Easily draw lines with Rigidbody2D and Collider attached

In this document are the instructions for start using the asset and which features it offers to your project.

To send feedback, or any question go to the publisher page and get in touch,
https://www.assetstore.unity3d.com/en/#!/search/page=1/sortby=popularity/query=publisher:19535

Daniel C Menezes

Contents

1. Project Settings

After getting the 2D Physics Draw asset, you have to make sure you got some simple configurations set right in your project so you can start drawing lines with physics attached.

1.1. Camera

Since the project is suited for 2D games, make sure the camera projection is set to **Orthographic** mode as in the Figure 1.
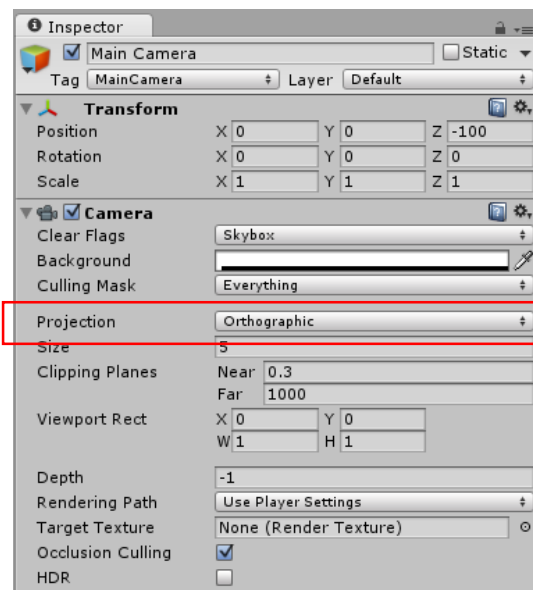

Figure 1 – Camera Settings

1.2. Layer

For the manager to work properly you need to add two layers in your project, those are **Drawing** and **OnDraw**. These layers can be placed in any position, since they are searched in code by the name.
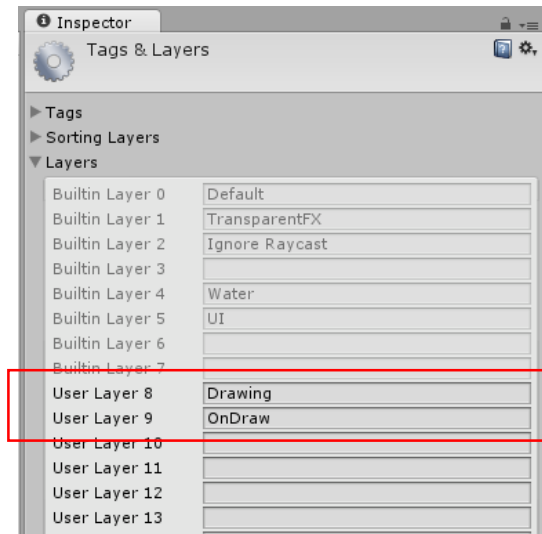
Figure 2 – Tags and Layers

1.3. Drawing Manager

The only object you should make sure is included on the game **Hierarchy** is the *DrawingManager* prefab that is in the asset folder **"…/2DPhysicsDraw/Prefabs"** (Figure 3). If this object is not in your Hierarchy, you just need to include it and you are ready.
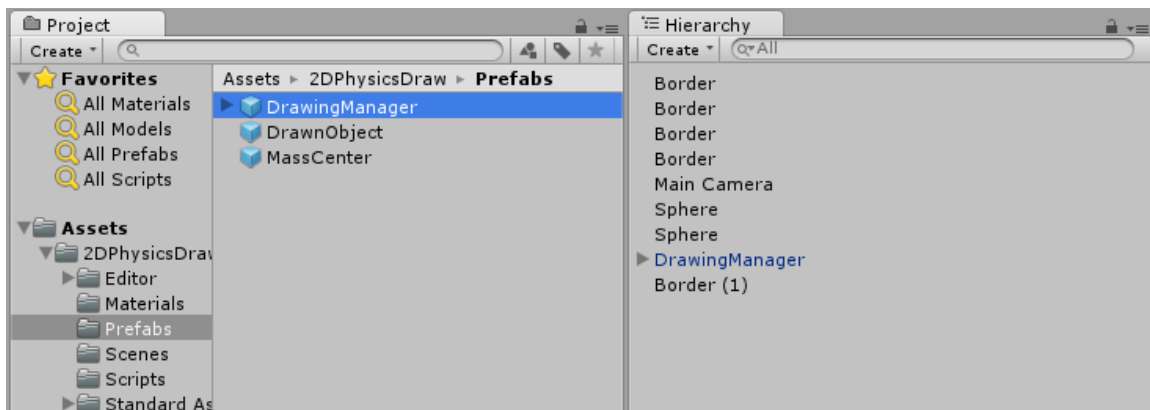

Figure 3 – DrawingManager Path and Game Hierarchy

This object includes the necessary script that manage the drawing process and the variables you can change directly on the **Inspector** to modify the lines you are going to draw.

