|  |  |
| --- | --- |
| |  | | --- | | **Lab 6 - Teleporting – Locomotion** |   **Technical Requirements**  To implement the projects and exercises in this lab, you will need the following:   * A PC or Mac capable of running Unity 2021.3.18 LTS or later, along with an internet connection to download files. * A VR headset supported by the Unity XR platform. * Lab 6 Files are available on Blackboard under Week 7 * THIS IS A TWO-WEEK LAB     **Implementing Basic Glide Locomotion**  create a similar new scene to use in this chapter, with the following steps:     * + 1. Create a new scene, using **File | New Scene**.     2. Create a ground plane using **GameObject | 3D Object | Plane**, rename this as Ground Plane, reset its transform with **Transform** | right-click | **Reset**, and apply a colored material such as the **Ground Material**     3. Create a photo wall using **GameObject | 3D Object | Quad**, rename this as Photo, then set **Transform | Position** (0, 3.75, 5), **Transform | Scale** (10, 7.5,   10), and apply the **Grand Canyon** photo material using **Mesh Renderer | Materials | Element 0 | doughnut-icon | GrandCanyon**.   * + 1. Add an **XR Rig**, and **Reset** its **Transform** to the origin.     2. Save the scene with the name 07-Locomotion-0 (**File | Save As**.)     To help visualize in the Unity editor, let's add a **Capsule** object to represent the player rig, as follows:     * + 1. With **XR Rig** selected in the **Hierarchy**, add a child **Capsule** (right-click | 3**D Object** | **Capsule**)     2. Set its **Y** position to 1 (**Transform | Position** to (0, 1, 0) and **Scale** 0.5, 1,   0.5 )  3. Add another **Capsule** as its "nose" to show the forward direction (as a child of the first **Capsule**, right-click | **3D Object** | **Capsule**, set **Position** (0, 0.75,  0.5), **Rotation** (90, 0, 0), and **Scale** (0.1, 0.2, 0.05))  Our capsule body will look like the following screenshot: |

Note that in VR, you may see this self-avatar, depending on your position. No worries: it's

intended to help demonstrate our development and will not be part of the final scene (later,

we'll disable its Mesh Renderer component). Now, we'll write a script to enable us to move

around using the thumbstick on the hand controller. After that, we'll see how to change

direction and avoid obstacles.

**Moving Forward with the Thumbstick**

The glide locomotion behavior will move the player rig—

**XR Rig**

—position based on the

thumbstick press, and maximum velocity value. To begin, create a new script,

GlideLocomotion

, on the

**XR Rig**

, as follows:

1.

Select the

**XR Rig**

in

**Hierarchy**

.

2.

Create a new C# script by selecting

**Add Component**

|

**New Script**

(

named

GlideLocomotion

)

|

**Create And Add**

.

3.

Open the script for editing, and write it as follows:

public

class

GlideLocomotion

:

MonoBehaviour

{

public

Transform

rigRoot;

public

float

velocity

=

f;

2

//

meters

per

second

private

void

Start()

{

if

(

rigRoot

==

null)

rigRoot

=

transform;

}

private

void

Update()

{

float

forward

=

Input.GetAxis("

XRI\_Right\_Primary2DAxis\_Vertical");

if

(

forward

!=

0

f)

{

Vector3

moveDirection

=

Vector3.forward;

moveDirection

\*=



|  |
| --- |
| -forward \* velocity  \* Time.deltaTime; rigRoot.Translate(moveDirection);  }  }  }     1. Save the script and go back to Unity. 2. With the **GlideLocomotion** script attached to **XR Rig**, drag the **XR Rig** game object onto the **Rig Root** slot. 3. Press **Play**.   In VR, using the right-hand controller, press the thumbstick forward and you'll glide  **Rotating with the Thumbstick**  Use the following steps to add this to the GlideLocomotion script. First, at the top of the class, add a rotationSpeed variable, as shown in the following code snippet:    public float rotationSpeed = 100f; // degrees per second    In the Update() function, add the following code:  float sideways = Input.GetAxis("  XRI\_Right\_Primary2DAxis\_Horizontal"); if (sideways != 0f)  { float rotation = sideways \* rotationSpeed  \* Time.deltaTime; rigRoot.Rotate(0, rotation, 0); }    **Moving in the Direction You're Looking or Pointing**  Declare a trackedTransform variable at the top of the GlideLocomotion class,    public Transform trackedTransform; // camera or controller, null for thumbstick    In Update(), we'll use the device's forward direction rather than the fixed Vector3.forward one, as follows:  float forward = Input.GetAxis("  XRI\_Right\_Primary2DAxis\_Vertical"); if (forward != 0f)  {  Vector3 moveDirection = Vector3.forward; **if (trackedTransform != null)**  **{ moveDirection = trackedTransform.forward; moveDirection.y = 0f;**  **}** moveDirection \*= -forward \* velocity  \* Time.deltaTime; rigRoot.Translate(moveDirection); } |

