Virtuality

WS 2 – Building your 1st VR Application

2022

VinMaker

VR Lab

Agenda

01

Introduction to VR

- The 3Is framework
- VR application
- Safety requirements

(10 minutes)

02

Building VR Application

- 1st development section
- 10’ Tea Break
- 2nd development section

(65 minutes)

03

VR Experience

- Further exploration
- VR Experience

(15 minutes)

Pre-setup!

Divide into groups

Group of 3 people

- 1 laptop with Unity installation
- 1 headset for testing and experiencing
- should have 1 CECS student in each group to assist development

What is VR?

- “A **simulated** experience in which **computer graphics** is used to create a **realistic-looking** world” [1]
- “The use of **computer technology** to create the effect of an **interactive 3D world** in which the objects have a sense of **spatial presence**” [2]

“With appropriate programming, such a display could literally be the Wonderland into which Alice walked.”

Sutherland, 1965 - “Father” of VR

A scene from the famous movie Ready Player One

How to understand VR?
What are its key characteristics?

There are many different answers,
and we will look into one of them - **the 3Is framework**

The 3Is of Virtual Reality[3]

Immersion

Within immersion, a person may **feel inclusive** into the virtual environment and **connected between perception and the virtual interface**.

Interaction

A VR system can **detect user's gestures** via multiple sensors and provide **real-time response** to the new activity instantaneously.

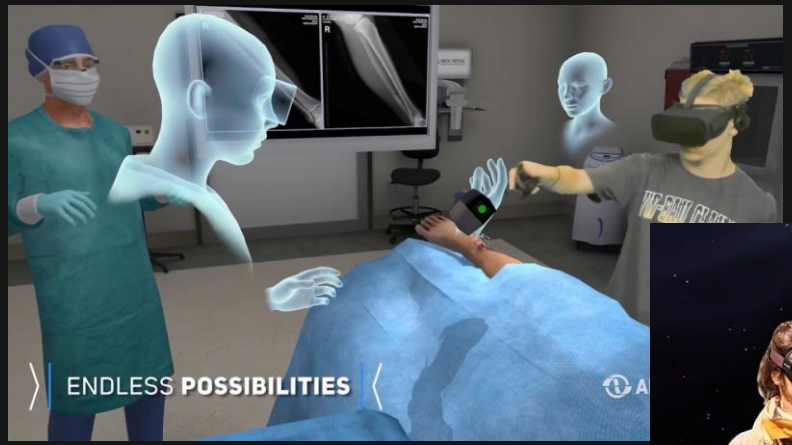
Imagination

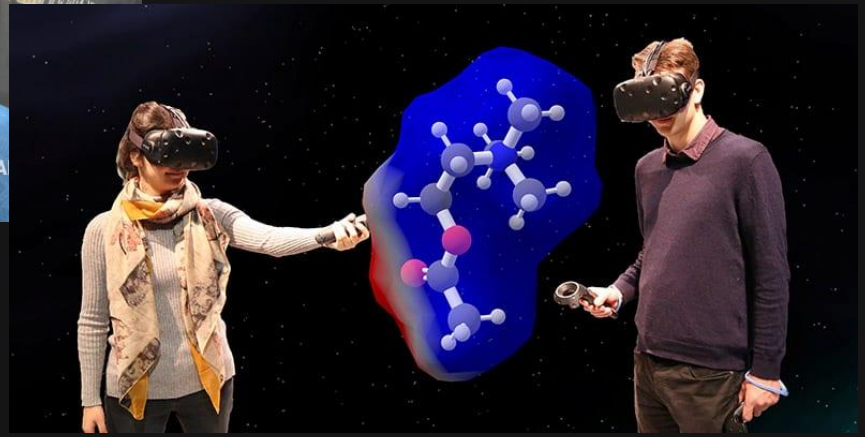
VR virtual environment supports the user to **elaborate on thoughts with virtual, imaginary objects**.

