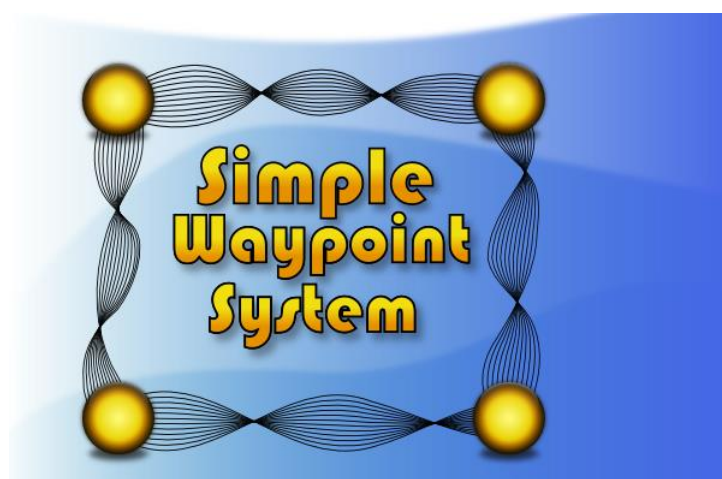


Simple Waypoint System (SWS)

by Rebound Games

v3.0.3



1. SWS – WHAT’S THAT?	2
2. SCRIPT CONNECTIONS AND PREPARATION	3
3. EXAMPLE SCENES	4
4. SETUP AND CREATE A PATH	8
5. EDIT A PATH	10
6. FOLLOW A PATH	12
7. METHODS, RUNTIME ACCESS	15
8. CONTACT	18
9. VERSION HISTORY.....	19

Thanks for buying SWS!

Your support is greatly appreciated!

1. SWS – WHAT'S THAT?

Simple Waypoint System (SWS) is an editor extension for Unity3d which allows you to create waypoints and paths very easily right within the editor. With those created, you can then tell any kind of game object to follow a specific path.

This kit is useful for every automated movement in your games.

For example for:

- AI Patrol behavior
- Movement on a Path
- Moving Platforms
- Camera and Game Object animation
- Cut Scenes
- 3D GUI animation

But I'm sure you will also find some other needs for this system.

Included features are:

- Easy to use
- Custom Path Manager
- Movement using iTween & HOTween
- Message Settings per waypoint
- Works in 2D & 3D space
- Compatible with C# & JavaScript
- Undo and Redo
- Full source code in C#
- Every line documented
- 3 Example Scenes

As you have seen in the features, SWS uses iTween and HOTween to simulate movement in 3d space. Both of them are free and open source animation libraries for Unity3d. More information and the latest versions are available from the Google Code repository link on their main sites here:

<http://itween.pixelplacement.com/index.php>

<http://www.holoville.com/hotween/>

Please feel free to donate to one or both of these projects.

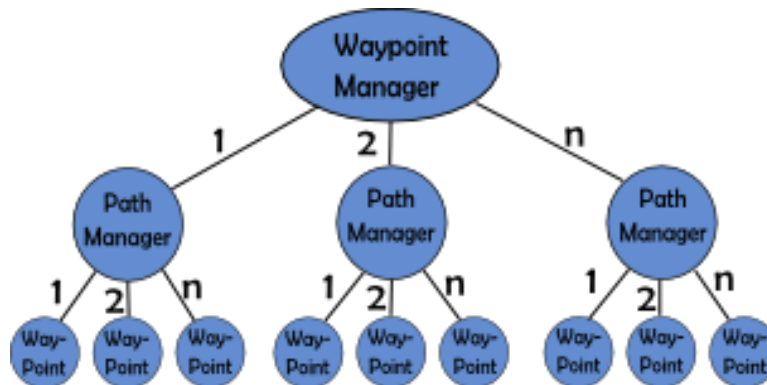
Your donation will support the development of iTween and HOTween.

2. SCRIPT CONNECTIONS AND PREPARATION

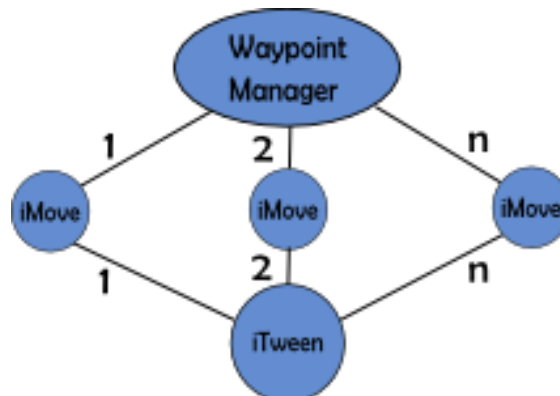
If you are new to Unity3d, please take a quick break and get dirty with its main functionalities, because this documentation will assume you have some basic knowledge regarding the interface and game mechanics.

Setting up SWS is very simple. But to understand internal connections between scripts without looking at code, the drawings below should give a short overview.

In the editor, the Waypoint Manager lets you create as many waypoints as you want. They get connected on the fly and are stored as separate game objects. Two and more waypoints form a path, so each path can consist of countless waypoints. This path is also a separate game object and holds a Path Manager component, which stores the array of created waypoints for this path. Waypoint Editor and Path Editor give you a custom inspector for both of them.




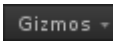
In order to actually move your game objects with the provided movement scripts and make use of their animation library, a copy of iTween and HOTween in your project is also required. These libraries need not be attached to any game object in the scene.




3. EXAMPLE SCENES

Once you downloaded SWS from the Unity Asset Store and imported it in your project, it is recommended that you get familiar with its mechanics and take a look at the provided example scenes. Please open

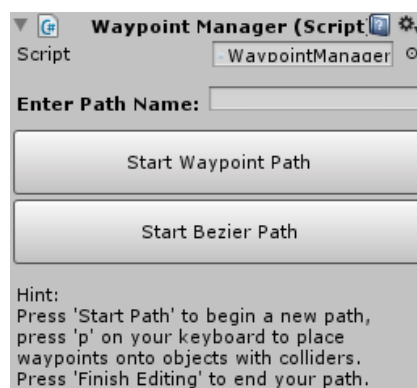
 Example_Straight

You will notice some yellow and blue paths, starting and ending with a purple box and a few yellow or blue spheres between them. If they do not show up, please make sure that the Path Manager component is enabled in the gizmo settings. 