Disable the use of the horizontal thumbstick for rotation, as these modes are mutually exclusive.

Wrap that code around another condition, as follows:

**if (trackedTransform == null)**

**{**

float

sideways

=

Input.GetAxis("

XRI\_Right\_Primary2DAxis\_Horizontal");

if

(

sideways

!=

f)

0

{

float

rotation

=

sideways

\*

rotationSpeed

\*

Time.deltaTime;

rigRoot.Rotate(0,

rotation,

;

0)

}

**}**

Save the script and go back to Unity and assign the

**Main Camera**

to the tracked transform, as

follows:

1.

With

**XR Rig**

selected in the

**Hierarchy**

, drag the child

**Main Camera**

object onto the

**Tracked**

**Transform**

slot. The component looks like the following in the

**Inspector**

:

2.

Press

**Play**

.

3.

In VR, using the right-hand controller, press the thumbstick forward and you'll glide forward.

As you move, turn your head to look in a different direction, and you'll move in the direction

you are looking for.

Change

**Tracked Transform**

from

**Main Camera**

to the

**RightHand Controller**

, as follows:

With

**XR Rig**

selected in the

**Hierarchy**

, drag its child

**RightHand Controller**

game object

onto the

**Glide Locomotion | Tracked Transform**

slot. The component now looks like

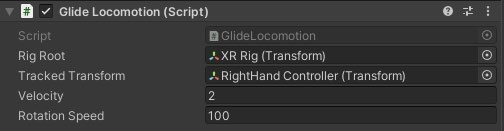
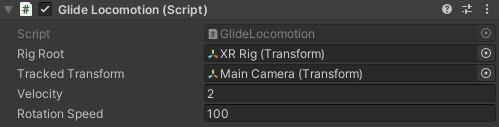
this:

Press

**Play**

, you can point with your hand controller in the direction you want to glide and

press the thumbstick forward. Simultaneously, you can push the thumbstick to the side to



|  |
| --- |
| also twist the rig's orientation and default forward direction.    **Avoiding Obstacles**  Add some solid obstacles to the scene, such as a couple of crates. Follow the steps given here:     1. Add a **Cube** object to the scene, using **GameObject | 3D Object | Cube**, positioned at (-2.5, 0.5, 1.5.) 2. Add another cube with right-click, then **Cube | Duplicate**, positioned at (2.5, 0.5, -1.5.) 3. If you'd like, apply the **Crate Material** we created in previous labs.     If you were to press **Play** now and move about, you'd find you can pass through the crates as if you're a ghost.    Introduce a **Rigidbody** to the **XR Rig** with appropriate constraints to keep it from falling over, as follows:     1. Add a **Rigidbody** to the **XR Rig**: with **XR Rig** selected, select **Component | Physics | Rigidbody**. 2. Uncheck the **Use Gravity** checkbox. 3. Check the **Freeze Rotation X, Y, Z** checkboxes.     Now, when you press **Play** and move around the scene, if you bump into an object, you'll either get stopped or slide around it as the curved surface of your capsule collider rubs against the surface of the object.    **Climbing a Wall**  Given a climbing wall, the player can reach up and grab a hold, then pull themself up, reach with the other hand to grab another hold, and so on.  Use the XRI Toolkit's **Interactor** and **Interactable** components.  Build a wall with a series of **GrabHold** prefabs.  Write two scripts. A GrabClimbl script notifies the ClimbController when the player has grabbed or released a hold. The ClimbController moves the **XR Rig**, and it detects when the player has completely let go, causing them to fall.  Begin with a basic scene such as the one defined in this lab, as follows:     1. Open the scene by clicking **File | Open Scene**, and select 07-Locomotion-0. 2. Save into a new scene named 07-Locomotion-2 for this topic, using **File | Save As**.     **Building a Wall with Grab Holds**  In this project, we are going to climb a climbing wall. The wall will be a simple cube slab with grab-holds that the player can use to pull themselves up. To create a wall, follow the steps given next: |