Art & Media

Science and Engineering

Healthcare, Biology & Chemistry

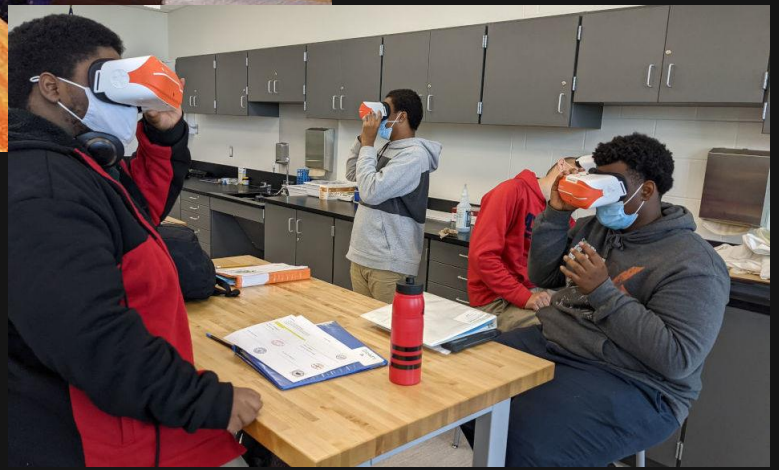




Education and Office

Virtual Speech

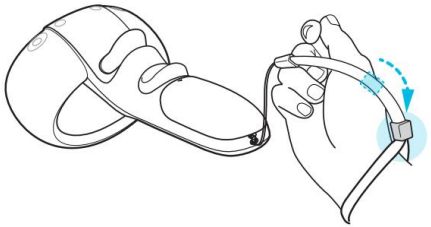
ClassVR



Usage guidelines

Putting on the controller

- Place the lanyard on your wrist and tighten it comfortably

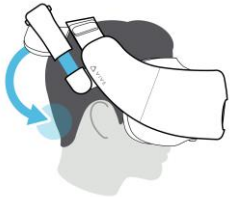


2022

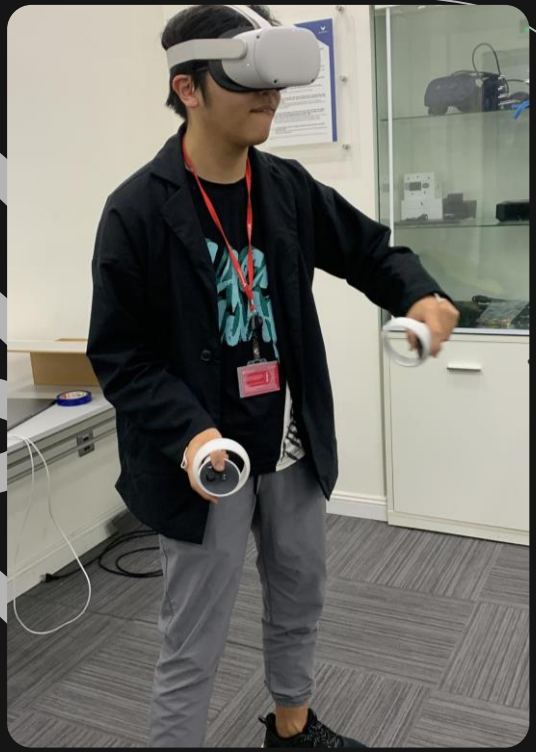
Usage guidelines

Putting on the headset

- Loosen the straps on top and at the back
- Put the headset on and tighten the straps until your preferred fit



2022



Coding time!

What is in the project?

Oculus Integration package

Oculus SDK for Unity and necessary settings

Preconfigured settings

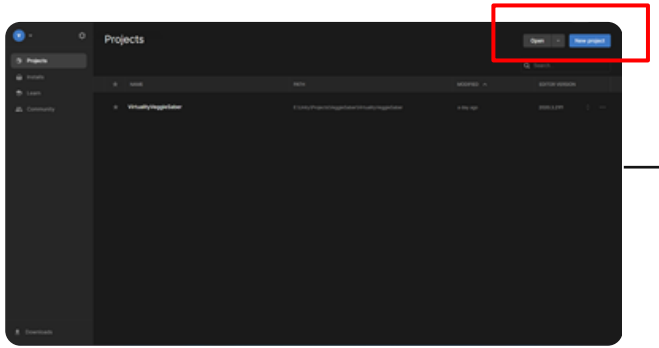
Remove redundancy and improve build time

