WhiteHat Jr
Live online Coding for Kids

Topic	A-Frame PHYSICS SYSTEM	
Class Description	Students learn to use physics in A-Frame VR. Students will learn about the static and dynamic body of the physics system.	
Class	C155	
Class time	45 mins	
Goal	• Learn how to add physics in A-Frame VR. • Learn to make bodies static and dynamic. • Learn to control the physics system bodies in A-Frame VR.	
Resources Required	• Teacher Resources ○ Visual Studio Code Editor ○ laptop with internet connectivity ○ earphones with mic ○ notebook and pen • Student Resources ○ Visual Studio Code Editor ○ laptop with internet connectivity ○ earphones with mic ○ notebook and pen	
Class structure	Warm-Up Teacher-led Activity Student-led Activity Wrap-Up	05 mins 15 mins 20 mins 05 mins

WARM-UP SESSION - 05 mins

CONTEXT

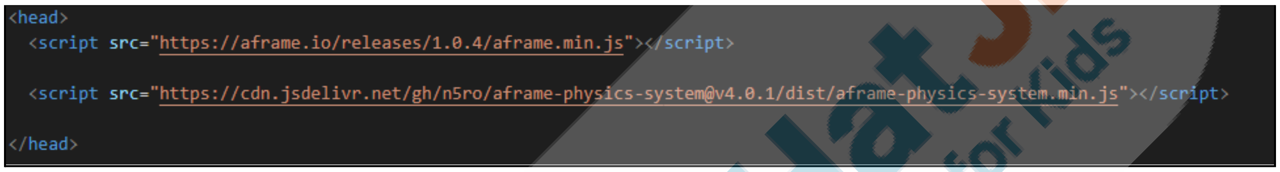
- **Physics in A-Frame VR**

Teacher starts slideshow from slides 1 to 9 Refer to speaker notes and follow the instructions on each slide.	
Activity details	**Solution/Guidelines**
Hi, how have you been? Are you excited to learn something new? ***Run the presentation from slide 1 to slide 3.*** **The following are the warm-up session deliverables:** ● Reconnect with previous class topics. ● Warm-Up quiz session.	**ESR**: Varied Response. Click on the slide show tab and present the slides.
Q&A Session	
Question	**Answer**
To create a ring target element through the component, we first need to _______. A. make a circular component B. make a cylindrical component C. register the component in A-Frame D. define the shape	**C**
How did we create multiple rings? A. by calling createRing function inside the loop B. by defining createRing function inside the loop C. by defining createRing function inside the loop D. by creating a separate library for creating rings	**A**
Continue the warm-up session	
Activity details	**Solution/Guidelines**

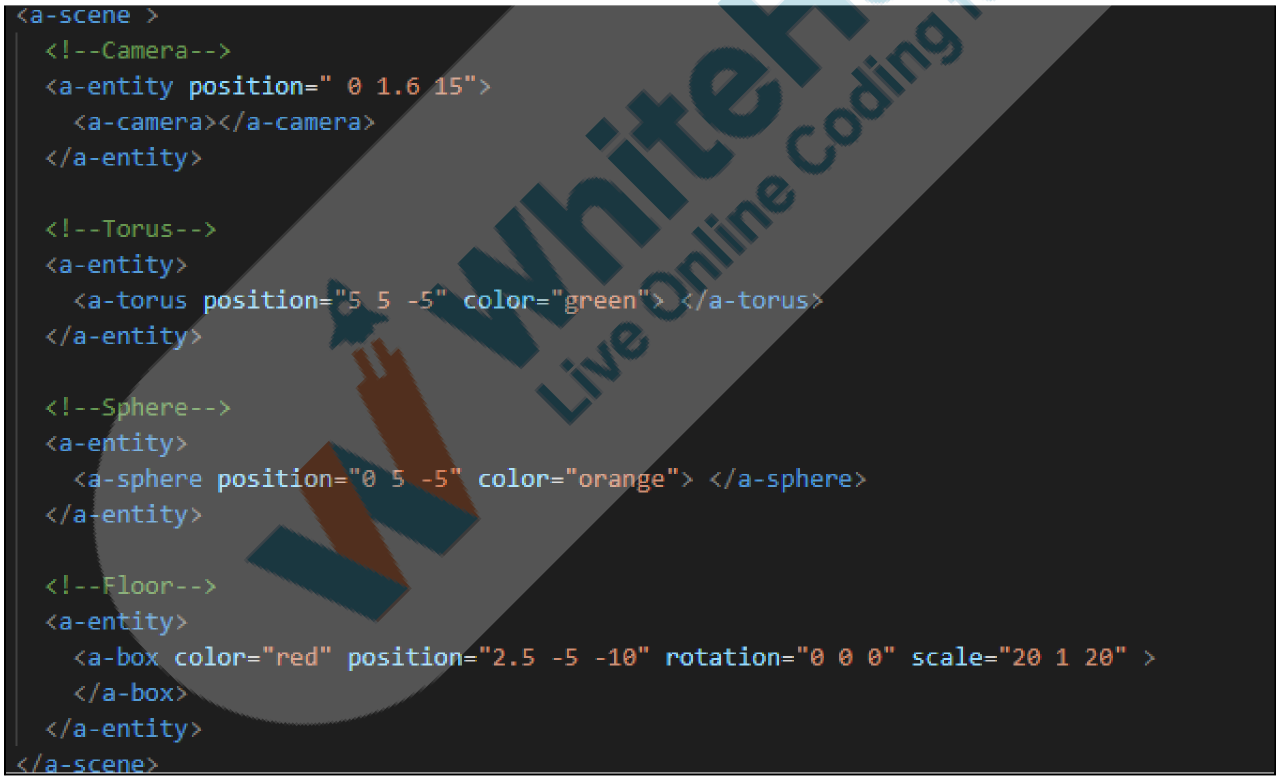
Run the presentation from slide 4 to slide 12 to set the problem statement. **The following are the warm-up session deliverables:** ● **A-Frame Physics System** ● **Static and dynamic bodies**		
Teacher ends slideshow		
Teacher Initiates Screen Share		
TEACHER-LED ACTIVITY - 15 mins		
<u>CHALLENGE</u> ● **Create a static and dynamic body in the A-Frame scene using the A-Frame physics system.**		
Teacher starts slideshow for slide 13 to 16		
Step 2: **Teacher-led Activity** **(15 min)**	Let's start by first understanding the physics system in A-Frame VR. In the flight simulation we need to know when the flight entity is colliding with other entities like two objects interact with each other in the real world. What do you think is important for any two objects to interact with each other in the real world? Yes, exactly!	 **ESR:** Any two objects must follow physics laws in nature like gravity, collision etc.

