Loading assets

I usually declare an assets object that includes all my asset names and URLs to each asset. Add in the following highlighted code (lines 6-26) after the `gameProperties` object.

```
gameProperties

screenWidth

screenHeight

graphicAssets

ballURL'assets/ball.png'

ballName'ball'

paddleURL'assets/paddle.png'

paddleName'paddle'

soundAssets

ballBounceURL'assets/ballBounce'

ballBounceName'ballBounce'

ballHitURL'assets/ballHit'

ballHitName'ballHit'

ballMissedURL'assets/ballMissed'

ballMissedName'ballMissed'

mp4URL'.m4a'

oggURL'.ogg'
```

For each asset, there is a relative path and name associated with it. Here property naming format that I use:

- Relative path: [asset name] + “URL”
- Asset name for use in code: [asset name] + “Name”

The “ball” sprite asset property for example is `ballURL` and points to the location of the `ball.png` sprite file located in the `assets` folder. The “ball” name asset property is `ballName` and is assigned the string `'ball'`. Do take note that the value assigned for an asset name must be a string.

Next we use the `game.load` function to preload our games assets. The code is placed within the `preload` function (line 30). Use the

following code:

```
mainStateprototype

preloadfunction

imagegraphicAssetsballNamegraphicAssetsballURL

imagegraphicAssetspaddleNamegraphicAssetspaddleURL

audiosoundAssetsballBounceNamesoundAssetsballBounceURLsoundAssetsmp4URLsoundAssetsballBounceURLsoundAssetsoggURL

audiosoundAssetsballHitNamesoundAssetsballHitURLsoundAssetsmp4URLsoundAssetsballHitURLsoundAssetsoggURL

audiosoundAssetsballMissedNamesoundAssetsballMissedURLsoundAssetsmp4URLsoundAssetsballMissedURLsoundAssetsoggURL
```

The `game.load` method is the Phaser.Loader class that handles the loading of all external content such as Images, Sounds, Texture Atlases and data files. In the `preload` function, we call the `image` and `audio` methods to load each asset type.

The `image` method contains three parameters:

- `key` (string) – The unique asset name of this image file. This name is used in our code later on to identify which image to load in our game.
- `url` (string) – The URL where the image file is located.
- `overwrite` (boolean <optional>, `false` by default) – If set to `true`, this will overwrite an asset if there is an existing key.

Adding graphics

Let's start by declaring the variables that will contain our graphics. Within the `mainState` object declaration (line 28), add the following code:

```
mainStatefunction

backgroundGraphics

ballSprite

paddleLeftSprite

paddleRightSprite
```

Line 29: `this.backgroundGraphics` will contain a vertical line drawing that divides the game world into left and right sections. We will be using the `Phaser.Graphics` class to code our line drawing.

Line 30-32: `this.ballSprite`, `this.paddleLeftSprite` and `this.paddleRightSprite` will contain our sprites for the ball and player paddles,

Next, we will want to declare properties for the vertical line drawing and the position of our player paddles. Add the following code to the `gameProperties` object:

```
gameProperties
```

screenWidth

screenHeight

dashSize

paddleLeft_x

paddleRight_x

Line 5: `this.dashSize` is the size and spacing of the dashed line we will use when drawing our vertical line.

Line 7-8: `paddleLeft_x` and `paddleRight_x` is the horizontal x-position where our left and right player paddles will be in our game world.

Also, to avoid cluttering up the `create` function with too many lines, we'll break it down into smaller functions. In this case, we will create a function that will add all my sprites to the stage. Let's call the function `initGraphics` and add it after the update function:

updatefunction

initGraphicsfunction

In the `initGraphics` function, include the following code to add sprites to the stage:

initGraphicsfunction

backgroundGraphicsgraphics

backgroundGraphicslineStyle0xFFFFFF

gamePropertiesscreenHeightgamePropertiesdashSize

backgroundGraphicsmoveToworldcenterX

backgroundGraphicslineToworldcenterXgamePropertiesdashSize

ballSpritespriteworldcenterXworldcenterYgraphicAssetsballName

ballSpriteanchor

paddleLeftSpritespritegamePropertiespaddleLeft_xworldcenterYgraphicAssetspaddleName

paddleLeftSpriteanchor

paddleRightSpritespritegamePropertiespaddleRight_xworldcenterYgraphicAssetspaddleName

paddleRightSpriteanchor

We'll start by drawing a vertical line in the middle of the game world. We use the `game.add.graphics` method to create a new `Graphic` object and assign it to the `backgroundGraphics` property (line 74).

