CODE NINJAS®

Silver Belt Ninja Guide
Activity 01: Robomania

Activity 1

Robomania

Objective: The ninja will be able to write conditional statements to program a non-playable character's movement using the **rigidbody** component and the component's built-in functions and properties in a futuristic robot game.

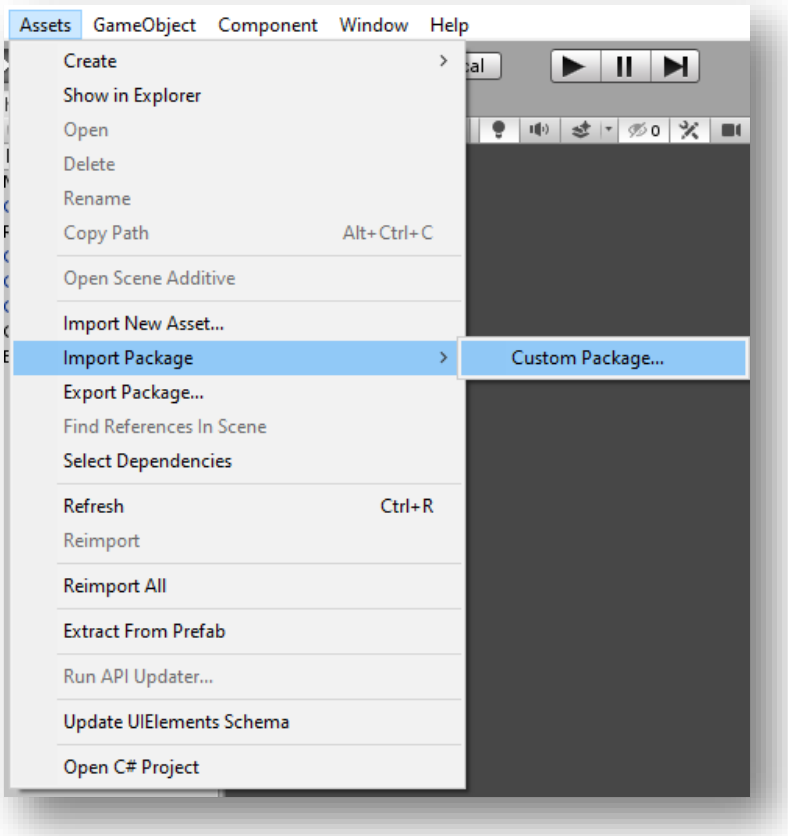
Our hero RoboMan must overcome two evil crushers to save the day!

1 Start a new **Unity** Project and name it *YOUR INITIALS – Robomania*.

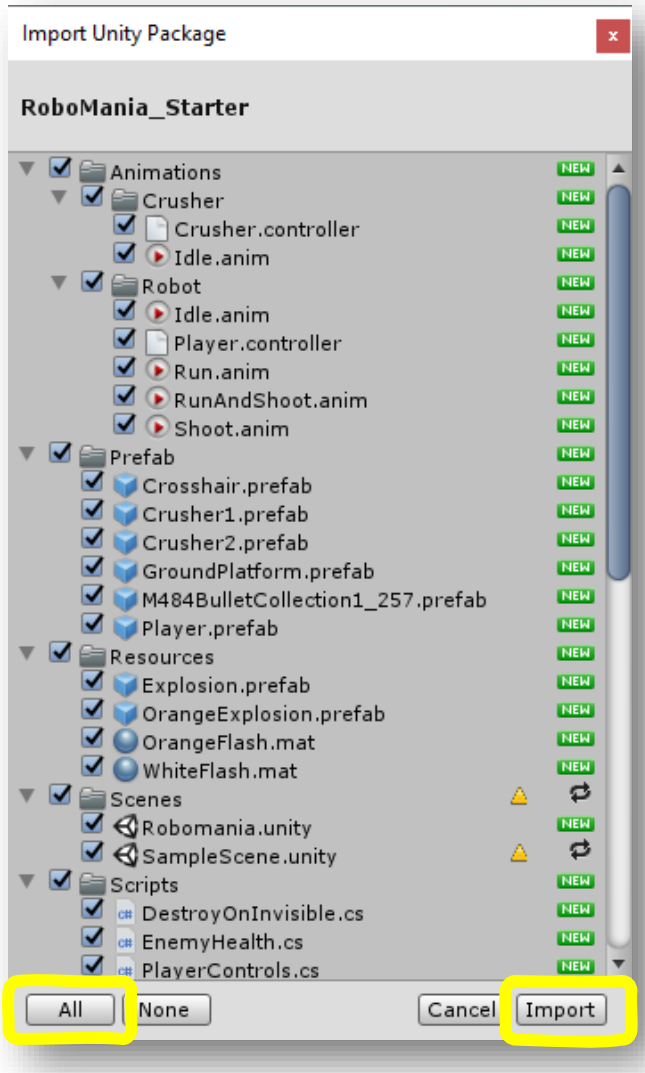
Select the **3D core** and place the project in the correct folder.