	Do you remember how we implemented physics in earlier classes? Amazing! ***[Teacher Activity 1]*** For this, A-Frame provides **components** which can be attached to the entities to make them physical bodies in the 3D scene. These components are a part of the aframe physics system library of their own- "**aframe-physics-system**" which is built using CANNON.js, the JavaScript physics engine library, to render physics-based 3D scenes. Link: https://cdn.jsdelivr.net/gh/n5ro/aframe-physics-system@v4.0.1/dist/aframe-physics-system.min.js To understand how the physics components work, let's include this A-Frame library in the index.html <head> tag. *<The teacher includes library link in index.html>*	**ESR:** We used a library called matter.js which had predefined physics functionalities for the bodies. .

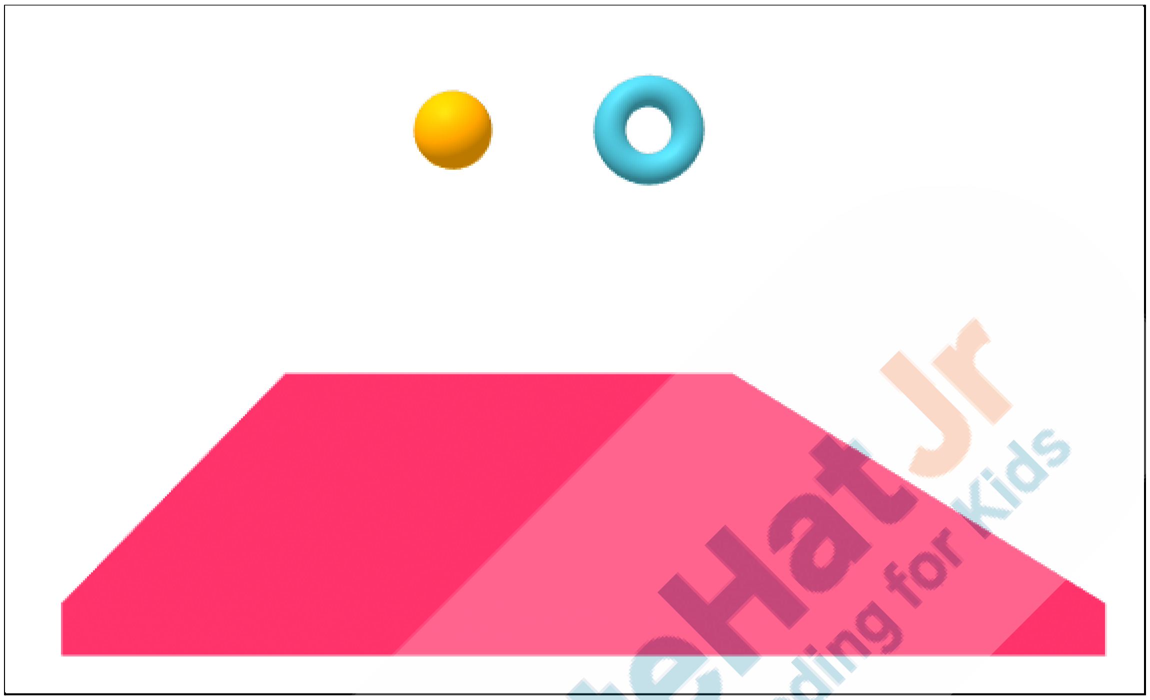
	Now let’s quickly add a few entities in the scene. *<The teacher codes to add a box, sphere, torus in the A-Frame scene.>* *<Set the position and rotation to get the output as below.>*	



```
<head>
  <script src="https://aframe.io/releases/1.0.4/aframe.min.js"></script>

  <script src="https://cdn.jsdelivr.net/gh/n5ro/aframe-physics-system@v4.0.1/dist/aframe-physics-system.min.js"></script>

</head>
```



```
<a-scene >
  <!--Camera-->
  <a-entity position=" 0 1.6 15">
    <a-camera></a-camera>
  </a-entity>

  <!--Torus-->
  <a-entity>
    <a-torus position="5 5 -5" color="green"> </a-torus>
  </a-entity>

  <!--Sphere-->
  <a-entity>
    <a-sphere position="0 5 -5" color="orange"> </a-sphere>
  </a-entity>

  <!--Floor-->
  <a-entity>
    <a-box color="red" position="2.5 -5 -10" rotation="0 0 0" scale="20 1 20" >
    </a-box>
  </a-entity>
</a-scene>
```



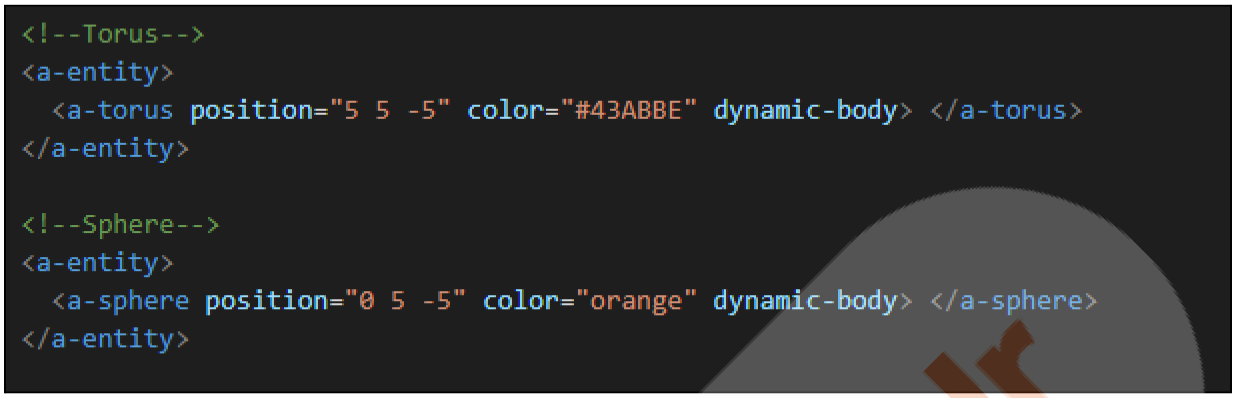
Now, we can use two components of this A-Frame library which can initiate the physics simulation in the scene:

- **static-body** component; and
- **dynamic-body** component.

