BOWLING GAME

# DEVANGMORE 22000224

# KEVIN VYAS 22000241

**Step 1: Setting Up the Scene**

**1. Create a New Unity Project:**

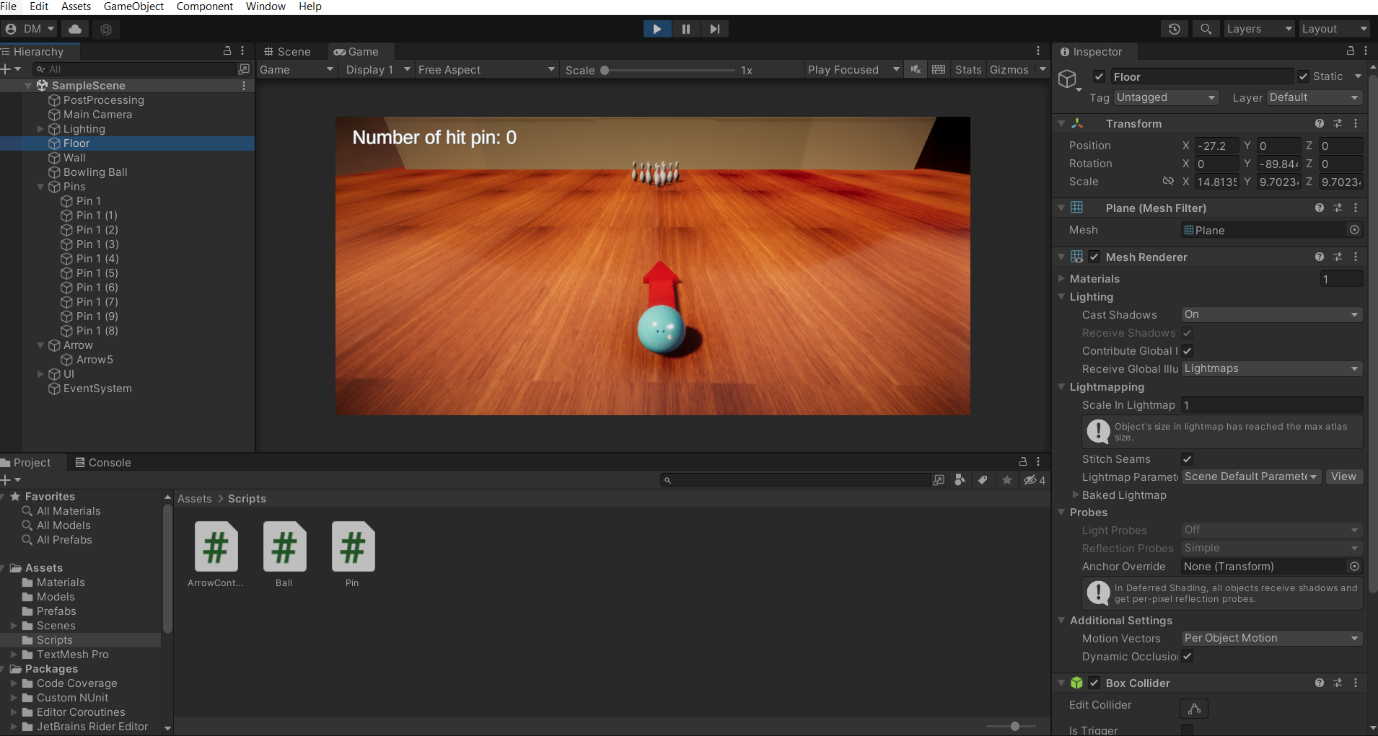
Open the Unity Hub, select new 3D project, and name it "BowlingGame".

**2. Set Up the Floor (Bowling Alley):**

GameObject > 3D Object > Plane.

- Rename the plane to "Floor" and resize it so it's as long as a bowling alley, e.g. `X=3, Y=1, Z=10`.

- Position the floor to `(0, 0, 0)` and give it the look of wood by adding a material.