|  |
| --- |
| 1. Create an empty game object (**GameObject | Create Empty**), and name it ClimbingWall. 2. Position it a half-meter from the player, at **Position** (0, 0, 0.5.) 3. Create a child **Cube** for the wall (**right-click** | **ClimbingWall** | **3D Object** | **Cube**), and name it Wall. 4. Set its **Scale** (3, 5, 0.1) and **Position** (0, 2.5, 0.)     Next, we'll add a grab-hold object on the wall, make it a prefab, and copy it to multiple positions up the wall. To do that, follow the steps given next:     1. Create a small cube on the wall (right-click **ClimbingWall** | **3D Object** | **Cube**), and name it GrabHold. 2. Set its **Scale** (0.1, 0.1, 0.1), and position it at an easy reaching distance—for example, (0.5,   0.8, -0.1.)   1. Give it a distinct color—for example, using the **Red Material** created in a prior chapter   (**Mesh Renderer | Materials | Element 0 | doughnut-icon | Red Material**.)   1. Make it a prefab by dragging the **GrabHold** from the **Hierarchy** window into your **Project** window's Prefabs/ folder.     Then, duplicate the **GrabHold** on the **Wall**, and move each to various locations such as the following X, Y coordinates: |

The next step is to make the grab-holds interactable.

**Adding the XRI Interactor and Interactable Components**

Replace the default hand controllers (that have an

**XR Ray Interactor**

)

with ones using an

**XR Direct Interactor**

. It's easy enough to just disable or delete the default ones and create

new ones, as follows:

1.

In the

**Hierarchy**

window, unfold the

**XR Rig**

so that you can see the

**RightHand Controller**

and

**LeftHand Controller**

objects.

2.

Disable both

**RightHand Controller**

and

**LeftHand Controller**

objects (uncheck its

**Enable**

checkbox.)

3.

Create a new

**Direct Interactor**

named RightHand

Direct

Interactor

(

**GameObject|XR|Device-based |XR|Device-based**

**|Direct Interactor**

)

and

rename

it

RightHand Direct

Interactor.

4.

Drag it as a child of

**Camera Offset**

(

sibling of the original

**RightHand Controller**

.)

5.

Change its

**Controller Node**

to

**Right Hand**

.

6.

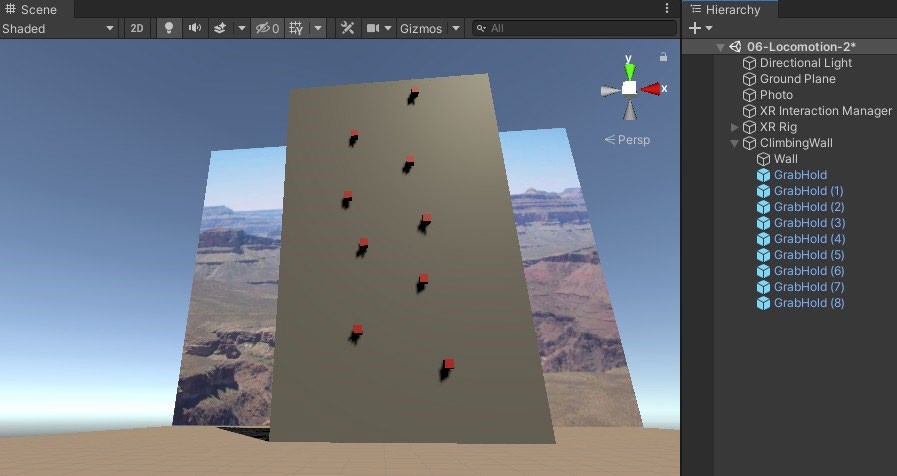
Repeat

*steps 3-5*

for the LeftHand

Director

Interactor.