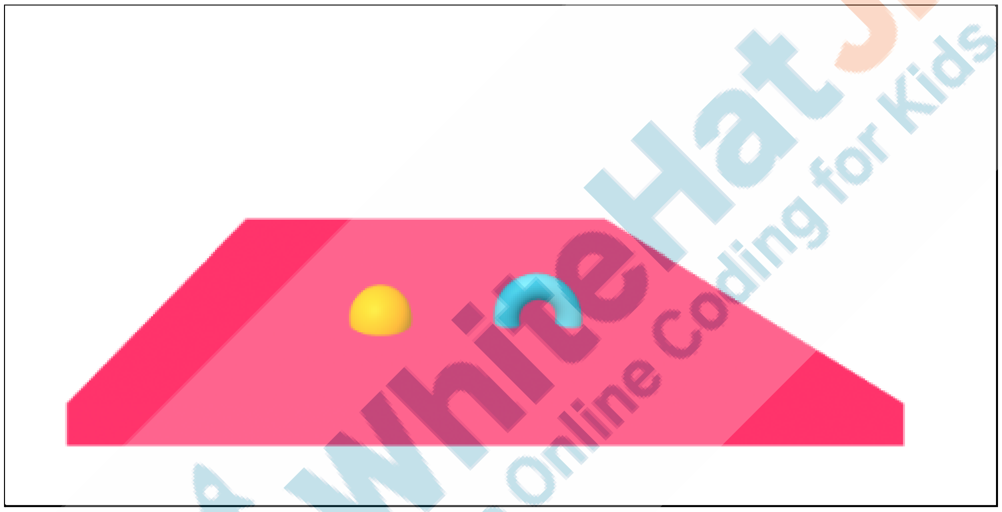
static-body: These are fixed-position objects (or animated objects). Static bodies are not affected by gravity and collisions, but these can interact with other objects in the scene.

dynamic-body: These are freely-moving objects. Dynamic bodies have mass; they fall if gravity is enabled, and bounce off or collide with other objects.

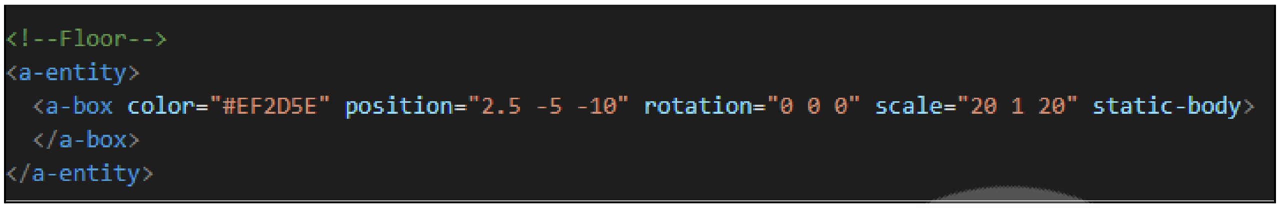
	The dynamic-body and static-body components can be attached to any <a-entity> that contains 3D objects. In general, <a-scene> will have at least one static-body for the ground, and one or more dynamic-body for the player to interact with. Now, let's attach these components in our scene. Can you tell me which entity I should make static or dynamic? Let's see what happens if I make the torus and the sphere entity dynamic. *<The teacher adds the dynamic-body component to the torus and the sphere entity and shows the result.>* Did you notice as soon as I made the torus dynamic it started falling down? Can you tell me why? Yes. That's impressive!	**ESR**: Varied. **ESR:** Yes! **ESR**: Because by default gravity is enabled.



```
<!--Torus-->
<a-entity>
  <a-torus position="5 5 -5" color="#43ABBE" dynamic-body> </a-torus>
</a-entity>

<!--Sphere-->
<a-entity>
  <a-sphere position="0 5 -5" color="orange" dynamic-body> </a-sphere>
</a-entity>
```



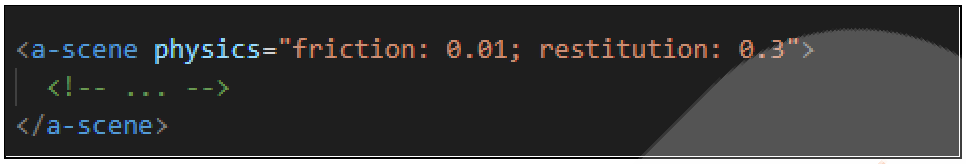
	But you must have noticed that the torus passes through the box, can you tell me why? Perfect! Let’s make the box a static body and see the difference. *<The teacher adds the static-body component to the box entity and shows the result.>*	**ESR**: Because the box must be static, so that it does not let the torus and the sphere to pass through it.



```
<!--Floor-->
<a-entity>
  <a-box color="#EF2D5E" position="2.5 -5 -10" rotation="0 0 0" scale="20 1 20" static-body>
  </a-box>
</a-entity>
```

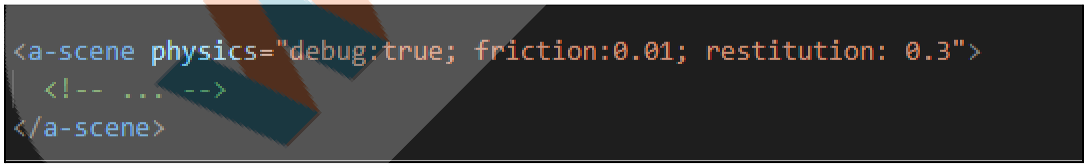
	You can now see the torus and sphere bouncing off the box for some time and then come to rest instead of passing through the box. Now, in the real world when objects/bodies come in contact there is always some friction, restitution (and other physical properties) between those two bodies. Do you remember we have discussed friction and restitution properties in the earlier classes? Can you tell me what are these? Awesome! In A-Frame these properties of physics can be configured using the physics component. Let's set these properties globally to all the physics bodies in the scene.	**ESR**: Yes! **ESR**: Friction is a force between any bodies in contact; restitution is the property which tells about the bounciness of the body.