Click the **Create** button.

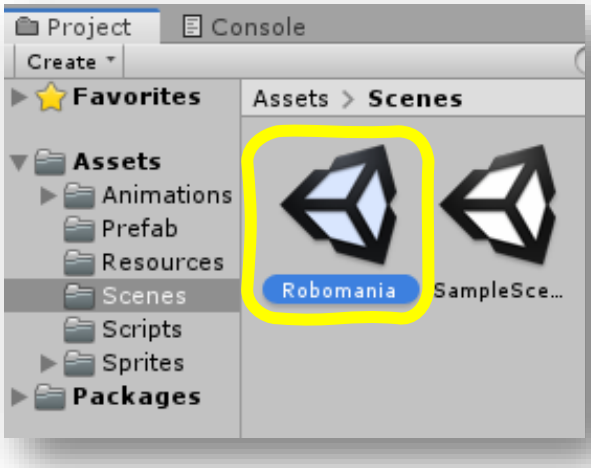
2 After Unity loads your new project, **import** the Robomania Starter Unity Package, named **Activity 01 - Robomania.unitypackage** by clicking on **Assets > Import Package > Custom Package**.



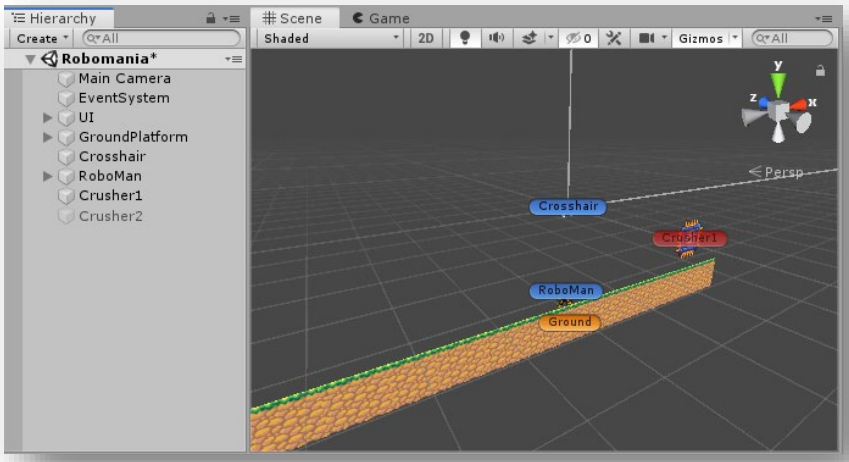
3 On the **Import Unity Package** window, click **All** and then **Import**.



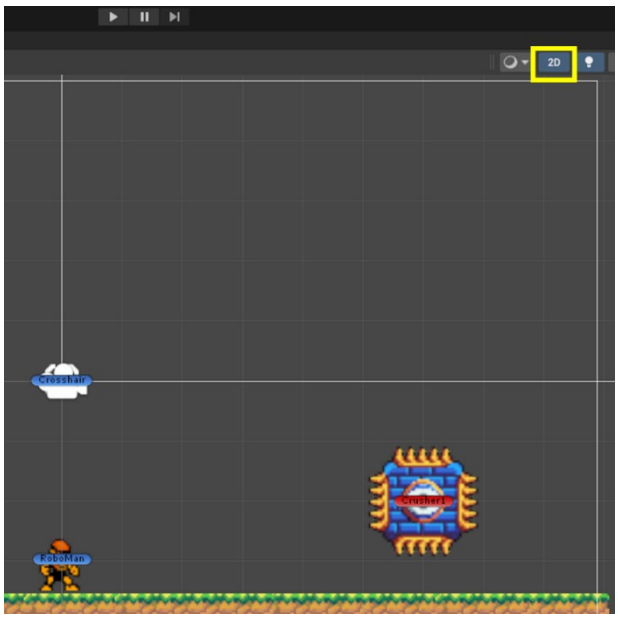
4 After it is done importing, double-click the **Robomania** scene. It will be located in the **Project** tab within the **Assets > Scenes** folder.



