# AutoMover Pro

# Table of Contents

*Table of Contents.....I*

*Introduction.....II*

*Script Reference.....1*

*AutoMover.....1*

*AutoMover.cs.....1*

# Introduction

AutoMover Pro is a script Component that makes movement easy, fast and efficient. With this tool, you can make your models move along curves with hardly any learning curve.

To use AutoMover Pro, follow these steps:

1. Import the AutoMover Asset. You can play the Demo Scene found in the ’Demo’ folder to see examples of the main features of AutoMover Pro. The demo scene is compatible with the built-in render pipeline.

2. Add the AutoMover as a component to an object, either in the editor or using C#.

3. Add anchor points and modify the other parameters as needed.

4. This document includes a script reference for all the necessary public methods and properties (C#) for AutoMover. You can use the script reference and the file ’Demo/AutoMoverScriptDemo.cs’ to get started.

The Pro version includes the following features over the free version:

- New anchor point space option: Child space
- New noise type: Smooth random
- New curve type: Spline through points
- Scale included in anchor points
- Option to stop the movement every X seconds for Y seconds
- Option to always face forward
- Option to slow down on tight curves, including adjustable turn rate and deceleration time
- Pause/Play-button on inspector for runtime
- Button to export the curve as GameObjects, with the option to include the mesh itself
- Multiple small improvements over the free version, such as more general undo support, movable anchor points in editor window, multi-object editing, spherical interpolation for rotations, customizable step count and anchor point weighting

# Script Reference

## AutoMover

### AutoMover.cs

#### Enumerations

##### *AutoMoverCurve*

```
public enum AutoMoverCurve{
     Linear,
     Curve,
     Spline,
     SplineThroughPoints,}
```

Curve option enumerations

Linear: Straight lines between the anchor points.

Curve: Form a single bezier curve from the anchor points.

Spline: Form as many curves as there are anchor points - 1. The curves are connected with C2 continuity.

SplineThroughPoints: Form as many curves as there are anchor points - 1. The curves are connected with C2 continuity. Stretches the spline to go through (or very close to) the anchor points.

##### *AutoMoverLoopingStyle*

```
public enum AutoMoverLoopingStyle{
     loop,
     repeat,
     bounce}
```

Looping style enumerations

loop: Form a closed loop from the anchor points.

repeat: Simply start the path again after finishing.

bounce: Travel the original path back to the start after reaching the end.

##### *AutoMoverRotationMethod*

```
public enum AutoMoverRotationMethod{
     spherical,
     linear}
```

Rotation method enumarations

spherical: For example when going from 300 degrees to 10 degrees, the rotation will actually go to 370 degrees.

linear: Rotation will always approach exactly the given value, even with multiple 360 degree spins.

## AutoMoverAnchorPointSpace

```
public enum AutoMoverAnchorPointSpace{
     world,
     local,
     child}
```

Anchor point space enumerations

world: The points are defined in World space. Moving the parent of the object will not move the anchor points.

local: The points are defined in the object's local space. Moving the parent of the object also moves the anchor points.

child: The points are defined in the object's 'child' space. Moving the object also moves the anchor points.

## AutoMoverNoiseType

```
public enum AutoMoverNoiseType{
     random,
     sine,
     smoothRandom}
```

Noise type enumerations

random: Generates random noise. Uses the specified noise amplitude and frequency.

sine: Sine noise. Uses specified amplitude, frequency and offset

smoothRandom: Generates smooth random noise. Uses the specified noise amplitude and frequency.

## AutoMoverTarget

```
public enum AutoMoverTarget{
     position,
     rotation,
     scale}
```

Target enumerations. Used when generating noise.

# Classes

## AnchorPoint

```
public struct AnchorPoint
```

Holds the necessary information for an anchor point (position and rotation).

## Variables

### *position*

```
public Vector3 position
```

Position of the anchor point.

### *rotation*

```
public Vector3 rotation
```

Rotation of the anchor point as an euler angle.

### *scale*

```
public Vector3 scale
```

Scale of the anchor point as an euler angle.

# *AutoMover*

```
public class AutoMover
```

## Properties

### *Pos*

```
public List<Vector3> Pos
     { get;
     }
```

List of the anchor point positions.

### *Rot*

```
public List<Vector3> Rot
     { get;
     }
```

List of the anchor point rotations.

### *Scl*

```
public List<Vector3> Scl
     { get;
     }
```

List of the anchor point scales.

### Length

```
public float Length
     { get;
     set; }
```

The length of the curve in seconds. Minimum value of 0.5f.