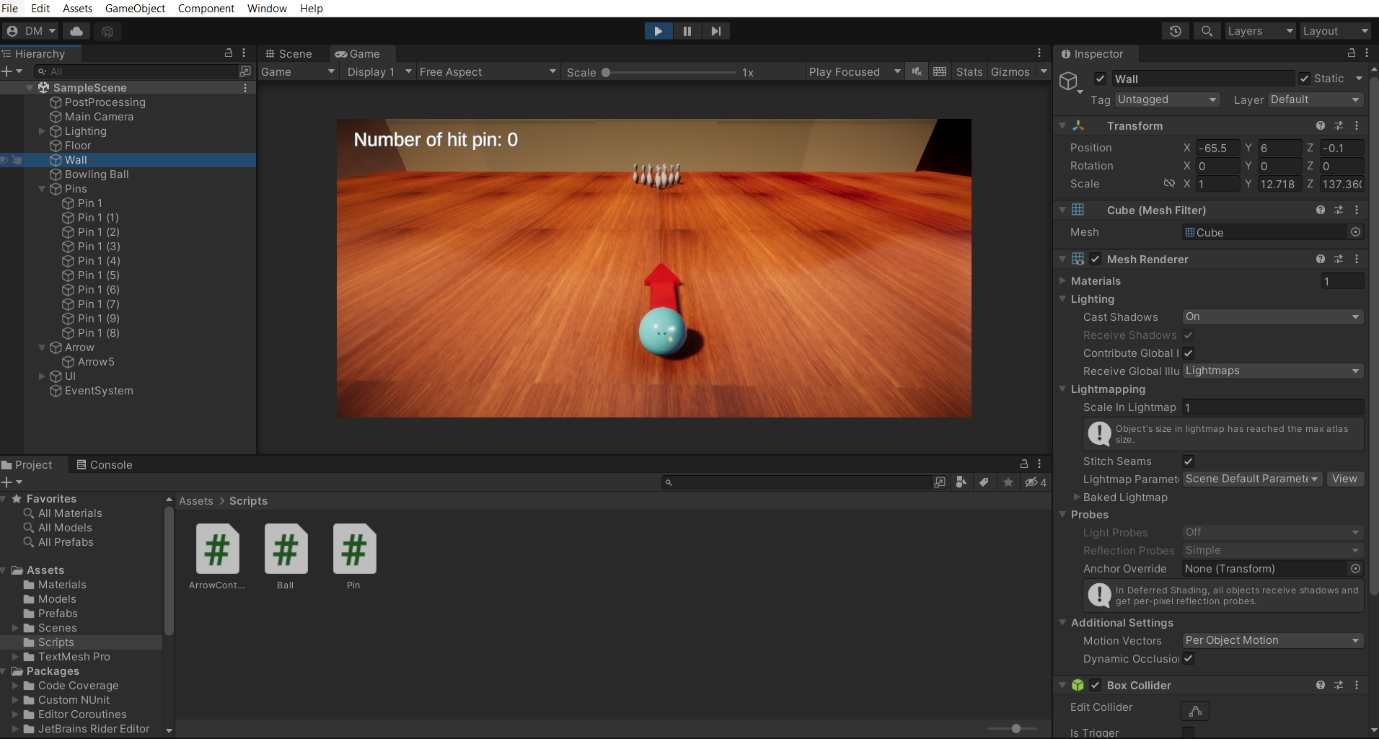


**3. Add Walls**

- Two 3D Cubes for the side walls.

- Stretch each wall along the `Z-axis` to position them on both sides of the alley to make sure the ball bounces inside from each side.

Rename them as "WallLeft" and "WallRight" for a clearer reason.



**Step 2: Create the Bowling Ball**

**1. Add Ball:**

- GameObject > 3D Object > Sphere to create your bowling ball.

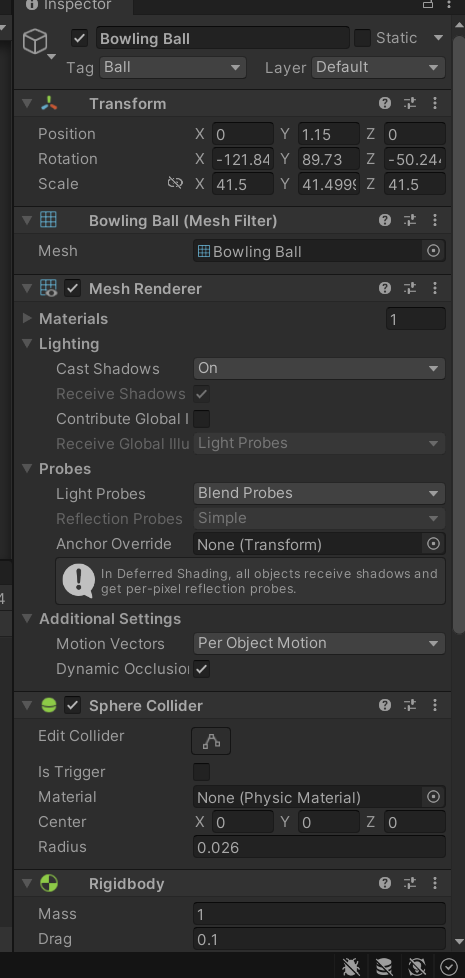
- Rename to "BowlingBall" and scale to size for a ball. For example, `X=0.3, Y=0.3, Z=0.3` so it looks like a true bowling ball.