	<The teacher adds the friction and restitution property of the physics component in the <a-scene> tag.>	

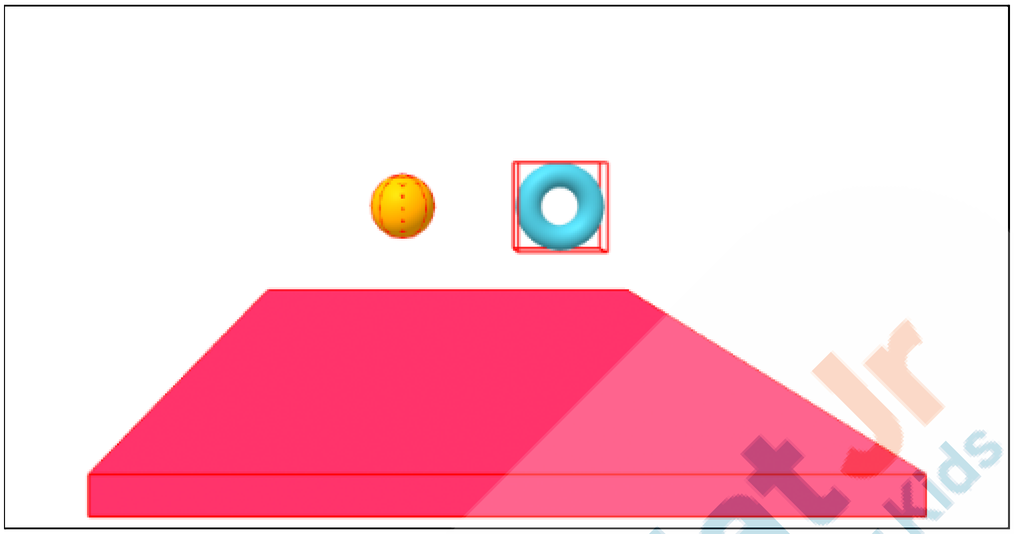


```
<a-scene physics="friction: 0.01; restitution: 0.3">
  <!-- ... -->
</a-scene>
```

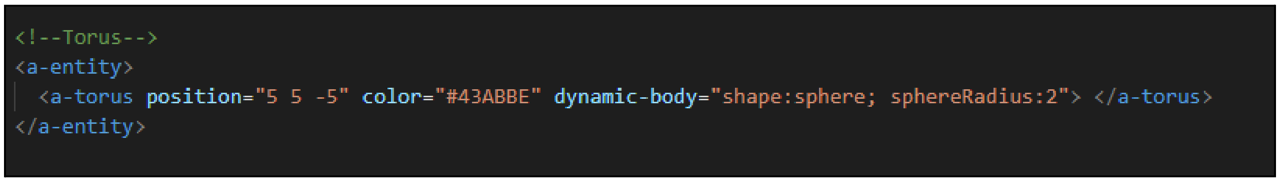
	Do you remember there's always some invisible shape surrounding the physics body to detect if they are colliding with any other physics body? In A-Frame also we see those shapes and modify them. To the colliding shape we can use the "debug" property and set it to true. *<Teacher add the debug property of the physics component and shows the output.>*	**ESR**: Yes!



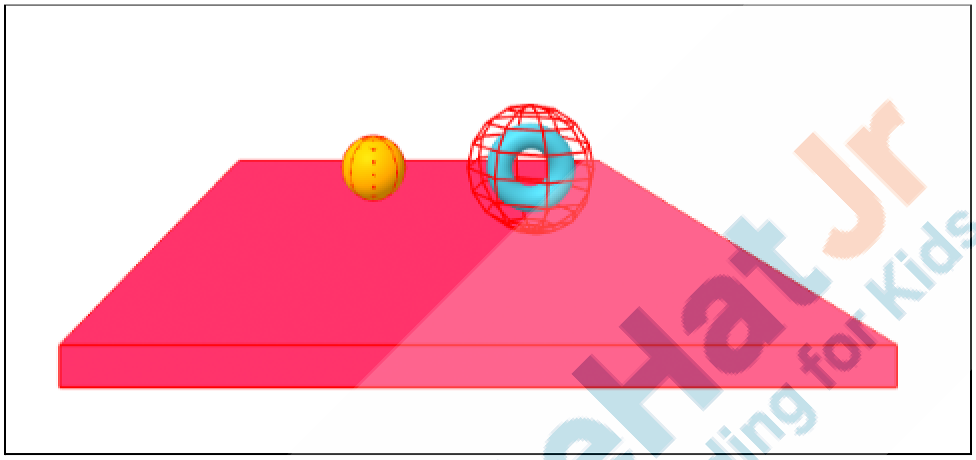
```
<a-scene physics="debug:true; friction:0.01; restitution: 0.3">
  <!-- ... -->
</a-scene>
```



	The default shape of the collider comes automatically based on the object. The default shape of the collider can be changed with the help of the shape **property** of the dynamic and static body component. Let's modify the shape around the torus from box to sphere. For this we can set the shape to a **sphere**. And if we want to modify the radius of the collider sphere we can use **sphereRadius** property. *<The teacher adds the shape property of the dynamic-body component of the torus entity and shows the output.>*	



```
<!--Torus-->
<a-entity>
  <a-torus position="5 5 -5" color="#43ABBE" dynamic-body="shape:sphere; sphereRadius:2"> </a-torus>
</a-entity>
```



	Now that we know how to add physics simulation in the 3D scene, let’s add it in our flight simulation. Are you excited?	**ESR**: Yes!

Teacher Stops Screen Share

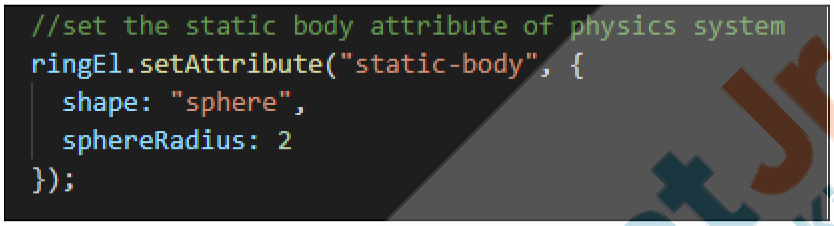
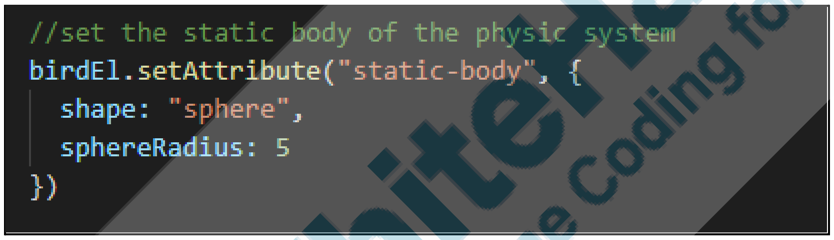
	Now it's your turn. Please share your screen with me.	