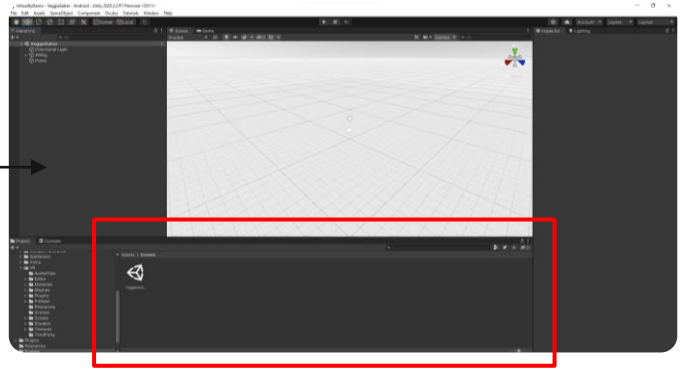
Helpers assets

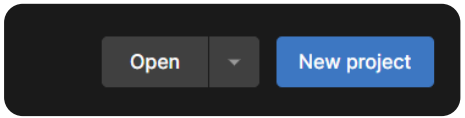
Speed up the development process

Download the template from: bit.ly/virtuality_materials

Open project & scene





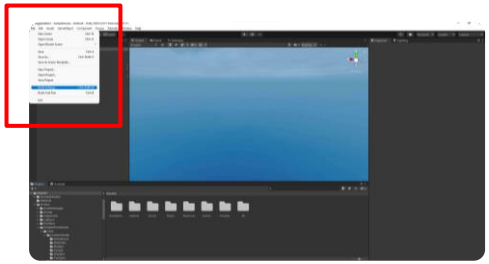


Project

- Start *Unity Hub* application
- Open the template Unity Project folder

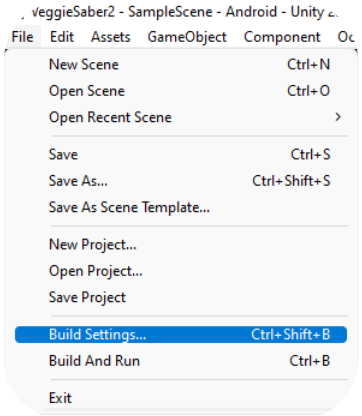
Scene

- Inside the Editor, navigate to Scenes/
- Open VeggieSaber scene

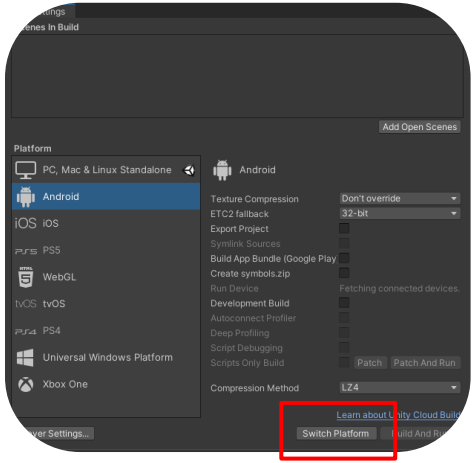


Switch build platform

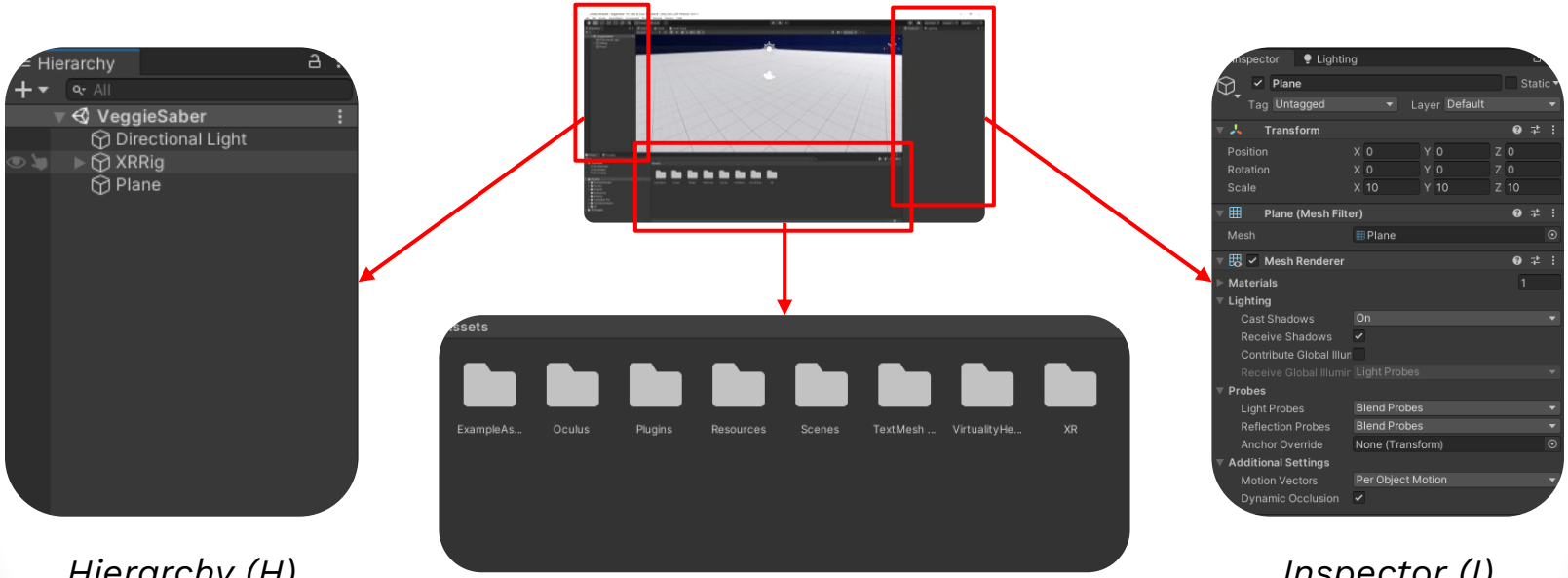
- *File → Build settings*



- Choose *Android* tab and confirm *Switch platform*



Unity Workspace Overview

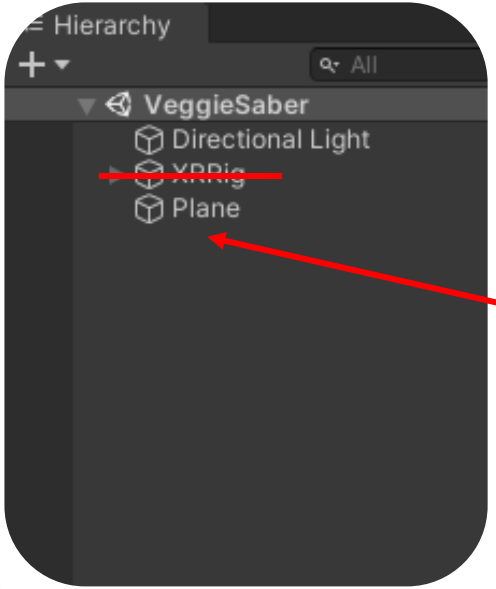


Hierarchy (H)

Assets (A)

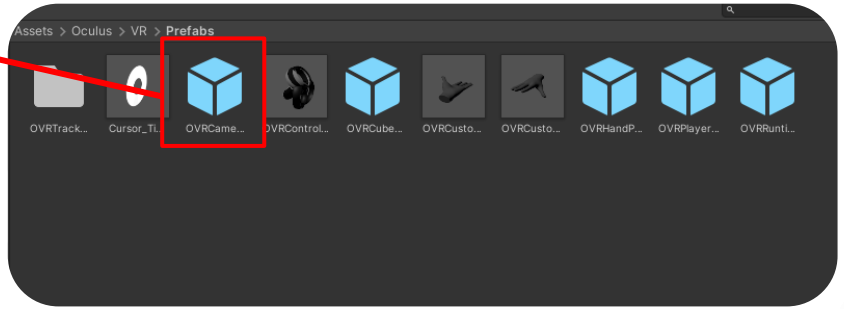
Inspector (I)