- Move it into position at the start near the bottom of the alley.

**2. Add Rigidbody to Ball:**

- Select the "BowlingBall" and add a Rigidbody component (found under Inspector > Add Component).

Set Mass to a reasonable value (e.g. `1`), adjust Drag and Angular Drag if needed.



**3. Make a Script to Control the Bal:**

In Project window, navigate to Assets > Script; create the folder if doesn't exist.

You right click inside the folder and choose Create C# script, name it BallController. Attach this to the "BowlingBall" and open it up. It's a pretty elementary script for controlling the ball with arrow keys:

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Xml.Serialization;

using TMPro;

using UnityEngine;

using UnityEngine.SceneManagement;

public class Ball : MonoBehaviour

{

public Rigidbody rb; // reference to the Rigidbody component of the ball

public float startSpeed = 40f; // the speed at which the ball starts moving

private Transform \_arrow;

private bool \_ballMoving;

private Transform \_startPosition;

private List<GameObject> \_pins = new();

private readonly Dictionary<GameObject, Transform> \_pinsDefaultTransform = new();

public int Point { get; set; }

[SerializeField] private Animator cameraAnim;

private TextMeshProUGUI feedBack;

private void Start()

{

Application.targetFrameRate = 60;

\_arrow = GameObject.FindGameObjectWithTag("Arrow").transform;

// get the reference to the Rigidbody component of the ball

rb = GetComponent<Rigidbody>();

\_startPosition = transform;

\_pins = GameObject.FindGameObjectsWithTag("Pin").ToList();

foreach (var pin in \_pins)

{

\_pinsDefaultTransform.Add(pin, pin.transform);

}

feedBack = GameObject.FindGameObjectWithTag("FeedBack").GetComponent<TextMeshProUGUI>();

}

void Update()

{

if (\_ballMoving)

{

return;

}

if(Input.GetKeyDown(KeyCode.Space))

{

StartCoroutine(Shoot());

}

}

private IEnumerator Shoot()

{

cameraAnim.SetTrigger("Go");

cameraAnim.SetFloat("CameraSpeed", \_arrow.transform.localScale.z);

\_ballMoving = true;

\_arrow.gameObject.SetActive(false);

rb.isKinematic = false;

// calculate the force vector to apply to the ball

Vector3 forceVector = \_arrow.forward \* (startSpeed \* \_arrow.transform.localScale.z);

// calculate the position at which to apply the force (in this case, the center of the ball)

Vector3 forcePosition = transform.position + (transform.right \* 0.5f);

// apply the force at the specified position

rb.AddForceAtPosition(forceVector, forcePosition, ForceMode.Impulse);

yield return new WaitForSecondsRealtime(7);

\_ballMoving = false;

GenerateFeedBack();

yield return new WaitForSecondsRealtime(2);

ResetGame();

}

private static void ResetGame()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().name);

}

private void GenerateFeedBack()

{

feedBack.text = Point switch

{

0 => "Nothing!",

> 0 and < 3 => "You are learning Now!",

>= 3 and < 6 => "It was close!",

>= 6 and < 10 => "It was nice!",

\_ => "Perfect! You are a master!"

};

feedBack.GetComponent<Animator>().SetTrigger("Show");