STUDENT-LED ACTIVITY - 20 mins

- **Ask the student to press the ESC key to come back to the panel.**
- **Guide the student to start screen share.**
- **Teacher gets into fullscreen.**

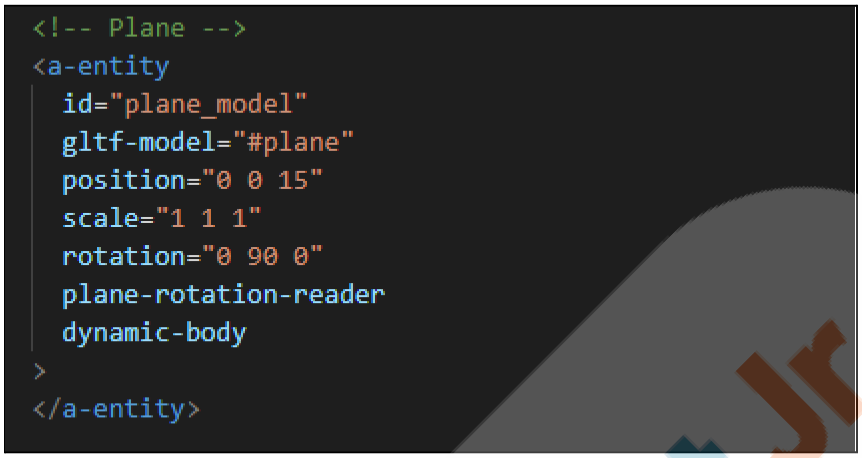
ACTIVITY ● **Set the attributes of the gLTF models to static and dynamic body** ● **Write a component to detect collision between entities.**		
Teacher starts slideshow from slides 13 to 16 Refer to speaker notes and follow the instructions on each slide.		
Step 3: Student-Led Activity (20 min)	*The teacher guides the student to open the code from the previous class.* *[Student Activity 1]*	*The student opens the code from the previous class.*
	Now what do we need to do to add the physics simulation in our flight simulation? *Guide the student to add the physics system library.*	**ESR**: Add the A-Frame physics system library.
	Now, we need to make entities as static-body and dynamic-body. Which one do you think will be static bodies? Yes. These entities can be static as we do not want to move them in the scene. Since these entities are created through components, how can we set the static-body component of the rings and birds?	**ESR**: Rings and the birds. **ESR**: We will use the **setAttribute()** method.

	Yes. Amazing! *Guide the student to set the static-body component and its shape properties.*	

	
	`//set the static body attribute of physics system`
	`ringEl.setAttribute("static-body", {`
	`  shape: "sphere",`
	`  sphereRadius: 2`
	`});`
	
	
	`//set the static body of the physic system`
	`birdEl.setAttribute("static-body", {`
	`  shape: "sphere",`
	`  sphereRadius: 5`
	`})`	

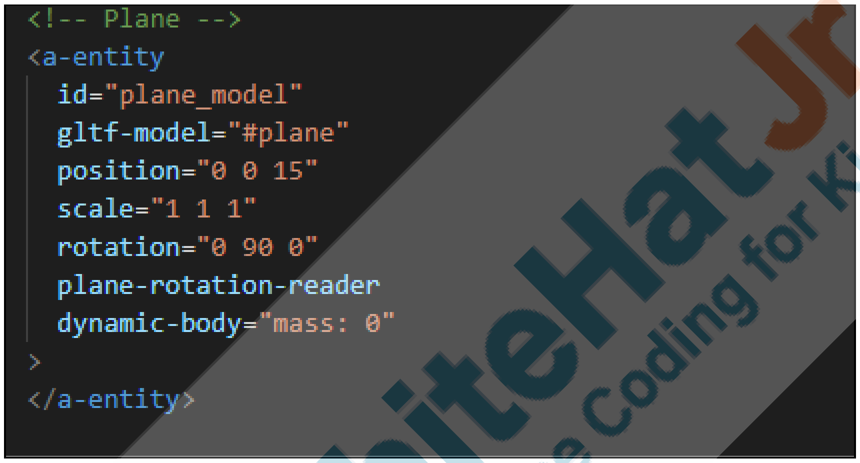
	Now what's next? We need one dynamic body in the scene. Since our flight is going to interact with the rings and the birds, we'll set the flight as the dynamic-body.	**ESR**: Varied.

WhiteHat Jr
Live online Coding for Kids



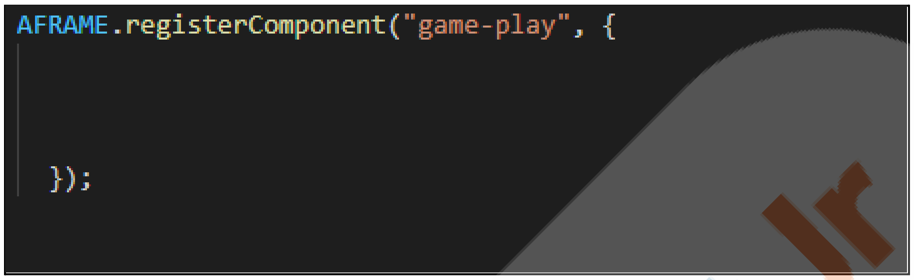
```
<!-- Plane -->
<a-entity
  id="plane_model"
  gltf-model="#plane"
  position="0 0 15"
  scale="1 1 1"
  rotation="0 90 0"
  plane-rotation-reader
  dynamic-body
>
</a-entity>
```

	Now as soon as we set the plane as a dynamic body it falls down because of gravity. But we do not want the flight to fall down, what can be done to prevent this? Do you know why everything falls down because of gravity in the real world? The reason why everythings falls down is because every object has some mass. So if we don't want the flight to fall down we can set its mass as 0. This will let the flight float at a particular position.	**ESR**: Varied. **ESR**: Varied.

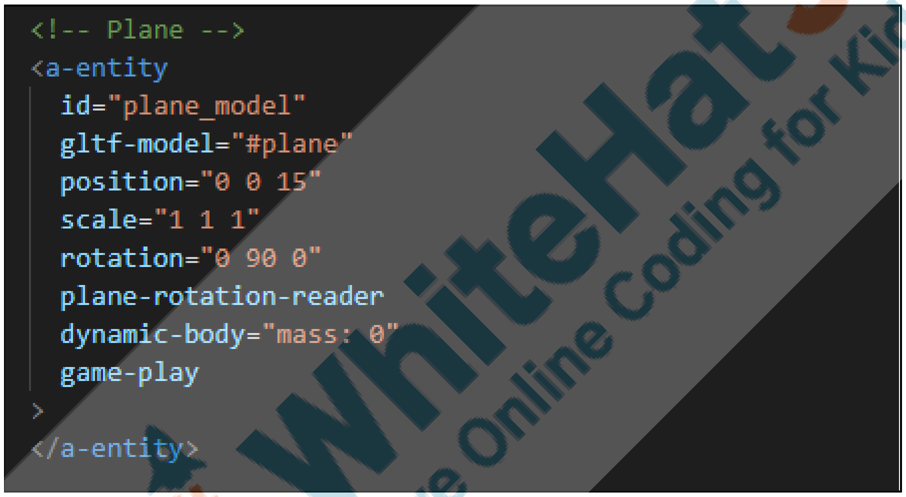
	In A-Frame, dynamic-body and static-body components have **mass** property. *Guide the student to add the mass property of the dynamic-body component.*	
 ```<!-- Plane --> <a-entity id="plane_model" gltf-model="#plane" position="0 0 15" scale="1 1 1" rotation="0 90 0" plane-rotation-reader dynamic-body="mass: 0" > </a-entity>```		
	Now, what's next? For this, we can use the **collide** event to detect the collision between these entities. Let's write a "game-play" component to add the "collide" event. *Guide the student to create a **Game.js** file and add it to index.html*	**ESR**: We need to detect when the flight collides with the rings and birds.