Add hand graphic so that the player will be able to see where their hands are in VR. For

simplicity, we'll just use a sphere "fist," as follows (or, instead, you could use a better graphic

prefab on the

**XR Controller | Model Prefab**

slot):

1.

Right-click, then go to

**RightHand Direct Interactor | 3D Object | Sphere**

.

2.

Set its

**Scale**

(0.1

,

0.1

,

0.1.)

3.

Repeat for the

**LeftHand Direct Interactor**

.

To make all the

**GrabHold**

objects interactable, open the

**GrabHold**

prefab for editing and add

an

**XR Simple Interactable**

component, as follows:

1.

Double-click the

**GrabHold**

to open it for editing

2.

With its root

**GrabHold**

object selected, add a

**Simple Interactable**

(

**Component | XR | XR**

**Simple Interactable**

.)

(

While we're calling these objects "grab holds" that the player will grab to climb the wall, it's not a

"grab interactable" such as a gun or a knife that the player can grab and throw.) Note that this

also adds a

**Rigidbody**

that the

**Interactable**

needs. But we do not want the

**GrabHold**

to move

when it's grabbed, so let's constrain it by following the steps given next:

1.

Uncheck the

**Use Gravity**

checkbox.

2.

Check the

**Constraints | Freeze Position | X, Y, Z**

checkboxes.

3.

Check the

**Constraints | Freeze Rotation | X, Y, Z**

checkboxes.

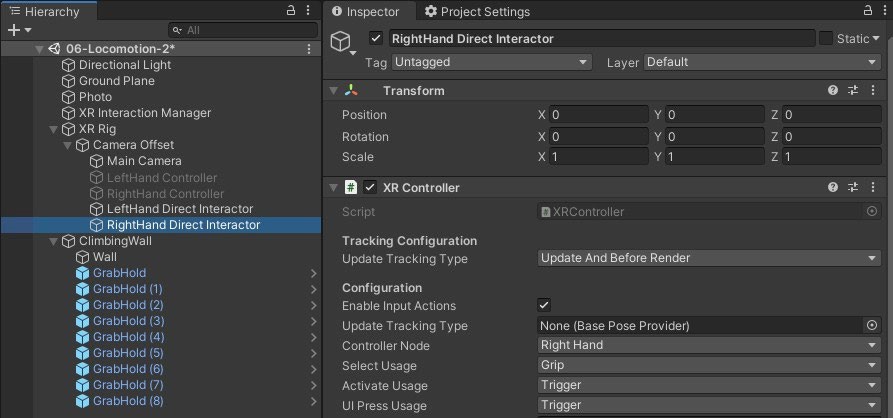
4.

**Save**

your changes to the prefab.

**Adding a ClimbController Script**

The first thing we're going to do to implement the climb mechanic is writing a ClimbController



|  |
| --- |
| script that will handle the grab/pull/release actions coming from any of the **GrabHold** objects. Add a new script to the **ClimbingWall** named ClimbController, as follows:     1. With **ClimbingWall** selected, create a new C# script (**Add Component | New Script |** ClimbController **| Create And Add**.) 2. Open the script for editing. 3. Create a first iteration of the script, as follows:     public class ClimbController : MonoBehaviour  { public GameObject xrRig;    private void Start()  { if (xrRig == null) xrRig = GameObject.Find("XR Rig");  } public void Grab()  {  }    public void Pull(Vector3 distance)  { xrRig.transform.Translate(distance); }    public void Release()  {  }  }    Write the GrabClimb script behavior that the **GrabHold** objects will use for calculating the pull distance and notify the ClimbController.    **Adding the GrabClimb Script and Actions**  Create a GrabClimb script on the **GrabHold** prefab, as follows:     1. Double-click the **GrabHold** prefab to open it for editing. 2. With its root **GrabHold** object selected, create a new C# script by selecting **Add Component | New Script |** GrabClimb **| Create And Add**. 3. Open the script for editing. 4. Start writing the script, as follows:     using System.Collections.Generic; using UnityEngine; using UnityEngine.XR.Interaction.Toolkit;    public class GrabClimb : MonoBehaviour |