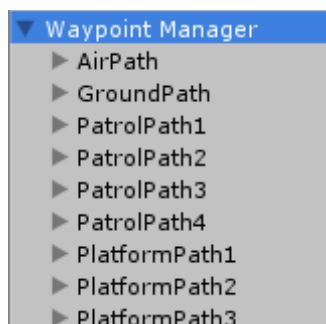
A red capsule, platform or soldier is positioned besides every starting point which will “walk” on this path. If you click on the Play button, you can see them in action. 

So, how this SWS stuff works?

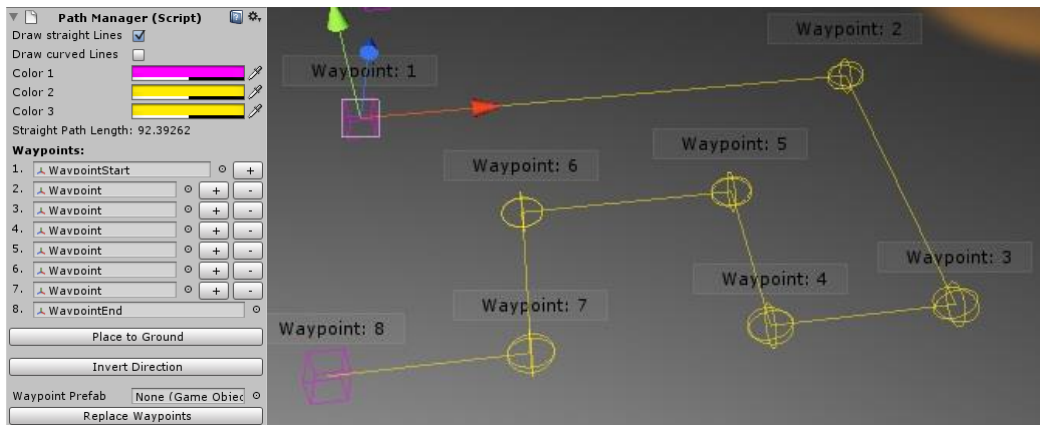
This scene consists of four game objects; the game objects “Walker” and “Waypoint Manager” are relevant for us. After selecting the Waypoint Manager game object, you will see this inspector window:



By clicking on the buttons you are able to create paths. But since creating a path is mentioned in the next chapter and this scene already has some paths, we do not create a new one and take a look at the existing ones instead. Expand the Waypoint Manager game object in the hierarchy and you will see all existing paths in this scene:

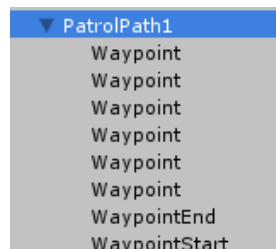


If you click on one of these paths, our Path Manager component shows up and we see some detailed information about the path we clicked. For example, this is what we get by inspecting “PatrolPath1”:

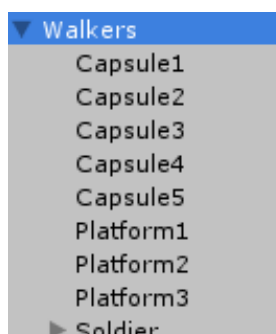


In the left image, our Path Manager component, we see two checkboxes and three gizmo color fields, the total length of this path and a transform slot for each waypoint involved. Due to this example should only show the use of straight paths, all paths have “Draw straight Lines” ticked. At the end we have the option to place all waypoints down to the ground or invert the existing path direction. Read more about its functionalities in chapter 5 – Editing a Path. In the right picture, a snapshot of our scene view, we see that every waypoint gets a small info box with an index above them.

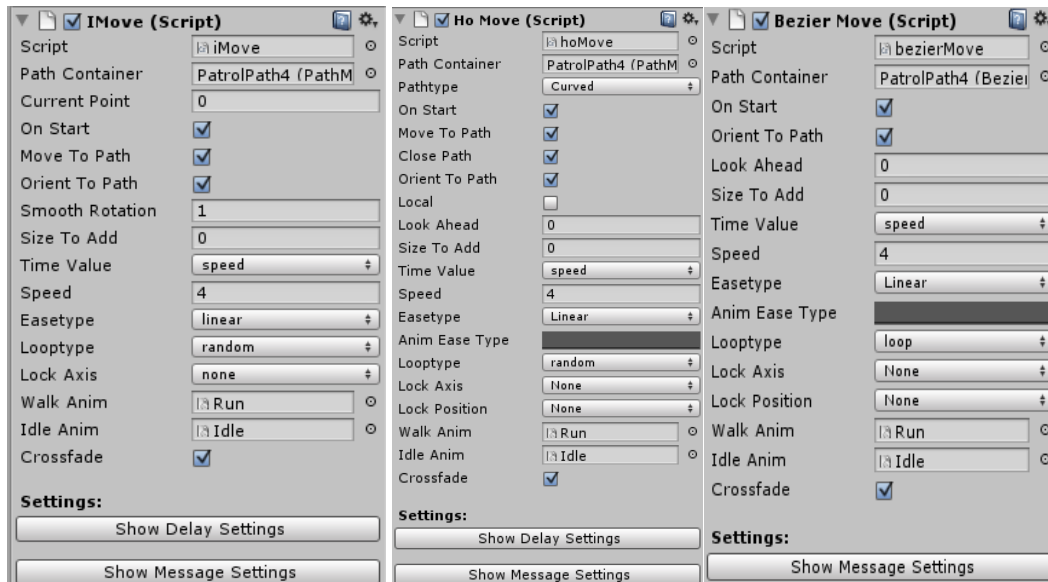
By expanding our selected path within the hierarchy, we are able to see every single waypoint game object of this path:



At this point, you should get the basic and simple structure of paths in SWS, so we continue with our walker objects. A “walker” is defined as a gameobject, which follows a path. Expand the “Walkers” gameobject to see what type of walkers we currently have in our scene.




