Title: Projectile Motion of objects

EXPERIMENT NO: 7 **Name:** SHAKYA JAY JITENDRA

Date: 4-9-2025 **Reg. No:** 24BCG10123

Aim: To simulate the projectile motion of a cannonball in Unity using physics-based scripting, and to study how the launch angle and initial velocity affect the range and trajectory, thereby verifying the laws of projectile motion.

Description/Concept:

Scene Setup: Place a cannon on a flat plane. Create a child object at the mouth of the cannon and name it FirePoint.

Projectile Creation: Create a sphere, scale it down, add a Rigidbody component, and save it as Projectile.prefab.

Attach Scripts:

Add NozzleController.cs to rotate cannon with mouse scroll.

Add NozzleFire.cs to spawn and launch the projectile.

Configure Inspector: Assign Projectile.prefab to Projectile Prefab slot, assign FirePoint, and set launch speed (e.g., 40).

Execution: Press Play, adjust nozzle angle, and press Spacebar to fire.

Observation: Record the horizontal distance travelled before the projectile hits the ground for different angles.

Program/Coding:

1) Cannon Movement:

```
using UnityEngine;

public class CannonController : MonoBehaviour
{
    [Header("Movement Settings")]
    public float moveSpeed = 5f;   // Speed of movement
    public float rotateSpeed = 100f; // Speed of rotation

    void Update()
    {
        // --- Movement ---
        float moveZ = 0f;
        float moveX = 0f;

        if (Input.GetKey(KeyCode.W)) moveZ = 1f;   // Forward
        if (Input.GetKey(KeyCode.S)) moveZ = -1f;  // Backward
        if (Input.GetKey(KeyCode.D)) moveX = 1f;   // Right
        if (Input.GetKey(KeyCode.A)) moveX = -1f;  // Left

        Vector3 move = new Vector3(moveX, 0, moveZ).normalized;
        transform.Translate(move * moveSpeed * Time.deltaTime, Space.World);

    }
}
```

2) Cannon Fire

```
using UnityEngine;

public class CannonFire : MonoBehaviour
{
    [Header("References")]
    public Transform firePoint;               // Assign: the FirePoint (child at
muzzle)
    public GameObject projectilePrefab;       // Assign: Projectile prefab (must
have Rigidbody & Collider)

    [Header("Ballistics")]
    [Tooltip("Initial speed in meters/second (m/s). Try 30-60 for visible arcs.")]
    public float launchSpeed = 40f;
    [Tooltip("If true we set rb.velocity directly. If false, AddForce with
VelocityChange is used.")]
    public bool setVelocityDirectly = true;

    [Header("Fire Control")]
    public float reloadTime = 0.75f;          // seconds between shots
    public float projectileLifetime = 12f;    // auto destroy after this many
seconds
    public float spawnOffset = 0.12f;         // move spawn slightly forward so
it doesn't intersect the barrel

    [Header("Collision Safety")]
    public bool ignoreCannonCollisions = true; // when true will ignore
collisions between projectile and cannon colliders
    public Collider[] cannonCollidersToIgnore; // drag the cannon/base/nozzle
colliders here in Inspector

    float _nextFireTime = 0f;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space) && Time.time >= _nextFireTime)
```

```
            Fire();
    }

    public void Fire()
    {
        _nextFireTime = Time.time + reloadTime;

        if (projectilePrefab == null || firePoint == null)
        {
            Debug.LogWarning("CannonFire: projectilePrefab or firePoint not
assigned.");
            return;
        }

        Vector3 spawnPos = firePoint.position + firePoint.forward * spawnOffset;
        Quaternion spawnRot = firePoint.rotation;

        GameObject proj = Instantiate(projectilePrefab, spawnPos, spawnRot);

        // Ensure projectile has a collider and rigidbody
        Collider projCol = proj.GetComponent<Collider>();
        Rigidbody rb = proj.GetComponent<Rigidbody>();

        // Prevent instant collision with the cannon (if requested)
        if (ignoreCannonCollisions && projCol != null &&
cannonCollidersToIgnore != null)
        {
            foreach (var c in cannonCollidersToIgnore)
                if (c != null)
                    Physics.IgnoreCollision(projCol, c, true);
        }

        if (rb != null)
        {
            // sane rb settings for projectile
            rb.linearVelocity = Vector3.zero;
            rb.angularVelocity = Vector3.zero;
            rb.useGravity = true;
            rb.linearDamping = 0f; // no Unity linear drag for ideal parabola
            rb.collisionDetectionMode = CollisionDetectionMode.ContinuousDynamic;
            rb.interpolation = RigidbodyInterpolation.Interpolate;

            Vector3 initialVelocity = firePoint.forward.normalized * launchSpeed;

            if (setVelocityDirectly)
            {
                // exact initial velocity  best match to projectile motion
formula
                rb.linearVelocity = initialVelocity;
            }
            else
            {
                // change velocity accounting for mass (VelocityChange ignores
mass; Impulse would scale with mass)
                rb.AddForce(initialVelocity, ForceMode.VelocityChange);
            }
        }
        else
        {
            Debug.LogWarning("CannonFire: spawned projectile has no Rigidbody.");
        }

        Destroy(proj, projectileLifetime);
    }
}
```

