Topic	Inheritance
Class Description	**Students will add images and animations to the box model angry birds game which they developed in the last class. Students will also modify the code to use inheritance to write sub-classes which inherit properties and functions from their parent class. Students will connect this to the DRY principle studied in the earlier classes.**
Class	C25
Class time	45 mins
Goal	● Add images and animation to the rectangular boxes in the angry birds game. ● Use inheritance to write sub-classes which extend the properties of a parent class.
Resources Required	● Teacher Resources ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ● Student Resources ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen

Class structure	Warm Up Teacher-led Activity Student-led Activity Wrap up	5 mins 10 mins 20 mis 5 mins

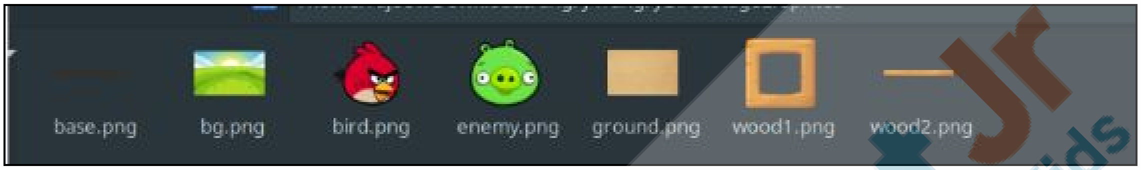
CONTEXT

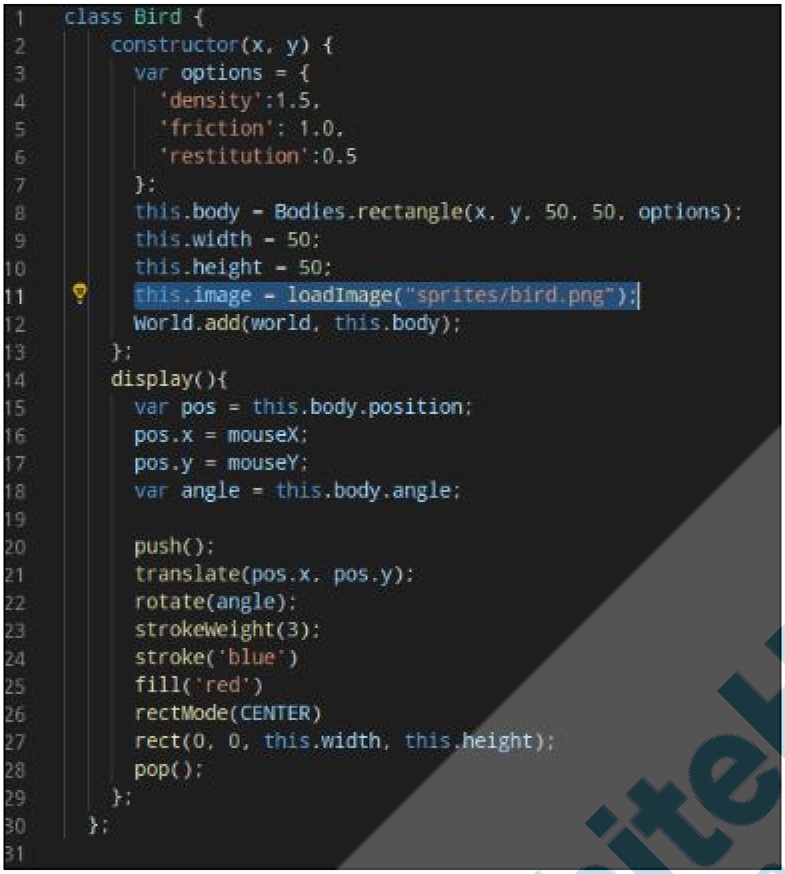
- **Review the concept of Class which we have covered in the earlier class.**
- **Each class has some properties and functions.**

Class Steps	Teacher Action	Student Action

Step 1: **Warm Up** **(5 mins)**	In the last few classes, we have learned about classes. Can you explain in your own words what a "class" is? What does a class contain? What is a class used for?	**ESR:** A class is a blueprint of an object. **ESR:** A class contains some properties (like width, height) and functions (like display()) **ESR:** A class is used to create one or more objects having the same properties and functions as defined in the class.
	Good effort! It is important to remember that each class has some properties and functions defined inside them. Each object which is made from the class has those properties and functions.	*Student listens.*
	I have an exciting quiz question for you! Are you ready to answer this question? Teacher click on the ? Quiz Time button on the bottom right corner of your screen to start the In-Class Quiz. A quiz will be visible to both you and the student.	**ESR:** **Yes**