In this example, each walker has an iMove component attached to it. As mentioned in chapter 2, this script is responsible for movement - more precisely, straight movement with iTween. HOTween supports both straight and curved movement. For this concern, the hoMove or bezierMove component is used, what we look at next.

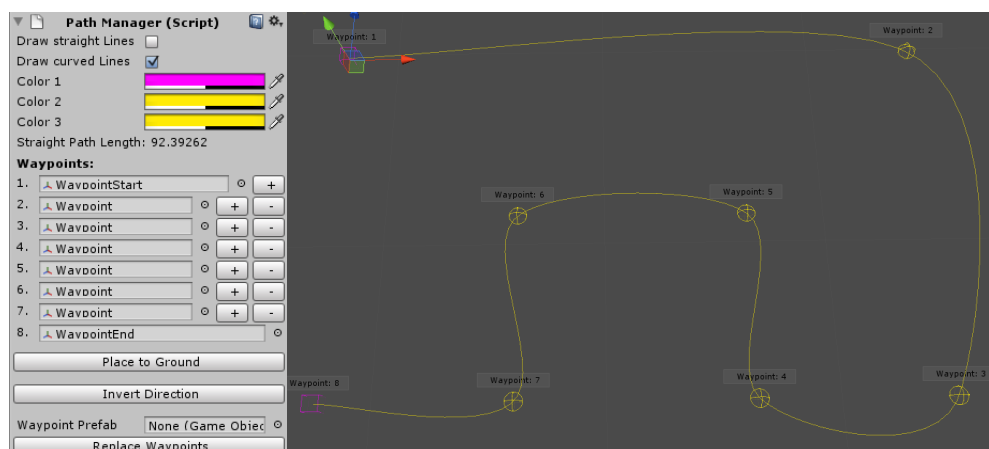



iMove, hoMove and bezierMove settings are described deeper in chapter 6 – Follow a Path.

Please open

 Example_Curved

The hierarchy of this scene looks very similar to the last, but now each path draws curved lines between its waypoints and every walker game object has the hoMove script attached to it.



Press Play to see the result. 

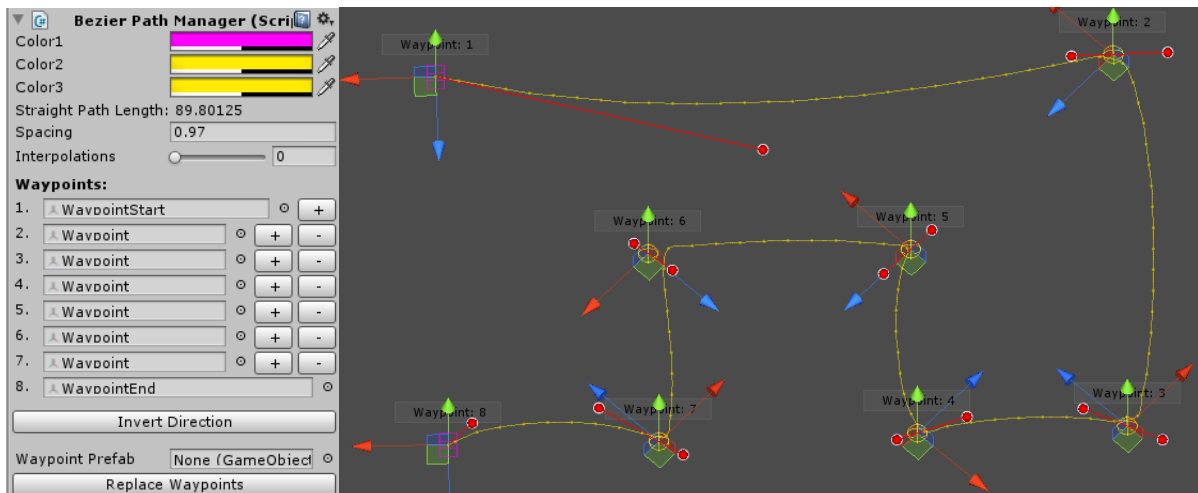
Lastly, please open



This scene utilizes curved paths too, but in a very different way than Example_Curved and hoMove does. In this context, the BezierPathManager component supports the creation of paths using bezier points. Bezier points shape the path differently based on their position. Every waypoint has at least one bezier point. In the scene view, you can reposition bezier points by using their small, red gizmo handles.

You'll find more details about bezier paths in chapter 4. Movement on a bezier path needs its separate movement script, called bezierMove. In the end, it all comes down to:

- straight or curved paths: PathManager -> iMove or hoMove
- bezier paths: BezierPathManager -> bezierMove



In the following chapters we will set up our project for creating and editing a new path.

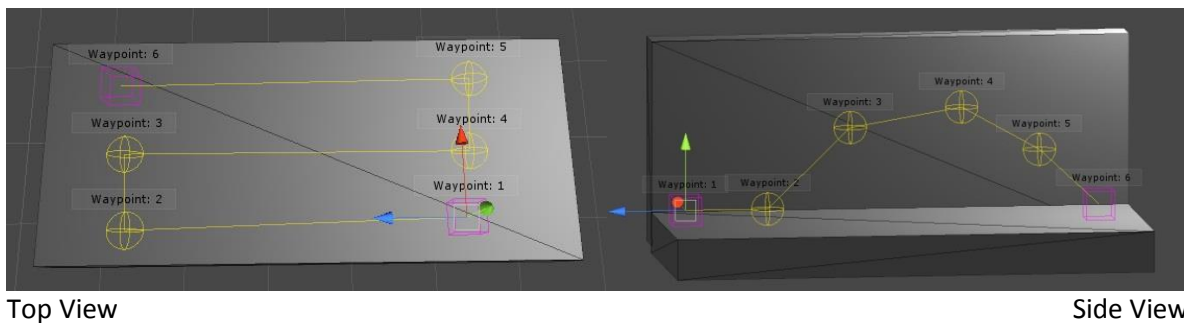
4. SETUP AND CREATE A PATH

To get SWS up and running in your own project, import the full package and (optionally) delete the example scenes including models, prefabs and textures. Ultimately, you only need the “Scripts” folder of SWS in order to function properly.