{

|  |
| --- |
| private XRSimpleInteractable interactable; private ClimbController climbController; private bool isGrabbing; private Vector3 handPosition;  private void Start()  { interactable = GetComponent<XRSimpleInteractable>(); climbController = GetComponentInParent<ClimbController>(); isGrabbing = false;  }    Add the following code:    public void Grab()  { isGrabbing = true; handPosition = InteractorPosition(); climbController.Grab();  }    private Vector3 InteractorPosition()  {  List<XRBaseInteractor> interactors = interactable.hoveringInteractors;  if (interactors.Count > 0) return interactors[0].transform.position;  else return handPosition; }  add the final two functions, as follows:    private void Update()  { if (isGrabbing)  {  Vector3 delta = handPosition - InteractorPosition(); climbController.Pull(delta);  handPosition = InteractorPosition(); }  }    public void Release()  { isGrabbing = false; |

climbController.Release();

}

Save the script. Now, back in Unity, let's connect the Grab and Release functions to the interactor

Select events, as follows:

1.

Double-click the

**GrabHold**

prefab to open it for editing.

2.

On its

**XR Simple Interactable | Interactable Events | On Select Enter**

, press the

**+**

button to add a new event action.

3.

Drag the same

**GrabHold**

game object onto the

**Runtime Only**

object slot.

4.

In its

**Function**

dropdown, select

**GrabClimb | Grab**

.

5.

Likewise, on its

**On Select Exit**

event, press the

**+**

button to add a new action.

6.

Drag the same

**GrabHold**

game object onto the

**Runtime Only**

object slot.

7.

In its

**Function**

dropdown, select

**GrabClimb | Release**

.

8.

Save the prefab changes.

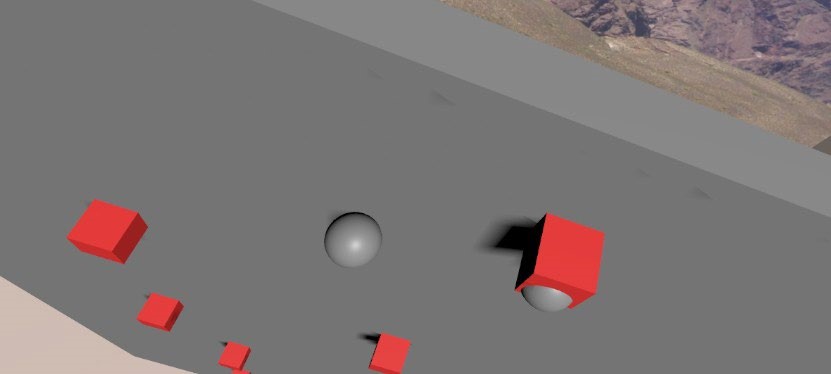
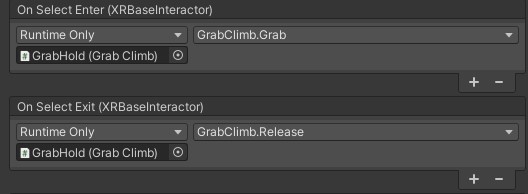
Press

**Play**

and go into VR. Reach out to one of the grab-holds and squeeze the grip button,

then pull yourself up. Repeat with each hand as you climb the wall.

**Falling**



Set up a basic vertical gravity fall now. Follow the steps given next:

1.

Add a

**Rigidbody**

to the

**XR Rig**

(

with

**XR Rig**

selected in

**Hierarchy**

, choose

**Component | Physics | Rigidbody**

.)

2.

Ensure the

**Use Gravity**

checkbox is checked.

3.

Check the

**Is Kinematic**

checkbox, to temporarily disable the physics.

4.

Constrain the fall to downward only. Check the

**Constraints | Freeze Position X, Z**

and

**Freeze Rotation X, Y, Z**

options. Leave the

**Position Y**

constraint

*unchecked*

.