2. Line Settings

In order to better suite every project you may want to use the asset on, some settings can be predefined for each line the user will draw, all the available variables used in the **DrawingManager** script displayed in the **Inspector** (Figure 4) are public and easily accessible by the user to modify before or during the gameplay.
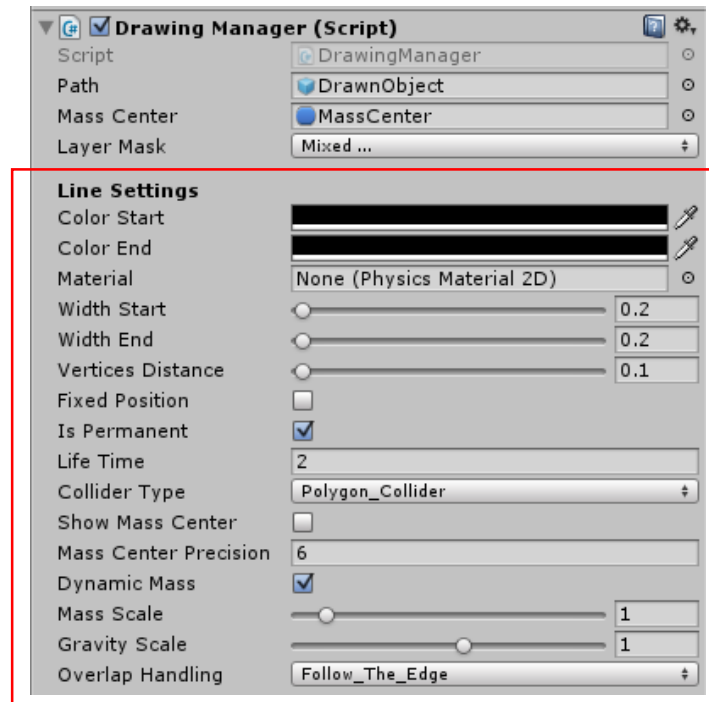

Figure 4 – Line Settings in the Inspector

The settings variables are shown and explained above.

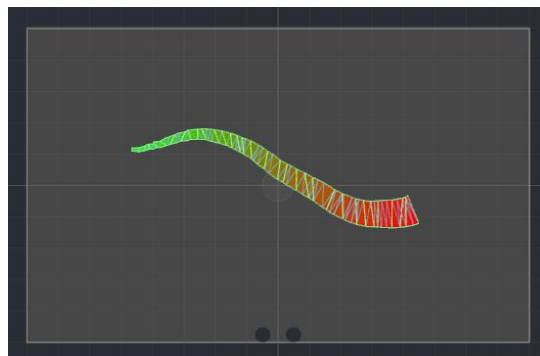**colorStart**: start color of the line;

colorEnd: final color of the line;

Figure 5 – Start and Final Color

material: physics material 2D attached to the collider (read more on http://docs.unity3d.com/Manual/class-PhysicsMaterial2D.html);

widthStart: start width of the line;

widthEnd: final width of the line;

verticesDistance: minimum distance between vertices of the line;

Each line is composed for many small straight lines that are a composition of two points, vertices.

fixedPosition: if object will not move after you draw it (true), or it will be gravity driven (false);

isPermanent: if drawn object will be destroyed after t = lifetime seconds, choose mark this box;

lifetime: lifetime, in seconds, for the non-permanent objects;

colliderType: choose between Polygon Collider 2D (Figure 6) or Edge Collider 2D (Figure 7)

Obs. 1: Be aware that Edge Collider 2D do not collider with each other cause they do not have volume (read more about Polygon Collider 2D on http://docs.unity3d.com/Manual/class-PolygonCollider2D.html

And more about Edge Collider 2D on http://docs.unity3d.com/Manual/class-EdgeCollider2D.html)