}

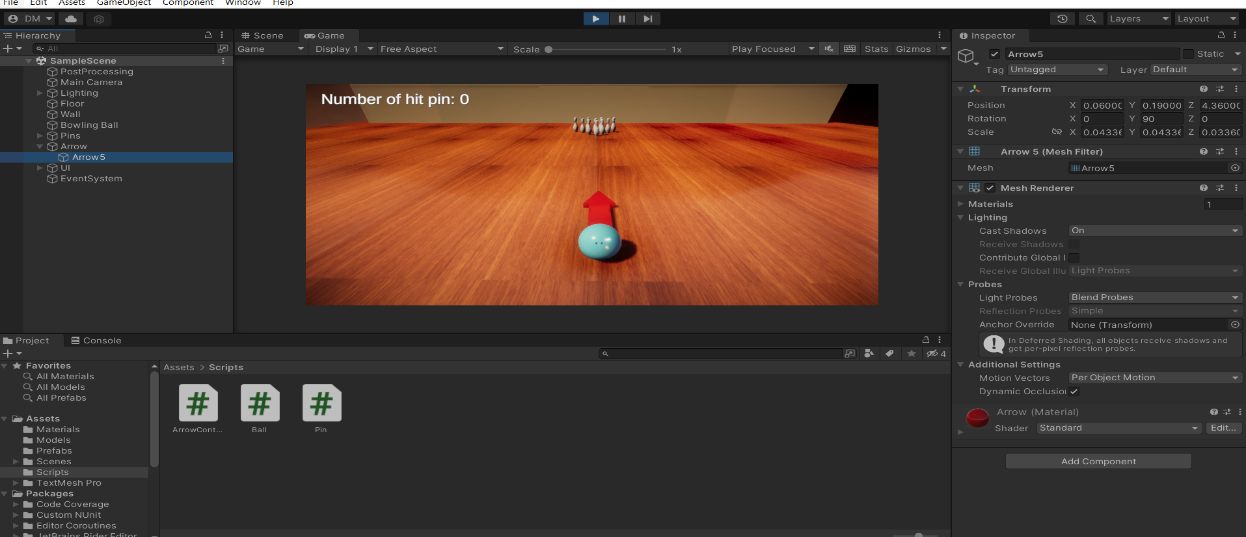
}

**4. Add an Arrow To Indicate Direction**

Make an arrow by 3D Object > Cylinde (scale and rotate it to point forward) or import an arrow model.

Attach it as a child of the BowlingBall so it moves and rotates with the ball.

This arrow can serve as a visual indicating tool for aiming.



**Arrow Controls**

To shift the arrow from left to right use the arrow key .

To drag the ball, press enter key.

**Code for Arrow Controll**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ArrowController : MonoBehaviour

{

void Update()

{

if (Input.GetKey(KeyCode.LeftArrow))

{

transform.Rotate(Vector3.down, Time.deltaTime \* 30f);

}

if (Input.GetKey(KeyCode.RightArrow))

{

transform.Rotate(Vector3.up, Time.deltaTime \* 30f);

}

if (Input.GetKey(KeyCode.UpArrow))

{

if (transform.localScale.z < 2)

{

transform.localScale = new Vector3(transform.localScale.x, transform.localScale.y,

transform.localScale.z + (1 \* Time.deltaTime));

}

else

{

transform.localScale = new Vector3(transform.localScale.x, transform.localScale.y, 2);

}

}

if (Input.GetKey(KeyCode.DownArrow))

{

if (transform.localScale.z > 0.1f)

{

transform.localScale = new Vector3(transform.localScale.x, transform.localScale.y,

transform.localScale.z - (1 \* Time.deltaTime));

}

else

{

transform.localScale = new Vector3(transform.localScale.x, transform.localScale.y, 0.1f);