backgroundGraphicsgraphics

There are two arguments required with an optional third argument when creating a new Graphic object:

- The x-position: A value of 0 moves the object to the horizontal left of the game world.
- The y-position: A value of 0 moves the object to the vertical top of the game world.
- Group: An optional argument that assigns the new Graphic object to a group of objects and is usually used for collision detection.

For our line to appear on the screen, we will need to set the line style for our drawing (line 75).

backgroundGraphicslineStyle0xFFFFFF

Here we have three arguments:

- The first argument is the line width or thickness of the line in pixels.
- The second argument is the colour of the line using hex code. You can use these free online tools to select a colour: Hex Color Tool[1] and Color Hex Color Codes[2].
- The third argument is the alpha or opacity value of the line. The range of values are between 0 and 1. The value 0 sets the opacity to 0%, 0.5 sets it to 50% and 1 sets it to 100%.

Next, we'll draw our vertical line using a `for` loop (line 77). A loop is used to execute a block of code as many times as needed. In this case, lines 78-79 will be repeated until the loop condition is no longer true.

gamePropertiesscreenHeightgamePropertiesdashSize

backgroundGraphicsmoveToworldcenterX

backgroundGraphicslineToworldcenterXgamePropertiesdashSize

Notice a `for` loop consists of three parts separated by a semicolon.

The first part: `var y=0;` we initialise our values. In this case, we set the variable `y` to a value of `0`.

The second part: `y < gameProperties.screenHeight;` is our conditional test. The code within the `for` loop will execute as long as the conditional test is true. In this case, the value of `gameProperties.screenHeight` is `480`. The condition proves to be `true` as long as the value of is less than `480`.

The third part: `y += gameProperties.dashSize * 2` is our update that runs after all the code within the `for` loop has been executed. So each time our code executes, the value of `y` is incremented by `10`.

Let's look at how this loop works in lines 78 and 79.

In line 78, we call the a `moveTo` method from the `Graphics` class. The `moveTo` method requires two arguments: the x-position and y-position. Calling this method moves our drawing position to a point within the game world using the two arguments.

In line 79, we call the a `lineTo` method from the `Graphics` class. The `lineTo` method also requires two arguments: the x-position and y-position. Calling this method draws a line from the current drawing position to the x-position and y-position set in the two arguments. After the line is drawn, the current drawing position is updated using the two arguments.

Both `moveTo` and `lineTo` methods use the `game.world.centerX` property which is the horizontal centre of the game world. As our game world width is 640 pixels, the `centerX` is 320 pixels.

Now that we have the vertical line drawn, we'll add the ball and two paddle sprites in lines 82-89:

ballSpritespriteworldcenterXworldcenterYgraphicAssetsballName

ballSpriteanchor

paddleLeftSpritespritegamePropertiespaddleLeft_xworldcenterYgraphicAssetspaddleName

