Week 29

Trinity Introduction to Computer Programming with Game development

Lets start with a review of what we accomplished last week with the Stars and Points.

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.

Next lets some audio!

```
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', { fr
    32, frameHeight: 48 });

    // load new sound effect
    //  this.load.audio("move", ["assets/move.ogg",
    "assets/move.mp3"]);
}
```

Demo: adding sound…

In order to round our game out it's time to add some baddies. This will give a nice element of challenge to the game, something that was previously missing.

The idea is this: When you collect all the stars the first time it will release a bouncing bomb. The bomb will just randomly bounce around the level and if you collide with it, you die. All of the stars will respawn so you can collect them again, and if you do, it will release another bomb. This will give the player a challenge: get as high a score as possible without dying.

The first thing we need is a Group for the bombs and a couple of Colliders:

```
bombs = this.physics.add.group();
this.physics.add.collider(bombs, platforms);


this.physics.add.collider(player, bombs, hitBomb, null, this);
```

The bombs will of course bounce off the platforms, and if the player hits them we'll call the hitBomb function. All that will do is stop the game and turn the player red:

```
function hitBomb (player, bomb)
{
    this.physics.pause();
    player.setTint(0xff0000);

   player.anims.play('turn');

    gameOver = true;
}
```

So far, so good, but we need to release a bomb. To do that we modify the collectStar function:

```
function collectStar (player, star)
{
    star.disableBody(true, true);
score += 10;
    scoreText.setText('Score: ' + score);


if (stars.countActive(true) === 0)
    {
        stars.children.iterate(function (child)
{child.enableBody(true, child.x, 0, true, true);


});


 var x = (player.x < 400) ? Phaser.Math.Between(400, 800) :
Phaser.Math.Between(0, 400);


var bomb = bombs.create(x, 16, 'bomb');
        bomb.setBounce(1);
        bomb.setCollideWorldBounds(true);
        bomb.setVelocity(Phaser.Math.Between(-200, 200), 20);


    }
}
```

We use a Group method called countActive to see how many stars are left alive. If it's none then the player has collected them all, so we use the iterate function to re-enable all of the stars and reset their y position to zero. This will make all of the stars drop from the top of the screen again.

The next part of the code creates a bomb. First we pick a random x coordinate for it, always on the opposite side of the screen to the player, just to give them a chance. Then the bomb is created, it's set to collide with the world, bounce and have a random velocity.

The end result is a nice little bomb sprite that rebounds around the screen. Small enough to be easy to avoid, at the start, but as soon as the numbers build up it becomes a lot harder!

Conclusion

You have now learned how to create a sprite with physics properties, to control its motion and to make it interact with other objects in a small game world. There are lots more things you can do to enhance this.