Gigi's Juicy Level Dots

Unity Plugin - v1.0

Developer's Guide - Get Juicy!

Modern games are juicy! They wiggle, jiggle, and bounce - giving extreme amounts of feedback with every user interaction. At the same time, they're simple and elegant - using minimalistic designs to create the compelling experiences that makes them successful.

Unfortunately, adding juicy elegance takes time. Time most of us would rather be spending on what we really want to work on - the game! Which is why Level Dots provides a complete solution for a common interface need.

Show the Player Which Level/Chapter/Mission Their On!

In Unity, the easiest way to show your players what level their on is this:

```
GUI.Label (new Rect (5,5,100,50), "Level: " + currentLevel);
```

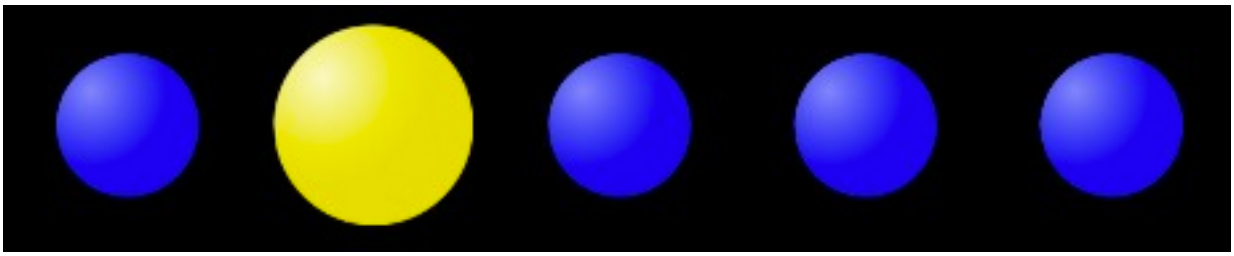
That line of code draws something like this:

On the plus side, it's a simple solution. On the minus side, it's ugly, inelegant, and doesn't convey enough information. Imagine how you would expand this to also indicate how many total levels there are, and pretty soon, you've got an ugly, text based interface that's taking away from your game.

The Level Dots solution is just as easy to code:

```
myDots.UpdateState (numLevels, currentLevel, true);
```

And now, you get a single draw call that creates something like this:



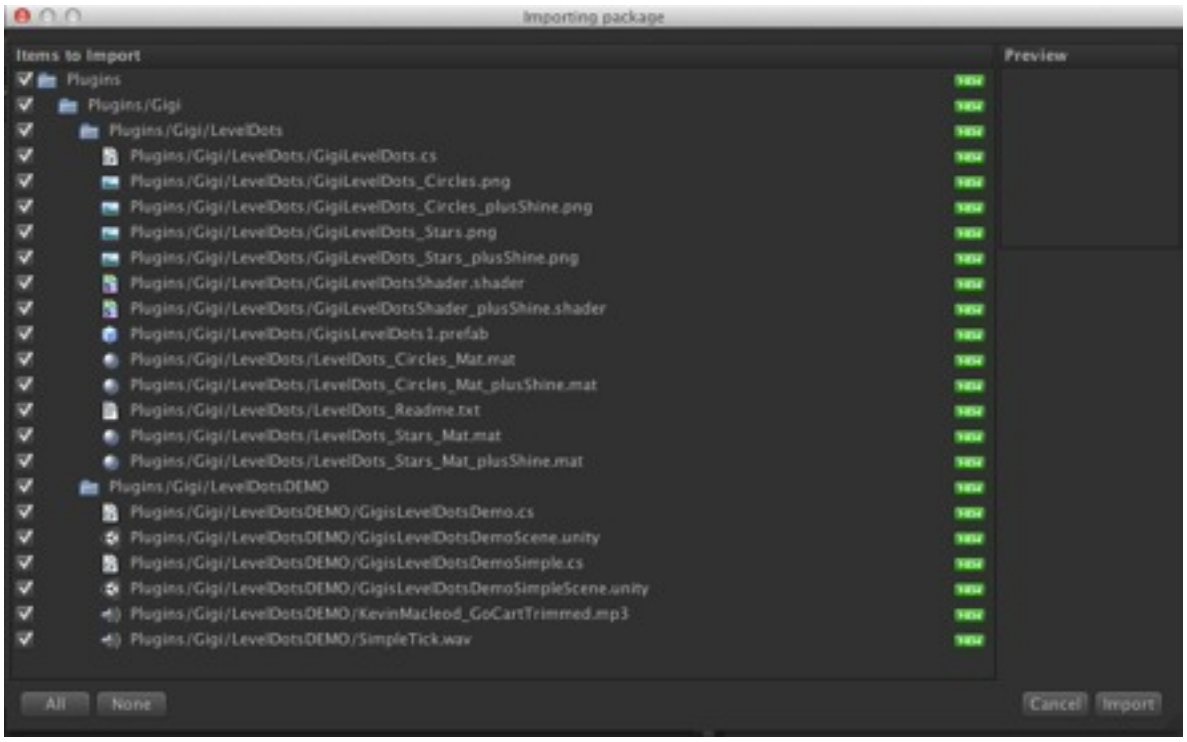
Easy, Elegant, Efficient.

