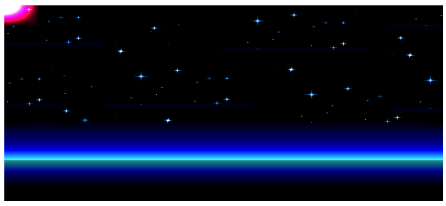
Week 25

Trinity Intro. To Computer Programming
With Game Development 03/27/2024

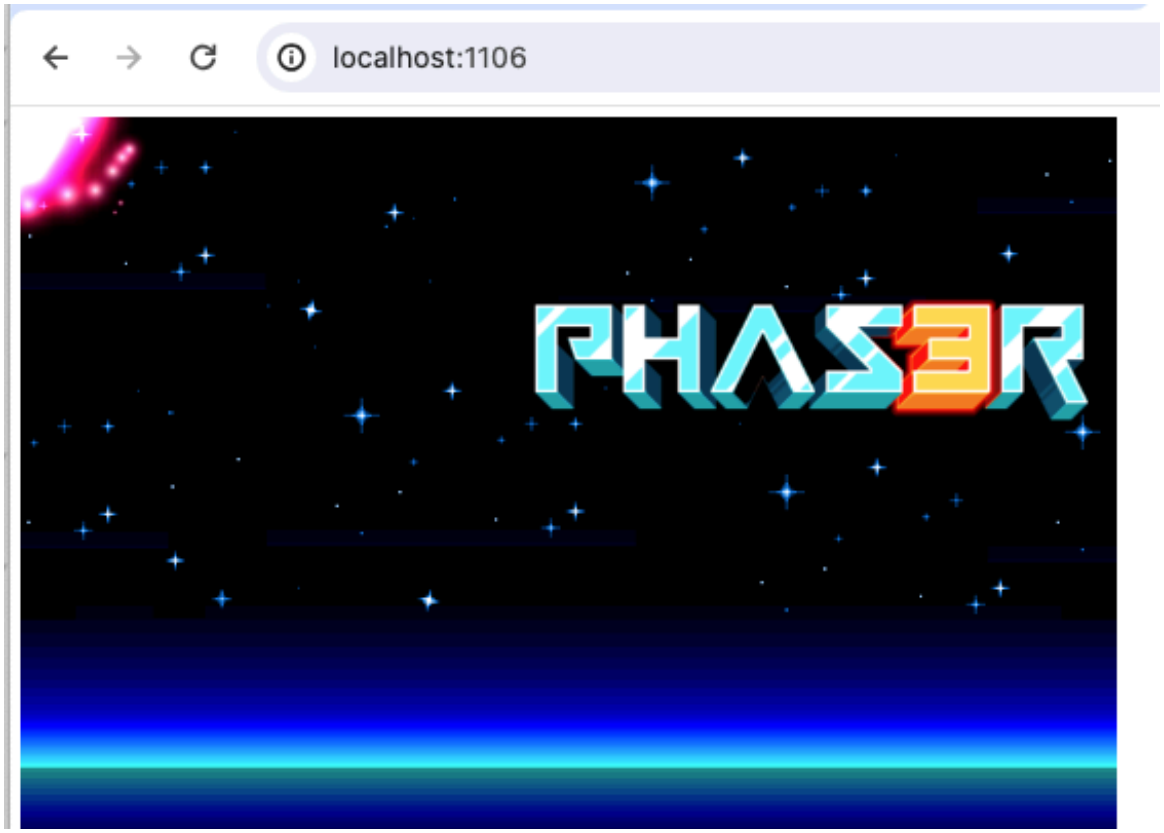
Todays class we will continue with our
Dude Runner - Player Game with the <Sprites>…