Obs. 2: The Edge collider is more recommended to use in case of fixed position drawings, since it is generally used for platforms and ground objects.
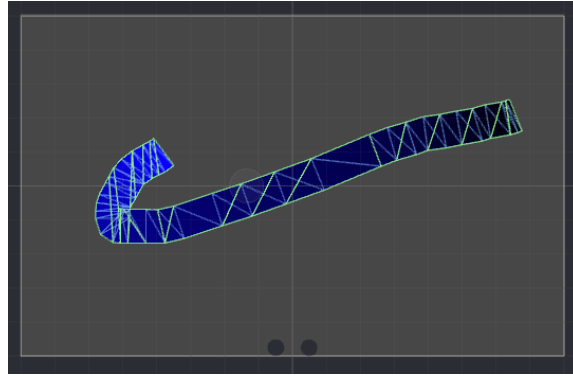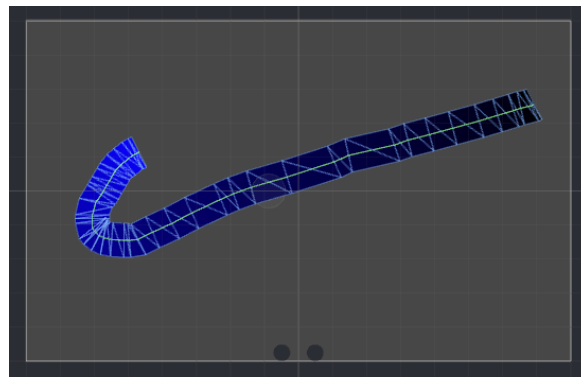
Figure 6 – Polygon Collider 2D

Figure 7 – Edge Collider 2D

showMassCenter: You can display the mass center of the drawn object in the gameplay (true) or not (false)

massCenterPrecision: multiplier to make the center of mass calculation more precise relative to the width of the line

dynamicMass: if mass is calculated relative to the width of the line (true), or it has a fixed mass (false)

massScale: mass scale when dynamicMass == true, or fixed mass when dynamicMass == false

gravityScale: gravity scale, when < 0 objects "float"

overlapHandling: to handle, or not, <u>overlaps</u> (Figure 8) when the player is drawing the lines, you can choose between Follow_The_Edge, Cut_After_Collision, or None.
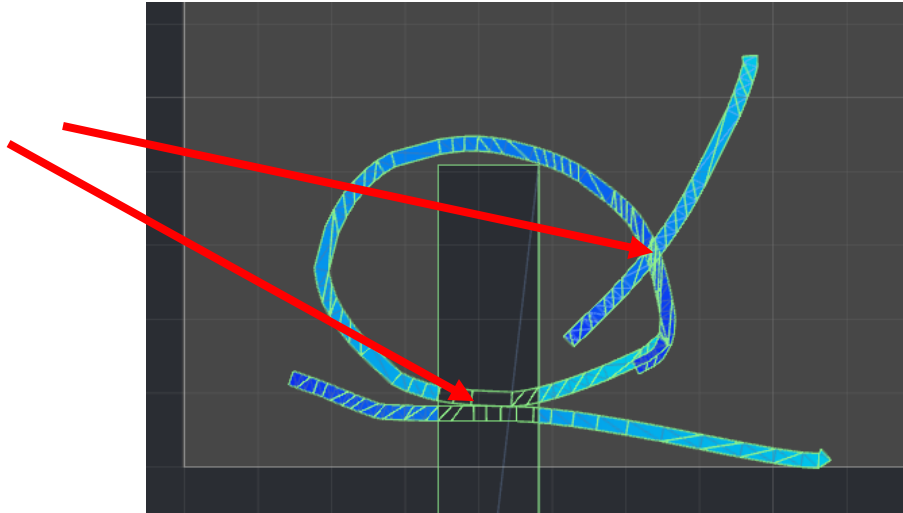

Figure 8 – Overlaps Examples

The types of overlap handlings are explained above.

