Extended Realities

Tutorials

MA Creative Technologies
10.11.2023

Image: Colour Fiction

Schedule

DAY 1

MORNING

- Introduction to realtime and XR
- Overview of technologies and available tools

LUNCH

AFTERNOON

- Introduction to the Unity game engine
- Game engine architecture

HOMEWORK: Install all necessary frameworks for XR development

DAY 2

MORNING

- Setting up a scene for XR development
- Basic Unity C# programming patterns

LUNCH

AFTERNOON

- Topics in VR and AR
- XR design criteria and challenges
- Thinking about personal projects

HOMEWORK: Storyboard/Wireframe/Design document for XR experience

DAY 3

MORNING

- Prototyping personal projects

HOMEWORK: Continue working on the prototype, iterating over wireframe model

DAY 4

MORNING

- Finishing up on personal projects and testing others
- Evaluating the process
- Directions for further learning and research

Anatomy of a Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EventHandler : MonoBehaviour
{
    void Start()
    {

    }

    void Update()
    {

    }
}
```

Namespaces: These are essentially libraries of premade functions that can be used in your code. They give you access to quite a few features of the Unity engine without you having to code them yourself.

Class: Think of this as the name of your script. Think hard about how you name your script when you make it because it is very difficult to change later...

Start function: Fires once when the script is loaded at the beginning of the program run time. Everything here is loaded before the first frame renders.

Update function: Fires every single frame. If your experience is running at 60 frames per second, this will fire 60 times per second. Therefore try not to overload this function!

Anatomy of a Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EventHandler : MonoBehaviour
{
    void Start()
    {

    }

    void Update()
    {
      if (Input.GetKeyDown(KeyCode.Space))
      {
        Debug.Log("A button pressed");
      }
    }
}
```

Event Listener

Event: *Log event to console*

Anatomy of a Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EventHandler : MonoBehaviour
{
      public GameObject canvas;

      private Text displayText;