First we will have a Demo & Exercise from Coach Arthur -
Phaser_Sample (Learning more about Phaser Physics)…
<<eye candy>>




```
6 ▼ <body>
7
```

```
    <script>
    class Example extends Phaser.Scene
    {
        preload ()
        {
           // this.load.setBaseURL('https://labs.phaser.io');

            this.load.image('sky', 'assets/skies/space3.png');
            this.load.image('logo', 'assets/sprites/phaser3-logo.png');
            this.load.image('red', 'assets/particles/red.png');
        }

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

            const particles = this.add.particles(0, 0, 'red', {
                speed: 100,
                scale: { start: 1, end: 0 },
                blendMode: 'ADD'
            });

            const logo = this.physics.add.image(400, 100, 'logo');

            logo.setVelocity(100, 200);
            logo.setBounce(1, 1);
            logo.setCollideWorldBounds(true);

            particles.startFollow(logo); // add this line

        }
    }

    const config = {
```



Add line 36 to the (dir) sample
Index.html
particles.startFollow(logo);

Students, how does this change the code???

Next -

It's time to give our little game a purpose. Let's drop a sprinkling of stars into the scene and allow the player to collect them. To achieve this we'll create a new Group called 'stars' and populate it. In our create function we add the following code

```
stars = this.physics.add.group({
    key: 'star',
    repeat: 11,
    setXY: { x: 12, y: 0, stepX: 70 }
});
stars.children.iterate(function (child) {


child.setBounceY(Phaser.Math.FloatBetween(0.4, 0.8));


});
```

The process is similar to when we created the platforms Group. As we need the stars to move and bounce we create a dynamic physics group instead of a static one.

Groups are able to take configuration objects to aid in their setup. In this case the group configuration object has 3 parts: First it sets the texture key to be the star image. This means that any children created as a result of the config object will all be given the star texture by default. Then it sets the repeat value to be 11. Because it creates 1 child automatically, repeating 11 times means we'll get 12 in total, which is just what we need for our game.

The final part is `setXY` - this is used to set the position of the 12 children the Group creates. Each child will be placed starting at x: 12, y: 0 and with an x step of 70. This means that the first child will be positioned at 12 x 0, the second one is 70 pixels on from that at 82 x 0, the third one is at 152 x 0, and so on. The 'step' values are a really handy way of spacing out a Groups children during creation. The value of 70 is chosen because it means all 12 children will be perfectly

spaced out across the screen.

The next piece of code iterates all children in the Group and gives them a random Y bounce value between 0.4 and 0.8. The bounce range is between 0, no bounce at all, and 1, a full bounce. Because the stars are all spawned at y 0 gravity is going to pull them down until they collide with the platforms or ground. The bounce value means they'll randomly bounce back up again until finally settling to rest.

If we were to run the code like it is now the stars would fall through the bottom of the game and out of sight. To stop that we need to check for their collision against the platforms. We can use another Collider object to do this:

```
this.physics.add.collider(stars, platforms);
```

As well as doing this we will also check to see if the player overlaps with a star or not:

```
this.physics.add.overlap(player, stars, collectStar, null, this);
```

This tells Phaser to check for an overlap between the player and any star in the stars Group. If found then they are passed to the 'collectStar'

function:

```
function collectStar (player, star)
{
    star.disableBody(true, true);
}
```

Quite simply the star has its physics body disabled and its parent Game Object is made inactive and invisible, which removes it from display. Running the game now gives us a player that can dash about, jump, bounce off the platforms and collecting the stars that fall from above.

-------> NEXT

There are two final touches we're going to add to our game: an enemy to avoid that can kill the player, and a score when you collect the stars. First, the score.

To do this we'll make use of a Text Game Object. Here we create two new variables, one to hold the actual score and the text object itself:

```
var score = 0;
var scoreText;
```

The `scoreText` is set-up in the `create` function:

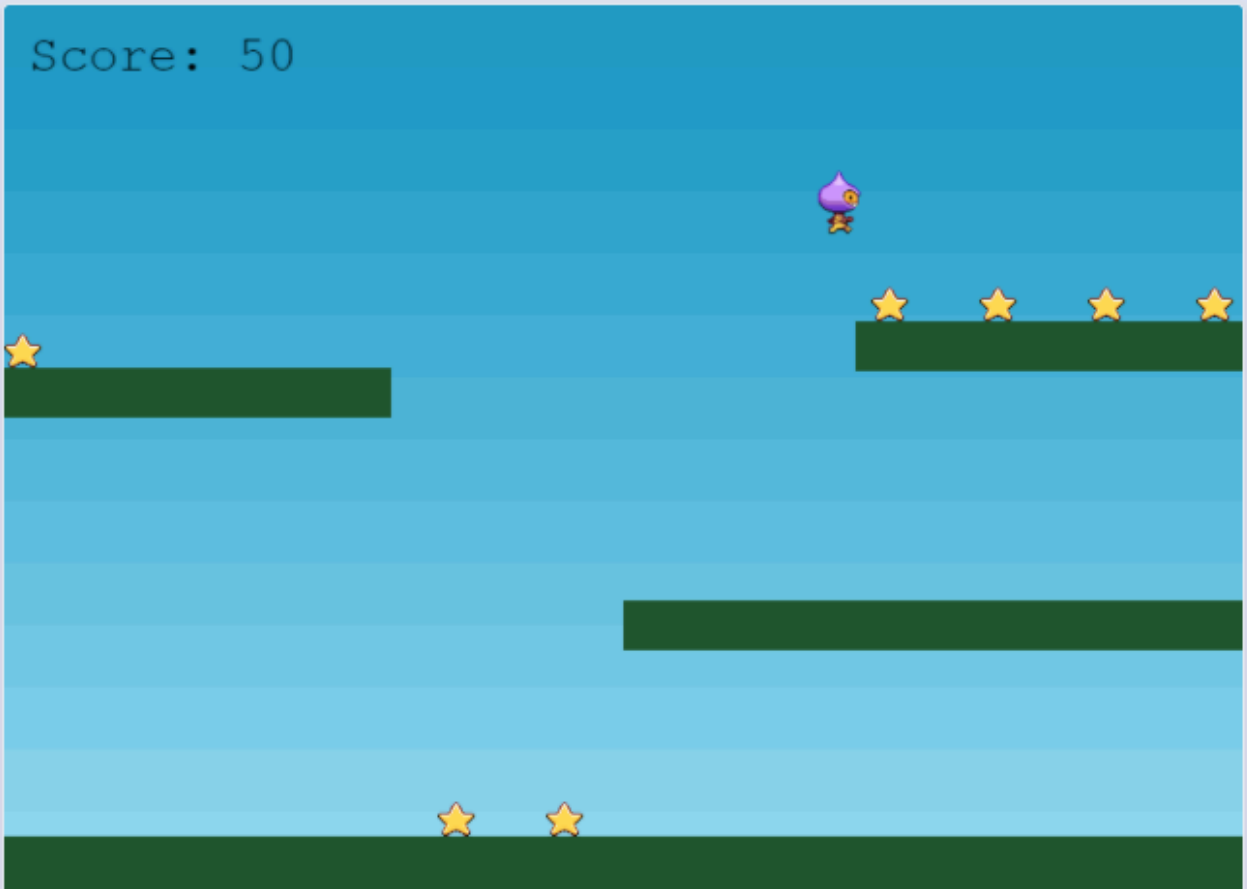
```
scoreText = this.add.text(16, 16, 'score: 0', { fontSize: '32px', fill: '#000' });
```

16 x 16 is the coordinate to display the text at. 'score: 0' is the default string to display and the object that follows contains a font size and fill color. By not specifying which font we'll actually use the Phaser default, which is Courier.

Next we need to modify the `collectStar` function so that when the player picks-up a star their score increases and the text is updated to reflect this:

```
function collectStar (player, star)
{
    star.disableBody(true, true);

    score += 10;
    scoreText.setText('Score: ' + score);
}
```

So 10 points are added for every star and the `scoreText` is updated to show this new total. If you run `part9.html` you will see the stars fall and the score increase as you collect them.



Demo with Coach Arthur -
(code) & Add Score.

Line 105 (add this line)

```
this.physics.add.collider(stars, platforms);
```

Run demo: (Notice what happens with the game)

Line 107 (add this line)

```
this.physics.add.overlap(player, stars, collectStar, null, this);
```

Line 140 modify function collectstar (add these 2 lines):

```
score = score + 10;
scoreText.setText('Score: ' + score);
```