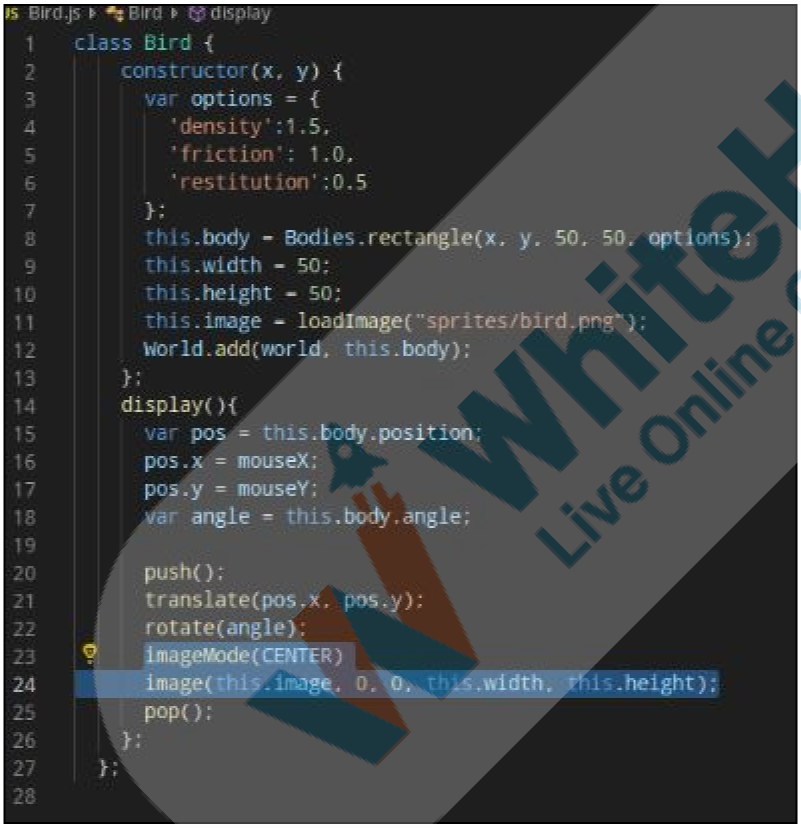
	Encourage the student to answer the quiz question. The student may choose the wrong option, help the student to think correctly about the question and then answer again. After the student selects the correct option, the **End Quiz** button will start appearing on your screen. Click the End quiz to close the quiz pop-up and continue the class.	
	In the last class we just had rectangular boxes for everything - for box, logs, birds, pigs, ground. In today's class, we will add images to our game.	-
Teacher Initiates Screen Share		
<u>CHALLENGE</u> ● **Add an image property to one of the classes and use it to make one of the game objects animated.** ● **Introduce the concept of inheritance and how a subclass can inherit the properties and functions of a parent class.** ● **Write a subclass which extends the properties and functions of a parent class.**		
Step 2: Teacher-led Activity (10 mins)	Before, we start - let's quickly revise the code from the previous class. *Teacher opens the link sent by the student to start live collaboration.*	*Student opens the code from the last class in VS Code. Enables live share and shares the link with the teacher.* *The student tries to explain broadly what they did in the*

		last class and what each block of code does.
	We have a new folder in our project now called - sprites. It contains all the images from the Angry Birds game. *Teacher opens the folder and shows them the different images.*	*The student observes and learns.*
 base.png bg.png bird.png enemy.png ground.png wood1.png wood2.png		
	Let's learn how to use these images and add them into our game. Let's start with the Bird class.	*Student observes and learns.*
	The Bird class blueprint has properties like body, width and height. Let's add an additional property to it called image. We will load the bird image in this property. *Teacher writes code for this.*	*Student observes and learns.*

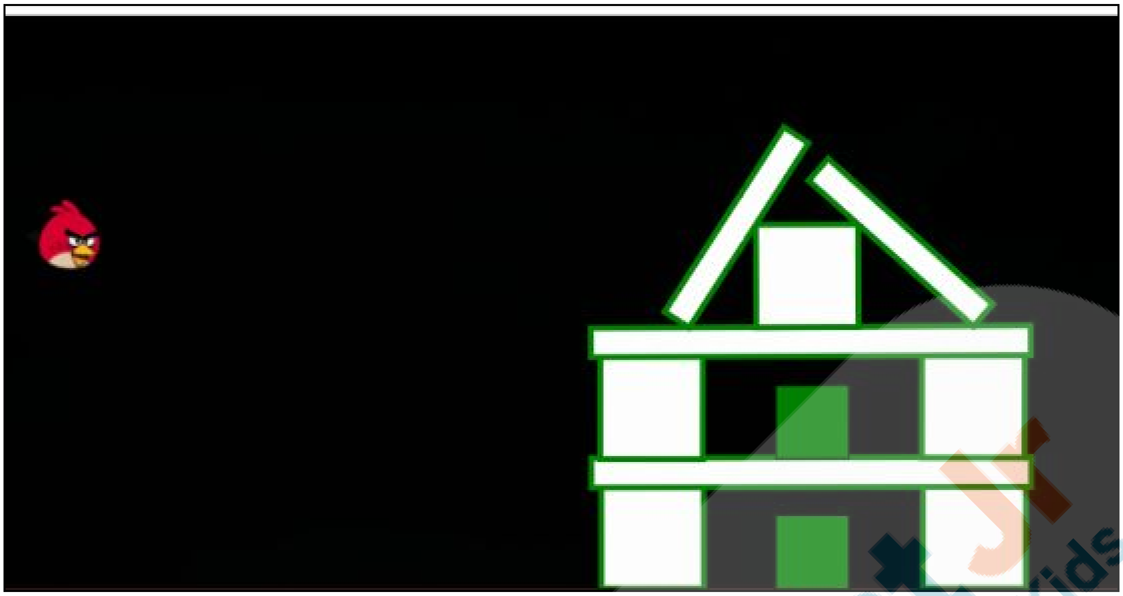


```
class Bird {
  constructor(x, y) {
    var options = {
      'density':1.5,
      'friction': 1.0,
      'restitution':0.5
    };
    this.body = Bodies.rectangle(x, y, 50, 50, options);
    this.width = 50;
    this.height = 50;
    this.image = loadImage("sprites/bird.png");
    World.add(world, this.body);
  };
  display(){
    var pos = this.body.position;
    pos.x = mouseX;
    pos.y = mouseY;
    var angle = this.body.angle;

    push();
    translate(pos.x, pos.y);
    rotate(angle);
    strokeWeight(3);
    stroke('blue')
    fill('red')
    rectMode(CENTER)
    rect(0, 0, this.width, this.height);
    pop();
  };
};
```