```
<!-- Plane -->
<a-entity
  id="plane_model"
  gltf-model="#plane"
  position="0 0 15"
  scale="1 1 1"
  rotation="0 90 0"
  plane-rotation-reader
  dynamic-body="mass: 0"
>
</a-entity>
```

	Guide the student to register a ***game-play*** *component and attach it to the plane entity.*	

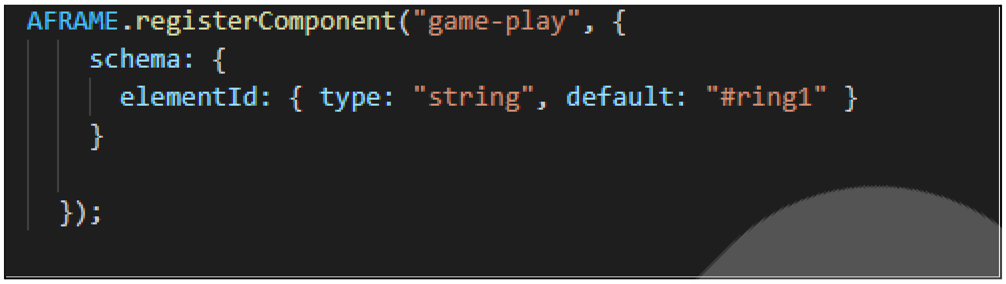


```
AFRAME.registerComponent("game-play", {



 });
```



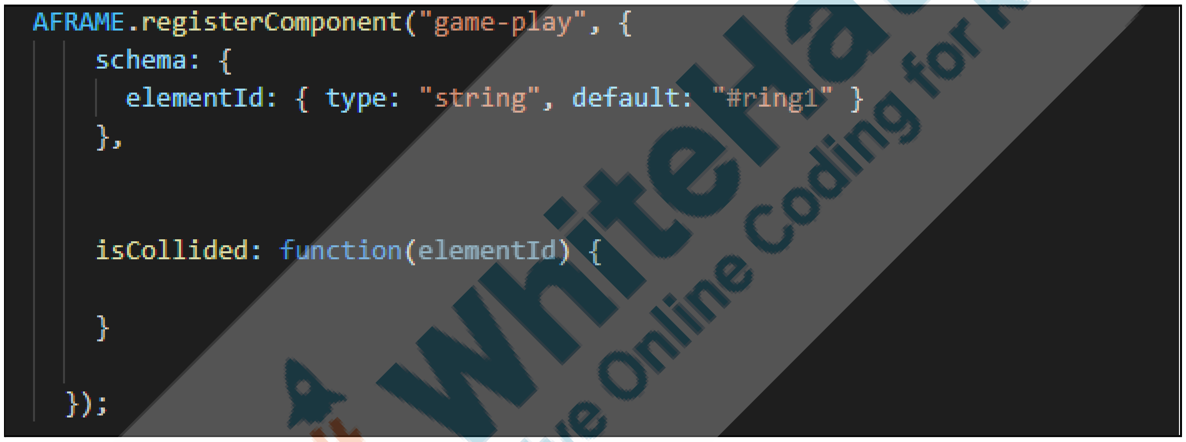
```
<!-- Plane -->
<a-entity
  id="plane_model"
  gltf-model="#plane"
  position="0 0 15"
  scale="1 1 1"
  rotation="0 90 0"
  plane-rotation-reader
  dynamic-body="mass: 0"
  game-play
>
</a-entity>
```

	Now, let's take a variable in the schema which will be used to set the element id of the entity to which the component will be attached. *Guide the student to take a variable* ***elementId*** *and set its* ***type as string*** *and* ***default value as "#ring1".***	



```
AFRAME.registerComponent("game-play", {
    schema: {
      elementId: { type: "string", default: "#ring1" }
    }

  });
```

	We can now define our own function to **isCollided()** and take the argument as **elementId.**	

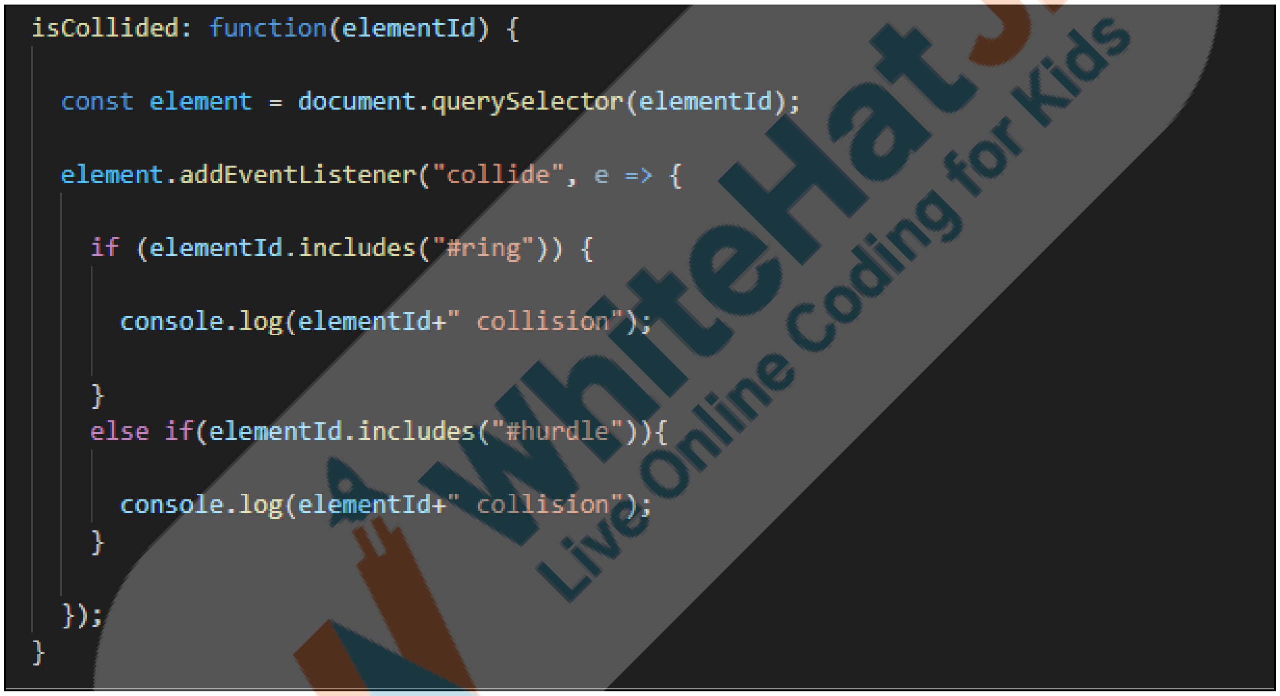


