Topic	REMOVING ELEMENTS	
Class Description	**Students learn to remove elements from the A-Frame scene. Students will learn how to remove event listeners from the elements.**	
Class	**C162**	
Class time	**45 mins**	
Goal	• Learn how to remove elements from the scene. • Learn to remove event listeners from the entity. • Learn to apply impulse using the physics library.	
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 - 5 mins		
CONTEXT • **Removing elements from the A-Frame scene.** • **Remove events listeners from the elements.** • **Apply force on the elements in A-Frame using Cannon.js method.**		

Teacher Starts Slideshow  **Slide 1 to 3** Refer to speaker notes and follow the instructions on each slide.	
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today? **Following are the WARM-UP session deliverables:** • Greet the student. • Revision of previous class activities. • Quizzes.	**ESR**: Hi, thanks! Yes I am excited about it! Click on the slide show tab and present the slides
WARM-UP QUIZ Click on In-Class Quiz	
Continue WARM-UP Session  **Slide 4 to 16**	
Following are the session deliverables: • Appreciate the student. • Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.	

Class Steps	Teacher Action	Student Action
	We were able to solve two issues- Remember when we were making bullets as a child of the camera element it was moving with the camera? How did we fix that issue? And also the other issue was the velocity direction and we fixed it with	**ESR**: We had to make it the child of the scene element.

	the help of camera direction using Three.js objects. But there is one major issue, can you tell me what the issue is? Every time we shoot the bullet in the A-Frame scene, it becomes a part of the scene. Now, suppose we keep shooting for a long time in the game. What will happen? Yes, exactly! And if we keep on shooting for a longer time, this will keep on increasing the entity in the scene. There will be too many bullets in the scene. This should not happen as it will bring the load in the game or the computer might crash after some time. Today we will see how we can fix this issue in the game. Are you excited? But before I can start today's class I have an exciting quiz for you! Are you ready?	**ESR:** Varied. **ESR**: The bullets will be continuously created and they will be part of the scene. **ESR**: Yes. **ESR**: Yes.

	Let's get started then.	
Teacher Ends Slideshow 		
TEACHER-LED ACTIVITY - 15 mins		
Teacher Initiates Screen Share		
CHALLENGE • **Remove elements from the A-Frame scene after shooting the bullet.** • **Apply impulse using Cannon.js method in A-Frame.**		
Step 2: Teacher-led Activity (15 mins)	*<The teacher opens the code from the previous class and opens the shoot.js file.>* *Teacher Activity 1* Now, we have a major issue to fix. What can we do about that? There should be some way to remove elements from the scene, right? We will be learning how we can remove elements from the scene which are not required all the time. When do you think we should remove the bullet? Correct! Let's get cracking. Once the bullets collide with the boxes, then these are not needed in the scene so we can remove them.	**ESR:** Varied. **ESR:** Yes! **ESR***:* We should remove bullets whenever the bullets collide with the object.

	We have already seen in previous classes how we can implement the collision between any two entities. Do you remember how we can do that? Amazing! Let's begin with it. *<The teacher writes the code to add "**collide**" event and "**removeBullet**" function over the bullet element, using addEventListener().>*	**ESR**: We can add an event listener called **collide** on the bullet element.
 		
	For a collision to happen between any two entities, what is important? These entities have to be a part of the physics system, which means the entities can be either static-body or dynamic body to follow any physics law.	**ESR**: Varied.

	Before that, we should make the bullet element a dynamic entity and all the boxes as static entities, right? Then only the collision detection can happen. So let's quickly make all the boxes in the scene as static-body and the bullet as the dynamic-body. Also let's assign the id to each box. *<The teacher updates the previous class code to add the **dynamic-body** attribute of the bullet element, using **setAttribute()** in **shoot.js** file>* *<The teacher updates the previous class the code to add **id** and the **static-body** attribute to all the box element in **index.html** file>*	**ESR:** Yes!

shoot.js

```
//set the bullet as the dynamic entity
bullet.setAttribute("dynamic-body", {
  shape: "sphere",
  mass: "0",
});
```

index.html

```
<!--Boxes-->
<a-box  id="box1" position="-2 1.5 -10" color="tomato" depth="1" height="1" width="1" static-body></a-box>

<a-box  id="box2" position="0 1.5 -10" color="tomato" depth="1" height="1" width="1" static-body></a-box>

<a-box  id="box3" position="2 1.5 -10" color="tomato" depth="1" height="1" width="1" static-body></a-box>

<a-box  id="box4" position="-1 2.5 -10" color="tomato" depth="1" height="1" width="1" static-body></a-box>

<a-box  id="box5" position="1 2.5 -10" color="tomato" depth="1" height="1" width="1" static-body></a-box>

<a-box  id="box6" position="0 3.5 -10" color="tomato" depth="1" height="1" width="1" static-body></a-box>
```