The

**Rigidbody**

settings are shown in the following screenshot:

Now, we should update the ClimbController script in the following ways. First, add new

variables for the grabCount, rigidBody, and groundLevel and initialize them in Start, as

follows:

public

class

ClimbController

:

MonoBehaviour

{

public

GameObject

xrRig;

**private int grabCount; private**

**Rigidbody rigidbody; private float**

**groundLevel;**

private

void

Start()

{

if

(

xrRig

==

null)

xrRig

=

GameObject.Find("XR

Rig");

**grabCount = 0;**

**rigidbody = xrRig.GetComponent<Rigidbody>(); groundLevel =**

**xrRig.transform.position.y;**

}

public

void

Grab()

{

grabCount++;

rigidbody.isKinematic

=

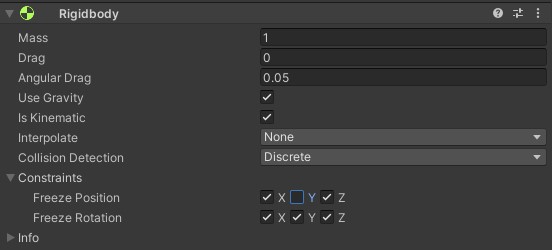
true;

}

public

void

Release()



|  |
| --- |
| { grabCount--; if (grabCount == 0)  { rigidbody.isKinematic = false; }  }    Add an update function that ensures the rig never goes below ground level, and if so, resets the Y position and **Rigidbody** kinematics, as illustrated in the following code snippet:    private void Update()  { if (xrRig.transform.position.y <= groundLevel)  {  Vector3 pos = xrRig.transform.position; pos.y = groundLevel;  xrRig.transform.position = pos; rigidbody.isKinematic = true; }  }    Save the script. In Unity, press **Play**. Now, you can climb the wall, but don't let go, or you'll fall down and have to start over. And there it is!    **Using the XRI Locomotion System**    **Turning in a Snap**  Add to this the ability to turn direction in discrete "snap" angles, using the **Snap Turn Provider** component.  Add a Locomotion System component to the **XR Rig**, as follows:     1. Open the 07-Locomotion-1 scene (**File | Open Scene**.) 2. Select the **XR Rig** game object in **Hierarchy**. 3. Add a **Locomotion System** component (in **Inspector**, **Add Component**, search locomotion **| Locomotion System**.) 4. Save it to a new working scene, 07-Locomotion-3 (**File | Save As**.)     Add the **Snap Turn** component to the **XR Rig**.     1. With **XR Rig** selected, in **Inspector**, press **Add Component**. 2. Search snap and select **Snap Turn Provider** to add the component. 3. To add the left-hand controller, grow the **Controller** list by setting the **Snap Turn Provider | Controller | Size** to 1. 4. Drag the child **LeftHand Controller** game object from the **Hierarchy** window onto the **Controllers | Element 0** slot. |

Clear the

**Glide Locomotion | Tracked Transform**

slot if you tried experimenting with that

mechanic earlier.

Now, when you press

**Play**

and press the left-hand controller's thumbstick sideways, you'll snap

turn in that direction. In accordance with the parameters we have set, when the thumbstick is

pressed to the side, at least 75% off-center (

**Dead Zone**

)

, the player rig will be rotated 45

degrees (

**Turn Amount**

in that direction. When you release the stick, you need to wait at least

)

0.5

seconds

(

**Activation Timeout**

)

before a new press will register another turn. The

**System**

value, which is unassigned, will get filled at runtime by the first

**Locomotion System**

found in

the hierarchy.

**Integrating scripts with Locomotion System**

Update the GlideLocomotion script developed earlier in this lab to integrate with the Locomotion

System. Follow the steps given next:

1.

Open the GlideLocomotion script file for editing, and update it as follows. First, we'll make it a

type of LocomotionProvider (which itself is a type of MonoBehaviour) by changing its

declaration, as follows:

**using UnityEngine.XR.Interaction.Toolkit;**

public

class

GlideLocomotion

:

**LocomotionProvider**

{

2.

Next, add a private flag to track when we're moving, like this:

**private bool isMoving;**

3.

