# **Unity 6 Spline Follower System**

A comprehensive guide for attaching and constraining GameObjects to follow spline paths in Unity 6 using the built-in Splines package. This guide explains how to set up the system and provides a detailed breakdown of the SplineFollower.cs script.

### **Table of Contents**

1. [Prerequisites](https://www.google.com/search?q=%23prerequisites)
2. [Installation](https://www.google.com/search?q=%23installation)
3. [Quick Start](https://www.google.com/search?q=%23quick-start)
4. [Script Documentation](https://www.google.com/search?q=%23script-documentation)
   * [Inspector Properties](https://www.google.com/search?q=%23inspector-properties)
   * [Public Properties & Methods](https://www.google.com/search?q=%23public-properties--methods)
   * [Full Script: SplineFollower.cs](https://www.google.com/search?q=%23full-script-splinefollowercs)
   * [Code Breakdown](https://www.google.com/search?q=%23code-breakdown)

## **Prerequisites**

* Unity 6.0 or later.
* A basic understanding of the Unity Editor and scene management.
* Basic C# programming knowledge.

## **Installation**

### **Step 1: Install Unity Splines Package**

1. Open your Unity project.
2. Navigate to **Window > Package Manager**.
3. In the Package Manager window, click the dropdown menu at the top-left and switch the context from "In Project" to **"Unity Registry"**.
4. Use the search bar to find the **"Splines"** package.
5. Select the package and click the **"Install"** button.

### **Step 2: Verify Installation**

Once the package is installed, you can verify it by checking for the new spline-related options in the Unity Editor:

* **GameObject > 3D Object > Spline**
* **Component > Splines** menu

## **Quick Start**

This section will guide you through creating your first spline-following object in a few simple steps.

1. **Create a Spline:**
   * In the Hierarchy window, right-click and select **3D Object > Spline**. This will create a new GameObject with a SplineContainer component.
2. **Edit the Spline Path:**
   * Select the Spline object in your scene.
   * The Scene view will now display the Spline editing tools.
   * Click and drag the control points (knots) to shape your path. You can add new knots by clicking on the spline line or by using the tools in the Inspector.
3. **Create an Object to Follow the Spline:**
   * In the Hierarchy window, right-click and select **3D Object > Cube** to create a simple object.
4. **Add and Configure the SplineFollower Script:**
   * Create a new C# script in your Assets folder and name it SplineFollower.
   * Copy the complete SplineFollower.cs code provided below into this script.
   * Attach the SplineFollower script to your **Cube** GameObject.
   * With the Cube selected, look at the Inspector. Drag the **Spline** GameObject from your Hierarchy into the **Spline Container** field on the SplineFollower component.
5. **Play and Test:**
   * Press the **Play** button in the Unity Editor. You should now see your cube moving along the spline path you created.
   * Experiment with the **Speed**, **Loop**, and **Look Forward** settings in the Inspector to see how they affect the behavior.

## **Script Documentation**

### **Inspector Properties**

These are the public fields you can configure directly from the Unity Inspector when the SplineFollower script is attached to a GameObject.

| **Property** | **Description** | **Default Value** |
| --- | --- | --- |
| **Spline Container** | The SplineContainer object that defines the path for the GameObject to follow. You must assign a spline to this field. | None |
| **Speed** | The movement speed of the object along the spline, measured in world units per second. | 5.0 |
| **Loop** | If true, the object will loop back to the start of the spline after reaching the end. If false, it will stop at the end. | true |
| **Look Forward** | If true, the object will automatically rotate to face the direction of movement along the spline's tangent. | true |
| **Show Debug Info** | If true, prints the current progress and position to the console. Useful for debugging purposes. | false |

### **Public Properties & Methods**

These can be accessed from other scripts to control the spline follower's behavior at runtime.

#### **Properties**

* public float Progress { get; }
  + **Description:** Returns the follower's current progress along the spline, represented as a normalized value from 0.0 (start) to 1.0 (end). This is a read-only property.
* public float SplineLength { get; }
  + **Description:** Returns the total length of the spline in world units. This is a read-only property.

#### **Methods**

* public void SetProgress(float progress)
  + **Description:** Manually sets the follower's position on the spline. The input value is clamped between 0 and 1.
  + **Example:** mySplineFollower.SetProgress(0.5f); // Moves the object to the halfway point.
* public void ResetToStart()
  + **Description:** Instantly moves the follower back to the beginning of the spline (progress = 0).
  + **Example:** mySplineFollower.ResetToStart();
* public void SetPaused(bool paused)
  + **Description:** Pauses or resumes the follower's movement. When paused, the Update loop is disabled.
  + **Example:** mySplineFollower.SetPaused(true); // Pauses movement.

**Code Breakdown**

The Update() method is the core of the script. Here’s a step-by-step explanation of its logic:

1. **Safety Check:** It first checks if splineContainer has been assigned. If not, it stops execution for this frame to prevent null reference errors.
2. **Calculate Progress Delta:** The script calculates how much to move in the current frame.
   * speed \* Time.deltaTime gives the distance to travel in world units.
   * This distance is then divided by the spline's total length (splineLength) to get a normalized progress value (movementDelta). This ensures the speed is consistent regardless of the spline's length.
3. **Update Progress:** The movementDelta is added to currentProgress.
4. **Handle Loop/Clamp:**
   * If loop is true, the modulo operator (% 1f) wraps the progress value. For example, 1.01f % 1f becomes 0.01f, effectively restarting the path.
   * If loop is false, Mathf.Clamp01 ensures the progress value stops at 1.0.
5. **Evaluate Spline:** The splineContainer.Evaluate() method is the key function from the Splines package. Given a normalized progress value, it calculates and returns:
   * position: The 3D point in world space.
   * tangent: The forward direction vector at that point.
   * up: The up vector, which helps define the correct orientation.
6. **Apply Transform:**
   * The GameObject's transform.position is set to the calculated position.
   * If lookForward is true, Quaternion.LookRotation is used to create a rotation where the object's Z-axis (forward) aligns with the spline's tangent and its Y-axis (up) aligns with the spline's up vector.

*Created for educational purposes. Feel free to modify and extend for your projects!*