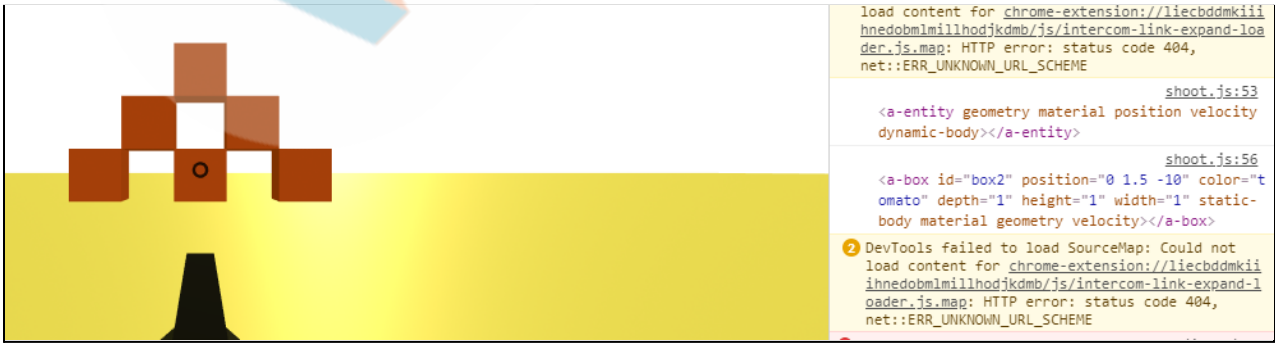
The “**e.detail.target.el**” gives the details about the original entity on which the event has been triggered.

The “**e.detail.body.el**” gives the details about the other entity which the original entity has touched.

Now let's console these values and see what we have got.

shoot.js

```
removeBullet: function (e) {
  //Original entity (bullet)
  console.log(e.detail.target.el);

  //Other entity, which bullet touched.
  console.log(e.detail.body.el);
},
```



We can see the A-Frame bullet entity's details and the box entity at which the collision happened.

Now, we should make sure once the bullet hits the ground or any of the boxes, we should remove the element from the scene.

We should also remove the event listener which is attached to the bullet.
This is a good practice that we should first remove the event listener attached to the element and then remove the child from the A-Frame scene.

Let's take two variables to store the details of the elements-

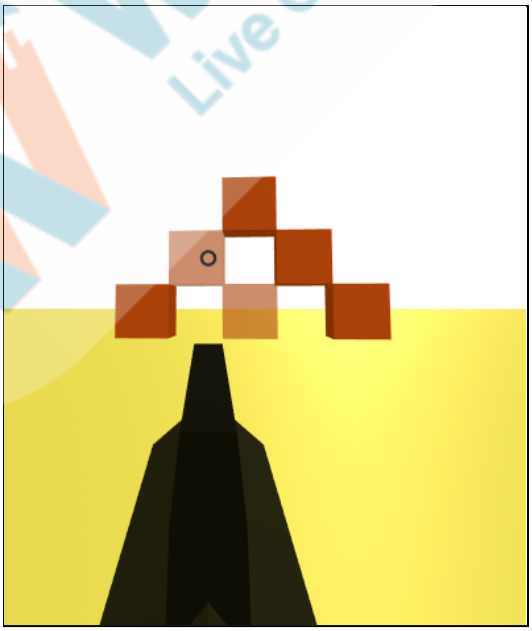
var **element = e.detail.target.el**
var **elementHit** = "**e.detail.body.el**"

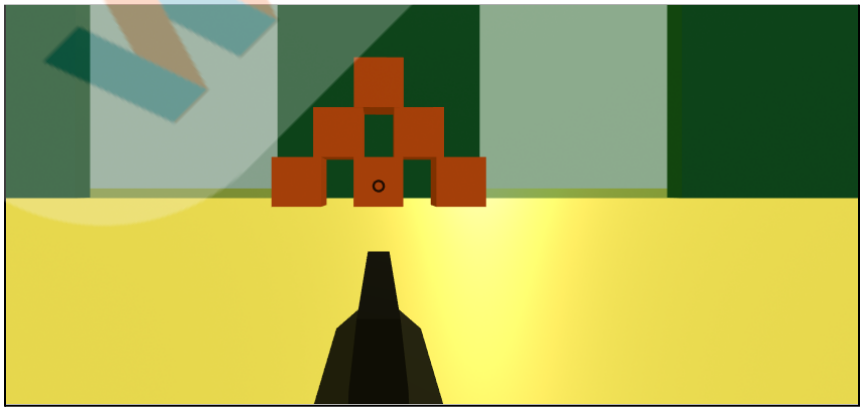
Now, since the bullet is the child of the scene element in the A-Frame, we should select the scene with the help of the **.querySelector()** method and then we should use the **.removeChild()** method to remove the child from the scene.