5 Opening **Robomania** will load the **Scene** with all our game objects. Make sure that you can see all the objects and the assets in the scene tab.



6 If the sprites look funny and out of place, click the 2D button at the top of the Scene tab to align the view to the game.



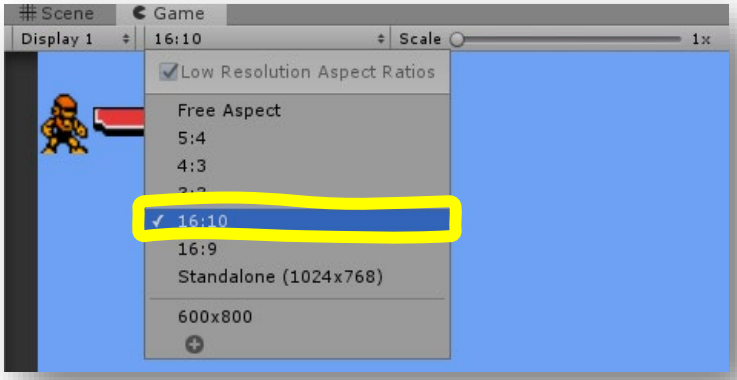
7 Click Unity's **play** button and do your first playtest of Robomania. You can move RoboMan with the **WASD** keys or the arrow keys, make RoboMan jump with the spacebar, and fire toward the cursor by clicking the left mouse button.

Try to destroy the crusher! Did you notice how the crusher just floats there? What happens when you try to use your ProtonBlaster to

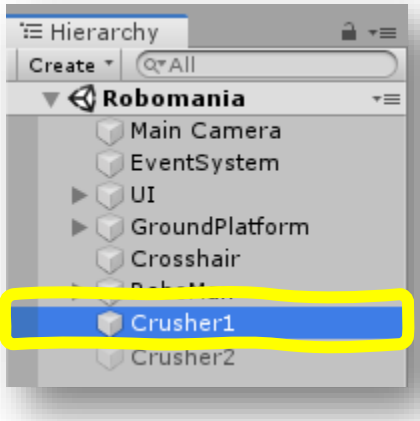
defeat the crusher? What happened when you run RoboMan into a crusher?

Right now, the crusher doesn't know how to do anything but blink!

8 If you tried to play and you couldn't see everything in the Game tab, set your resolution to **16:10** and make sure your **Scale** is at **1x**.



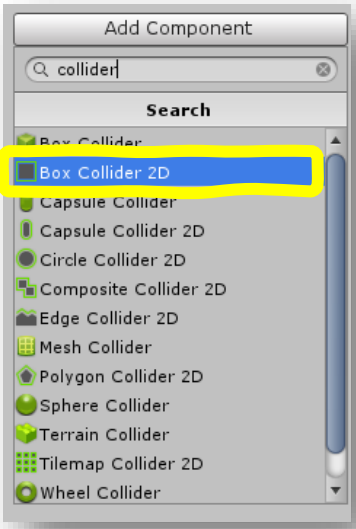
9 How do we get the crusher to respond to the proton blasts and **RoboMan**? Think back to other games you made where game objects needed to **collide** with other game objects.



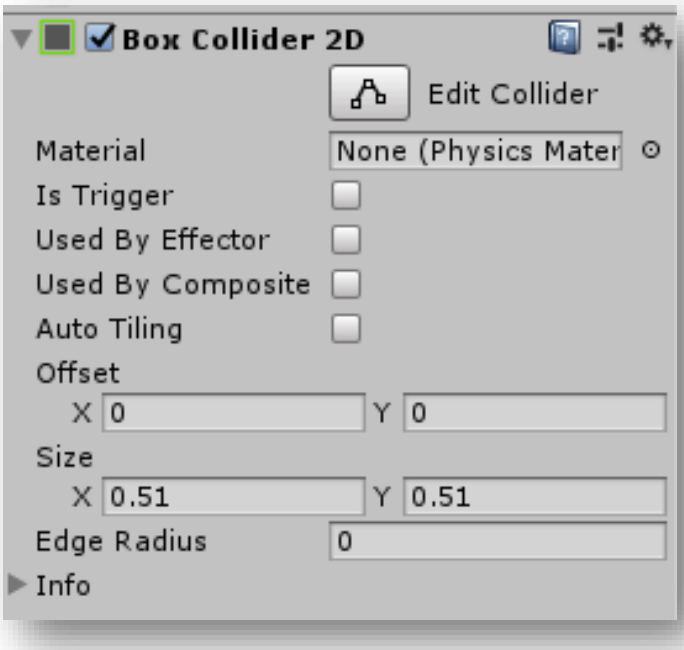
We need to add a **collider** to the crusher for it to interact with other game objects! Click on **Crusher1** in the **Hierarchy** or the scene to open it in the Inspector.

