1.

Pointers

Phaser considers all mouse, touchscreen and gesture device input to be a pointer. Note that because touchscreens support multiple touchpoints simultaneously, your code can receive multiple touch events and access multiple touchpoints' coordinates.

The following pointer properties are useful:

- **x**: horizontal coordinate of pointer / touchpoint

- **y**: vertical coordinate of pointer / touchpoint
- **isDown**: boolean status of mouse button or touch device touch
- **movementX**: how far pointer moved horizontally since last frame
- **movementY**: how far pointer moved vertically since last frame

This topic is discussed in textbook section 6.11.2.

Gamepad

Traditional gaming input devices, such as gamepads from XBOX, PlayStation, are supported in limited fashion by Phaser in Chrome and Firefox browsers.

Before allowing gamepad input, be sure to detect the presence of the gamepad input device.

```
if ( game.input.gamepad.active ) {
  // we have gamepad!
}
```

This topic is discussed in textbook section 6.11.3.