      void Start()
      {
         displayText = canvas.GetComponent<Text>();
      }

      void Update()
      {
         if (Input.GetKeyDown(KeyCode.Space))
         {
            displayText.text = "A button pressed";
         }
      }
}
```

Public variables: *can be changed by other scripts; visible in Unity editor interface.*

Private variables: *cannot be changed by other scripts; can only be accessed here.*

displayText is set in the "Start" function - the first thing the script will do - so that the component is ready to be altered in the "Update" function.

Here, the text field of displayText is now being written every time it detects a press of the space bar.

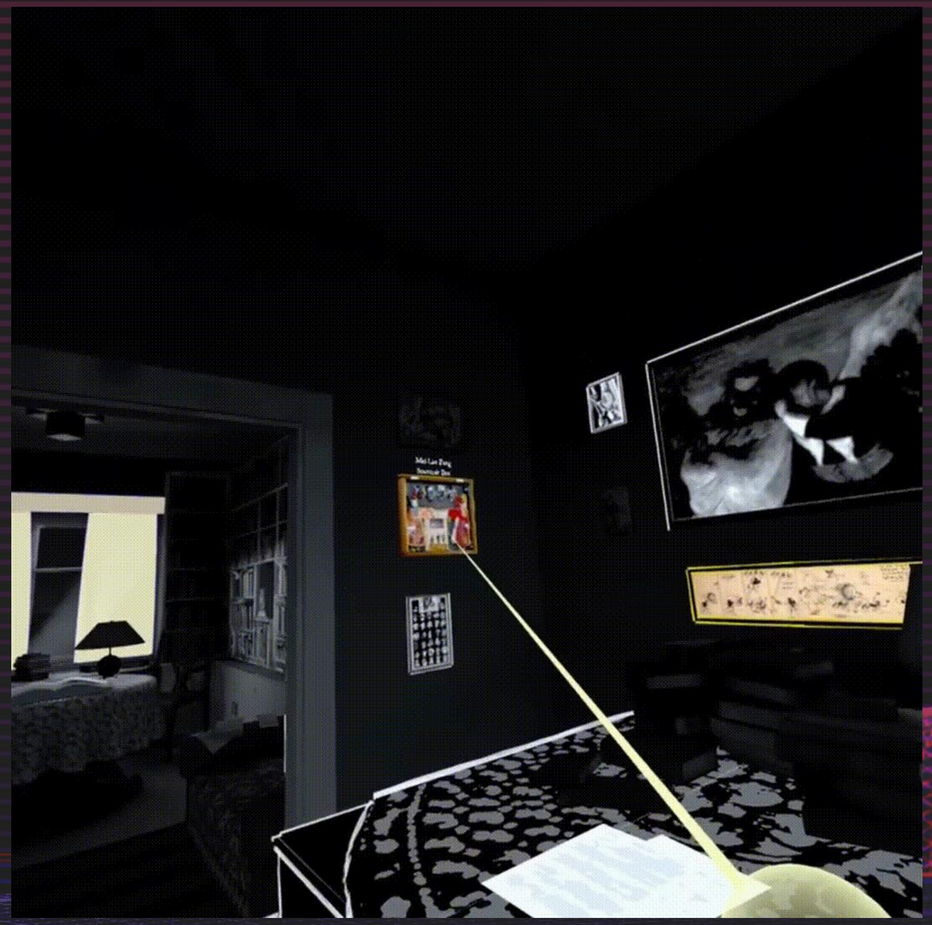
Anatomy of a Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SpawnSpeakingObject : MonoBehaviour
{
    [SerializeField] public SpeakingObjectsRay speakingObjectsRay;
    [System.NonSerialized] public GameObject speakingObject;
    private GameObject spawnedObject;
    private GameObject objectInRoom;
    private Camera mainCamera;
    private Animator descendAnimation;

    void Start()
    {
        mainCamera = Camera.main;
        descendAnimation = gameObject.GetComponent<Animator>();
    }
```

Anatomy of a Script

```
void Update()
    {
        if (speakingObjectsRay.hitObject.layer == 10)
        speakingObject = speakingObjectsRay.hitObject;
        if(spawnedObject != null)
        {
            spawnedObject.transform.LookAt(mainCamera.transform);
        }
    }
```

Anatomy of a Script



```
public void SpawnObject()
    {
        spawnedObject = Instantiate(speakingObject, transform.position, transform.rotation,
this.gameObject.transform);
        objectInRoom = speakingObject;

        float size;
        if(spawnedObject.GetComponent<Renderer>().bounds.size.x <
spawnedObject.GetComponent<Renderer>().bounds.size.y)
        {
            size = spawnedObject.GetComponent<Renderer>().bounds.size.y;
        } else
        {
            size = spawnedObject.GetComponent<Renderer>().bounds.size.x;
        }
        Vector3 rescale = spawnedObject.transform.localScale;
        rescale = 0.2f * rescale / size;
        spawnedObject.transform.localScale = rescale;

        foreach (Transform child in spawnedObject.transform)
        {
            Destroy(child.gameObject);
        }

        descendAnimation.Play("Base Layer.SpawnObjectAnimation", 0, 0);
        OVRInput.SetControllerVibration(0.1f, 0.1f, OVRInput.Controller.RTouch);
        objectInRoom.SetActive(false);
    }
```

Anatomy of a Script

```
public void DestroyObject()
    {
        Destroy(spawnedObject);
        objectInRoom.SetActive(true);
    }
}
```

Anatomy of a Script

Add a point light to your scene.

Create a new C# scripted component for your light. The script should turn the light on and off at the click of the spacebar.

1. Soft code the script - write in your own words in the C# document what other objects you will need to access, what you need to put in your "Start" function, and what you will need to put in your "Update" function.

2. Have a go at programming the script.

Probes + Lighting