```
AFRAME.registerComponent("game-play", {
    schema: {
      elementId: { type: "string", default: "#ring1" }
    },


    isCollided: function(elementId) {

    }
  });
```

	Now we can add an event listener to trigger a "**collide**" event. For this we need to select on which element the event can be fired. Till now we have been triggering events on the window. But we can select a particular element using **querySelector()** and then add a listener on that element using **element.addEventListener()**.	

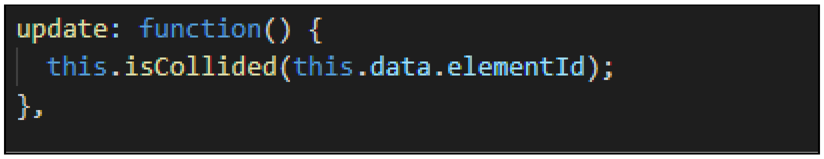
Now once the event will be fired, we can check if the variable **elementId** has the string “#ring” or “#hurdle”.

For this we can use the JavaScript method **includes()** which finds whether the one string is the part of the other string value.



```
isCollided: function(elementId) {

  const element = document.querySelector(elementId);

  element.addEventListener("collide", e => {

    if (elementId.includes("#ring")) {

      console.log(elementId+" collision");

    }
    else if(elementId.includes("#hurdle")){

      console.log(elementId+" collision");
    }

  });
}
```

Now let’s call the function inside the **.update()** handler of the A-Frame component and pass **this.data.elementId**.

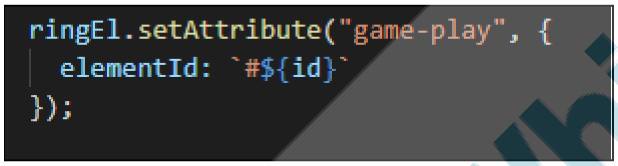


```
update: function() {
  this.isCollided(this.data.elementId);
},
```

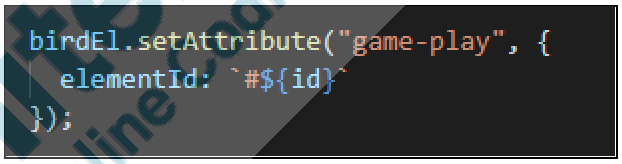
Now the “game-play” is the custom component written by us but it can be used as any other component.

Since this **elementId** will have unique ids of the ring and the bird element, we need to get these values for the “game-play” component to detect collisions.

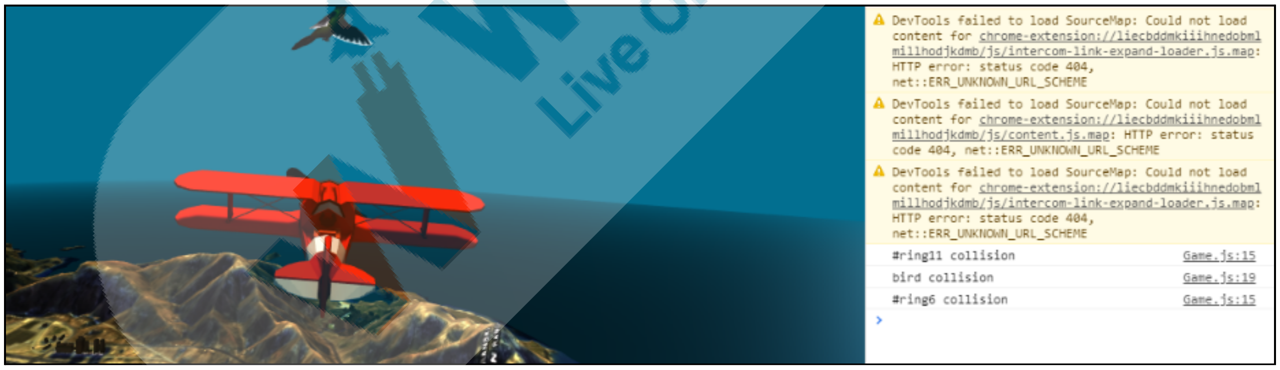
Let’s add the “game-play” component to the ring and the bird element using **setAttribute()** and set it’s property **elementId** using template literal **${}** and test the output.



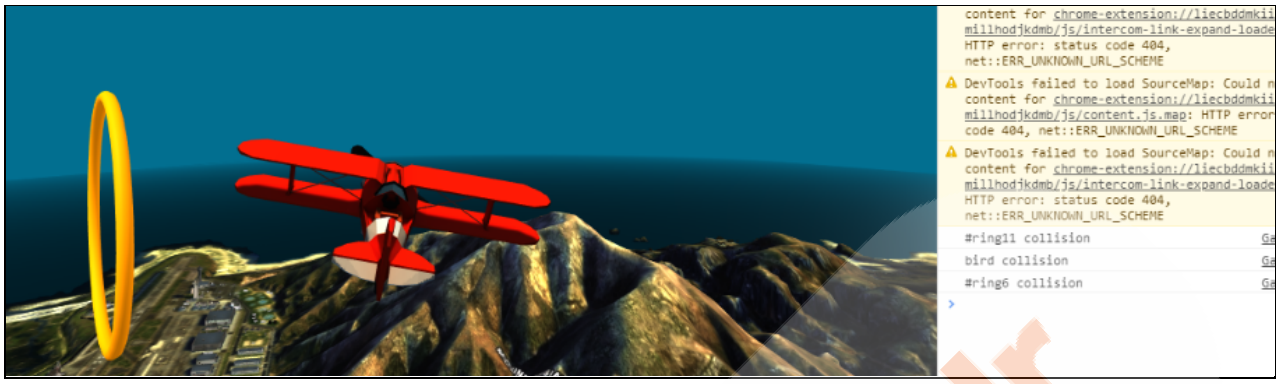
```
ringEl.setAttribute("game-play", {
  elementId: `#${id}`
});
```