3) Nozzle Controller

```
  using UnityEngine;

public class NozzleController : MonoBehaviour
{
    [Header("Rotation Settings")]
    public float rotationSpeed = 50f;
    public float minPitch = 0f;
    public float maxPitch = 60f;

    [Range(0f, 360f)] public float yawAngle = 0f;  // left/right rotation in
inspector

    private float currentPitch = 0f;   // up/down tilt

    void Update()
    {
        float scroll = Input.GetAxis("Mouse ScrollWheel");

        // Left mouse button = pitch (up & down)
        if (Input.GetMouseButton(0) && scroll != 0)
        {
            currentPitch += scroll * rotationSpeed;
            currentPitch = Mathf.Clamp(currentPitch, minPitch, maxPitch);
        }

        // Right mouse button = yaw (left & right, full 360°)
        if (Input.GetMouseButton(1) && scroll != 0)
        {
            yawAngle += scroll * rotationSpeed;
            if (yawAngle > 360f) yawAngle -= 360f;
            if (yawAngle < 0f) yawAngle += 360f;
        }

        // Apply both rotations (pitch on X, yaw on Y)
        transform.localRotation = Quaternion.Euler(-currentPitch, yawAngle, 0f);
    }
}
```

4) Projectile Tracker

```
  using UnityEngine;

public class ProjectileTracker : MonoBehaviour
{
    private Vector3 startPos;
    private float pathTravelled = 0f;
    private Vector3 lastPos;
    private bool hasReported = false;

    void Start()
    {
        startPos = transform.position;
        lastPos = startPos;
    }

    void Update()
    {
        // Accumulate the actual path length (arc distance)
        pathTravelled += Vector3.Distance(transform.position, lastPos);
        lastPos = transform.position;
    }

    void OnCollisionEnter(Collision collision)
    {
```

```
        if (hasReported) return; // avoid multiple triggers
        hasReported = true;

        // Horizontal range (ignoring height difference, like physics class)
        Vector3 flatStart = new Vector3(startPos.x, 0f, startPos.z);
        Vector3 flatHit = new Vector3(transform.position.x, 0f,
transform.position.z);
        float horizontalRange = Vector3.Distance(flatStart, flatHit);

        Debug.Log($"  Projectile hit {collision.gameObject.name}");
        Debug.Log($"➡ Horizontal Range: {horizontalRange:F2} meters");
        Debug.Log($"  Total Path Travelled: {pathTravelled:F2} meters");

        // Optionally destroy after short delay
        Destroy(gameObject, 0.2f);
    }

    void OnDestroy()
    {
        // Safety check: if destroyed by lifetime and not collision
        if (!hasReported)
        {
            Vector3 flatStart = new Vector3(startPos.x, 0f, startPos.z);
            Vector3 flatEnd = new Vector3(transform.position.x, 0f,
transform.position.z);
            float horizontalRange = Vector3.Distance(flatStart, flatEnd);

            Debug.Log("⚠ Projectile destroyed (lifetime expired)");
            Debug.Log($"➡ Horizontal Range: {horizontalRange:F2} meters");
            Debug.Log($"  Total Path Travelled: {pathTravelled:F2} meters");
        }
    }
}
```

Observations

(Assume Launch Speed = 40 units/s, Ground level = 0, Unity gravity = 9.81 m/s^2)

Launch Angle (θ)	Time of Flight (T)	Max Height (H)	Range (R)	Path Shape
15°	~2.1 s	~5.4 m	~82 m	Low arc
30°	~4.1 s	~20.4 m	~141 m	Higher arc
45°	~5.7 s	~40.8 m	~163 m	Maximum range
60°	~7.1 s	~61.2 m	~141 m	Steep arc
75°	~7.9 s	~78.5 m	~82 m	Very steep