To make sure the bullet is removed from the scene after collision, let's set the opacity of the box as 0.6.

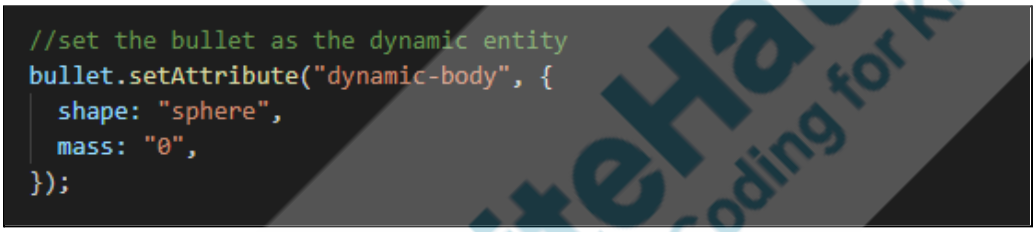
Let's verify if this is happening.

```
removeBullet: function (e) {
  //Original entity (bullet)
  console.log(e.detail.target.el);

  //Other entity, which bullet touched.
  console.log(e.detail.body.el);

  //bullet element
  var element=e.detail.target.el

  //element which is hit
  var elementHit = e.detail.body.el;

  if (elementHit.id.includes("box")) {
    elementHit.setAttribute("material", {
      opacity: 0.6,
      transparent: true,
    });

    //remove event listener
    element.removeEventListener("collide", this.shoot);

    //remove the bullets from the scene
    var scene = document.querySelector("#scene");
    scene.removeChild(element);
  }
},
```



	It was quite interesting to solve such a big problem so easily, right? Now, let's increase the challenge in the game. Let's add some walls to pass through. The boxes are supposed to cross the walls. A few of them will be static-body walls and a few of them will not be any physics body so that we box to pass through. The walls through which we can pass will have less opacity. *<The image below is not the output of any teacher activity conducted so far, this image only for teacher reference to understand the the further activity that will be conducted by the student.>*	**ESR**: Yes!
		

	Now that we have learned how to remove elements from the scene, you will add this functionality to your game. Also you will add walls to make this game more challenging! Are you excited?	**ESR**: Yes!
Teacher Stops Screen Share		
So now it's your turn. Please share your screen with me.		
Teacher Starts Slideshow  **Slide 17 to 18** Refer to speaker notes and follow the instructions on each slide.		
We have one more class challenge for you. Can you solve it? Let's try. I will guide you through it.		
Teacher Ends Slideshow 		
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 • **Remove the bullet element after collision** • **Apply impulse using Cannon.js method in A-Frame.**		

Step 3: Student-Led Activity (20 mins)	*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.*
	Let's start by making the boxes and the bullet as "**dynamic-body**". *Guide the student to make the bullet as a **dynamic-body**.*	
 ``` //set the bullet as the dynamic entity bullet.setAttribute("dynamic-body", { shape: "sphere", mass: "0", }); ```		
	What should we do now? Yes. ***Note**: The following is part of Boilerplate code in **shoot.js** file*	**ESR**: Add the collide event.

```
removeBullet: function (e) {
  //Original entity (bullet)
  console.log(e.detail.target.el);

  //Other entity, which bullet touched.
  console.log(e.detail.body.el);

  //bullet element


  //element which is hit


  if (elementHit.id.includes("box"))
    {
      //set material attribute


      //impulse and point vector


      //remove event listener


      //remove the bullets from the scene

    }
  },
});
```

	*Guide the student to take two variables **element** and **elementHit** and **add the condition to remove the event listener and then remove the bullet as the child entity** from the scene when it collides with the boxes in **shoot.js** file.*	

```
removeBullet: function (e) {
  //Original entity (bullet)
  console.log(e.detail.target.el);

  //Other entity, which bullet touched.
  console.log(e.detail.body.el);

  //bullet element
  var element=e.detail.target.el

  //element which is hit
  var elementHit = e.detail.body.el;

  if (elementHit.id.includes("box")) {
    elementHit.setAttribute("material", {
      opacity: 0.6,
      transparent: true,
    });

    //remove event listener
    element.removeEventListener("collide", this.shoot);

