**Feed the Bunny**

In this game, we have a watermelon fruit hanging from the rope. The bunny is waiting at the bottom to eat the watermelon fruit. When the user clicks on the cut the rope button, the watermelon falls and the bunny eats it.

Our Bunny will be standing on the ground. The first thing we need to do is to create the ground body and for that, we will create the ground class.

● Create a new file and name it ground.js.

● Now add this file in the index.html, so that we can use it in our code.

Open the ground.js file, here we are going to create the Ground class. The ground will be a stationary body, so we only need to create the constructor() function and the function to display it. In the constructor, we will specify the x, y positions, and the width and height, which will be entered by the user while creating the ground object for the ground() class.

***Note***: **Flags** are variables that we can use to change the state of an object, flags have only **boolean** values either true or false.

The ground is a rectangular body, that is why we are using the Bodies.rectangle() function to create this body. Once the body creation is done, we will add this body to the world.

When we create a rectangle in p5, it takes the top-left point as the origin of the rectangle. But in order to change the origin point of the rectangle from the top left corner to the center, we use the rectMode() function. In this function, we pass CENTER as an argument. This will create a rectangle with the center point as origin. As we want to have a specific color and stroke to the ground, which is why we are using the push() function, and after this will add the color and stroke properties of the ground.

**noStroke()** is to have no outline around the body.

We use the **fill**() function to add color to a shape.RGB code of the color is passed to the fill() function as an argument.

Then we create the shape of the ground, which is a rectangle using the **rect**() function.

And then we write the **pop**() function. This will end our shape creation. Our ground class is complete, now we can create the object of the class.

First, declare a variable as ground. Then in the setup() function create the object of the ground class and store that in this variable.

create the object of the class by using the new keyword and class name.

While creating a ground object, we need to provide the x, y positions and the width, height of the ground.

**display** the ground By calling the **ground.show()** function in the draw() function

Now we need to create the **rope** object and hang it on a point in the canvas, then we will create a fruit body and attach it with the rope. We will also add background images and images for the fruit to make our game look visually appealing.

**rope.js**

● In the constructor() we are taking two parameters **nlinks** - number of links and **pointA** - points of connection. Using **Composites.stack()** function we create the multiple rectangular bodies and store it in the rect variable. Using the **Composites.chain()** function we create the chain of the rectangles. And then using the **Constraints.create()** we add the constraints to the chain which connects all the bodies of the chain together like we have string in a necklace. We have the **break**() function which helps us to break the chain.It simply makes the rope body null.

For our code we are only going to use the rope.js to create the rope and **break**() function to break the rope when the user clicks on the cut button(which will be added in upcoming classes). Add rope.js in index.html To create a rope we need two pieces of important information, first is how long our rope will be, and where we want to hang our rope. To define the length of the rope, we define a number that creates those many sections in the rope. One section essentially is a rectangle, so you can imagine our rope to be multiple rectangles connected together. Next, we need the **point** where we are going to hang the rope, which will be a certain x and y **position** on the canvas.

define a variable named rope.

Then in the setup() function create an object from the Rope class.

The next step is to call the rope.show() function in the draw() function so that we can see the rope hanging.

create the fruit body and hang it with our rope.

in the setup function, we first create **fruit**\_**options** for the fruit body. In the options, we are only defining the density of the fruit as 0.001. Then we create a fruit body using the **Bodies.circle()** function. In the function, we need to provide the x and y positions and the radius of the fruit along with fruit\_options. Once we create the body, we need to add this body in the composite

a composite consists of multiple bodies within it. When we want multiple bodies to have the same properties such as shape and size and behave in a certain manner, we make a composite of these bodies. In our game, the rope we are creating is made up of multiple rectangles, hence we call it a composite. But we also have to add our fruit in the same composite. To add a body to the composite, we use the function,

Matter.Composite.add(name\_of\_composite, body\_to\_add).

Here the **composite** is a body of the Rope class, and we are adding the fruit to it.

Now we have the fruit body in the composite, but we need to display this fruit as well. To display this body, we will create a circle, using the ellipse function in the draw() function. For the x and y positions of the circle, we will pass the x and y position values from the fruit body.

Because we have not added any constraint between the fruit body and the rope. They are part of the same composite but not attached to each other with any constraint.

Now we need to write the code to hang the fruit with our rope. For this, we are going to use constraints. If you remember our rope is made up of various rectangles to be more specific 6 rectangles, which we specified while creating the rope object. We will create a constraint between the fruit body and the last rectangle of our rope.

We create the class for the constraint because in the future classes we will hang the fruit with multiple ropes. To make our code clean, we will keep all the code related to creating the constraint in the link.js file.

Create a new file named **link.js** and add this to the html file as source.

The constraint is going to be between 2 bodies, the last body (rectangle) of the rope and the fruit body. That is why in the link class when we write the constructor we need to keep 2 parameters as bodyA and bodyB. We want to connect the fruit body at the last rectangle of the rope. So we will create a variable to get the index of the last rectangle(or element, composite can be assumed like an array) That will be

var lastlink = bodyA.body.bodies.length-2

The last element will be 2 less than the length because the index always starts from 0, and we also added fruit in the composite that increases the length 1. So to get the last element index we need to subtract the 2 from the length. So the function bodyA is rope.body.bodies[lastlink].

bodyB is the fruit. We will create the constraint using the createConstraint() function of the matter.js library. To connect 2 bodies we also need to specify at which point, on the body of the constraint, is it going to be connected. For both bodies or we can say pointA and pointB, which are the X and Y positions so, we will write these as x:0 and y:0. We need to define two more parameters:

● The length of the constraint, which we also keep 0 for now.

● The stiffness, which we write as 0.01. This value of stiffness will prevent vibration of the constraint (Here vibration refers to the small movements in the rope). So that our constraint stays stable.

Once the constraint is complete, we will add this to our world. But before we run the code, we need to create the object of the link() class in the sketch.js file and pass the 2 bodies. To create the object, first declare a variable as var fruit\_con; In the setup function we will create the object and assign it to the fruit\_con variable; While creating the object of the class, we need to pass the rope and the fruit as the parameters in the Link() class object.