### CurveStyle

```
public AutoMoverCurve CurveStyle
     { get;
     set; }
```

The type of the curve.

### LoopingStyle

```
public AutoMoverLoopingStyle LoopingStyle
     { get;
     set; }
```

The looping style.

### RotationMethod

```
public AutoMoverRotationMethod RotationMethod
     { get;
     set; }
```

The way rotations are done.

### AnchorPointSpace

```
public AutoMoverAnchorPointSpace AnchorPointSpace
     { get;
     set; }
```

The space in which anchor points are defined. Converts existing anchor points when changed.

### Moving

```
public bool Moving
     { get;
     }
```

True if the mover is moving the object.

### PositionNoiseType

```
public AutoMoverNoiseType PositionNoiseType
     { get;
     set; }
```

The type of the position noise.

### RotationNoiseType

```
public AutoMoverNoiseType RotationNoiseType
     { get;
     set; }
```

The type of the rotation noise.

### ScaleNoiseType

```
public AutoMoverNoiseType ScaleNoiseType
     { get;
     set; }
```

The type of the scale noise.

### PositionNoiseAmplitude

```
public Vector3 PositionNoiseAmplitude
     { get;
     set; }
```

The amplitude of position noise in each direction.

### PositionNoiseFrequency

```
public Vector3 PositionNoiseFrequency
     { get;
     set; }
```

The frequency of position noise in each direction.

### RotationNoiseAmplitude

```
public Vector3 RotationNoiseAmplitude
     { get;
     set; }
```

The amplitude of rotation noise in each direction. (Degrees)

### ScaleNoiseAmplitude

```
public Vector3 ScaleNoiseAmplitude
     { get;
     set; }
```

The amplitude of scale noise in each direction.

### RotationNoiseFrequency

```
public Vector3 RotationNoiseFrequency
     { get;
     set; }
```

The frequency of rotation noise in each direction.

For random noise, this means the amount of half rotations (180 degrees) in a second.

For sine noise, this means the frequency of the sine wave. With value of 1, the wave takes the time of 2*pi seconds.

### ScaleNoiseFrequency

```
public Vector3 ScaleNoiseFrequency
     { get;
     set; }
```

The frequency of scale noise in each direction.

### RotationSineOffset

```
public Vector3 RotationSineOffset
     { get;
     set; }
```

The phase offset of rotation sine noise in each direction. Values equal with the remainder when divided by 2*pi.

### ScaleSineOffset

```
public Vector3 ScaleSineOffset
     { get;
     set; }
```

The phase offset of scale sine noise in each direction. Values equal with the remainder when divided by 2*pi.

### PositionSineOffset

```
public Vector3 PositionSineOffset
     { get;
     set; }
```

The phase offset of position sine noise in each direction. Values equal with the remainder when divided by 2*pi.

### RunOnStart

```
public bool RunOnStart
     { get;
     set; }
```

The movement is started automatically during Start. If set to false, the movement will have to be manually started.

### DelayStartMin

```
public float DelayStartMin
     { get;
     set; }
```

Minimum delay on start in seconds. The actual delay is a random number between DelayStartMin and DelayStartMax.

### DelayStartMax

```
public float DelayStartMax
     { get;
     set; }
```

Maximum delay on start in seconds. The actual delay is a random number between DelayStartMin and DelayStartMax.

### DelayMin

```
public float DelayMin
     { get;
     set; }
```

Minimum delay between loops in seconds. The actual delay is a random number between DelayMin and DelayMax.

### DelayMax

```
public float DelayMax
     { get;
     set; }
```

Maximum delay between loops in seconds. The actual delay is a random number between DelayMin and DelayMax.

### StopAfter

```
public uint StopAfter
     { get;
     set; }
```

How many times the object moves the path. 0 Means that the movement will run until stopped.

### StopEveryXSeconds

```
public float StopEveryXSeconds
     { get;
     set; }
```

How often (seconds) should the movement stop.

The time the movement is stopped for is defined by StopForXSeconds.

### StopForXSeconds

```
public float StopForXSeconds
     { get;
     set; }
```

For how long should the movement stop. Value of 0 means no stopping.

The interval for how often the movement is stopped is defined by StopEveryXSeconds.

### SlowOnCurves

```
public bool SlowOnCurves
     { get;
     set; }
```

Should the movement be slower for sharp curves and faster during straight paths.

Turn rate and deceleration can be adjusted.

### FaceForward

```
public bool FaceForward
     { get;
     set; }
```

Ignore rotations in anchor points after the first one and always face forward.

Default ’forward’ is along the X-axis, so align the object in the first anchor point along that.