10 Click the Add Component button.

Search for "*collider*" and select **Box Collider 2D**.



11 The **Crusher1** game object has a **Box Collider 2D** component. We can leave the default settings as they are.



12 **Play** the game and see what adding a **collider** changed. Based on the changes, talk to your **Code Sensei** about the purpose of a **collider** and how it changes the way **game objects** interact with each other.

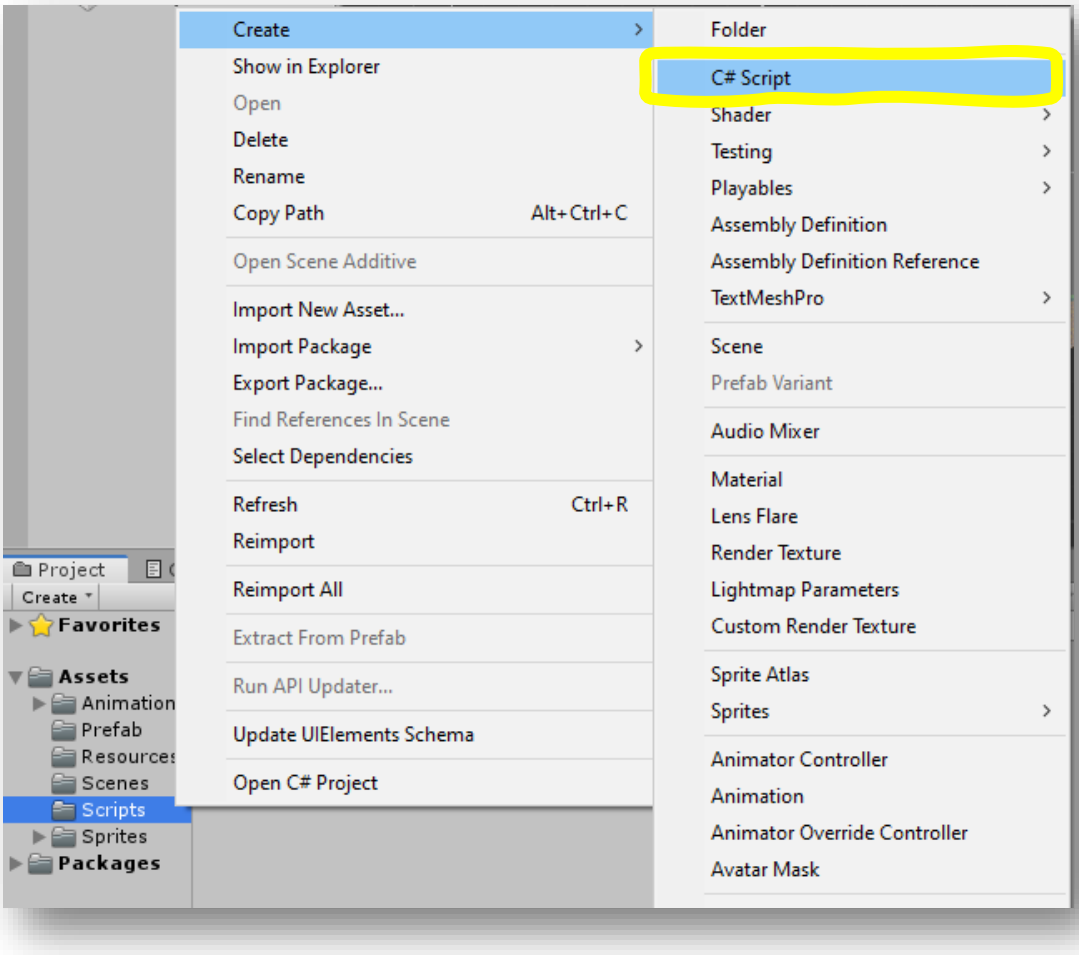
How can we make the game more challenging?

