```
using System;
using UnityEngine;

/// <summary>
/// Made by Marissa Rowles-Stewart @ 2023
/// Controls a Drone entity's movement and death
/// </summary>

public class PhysicsDrone : MonoBehaviour
{
    [Header("Model")]
    [Tooltip("The model to use for displaying rotations")]
    [SerializeField]
    private Transform model;
    [SerializeField]
    [Tooltip("The maximum rotation angle the model can reach in the x and z axis")]
    private Vector2 maxRotations;

    [Header("Control")]
    [SerializeField]
    [Tooltip("How quickly throttle reaches full speed")]
    private float throttleIncrement = 0.1f;
    [SerializeField]
    [Tooltip("Base speed")]
    private float maxThrust = 200f;
    [SerializeField]
    [Tooltip("Multiplier to determine the speed of which yaw and pitch affect position.")]
    private Vector2 responsiveness = new Vector2(10f, 10f);

    // Forward movement speed
    private float throttle;

    // Input values
    private float horizontalInput;  // x / horizontal = Yaw
    private float verticalInput;    // y / vertical   = Pitch

    /// <summary>
    /// Makes it easier for the model rotation to reach its full value
    /// </summary>
    private Vector2 ScaledResponse => responsiveness / 10f;

    /// <summary>
    /// Modifies responsiveness to consider mass
    /// </summary>
    private Vector2 ResponseModifier => (rb.mass / 10f) * responsiveness;

    private Rigidbody rb;

    /// <summary>
    /// Where the drone will respawn when colliding
    /// </summary>
    private Vector3 startLocation;

    public event Action<PhysicsDrone> onDeath;
```

```
    void Start()
    {
        rb = GetComponent<Rigidbody>();

        startLocation = transform.position;
    }

    /// <summary>
    /// Update the inputs
    /// </summary>
    private void UpdateInputs()
    {
        horizontalInput = Input.GetAxis("Horizontal");
        verticalInput   = Input.GetAxis("Vertical");
    }

    /// <summary>
    /// Called via other classes, disables the drone until the game resumes
    /// </summary>
    /// <param name="newIsPaused">Is the game paused</param>
    public void SetPaused(bool newIsPaused)
    {
        this.enabled = !newIsPaused;
    }

    // Update is called once per frame
    void Update()
    {
        // Update/Increase throttle until it reaches max
        throttle = Mathf.Clamp(throttle + throttleIncrement, 0f, 100f);
        UpdateInputs();
        UpdateModelRotation();
    }

    private void UpdateModelRotation()
    {
        // - 0.5 allows the range of both x and y to fall into negatives
        // * 2 makes sure the result falls between -1 and 1

        var xRotation = (Mathf.InverseLerp(-(ScaledResponse.x), ScaledResponse.x, rb.velocity.x)
- 0.5f) * 2;
        xRotation *= maxRotations.x;

        var yRotation = (Mathf.InverseLerp(-(ScaledResponse.y), ScaledResponse.y, rb.velocity.y)
- 0.5f) * 2;
        yRotation *= maxRotations.y;

        model.localRotation = Quaternion.Euler(Vector3.forward * -xRotation + Vector3.right * -
yRotation);
    }

    private void FixedUpdate()
    {
        // Constant forward force
        rb.AddForce(transform.forward * (maxThrust * throttle));

        // Input determined force
        // Horizontal Movement
        rb.AddForce(transform.right * (horizontalInput * ResponseModifier.x));
        // Vertical Movement
        rb.AddForce(transform.up    * (verticalInput   * ResponseModifier.y));
    }
```

```
    private void OnCollisionEnter(Collision other)
    {
        KillPlayer();
    }

    public void KillPlayer()
    {
        onDeath?.Invoke(this);
        Respawn();
    }

    /// <summary>
    /// Returns the drone to the initial position when the game started & resets rigidbody speed
    /// </summary>
    private void Respawn()
    {
        transform.position = startLocation;
        rb.velocity = Vector3.zero;
        throttle = 0;
    }
}
```