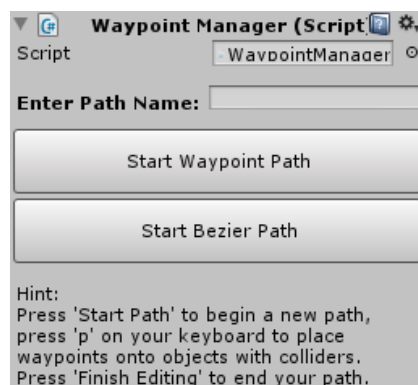
Hint: In the unfortunate case you have problems accessing these scripts via JavaScript, you will have to place all files and the “Editor” folder in another folder called “Standard Assets”, so Unity detects them as plugins.

Next, we set up our scene. Open Window > Waypoint Manager. For being able to place waypoints onto ground, please make sure you have some environment with colliders in your scene. If you are just experimenting with SWS, a big flat cube should do.

For developers of 2D games, at this point there is a small trick to consider: because in 2D you will not place waypoints onto ground but more likely in the air, the setup of a “background wall” will facilitate path creation. Remove that wall again when you’re done.



The creation of a path is a simple process. With your Waypoint Manager game object selected, name the new path in the text field next to “Enter Path Name:” and press “Start Path” or “Start Bezier Path”. Now the creation is enabled and your new path game object was parented to the Waypoint Manager.



A short warning: Do not click on other objects in scene view while creating a path to let the Waypoint Manager inspector window disappear, this will break internal functionality and most likely your path won't work correctly afterwards. If your path does not contain at least two waypoints, this will also spit out an error.

With the creation mode enabled, **position your mouse** in the scene view and **press P** to place new waypoints onto other objects (**they must have colliders!**). If you are satisfied with the result, press "Finish Editing" in the Waypoint Manager menu.

There is no need to setup layers, tags or something else to make SWS work. On the next page we continue finalizing paths, before we attach movement scripts to a walker game object.

During runtime, HOTween automatically draws its calculated paths with white lines and white spheres within the editor. If you do not want that to happen, uncheck "HOTween" in the gizmo settings.

Attention: Bezier paths underlie certain restrictions when creating them:

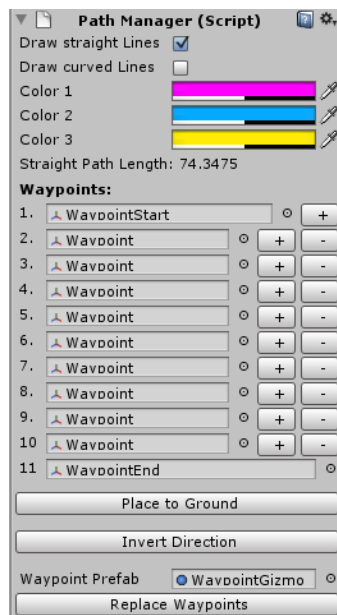
Bezier paths are still in development in the v3.x cycle. For now, these restrictions apply:

- Bezier paths do only work in 2D space, more importantly on the x/z axes
- The main path object has to be positioned at (0,0,0) - this happens automatically
- When moving a bezier path, its bezier points will be shifted slightly
- Message positions displayed in the editor are slightly off from their actual position

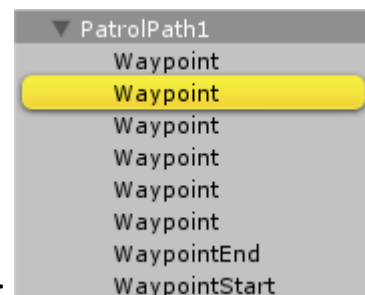
5. EDIT A PATH

Once you created a path and made a small change to the environment, so it does not fit anymore, there is still the option to edit your path to avoid the process of creating a new one.

- First, we investigate straight or curved paths (Example_Straight / Example_Curved) using the PathManager component. Select one of the paths (Image 1).



1:



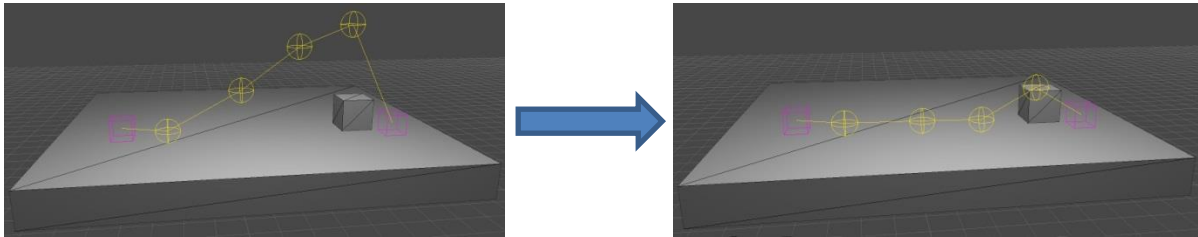
2:

In the scene view all waypoints now get a small info box above them, including a number corresponding to this overview. As discussed in chapter 3, you have the choice between straight and curved lines by simply checking the appropriate box, or both of them. The first color field is used for the starting and ending waypoint of this path, the second for straight line color and the third for curved line color. To move an existing waypoint, simply click its transform slot in this component, so it gets highlighted in the hierarchy (Image 2). Select that highlighted waypoint game object and move it through Unity's built in x/y/z handles.