### DynamicUpVector

```
public bool DynamicUpVector
     { get;
     set; }
```

Allow the up-vector to be dynamically recalculated while facing forward to allow for the object to turn smoothly along the curve.

### DrawGizmos

```
public bool DrawGizmos
     { get;
     set; }
```

Should the path be visualized in the editor.

### InstantRuntimeChanges

```
public bool InstantRuntimeChanges
     { get;
     set; }
```

Apply changes made during runtime instantly and restart the movement.

If false, changes are applied after the current loop.

### PopulateWithMesh

```
public bool PopulateWithMesh
     { get;
     set; }
```

Populate the exported curve's gameobjects with the MeshRenderer or SpriteRenderer attached to the object.

### MovableGizmos

```
public List<bool> MovableGizmos
     { get;
     }
```

Should the gizmos in editor have position control.

### IsPaused

```
public bool IsPaused
     { get;
     }
```

Tells if the movement is paused. False if the object is not moving, or if it is moving and is not paused.

Control with Pause() and Resume()

### CurveWeight

```
public float CurveWeight
     { get;
     set; }
```

The control point weighting used for the creation of curves and splines.

Values from 0 to 1 are allowed

### TurnRate

```
public float TurnRate
     { get;
     set; }
```

The turn rate when SlowOnCurves is enabled.

Values from 1 to 360 are allowed.

### DecelerationTime

```
public float DecelerationTime
     { get;
     set; }
```

The time in seconds that is spent decelerating for a sharp turn when SlowOnCurves is enabled.

Values must be non-negative.

### Steps

```
public int Steps
     { get;
     set; }
```

The number of segments in the curve.

Value cannot be lower than the number of anchor points.

Real step count may differ slightly in reality.

## Methods

### AddAnchorPoint

```
public void AddAnchorPoint()
```

Adds the current position and rotation of the object as an anchor point.

### AddAnchorPoint

```
public void AddAnchorPoint(
     Vector3 position,
     Vector3 rotation,
     Vector3 scale)
```

Adds the given position, rotation and scale as a anchor point at the end of the anchor point list. If the object is already moving, the new anchor point will be taken into account during the next lap.

position: Position of the anchor point.
rotation: Rotation of the anchor point as an euler angle.
scale: Scale of the anchor point.

### AddAnchorPoint

```
public void AddAnchorPoint(
     Vector3 position,
     Vector3 rotation)
```

Adds the given position and rotation as a anchor point at the end of the anchor point list. Scale will be defaulted to the transform's current scale. If the object is already moving, the new anchor point will be taken into account during the next lap.

position: Position of the anchor point.
rotation: Rotation of the anchor point as an euler angle.

### AddAnchorPoint

```
public void AddAnchorPoint(
     AnchorPoint anchorPoint)
```

Adds the given AnchorPoint at the end of the anchor point list. If the object is already moving, the new anchor point will be taken into account during the next lap.

anchorPoint: Anchor point to be added.

### GetAnchorPoint

```
public AnchorPoint GetAnchorPoint(
     int index)
```

Returns the AnchorPoint (position, rotation and scale) at the given index.

index: Which element should be returned.

Returns: The anchor point at the given index.

### GetAnchorPointPosition

```
public Vector3 GetAnchorPointPosition(
     int index)
```

Returns the position of the anchor point at the given index.

index: Which element should be returned.

Returns: The position of the anchor point at the given index.

### GetAnchorPointRotation

```
public Vector3 GetAnchorPointRotation(
     int index)
```

Returns the rotation of the anchor point at the given index.

index: Which element should be returned.

Returns: The rotation of the anchor point at the given index.

### GetAnchorPointScale

```
public Vector3 GetAnchorPointScale(
     int index)
```

Returns the scale of the anchor point at the given index.

index: Which element should be returned.

Returns: The scale of the anchor point at the given index.

### SetMovableGizmo

```
public bool SetMovableGizmo(
     int index,
     bool value)
```

Set a gizmo to be movable in the editor or not.

index: Which element should be changed.
value: New value for the boolean.

Returns: The boolean value if an anchor point's gizmo is movable or not at the given index.

### GetMovableGizmo

```
public bool GetMovableGizmo(
     int index)
```

See if a gizmo is movable in the editor or not.

index: Which element should be returned.

Returns: The boolean value if an anchor point's gizmo is movable or not at the given index.

### SetAnchorPoint

```
public void SetAnchorPoint(
     int index,
     AnchorPoint anchorPoint)
```

Sets the anchor point (position, rotation and scale) at the given index.

index: Which element should be set.
anchorPoint: The anchor point.