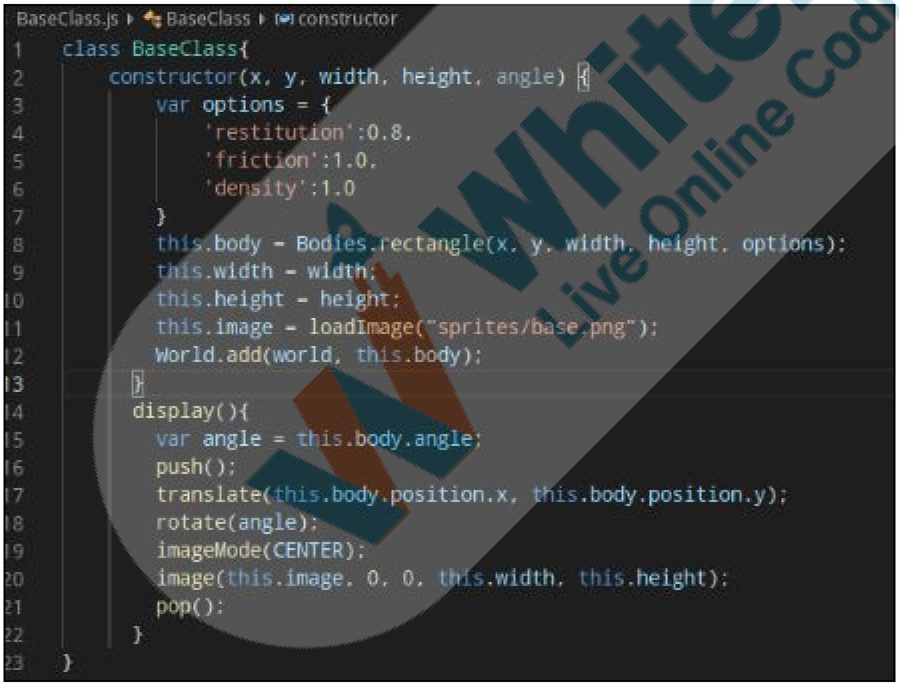
	Now, we don’t need to draw the rectangle any more for the bird. We want an image here. We can use the image() instruction instead of the rect() instruction. *Teacher shows the use of image() (from the second example) instruction in the documentation page for image(). **[Teacher Activity 2]*** - The first argument is for the image. - The second and third arguments are for the position. Here we have translated the position to where we want it to be. So we can use 0, 0 - The fourth and fifth are for the width	*Student observes and learns.*

	and height. We can use the width and height from the property of the class (defined in the constructor).	
	Teacher writes code to replace rect() with image(). *Teacher also replaces rectMode(CENTER) with imageMode(CENTER).* *Teacher runs the code.*	*Student observes and learns.*
		

```
class Bird {
    constructor(x, y) {
      var options = {
        'density':1.5,
        'friction': 1.0,
        'restitution':0.5
      };
      this.body = Bodies.rectangle(x, y, 50, 50, options);
      this.width = 50;
      this.height = 50;
      this.image = loadImage("sprites/bird.png");
      World.add(world, this.body);
    };
    display(){
      var pos = this.body.position;
      pos.x = mouseX;
      pos.y = mouseY;
      var angle = this.body.angle;

      push();
      translate(pos.x, pos.y);
      rotate(angle);
      imageMode(CENTER)
      image(this.image, 0, 0, this.width, this.height);
      pop();
    };
  };
```

		
	You can now replace all the other rectangular boxes with the images in the game. But before we do that, let's look at an important problem. Let's look at all the different classes we have created. Do you see several lines of code which are the same in all the classes? *Teacher opens the different class files and shows lines of code which are the same.*	*Student observes and learns.*
	Which important principle did we learn in earlier classes that is violated here?	**ESR:** DRY - Do Not Repeat Yourself Principle
	To avoid writing the same code for all the classes, in programming language we have a concept of a Parent / Base class and Children / Sub classes.	*Student listens and learns.*

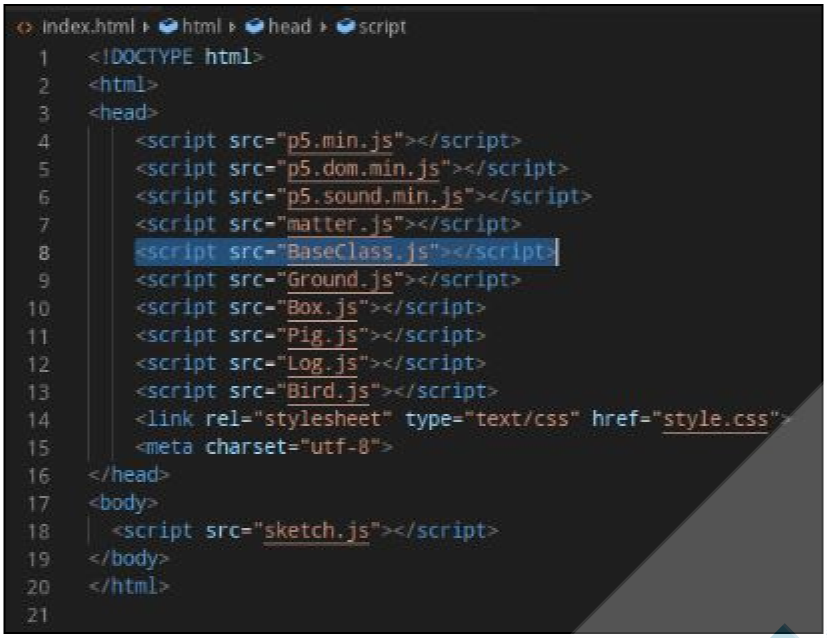
	Children/Sub classes created using Parent/Base class inherit all the properties and functions from the parent class.	
	Let's see how this works. Let's write a base class called BaseClass. *Teacher writes code to create a BaseClass.* Our Base object can have all the properties and functions which we had in the Bird class. We can choose any placeholder image for the BaseClass.	*Student observes and learns.*