Building the virtual avatar

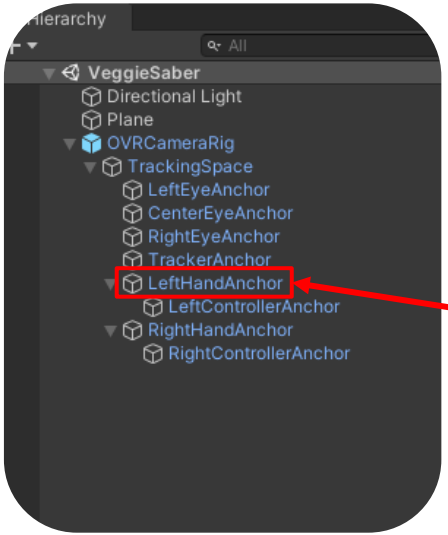


Track head movement with OVRCameraRig

- (H): Delete the XRRig GameObject
- (A → H) Drag and drop the **Oculus/VR/Prefabs/ OVRCameraRig** prefabs into the *Hierarchy*
- (I) Set *Tracking Origin Type* to *Floor Level*

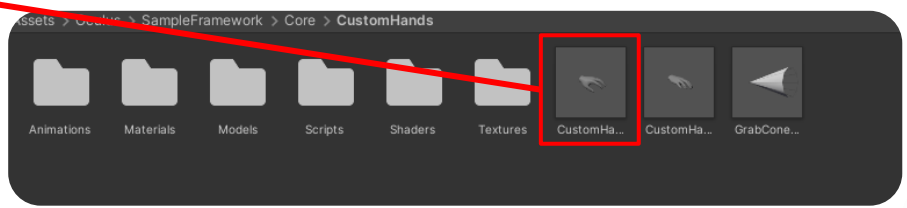


Building the virtual avatar



Setting up hands with OVRCustomHands

- (A): Oculus/SampleFramework/Core /CustomHands/
- (A): Drag and drop the CustomHandLeft prefabs into the LeftHandAnchor
- Do the same for the right hand



Test interaction in VR

Connect the headset

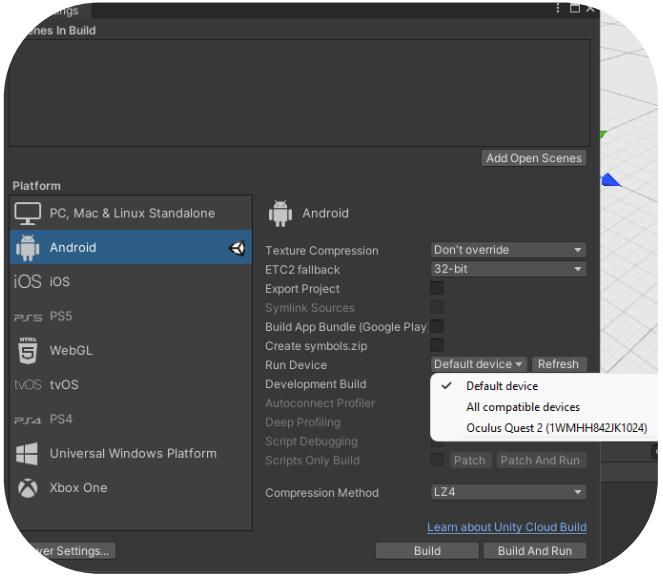
- Plugin the headset to your computer

Set Quest 2 as run device

- *File → Build settings*
- In the *Run Devices*, choose *Oculus Quest 2*

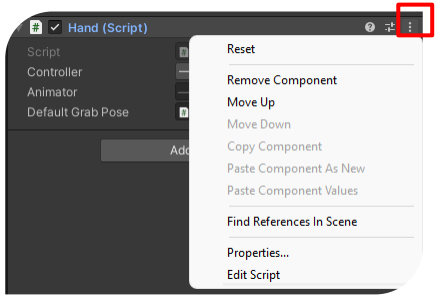
Build

- Choose *Build And Run* to launch the application



Removing pointing and thumbs up

- Choose both CustomHandLeft & CustomaHandRight in the Hierarchy
- In the ⋮ menu, select *Edit Script*



- Comment out the following section

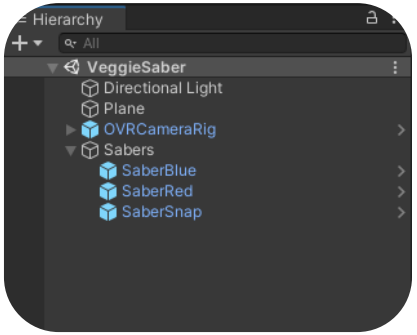
```
float flex = OVRInput.Get(OVRInput.Axis1D.PrimaryHandTrigger,
m_controller);
m_animator.SetFloat(m_animParamIndexFlex, flex);
/*
// Point
bool canPoint = !grabbing || grabPose.AllowPointing;
float point = canPoint ? m_pointBlend : 0.0f;
m_animator.SetLayerWeight(m_animLayerIndexPoint, point);

// Thumbs up
bool canThumbsUp = !grabbing || grabPose.AllowThumbsUp;
float thumbsUp = canThumbsUp ? m_thumbsUpBlend : 0.0f;
m_animator.SetLayerWeight(m_animLayerIndexThumb, thumbsUp);
*/
float pinch = OVRInput.Get(OVRInput.Axis1D.PrimaryIndexTrigger,
m_controller);
m_animator.SetFloat("Pinch", pinch);
```

Add Sabers!!!