### SetAnchorPoint

```
public void SetAnchorPoint(
     int index,
     Vector3 position,
     Vector3 rotation,
     Vector3 scale)
```

Sets the anchor point (position, rotation and scale) at the given index.

index: Which element should be set.
position: The position of the anchor point.
rotation: The rotation of the anchor point.
scale: The scale of the anchor point.

### SetAnchorPoint

```
public void SetAnchorPoint(
     int index,
     Vector3 position,
     Vector3 rotation)
```

Sets the anchor point (position and rotation. scale is assumed) at the given index.

index: Which element should be set.
position: The position of the anchor point.
rotation: The rotation of the anchor point.

### SetAnchorPointPosition

```
public void SetAnchorPointPosition(
     int index,
     Vector3 position)
```

Sets the position of the anchor point at the given index.

index: Which element should be set.
position: The position of the anchor point.

### SetAnchorPointRotation

```
public void SetAnchorPointRotation(
     int index,
     Vector3 rotation)
```

Sets the rotation of the anchor point at the given index.

index: Which element should be set.
rotation: The rotation of the anchor point.

### SetAnchorPointScale

```
public void SetAnchorPointScale(
     int index,
     Vector3 scale)
```

Sets the scale of the anchor point at the given index.

index: Which element should be set.
rotation: The rotation of the anchor point.

### InsertAnchorPoint

```
public void InsertAnchorPoint(
     int index,
     AnchorPoint anchorPoint)
```

Inserts the given anchor point at the given index.

index: The index where the anchor point should be inserted.
anchorPoint: The anchor point to be inserted.

### InsertAnchorPoint

```
public void InsertAnchorPoint(
     int index,
     Vector3 position,
     Vector3 rotation,
     Vector3 scale)
```

Inserts the given position, rotation and scale as an anchor point at the given index.

index: The index where the anchor point should be inserted.
position: The position of the anchor point to be inserted.
rotation: The rotation of the anchor point to be inserted.
scale: The scale of the anchor point to be inserted.

### InsertAnchorPoint

```
public void InsertAnchorPoint(
     int index,
     Vector3 position,
     Vector3 rotation)
```

Inserts the given position and rotation as an anchor point at the given index. Scale is assumed

index: The index where the anchor point should be inserted.
position: The position of the anchor point to be inserted.
rotation: The rotation of the anchor point to be inserted.

### DuplicateAnchorPoint

```
public void DuplicateAnchorPoint(
     int index)
```

Duplicates the anchor point (position, rotation and scale) at given index.

index: Which element should be duplicated. Does nothing if it is out of bounds.

### RemoveAnchorPoint

```
public void RemoveAnchorPoint(
     int index)
```

Removes the anchor point (position, rotation and scale) at given index.

index: Which element should be removed. Does nothing if it is out of bounds.

### MoveAnchorPointUp

```
public void MoveAnchorPointUp(
     int index)
```

Moves the anchor point at the given index up in the list (decreasing its index by one).

index: Which element should be moved. Does nothing if it is out of bounds or 0.

### MoveAnchorPointDown

```
public void MoveAnchorPointDown(
     int index)
```

Moves the anchor point at the given index down in the list (increasing its index by one).

index: Which element should be moved. Does nothing if it is out of bounds or the last element in the list.

### MoveAnchorPoint

```
public void MoveAnchorPoint(
     int from,
     int to)
```

Moves the anchor point from index ’from’ to index ’to’.

from: Index of the anchor point that will be moved.
to: Index where the anchor point will be moved.

### GetProgress

```
public float GetProgress()
```

Get the progress of the current loop. 0 if not moving.

Returns: progress as a float from 0 to 1.

### Pause

```
public void Pause()
```

Pauses the movement. Does nothing if the object is not moving.

### Resume

```
public void Resume()
```

Resumes the object from a pause. Does nothing if IsPaused is false.

### StartMoving

```
public void StartMoving()
```

Starts moving the object along the specified curve.

### StopMoving

```
public void StopMoving()
```

Stops moving the object. Starting the movement again will begin from the starting point of the curve.

### ExportCurve

```
public GameObject ExportCurve()
```

Creates as many gameobjects in the current scene as there are steps in the path.

The gameobjects’ transforms contain the position, rotation and scale information of that step.

Returns: The parent gameobject that contains the path.

## *AnchorPointVersionCheck*

```
public bool AnchorPointVersionCheck()
```

Makes sure that the position, rotation etc. lists are coherent and there are no missing or extra values.

This is executed at the start of the movement.