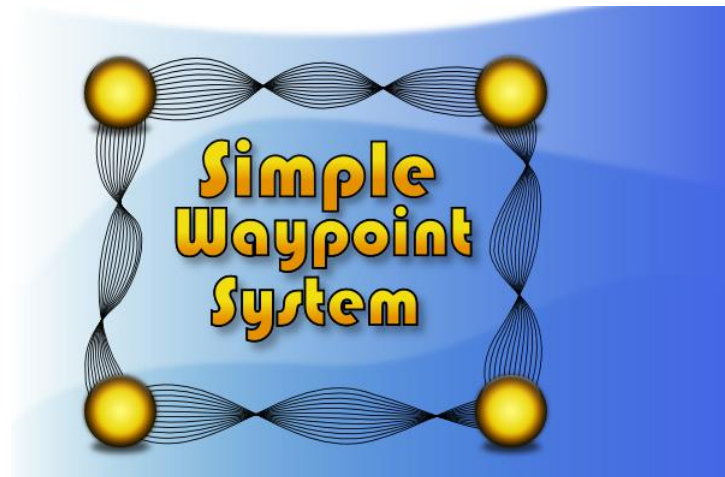


Simple Waypoint System (SWS)

by baronium3d

v1.2



1. SWS – WHAT’S THIS?	2
2. SCRIPT CONNECTIONS AND PREPARATION	3
3. EXAMPLE_WAYPOINTS	4
4. SETUP AND CREATE A PATH	6
5. EDIT A PATH	8
6. FOLLOW A PATH	9
7. CONTACT	12
8. VERSION HISTORY.....	13

Thanks for buying SWS!

Your support is greatly appreciated!

1. SWS – WHAT'S THIS?

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
- 2D or 3D space
- Compatible with C# or JavaScript
- Undo and Redo
- Full source code in C#
- Every line documented
- Example Scene

As you have seen in the features, SWS uses iTween to simulate movement in 3d space. iTween is a free and open source animation library for Unity3d. More information and the latest version is available from the Google Code repository link on the main site here:

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

Please feel free to donate to this project.

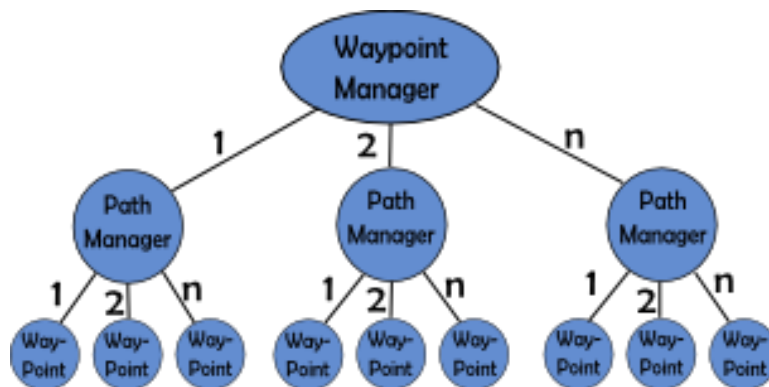
Your donation will support the development of iTween.

2. SCRIPT CONNECTIONS AND PREPARATION

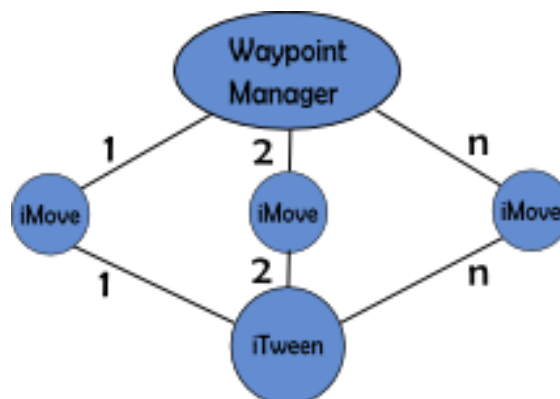
If you are new to Unity3d, please take a quick break and get dirty with its main functionalities, because this documentation will assume you have some basic knowledge regarding the interface and game mechanics.

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, 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 script iMove and make use of the animation library, a copy of iTween is also required in your project. iTween need not be attached to any game object.



