// Lesson 3 - Physical Events

> What you will learn:

- How to control a game logic by reacting to collisions and trigger overlapping
- Differences between a collider and a trigger
- Usage of Events in Type-/Javascript

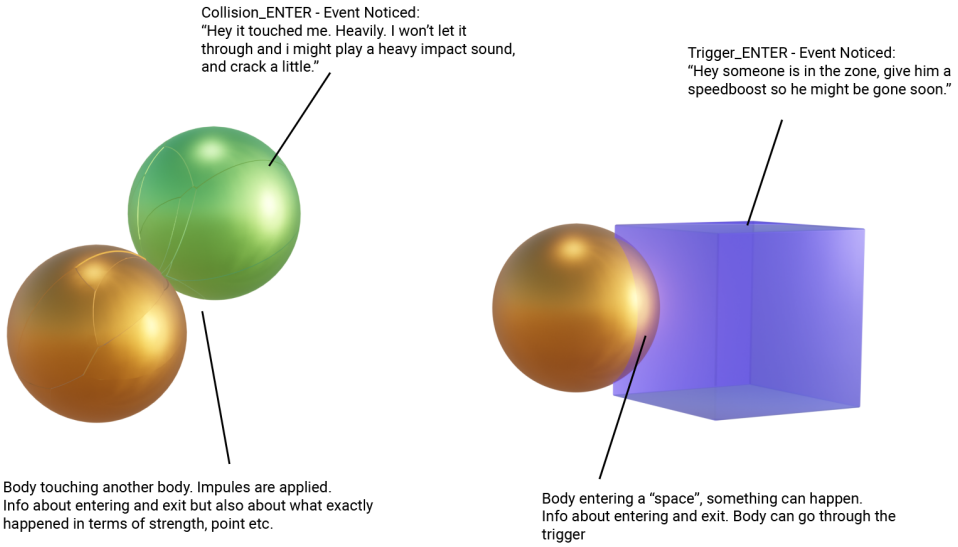
> Requirements Knowledge/Files:

- Knowledge how to setup a (physics) scene in Fudge
- Understanding of the basic physics, learned in lesson 1
- Hint! - You can use the physics boilerplate in the tutorial folder

> TL;DR; - But it's worth to read the long text, to get detailed infos and deeper understanding

- Physical Events are either a body is in a defined shape, trigger, or touching another body, collision
- Physical Events are constructed like normal Javascript Events but need to be listened in on the ComponentRigidbody
- 2 Types per Event, Enter/Exit you find in EVENT_PHYSICS enum
- They return the same variable type EventPhysics but filled with different infos

> Step 1 - What is a physics event and how to use it



Simplified, physical events are like other javascript events, they are happening when a condition is met and happening for those objects that are listening to the event. But they return a lot of different information and have differences in usage.

The general structure of a event is therefore:

```
objectListeingToTheEvent.addEventListener("eventType",
functionReceivingTheEvent);
functionReceivingTheEvent(_event : Event) : void {
}
```

Now for physics events you have 4 possible events that can be found in EVENT_PHYSICS enum of Fudge. A physical body can either collide with an object or trigger it by overlapping with it. In general objects do not overlap since they are rigid and real rigid objects can only touch and let go, but not go through. But there is a thing in game development called a trigger which is nothing else than a normal body but with no collision instead it checks if something has entered, or left, the space it's possessing.

So these are the 4 types of events, either a enter, or exit on a collider, or trigger.

Since they are part of the physics system within Fudge they need to be listened to in the physics system and not in the Node system. As a result we add it to the ComponentRigidbody.

```
node.getComponent(f.ComponentRigidbody).addEventListener(f.EVENT_PHYSICS.COLLISION_ENTER, functionReceivingTheEvent);
```

And also the function that is handling what's happening when the event is fired, is receiving a specific physical event.

```
functionReceivingTheEvent(_event : f.EventPhysics) : void {
}
```

Now that's simply how events are listened to and handled. But there are things to know about.

First, what is an event good for? Mainly to create, control your gamelogic. Something like if a player enters a trigger a key button press does a specific thing, only while in the trigger. In game development that is called context sensitive button mapping. You'll have noticed this probably a dozen times in games where you can only talk to a character when standing in front of him and near him for example.

Similar to triggers are collision events, they are needed for things like the player receiving dmg when touching something or adding atmosphere by playing sounds on falling from a height, and also an particle effect when crashing in a wall with your car.

Second, Collision-Events are only fired when a dynamic body hits a dynamic, static or kinematic body. Since a kinematic body goes through static bodies it does not have any collisions with those, only with dynamic bodies.

On the other hand everything fires a Trigger-Event but triggers are special bodies they are marked with the isTrigger property keep that in mind. To create a trigger, you need to create your body as a trigger first and then add the EventListener for trigger events. The Body then behaves like a standard body in terms of forces etc, but does not collide with anything, or transfer impuls.

```
let cmpRigidbody: f.ComponentRigidbody = new f.ComponentRigidbody(mass, physicsType, colliderType);
cmpRigidbody.isTrigger = true
```

```
node.addComponent(cmpRigidbody);

node.getComponent(f.ComponentRigidbody).addEventListener(f.EVENT_PHYSICS.TRIGGER_ENTER, hndOnTriggerEnter);
```

Third, collisions and triggers both receive the same EventPhysics event but the information received vary so keep that in mind.