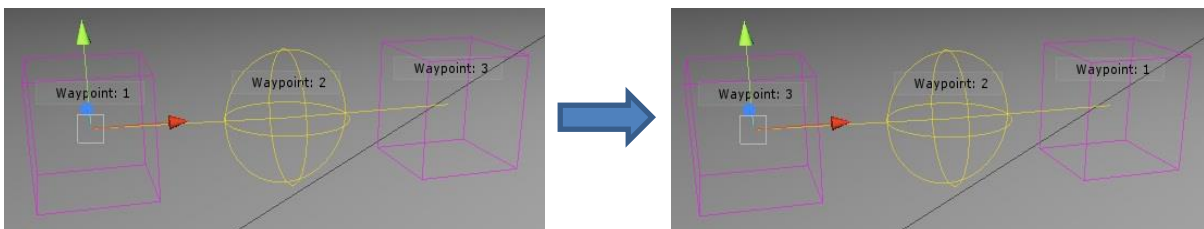
For adding new waypoints to an existing path, press the small "+" button next to a waypoint slot. This will create a new waypoint *after* the one selected, that gets placed at the same position. Furthermore, it gets the active selection, so you can move it directly to a desired position.

Removing a waypoint is as easy as that. Just press the small "-" button next to the waypoint slot you want to be removed. This automatically adjusts the waypoint array, deletes the selected waypoint from your scene and updates connections between them. The first and last waypoint of a path cannot be deleted, just delete the whole path game object instead.

The button “Place to Ground” will do exactly what it says: Every waypoint calls a ground detection method, which places them onto collided environment.

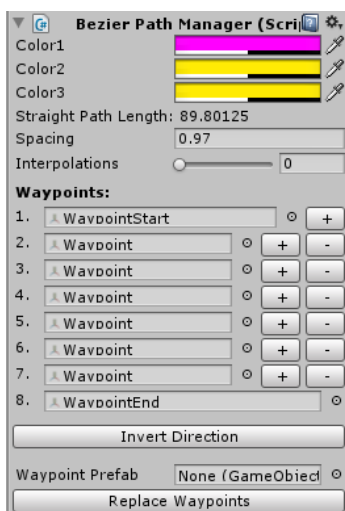


The next button, “Invert Direction”, inverts the order of waypoints which belong to this path. This causes no effect in the movement setting loop type: random.



The last button, “Replace Waypoints” can be used for replacing all waypoints of this path with a predefined prefab, that could hold custom components or a 3D gizmo, as seen in Example_Straight and the path named “GroundPath”. This way, all waypoints are clickable within the scene view.

- Next, we investigate the BezierPathManager component (Example_Bezier). Due to more complicated calculations, bezier paths, as the name implies, have more configurable options than curved ones.



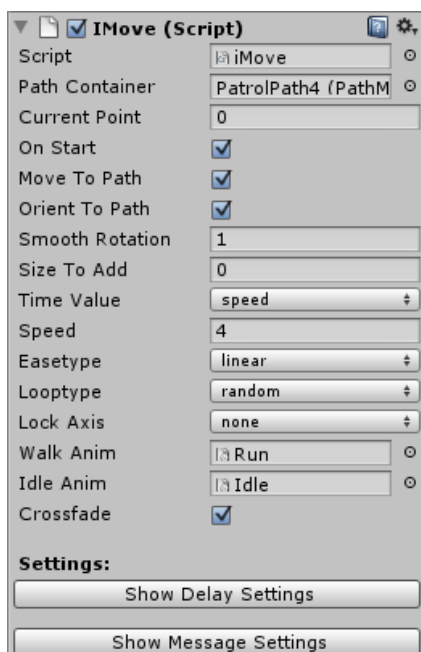
Spacing: This variable defines the spacing between points on the path. A higher value typically means fewer points, thus angular or smoothed paths, while a smaller value means more detail to its original form.

Interpolations: 0 results in no interpolation. Higher values cause interpolation between points on the path; therefore the original form gets smoothed out. From 2 and upwards the spacing between points will be halved.

The next section describes how to let a game object follow existing paths.

6. FOLLOW A PATH

For telling a game object it should follow a path, there is nothing more than the iMove, hoMove or bezierMove component needed. iMove is derived from iTween and will straightly move your walker, while hoMove is an implementation of HOTween, which supports straight and curved movement. Ultimately, it's up to you which plugin to choose, although HOTween is more optimized regarding performance. bezierMove is the only choice when using bezier paths and implements HOTween internally. First let's take a look at **iMove**. If attached to a game object that should move, you will see the following settings:



Path Container: Which path this object should follow. Expand your Waypoint Manager game object and drag the desired path transform onto this slot, or click the little circle at the end of this line for selecting an available path.

Current Point: At which waypoint this object should start moving. 0 = first waypoint.

On Start: Whether this object should start to move when the game launches.

Move To Path: Whether this object should walk to the first waypoint (true), or spawn there on start (false). If loop-type was set to loop, this will automatically close the path.

Orient To Path: Whether the walker will orient to its direction of travel.

Smooth Rotation: Smoothing parameter for rotation between waypoints (in seconds), enabled when "Orient To Path" is checked.

Size To Add: Additional height for the walker object.

Time Value: Speed determines the walker's velocity, while time is a constant value and describes how long it should take from one waypoint to the next.

Speed: Speed or time value, depends on "Time Value". How fast waypoints are passed.

Easetype: Ease type function of the movement. There are many ease types possible such as linear, spring, cubic, etc.

Looptype: Whether path movement should play once, loop, ping pong or randomize its selection between waypoints.