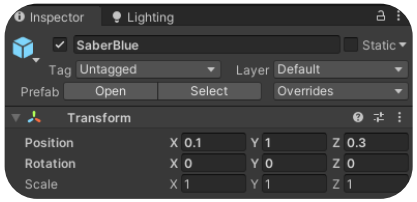
Add Saber

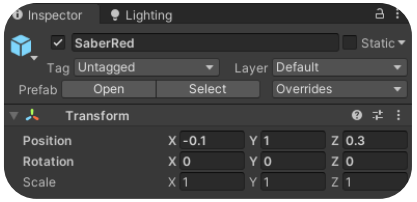
- Right click the **Hierarchy** and select *Create Empty*. Name it as *Sabers*
- Navigate to VirtualityHelpers → Prefabs
- Drag and drop SaberBlue, SaberRed, SaberSnap prefabs into Sabers



Adjust the position

- SaberBlue's position (right hand):
  - *X: 0.1 Y: 1 Z: 0.3*
- SaberRed's position (left hand):
  - *X: -0.1 Y: 1 Z: 0.3*





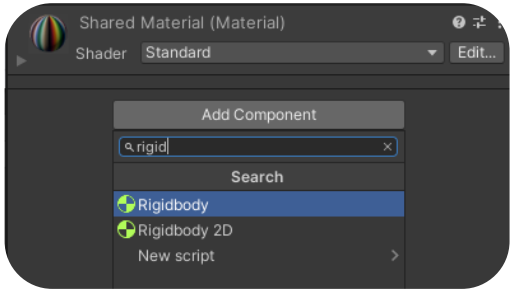
But how to grab the Sabers?

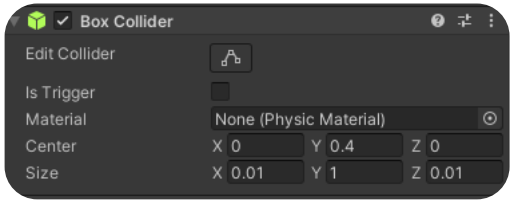
Make the Saber *stronk*!

- Select both SaberRed and SaberBlue
- Add a *Rigidbody* components, uncheck *UseGravity*, check *IsKinematic*

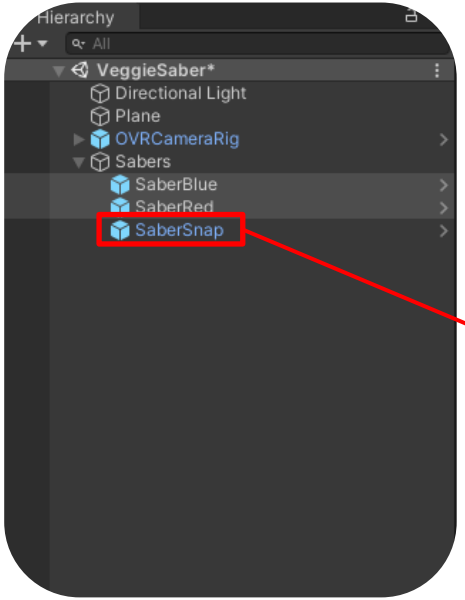
Detect grabbing by collision

- Select both SaberRed and SaberBlue
- Add a *BoxCollider* component and adjust the parameters:
  - Center: X: 0 Y: 0.4 Z: 0
  - Size: X: 0.01 Y: 1 Z: 0.01



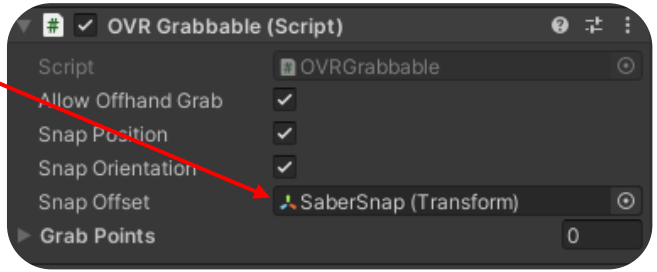


But how to grab the Sabers?



Make the Saber grabbable

- Select both SaberRed and SaberBlue
- Add a *OVR Grabbable* components
- Check both *Snap Position* and *Snap Orientation* option
- Drag the SaberSnap into the *SnapOffset*



Trigger the slice

- Navigate to VirtualityHelpers/Scripts
- Open Slicer.cs

```
public class Slicer : MonoBehaviour
{
    public GameObject gameManager;
    public Controller controller;

    IEnumerator HapticEvent() {...}

    private void OnTriggerEnter(Collider other) {...}

    // Get a cutting plane from the rotation/position of the saber
    private Plane GetPlane(GameObject go) {...}

    // Clone a Mesh "half"
    private Mesh CloneMesh(Plane p, Mesh oMesh, bool halve) {...}

    // Configure the GameObject
    private GameObject MakeHalf(GameObject go, bool isLeft) {...}

    // Make two GameObjects with "halves" of the original
    private void SplitMesh(GameObject go) {...}
}
```

Invoke when the Saber collides

Then split the Veggies (Mesh)

Trigger the slice

```
private void OnTriggerEnter(Collider other)
    {
        StartCoroutine(HapticEvent());
        SplitMesh(other.gameObject);
        Destroy(other.gameObject);
    }
```

Invoke when the Saber collides

Add haptic feedbacks when collides
Split the Veggies in half
Destroy the current Veggies

```
private void SplitMesh(GameObject go)
    {

        GameObject leftHalf = MakeHalf(go, true);
        GameObject rightHalf = MakeHalf(go, false);

        GetComponent<AudioSource>().Play();
    }
```

Split the Veggies (Mesh)

