**Lab 4: - Unity – Scripting**

**Aim of this lab session:**

To learn the basic and fundamental Unity scripts using C#. For those who are already very familiar with Unity scripting, you may want to explore some intermediate scripting techniques by following the official Unity3D tutorial <https://learn.unity.com/project/intermediate-gameplay-scripting> (such as polymorphism).

**Creation and attachment of C# scripts to GameObjects and basic transformation using keyboard**

1. Start the Unity and create a new project, called Lab4\_UnityScripting.
2. Create a ‘Plane’ and then create a ‘Cube’ and put it on the top of the ‘Plane’.
3. Create a folder under the “Assets” folder and then create a ‘C# Script’ (which is put inside the ‘Assets’ folder). Rename this script as “*CubeController*”. Note that it’s good to organize all your game assets by folders such as ‘Materials’, ‘Scripts’, etc.
4. Double click the “CubeController” icon will open it in the Visual Studio. The script will look like below. Recall what’s covered in the lecture such as the meaning of ‘using…’, the class inheritance by ‘:’, etc. Also note the comments for the method ‘Start()’ and ‘Update()’.

using UnityEngine;

using System.Collections;

public class CubeController : MonoBehaviour {

// Use this for initialization

void Start () { }

// Update is called once per frame

void Update () { }

}

1. Now modify the method ‘Update()’ as follows to move the cube horizontally (along the ‘x’ axis):

void Update () {

transform.Translate(1, 0, 0);

}

1. Attach the script to the object ‘Cube’ by dragging the icon of ‘CubeController’ to the ‘Cube’ in the ‘Hierachy’.
2. ‘Play’ the game and see what happens to the cube. Does the cube move much faster than what you expected?
3. The code above actually moves the cube by 1 meter per frame. To make the speed of movement more predictable, it’s usually better to move by per seconds. Using the ‘deltaTime’ will give us the effect. Modify the code as follows:

void Update () {

transform.Translate(1 \* Time.deltaTime, 0, 0);

}

1. Now, let’s rotate the cube. Add the following lines to the method ‘Update()’ above. Try one of the options below to see the different effects.

transform.Rotate(45, 45, 45);

transform.Rotate(45\* Time.deltaTime, 45\* Time.deltaTime, 45\* Time.deltaTime)

Before taking the next step, create another object, say a ‘Sphere’, attach it with another C# script and practice the steps 3-9 to familiarize yourselves with these fundamental steps. Note that you could attach the same script to different objects so that they all behave same.

1. Instead of letting the ‘Cube’ move by itself, we now control its movement or rotation using arrow keys. To do so, we add a following method to the script ‘CubeController’:

void ControlByArrowKeys()

{

float xMove = Input.GetAxis("Horizontal"); // move along the x direction

float zMove = Input.GetAxis("Vertical"); // move along the z direction

transform.Translate(xMove, 0.0f, zMove);

}

1. Then amend the ‘Update()’ method as follows:

void Update () {

ControlByArrowKeys();

}

1. Save the script and back to the Unity editor. ‘Play’ and try using the arrow keys to move the ‘Cube’. The speed of movement is too fast? Let’s control the speed by defining a variable called ‘speed’ and giving a parameter for the method ‘ControlByArrowKeys()’ as:

public class CubeController : MonoBehaviour {

private float speed;

// Use this for initialization

void Start () {

speed = 0.5;

}

// Update is called once per frame

void Update () {

ControlByArrowKeys(speed);

}

void ControlByArrowKeys(float speed)

{

float xMove = Input.GetAxis("Horizontal") \* speed;

float zMove = Input.GetAxis("Vertical") \* speed;

transform.Translate(xMove, 0.0f, zMove);

}

}

1. Try different ‘speed’ and see the effect. Note that we use the ‘speed’ for the name of both the class attribute (which is defined as ‘private float speed;’ and the local variable (which is in the method ‘ControlByArrowKeys(float speed)’, but the compiler show no errors.
2. We also could directly adjust the speed through the Unity editor. To enable us to do so, the variable ‘speed’ must be defined as ‘public’ as below (which is otherwise invisible/inaccessible to any class/object other than the class ‘CubeController’ itself).

public class CubeController : MonoBehaviour {

public float speed;

… …

}

1. Save the script and get back to the editor. You will notice the ‘speed’ now appears under the script in the ‘Inspector’ view. Assign a different value and ‘Play’ to see the effect. Note that you may also modify the value of the speed in the ‘Play’ mode! Try it!
2. Before going further, try one more thing here. Instead of arrow keys, try pressing the keys ‘A’, ‘S’, ‘D’ and ‘W’. What happens?