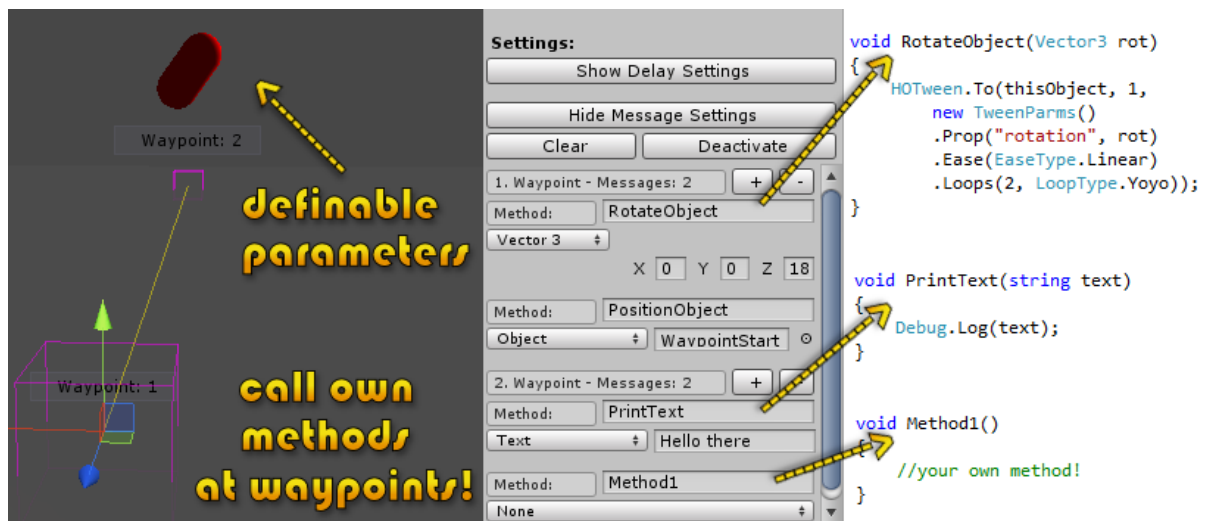
Lock Axis: Rotation axis to lock while moving on the path. Useful if waypoints have different heights and the walker should be restrained on one axis.

Walk Anim & Idle Anim: Animation to play during walk or waiting time. If no waypoint delay is set under “Delay Settings”, there is no waiting time.

Crossfade: Whether animations should fade in and out over a period of time.

Delay Settings: If a path is set, you will see a button to set all delay slots to a specified value or input fields to set delay for a specific waypoint. Useful if your walker should take a short break (in seconds) at a waypoint and then continues its walk.

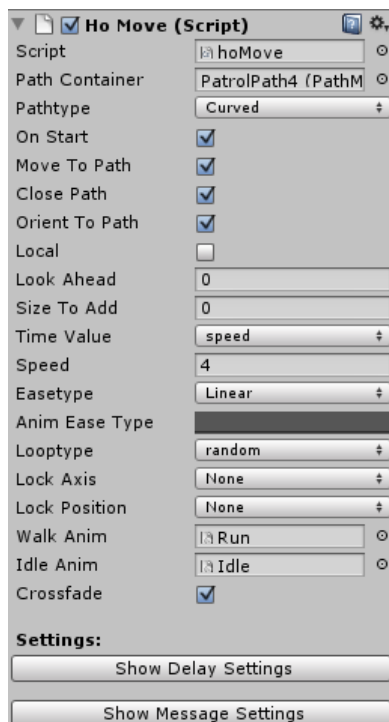
Message Settings: If a path is set, you will see a button for defining a method to call at each waypoint. You can call more than one method per waypoint. A script containing this method has to be attached to the game object or its children. The example scene “Example_Straight” and its walker game object “Capsule4” has a sample use case and a script named “MessageExample” attached to it:



When pressing “Show Message Settings”, message slots for each waypoint are being initialized. “Clear” will delete all slots and reinitialize them. “Deactivate” will delete all slots and won’t reinitialize them to save memory. Once you’ve shown the message settings, you can pass in the name of a method, contained in your own script attached to the object, to call at a specific waypoint. You can also submit a value to the method, the method then needs the type passed in as a parameter (see the image above):

- Objects – game objects, sound or animation files, scenes etc.
- Text – string value
- Numerical value – int, float
- Vector2 or Vector3

The **hoMove component** is very similar to iMove, however it does have some differences:



Pathtype: Lets you choose between straight and curved movement.

Local: Whether the tween should simulate in local space, rather than world space. Check this if your walker is parented to a moving path.

Look Ahead: For how much (in percentage) the object should look ahead on the path, enabled when “Orient To Path” is checked.

Move To Path: Whether this object should walk to the first waypoint, regardless of the Looptype setting.

Close Path: Tries to close the path smoothly.

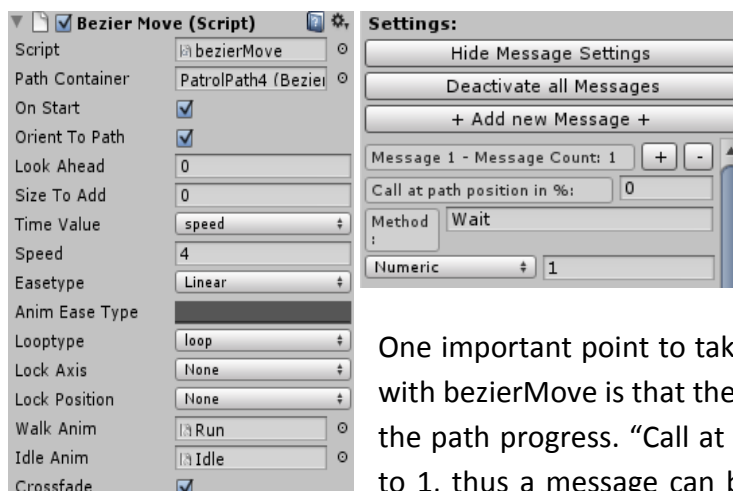
Speed: If “Time Value” equals time, this value describes the time for the whole path, not the time between waypoints.

Anim Ease Type: Custom easing curve to use, if Easetype = AnimationCurve is selected.

Looptype: For performance reasons and smooth loops “random” does not calculate a random waypoint after each waypoint, it calculates a random order of all waypoints once.

Lock Position: Constrains the movement of this object on the selected axis.

Lastly, we take a look at the **bezierMove component**.



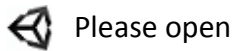
In contrast to hoMove, bezierMove only supports curved (bezier) paths.

Local paths, close path and looptype random aren't supported at the moment.

One important point to take into account when using Messages with bezierMove is that they are not based on waypoints, but on the path progress. “Call at path position in %” is a value from 0 to 1, thus a message can be called anywhere on the path, not just at waypoints. In the editor, the message position is indicated by a small info box on the path. That's just an approximate position for the message. The exact position is displayed at runtime, when the object moves on the path, via a purple gizmo sphere.