Create 2 copies of the Veggie

Play the hit sound

Trigger the slice

```
private GameObject MakeHalf(GameObject go, bool isLeft)
{
    Plane cuttingPlane = GetPlane(go);
    GameObject half = Instantiate(go);
    MeshFilter filter = half.GetComponent<MeshFilter>();
    filter.mesh = CloneMesh(cuttingPlane, filter.mesh, isLeft);

    float sign = isLeft ? -1 : 1;
    half.transform.position = go.transform.position +
transform.rotation * new Vector3(sign * 0.05f, 0, 0);

    half.GetComponent<Rigidbody>().isKinematic = false;
    half.GetComponent<Rigidbody>().useGravity = true;

    half.GetComponent<Collider>().isTrigger = false;
    Destroy(half, 2);

    return half;
}
```

Get the slicing plane
Make a copy of
- Object: Instantiate()
- Mesh: CloneMesh()

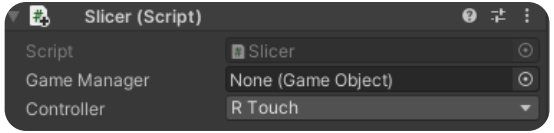
Separate the halves slightly

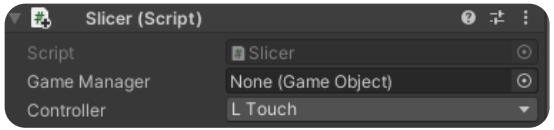
Re-enable physics

Disable further slicing
Set destroy time

Trigger the slice

- Select both SaberBlue and SaberRed
- Drag and drop the *Slicer.cs* script into both Sabers
- In the *Controller* option, select
  - *R Touch* for SaberBlue
  - *L Touch* for SaberRed

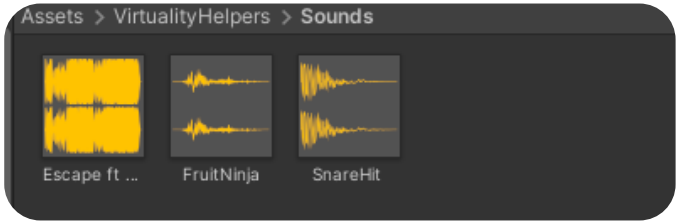


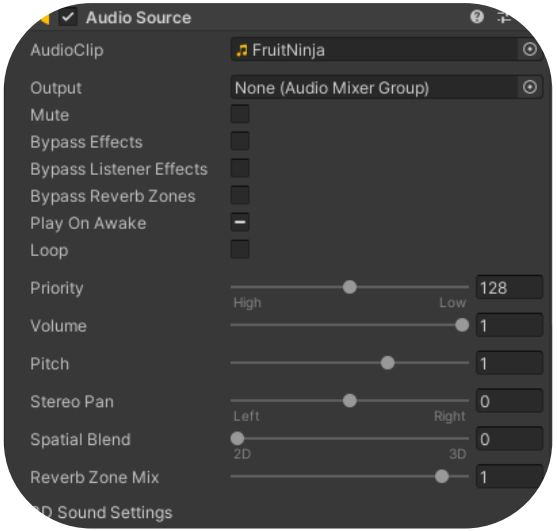


Make some noise

Adding sound effect

- Select both SaberRed and SaberBlue
- Add an *AudioSource* component
- Uncheck *Play On Awake*
- Drag and drop the VirtualityHelpers/Sounds/FruitNinJa sound effects into the *AudioClip* slot

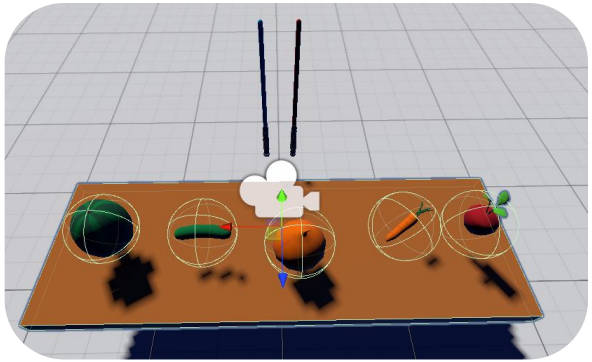




Let's chop some Veggies

Add a demo chopping tables

- Drag and drop VirtualityHelpers/Prefabs/ ChoppingTable onto the hierarchy
- Build the application and test it out!



Tea breakkk!

We will be back in 10 minutes

Make the Veggie fly

Remove the demo chopping tables

Define how Veggie will move

- Navigate to VirtualityHelpers/Scripts
- Open **VeggieBehaviours.cs**

```
void Update()
{
    if (rb.isKinematic)
    {
        dTime += Time.deltaTime;
        if (dTime < 1.0f)
        {
            Vector3 position = transform.position;
            float z = position.z;
            position.z = destination.z;

            position = Vector3.Lerp(position, destination, dTime);
            position.z = z;
            transform.position = position;
        }

    transform.Translate(movement * Time.deltaTime);
    }
}
```

Only update if the object is moving (not sliced)

Keep track of the time ($t_{now} = t_{past} + \Delta t$)

Move veggie to the correct lane in 1 sec

Move veggie to the destination (along z-axis)

Make more Veggies!!!