What if we made the crusher move?

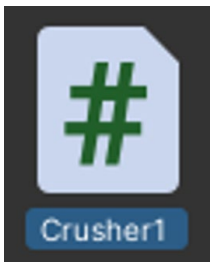
What type of **component** do we need to add in order to **script** out an enemy's movement?

13 If we want to control how the crusher moves, we need to create a new script and provide it instructions on what to do.

Create a new script by opening the **Project** tab, right-clicking on the **Scripts** folder then selecting **Create > C# Script**.



14 Name the script **Crusher1**.

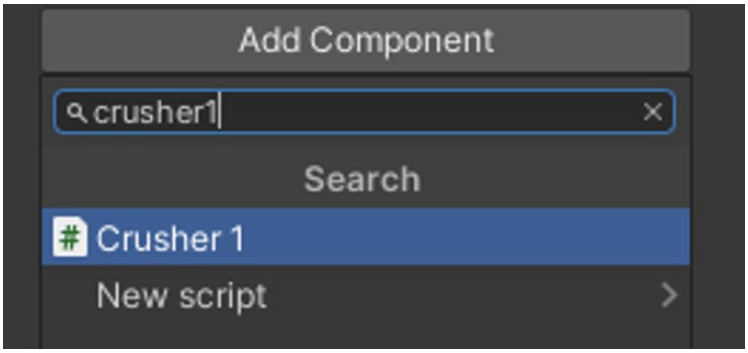


15 In the **Hierarchy**, click on **Crusher1** to open it in the **Inspector**.

16 Click the 

button.

Search for "Crusher1" to find the **Crusher1** script you just made.



17 Double-click the script to open it.

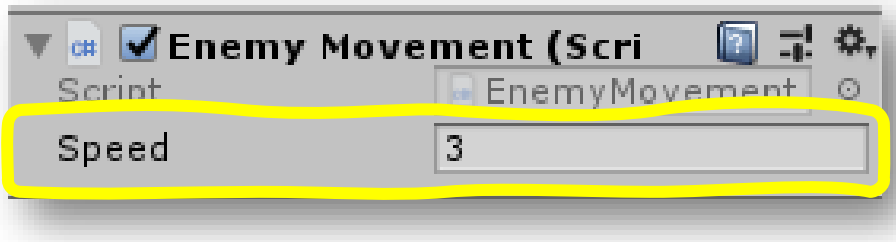
18 We do not need the `Start` or `Update` functions, so delete both of those so you only have **three** `using` **statements** and the `public class` line in the file.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


public class Crusher1 : MonoBehaviour
{

}
```

19 We want to move the **crusher** at a constant speed, so on the line after the open curly brace type `public float speed;` to create a **variable** that we can change in the **Inspector** tab.

```
public class Crusher1 : MonoBehaviour
{
    public float speed;
}
```

20 In the **Inspector**, set the value of **Speed** to 3. You can change this later, but `3` is a good starting value.



21 After the `public float speed;` line, create a `FixedUpdate` function by typing `private void FixedUpdate() { }` making sure there is an empty line between the curly brackets.

```
private void FixedUpdate()
{

}
```

Why are we using `FixedUpdate`? FixedUpdate always runs the same amount of times every second (usually 50 times per second). Regular Update runs as fast as the computer can update, which sometimes will be 50, but it could be more if the computer is faster, or less if the computer is slower.

Anytime you are changing a game object's **position** or **physics**, you should be using Unity's `FixedUpdate` function to make sure that physics is the same on any computer that plays the game.

22 In the `FixedUpdate` function, we need to calculate the crusher's new position. We want the **crusher** to move left and right, but *not* up and down. Declare a new float variable named `newXPosition` and set it equal to the crusher's current x position and add the speed times the amount of time that has passed.

Type `float newXPosition = transform.position.x + speed * Time.fixedDeltaTime;`

```
private void FixedUpdate()
{
    float newXPosition = transform.position.x + speed * Time.fixedDeltaTime;
}
```

23 Since we do not want to move the **crusher** up and down, type `float newYPosition = transform.position.y;`

This line of code will make sure the crusher's new **y position** is equal to its old **y position**.

```
private void FixedUpdate()
{
    float newXPosition = transform.position.x + speed * Time.fixedDeltaTime;
    float newYPosition = transform.position.y;
}
```

24 We need to combine these two new **position** values into one **vector**. On the next line, type `Vector2 newPosition = new Vector2(newXPosition, newYPosition);` to declare a `new Vector2` variable named `newPosition`.