paddleLeftSpriteanchor

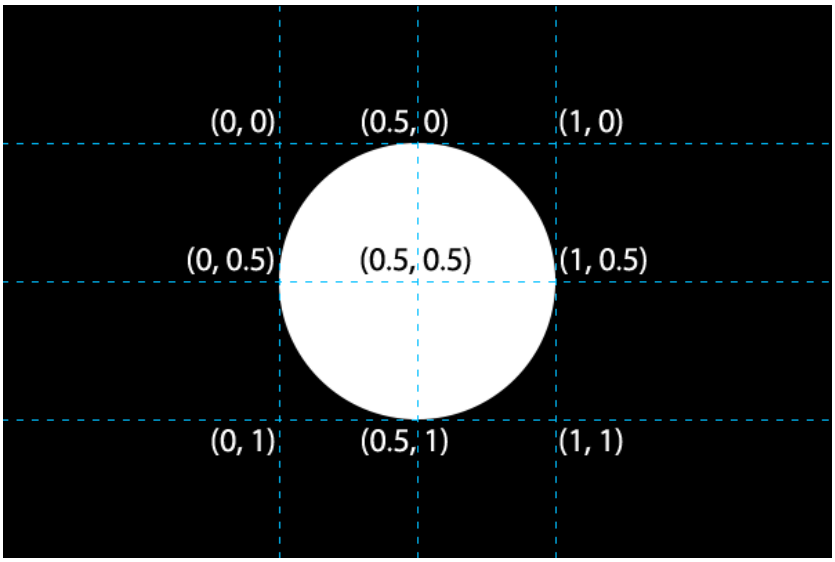
paddleRightSpritespritegamePropertiespaddleRight_xworldcenterYgraphicAssetspaddleName

paddleRightSpriteanchor

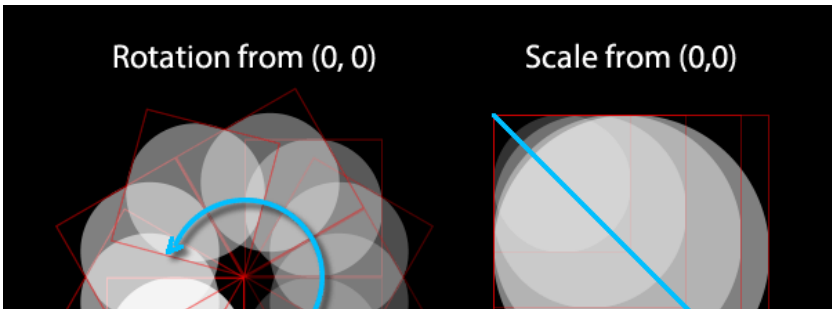
To add a sprite, the `game.add.sprite` method is used. Adding a sprite requires three arguments: the x-position, y-position and name of the asset to load. We declared the names of our sprite assets earlier in the `graphicAssets` object.

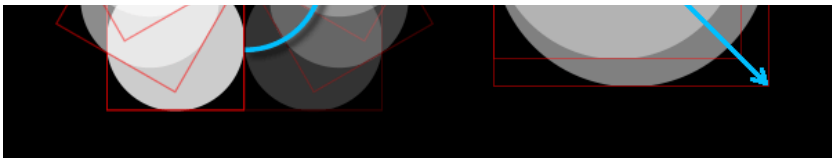
After adding new sprites, we need to adjust the anchor point for each sprite. By default, the anchor point for newly added sprites is located at the `(0, 0)` coordinates or the top-left of the sprite. Setting the anchor to `(0.5, 0.5)` moves our anchor point to 50% width and 50% height (or the centre) of a sprite.

The illustration below shows where the anchor point will be depending on what you set.



The reason for moving the anchor point is to ensure the sprite rotates and scales according to the anchor point. Leaving the anchor point at the default (0, 0) will cause the sprite to rotate and scale as illustrated below:





Also notice that we are reusing the same paddle sprite asset when we add the left and right paddles. The only difference between the left and right paddles is the first argument when adding the sprite. For the left paddle, we are using `gameProperties.paddleLeft_x` while the right paddle uses `gameProperties.paddleRight_x`.

One final step before we can see our work in progress. In the `create` function, add the following code to execute the `initGraphics` function after the assets are loaded:

createfunction

initGraphics

Here is what our current progress should look like.

Download the source files for this step here[3].

In the next step[4], we'll get the ball moving and colliding with our boundaries.

1. http://www.hexcolortool.com/
2. http://www.color-hex.com/
3. https://github.com/zekechan/phaser-html5-tutorial-pong/releases/download/1.0/2-Add_Sprites.zip
4. http://zekechan.net/getting-started-html5-game-development-pong3/