Programming Tutorials

1) Character movement with WASD

Open Unity Hub and Create a new 3D Project, name it "Programming Tutorials".

Go to the top tab in unity, click Game *Object > 3D Object > Plane*

Then insert a cube by going to *Game Object > 3D Object > Cube*. This will be used for the player.

Select the cube and go to the inspector on the right-hand side. Go to the cubes Y axis in position and change it to 0.5. This will move the cube so that it is sitting on top of your plane.

Before starting our script, we need to add a Collider to our cube so that it doesn't fall through the plane below it when play is pressed. To do this, select your cube, go to the inspector on the right-hand side, select the 'Add Component' Button, search 'Box Collider' and apply it to your cube. In this case we do not need to add it to our plane as it spawns with a Mesh collider upon creation.

To create a C# script: right click on the assets folder, create > C# script and name this script **"PlayerMovement".** Then select your cube and attach the script to it in the inspector by using the 'Add Component' button, searching for your script name and selecting it. Double click this script to open it.

To move our cube all we need is a line of code in our Update method so that the movement can be updated every frame. To start our code, we want to use **transform.translate** as it allows the transformation of the object on the X, Y, and Z axis.

```
transform.Translate(0, 0, 0);
```

We also use *** Time.deltaTime** as this is a way of making sure the movement of the object is the same as the configuration that your game is being played on, and essentially means it will move your object by time rather than frame by frame that the update method uses, which stops your game running differently on different pieces of hardware.

1f > means it's a float and is used as a speed variable.

Your code should now look like this:

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

If you go back to the game and hit play you will see that the cube is now moving by itself on the Z axis. But now we want to get the cube moving by using the WASD or Arrow Keys. To do this, return back to your script and replace the X axis in your brackets to:

```
Input.GetAxis("Horizontal") * Time.deltaTime
```

So, your code should look like this:

```
transform.Translate(Input.GetAxis("Horizontal") * Time.deltaTime, 0f, 0f);
```

Save your code and return to the game and press play. Using the left and right arrow keys or A and D keys on the keyboard you will be able to move your cube on the X axis as you have taken your Input, defined it to an axis, in our case Horizontal and * it by **time.deltaTime.**

To create movement on the Z axis you do the same but change the axis to "Vertical" so that the vertical keys can be used. Your code should then look like this:

```
transform.Translate(Input.GetAxis("Horizontal")*Time.deltaTime,
0f ,Input.GetAxis("Vertical") * Time.deltaTime);
```

As of now our cube moves very slowly across the plane, and one thing you will want to do while creating movement is be able to change the speed of the player to the type of game you are making. Return to your script, and above **'void Start()'**, write out the variable:

```
public float moveSpeed;
```

We use **'public'** so that the variable shows in the object's inspector under the script properties and can be changed at any time without having to return to the script.

We use a float as this is what stores the number we set for the speed and allows it to use decimal points, unlike an integer which only uses whole numbers.

To get the movement speed working we need to apply it to our current movement code. Take the variable **'moveSpeed'** and place it at the beginning of each axis the cube is currently moving on and * it by the Input.

```
transform.Translate(moveSpeed * Input.GetAxis("Horizontal") * Time.deltaTime, 0f,
moveSpeed * Input.GetAxis("Vertical") * Time.deltaTime);
```

To change your cubes variable speed and test it, go to the cube's inspector and under the 'PlayerMovement' script properties, change the 'moveSpeed' number to 30 and test it.

2) Player jumping and ground detection

Select your already existing cube and in the inspector on the right-hand side, select the 'Add Component' Button and search 'Rigidbody' and apply it to your cube by selecting it. Make sure to not choose Rigidbody2D as we are working with 3D objects.

Create a C# script: right click on assets folder, create > C# script > name it **"PlayerJump".** Then select your cube and attach the script to it in the inspector by using the 'Add Component' button.

In the script, we first want to reference the **rigid body** that is on the cube. To do this we will write out the variable at the top of the script.

```
    public Rigidbody rb;
```

and in **'void Start()'** write out the code:

```
        rb = GetComponent<Rigidbody>();
```

This will reference the Rigidbody that is applied on your cube but within the code.

Now we want the cube to jump upwards in the air on key press to essentially create a player jump.

To do this, we need to start with an if statement that identifies if the jump key is being pressed or not. In **'void Update()'**, we start with the line:

```
if (Input.GetKeyDown("space"))

{
```

```
// Player jumps

}
```

This simply states that if the key that is being inputted is space (the spacebar key) then the player will do the executed.

To add a jump, we need to write a line of code that will make our player jump upon input. To do this we will write:

```
rb.AddForce(new Vector3(0, 5, 0), ForceMode.Impulse);
```

This code takes the Rigidbody and adds force to it on the Y axis. As you can see, we gave the Y axis a value of 5, meaning that when the player presses space, the cube will force itself upwards into the air. So now our jumping code should look like this:

```
if (Input.GetKeyDown("space"))
        {
            rb.AddForce(new Vector3(0, 5, 0), ForceMode.Impulse);
        }
```

If you hit play and jump you will notice that the cube can infinitely jump in the air without returning to the ground. To fix this we need to create a **Boolean** that will only let the player jump in the air when they are colliding with the ground.

To do this we need to return to our script and create a variable that is a Boolean at the top of our code. We use a Boolean as it allows our variable to have one of two possible values, true and false. To do this we need to write out the code:

```
public bool isGrounded = true;
```

Now we need to add our **'isGrounded'** variable to our if statement to check if the cube is on the floor or not, and also write a line of code that sets our 'isGrounded' variable to false when space is pressed and the player is in the air.

