Alyssa Burdeos
Intro to Scripting

Conventions and Syntax

C# is a coding language that we are using in class to code our scenes in Unity. Like any other language, it has its own conventions and syntax. Using the correct conventions and syntax is vital to the function of the scene, as one small error could result in the entire game not working properly, or even at all.

So far, we have learned about basic conventions and syntax when using C#. These include how to make classes and how to use variables in each class. We learned that a class is similar to a blueprint. It provides important features about an object or a function that will be implemented in the scene. The variables are the attributes that are contained in the class.

When using classes in code, we begin by stating the class, then making a container for its variables by using opening and closing curly brackets ({}). The variables are the details that the object will have or functions that it will perform.

An example of using classes and variables is seen below:

```
4  public class changeColor : MonoBehaviour {
5          // Update is called once per frame
6          void Update () {
7                  if (Input.GetKeyDown (KeyCode.R)) {
8                          GetComponent<Renderer> ().material.color = Color.red;
9                  }
10                 if (Input.GetKeyDown (KeyCode.G)) {
11                         GetComponent<Renderer> ().material.color = Color.green;
12                 }
13                 if (Input.GetKeyDown (KeyCode.B)) {
14                         GetComponent<Renderer> ().material.color = Color.blue;
15                 }
16                 if (Input.GetKeyDown (KeyCode.Space)) {
17                         GetComponent<Renderer> ().material.color = Color.black;
18                 }
19                 if (Input.GetKeyDown (KeyCode.Delete)) {
20                         GetComponent<Renderer> ().material.color = Color.white;
21                 }
22                 if (Input.GetKeyDown (KeyCode.Escape)) {
23                         GetComponent<Renderer> ().material.color = Color.yellow;
24                 }
25
26         }
27
28 }
```

As seen in our example project, the overall class is identified as “changeColor”, which is what this particular script was written for (changing the color of an object in the scene). The name of the class itself does not need to be included in the curly brackets. However, all of the attributes within the class are contained inside one set of curly brackets. The “void Update” text indicates another class, which updates the variables contained in it once per frame.

The variables (written as “if” statements) instruct the program of what to do when a particular event happens. In the example provided, each variable states that when a particular key is pressed on the keyboard, then a certain action will take place. As shown on line 10 of the example, if the “G” key is pressed on the keyboard, then the object in the scene will change to the color green. It is also important to note that each variable ends in a semi-colon (;), which indicates the end of the statement.

On line 5 of the example script, a statement is italicized using slashes (//). These slashes indicate a comment in the code. Comments are text placed inside the script that do not affect the actual scene. Rather, they are for the coder to make notes regarding the code that has been written. In the example script, the comment states “*//Update is called once per frame”*. This comment lets the coder know that each variable listed below it is updated once per frame in the scene.

Another aspect to note is that periods are used to narrow down a specific attribute. For example, in line 16 of the example, it states:

```
16                  if (Input.GetKeyDown (KeyCode.Space)) {
```

The code inside the parentheses includes information about which keyboard command will be used for the function. The information is broken up in a funnel fashion, going from the most broad detail to the most narrow. First, we state that we need an input. We then state that the input we need must come from pressing a key down. Then, we state that we need a particular key, and finally, we state that the key must be the space key.

Lastly, it is important to maintain organization while coding. Doing so makes it much easier for the coder to find and resolve bugs in the code. Organizing code using indentations does not affect the functionality of the script, but it does make it significantly easier to read.

These are just a few of the basics that we have learned about so far regarding conventions and syntax in C# coding.