Lab 2: Working with Physics and Rotation

### This is the lab consists of fixing an already existing scene. In order to download the project you will have to clone this git repository :

<https://classroom.github.com/a/oGl1ZDwr>

## Setup

## Accept the assignment



## Click on the created assignment link

* 
* Then clone the created repo with github desktop.
* 
* 
* 
* Choose a folder and clone the repo
* 
* Once the folder is cloned open it in Unity by opening the scene

## Open the project in Unity.

* + In the scene folder double click on the sample scene
  + Click convert in Place once the project is converted.
  + Open the Scene by double clicking on sample Scene.
  + 

## 

## 

## Task 1: Adjust Object Physics Settings (1 pt)

The project should open to a level that looks like this :

### **Green Goal Object**:

* + Select the green goal object called EndGoal in the scene Hierachy.
  + Make sure that it is set as a **static** and a **Trigger** object:
    - In the Inspector, check that the object is **not** using Rigidbody.
    - Add a **Collider** (if not already present) and check the box for **Is Trigger**.
    - 

1. **Red Player Object**:
   * Select the player object (the red ball) .
   * Ensure it is a **dynamic** object:
     + Add a **Rigidbody** component (if not already present).
     + Make sure the **Use Gravity** option is checked.
     + Set the **Is Kinematic** property to **false**.
2. 
3. **Yellow Trap Object**:
   * Select the yellow trap object.
   * Ensure it is **NOT** a **Trigger** by making sure the IsTrigger Option is **NOT** checked.
   * 
   * Ensure it is a **kinematic** object:
     + Add a **Rigidbody** component.
     + Set **Is Kinematic** to **true** and **Use Gravity** to **false**.



#### **5.Setup the death Plane**

* + Select the Blue object called Death plane in the scene Hierachy.
  + 
  + Make sure that it is set as a **static** and a **Trigger** object:
    - In the Inspector, check that the object is **not** using Rigidbody.
    - Add a **Collider** (if not already present) and check the box for **Is Trigger**.



#### **Task 2: Create Player Movement Script (1 pt)**

1. Create a new C# script named PlayerController.
2. Attach this script to the red player object (the red ball).
3. Open the script and use the following code to move the player using the Rigidbody based on user input:

using UnityEngine;

public class PlayerController : MonoBehaviour

{

public float moveSpeed = 10f;

private Rigidbody rb;

void Start()

{

rb = GetComponent<Rigidbody>();

}

void FixedUpdate() // Physics-related operations should go here

{

// Get input for horizontal and vertical axes

float moveX = Input.GetAxis("Horizontal");

float moveZ = Input.GetAxis("Vertical");

// Calculate the movement direction based on the player's forward and right vectors

Vector3 movement = (transform.right \* moveX) + (transform.forward \* moveZ);

// Apply movement to the rigidbody

rb.AddForce(movement \* moveSpeed, ForceMode.Force);

}

}

1. Save the script and test the movement by playing the scene.

5. What happens when you try to move the ball ? :   
  
The ball will accelerate very rapidly without any delay making it a little janky. Because of this changing directions does not feel great.

6. Lets fix the ball movement. Right now the weight of the ball is too small for the force we apply on it. So the acceleration that the physics engine impart to the ball is very high so lets make our Player weight bigger

* Change the mass parameter in the RigidBody Component
* 

Let set it to 1 kg

**Rigidbody.MoveRotation**

In the inspector you can change the movespeed if you want to move faster

#### **Task 3: Add a Physics Material to the Ball (1 pt)**

1. Create a **Physics Material**:
   * Right-click in the **Project** window and select Create > Physics Material.
   * 
   * Name it BouncyMaterial.
   * Add it to a new folder called PhysicsMaterial
2. Double click on BouncyMaterial and
3.   
   Set the **Bounciness** value to 1 to make the ball extra bouncy.
4. 
5. Assign the BouncyMaterial to the red ball’s **Collider** component.
   * 

#### **Task 4: Rotate the Yellow Trap (1 pt)**

1. **Initial Attempt (with Transform.Rotate)**:
   * Create a new C# script called TrapController.
   * Attach the script to the yellow trap object.
   * Use the following code to try rotating the trap with Transform.Rotate:

using UnityEngine;

public class TrapController : MonoBehaviour

{

public float rotationSpeed = 100f;

void Update()

{

transform.Rotate(0, rotationSpeed \* Time.deltaTime, 0);

}

}

In the inspector set the rotation speed to 1000



Now move your player to the middle of the trap what do you notice ?