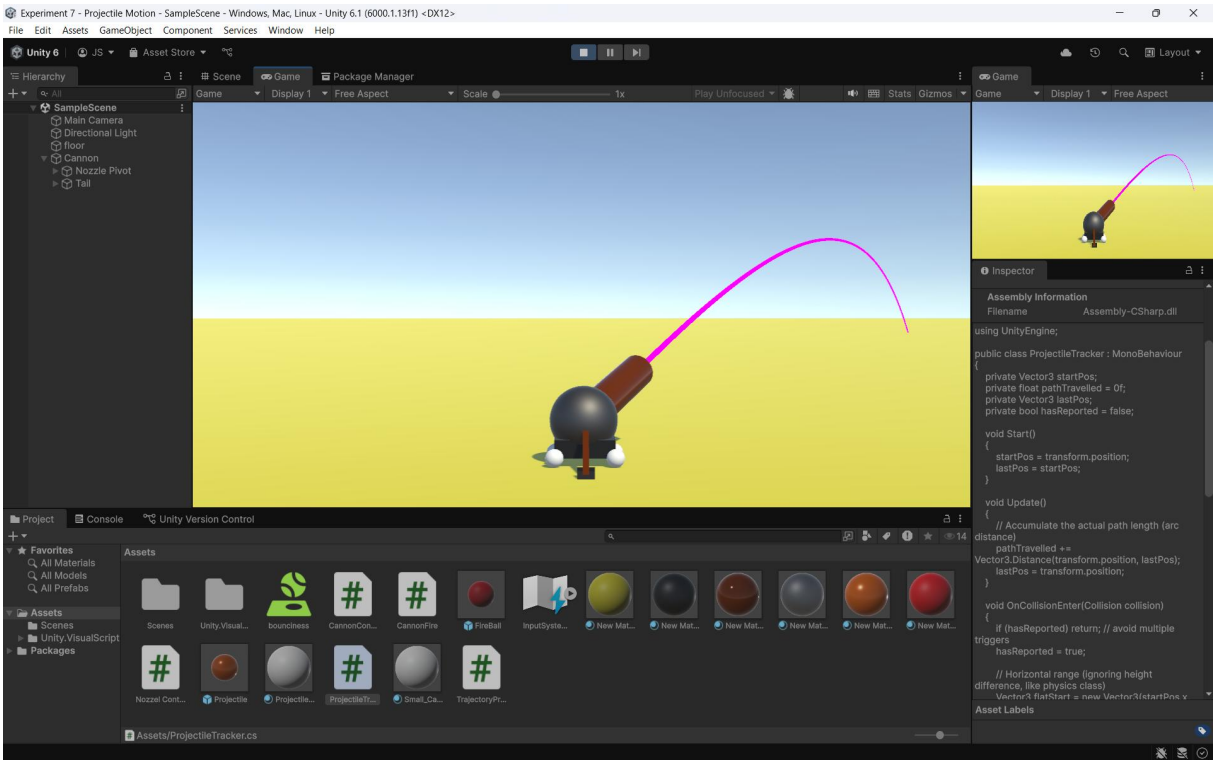
Output:

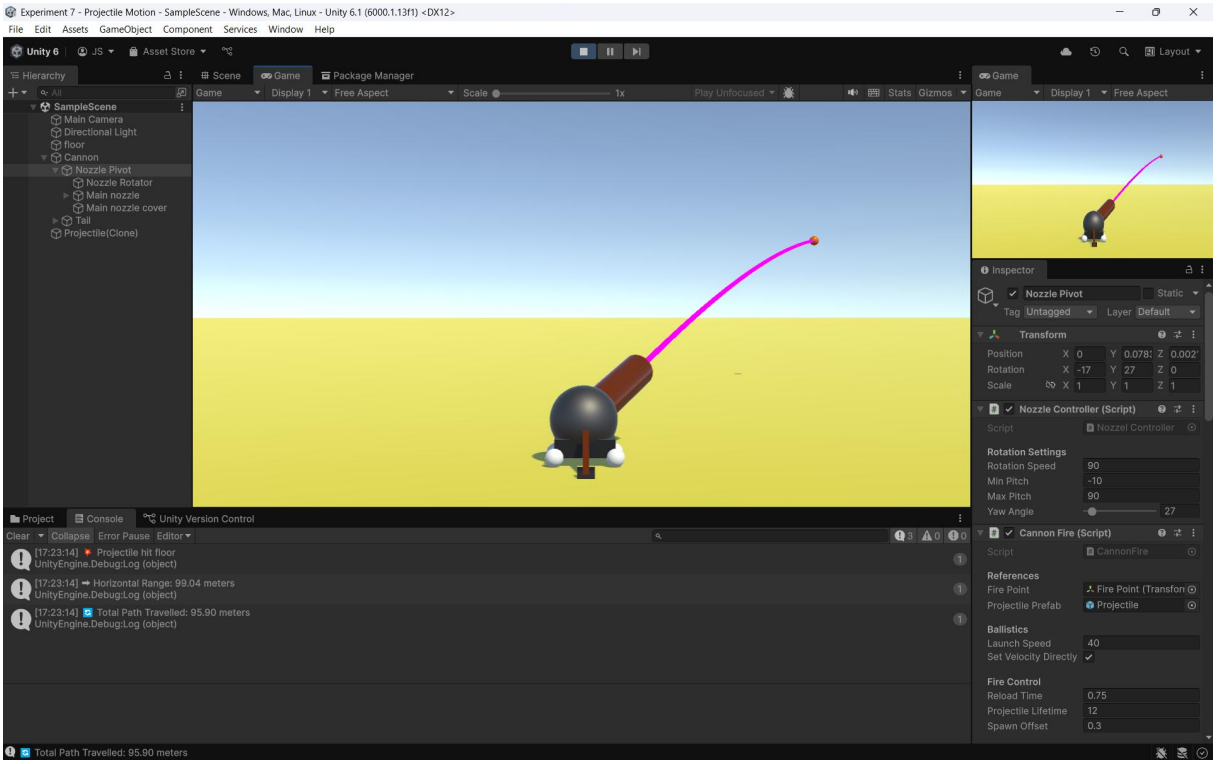
The projectile successfully followed a parabolic trajectory under gravity.

