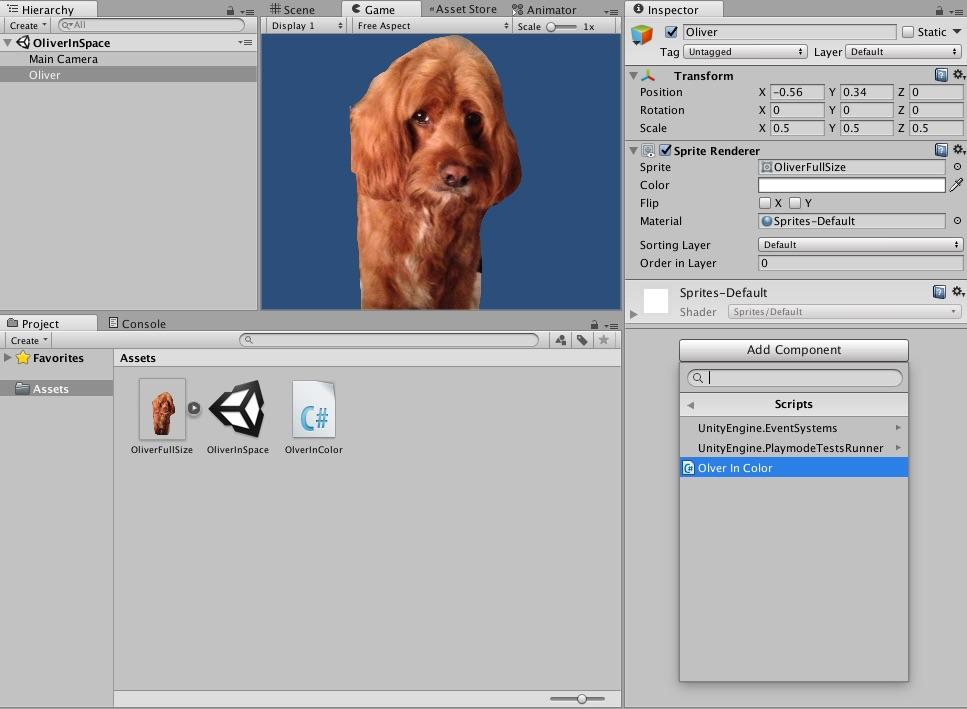
**Topic: Scripting Components**

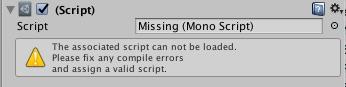
**Overview:** This tutorial shows you how to create scripts as components of a game object, and steps you through building an actual script. It demonstrates how objects can communicate with each other through public data of a class. Manipulating color is used as an example. Although you will immediately see how this could be applied to moving objects around, please hold off on that until the next lesson where the physics engine is introduced.

More thorough treatment of these ideas can be found in the [Unity3D tutorials on Scripting](https://unity3d.com/learn/tutorials/topics/scripting), Beginner lessons 1,2, and 7. For those with little or no Java experience please also study the beginner lessons between 2 and 7.

A word of **caution** about manipulating 2D texture. It is more complicated than suggested here. Please follow these steps and refrain from going too far off task. Color is used here to provide a vivid example of manipulating data in an object and sharing data between objects.

**Your Turn (Please do the following):**

1. Open the project OliverColorBook found in the Sample Projects folder.
2. Using your OS, locate the script OliverInColor in the SampleScripts folder. Drag it into the Assets panel.
3. Double click on it to view it in the MonoDevelop-Unity editor. Although you can set preferences to point to another IDE, (integrated development environment such as Sublime,) it is recommended that you start with MonoDevelop[[1]](#footnote-1). Read the comments to understand what the code does. (This technique will be used often in the future).
4. Select the game object *Oliver* in the Hierarchy. Click on *Add Component,* and select *Scripts.* The *Oliver In Color* script should appear. Note that Unity3D will only show scripts that compiled properly (e.g. no syntax errors in your script.) If you already edited the script and it doesn’t appear, grab a copy from the sample projects folder.



1. Run the project. Oliver should change colors. Note that if you inadvertently delete, or rename or otherwise break the script, Unity3D will give you a warning in Oliver’s inspector window. WARNING: You must stop, not merely pause the Game (all buttons off) to make permanent changes. You can modify values in the Inspector while the game is running, but those changes will disappear when you stop the Game. The Inspector is a very sophisticated interface into the project code! But it takes practice to remember to stop the Game to make permanent changes.

It is also helpful to save your project now and then. In particular, when you change scenes you will be asked to save the scene you are about to leave.

1. Now add another sprite to the project such as the tennis ball. Create a new script for the ball by clicking on *Add Component* in its inspector*.* Scroll down to the bottom and select New Script. Name the script *Color Ball.* Create a script that will use keys to change the color of the ball from yellow, its current color, to red and back again. You will need to set the tint to ‘white’ to get back to the original ball. Hint: cut and paste the code from Oliver’s script. That’s fair and practical because it reduces the likelihood of typos. Also play with the syntax prompt. For example, if you type ‘Color.’ It will prompt you with proper completions. Press tab to select the one you want. Note that you also get a definition of the class, method, data.
2. It is useful to allow game objects to share information through *public variables.* Remove the OliverInColor script, and replace it with *OliverShares* from the SampleScripts folder. Remove your script from the tennis ball, and replace it with *BallGetsColor.* Study the scripts carefully, and note in particular which data in the *OliverShares* can be viewed from the tennis ball. Explore how and where you can manipulate data. Try changing values in code and in the Inspector.
3. A good exercise is to have Oliver know about the ball just like the ball knows about Oliver. Modify OliverShares so that Oliver can grab the current color of the ball.
4. There are a lot of coding tidbits in these scripts. If they aren’t obvious refer to the Unity3D documentation. Studying documentation is a great way to understand what is going on. In the OliverShares example, the color is set using the form new Color(), which instantiates a new color. See the [Unity reference on color](https://docs.unity3d.com/ScriptReference/Color.html) for more on manipulating color. Like all graphics systems, Unity3D uses a color model that has four components in this order: red, green, blue, alpha. All potential colors are mixed from these. The first three are self explanatory, the fourth is the opacity. Colors are recorded in percentages for 100% (1) to 0% (0). If you want some, but not all of a color component, you must use a floating point. For example 0.5F is 50% and represents the fraction 5/10.

**Summary:** A lot of Unity3D specific coding was introduced here that assumes you have some knowledge of Java or C#. The emphasis was on referencing data and methods in both code and the inspector. Please (please) realize that this is the tip of the iceberg. In particular the GetComponent method was exploited even though it is expensive to invoke (e.g. it takes time.) There will be more to say about this in subsequent tutorials. You were also expected to learn by example – for example you should have an idea of how to get information from a keyboard. It can’t be sufficiently emphasized that doing this in Update is a cheap trick, and that callbacks in a later lesson provide a move elegant and efficient way to do this. Any time you find yourself writing code that is an involved sequence of selection (if/else) statements, realize that you can do better.

1. Don’t worry if Monodevelop doesn’t immediately open; it can take a little while to start up (5-10 minutes). Unity may also stop responding during this time and you may see an error message/loading bar which references visual studio. [↑](#footnote-ref-1)