Generate Veggies on beats

- Navigate to VirtualityHelpers/Scripts
- Open **VeggieGenerator.cs**

```
void Update()
{
    interval += Time.deltaTime;
    float beatInterval = 60.0f / BPM;

    if (interval > beatInterval)
    {
        interval = 0f;
        if (Random.Range(0.0f, 1.0f) < cutoff)
        {
            CreateVeggie();
        }

    cutoff += 0.01f;
    }
}
```

Keep track of the time ($t_{now} = t_{past} + \Delta t$)
Duration between 2 beats

For every beat, there's some chance of creating Veggie

cutoff is larger every Update()
⇒ more chance of creating Veggie
⇒ more Veggies!!!

Make more Veggies!!!

```
void CreateVeggie()
{
    if (veggies.Length == 0) return;

    int randomVeggie = Random.Range(0, veggies.Length - 1);
    GameObject veggie = Instantiate(veggies[randomVeggie]);
    veggie.transform.position = transform.position;

    int pos = Random.Range(0, 5);
    Vector3 destination = transform.position + new  Vector3(
                                    startPositions[pos, 0],
                                    startPositions[pos, 1],
                                    startPositions[pos, 2]);

    VeggieBehaviour comp = (VeggieBehaviour)
                    veggie.AddComponent(typeof(VeggieBehaviour));

    comp.movement = new Vector3(0, 0, -6);
    comp.destination = destination;
}
```

Create random Veggies

Choose random lane for Veggie to run

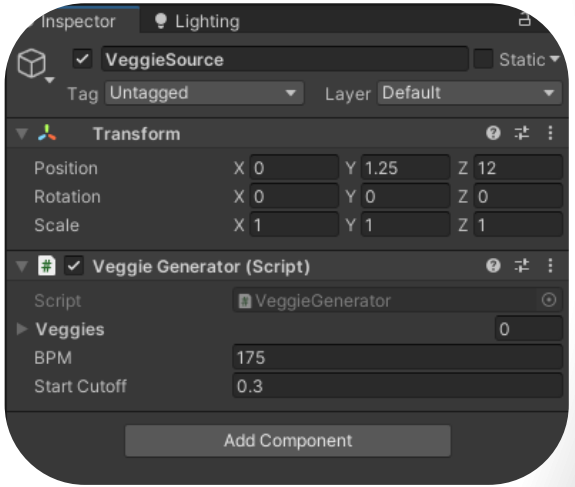
Add VeggieBehaviour component to the Veggie

Motion direction and destination

Make more Veggies!!!

Generate Veggies on beats

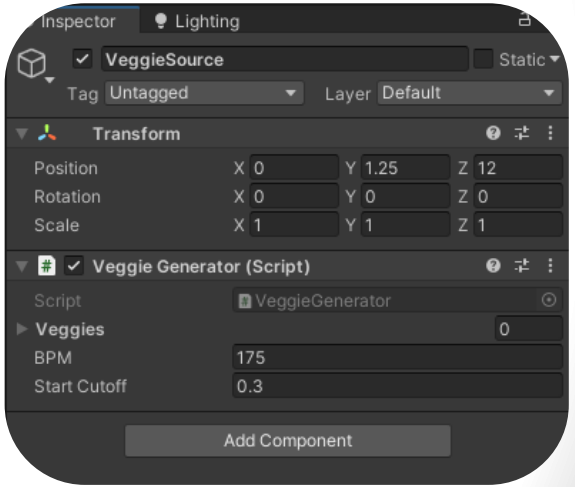
- Create an empty *GameObject* named Level
- Create an empty child *GameObject* of Level named VeggieSource
- Adjust the position:
  - *X: 0 Y: 1.25 Z:12*
- Add the *VeggieGenerator* component
- Drag and drop all Veggies in the VirtualityHelpers/Prefabs/Veggies into the list in VeggieSource



What if I misses?

Say “Missed”!

- Create an empty child GameObject of Level named EndWall
- Adjust the position to put it behind:
  - *X: 0 Y: 0 Z: -1*
- Add *Rigidbody* component and check *isKinematic*
- Add *BoxCollider* component and make a large wall by seting size to
  - *X: 5 Y:5 Z: 1*



What if I misses?

Say “Missed”!

- Navigate to VirtualityHelpers/Scripts
- Open **TrapMisses.cs**

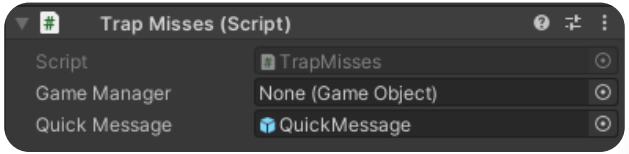
```
void Update()
{
    GameObject textMessage = Instantiate(quickMessage);
    textMessage.transform.position = gameObject.transform.position;
    textMessage.GetComponent<TextMeshPro>().text = "Missed!";

    Destroy(other.gameObject);
}
```

Create a copy of QuickMessage
Set it to “Missed!”

Remove the colliding Veggie from the scene

- Attach *TrapMisses.cs* component to EndWall
- Drag and drop VirtualityHelpers/Prefabs/ QuicMessage

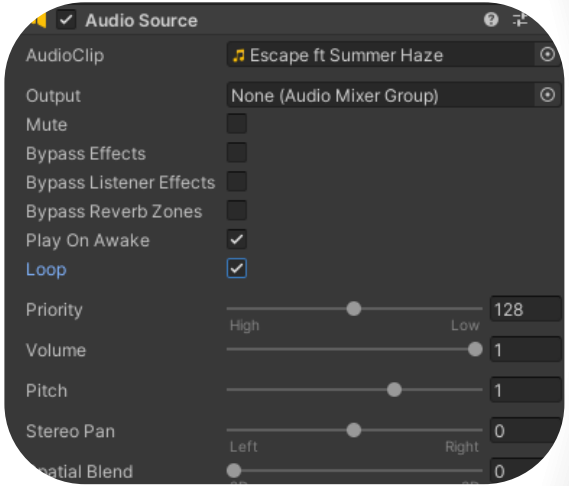


Almost there... Musik!