3. EXAMPLE_WAYPOINTS

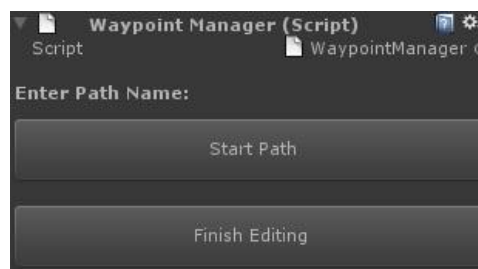
Once you downloaded SWS from the Unity Asset Store and imported it in your project, it is recommended that you get familiar with its mechanics and take a look at the provided example scene. Please open

 Example_Waypoints

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 Path Manager is enabled in the gizmo settings. A red capsule or platform is positioned besides every starting point which will “walk” on this path. If you click on the play game button, you can see them in action.

So, how this SWS stuff works?

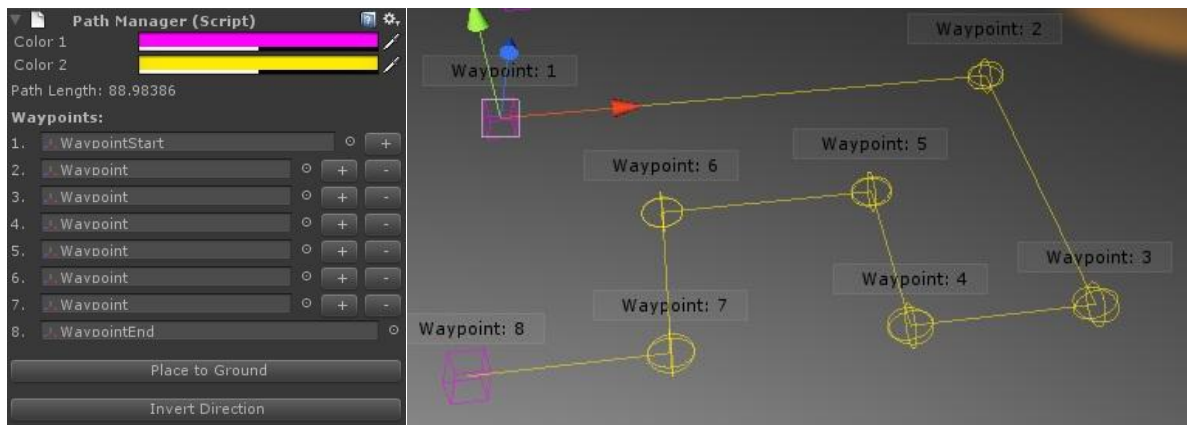
This scene consists of four game objects; the game objects “Walker” and “Waypoint Manager” are relevant for us. By clicking the Waypoint Manager game object, you will see this inspector window with two buttons:



By clicking on these 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 hierarchy and you will see all integrated paths in this scene:

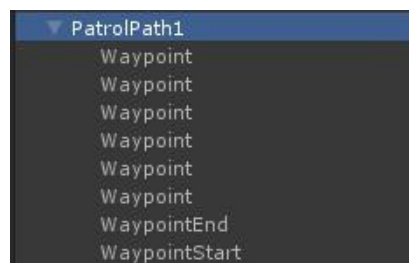


If you click on one of these paths, our Path Manager component shows up and we see some detailed information about the path we clicked. For example, this is what we get by inspecting “PatrolPath1”:

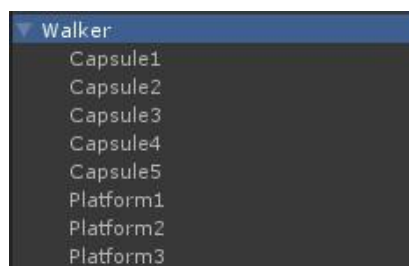


In the left picture, our Path Manager component, we see two gizmo color fields, the total length of this path and also a transform slot for each waypoint involved. At the end we have the option to place all waypoints down to the ground or invert the existing path direction. Read more about its functionalities 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.

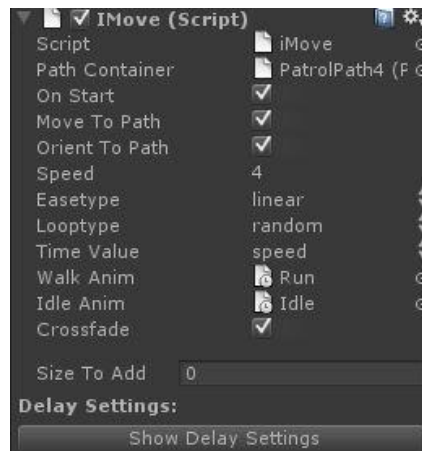
By expanding our selected path within the hierarchy, we are able to see every single waypoint game object of this path:



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 game object, which follows a path. Expand the “Walkers” game object to see what walkers we currently have in our example scene.



Each walker has an iMove component attached to it. As mentioned in chapter 2, this script is responsible for movement.