WEB DEMO: http://www.goodgamesbydesign.com/Files/GigisLevelDotsDemo.html

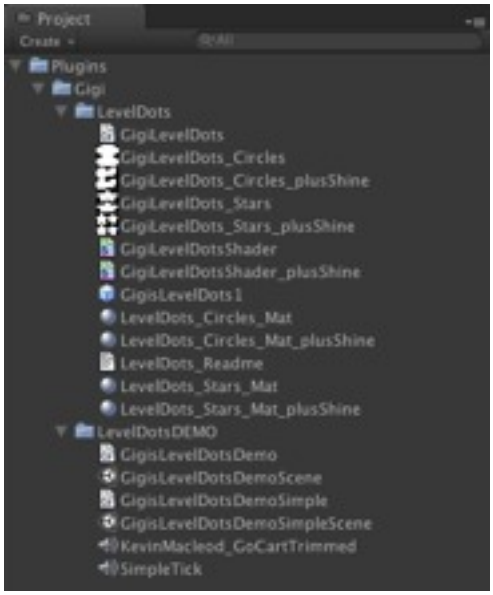
LevelDots is easy to use, adds a flare of elegance, and runs efficiently on all Unity platforms, including mobile devices. In a nutshell, you simply add the prefab to your scene; configure it as desired; and call UpdateState() whenever your level changes. Here's the entire sequence of steps, in detail.

Step 1: Import the Package

It's safest to try asset packages in an empty project. So, create a new project and then select Download and Import from the Asset Store. Alternately, if you have stored the unitypackage file on your hard drive, you can use [Assets | Import Package | Custom Package ...] from the menu. When you see this, press Import:

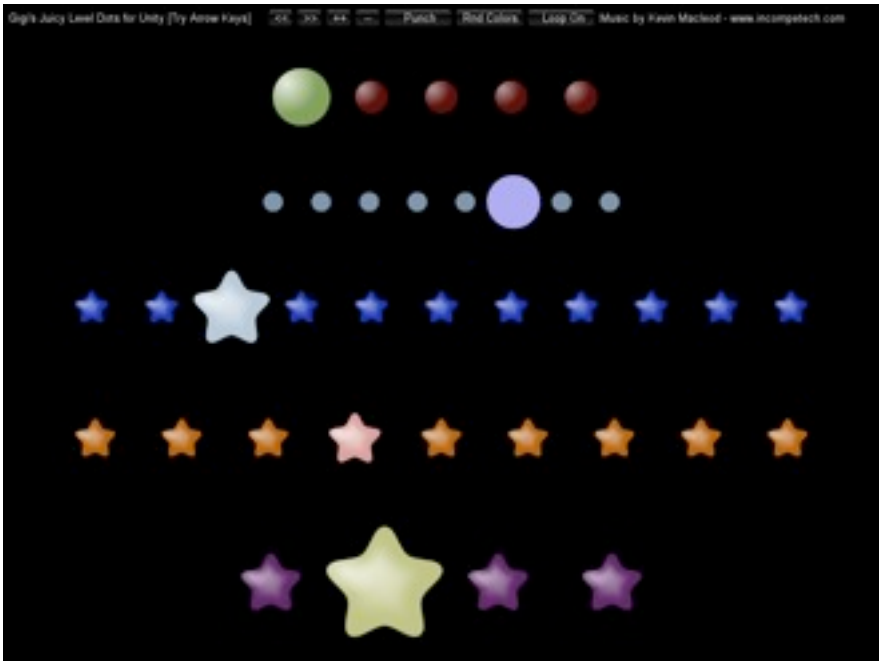


The import will add assets to your hierarchy:



The assets are separated into two sections: LevelDots, which includes the assets you will typically use in your own projects; and LevelDotsDEMO, which includes the sample code for the web demo as well as a Simple demo, which shows a trivial use of LevelDots.

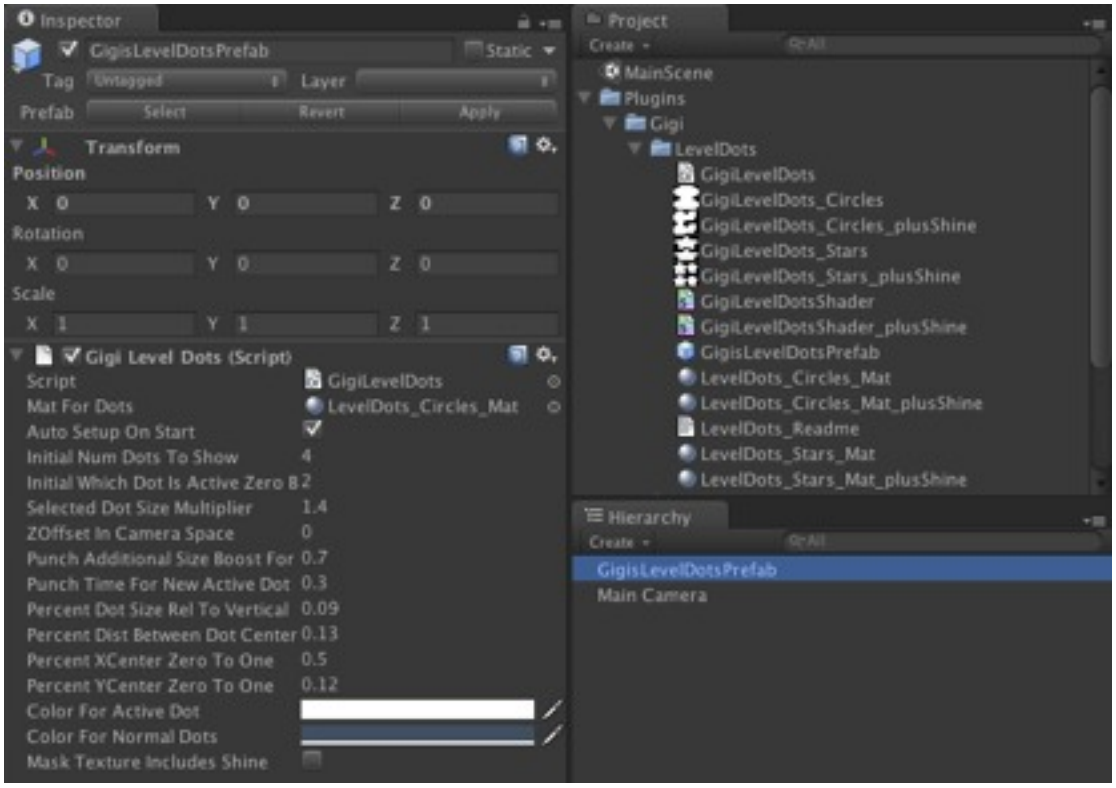
Double-click GigisLevelDotsDemoScene. When prompted, save your scene as 'MainScene' - so we can come back to it later. Once the DemoScene is open, run the app to see this:



You can refer to the demo scripts to see an advanced use of Level Dots including changing number, shape, materials, and timing sequences.