}

}

}

}

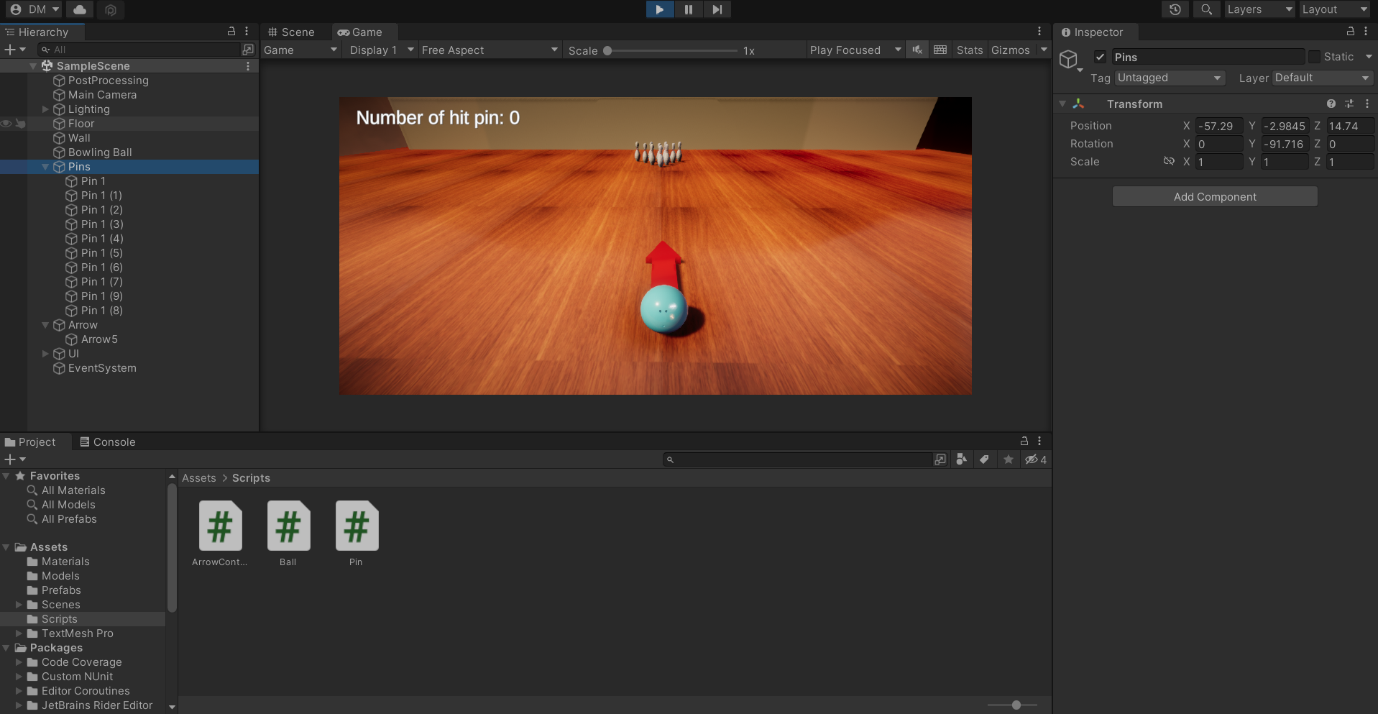
**Step 3: Create the Pins**

**1. Create a Pin:**

You can make pins using 3D Object > Cylinder (it kind of resembles a pin) or by importing your own models for the pins.

Add a Rigidbody component to each pin so that, when the ball strikes them, they fall over.

Each pin should have a Collide (Capsule Collider or custom collider) that covers its shape.



**2. Try The Pins in a Triangular Shape**

- Create a copy of the pin model and set them in a triangular shape at the end of the lane.

- For the standard 10-pin, create four rows - 4, 3, 2, 1

**3. Write a Script that Tracks the Number of Fallen Pins**

**4. Label the Bowling Ball:**

- Select the BowlingBall object and assign to it the tag "BowlingBall you can create a new tag if "BowlingBall" is not already created.

**Step 4: Creating a Score System**

**1. Create a UI Text for Score Display:**

- In Hierarch, find GameObject > UI > Text - TextMeshPr (you may be prompted to import TextMeshPro to continue).

- Name it "ScoreText" and drag it up to the top of the screen.

- Set the text to read "Number of hit pins: 0".

Script Folder

Create a new script, called ScoreManage

Attach it to an empty game object named "GameManager" in your hierarch

Add the below code

using System;

using System.Collections;

using System.Collections.Generic;

using TMPro;

using UnityEngine;

public class Pin : MonoBehaviour

{

private bool \_done;

private void OnCollisionEnter(Collision collision)

{

if ((collision.collider.CompareTag("Ball") || collision.collider.CompareTag("Pin")) && !\_done)

{

// get the velocity of the pin after the collision

float velocity = GetComponent<Rigidbody>().velocity.magnitude;

// check if the velocity has dropped below the fall threshold

if (velocity < 10)

{

var point = GameObject.FindGameObjectWithTag("Ball").GetComponent<Ball>().Point;

point += 1;

GameObject.FindGameObjectWithTag("Poing").GetComponent<TextMeshProUGUI>().text = $"Number of fallen pins: {point}";

GameObject.FindGameObjectWithTag("Ball").GetComponent<Ball>().Point = point;

\_done = true;

}

}

}

}

**3. Associate UI Text with ScoreManage**

For theInspector of ScoreManager, drag the\"ScoreText" UI element into the `ScoreText` field.

**Step 5: Testing and Fine-Tuning Physics**

**1.Test Scene**

Run the game in Play Mod and see if everything works alright. Roll the ball and try to knock the pins down in an attempt to test whether scoring happens the correct way or not.

- The ball, as well as the pins should respond to input appropriately.

- Adjust Rigidbody of both, on the pins and on the ball, to make the physics appear realistic

Play around with Drag, Angular Drag and Mass on every object

**3. Fine-Tuning Pin Script**

- If pins fail to regular detection of collision with the ball change On Collision Enter logic to take into account any possible other triggers as well.

**Step 6: Add Reset and Replay Features**

**1. Implement a Reset Method:**

Create a reset button in the UI after each game round, which calls a method in ScoreManager to reset the score and reposition the pins and the ball for a new round.

**2. Add Sound Effects and Final Details:**

- Incorporate sound effects that happen whenever the ball hits the pins, and make the game more realistic.

Materials and lighting effects to be used to enhance visuals.