Adding level music

- Attach an *Audio Source* component to EndWall
- Drag and drop VirtualityHelpers/Sounds/Escape into the *AudioClip* slot
- Check the *Loop* option

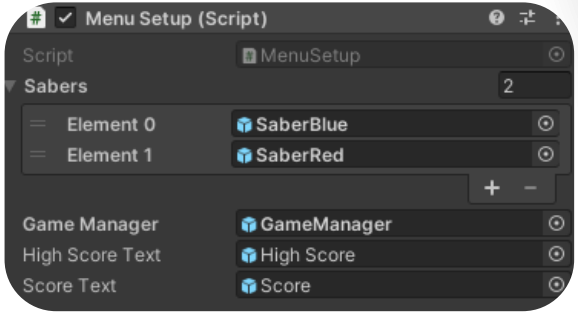
And you are ready to go! Build and test it out!



Finalizing steps...

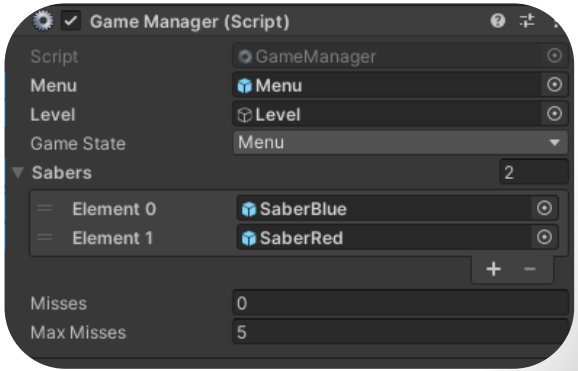
Creating a game menu

- Drag and drop VirtualityHelpers/Prefabs/Menu into the **Hierarchy**
- In the Menu's Inspector panel, drag and drop the Saber into their slots



Creating a game manager

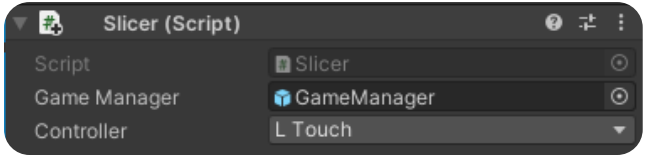
- Drag and drop VirtualityHelpers/Prefabs/GameManager into the **Hierarchy**
- In the GameManager's Inspector panel, drag and drop the Sabers, Menu, Level into their respective slots
- In the Menu's Inspector panel, drag and drop the GameManager into its slot

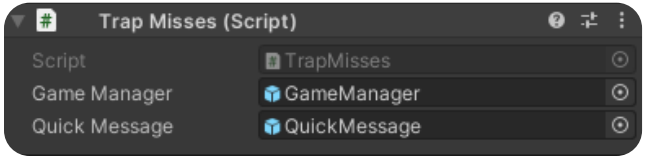


Finalizing steps...

Add GameManager to others

- In the Inspector windows of SaberRed, SaberBlue, EndWall, drag and drop the GameManager into its slot



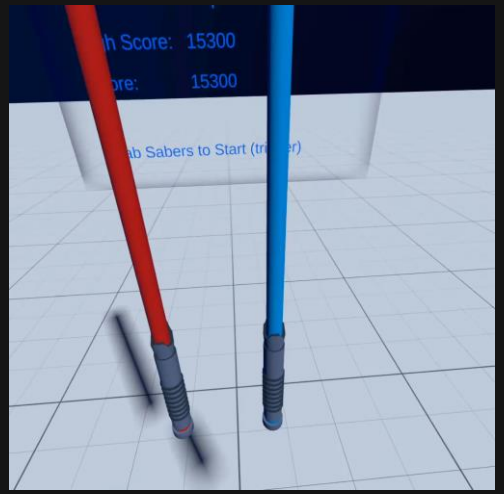


But what does the GameManager do?

- Record the scores
- Count number of misses
- Switch between Menu and Level

Woohoo, we're done!

Enjoy your game



Exploration & experience

Tweak your app or play VR games

Thanks!

2022

References

[1] M. Larson, “Veggie Saber – Introduction to Unity Development with the Oculus Quest,” *raywenderlich.com*. https://www.raywenderlich.com/4912095-veggie-saber-introduction-to-unity-development-with-the-oculus-quest

[2] G. C. Burdea and P. Coiffet, “Virtual Reality Technology, 2nd Edition | Wiley,” *Wiley.com*. https://www.wiley.com/en-us/Virtual+Reality+Technology%2C+2nd+Edition-p-9780471360896

[3] NASA Advanced Supercomputing Division, “Virtual Reality: Definition and Requirements.” https://www.nas.nasa.gov/Software/VWT/vr.html

[4] M. Mulders, J. Buchner, and M. Kerres, “A Framework for the Use of Immersive Virtual Reality in Learning Environments,” *International Journal of Emerging Technologies in Learning (iJET)*, vol. 15, pp. 208–224, Dec. 2020, doi: 10.3991/ijet.v15i24.16615.

[5] Clay, Viviane & König, Peter & Koenig, Sabine. (2019). Eye Tracking in Virtual Reality. Journal of Eye Movement Research. 12. 10.16910/jemr.12.1.3.