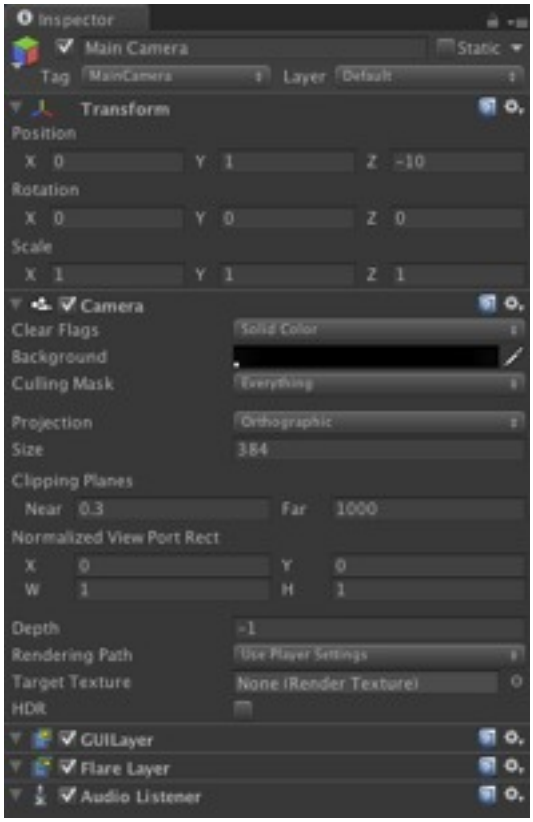
Step 2: Configure Your Scene

Re-open your "Main Scene" so we can add Level Dots from scratch. Drag 'GigisLevelDotsPrefab' into your hierarchy, like this:



Most Graphical User Interfaces (GUI) work with an orthographic camera, and Level Dots is no exception. To learn more about orthographic cameras, see the Unity camera tutorial here: http://unity3d.com/learn/tutorials/modules/beginner/graphics/cameras.

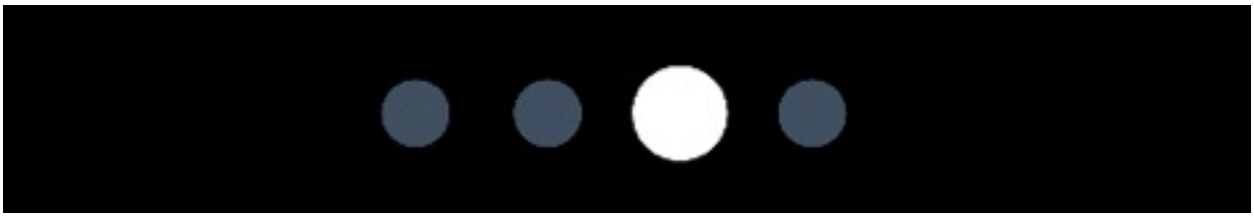
Configure your camera so that the settings look like this:



Advanced Note: In a fully developed game, you will likely have multiple cameras. When you do, you can make GUI development easier on yourself by keeping the orthographic camera at a 1:1 ratio of pixel size. Set the 'Size' property equal to half of the screen height in the editor, or in code like this:

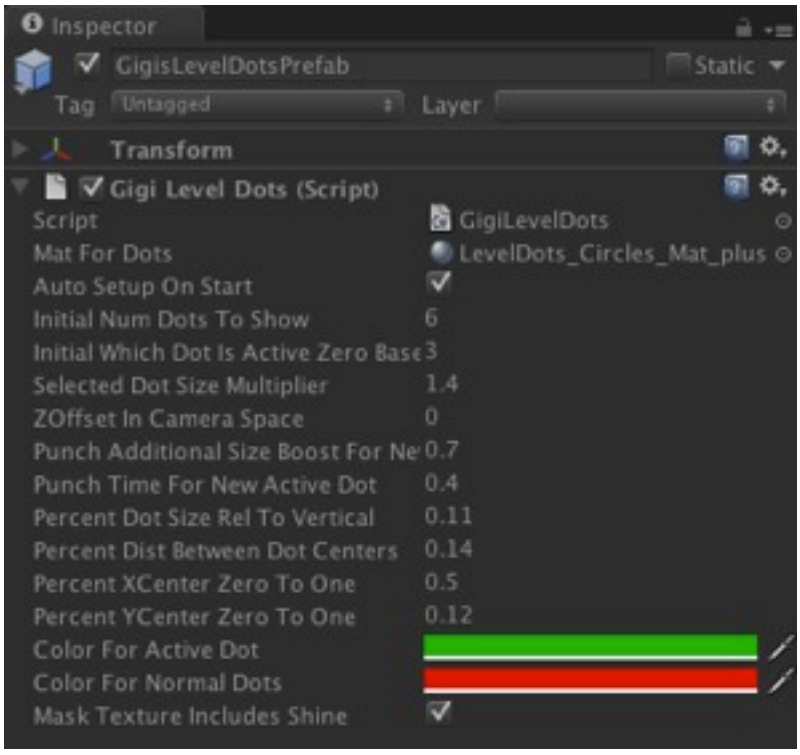
```
myCamera.orthographicSize = Screen.height / 2.0f;
```

