AutoMover Asset

Table of Contents

Introduction

AutoMover is a Unity component that can be used to easily create movement for objects. The main use case for this component is to move objects that don’t require heavy interaction with other objects, such as flying stars in the background or orbs rotating around the tip of a staff. AutoMover allows you to easily create and edit (using scene editor or C#) paths that the object will move along, and changing the options of the movement.

To use AutoMover, 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.

2. Add the AutoMover as a component to an object, either in the editor or using C#. More about using C# on step 4.

3. Add anchor points (using the component editor or C#) and modify the other options 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/ScriptDefinedAutoMover.cs’ to get started.

Script Reference

AutoMover

AutoMover.cs

Enumerations

AutoMoverCurve

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

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 waypoints - 1. The curves are connected with C2 continuity.

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{
     shortestPath,
     absoluteValue}
```

Rotation method enumarations

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

absoluteValue: Rotation will always approach exactly the given value, even with multiple rotations.

AutoMoverAnchorPointSpace

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

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.

AutoMoverNoiseType

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

Noise type enumerations

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

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

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.

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.

Length

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

Specifies the length of the curve in seconds. Minimum value of 0.001f.

CurveStyle

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

Specifies the type of the curve.

LoopingStyle

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

Specifies the looping style.

RotationMethod

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

Specifies the way rotations are done.

AnchorPointSpace

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

Specifies the space of the anchor points.

Moving

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

True if the mover is moving the object.

PositionNoiseType

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

Specifies the type of the position noise.

RotationNoiseType

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

Specifies the type of the rotation noise.

PositionNoiseAmplitude

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

Specifies the amplitude of position noise in each direction.

PositionNoiseFrequency

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

Specifies the frequency of position noise in each direction.

RotationNoiseAmplitude

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

Specifies the amplitude of rotation noise in each direction. (Degrees)

RotationNoiseFrequency

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

Specifies the frequency of rotation noise in each direction.

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

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

RotationSineOffset

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

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

PositionSineOffset

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

Specifies 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; }
```

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

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; }
```

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

PrecomputePath

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

Specifies if the path should be precomputed once at the start of the movement, rather than computing it for every loop.

If set to true, changing any parameter will trigger a new precalculation, so it is recommended to be set to false if the parameters are modified often.

DrawGizmos

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

Specifies if the path should be visualized in the editor.

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()

Methods

Start

```
void Start()
```

Update

```
void Update()
```

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)
```

Adds the given position and rotation 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.

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 and rotation) at the given index.

index: Specifies 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: Specifies 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: Specifies which element should be returned.

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

SetAnchorPoint

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

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

index: Specifies which element should be set.
anchorPoint: The anchor point.

SetAnchorPoint

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

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

index: Specifies 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: Specifies 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: Specifies 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)
```

Inserts the given position and rotation 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.

DuplicateAnchorPoint

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

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

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

RemoveAnchorPoint

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

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

index: Specifies 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: Specifies 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: Specifies 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.

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.

OnDrawGizmosSelected

```
void OnDrawGizmosSelected()
```