BaseClass.js ▸ BaseClass ▸ constructor

```
class BaseClass{
    constructor(x, y, width, height, angle) {
        var options = {
            'restitution':0.8,
            'friction':1.0,
            'density':1.0
        }
        this.body = Bodies.rectangle(x, y, width, height, options);
        this.width = width;
        this.height = height;
        this.image = loadImage("sprites/base.png");
        World.add(world, this.body);
      }
      display(){
        var angle = this.body.angle;
        push();
        translate(this.body.position.x, this.body.position.y);
        rotate(angle);
        imageMode(CENTER);
        image(this.image, 0, 0, this.width, this.height);
        pop();
      }
}
```

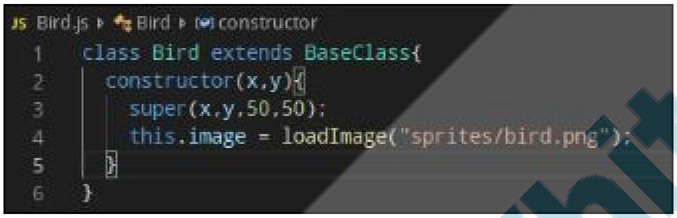
	Let's include the src of the BaseClass in the index.html file.	*Student observes and learns.*



```
<!DOCTYPE html>
<html>
<head>
    <script src="p5.min.js"></script>
    <script src="p5.dom.min.js"></script>
    <script src="p5.sound.min.js"></script>
    <script src="matter.js"></script>
    <script src="BaseClass.js"></script>
    <script src="Ground.js"></script>
    <script src="Box.js"></script>
    <script src="Pig.js"></script>
    <script src="Log.js"></script>
    <script src="Bird.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <meta charset="utf-8">
</head>
<body>
  <script src="sketch.js"></script>
</body>
</html>
```

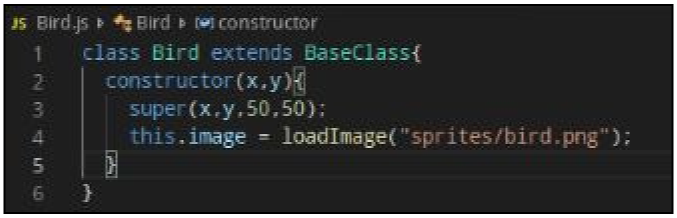
	Box and the Pig classes are very similar to the BaseClass. These classes can become the child class for this parent BaseClass and can inherit all the properties and functions. All the properties and functions of a parent class will be present in the child class. We can also add custom images and all other properties we want to our child class. Let's create a child BirdClass which inherits all the properties and functions of our BaseClass.	*Student listens and learns.*
	In BirdClass.js file, the teacher creates a child BirdClass by extending the parent BaseClass. We use "extends" to create a child class.	*Student observes and learns.*

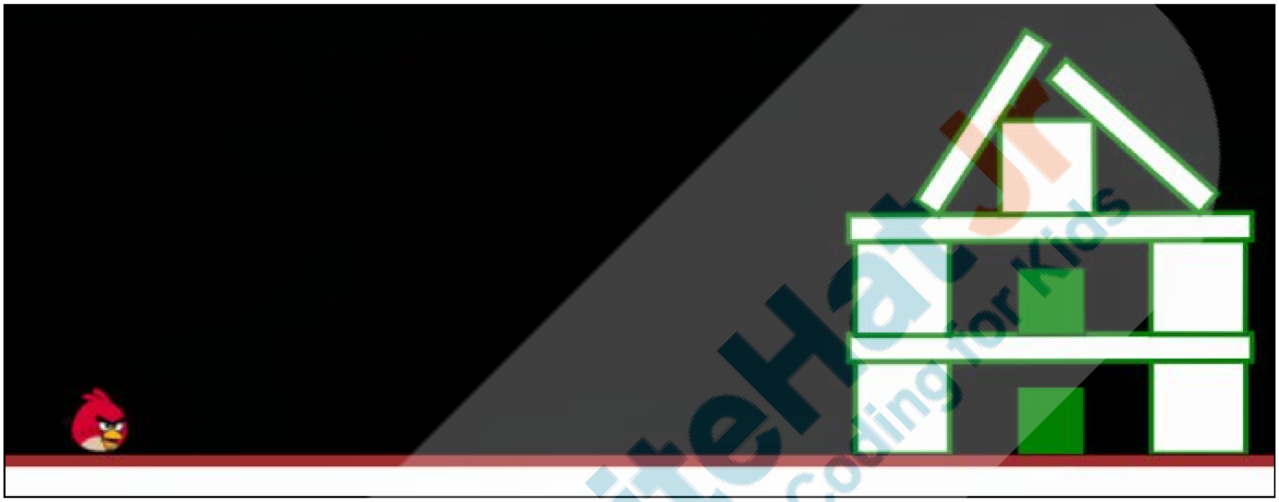
	Inside this child class, we can create a constructor which takes the same arguments that the Bird objects take. We use super() to transfer all the properties of the parent class to the child class through the parent class constructor. Here, we are passing width and height as 50, because the parent class constructor expects width and height. *Teacher writes code.*	



```
class Bird extends BaseClass{
  constructor(x,y){
    super(x,y,50,50);
    this.image = loadImage("sprites/bird.png");
  }
}
```