There is no need to setup layers, tags or something else to make SWS work. Just create a path, drag iMove onto your game object you want to move, select the path, check the other settings and you are ready to go. iMove settings are described deeper in chapter 6 – Follow a Path.

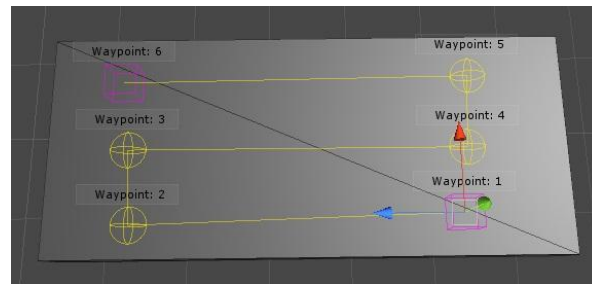
4. SETUP AND CREATE A PATH

To get SWS running in your own project, you first need to drag these scripts into it:

- ✓ CreateWPMManager.cs
- ✓ WaypointManager.cs, WaypointEditor.cs
- ✓ PathManager.cs, PathEditor.cs
- ✓ iMove.cs, iMoveEditor.cs
- ✓ iTween.cs

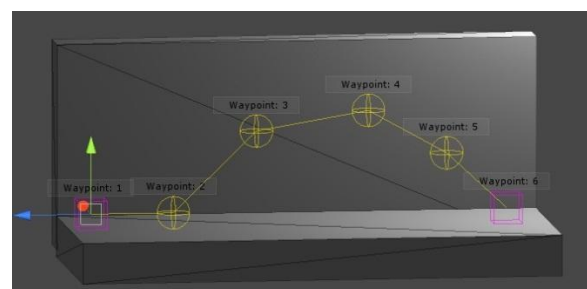
CreateWPMManager, WaypointEditor.cs, PathEditor.cs and iMoveEditor.cs need to be in a folder called “Editor” to work correctly. Please create a new folder for this purpose and drag them in. Next, we set up our scene.

Open Window > Waypoint Manager. Basically that’s all. But for being able to place waypoints onto ground, please make sure you have some environment in your scene. If you are just experimenting with SWS, a big flat cube should do.



Top View

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, and make it more effective. Remove that wall again when you’re done.



Side View

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”. Now the creation is enabled and your new path game object was parented to Waypoint Manager.

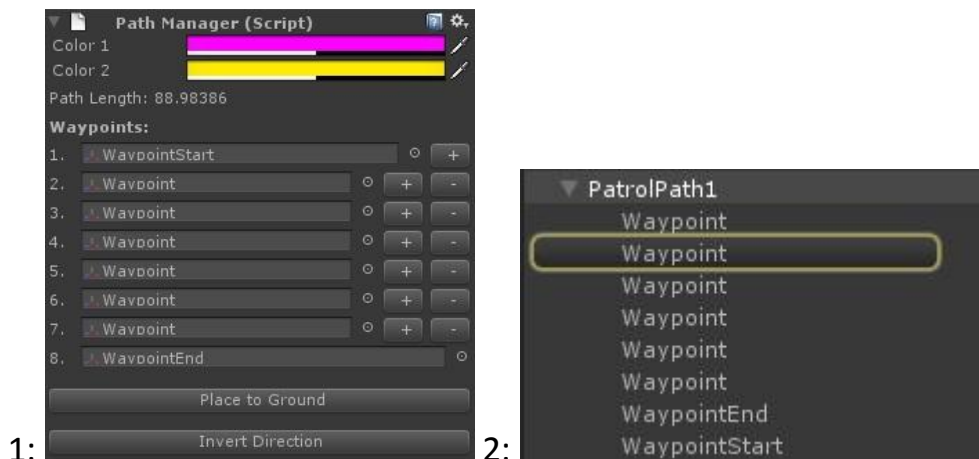
A short warning: Do not click on other objects in scene view while creating a path to let the Waypoint Manager inspector window disappear, this will skip the renaming task of the first and last waypoint for better visibility, so you would have to rename them yourself. That’s nothing critical, but wasted time.

With the creation mode enabled, hold down **left alt** and press **left mouse button** to place new waypoints onto other objects. If you are satisfied with the result, release left alt and press “Finish Editing” in the Waypoint Manager. This will automatically rename the first and last waypoint and switch their gizmo icons.

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.

