Exercise 11: Growing Teddies

Although this exercise isn't worth any points, it gives you valuable programming experience. You're almost definitely going to have to complete the exercises to succeed in the course.

Clone your repository

1. Accept the assignment to create your repository for submitting your work:

   https://classroom.github.com/a/zoisH-sO

2. In GitHub Desktop, clone the repository you just created to your desktop.
3. Open the **Exercise11** project in Unity.

   Note: When you open the provided Unity project, it will probably start in an Untitled scene. Double-click **Scene0** in the Scenes folder in the Project window to open the correct scene.

Problem 1 - Make yellow teddy bear 4 times as large

First, you'll change the game, so the yellow teddy bear is drawn four times as large. You'll make these changes in code, not in the editor

4. Right click in the Project window and create a new folder named Scripts.
5. Right click the Scripts folder in the Project window and select Create > C# script. Rename the new script YellowTeddyBear.
6. Double click the script in the Project window to open it in Visual Studio and add a documentation comment for the class.

   Caution: Because we named the new script YellowTeddyBear when we created it, the name of the class that's created (the name right after **`public class`** in the script) is also **`YellowTeddyBear`**, which is exactly correct. Unity requires that the name of our .cs file and the name of the class in that .cs file match exactly. This is good programming practice and is perfectly reasonable.

   Unfortunately, if you leave the script name **`NewBehaviourScript`** (the default script name) when you create the script and then rename the script to the name you really want later in the Unity editor, Unity changes the name of the .cs file but leaves the name of the class in that file NewBehaviourScript instead of changing it to the new name. The script will compile fine in Visual Studio, but Unity will give you an error when you try to attach it to a game object in your scene.

   To fix this, right click on the **`NewBehaviourScript`** class name in the script and select Rename... (the second choice from the top) in the popup menu. Type in the name of the script as the class name and press <Enter> or the Apply button in the renaming dialog. Now the name of class and the name of the .cs file match, so Unity will let you attach the script to the game object in your scene (we'll do that in Step 8 below).

7. Delete the **`Update`** method, including the comment above it and all its curly braces. We don't need that method for our solution.

8. To change the location and scale of the game object the script is attached to, we need to modify the Transform component of the game object. If you look at the documentation for the Transform class in the Unity Scripting Reference, you'll see that it has a **localScale** property we can access. Save a copy of that using the following code (don't put a blank line between the comment and the code):

```
// quadruple width and height
Vector3 newScale = transform.localScale;
```

9. Now that we have a copy, we can make the width and height four times as large using the following (don't put a blank line between the two lines of code):

```
newScale.x *= 4;
newScale.y *= 4;
```

   Note: Using **newScale.x *= 4;** is the same as using **newScale.x = newScale.x * 4;.** You'll see lots of programmers using the ***=** (and **+=**, **-=**, and **/=**) shorthand.

10. Finally, we need to change the actual local scale of the game object using:

```
transform.localScale = newScale;
```

    You might have thought that we could simply change the x and y properties of the **transform.localScale** property directly instead of copying it into **newScale**, changing **newScale**, then copying **newScale** back into **transform.localScale**, but that doesn't work because of the rules about how C# properties work. You'll end up using this copy-change-copy back approach regularly in your Unity scripts, especially when working with the **Transform** class.

11. Drag the script from the Project window onto the YellowTeddyBear game object in the Hierarchy window to attach the script to that game object as a component. Left click the YellowTeddyBear game object in the Hierarchy window and look at the Inspector to confirm the script has been added.

12. Run the game to confirm your code works properly.

13. In GitHub Desktop, commit your changes with the message: “Completed problem 1”.

Problem 2 - Make the green teddy bear 3 times as tall

14. Next you'll change the game so the green teddy bear is drawn at normal width but three times as tall. Use Steps 5-12 above as a guide to implement the required change.

15. In GitHub Desktop, commit your changes with the message: “Completed problem 2”.

Problem 3 - Make the purple teddy bear 3 times as wide

16. Finally, you'll change the game so the purple teddy bear is drawn three times as wide but normal height. Use Steps 5-12 above as a guide to implement the required change.

17. In GitHub Desktop, commit your changes with the message: “Completed problem 3”. Push your changes to the remote.

18. Return to CodeHS and respond to the prompt.