Once you have the settings above, you can run the app, to see something like this:



Step 3: Juice It Up!

You're ready to Juice It Up! Create a Christmas theme by changing to the shiny material, setting the 'Mask Texture Includes Shine' to true, and changing the colors to red and green. Note that these changes should be made on the game object in your hierarchy, not the prefab. It'll look like this:



And now, when you run, you'll see this:

It's elegant with or without shine. Just be sure to toggle the 'Mast Texture Includes Shine' variable to match whichever material you're using. Here it is without the shine:

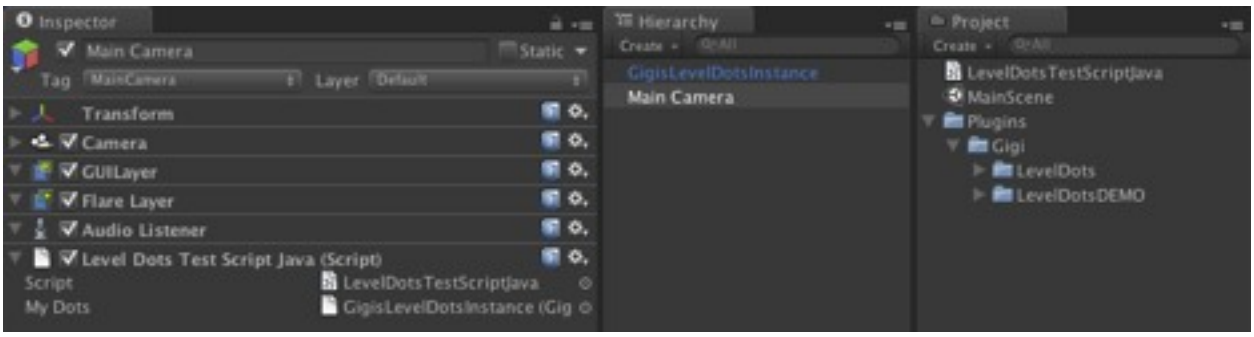
Step 4: You're First Script

To really bring LevelDots to life, you'll need at least one function call in a script. The easiest way to animate it at runtime, is to call UpdateState(), like this:

```
myDots.UpdateState (numLevels, currentLevel, true);
```

If you're familiar with scripting in Unity, skip ahead to Step 5.

From within the Project window, right-click in an empty area and create a C# or Javascript named 'LevelDotsTestScript.' NOTE - Do NOT create this file within the Plugins folder as it affects compilation order. Now, drag this new script and drop it on your camera to look like this:



Double click your script and enter code as follows, for Java:

```
var myDots : GigiLevelDots;
function Start () {
	myDots.UpdateState(4, 2, true);
}
```

Or for C#:

```
public class LevelDotsTestScriptC : MonoBehaviour {
	public GigiLevelDots myDots;
	void Start () {
		myDots.UpdateState (4, 2, true);
	}
}
```

Back in the editor, assign your GigisLevelDotsInstance game object the myDots variable on the camera. Be sure NOT to use the prefab from your Project folders - you can do it that way, but you'll need to use Instantiate. If you've done it correctly, then the level dots will show up, and you'll see the active dot punch in and out when you run the app.