To do this we will include the code "&& isGrounded" to our already existing if statement, which will look like this:

```
if (Input.GetKeyDown("space") && isGrounded)
```

Then we will add a line of code that will set our variable to false when space has been pressed. This will be:

```
isGrounded = false;
```

Now, your if statement code should look like this:

```
        if (Input.GetKeyDown("space") && isGrounded)
        {
            rb.AddForce(new Vector3(0, 5, 0), ForceMode.Impulse);
            isGrounded = false;
        }
```

Now we need to use **OnCollisionEnter** so that we can check if the cube is colliding with the floor, and if it is then the cubes grounded variable is set back to true so the player can jump again.

To do this we will need to use **OnCollisionEnter** and inside of that use an if statement. Which will look like this:

```
private void OnCollisionEnter(Collision collision)
    {
        if(collision.gameObject.name == "Plane")
        {
```

```
            isGrounded = true;
        }
    }
```

This piece of code states that if the object that this script is attached to, in our case the cube collides with the object that is named Plane, which is our floor, then set the 'isGrounded' variable to true so that the player can jump again.

Return back to unity and play your game. Your player will now only be able to jump when on the ground and not infinitely in the air.

3) Player item pick-up on collision

Insert another cube by going to *Game Object > 3D Object > Cube* (this will be used for the pickup).

Select the cube and go to the inspector on the right-hand side. Go to the cubes Y axis in position and change it to 0.5. This will move the cube so that it is sitting on top of your plane. Then move your cube anywhere on the plane so it's not on top of the player.

Before we start writing our script for the pickup, we need to add a Collider to our pick-up cube so that it doesn't fall through the plane below it when Play is pressed. To do this, select your cube, go to the inspector on the right-hand side, select the 'Add Component' Button, search 'Box Collider' and apply it to your cube. We also need to ensure that the **'Is Trigger'** is checked to true in the box collider properties so that it can be picked up by the player.

So that we can reference our player object colliding with the cube in our script, we need to give the main cube the Tag of **'Player'**. To do this, select the cube that is your player, go to the inspector, click the Tag drop down box located at the top under the objects name and click the tag 'Player'.

Now we need to create a script to allow the player to collide with the white cube and result in it being picked up and disappearing. To do this, right click in your scripts folder > create > C# script > name this **"PickUp"** and double click to open it. Also attach this script to the cube you're trying to pick up in the inspector.

Once the script is open, we need to use **OnTriggerEnter** to identify if the player is colliding with the object that this script is attached to, in this case the cube that is being picked up, and if so, then visually show it has been collected by destroying it. To do this we need to create the void **'OnTriggerEnter'** and include and if statement that checks if the cube is being collided with or not.

```
    void OnTriggerEnter(Collider collider)
    {
        if (collider.gameObject.tag == "Player")
        {
            Destroy(gameObject);
        }
    }
```

Now if you return to your game, play it, and move your cube so it collides with the white pickup cube, it will destroy on collision.

4) Object drag with mouse

Create a script called **objectDrag** and attach it to the cube. Double click the script to open it.

In this script, we first use the void **OnMouseDrag().** This will be used to identify when the object is dragged with the mouse. We set the position of the offset to the position of the cursor. Offset is the difference of the position of the game object and the cursor in the 3d world. Our code will look like this:

```
private void OnMouseDrag()
    {
        transform.position = GetMouseWorldPos() + mouseOffset;
    }
```

We also need to create a variable for our offset at the top of our code. This will look like this:

```
private Vector3 mouseOffset;
```

Then create the void **OnMouseDown** to store the offset and to identify when the mouse button is being held down.

```
private void OnMouseDown()
```

In the void **OnMouseDown** we need to write code that gets the position of the cursor in the 3d world and not just where it is on screen. To do that we need the code:

```
mouseOffset = gameObject.transform.position – GetMouseWorldPos();
```

Next, we need to create a method for **GetMouseWorldPos**. This gets the mouse point from the mouseDown method and changes it from the pixel co-ordinate of where it is on the screen to where it is on the Z axis in the 3D world. To do this we need the code:

```
private Vector3 GetMouseWorldPos()
    {
        Vector3 mousePoint = Input.mousePosition;
        mousePoint.z = mouseZCoord;
    }
```

We use mousePoint.z = mouseZCoord to get the coordinate of the game object on screen.

Go to the top of your code and create a variable for **mouseZCoord** so it can store the game objects z coordinate. This variable should look like this:

```
private float mouseZCoord;
```

We also need to assign the mouseZCoord variable to our **OnMouseDown** method. The code will look like this:

```
  mouseZCoord = Camera.main.WorldToScreenPoint(gameObject.transform.position).z;
```

we use **Camera.main.WorldToScreenPoint** to use the world point of the games main camera that the mouse is being seen through and set it as mouseZCoord to set it as the mouses current 3D world position.

Then in the **GetMouseWorldPos()** method we return the screen points to world points, which will look like this:

```
return Camera.main.ScreenToWorldPoint(mousePoint);
```

Now save this script, return to your game, and play, using your mouse you will be able to pick up and move your object around your scene.

Tutorial References:

Unity – Simple Move a Cube with the Arrow Keys (Aaron Hibberd) - https://youtu.be/sXQI_0ILEW4

How to Jump in Unity – Unity3D Fundamentals (Deniz Simsek) –

https://youtu.be/xvLMD2qWaKk

Object Pickup with OnTriggerEnter – Unity 3D 2018 (Single Sapling Games) –

https://youtu.be/qD7fDop-Ptw

Unity Tutorial : Drag Gameobject with Mouse (Jayanam) –

https://youtu.be/0yHBDZHLRbQ