## 23/11/2020

This session I decided to work on the SceneTransition component.

## 03/11/2020

I have decided to work on the final component which is the SceneTransition component. This component is meant to transition you from one scene to another when you have completed the task of the first scene.

## 28/10/2020

There are three prominent bugs that needed to be resolved in the ShapeTrace code: the linerenderer scaling disproportionately, the corner cale count increasing as long as you hover over a corner, and even though you had let go of the mouse button and you dragged the mouse diagonally between two points a line would be drawn. Those bugs needed to be resolved before continuing.

I asked David for some help and he helped me to resolve the bugs and improve the code more. After resolving the size of the linerenderer, which was the default size we set for it in the code we reevaluated how the ShapeTrace should function. We decided that rather than a click and hold down mechanic it should be a mechanic where you click on a sequence of corners and the linerenderer render between each point once you have pressed on the corner. This change resolved the corner count and the diagonal linerenderer bugs.

Once we implemented that change we could continue the mechanic. The mechanic is named ShapeTrace but has another layer to it. Once you have finished the shape, it will wait a couple of seconds before disappearing and revealing the next shape to trace.

The ShapeTrace mechanic is finished and now bug free.

## 26/10/2020

It has been a little bit of a break/interruption. Today I catch myself up on what I have done in the past and subsequently continue on the Shape Trace code to try and refine it and solve certain bugs there are.

The issue with the Shape Trace seems to be the line renderer where the line looks disproportionate until you finish the shape.

When you let go of the left mouse button it seems like the code thinks that it’s still down and keeps adding corners and drawing lines between the corners.

## 19/11/2019

Today I wanted to focus on a jump and double jump mechanic.

### Before starting:

* Make sure you have an object you want to jump and a plain for it to jump on
* Add a Rigidbody to the jumping object
* Create a couple of variables and tags
  + int jump = 0;
  + public int doubleJump = 2;
  + private bool canJump;
  + private Rigidbody selfRigidbody;
  + Create tag in Unity for “Ground”
* We need some functions too
  + void FixedUpdate(){ }
  + void OnTriggerEnter(Collider other){ }

### Starting:

* In the “Start()” function:
  + selfRigidbody = GetComponent<Rigidbody>();
* In the “FixedUpdate()” function:

if (canJump)

{

jump += 1;

if (jump <= doubleJump)

{

selfRigidbody.AddForce(0, forceConst, 0, forceMode.Impulse);

}

canJump = false;

}

* In the “Update()” function:

if (Input.GetKeyUp(KeyCode.Space))

{

canJump = true;

}

* In the “OnTriggerEnter(Collider other)” function:

if (other.tag == “Ground”)

{

jump = 0;

canJump = false;

}

## 

## 22/10/2019

Focused on character rotation today. The character/player is supposed to be rotating or facing the object, in this case a wall, whenever you are moving the character forward pressing the “W” key.

## Tutorial:

### Before starting:

* Create the object that you want the player to face at the final stage of the rotation
* Make sure it has two colliders
  + One that is the size of the object and is NOT an “Is Trigger”
  + One that extends further out from the object to give it a perimeter of which the player can rotate when colliding and IS an “Is Trigger”

### Starting:

* Start off with writing the variable “Transform touching;” above the “void Start()”-function
  + (Make sure it is above the “Start”-function but underneath the first “curly bracket”/“{“
* Now within the “void Update()”-function write “if” and double tap the “tab”-key and now write “Input.GetKey(KeyCode.W) && touching != null” within the brackets that you created instead of where it says “true”
* We want to make sure that the code is triggered so we start with writing a debug line
  + “Debug.Log("Rotate");”
* Next we are going to write the actual rotation code allowing us to rotate 90 degrees so that we face the same way as the object we are colliding with on the z-axis
  + “touching.rotation = Quaternion.RotateTowards(touching.rotation, transform.rotation, 90.0f \* Time.deltaTime);”
* By now you should have a code within the “Update”-function looking something like this:
* Next we need two new kind of functions underneath the “Update”-function
  + “void OnTriggerEnter(Collider other)”
  + “void OnTriggerExit(Collider other)”
  + Make sure to make “curly brackets”/”{ }” after each new function
* Within the “OnTriggerEnter” we want to write another “if”-statement
  + “if (other.tag == “Player”)
  + Make sure to hit the “tab”-button twice so you get “if (true) { }” and just replace “true” within the brackets with what is needed
* Within the “if”-statement now between the “curly brackets”/”{ }” we need to specify what we want to happen
  + “touching = other.transform;”
* Now repeat the same thing within the “OnTriggerExit”-function but with a slight alteration
  + Instead of writing “touching = other.transform;” like in the “OnTriggerEnter”-function we write “touching = null;” because this means that the “Player” is not colliding with the object or “wall” so nothing should happen beside going forward when you hit the “W”-key

## 

## 15/10/2019

Decided to focus on simple character movement today in a 3D space.

First hour of the class I opened up a project I had made earlier though ran into some problems early on in the project. There were some package files that were not supposed to be there which did not allow the script to compile properly.

## Tutorial:

### Before starting:

* Make sure to have capital letters where needed or the script will not compile properly

### Starting:

* Start off with creating a ground and character asset
* Create a script and attach it to your designated character asset, it will be script for movement
* Open the character movement script
* Start off with writing a variable above the “Start”-section write down
  + public float speed;
* Write within the “Update”-section
  + float horizontal = Input.GetAxisRaw(“Horizontal”);
  + float vertical = Input.GetAxisRaw(“Vertical”);
  + Vector3 direction = new Vector3(horizontal, 0, vertical);
  + gameObject.transform.Translate(direction.normalized \* Time.deltaTime \* speed);
* Hit Ctrl+S and try it out in Unity