```
private void FixedUpdate()
{
    float newXPosition = transform.position.x + speed * Time.fixedDeltaTime
    float newYPosition = transform.position.y;
    Vector2 newPosition = new Vector2(newXPosition, newYPosition);
}
```

25 Save your script and play the game.

What happened? Nothing?

We calculated everything correctly, but what did we miss? Did we ever use our `newPosition` **vector**?

26 We need to set the crusher's position to be equal to the `newPosition` variable we just created. On the next line, type `transform.position = newPosition;`

```
private void FixedUpdate()
{
    float newXPosition = transform.position.x + speed * Time.fixedDeltaTime;
    float newYPosition = transform.position.y;
    Vector2 newPosition = new Vector2(newXPosition, newYPosition);
    transform.position = newPosition;
}
```

27 **Save** your script and play your game. Is the **crusher** moving? Is the **crusher** moving how you want it to move?

With your **Code Sensei**, discuss two different possible solutions on how to make your **crusher** move to the left toward **RoboMan**.

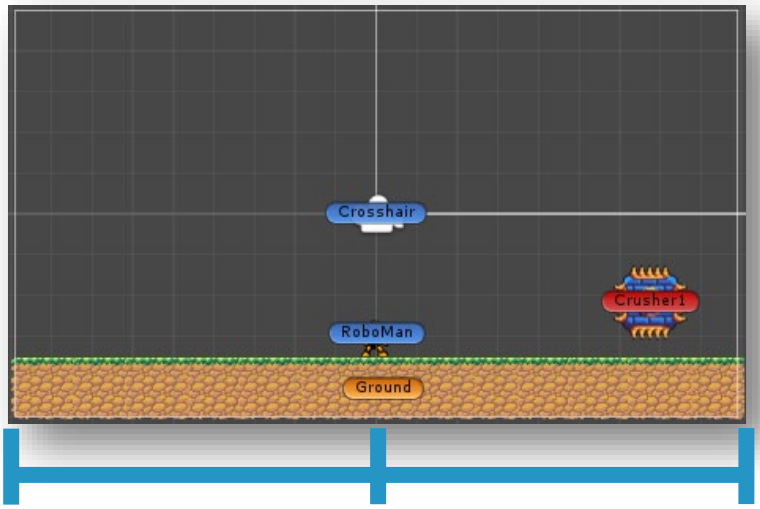
28 Since the **crusher** is moving to the right and we want it to move to the left, we will fix it by setting the **Speed** in the **Inspector** to be equal to `-3` instead of `3`.

29 **Play** your game and see if the problem is solved.

Now is a good time to think about what parts of your game you still need to change.

What happens when the crusher reaches the edge of the screen? What happens to **RoboMan** when he hits the crusher?

30 Let's first make sure that the **crusher** doesn't fly off the screen! The **scene** is set up so the **camera** at the **center** of the screen has an **x coordinate** of `0`, the far left is at an **x coordinate** of `-8`, and the far right is at an **x coordinate** o `8`.




```
-8          0          8
```

31 We need to make sure that if the crusher reaches the edge of the screen that it bounces back in the opposite direction. With your **Code Sensei**, discuss how we check a value and execute different pieces of code based on the result.

32 We need to first check to see if the crusher hits the left side of the scene. At the start of the `FixedUpdate` function before the `float`

`newXPosition` line, add an `if` **statement** that checks to see if the position of the crusher is less than or equal to `-8`. Type `if (transform.position.x <= -8) { }` making sure to put an empty line between the curly brackets.

```
private void FixedUpdate()
{
    if (transform.position.x <= -8)
    {

    }
    float newXPosition = transform.position.x
    float newYPosition = transform.position.y
    Vector2 newPosition = new Vector2(newXPos
```

33 Whenever the **crusher** hits the left side of the scene, the code inside this if statement will execute. What do we need to do to the **speed** of the **crusher** to get it to move in the opposite **direction**?

We need to change a **negative speed** to a **positive speed**! We can flip the sign of the **speed** by simply **multiplying** the **speed** by `-1`.

Type `speed *= -1;` inside the **if statement** to **flip** the sign when the **crusher** reaches the edge of the scene.

