**Lab 3: - Unity – C# Scripting**

**Understanding two C# data type: value type and reference type**

1. Start the Unity and create a new project, called Lab3\_CSharpScripting.
2. Create a gameObject ‘Sphere’, by positioning your mouse cursor in the Hierarchy Window, right-click and choose ‘Sphere’.
3. Create a folder under the “Assets” folder, called ‘Scripts’.
4. Create and attach a new ‘C# Script’ to the sphere – name it as ‘CSharpDemo’. In the Project Window, move this script to the folder ‘Scripts’ which you just created.
5. Double click the script “CSharpDemo” to open it in the Visual Studio (or VS Code, or other tools). The script will look like below.

using UnityEngine;

using System.Collections;

public class CSharpDemo : MonoBehaviour {

// Use this for initialization

void Start () { }

// Update is called once per frame

void Update () { }

}

Review the lecture slides and **answer the** **two questions**: (1) what’s the operator ‘using’ used for? (2) which class does the class ‘MonoBehaviour’ inherit? What are the new/overridden members (attributes and functions) defined? (Task 3.1)

1. Next, we will define new methods for the class ‘CSharpDemo’ to illustrate the difference between the value type of data and reference type of data in C#. Reference the lecture slides for more details.
2. First, define the following method (within the class ‘CSharpDemo’):

void value\_DataType()  
    {  
        // ‘transform.position’ gives the position of the sphere  
        Debug.Log ("x = " + transform.position.x + "  y = "   
            + transform.position.y + "  z = " + transform.position.z);

Vector3 v1 = transform.position;  
        v1.x += 1f; v1.y += 1f; v1.z = +1f;  
  
        Debug.Log ("x = " + transform.position.x + "  y = "   
            + transform.position.y + "  z = " + transform.position.z);  
  
    }

1. Call the method value\_DataType() within ‘Start()’ as below:

public class CSharpDemo : MonoBehaviour {

// Use this for initialization

void Start () {

value\_DataType();

}

// Update is called once per frame

void Update () { }

void value\_DataType() {

…

}

}

1. Save the script and back to Unity editor for playtest. Answer the following questions: (1) What is the printout in the Console window? Does the sphere change its position? (2) Is the Vector3 a value type or reference type? When we amend ‘v1’, does the ‘transform.position’ change as well? Why? (Task 3.2)
2. Then define the following method (within the class ‘MonoBehaviour’):

Void reference\_DataType()  
    {  
        // ‘transform.position’ gives the position of the sphere  
        Debug.Log ("x = " + transform.position.x + "  y = "   
            + transform.position.y + "  z = " + transform.position.z);

Transform trans = transform;

// note that, here we change the ‘trans.position’, not ‘transform.position’  
        trans.position = trans.position + new Vector3(1f, 1f, 1f);  
  
        Debug.Log ("x = " + transform.position.x + "  y = "   
            + transform.position.y + "  z = " + transform.position.z);  
  
    }

1. Call the method reference\_DataType() within ‘Start()’ as below:

public class CSharpDemo : MonoBehaviour {

// Use this for initialization

void Start () {

// value\_DataType();

reference\_DataType();

}

// Update is called once per frame

void Update () { }

void value\_DataType() {

…

}

Void reference\_DataType() {

…

}

}

1. Save the script and back to Unity editor for playtest. What is the printout in the Console window? Does the sphere change its position? Why? Is the variable ‘trans’ with the type ‘Transform’ a value type or reference type (of data)? When we amend ‘trans’ (by changing its ‘position), does the ‘transform.position’ change as well? Analyze the code and explain. (Task 3.3)

**Understanding class hierarchy, accessibility of class members, foreach, the data structure ‘list’**

1. Define a new class called GameObject2D in the script ‘CSharpDemo.cs’ as below:

using UnityEngine;

using System.Collections;

class GameObject2D { string name; }

public class CSharpDemo : MonoBehaviour { … }

**Answer the questions**: what’s the accessibility of the attribute ‘name’? public, protected or private? (Task 3.4)

1. Add into the class ‘GameObject2D’ two public methods: setName(string) to assign a name to an instance of the class; getName() to return the name of the instance.
2. Define a new method in the class ‘CSharpDemo’, called ‘createGameObject2D()’, with the skeleton as below:

void createGameObject2D () {

// create 3 instances of the class ‘GameObject2D’

// and their names are Peter, David and John

// define a List (C# data structure, as introduced in the lecture)

// which holds 3 instances

// note that, if ‘List’ is not recognized by Visual Studio, you need to

// include another namespace, i.e. ‘using …’.

// using the loop structure, foreach, to print names of the 3 instances

// created earlier

}

Complete the method according to the steps given as comments above. (Task 3.5)

1. In the method Start() call the method above, i.e. CreatGameObject2D().
2. Save the script and back to Unity editor for playtest. You are expected to see the names of the three instances of GameObject2D printed out to the Console window.
3. Now, define a new class called ‘Rectangle’, which derives from the class ‘GameObject2D’:

class Rectange : GameObject2D { … }

Then, finish the following:

(1) Add two new private members into the class, *width* and *height*, whose type is ‘int’.

(2) Add a constructor with 3 parameters: “public Rectangle (string, int, int) { … }”. Complete this constructor by initializing the attributes ‘*width*’, ‘*height*’ and ‘*name*’. Note that the ‘name’ is defined in the base/super/parent class. (Task 3.6)

1. Define a new method in the class ‘CSharpDemo’, called ‘createRectangle()’. Similar to the step 15, but instead of creating 3 instances of the class ‘GameObject2D’, you are required to create 3 instances of the class ‘Rectangle’. And to do so, you need to use the constructor you defined in the previous step. Also, instead of using the data structure ‘***List***’, this time you are required to use ‘***ArrayList***’. See the lecture slides for more details.
2. In the method Start() call the method above, i.e. CreatRectangle().
3. Save the script and back to Unity editor for playtest. You are expected to see the names, width and height of the three instances of Rectangle printed out to the Console window. Attach the screenshot of the output. (Task 3.7)
4. If you still have time, in the method ‘reference\_DataType()’, create an instance of Rectangle with the reference, say ‘rec1’ with certain attributes (names, width, height), then assign it to another variable like ‘Rectangle rec2 = rec1’. Assign different attributes to rec2. Playtest to see if the attributes of rec1 has changed or not. This will help you improve your understanding the two types: value type and reference type.