The player phases through the trap, because the trap is moving so quickly due to Update() and the object doesn’t get enough time to check if it is not there.

This issue is due to the fact that when were modifying the transform directly. It does not check if we have a collision in the way. So if the movement is fast enough the collision gets skipped. You can notice this issue closser if you use the pause button and then update frame by frame.

One way to correct for that is to use Continuous Collision Detection but it is very heavy for the CPU so we will use another method that allows collision to be properly updated

1. **Fix Rotation (Rigidbody.MoveRotation )**:
   * Notice clipping issues with the initial attempt.
   * In order to make sure that during a rotation the physics calculation are done properly we will use the RigidBody.MoveRotation
   * Modify the code to rotate the object using **Rigidbody.MoveRotation**:

using UnityEngine;

public class TrapController : MonoBehaviour

{

public float rotationSpeed = 100f;

private Rigidbody rb;

void Start()

{

rb = GetComponent<Rigidbody>();

}

void FixedUpdate()

{

Quaternion deltaRotation = Quaternion.Euler(0, rotationSpeed \* Time.fixedDeltaTime, 0);

rb.MoveRotation(rb.rotation \* deltaRotation);

}

}

* **Set the rotation speed back down to 100 now.**
* 

#### **Task 5: Detect Player Reaching the Goal (1 pt)**

1. Add a tag named Goal to the green goal object:
   * Select the green goal object, go to the **Inspector**, and assign the Goal tag.
   * 
   * If one doesnt exist create one. With the AddTag option
   * 
   * 
   * Press the + button to add a tag and Name it Goal
2. Modify the PlayerController script to detect collisions by adding this method to the Player Controller Script:

void OnTriggerEnter(Collider other)

{

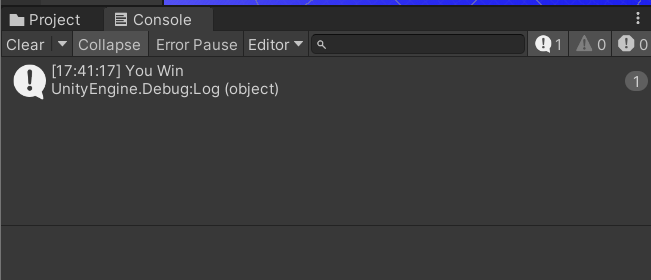
if (other.CompareTag("Goal"))

{

Debug.Log("You Win");

}

}

1. Test to ensure the message "You Win" appears in the console when the player touches the green goal.
2. The line if (other.CompareTag("Goal")) allows us to check those tags on our game object to differentiate the different objects.
3. **Add a screenshot here of the debug :**

#### **Task 6: Restart Game on Death (1 pt)**

1. Add a tag named DeathPlane to the blue death plane object.
2. 
3. 
4. Click Add Tag …
5. 
6. Then the plus ane write deathplane
7. 
8. Then select it from the same drop dowm option.
9. Then Modify the PlayerController script to teleport the player back to its original position when it touches the death plane by adding these lines of code before the FixedUpdate loop:

private Vector3 startPosition;

void Start()

{

rb = GetComponent<Rigidbody>();

startPosition = transform.position;

}

void OnTriggerEnter(Collider other)

{

if (other.CompareTag("Goal"))

{

Debug.Log("You Win");

}

else if (other.CompareTag("DeathPlane"))

{

rb.position = startPosition;

}

}

1. Test to ensure the player is reset to the original position when touching the death plane.

How to submit your project :

1. Add this filled in document to your project folder
2. Submit your code on github. I should see your repo on github classroom.