**Communication with other components attached to the object**

1. Add a ‘Rigidbody’ component to the ‘Cube’.
2. To access the Rigidbody component, we need to get the component’s reference. The primary way is to call the method ‘GetComponet<TypeOfObject>( )’ which is a template method. With the reference, you could get attributes or methods associated with the Rigidbody. To know more about it, simply type in the Visual Studio the word Rigidbody, move the cursor to it, then hold both ‘Ctrl’ and single quote ‘, which will bring you to a documentation page about ‘Rigidbody’ at: <http://docs.unity3d.com/ScriptReference/Rigidbody.html>. This works to other components as well.
3. Modify the scripts as follows (with all changes from the previous highlighted):

public class CubeController : MonoBehaviour {

//private float speed = 0.2f;

public float speed = 0.02f;

*private Rigidbody rb;*

// Use this for initialization

void Start () {

rb = GetComponent<Rigidbody> ();

}

// Update is called once per frame

void Update () {

//ControlByArrowKeys(speed);

}

void FixedUpdate() {

ControlByArrowKeys (speed);

}

void ControlByArrowKeys(float speed)

{

float xMove = Input.GetAxis("Horizontal") \* speed;

float zMove = Input.GetAxis("Vertical") \* speed;

rb.AddForce (xMove, 0.0f, zMove);

}

1. Note that we call the method ‘ControlByArrowKeys()’ in the method ‘FixedUpdate()’. Generally, the Update() is used for most operations, such as changing transforms, or rotating items, while you should use the method FixedUpdate() when you are calculating physics. More details on the difference of these two update methods can be found at Unity official website: <http://answers.unity3d.com/questions/10993/whats-the-difference-between-update-and-fixedupdat.html>.
2. Save the script and back to Unity to ‘Play’. Adjust the ‘speed’ for different effect. Then try calling the method ‘ControlByArrowKeys(speed)’ in the Update() and compare the result.

**Communication with other objects in the scene**

1. Add a ‘Sphere’ object into the scene.
2. Create a new script called ‘SphereController’ and attach it to the ‘Sphere’.
3. Amend the anatomy of the script as bellows. Read the comments in the codes to understand the reasoning behind for each line of codes.

public class SphereController : MonoBehaviour {

public float speed; // control the speed of translation

// a public reference/variable to the ‘Cube’ object in the scene

// being pubic allows it visible in the Unity editor

// so we could associate it with the ‘Cube’ object by drag & drop

public GameObject myCube;

void Start () {

speed = 5; // it’s always good to initialize the variable

}

void Update () {

ControlByArrowKeys(speed);

}

void ControlByArrowKeys(float speed)

{

float xMove = Input.GetAxis("Horizontal") \* speed;

float zMove = Input.GetAxis("Vertical") \* speed;

transform.Translate(xMove, 0.0f, zMove);

// to transform the ‘Cube’ in the same pace/speed

// as ‘this’ object, i.e. Sphere

// Note that the script is attached to the ‘Sphere’.

myCube.transform.Translate(xMove, 0.0f, zMove);

}

}

1. Save the script and back to the editor. Click/pick the ‘Sphere’ in the scene to show the public variable ‘My Cube’ in the ‘Inspector’. Move the cursor to the ‘Cube’ in the ‘Hierarchy’ and drag the ‘Cube’ to the field of ‘My Cube’.
2. Click the ‘Cube’ in the scene and deactivate the script ‘CubeController’ by clicking the small square box which is left to the ‘Cube Controller’.
3. ‘Play’ the game and use the arrow keys. You should expect the cube and sphere move together. Adjust the ‘Speed’ accordingly if necessary.
4. Activate the ‘CubeController’ script and see the effect.
5. Add more objects, scripts or amend the current scripts to practice this task.

**Print messages into the ‘Console’ for debugging**

1. A basic debugging technique is to print out some messages from the script. Add the following line to the method ‘void ControlByArrowKeys (float speed)’ as:

void ControlByArrowKeys(float speed)

{

float xMove = Input.GetAxis("Horizontal") \* speed;

float zMove = Input.GetAxis("Vertical") \* speed;

//transform.Translate(xMove, 0.0f, zMove);

rb.AddForce (xMove, 0.0f, zMove);

Debug.Log("xMove = " + xMove);

}

1. Save and ‘Play’. Click the tab ‘Console’ in the ‘Project’ view. What messages do you get? Then start playing by pressing arrow keys and notice change of the output.
2. Try amending the script and print other information, perhaps from different methods.

**Additional work to do (Bonus tasks)**

1. Use mouse to transform (move or rotate) the ‘Cube’.
2. Use other keys to increase or decrease the ‘speed’, instead of going through the editor.