Select the path you want to edit, so the Path Manager component shows up.
(Picture 1)

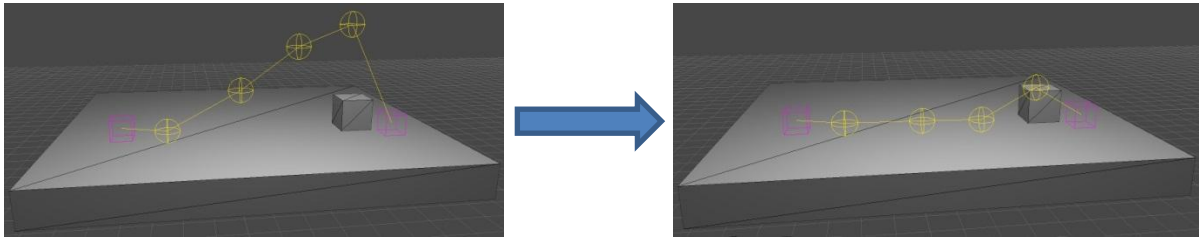


In scene view all waypoints now get a small info box above them, including a number corresponding to this overview. To move an existing waypoint, simply click its transform slot in this component, so it gets highlighted in the hierarchy (Picture 2). Select this highlighted waypoint game object and move it through Unity's built in x/y/z handles.

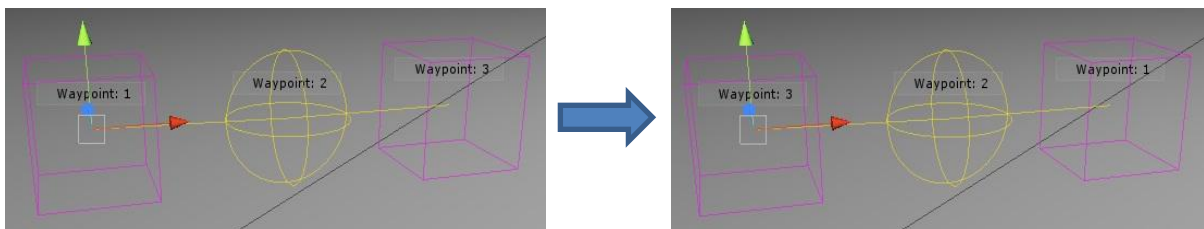
To add new waypoints to an existing path, press the small "+" button next to a waypoint slot. This will create a new waypoint *after* the one selected, that gets placed at the same position. Furthermore, it gets the active selection, so you can move it directly to a desired position.

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

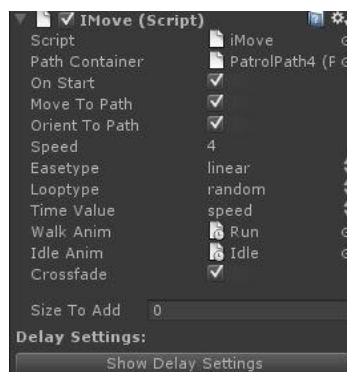


The last button, “Invert Direction”, inverts the order of waypoints which belong to this path. This causes no effect in loop type random.



6. FOLLOW A PATH

To tell a game object it should follow a path, there is nothing more needed than the iMove component. Attach it to a game object which should move, and 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.

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).

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

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 random selection between waypoints.

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.

Walk Anim & Idle Anim: Animation to play during walk or waiting time.

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

Size To Add: Additional height for the walker object.

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 at a waypoint and then continues its walk.

Just play a bit with these settings and see what they do!

Calling iMove by code

You could also start movement at a specific event or time in your game.

Just place these lines into your custom script attached to your walker object and call it whenever you need it to start.

C# and JavaScript:

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

or (C#):

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

**But do not place this code into a Start() method,
just check “OnStart” in iMove settings for this purpose.**

Runtime Instantiation – Change Path – Stop & Continue Movement

Version 1.2 adds the support of runtime instantiation and changes. Please have a look at the provided C# script “**RuntimeExample.cs**” to understand the new functionality.

Basically, 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 exists already.
- ✓ You can change the path container of a walker object at runtime using
(C# and JavaScript):
`(iMove component).SetPath(WaypointManager.Paths[“path name”]);`
or (C#):
`(walker gameObject).GetComponent<iMove>().SetPath(WaypointManager.Paths[“path name”]);`
or (C# and JavaScript)
`(walker gameObject).SendMessage(“SetPath”,
WaypointManager.Paths[“path name”]);`
- ✓ Besides the “SetPath(..)” method, Version 1.2 of iMove.cs comes with two other new methods:
 - Stop() – disables any running movement routines
 - Reset() – stops any movement and resets to first waypoint
- ✓ Repositioning a path will result in updated waypoints of its walker objects
(they always follow the current path/waypoint positions)

7. 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 me. I will be pleased to answer them.

Visit my 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 me an email at:

baronium3d@googlemail.com

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

baronium3d

8. 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 Scene: 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