	Finally we want to add the bird image to the bird class constructor as well. You can do it inside the constructor. You can overwrite any of the properties of the parent class inside the child class and change it. *Teacher writes code to add the image property for the class.* *Teacher runs the code to see the output.* Notice how we didn't have to write the display function for the bird class because it is already defined in the	*Student observes and learns.*

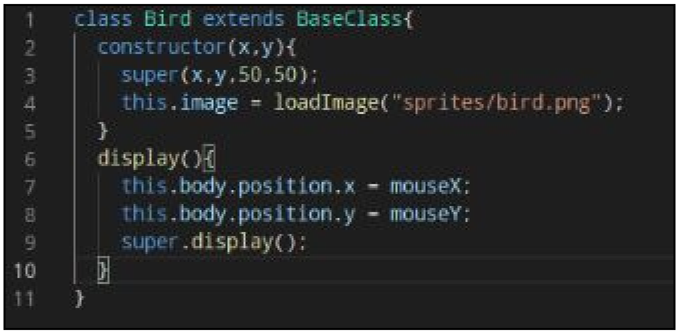
<table>
<tr><td></td><td>BaseClass and it inherits it from its parent class.</td><td></td></tr>
<tr><td colspan="3">


<pre>
class Bird extends BaseClass{
  constructor(x,y){
    super(x,y,50,50);
    this.image = loadImage("sprites/bird.png");
  }
}
</pre>

</td></tr>
<tr><td></td><td>What's the problem right now?

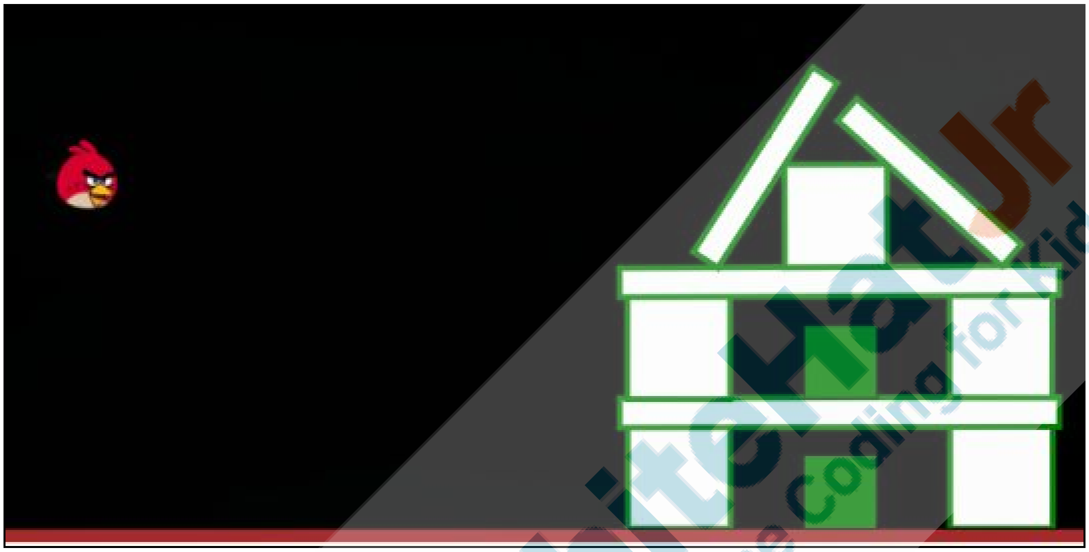
That is because the BaseClass display function is defined to do that. We can override it.

<em>Teacher shows how to override the display function of the base class by writing code for it. super.display() is used to refer to the parent class display function.</em>

<em>Teacher runs the code and shows the expected output.</em></td><td><strong>ESR:</strong>
The bird is made at a fixed position - given x and y. It does not move with the mouse.</td></tr>
</table>



```
class Bird extends BaseClass{
  constructor(x,y){
    super(x,y,50,50);
    this.image = loadImage("sprites/bird.png");
  }
  display(){
    this.body.position.x = mouseX;
    this.body.position.y = mouseY;
    super.display();
  }
}
```

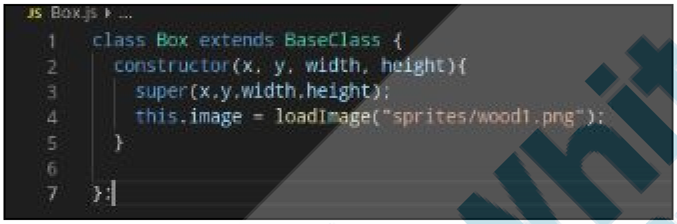


	Why don't you add images to all the other objects in the game by modifying their class blueprint. You can also extend other classes from the BaseClass as an activity to adhere to the principle of Do Not Repeat Yourself.	-
Teacher Stops Screen Share		
	Now it's your turn. Please share your screen with me.	
• **Ask Student to press ESC key to come back to panel** • **Guide Student to start Screen Share** • **Teacher gets into Fullscreen**		

