Lesson of Our Favourite Programming Technique

Our favorite programming technique this time around was Box2D lights because it was similar to the way Box2D is setup and gave same pretty interesting effects.

Box2D lights uses *RayHandler* to manage all the different cone, point, and chain lights, much like Box2D uses *world.* To create a RayHandler you need to declare a RayHandler rayHandler and in your create() function have

rayHandler = new RayHandler(world);

This will let it interact with all of the objects in the world. The rayHandler has a few properties you can adjust such as ambient light, blur, blur amount, and culling. Some of these effects can not be seen unless you have actual lights set up.

rayHandler.setAmbientLight(0.1f);

rayHandler.setBlur(bBlur);

rayHandler.setBlurNum(50);

rayHandler.setCulling(true);

This link <https://github.com/libgdx/box2dlights/blob/master/src/box2dLight/RayHandler.java> has some more information on RayHandlers, although we have not tried all of it.

There are three types of lights we know how to use. These are point, cone, and chain lights.

Point lights cast a circle all around themselves. To create a PointLight use

pointLight = new PointLight(rayHandler, 20, Color.RED, 128 / PPM, 17, 11);

First, you reference the rayHandler so your light gets rendered. The second number is the amount of rays the light will cast, between 20 and 200 creates good looking lights. Less than 20 can look incomplete, more than 200 is unnecessary. Next you choose a colour. After that you set the distance the rays go. You finish up by setting the X and Y coordinates.

Cone lights cast light out in a cone in a certain direction. To create a ConeLight use

coneLight = new ConeLight(rayHandler, 50, Color.GOLDENROD, 5, 7, 7, 45, 30);

This should look similar to creating a pointLight, except now there are two additional numbers on the end. These are for the angle as well as the cone size.

Cone and point lights can both use these functions.

pointLight.setSoftnessLength(0f);

pointLight.attachToBody(body);

pointLight.setXray(true);

setSoftnessLength can stop the light from bleeding through the terrain. attachToBody can make it so your light stays on top of a body in your world. setXray can make your light ignore walls and cast right through them.

Chain lights are slightly more complicated than the others. They make a chain of lights that connects a series of points.

chainLight = new ChainLight(rayHandler, 20, Color.GREEN, 5, 4, new float[]{1, 6, 12, 6, 12, 15});

The coordinates of the points they go to are stored in the array. The first number is the first x coordinate, the second number is the second y coordinate. The number we have set to 5 is the length of the ray, the number we have as 20 is the number of rays, and the number we have set as 4 is the ray direction, although it seems to not do anything.

To make all of these appear on the screen you have to add rayHandler.render() to your rendering. Keep in mind render order matters, and you should render lights after the world. You should also use rayHandler.dispose() wherever you do your disposal.