Step 5: Going Further

Our one line *UpdateState*() was over too quickly. So, now, let's look at the rest of the API. First, let's look at the various parameters you can use to configure LevelDots. The defaults for these should be a good place to start for most games:

- *initialNumDotsToShow* - How many dots it'll show before calling *UpdateState*(). Once started, use *UpdateState*() to change the number of dots.
- *initialWhichDotIsActiveZeroBased* - Which dot is active prior to calling *UpdateState*(). Once started, use UpdateState() to change which dot is active.
- *matForDots* - Which material to use. Typically, this is one of the circle or star materials, with or without shine, depending on your preference. You may also create your own texture to use for this - see the Advanced section below.
- *autoSetupOnStart* - Causes LevelDots to call *Setup*() when it first enters the scene, which will cause it to appear, using the initial values above.
- *selectedDotSizeMultiplier* - Size of the active dot, relative to the normal dots. Ex. 2.0 would be twice the diameter, 1.0 would be the same.
- *punchAdditionalSizeBoostForNewActiveDot* - When the active dot changes, it will punch this much larger, for a short period. The value is cumulative, so a value of 1.5 for *selectedDotSizeMultiplier* and a value of 1.0 for *punchAdditionalSizeBoostForNewActiveDot* would result in a punch up to 2.5 times the diameter of a normal dot, that quickly shrinks back to 1.5x.
- *punchTimeForNewActiveDot* - How long the punch lasts, in seconds (ex 0.4 seconds)
- *zOffsetInCameraSpace* - an additional z amount that is added to the transform of the object the script is attached to in order to sort GUI elements (ex -9.0 or 0.0).
- *colorForActiveDot* - the RGBA color passed to the shader.
- *colorForNormalDots* - the RGBA color passed to the shader.
- *maskTextureIncludesShine* - modifies the UV coords to include the shine effect. Use in conjunction with matForDots.
- *percentDotSizeRelToVertical* - the base size of each dot, relative to the vertical (i.e. Screen.height). A value of 0.1 at 1024x768, would yield a dot diameter of 76.8 pixels, which is quite large. Most games will use values in the 0.2 to 0.5 range.
- *percentDistBetweenDotCenters* - similar to percentDotSizeRelToVertical, this defines how many pixels will be between the centers of each dot. Note that this is not affected by *selectedDotSizeMultiplier* or *punchAdditionalSizeBoostForNewActiveDot*, so leave enough room for the punch. Most games will use values in the 0.4 to 0.8 range.
- *percentXCenterZeroToOne* - The entire collection of dots is centered. So, this determines where that center point goes. 0.0 maps to the far left, 0.5 is the center of the screen, and 1.0 would be the far right. Note that a value of 0 or 1, will almost certainly push half the dots off the screen. The final transform is based off Screen.width. This value will often be 0.5.
- *percentYCenterZeroToOne* - The vertical center point from 0.0 (bottom of screen), 0.5 (middle), to 1.0 (top). The final transform is based off Screen.height.

Though many of these values will probably be set in the editor, you can set them in code anytime you like. After you're done with your changes, call UpdateState() or ForceUpdate(), so the geometry will be updated correctly

In addition to the parameters, Level Dots has a variety of methods you can use, including:

- *UpdateState(*numDots, newActiveDotZeroBased, isVisible*)* - The most common way of changing the state. Sets the number of visible dots, which one is active, and whether to make it visible. It's like calling *SetNumDotsToShow*(), *SetActiveDotZeroBased*(), and *SetVisible*(), except it only recreates the geometry once.
- *SelectNext()* - Advances the active dot by +1 (clamped at last dot). Recreates geometry.
- *SelectPrevious*() - Decrements the active dot by -1 (clamped at 0). Recreates geometry.
- *SetNumDotsToShow*(numDots) - Changes only the number of dots (one based, so 5 means 5). Recreates geometry.
- *SetActiveDotZeroBased*(newDotZeroBased) - Changes only which dot is active (zero based, so 1 means the 2nd dot). Recreates geometry.
- *SetVisible*(isVisible) - Use to hide or show the LevelDots. Does not recreate geometry.
- *ForceUpdate*(punchNow) - Recreates geometry, and allows you to restart a punch.