ACTIVITY

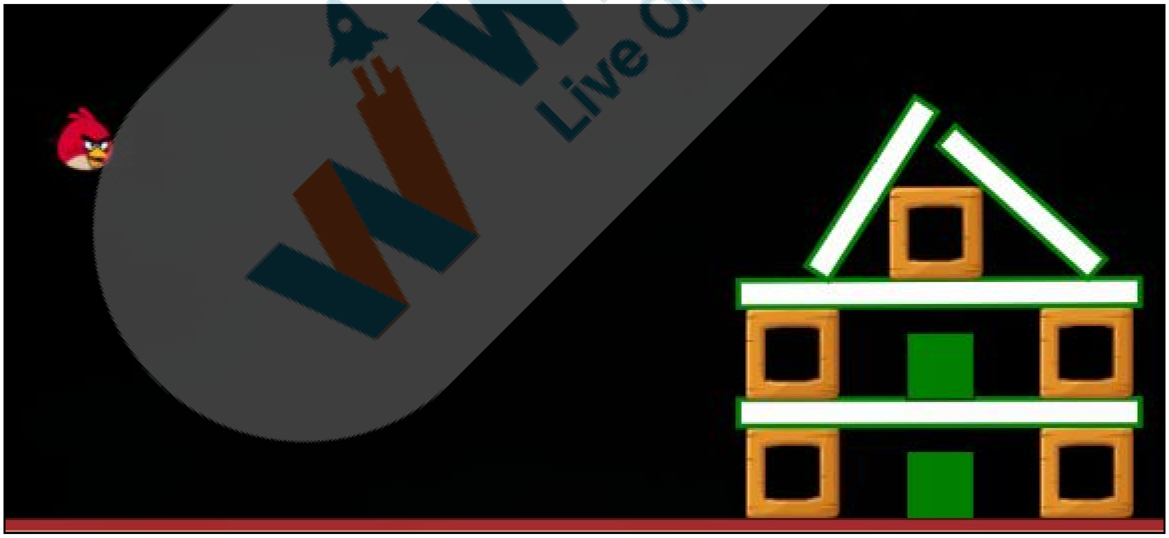
- **The student adds images and animations to all the objects in the game.**
- **The student writes sub classes which extend the properties and functions of a base parent class.**

Step 3: Student-Led Activity (15 min)	*Teacher guides the student to get the code downloaded from Github for* ***Student Activity 2.***	*Student downloads code for* ***Student Activity 2***
	Guide the student to write a Box subclass which extends the parent BaseClass. Inside this subclass, you can also add the image.	*Student writes code to create a Box Subclass which inherits from the BaseClass.*

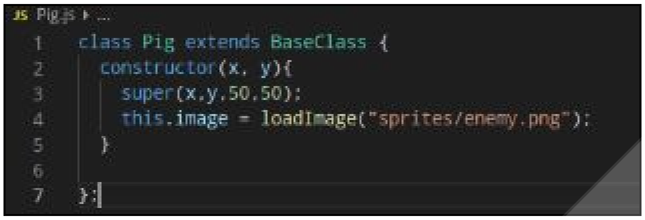


Box.js ▸ ...

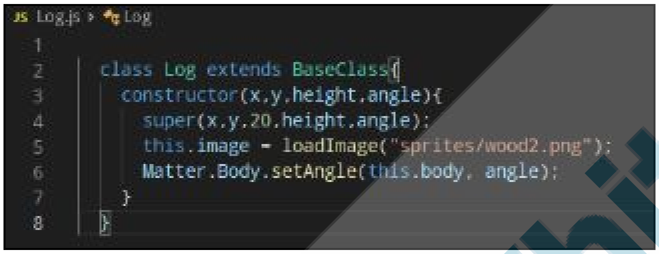
```
class Box extends BaseClass {
  constructor(x, y, width, height){
    super(x,y,width,height);
    this.image = loadImage("sprites/wood1.png");
  }

};
```



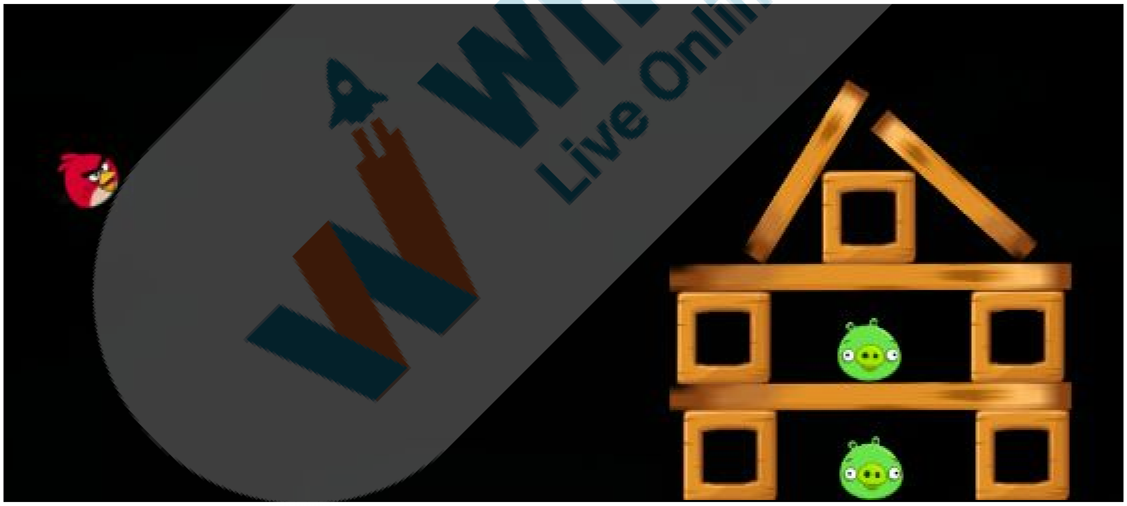
	Good! Now you know how to write a class which inherits from a Parent class. Awesome work! Now can you quickly do the same for the Pig class and the Log class. The *student does the same for both the Pig and Log Classes.*	*Student writes code to add images and extend the Pig and Log class from the BaseClass.*