```
birdEl.setAttribute("game-play", {
  elementId: `#${id}`
});
```



```
DevTools failed to load SourceMap: Could not load
content for chrome-extension://liecbddmkiiihnedobml
millhodjkdmb/js/intercom-link-expand-loader.js.map:
HTTP error: status code 404,
net::ERR_UNKNOWN_URL_SCHEME
DevTools failed to load SourceMap: Could not load
content for chrome-extension://liecbddmkiiihnedobml
millhodjkdmb/js/content.js.map: HTTP error: status
code 404, net::ERR_UNKNOWN_URL_SCHEME
DevTools failed to load SourceMap: Could not load
content for chrome-extension://liecbddmkiiihnedobml
millhodjkdmb/js/intercom-link-expand-loader.js.map:
HTTP error: status code 404,
net::ERR_UNKNOWN_URL_SCHEME
#ring11 collision                         Game.js:15
bird collision                            Game.js:19
#ring6 collision                          Game.js:15
>
```



```
content for chrome-extension://liecbddmkii
millhodjkdmb/js/intercom-link-expand-loade
HTTP error: status code 404,
net::ERR_UNKNOWN_URL_SCHEME
⚠ DevTools failed to load SourceMap: Could n
content for chrome-extension://liecbddmkii
millhodjkdmb/js/content.js.map: HTTP error
code 404, net::ERR_UNKNOWN_URL_SCHEME
⚠ DevTools failed to load SourceMap: Could n
content for chrome-extension://liecbddmkii
millhodjkdmb/js/intercom-link-expand-loade
HTTP error: status code 404,
net::ERR_UNKNOWN_URL_SCHEME
#ring11 collision                         Ga
bird collision                            Ga
#ring6 collision                          Ga
>
```

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 05 Mins

FEEDBACK

- **Compliment the student for her/his effort in the class.**
- **Encourage the student to think and come up with their own solutions.**

Teacher starts slideshow from slide 17 to slide 27

Activity details	**Solution/Guidelines**
Run the presentation from slide 17 to slide 27 **Following are the wrap-up session deliverables:** ● **Explain the facts and trivias** ● **Next class challenge** ● **Project for the day** ● **Additional Activity**	Guide the student to develop the project and share with us.

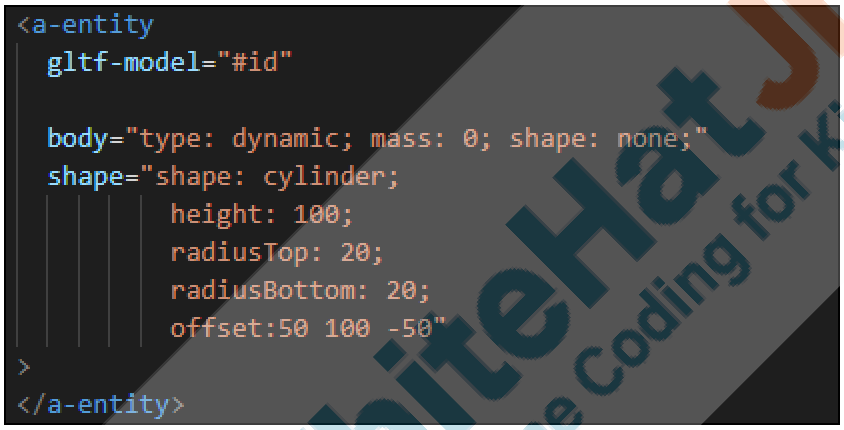
Quiz time - Click on in-class quiz

Question	**Answer**
Which event can we use to detect the collision between entities? A. touch	**D**

B. hit C. collision D. collide		
Which of the following is/are components of an A-Frame library which can initiate the physics simulation? A. static-body component B. dynamic-body component C. mechanical-body component D. Both A & B		**D**
What should be the value of the mass if we do not want the flight to fall down? A. Undefined B. 0 C. 1 D. 100		**B**
	● **End the quiz panel**	
	You get a “hats-off”. End the quiz panel Alright. See you in the next class.	*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	**COLLECT COINS** **Goal of the Project:**	

	In this project, you will add collision detection between the scuba diver and the other elements under the ocean created in the previous class. **Story:** Your friend always wanted to go scuba diving, but he is terrified of water and high waves in the ocean. He always wished to travel under the ocean without going underwater. Write an A-Frame program to add the “collide” event between elements in the scene like fish, treasure coins and the scuba diver using components. I am very excited to see how you will create a virtual ocean for your friend. Bye!	
Teacher ends slideshow		
Teacher Clicks ✖ End Class		
Additional Activities	*Guide the student to change the collider offset of a physics system static or dynamic body.* This can be done with the help of the “body” and “shape” component of A-Frame physics library.	

	• Use the “body” component to set the type of the body(static or dynamic) and keep the default shape as none. • Use the “shape” component to set the shape of the collider and offset property.	



```
<a-entity
  gltf-model="#id"

  body="type: dynamic; mass: 0; shape: none;"
  shape="shape: cylinder;
          height: 100;
          radiusTop: 20;
          radiusBottom: 20;
          offset:50 100 -50"
>
</a-entity>
```

Activity	Activity Name	Links
Teacher Activity 1	A-Frame Physics System Example Code Reference	https://github.com/whitehatjr/A-Frame-Pysics-System-Example
Teacher Activity 2	Teacher Reference Code	https://github.com/whitehatjr/PRO-C155-Teacher-Ref-Code
Teacher Activity 3	A-Frame Physics System Library Link	https://cdn.jsdelivr.net/gh/n5ro/aframe-physics-system@v4.0.1/dist/aframe-physics-system.min.js
Student Activity 1	Flight Simulation Stage 3	https://github.com/whitehatjr/PRO-C155-Student-Activity
Project Solution Link	Collect Coins	https://github.com/whitehatjr/PRO-C155-Project

Teacher Ref. Visual Aid Link	Visual Aid Link	https://curriculum.whitehatjr.com/Visual+Project+Asset/PRO_VD/PRO_C155_withcues.html
Teacher Ref. In-Class Quiz	In-Class quiz	https://s3-whjr-curriculum-uploads.whjr.online/d1965828-3c0a-4f38-98e9-2ad4f78cb970.pdf