    //remove the bullets from the scene
    var scene = document.querySelector("#scene");
    scene.removeChild(element);
  }
},
```

	Now, let's increase the challenge in the game. Let's add some walls to pass through. The boxes are supposed to cross the walls. A few of them will be static-body walls and a few of them will not be any	

	physics body so that we box to pass through. The walls through which we can pass will have less opacity. *Guide the student to **uncomment** the code to create the walls using <a-box> and set their color, position, depth, width, static-body and opacity in the **index.html** file.*	

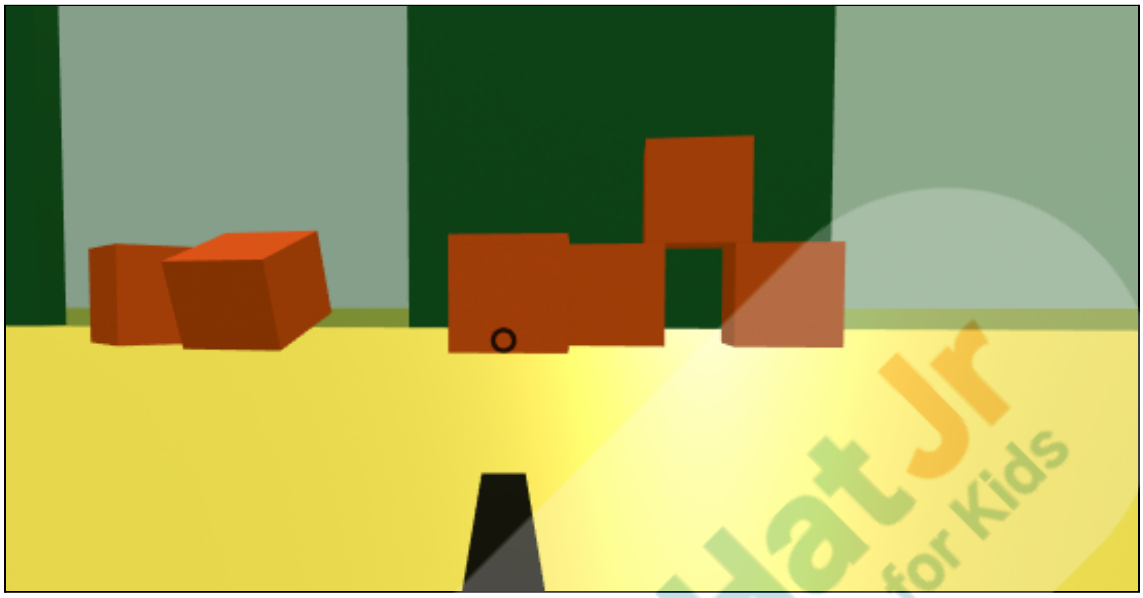
```
<!--Walls-->
<!-- <a-box color="#1469C9" id="wall1" position="-20 1.5 -20"  depth="1" height="50" width="10" static-body></a-box>

<a-box color="#1469C9" id="wall2" position="0 1.5 -20"  depth="1" height="50" width="10" static-body></a-box>

<a-box color="#1469C9" id="wall3" position="20 1.5 -20"  depth="1" height="50" width="10" static-body></a-box>

<a-box color="#1469C9" id="wall4" position="-10 1.5 -20"  depth="1" height="50" width="10" opacity="0.5"></a-box>

<a-box color="#2889DF" id="wall6" position="10 1.5 -20"  depth="1" height="50" width="10" opacity="0.5" ></a-box>

<a-box color="#2889DF" id="wall6" position="-30 1.5 -20"  depth="1" height="50" width="10" opacity="0.5"></a-box>

<a-box color="#2889DF" id="wall7" position="30 1.5 -20"  depth="1" height="50" width="10" opacity="0.5" ></a-box> -->
```

	Now, what should we do to shoot the boxes through the walls? Yes, exactly! And how can we do that? There should be something to apply force over the boxes when the bullet hits them, right?	**ESR:** The boxes should be pushed towards the wall when we shoot. **ESR**: Varied. **ESR**: Yes.

	Do you remember that the A-Frame 3D physics system is built on a JavaScript physics engine library called **Cannon.js**? If we want to apply force or impulse or any other physics functionality over the elements, we can use Cannon.js methods over the A-Frame entity elements. Let's see how we can apply impulse on the box as soon as the bullet hits the box. **Impulse** is the amount of very small amount of force that can be applied over the body in a particular direction. It's like giving a jerk to the body which is hit. For this we will have to use the Cannon.js **applyImpulse(impulse, worldPoint)** method on the element body. The parameters: • **impulse** is of type Cannon.js **Vec3**. It is the amount of impulse to add to the body. • **worldPoint** is of type Cannon.js **Vec3**. It is the point at which the force is applied.	**ESR**: Yes.

	To pass these parameters, we need to create a new Cannon.js object using **new CANNON.Vec3()**, for the impulse and the point. The **CANNON.Vec3().copy()** method is used to copy the elementHit position vector from the element. Let's test what happens when the bullet hits the boxes. *Guide the student to create an **impulse** and **worldPoint variable** and apply a small amount of force using **applyimpulse() method** in the **shoot.js** file.*	