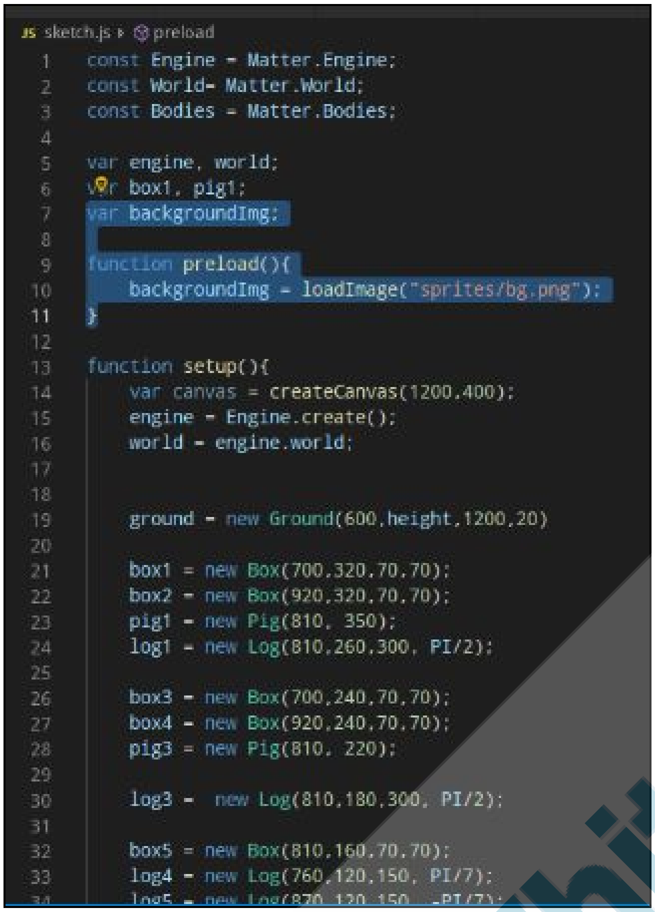
```
class Pig extends BaseClass {
  constructor(x, y){
    super(x,y,50,50);
    this.image = loadImage("sprites/enemy.png");
  }

};
```



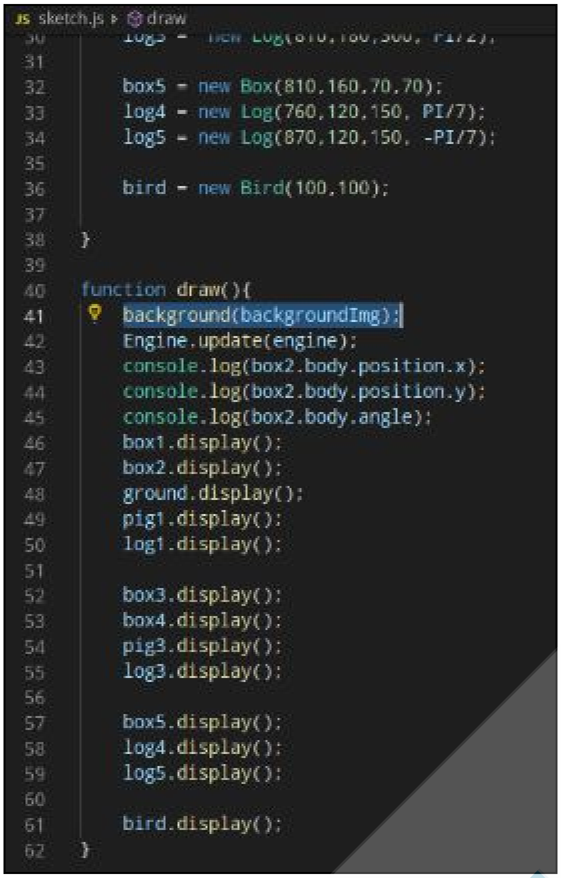
```

class Log extends BaseClass{
  constructor(x,y,height,angle){
    super(x,y,20,height,angle);
    this.image = loadImage("sprites/wood2.png");
    Matter.Body.setAngle(this.body, angle);
  }
}
```