7. METHODS, RUNTIME ACCESS

Another example scene demonstrates runtime access and instantiation.



Example_Runtime

This scene consists of three examples.

Using Unity's built-in GUI System, in "Example 1" you can instantiate a walker and a path during runtime. When they are instantiated, there will be a button to start the movement, but also to reposition the instantiated path, so the walker will update its path after each waypoint. Once the movement was started, you can reset, stop and continue the tween.

In "Example 2", you can also instantiate a walker along with a path and start movement. If you instantiated the objects of the first example, there will be a button for switching paths.

"Example 3" introduces movement based on user input. The main concept of this is to use iTween's `PointOnPath()` method for the movement of objects from waypoint to waypoint. While still following the basic principles of SWS, this approach has nothing really in common with it. `UserInput.cs` is well documented for further examined purposes.

Next, we investigate the coding possibilities of SWS. As programmed in `RuntimeExample.cs`, you could start movement at a specific event or anytime in your game. Just call these methods on a walker object whenever you need it to start.

Calling movement settings via code

C# and JavaScript:

```
gameObject.SendMessage("StartMove");
```

or (C#, iMove – works with other movement scripts too):

```
iMove moveScript = gameObject.GetComponent<iMove>();  
moveScript.StartMove();
```

Do not place this code into `Start()`, just check "OnStart" in the script's movement settings to achieve the same result.

Runtime Instantiation

Basically, for runtime instantiation there are these few additions:

- ✓ To have access to an instantiated path, call
`WaypointManager.AddPath(instantiated path gameObject)`
– note that this will abort if the passed path gameObject name already exists.
- ✓ You can change the path container of a walker object at runtime using

(C# and JavaScript):

`(movement component).SetPath(WaypointManager.Paths["path name"]);`

or (C# and JavaScript)

`(walker gameObject).SendMessage("SetPath",
WaypointManager.Paths["path name"]);`

SetPath() automatically calls StartMove() afterwards.

- ✓ Repositioning a path will result in updated waypoints of its walker objects (they always follow the current path/waypoint positions). iMove updates its waypoints after each waypoint. hoMove can update its waypoints after each loop.
Do not reposition hoMove's path while moving as this will break the iteration.
It is not recommended to move a bezier path at runtime either.

Wait, Reset, Stop & Pause, Resume

Movement scripts have some extra methods to facilitate movement control.

`Wait(float value)`: Pauses the animation and waits the value defined before moving on.

`Reset()`: Immediately stops the walker's movement and resets it to the first waypoint.

`Stop()`: Immediately stops the walker's movement (this destroys the tween).

iMove: For resuming movement, call `StartMove()` again.

hoMove/bezierMove: Do not call `Stop()` if you want to be able to continue movement from the same position later, use `Pause()` & `Resume()` instead.

Only hoMove/bezierMove:

`Pause()`: Pauses the animation at the current position and does not destroy the tween.

`Resume()`: Resume the paused animation/tween.

Changing a tween's speed at runtime

Movement scripts have a method called "ChangeSpeed" which takes a float as a parameter. Since iTween can't change existing tweens, it is necessary to recreate or overwrite the existing tween with a new one and new speed parameters. This method automatically does that for you. Just pass in a float value determining the new speed value at any time. Please do note that recreating a tween does affect performance, so you shouldn't call this method at every frame. In hoMove/bezierMove, this parameter only sets the tween's timescale internally and does not recreate the tween. None of these methods manipulate the speed of your object's animation, so you would have to adjust it separately.

Populating messages at runtime

If you want to populate message options at runtime through code instead of setting them in the inspector, all movement scripts have a method named "GetMessageOption" for this reason. Basically you get the message option for a waypoint and set its properties via script:

C# Example:

```
void Start () {
    // get movement script reference
    hoMove hoScript = gameObject.GetComponent<hoMove>();

    // get the message option at the 3rd waypoint, first message
    MessageOptions opt = hoScript.GetMessageOption(2, 0);
    // set the first method name to "DebugSomeText"
    opt.message[0] = "DebugSomeText";
    // use a string / text value as parameter and pass in "Hello World"
    opt.type[0] = MessageOptions.ValueType.Text;
    opt.text[0] = "Hello World";
}

// with the above code, this method will print "Hello World" at the 3rd waypoint
void DebugSomeText(string text)
{
    Debug.Log(text);
}
```

In the example described above we modified the first message of the 3rd waypoint. As waypoints can have more than one message, you could also access further messages, e.g. the second message. GetMessageOption() automatically adds new messages if necessary.

```
// get the message option at the 3rd waypoint, second message
MessageOptions opt = hoScript.GetMessageOption(2, 1);
// set the second method name to "DoSomething"
opt.message[1] = "DoSomething";
```

In case you're using bezierMove, you also have to define a value for the path progress - where the event should happen - named "opt.pos" (float value from 0 to 1).

8. CONTACT

As full source code is provided and every line is well-documented, feel free to take a look at the scripts and modify them to fit your needs.

If you have any questions, comments, or suggestions,
do not hesitate to contact us. We will be pleased to answer them.

Visit the Unity3d SWS thread here:

[http://forum.unity3d.com/threads/115086-Simple-Waypoint-System-\(SWS\)-RELEASED](http://forum.unity3d.com/threads/115086-Simple-Waypoint-System-(SWS)-RELEASED)

or send us an email at:

info@rebound-games.com

***Again, thanks for your support,
and good luck with your games!***

Rebound Games

www.rebound-games.com

9. VERSION HISTORY

V 1.0

- Initial Release.