```
if (transform.position.x <= -8)
{
    speed *= -1;
}
```

34 **Play** your game and let the **crusher** bounce across the scene.

Is everything working like you expected? Did your crusher bounce off the left side of the scene? What about the right side?

35 We need to also **flip** the sign of the **speed** if the **crusher** reaches the right side of the scene! Since we are already flipping the speed in one

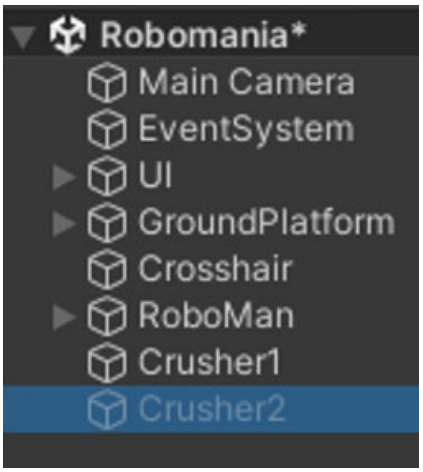
if statement, lets use an **or** condition to also flip it we reach the right side!
Type `|| transform.position.x >= 8` inside the condition for the if statement.

```
if (transform.position.x <= -8 || transform.position.x >= 8)
{
    speed *= -1;
}
```

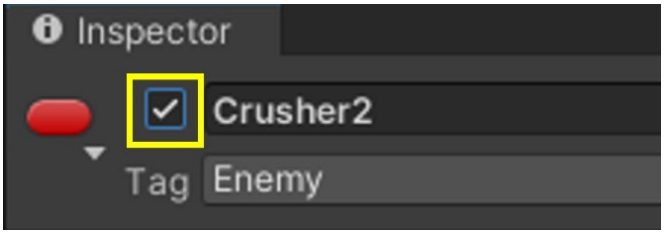
36 **Play** your game. Is the crusher behaving how you expect it to? It should be bouncing off the walls and changing direction!

That was a lot of work! Let's take a look at how Unity has some tools to save us some time, using the powerful Rigidbodies!

37 In the **Hierarchy**, click on **Crusher2** to show it in the **Inspector**.



38 In the **Inspector**, enable **Crusher2** by checking the box to the left of its name.

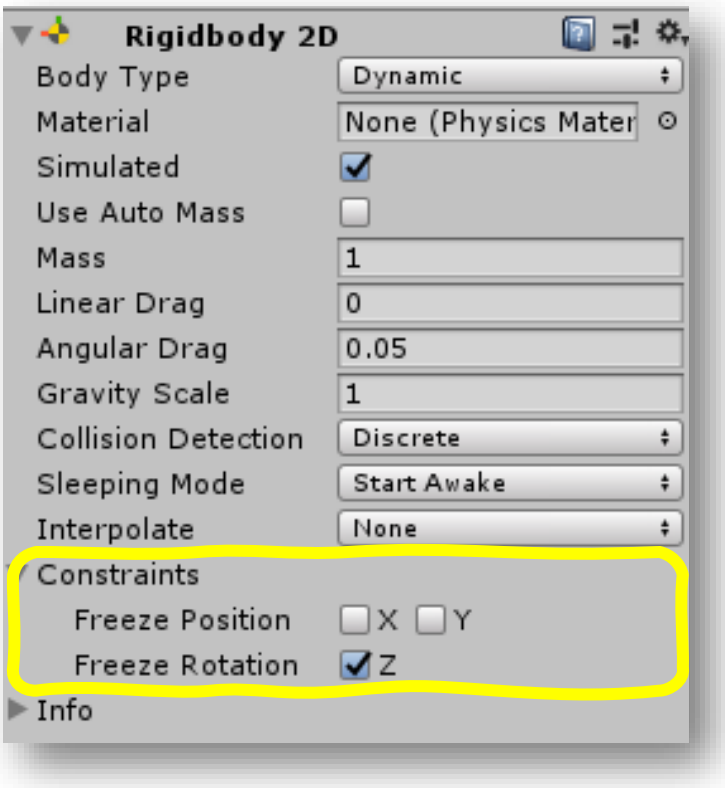


39 This **crusher** is now in the game, but it does not have a **BoxCollider2D** yet. Click **Add Component** to add in the component.

40 We also want to add in the Rigidbody component. Click on **Add Component** and search for "**rigid**" and add a **Rigidbody 2D component** to **Crusher2**.

41 The **Crusher2** game object now has a **Rigidbody 2D** component. The only change we need to make is to **freeze rotation** on the **z axis**.

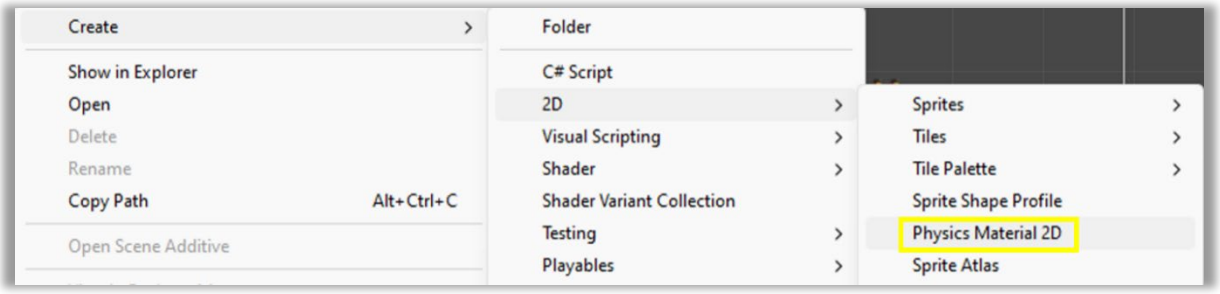
Open the **Constraints** dropdown and click the **Z checkbox** next to **Freeze Rotation**.



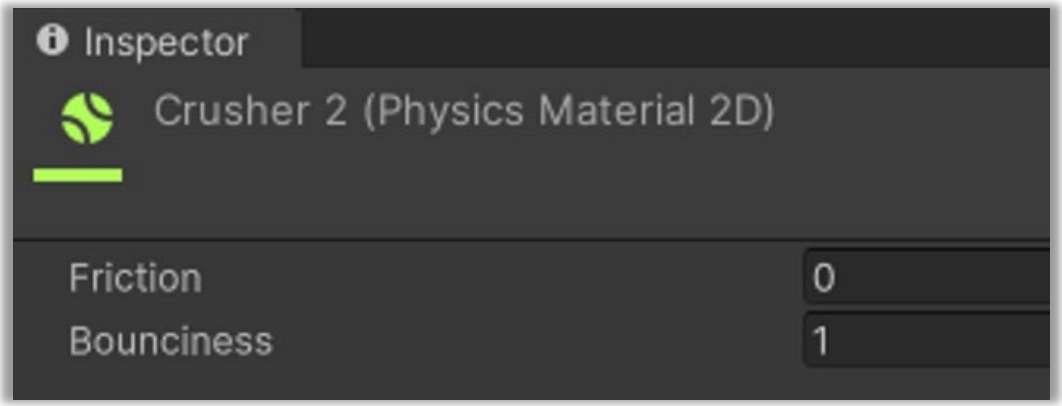
