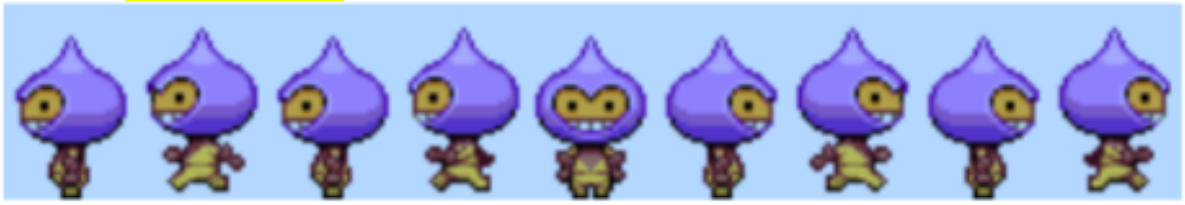
Week 27

# Trinity - Intro. To Computer Programing with Game Development.

## Quick review -

## We added 2 new variables:

1) player (dude.png frames 0 - 8)



2) anims (ANIMATIONS)
(left, right, & turn)

```
player = this.physics.add.sprite(100, 450, 'dude');
player.setBounce(0.2);
player.setCollideWorldBounds(true);


this.anims.create({
    key: 'left',
    frames: this.anims.generateFrameNumbers('dude', { start: 0, end: 3 }),
    frameRate: 10,
    repeat: -1
```

```
});

this.anims.create({
    key: 'turn',
    frames: [ { key: 'dude', frame: 4 } ],
    frameRate: 20
});

this.anims.create({
    key: 'right',
    frames: this.anims.generateFrameNumbers('dude', { start: 5, end: 8 }),
    frameRate: 10,
    repeat: -1
});
```

# Lets Control our Dude With Keyboard Controls

A cursor is an Object (we will store 3 keys - Left, Right, & Up)

Phaser has a built-in Keyboard manager and one of the benefits of using that is this handy little function:

cursors = this.input.keyboard.createCursorKeys();

This populates the cursors object with four properties: up, down, left, right, that are all instances of Key objects. Then all we need to do is POLL these in our update loop. To POLL means to "check the status".

## How we make this work

The first thing it does is check to see if the left key is being held down. If it is we apply a negative horizontal velocity and start the 'left' running animation (dude.png frames 0-3).

```
if (cursors.left.isDown)
{
    player.setVelocityX(-160);
player.anims.play('left', true);
}
```

If they are holding down 'right' instead we literally do the opposite: (dude.png frames 5-8)

```
else if (cursors.right.isDown)
{
    player.setVelocityX(160);
player.anims.play('right', true);
}
```

By clearing the velocity and setting it in this manner, every frame, it creates a 'stop-start' style of movement. Dude.png(frame 4)

```
else
{
    player.setVelocityX(0);
player.anims.play('turn');
}
```

The player sprite will move only when a key is held down and stop immediately they are not. Phaser also allows you to create more complex motions, with momentum and acceleration, but this gives us the effect we need for this game. The final part of the key check sets the animation to 'turn' and zero the horizontal velocity if no key is held down.

# Jump to it

The final part of the code adds the ability to jump. The up cursor is our jump key and we test if that is down.

However we also test if the player is touching the floor, otherwise they could jump while in mid-air.

If both of these conditions are met we apply a vertical velocity of 330 px/sec sq. The player will fall to the ground automatically because of gravity. With the controls in place we now have a game world we can explore.Try tweaking values like the 330 for the jump to lower and higher values to see the effect it will have.

```
if (cursors.up.isDown &&
player.body.touching.down)
{
    player.setVelocityY(-330);
}
```

# ***Stardust***

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 some code.

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:

1) 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.

2) 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.

game.

3) 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.

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

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.

```
stars.children.iterate(function (child) {
child.setBounceY(Phaser.Math.FloatBetween(0.4, 0.8));

});
```

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);
```

## *How do we tell phaser we "HIT" a Star?*

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.**