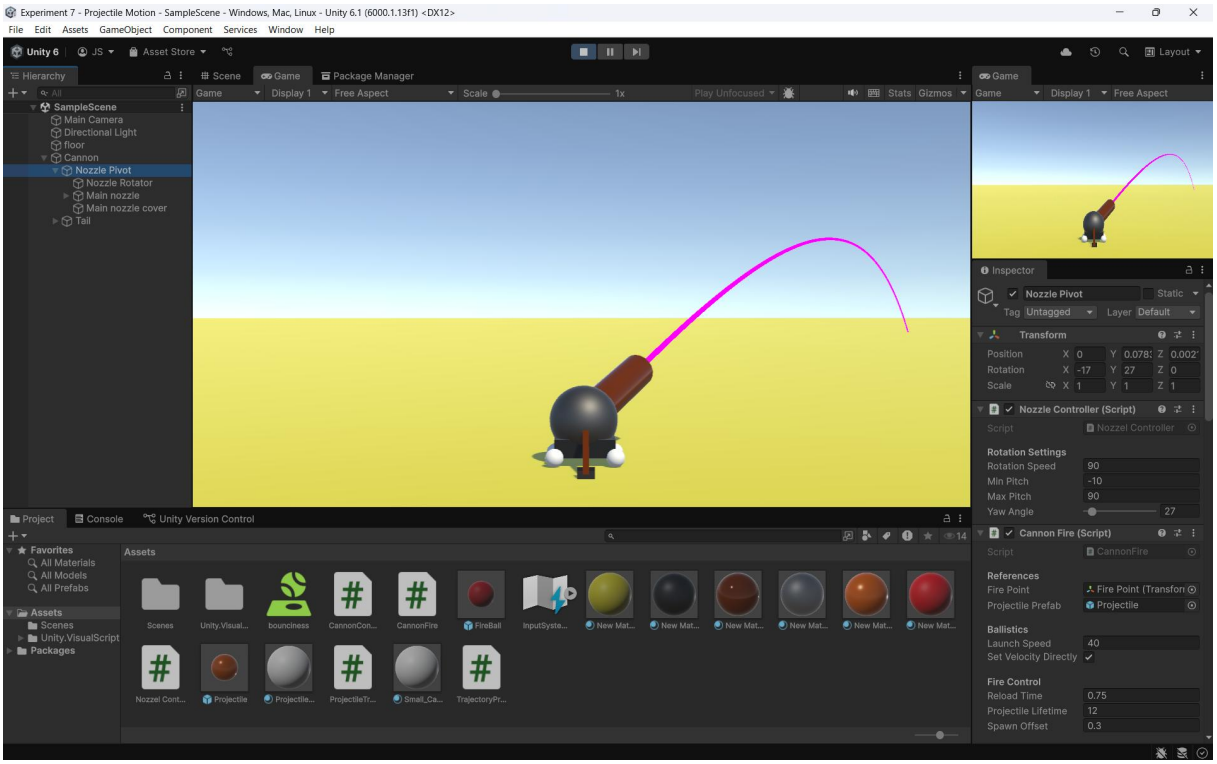
At 45°, the range was maximum, confirming classical physics.

Smaller angles gave flatter, shorter arcs; larger angles produced taller but shorter-ranged arcs.

Without applying initial velocity (only gravity), the projectile dropped straight down like a “failed fireball.”







Conclusion:

Unity's physics engine can be effectively used to simulate real-world projectile motion.

This experiment validates the equations of motion for projectiles and demonstrates their practical application in game development and educational physics simulations.

Result:

Script was created for projectile motion of object were successfully implemented and tested in Unity Game Engine 6.1 Their application in game mechanics was understood through coding and console outputs.