42 **Play** your game and experiment! How does Crusher2 behave? Does it bounce around like Crusher1?

43 In order to tell Unity how a rigidbody object should behave, we need to create a physics material. This has some properties to determine how bouncy an object is, and how much friction applies to it.

44 Right click in the Assets folder and create a new physics material by clicking in **Create > 2D > Physics Material 2D**. Name this material "Crusher2"



45 Select the physics material. In the Inspector, set the friction to 0 and bounciness to 1.



46 Drag the physics material onto Crusher2 in the Hierarchy.

47 Play your game! Do you notice the difference between the two crushers?

48 Let's give Crusher2 a bit of a push. Create a script called "**Crusher2**" in the scripts folder.

49 Open the script and remove the `Update` method.

```
public class Crusher2 : MonoBehaviour
{
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

    }
}
```

50 Just like Crusher1, create a speed `float` variable, but set this one at `5`.

```
public float speed = 5f;
```

51 In the `Start` method, get the rigidbody component using `GetComponent`, then we add force using the `AddForce` rigidbody method. We will add force to the right with a mode of `Impulse`. Impulse means the force is applied right away instead of over time.

```
public class Crusher2 : MonoBehaviour
{
    public float speed = 5f;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        GetComponent<Rigidbody2D>().AddForce(Vector2.right * speed, ForceMode2D.Impulse);
    }
}
```

52 In Unity, add the Crusher2 script to the GameObject.

53 Play your game. Crusher2 should be bouncing all around the screen!

Tell your sensei about when you think we might manually move objects like Crusher1, and when we might use rigidbody like Crusher2?