In this class we are going to:

● Add animations to our bunny sprite.

● We will also detect the collision of the fruit with the bunny.

● We will also change the animation based on different conditions.

First, we will load the animation in the **preload**() function just like we loaded the images in the previous class. Animation consists of multiple images, if we play these images continuously it looks like an animation. In the preload() function we will load the animation with help of the **loadAnimation**() function, the argument for the function is going to be the image

We will play only two animations, for now, **blink** and **eat**, the bunny blinks its eyes, and once the fruit falls and collides with the bunny it will start eating, after which we will play the eating animation.

Once we loaded the animation, we needed to set some **properties** of the animation. First, we need to set both animations as **playing** equal to **true**. This will enable the play mode of the animation. This makes our animation when we run the code. If we set this to false, then our animation will not run until we specify that in the code.

Next for the eat animation we don’t want the animation to loop or play again and again, so to prevent that we will keep **eat.looping = false**.

We have loaded the animation, now we need to add this animation to our bunny sprite so that we can switch between animations. But we need to do one more thing before that, which is to set the **speed** of the animation. The computer tries to play the animation as fast as possible, but we want our animation to play a little slower so that we can see what is happening in the animation. To slow the speed of the animation we need to set a frame delay. This is going to be a number, the higher the number the slower will be the speed. We will set this for both the loaded animations, blink and eat. = **blink.frameDelay = 20;**

**eat.frameDelay 20**;

We can keep any value between 15 and 20, that will look good for our animation.

the **addAnimation**() function. In the parameters of the function, we will pass a string value as the first argument that will identify the animation, you can think of it as the **name of the animation**, then the second argument will be loading the animation. So the syntax will look like this:

**bunny.addAnimation(‘blinking’,blink);**

A string could be anything, but it is good practice to keep it relevant

We will also change the animation to blinking by using the **bunny.changeAnimation()** function

Because in the beginning, we want to play the blinking animation. Which means when our code runs, the bunny will be playing the blinking animation to show that the bunny is waiting for the fruit

Apart from the blinking animation, we need animation for the happy bunny and the bunny being sad. We will use the sad animation only when the bunny misses eating the fruit. Just like, we added the animation of blinking and eating. You need to add the animation for the happy and sad.

Apart from loading the animation, we need to set them as playing = true for playing the animations for sad and the eating bunny however, we don’t want the eating and sad animation to be played over and over again. So we set their looping as false

**set the animation speed**

**animate**() was one way to achieve it,

another concept to control the speed of the animation is called **frameDelay**.

As animations are created using frames; we can change the number for frameDelay.

**detect** the **collision** of the fruit with the bunny and then change the animations accordingly.

To detect the collision, you are going to use a very simple algorithm. Which is to find the **distance** between fruit and bunny.

As the fruit moves toward the bunny, the distance decreases. We can write the condition in such a way that if the distance is less than a certain value we can say both the objects have collided.

here are some important details which we have to pay attention to, otherwise our program will give us multiple errors.

The details are as follows:

● Once the fruit falls down, we need to remove the fruit from the world. But if we remove the fruit then we can not calculate the distance between the fruit and the bunny, so we need to put the condition in such a way, it will only calculate the distance if the fruit body exists.

● The second important thing is if we remove the fruit from the world then we can't draw it. Because we have deleted the body from the world, and while drawing we are referencing the x and y position of the fruit body. If a fruit body does not exist, and we are still drawing it on the canvas, then it will give us an error

We only want to show the fruit if its body exists. So we write the code as: **if(fruit!=null) { Show the fruit body. }**

This code will ensure that it will only show the fruit body, if it exists, if we delete the fruit body, it will not give us any error.

Now comes the tricky part, where we will create a function to detect the collision. Let’s first define the function as **collide(body,sprite)**, this will take two arguments one is the body second is the sprite.

In the function first, we need to check whether the body exists or not. The body here is the fruit body

So we will test this by using the **not equal to (!=)** operator, in the condition, we will write if(fruit!= null).

the Not equal to operator is used to test whether a variable is not equal to a certain value. Here we want to check if our fruit is not equal to null, then only we will proceed further to detect collision. If the fruit is null then we can not detect collision. Then in the parenthesis of this condition, we need to write the algorithm to detect the collision.

When 2 bodies have collide, they touch each other or come very close.

we can say that when the distance between 2 bodies is 0 we can say they have collided.

We will use the same logic here, we need to find the distance between the fruit and the bunny.

That we will do using the **dist**() function, which is an in-built function of the p5.js library.

This function needs four points:

● X position of first body or shape.

● Y position of first body or shape then,

● X position of the second body or shape.

● Y position of the second body or shape.

It will then calculate the distance between two points. We will need a variable to store this value, so we can declare a variable here and assign the dist() function to it.

Once we have the distance between the fruit and the bunny, it then becomes very easy. We just need to write a condition that if the distance is less than or equal to some value we say, it would cause a collision. In our case, we will set it as 80. If this condition is true then we need to remove the fruit from the world and make it null. Because when the fruit collides with the bunny it will eat the fruit, so it is supposed to disappear from the scene. For that, we will use the **World.remove()** function and then set the fruit null. We also need to return true as a **callback**. If this condition is not satisfied then we will simply return false. And the fruit will stay in the scene.

We will call this function in the **draw**() loop and if this function returns true we will change the animation to eating. We will also detect the collision with the ground. If the fruit falls on the ground, not colliding with the bunny. Then we will play the sad animation

In the coming classes we will add an air balloon that pushes the fruit in a direction, and we will also add multiple ropes to hold the fruit, where the user has to cut the rope in such a way that the fruit falls directly on the bunny. We will call the collide() function in the draw() loop. If the fruit collides with the bunny we will change the animation to eating and if the fruit collides with the ground we will change the animation to sad. Here our fruit may not collide with the ground, because we kept it directly on the bunny. But in future classes, we can have such circumstances.

We are calling the collide function two times, one for **fruit and bunny,** the other for **fruit** **and** **ground.body**. Based on the result of the collision of fruits with the bunny we will change the animation to happy and with the collision of fruit with the ground, it should be sad.