Week_23

03/06/2023

Good Morning Trinity - Introduction to Computer Programming with Game Development…

We will begin with a "PHASER"<Logo> (animation type) Demo, to learn more about the framework of Game Development…

Demo from Coach Arthur
(hands on Exercise - making the particle emitter follow the animated LOGO)…

Then, we Launch a series of classes that will build a new game from the ground up. Many students have been where you are today, before. After this class you will be the one helping others to cross the same hurdles. Be an encouragement to others & lend a helping hand.

Today's Class is about Building the World that your new game will live in!!!!

More about **Game Design** through: Phaser

Making a Game with Phaser 3 is both fun and easy following these steps. Here we will learn how to create a small game involving a player running and jumping around platforms, collecting stars and more. While going through this process we'll explain some of the core features of the Phaser framework.

Quick Review - What is Phaser?

Phaser is an HTML5 game framework which aims to help developers make powerful, cross-browser HTML5 games really quickly. It was created specifically to harness the benefits of modern browsers, both desktop and mobile. The only browser requirement is the support of the canvas tag.

** it takes 3 things to get something done:

1) Decide What to do… (this is the hardest step)
2) Decide How you will do it…
3) Begin, and get it all done. Do it

3) Begin, and get it all done, Do it...

Making a Phaser Game: Loading Assets

Let's load the assets we need for our game. You do this by putting calls to the Phaser Loader inside of a Scene function called preload. Phaser will automatically look for this function when it starts and load anything defined within it.
Currently the preload function is empty. Change it to:

```
function preload ()
{
    this.load.image('sky', 'assets/sky.png');
    this.load.image('ground', 'assets/platform.png');
    this.load.image('star', 'assets/star.png');
    this.load.image('bomb', 'assets/bomb.png');
    this.load.spritesheet('dude',
        'assets/dude.png',
        { frameWidth: 32, frameHeight: 48 }
    );
}
```

This will load in 5 assets: 4 images and a sprite sheet. It may appear obvious to some of you, but I would like to point out the first parameter, also

known as the asset key (i.e. 'sky', 'bomb'). This string is a link to the loaded asset and is what you'll use in your code when creating Game Objects. You're free to use any valid JavaScript string as the key.

(Key:Value)

Display an Image

In order to display one of the images we've loaded place the following code inside the `create` function:

```
this.add.image(400, 300, 'sky');
```

The values `400` and `300` are the x and y coordinates of the image. Why 400 and 300? It's because in Phaser 3 all Game Objects are positioned based on their center by default. The background image is 800 x 600 pixels in size, so if we were to display it centered at 0 x 0 you'd only see the bottom-right corner of it. If we display it at 400 x 300 you see the whole thing.

Hint: You can use `setOrigin` to change this. For example the code: `this.add.image(0, 0, 'sky').setOrigin(0, 0)` would reset the drawing position of the image to

the top-left. In Phaser 2 this was achieved via the anchor property but in Phaser 3 it's the originX and originY properties instead.

The order in which game objects are displayed matches the order in which you create them. So if you wish to place a star sprite above the background, you would need to ensure that it was added as an image second, after the sky image:

```
function create ()
{
    this.add.image(400, 300, 'sky');
    this.add.image(400, 300, 'star');
}
```

If you put the star image first it will be covered-up by the sky image.

World Building

Under the hood this.add.image is creating a new Image Game Object and adding it to the current Scenes display list. This list is where all of your Game Objects live. You could position the image anywhere and Phaser will not mind. Of course, if it's outside of the region 0x0 to 800x600 then you're not going to visually see it, because it'll be "off screen", but it will still exist within

because it'll be "off screen", but it will still exist within the Scene.

The Scene itself has no fixed size and extends infinitely in all directions. The Camera system controls your view into the Scene and you can move and zoom the active camera as required. You can also create new cameras for other views into the Scene. This topic is beyond the scope of this specific tutorial, suffice to say the camera system in Phaser 3 is significantly more powerful than in v2. Things that were literally not possible before now are.

For now let's build up the Scene by adding a background image and some platforms. Here is the updated `create` function:

```
var platforms;
function create ()
{
    this.add.image(400, 300, 'sky');

platforms = this.physics.add.staticGroup();

platforms.create(400, 568,
'ground').setScale(2).refreshBody();

//   Collider

platforms.create(600, 400, 'ground');
```

```
    platforms.create(50, 250, 'ground');
    platforms.create(750, 220, 'ground');
}
```

Glancing quickly at the code you'll see a call to `this.physics`. This means we're using the Arcade Physics system, but before we can do that we need to add it to our Game Config to tell Phaser our game requires it. So let's update that to include physics support. Here is the revised game config:

```
var config = {
    type: Phaser.AUTO,
    width: 800,
    height: 600,
    physics: {
        default: 'arcade',
        arcade: {
            gravity: { y: 300 },
            debug: false
        }
    },
    scene: {
        preload: preload,
        create: create,
        update: update
    }
};
```

The new addition is the `physics` property.

The Platforms

(important for our player to jump,

And explore the scene)

We just added a bunch of code to our `create` function that deserves a more detailed explanation. First, this part:

```
platforms = this.physics.add.staticGroup();
```

This creates a new Static Physics Group and assigns it to the local variable `platforms`. In Arcade Physics there are two types of physics bodies: Dynamic and Static. A dynamic body is one that can move around via forces such as velocity or acceleration. It can bounce and collide with other objects and that collision is influenced by the mass of the body and other elements.

In stark contrast, a Static Body simply has a position and a size. It isn't touched by gravity, you cannot set velocity on it and when something collides with it, it never moves. Static by name, static by nature. And perfect for the ground and platforms that we're going to let the player run around on.

But what is a Group? As their name implies they are ways for you to group together similar objects and control them all as one single unit.

You can also check for collision between Groups and other game objects. Groups are capable of creating their own Game Objects via handy helper functions like `create`. A Physics Group will automatically create physics enabled children, saving you some leg-work in the process.

With our platform Group made we can now use it to create the platforms:

```
platforms.create(400, 568,
'ground').setScale(2).refreshBody();
platforms.create(600, 400, 'ground');
platforms.create(50, 250, 'ground');
platforms.create(750, 220, 'ground');
```

As we saw previously it creates this scene:

During our preload we imported a 'ground' image. It's a simple green rectangle, 400 x 32 pixels in size and will serve for our basic platform needs:

The first line of code above adds a new ground image at 400 x 568 (remember, images are positioned based on their center) - the problem is that we need this platform to span the full

is that we need this platform to span the full width of our game, otherwise the player will just drop off the sides. To do that we scale it x2 with the function `setScale(2)`. It's now 800 x 64 in size, which is perfect for our needs. The call to `refreshBody()` is required because we have scaled a *static* physics body, so we have to tell the physics world about the changes we made.

The ground is scaled and in place, so it's time for the other platforms:

```
platforms.create(600, 400, 'ground');
platforms.create(50, 250, 'ground');
platforms.create(750, 220, 'ground');
```

The process is exactly the same as before, only we don't need to scale these platforms as they're the right size already.

3 platforms are placed around the screen, the right distances apart to allow the player to leap up to them.