V 1.1

➤ Fixes & Changes

- iMove: unnecessary bool repeat check in StartMove() removed
- iMove: Delay variable for all waypoints removed
- iMove: variable pathContainer is now of type "PathManager"
- iMove: implemented orientToPath to define walker orientation
- WaypointEditor: displays hint for more transparent handling
- Example Straight: Bootcamp Soldier uses new path to present animations, Capsule3 on path "PatrolPath1" also

➤ Features

- iMove: looptype "random" added
- iMove: walk height is now settable per walker object (sizeToAdd)
- iMove: Delay array added to allow delay at a specific or for all waypoints
- iMove: support for animation during walk and waiting time added
- iMoveEditor: added, displays custom iMove Inspector
- Documentation: modified to represent the current version

V 1.2

➤ Fixes & Changes

- replaced some calculations with iTween's internal methods
- iMove now has direct access to Path Manager waypoint positions (your path will update at runtime on position changes)
- reset delay at waypoints (StopAtPoint[]) if path container gets null

➤ Features

- WaypointManager.cs: AddPath() added to support more flexibility and path instantiation at runtime
- iMove.cs: added SetPath(), Stop() and Reset() methods
- Example_Runtime.cs: new script to demonstrate combinations of those new methods

V 2.0

➤ Fixes & Changes

- iMove: animation checks outsourced in own methods
- iMove: animation does now stop correctly at the end using loop type none
- Example_Waypoints renamed to Example_Straight
- iMoveEditor: custom representation of variable sizeToAdd removed
- iMoveEditor: delay array StopAtPoint gets created at start if not set in editor
- WaypointManager: path dictionary gets cleared at scene change
- WaypointManager: now also responsible for initialization of HOTween

➤ Features

- curved movement using HOTween
- introduced hoMove & new sample scene Example_Curved
- PathManager: checkboxes for straight or curved lines between waypoints
- PathManager: additional color field for curved lines
- new example scene Example_Runtime for runtime access and instantiation
- iMove: smooth rotation variable added
- UserInput.cs added, demonstration of movement based on user input
- documentation restructured and updated

V 2.0.1

➤ Fixes & Changes

- iMove/hoMove: bugfix StopAtPoint[] length null reference at resizing paths
- iMove/hoMove: animation component need not be attached to main object
- iMove: current location marked as public, allowing multiple starting points

➤ Features

- PathManager: added button for replacing all waypoints with a prefab
- iMove: lock-axis option added

V 2.0.1a

➤ Fixes & Changes

- hoMove: hotfix animation component null reference on Start()

V 2.1

➤ Fixes & Changes

- hoMove: removed critical waypoint distance value that led to unexpected behavior when tweening an object with a very high speed value or timescale
- hoMove: completely rewritten to incorporate with partial tweens. The tween and its runtime checks should be a significant amount faster now
- hoMove: partial tweens allow easing options between waypoints instead of an easing curve for the whole path
- hoMove: separated MoveToPath and its internal ClosePath option, you can toggle them independently now as ClosePath has its own checkbox
- hoMove: movement parameters (waypoint positions) do not update at each iteration anymore, you have to call InitWaypoints() and recreate the tween manually with StartMove(), for example at the last waypoint using messages
- simplified the Waypoint Manager even more, it shows only one button now
- switched widget position. You'll find the Waypoint Manager setup widget under "Window > Simple Waypoint System" and iMove/hoMove under "Component > Simple Waypoint System"
- documentation updated, contact info & email changed

➤ Features

- iMove/hoMove: custom methods callable per waypoint. Both editor scripts for iMove/hoMove were changed accordingly, so that this feature comes with a new message settings field, easily adjustable by the user
- custom methods are using SendMessage() and allow one parameter of the type object, text, numeric, vector2 or vector3. The class of these types was centrally placed in WaypointManager.cs
- new example script "MessageExample.cs" demonstrates the usage of custom messages in the scene "Example_Straight" on walker "Capsule4"
- hoMove: integrated HOTween's new lock-position option that limits movement on the selected axis

V 2.1.1

➤ Fixes & Changes

- hoMove: fixed Pause() so it actually does pause the tween correctly
- hoMove: fixed bug on "closePath" not playing a walk animation while moving
- MessageExample.cs: added methods for pausing a tween with messages
- HOTween: updated to version 1.1.333, big thanks to Daniele who fixed the constrained rotation using "lockAxis" and improved "lookAhead" rotation, also optimized partial tweens and made them more garbage collector friendly

➤ **Features**

- PlayMaker actions: added SWS_SetPath, SWS_GetPathNode, SWS_SetPathNode under “Simple Waypoint System” in the Action Browser

V 2.1.2

➤ **Fixes & Changes**

- hoMove: skips unnecessary delay yield if no delay was set in the inspector
- iMove/hoMove: message initialization refactored to be runtime compatible
- PlayMaker readme added to SWS > Scripts > PlayMaker
- HOTween: updated to version 1.1.724, now supporting linear paths and tweening parented objects + much more (thanks@HOTween’s author!)
- New “RotatingPath” in scene “Example_Curved” demonstrates local tweens
- Documentation: updated to explain new runtime methods

➤ **Features**

- Methods to change speed and populate messages at runtime added

V 3.0

➤ **Fixes & Changes**

- WaypointEditor: changed waypoint placement combination from alt + mouse click to keyboard key ‘P’. Auto-focuses scene view when starting a new path
- hoMove: fixed an issue where Pause() + Resume() could affect other tweens
- hoMove: added slot for an animation curve when using a custom easetype
- HOTween: updated to v1.1.727

➤ **Features**

- Bezier paths! (only 2D - x,z axis - for now)
 - At the cost of some movement settings, bezier paths give you more control over the shape of your path: drag small handlers per waypoint in order to further define the curves
 - WaypointManager: now creates normal or bezier paths
 - BezierPathManager, BezierPathEditor and bezierMove scripts added
 - bezierMove can trigger messages anywhere on the path, at a given percentage – not restricted to waypoint positions
 - New example scene “Example_Bezier”

V 3.0.1

➤ **Fixes & Changes**

- hoMove: fixed NullReferenceException when accessing an empty tween

V 3.0.2

➤ Fixes & Changes

- hoMove: fixed Exception when destroying tween while moving via Stop()
- WaypointManager.cs: added waypoint renaming scheme on path creation
- PathManager, BezierPathManager: gizmos consider new waypoint names

V 3.0.3

➤ Fixes & Changes

- hoMove: fixed bug where stopping movement at a waypoint could potentially stop executing other messages
- iMove: fixed Stop() method, now actually stops movement