There are other public methods on GigisLevelDots, but you should almost never call them. They are made public as a convenience for advanced users.

To see a few of these behaviors in action, here's some C# snippets from the demo scripts. For additional examples, see GigisLevelDotsDemo and GigisLevelDotsDemoSimple.

```
public void RandomizeColors ()
{
	theDot.colorForActiveDot = new Color (Random.Range (0.6f, 1.0f),
		Random.Range (0.6f, 1.0f), Random.Range (0.6f, 1.0f), 1.0f);
	theDot.colorForNormalDots = new Color (
		Random.Range (0.2f, theDot.colorForActiveDot.r),
		Random.Range (0.2f, theDot.colorForActiveDot.g),
		Random.Range (0.2f, theDot.colorForActiveDot.b), 1.0f);
	theDot.ForceUpdate (true);
}

public void ChangeDotsStatus (int changeNumberOfDots, int changeSelectionNum)
{
	theDot.UpdateState (theDot.numDotsToShow + changeNumberOfDots,
		theDot.whichDotIsActiveZeroBased + changeSelectionNum, true);
}
```

These two are utility methods that are used by the following GUI code. The following snippets use these behaviors to increment/decrement the number of dots and which is active. Running the demo shows an image like the one seen earlier.

```
void OnGUI ()
{
	GUILayout.BeginArea (new Rect (5, 5, Screen.width - 10, 30));
	GUILayout.BeginHorizontal ();

	GUILayout.Label ("Gigi's Juicy Level Dots Demo!", GUILayout.Width (300));
	if (GUILayout.Button ("<<", GUILayout.Width (30)))
		ChangeDotsStatus (0, -1);
	if (GUILayout.Button (">>", GUILayout.Width (30)))
		ChangeDotsStatus (0, +1);
	if (GUILayout.Button ("++", GUILayout.Width (30)))
		ChangeDotsStatus (+1, 0);
	if (GUILayout.Button ("--", GUILayout.Width (30)))
		ChangeDotsStatus (-1, 0);
	if (GUILayout.Button ("Punch", GUILayout.Width (80)))
		PunchNow ();
	if (GUILayout.Button ("Rnd Colors", GUILayout.Width (80)))
		RandomizeColors ();

	GUILayout.EndHorizontal ();
	GUILayout.EndArea ();
}
```

This will show buttons along the top that allow you to increment/decrement the number of dots, which dot is active, and randomize the colors.

Step 6: Advanced

Below are some final advanced bits about LevelDots:

- Performance - Level Dots is highly efficient on most devices. All dots are drawn with a single draw call. The geometry is recreated when most significant parameters are changed changed, but the impact should be minor in most setups. The main hit on performance will likely be the number of pixels on the screen, which again should be nearly trivial, as the dots are usually quite small.
- Unusual Use - Level Dots is intended as a way to indicate a simple selection, among choices. However, by using a variety of level dots, in combination, you can achieve interesting effects like the demo. This can be further enhanced by changing the number, colors, sizes, and positions of the dots dynamically,
- Textures - The dot texture is broken into four quadrants: upper left (active dot mask), lower left (normal dot mask), upper right (active dot shine), and lower right (normal dot shine). Although the package includes circles and stars, with and without shine, you can create your own textures using packages such as Inkscape and Paint.Net to achieve a variety of effects and shines. Depending on your final implementation, you can shrink the provided texture to save memory.
- Unity 3.x, 4.x, Free, Pro - Level Dots should be usable by Free and Pro versions of unity and should run in both 3.x and 4.x. It was tested the best in v 3.5.6, Unity Pro. If you experience troubles with any other version, please post about it in the Unity forums.

CONTACT

The best way to contact me is via the Unity forum user: Gigiwoo2.

MUSIC

A final note about the music. The song in the demo called, "Go Cart", is © to Kevin Macleod of incompetech.com. It is used and released in this demo with his express permission. You can probably use it in your own games, but you should see his website for his actual license restrictions. Every Indie should do themselves a favor and visit his website - he has 100s of custom, professional songs - something for everyone.

Being juicy is a state of mind.

JUICE IT UP!