```
//impulse and point vector
var impulse = new CANNON.Vec3(-2, 2, 1);
var worldPoint = new CANNON.Vec3().copy(elementHit.getAttribute("position"));

elementHit.body.applyImpulse(impulse, worldPoint);
```



	Well, nothing happens! Can you tell me why?	**ESR**: Varied.
	The reason is boxes are not having any force applied to them. Can you tell me the reason why the boxes are not able to feel the force?	**ESR**: Varied.
	Before we fix this, can you tell me the difference between static and dynamic bodies?	**ESR**: Yes, • The **static-body** are fixed-positioned physics bodies which are unaffected by gravity, collisions, and force on their own.But they can collide with other **static-body** or **dynamic-body**.

	Do you remember in our code the boxes are static-body? Then we should make them **"dynamic-body"** to apply the force on them. But as soon as we load the page, they fall down because of gravity. The boxes' width have been increased a little bit to make them stay one over the other. *Guide the student to student to update the boxes from static-body to dynamic-body in **index.html** file and test the output.*	• The **dynamic-body** are free moving bodies which are affected by gravity, collisions. **ESR**: Yes.

```
<!--Boxes-->
<a-box  id="box1" position="-2 1.5 -10" color="tomato" depth="1" height="1.2" width="1.2" dynamic-body></a-box>

<a-box  id="box2" position="0 1.5 -10" color="tomato" depth="1" height="1.2" width="1.2" dynamic-body></a-box>

<a-box  id="box3" position="2 1.5 -10" color="tomato" depth="1" height="1.2" width="1.2" dynamic-body></a-box>

<a-box  id="box4" position="-1 2.5 -10" color="tomato" depth="1" height="1.2" width="1.2" dynamic-body></a-box>

<a-box  id="box5" position="1 2.5 -10" color="tomato" depth="1" height="1.2" width="1.2" dynamic-body></a-box>

<a-box  id="box6" position="0 3.5 -10" color="tomato" depth="1" height="1.2" width="1.2" dynamic-body></a-box>
/a-scene>
```

Teacher Guides Student to Stop Screen Share

WRAP UP SESSION - 5 mins

Teacher Starts Slideshow
Slide 19 to 22

Activity details
Following are the WRAP-UP session deliverables:

- Appreciate the student.
- Revise the current class activities.
- Discuss the quizzes.

WRAP-UP QUIZ
Click on In-Class Quiz

Continue WRAP-UP Session
Slide 23 to 28

Activity Details

Following are the session deliverables:

- Explain the facts and trivia
- Next class challenge
- Project for the day
- Additional Activity (Optional)

FEEDBACK

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

	You get a “hats-off”. Alright. See you in the next class.	*Make sure you have given at least 2 Hats Off during the class for:* 

PROJECT OVERVIEW DISCUSSION

Refer the document below in Activity Links Sections

Teacher Clicks ✖ End Class

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.	*The student uses the markdown editor to write their reflections in a reflection journal.*

	○ The 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?	

Activity	Activity Name	Links
Teacher Activity 1	Previous Class Code	https://github.com/whitehatjr/PRO-C161-Teacher-Ref
Teacher Activity 2	Teacher Reference Code	https://github.com/whitehatjr/PRO-C162-Teacher-Ref
Teacher Activity 3	Output Reference	https://curriculum.whitehatjr.com/PRO+Asset/shoot-boxes.mp4
Student Activity 1	Boilerplate Code.	https://github.com/whitehatjr/PRO-C162-Student-Boilerplate
Teacher Reference 1	Project Document	https://s3-whjr-curriculum-uploads.whjr.online/8b1c2eab-8d99-49f7-b0f6-17b6a8609447.pdf
Teacher Reference 2	Project Solution	https://github.com/whitehatjr/PRO-C162-Project-Solution
Teacher Reference 3	Visual-Aid	https://s3-whjr-curriculum-uploads.whjr.online/bbc036d8-ef42-4e34-b458-c1c7a2aff722.html
Teacher Reference 4	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/449f4699-61ac-41f0-8a60-10248f9f21e5.pdf