<u>Follow_The_Edge:</u> if, during the drawing, collide with an object, it follows the edge without penetrating (Figure 9).
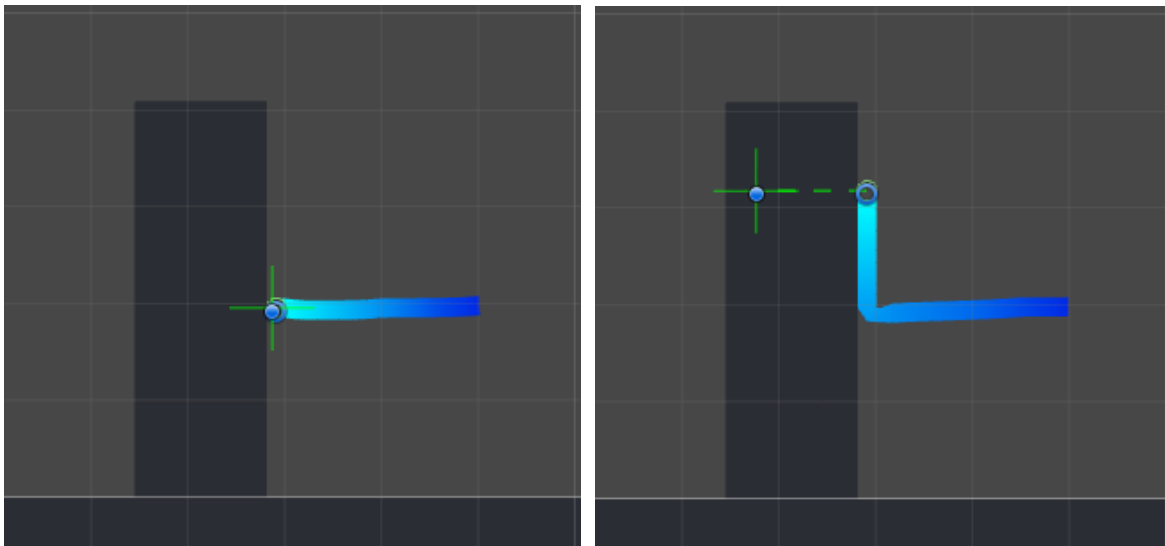

Figure 9 – Follow the Edge Handling Type

Cut After Collision: if the drawing overlaps an object, when you release the mouse button, it redraw the line without the part of the line after the first collision (Figure 10).
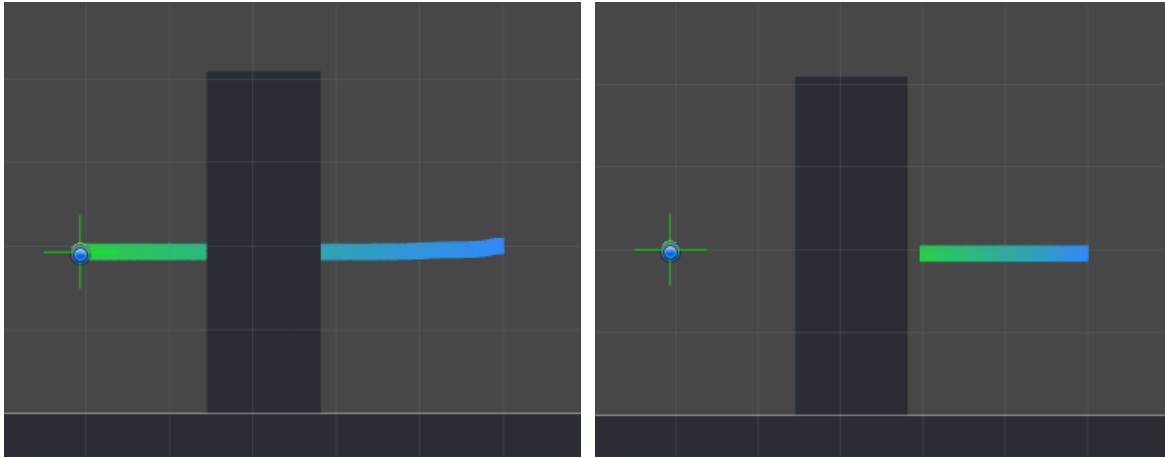

Figure 10 – Cut After Collision Handling Type

Obs.: The blue variables are also public, and they can have values beyond the range showed in the Inspector, you just have to modify it in the *DrawingManager* script.

freezeWhileDrawing: Check this box makes the drawn objects freeze in the position they are in the moment you start drawing another.

Obs.: To make any other object freeze while drawing, add the script **FreezeMoving.cs** to it.

tagCantDraw: Write the tags of the objects that you can't draw over. You should write all the tags you need <u>without space</u> and <u>separated by comma</u> ",".
If you want the objects with the tags "Wall" and "Ground" to be areas where you can't draw, write **Wall,Ground**

3. Version Control

Version	Description
1.0	- DrawingManager: colorStart / colorEnd fixedPosition massCenterPrecision Material isPermanent dynamicMass widthStart / widthEnd Lifetime massScale verticesDistance showMassCenter gravityScale
1.1	- Bug correction: while drawing with edge collider, an invisible the collider were created in the center of the scene
2.0	- Feature addition to the Drawing Manager: overlapHandling
2.2	- Bug correction: touch on mobile devices - Bug correction: Layer collision matrix is now set in code - Center of Mass bug correction.
3.0	- Bug correction: Moving camera is now possible; - Feature addition: freezeWhileDrawing, tagCantDraw; - New scene addition: Moving Camera Scene for testing with a player and a moving camera;