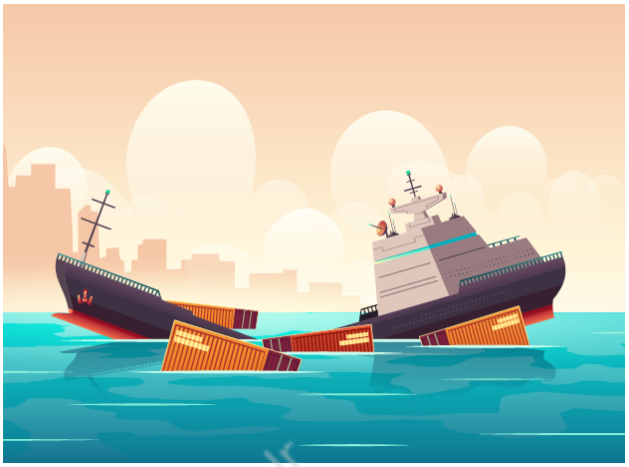
COLLISION DETECTION



What is our GOAL for this MODULE?

In this class, we learned about nested loops; we also learned to detect the collision between boats and cannonballs.

What did we ACHIEVE in the class TODAY?

- Wrote a **collisionWithBoat()** function to check the collision between the boat and the cannonball using the **Matter.SAT.collided()** function.
- Wrote **remove()** function to remove the ball and the boat from the world and the array.

Which CONCEPTS/ CODING BLOCKS did we cover today?

- Nested loops concept
- **Matter.SAT.collides()**
- **remove()**
- **setTimeout()**

How did we DO the activities?

1. Create another function called **collisionWithBoat()** to check if there was a collision between the boat and the cannonball. This function took only the **index** number value as a parameter. Declare a **collision** variable and set the value from **Matter.SAT.collides()** function.

```
function collisionWithBoat(index) {
  for (var i = 0; i < boats.length; i++) {
    if (balls[index] !== undefined && boats[i] !== undefined) {
      var collision = Matter.SAT.collides(balls[index].body, boats[i].body);

    }
  }
}
```

2. Write a **remove** function to remove the boat from the world and array and use **delete** method to delete the boat from the array.

```
remove(index) {
  setTimeout(() => {
    Matter.World.remove(world, boats[index].body);
    delete boats[index];
  }, 2000);
}
```

3. Write **remove()** function inside the cannonball class as we also need to remove the cannonball from the screen.

```
remove(index) {
  Matter.Body.setVelocity(this.body, { x: 0, y: 0 });

  setTimeout(() => {
    Matter.World.remove(world, this.body);
    delete balls[index];
  }, 1000);
}
```

4. Use another **if** condition to check if the **collision.collided** is true. If it is true then, inside this condition we call the **boats[i].remove[i]** function and call **Matter.World.remove() and delete balls[index]** to remove balls from the world.

```
function collisionWithBoat(index) {
  for (var i = 0; i < boats.length; i++) {
    if (balls[index] !== undefined && boats[i] !== undefined) {
      var collision = Matter.SAT.collides(balls[index].body, boats[i].body);

      if (collision.collided) {
        boats[i].remove(i);

        Matter.World.remove(world, balls[index].body);
        delete balls[index];
      }
    }
  }
}
```

5. Inside the **showCannonBalls()** function write a condition to check if the **X position** of the cannonball is more than or equal to the width of the screen or if the **y position** is greater than the height-50 which means little above the ground.

```
function showCannonBalls(ball, index) {
  if (ball) {
    ball.display();
    if (ball.body.position.x >= width || ball.body.position.y >= height - 50) {
      ball.remove(index);
    }
  }
}
```

6. Write **collisionWithBoat()** function to check if the value inside the boats array is not undefined using the **if** condition and check if the collision is happening between the ball and boat and if they are colliding we remove them from the world and the array.

```
function collisionWithBoat(index) {
  for (var i = 0; i < boats.length; i++) {
    if (balls[index] !== undefined && boats[i] !== undefined) {
      var collision = Matter.SAT.collides(balls[index].body, boats[i].body);

      if (collision.collided) {
        boats[i].remove(i);

        Matter.World.remove(world, balls[index].body);
        delete balls[index];
      }
    }
  }
}
```

7. Create a **remove()** function to remove cannonballs from the world and the array.

```
remove(index) {
  Matter.Body.setVelocity(this.body, { x: 0, y: 0 });

  setTimeout(() => {
    Matter.World.remove(world, this.body);
    delete balls[index];
  }, 1000);
}
```

8. Also add a condition in the **showCannonBalls()** function to remove the cannonballs.

```
function showCannonBalls(ball, index) {
  if (ball) {
    ball.display();
    if (ball.body.position.x >= width || ball.body.position.y >= height - 50) {
      ball.remove(index);
    }
  }
}
```

What's next?

In the next class, let's add some proper animations to the boats to give a feel of them traveling on water and adding sounds to make the game more interesting.

EXTEND YOUR KNOWLEDGE

1. Bookmark the following link to know more about setting velocity to physics engine bodies: https://brm.io/matter-js/docs/classes/Body.html#method_setVelocity