```
using UnityEngine;
using System.Collections;
public class CatmullRomCurveInterpolation : MonoBehaviour
{
    const int NumberOfPoints = 8;
    Vector3[] controlPoints;
    const int MinX = -5;
    const int MinY = -5;
    const int MinZ = 0;
    const int MaxX = 5;
    const int MaxY = 5;
    const int MaxZ = 5;
    float u = 0;
    int segNum = 0;
    int PositiveMod(int a, int b)
    {
        return (a % b + b) % b;
    }
    /* Returns a point on a cubic Catmull-Rom/Blended Parabolas curve
    * u is a scalar value from 0 to 1
    * segment_number indicates which 4 points to use for interpolation
    */
    Vector3 ComputePointOnCatmullRomCurve(double u, int segmentNumber)
    {
        int index0 = PositiveMod(segmentNumber - 2, NumberOfPoints);
        int index1 = PositiveMod(segmentNumber - 1, NumberOfPoints);
        int index2 = PositiveMod(segmentNumber, NumberOfPoints);
        int index3 = PositiveMod(segmentNumber + 1, NumberOfPoints);
        float tau = .5f;
        Vector3 c3 = -tau * controlPoints[index0] + (2 - tau) *
          controlPoints
        [index1] + (tau - 2) * controlPoints[index2] + tau * controlPoints
        [index3];
        Vector3 c2 = 2 * tau * controlPoints[index0] + (tau - 3) *
        controlPoints[index1] + (3 - 2 * tau) * controlPoints[index2] + -
          tau
        * controlPoints[index3];
        Vector3 c1 = -tau * controlPoints[index0] + tau * controlPoints
        [segmentNumber];
        Vector3 c0 = controlPoints[index1];
        return Mathf.Pow((float)u, 3) * c3 + Mathf.Pow((float)u, 2) * c2 +
        (float)u * c1 + c0; ;
    }
    void GenerateControlPointGeometry()
    {
        for (int i = 0; i < NumberOfPoints; i++)
        {
            GameObject tempcube = GameObject.CreatePrimitive
            (PrimitiveType.Cube);
```

```
48              tempcube.transform.localScale -= new Vector3(0.8f, 0.8f,
                  0.8f);
49              tempcube.transform.position = controlPoints[i];
50          }
51      }
52
53      float EaseIn(float t)
54      {
55          return (Mathf.Sin(t * Mathf.PI - Mathf.PI / 2) + 1) / 2;
56      }
57
58      float EaseOut(float t)
59      {
60          return (Mathf.Sin(t * Mathf.PI - Mathf.PI / 2) + 1) / 2;
61      }
62
63
64      // Use this for initialization
65      void Start()
66      {
67          controlPoints = new Vector3[NumberOfPoints];
68          //Set points randomly
69          controlPoints[0] = new Vector3(0, 0, 0);
70          for (int i = 1; i < NumberOfPoints; i++)
71          {
72              controlPoints[i] = new Vector3(Random.Range(MinX, MaxX),
                    Random.Range
73              (MinY, MaxY), Random.Range(MinZ, MaxZ));
74          }
75          GenerateControlPointGeometry();
76      }
77      // Update is called once per frame
78      // Update is called once per frame
79      void Update()
80      {
81          u += Time.deltaTime;
82
83          // Check if u reaches the end of current segment
84          if (u >= 1.0f)
85          {
86              u -= 1.0f; // Reset u
87              segNum++; // Move to the next segment
88              if (segNum >= NumberOfPoints) // Ensure segNum stays within
                    bounds
89                  segNum = 0;
90          }
91
92          // Compute the position on the curve
93          Vector3 position = ComputePointOnCatmullRomCurve(u, segNum);
```

```
 94
 95          // Calculate the tangent vector at the current position
 96          Vector3 tangent = CalculateTangent(u, segNum);
 97
 98          // Rotate the object to face forward in the direction of motion
 99          if (tangent != Vector3.zero)
100          {
101              transform.rotation = Quaternion.LookRotation(tangent,
                    Vector3.up);
102          }
103
104          // Move the object to the computed position
105          transform.position = position;
106      }
107
108      // Function to calculate the tangent vector at a given position on the
         curve
109      Vector3 CalculateTangent(float u, int segmentNumber)
110      {
111          float delta = 0.01f; // Small step size for numerical
               differentiation
112          Vector3 tangentA = ComputePointOnCatmullRomCurve(u - delta,
               segmentNumber);
113          Vector3 tangentB = ComputePointOnCatmullRomCurve(u + delta,
               segmentNumber);
114          return (tangentB - tangentA).normalized;
115      }
116
117  }
118
```