> Step 2 - Handling the information received

As noted the function that is handling the event is receiving a plethora of information in the form of a EventPhysics variable. This information varies, think of it this way. A triggering event is entering a space that is defined but not solid, (that's why triggers are often invisible, meaning no mesh component). You therefore get information about who is involved in this event, and where it did happen.

```
_event.collisionPoint; //Where in Worldspace
_event.cmpRigidbody; //Who fired the event
```

But a collision is one body telling the other to leave the occupied space, resulting in forces/impulses happening.
The extra information you can get is therefore information about the intensity of the happening.

```
_event.collisionNormal; //The direction the collision is coming from / going to
_event.normalImpulse; //The intensity of this collision
```

There are two more impulse intensities the binormal and the tangent impulse but those are for very specific use cases.

> Step 3 - Practical example for a trigger and a collision

For this it's expected that Lesson 1 is of course known. Create 4 bodies, 1 dynamic, 3 static. First a ground floor where our dynamic body is walking on. 1 Static cube that we use for a collider event and one static cube for the trigger-event.
As a refresher:
We have built a function to create complete Nodes with Transform, Mesh, Material, Rigidbody. And giving it the parameters for the physicsType, physicsGroup and such. So creating our scene is basically only 4 calls of this function and storing the resulting node in a reference so we can access it easily later, for positioning. Be sure to add the trigger to the correct physics group.

```
triggerNode = createCompleteNode("Trigger", triggerMaterial, new
f.MeshCube(), 1, f.PHYSICS_TYPE.STATIC);
```

After building our scene we now only have to add events to our bodies and move our player cube to fire the events.

To fire events we learned to add an event listener directly to the ComponentRigidbody with the handling function and type of event.

```
collisionNode.getComponent(f.ComponentRigidbody).addEventListener(
f.EVENT_PHYSICS.COLLISION_ENTER, hndCollisionEventEnter);

collisionNode.getComponent(f.ComponentRigidbody).addEventListener(
f.EVENT_PHYSICS.COLLISION_EXIT, hndCollisionEventExit);


triggerNode.getComponent(f.ComponentRigidbody).isTrigger = true
triggerNode.getComponent(f.ComponentRigidbody).addEventListener(
f.EVENT_PHYSICS.TRIGGER_ENTER, hndTriggerEventEnter);

triggerNode.getComponent(f.ComponentRigidbody).addEventListener(
f.EVENT_PHYSICS.TRIGGER_EXIT, hndTriggerEventExit);
```

Now we need to handle the incoming events:

```
function hndCollisionEventEnter(_event: f.EventPhysics): void {
//If our Event is happening with the body NODE called "Player"
  if (_event.cmpRigidbody.getContainer().name == "Player") {
```

```
//We let this collider act like a bumper through this event. We
take the event normal and shoot the player away from the bumper on
the contact point.

playerBody.getComponent(f.ComponentRigidbody).applyForceAtPoint(
new f.Vector3(_event.collisionNormal.x * 500,
              _event.collisionNormal.y * 500,
              _event.collisionNormal.z *
              500), _event.collisionPoint);
   }
}

function hndCollisionEventExit(_event: f.EventPhysics): void {
  if (_event.cmpRigidbody.getContainer().name == "Player") {
      f.Debug.log("Player left me - Collider");
    }
}

function hndTriggerEventEnter(_event: f.EventPhysics): void {
  if (_event.cmpRigidbody.getContainer().name == "Player") {
     f.Debug.log("Player entered me - Trigger");
//We get the rigidbody from the event and take the material
component of the corresponding node and change it, just to show
the effect
     _event.cmpRigidbody.getContainer().
            getComponent(f.ComponentMaterial)
            .material = playerTriggeredMat;
  }
}

function hndTriggerEventExit(_event: f.EventPhysics): void {
  if (_event.cmpRigidbody.getContainer().name == "Player") {
     f.Debug.log("Player left me - Trigger");
     _event.cmpRigidbody.getContainer().
            getComponent(f.ComponentMaterial)
            .material = playerDefaultMat;
  }
}
```

We are moving our player with a standard Fudge keypress that should be known. And just add some impulses to push the cube. We define the 3 input axis and on a specific key_code we set the input to a impulse we want.

```
playerBody.getComponent(f.ComponentRigidbody).applyLinearImpulse(
new f.Vector3(horizontalInput, heightInput, verticalInput));
```

Using physical events is easy and intuitive but the possibilities are limitless, it's a basic tool for every game developer.

Hint! - This concludes this tutorial but keep in mind not everything is explained in detail you can explore things better by yourself! It's helpful to look at the complete examples in the tutorial folder to have a in depth look.