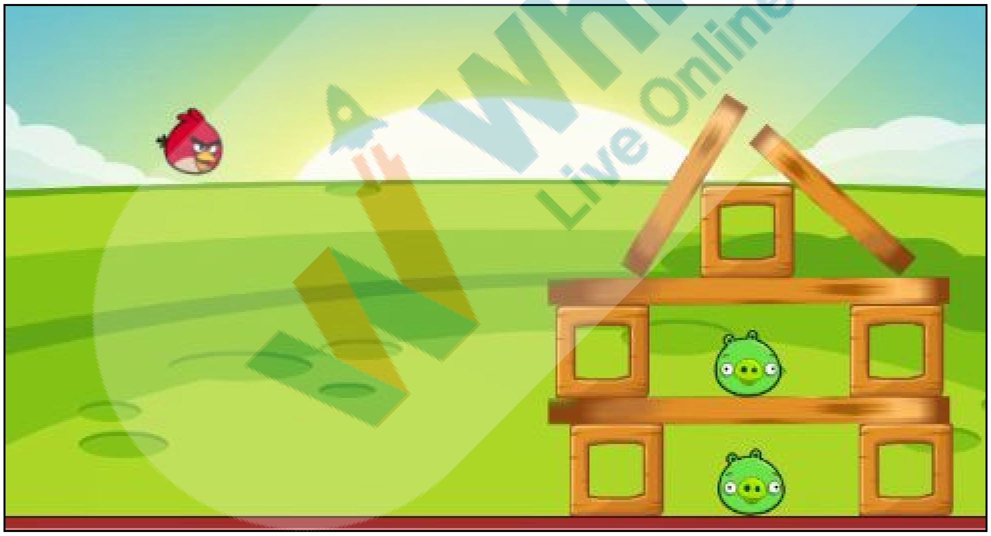
	Good work! The only thing missing now is the background image. Let's add that in our sketch file.	*Student writes code to load the background image and then add it to the sketch file.*



```
const Engine = Matter.Engine;
const World= Matter.World;
const Bodies = Matter.Bodies;

var engine, world;
var box1, pig1;
var backgroundImg;

function preload(){
    backgroundImg = loadImage("sprites/bg.png");
}

function setup(){
    var canvas = createCanvas(1200,400);
    engine = Engine.create();
    world = engine.world;


    ground = new Ground(600,height,1200,20)

    box1 = new Box(700,320,70,70);
    box2 = new Box(920,320,70,70);
    pig1 = new Pig(810, 350);
    log1 = new Log(810,260,300, PI/2);

    box3 = new Box(700,240,70,70);
    box4 = new Box(920,240,70,70);
    pig3 = new Pig(810, 220);

    log3 =  new Log(810,180,300, PI/2);

    box5 = new Box(810,160,70,70);
    log4 = new Log(760,120,150, PI/7);
    log5 = new Log(870,120,150, -PI/7);
```



```
    log3 =  new Log(810,180,300, PI/2);

    box5 = new Box(810,160,70,70);
    log4 = new Log(760,120,150, PI/7);
    log5 = new Log(870,120,150, -PI/7);

    bird = new Bird(100,100);

}

function draw(){
    background(backgroundImg);
    Engine.update(engine);
    console.log(box2.body.position.x);
    console.log(box2.body.position.y);
    console.log(box2.body.angle);
    box1.display();
    box2.display();
    ground.display();
    pig1.display();
    log1.display();

    box3.display();
    box4.display();
    pig3.display();
    log3.display();

    box5.display();
    log4.display();
    log5.display();

    bird.display();
}
```



	Amazing, we have a good portion of our game ready! In the next classes, we will build the slingshot to shoot the pigs. Pigs, Be Aware! Angry Birds are coming.	-
Teacher Guides Student to Stop Screen Share		
FEEDBACK ● **Encourage the student to make reflection notes in markdown format.** ● **Complement the student for her/his effort in the class.** ● **Review the content of the lesson.**		
Step 4: **Wrap-Up** **(5 mins)**	Before, finishing off the class, can we quickly review what we have learned in today's class?	**ESR:** - We learned how to load images and animations into our game using image() instruction. - We learned the concept of class inheritance and how child class can be created from a parent class. How the child class inherits all the properties and functions of a parent class. - We learned how to override the parent class and add extra properties and functions to our child class.
	Amazing! How confident do you feel to create your own game world and add your	**ESR:** varied

	own characters to your game like the angry birds game we are making?	
	You get Hats Off for your excellent work! I am eagerly waiting for the next class to progress in this game. We are so close to finishing this game up!!	Make sure you have given at least 2 Hats Off during the class for: Creatively Solved Activities +10 Great Question +10 Strong Concentration +10
Project Overview	**CRUMPLED BALLS - 2** **Goal of the Project:** In Class 25, you learnt how to assign images to bodies created by changing the blueprint of the class. In this project, you will have to practice and apply what you have learnt in the class and modify the blueprints of the objects in the Crumpled Ball Game. ** This project is dependent on concepts covered in Project 24. Please complete project 24 before attempting this project. **Story:** You want to inculcate the habit of throwing the waste in the trash bin in young individuals and help keep your city clean. So you have decided	*Student engages engages with the teacher over the project.*

	to create a simple game of throwing crumpled paper balls in a waste paper basket. I am very excited to see your project solution and I know you will do really well. Bye Bye!	
Teacher Clicks		
Additional Activities	*Encourage the student to write reflection notes in their reflection journal using markdown.* Use these as guiding questions: ● What happened today? - Describe what happened - Code I wrote ● How did I feel after the class? ● What have I learned about programming and developing games? ● What aspects of the class helped me? What did I find difficult?	*Student uses the markdown editor to write her/his reflection as a reflection journal.*

Activity	Activity Name	Links
Teacher activity 1	Github Link for teacher activity	https://github.com/whitehatjr/angryBirdsStage2TeacherActivity
Teacher Activity 2	image() reference	https://p5js.org/reference/#/p5/image
Teacher Activity 3	Reference Link	https://github.com/whitehatjr/angryBirdsStage2
Student Activity 1	image() reference	https://p5js.org/reference/#/p5/image
Student Activity 2	Github Link for student activity	https://github.com/whitehatjr/angryBirdsStage2StudentActivity