Then, change Update to call the LocomotionProvider functions, as follows:

private

void

Update()

{

**if (!isMoving && !CanBeginLocomotion()) return;**

float

forward

=

Input.GetAxis("

XRI\_Right\_Primary2DAxis\_Vertical");

float

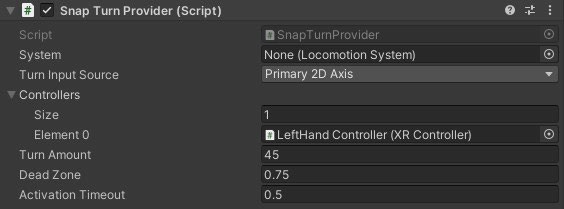
sideways

=

Input.GetAxis("

XRI\_Right\_Primary2DAxis\_Horizontal");

**if (forward == 0f && sideways == 0f)**



|  |
| --- |
| **{ isMoving = false;**  **EndLocomotion(); return;**  **}**    **if (!isMoving)**  **{**  **isMoving = true; BeginLocomotion();**  **}**  if (forward != 0f)  {  Vector3 moveDirection = Vector3.forward; if  (trackedTransform != null)  {  moveDirection = trackedTransform.forward; moveDirection.y = 0f;  }    moveDirection \*= -forward \* velocity \* Time.deltaTime; rigRoot.Translate(moveDirection);  }    **if (trackedTransform == null && sideways != 0f)**  **{**  float rotation = sideways \* rotationSpeed \* Time.deltaTime; rigRoot.Rotate(0, rotation, 0);  }  }      Press **Play** and use the right thumbstick to move through the scene, the left thumbstick, **Snap Turn**, is locked out and will not operate. Once you stop moving, the system is unlocked and you can use the left hand controller to snap turn.  **Teleporting Between Locations**    **Installing the XRI Examples**  Follow the steps given next:     1. If you do not presently have **GitHub Desktop**, use your internet browser to go to <https://desktop.github.com/>to download and install the tool. 2. Then, in your internet browser, go to the XR Interaction Toolkit Examples.   repository [(https://github.com/Unity-Technologies/XR- Interaction-ToolkitExamples)](https://github.com/Unity-Technologies/XR-Interaction-Toolkit-Examples)   1. Click the **Clone Or Download** button, then **Open In Desktop**. 2. This will open the GitHub Desktop package and you should see the **Clone A Repository** dialog box. You may choose a **Local Path** into which you want to place |

|  |
| --- |
| your cloned files. Then, click the **Clone** button.    Once the repository has been cloned and downloaded, open it in Unity Hub     1. Open the **Unity Hub** desktop application on your PC. 2. Navigate to the **Projects** tab, and select **Add**. 3. Find the repository folder you just downloaded, and navigate to the VR folder (it will contain subfolders named Assets, Packages, and ProjectSettings). Click **Select Folder** (this will add the project to the Unity Hub **Projects** list.) 4. In the **Projects** list, if necessary, select a **Unity Version** from the drop-down list with a version you presently have installed on your machine. 5. Then, click the VR project name to open the XRI sample VR project in Unity. If it asks you *Do you want to upgrade your Project to a newer version of Unity?*, press **Confirm**.     In Unity, export everything from the Unity package to a file named XRI-ExamplesVR.unitypackage, using the following steps:     1. In the **Project** window, right-click the root Assets/ folder (we want to export all of the **Project** folders we just cloned.) 2. Choose **Export Package...**, then **Export**. 3. Choose a filename and location, such as XRI-Examples-VR.unitypackage.     You now have a Unity package with the current assets of the Examples repository that you can use in this and other projects.  With the current project open (for example, VR\_Is\_Awesome do the following:     1. Select **Assets | Import Package | Custom Package..**. 2. Navigate to and choose the XRI-Examples-VR package we just created, and choose **Open**. 3. In the **Import Unity Package** dialog box, click **Import**. 4. Because we're using the **Universal Windows Platform** (**UWP**) render pipeline, you might need to convert the imported materials (select **Edit | Render Pipeline**   **| Universal Render Pipeline | Upgrade Project Materials to UWP Materials**.)    **Adding Teleportation**  For our example, let's begin with the first locomotion scene we created earlier in this chapter— that is, 06-Locomotion-1.  First, add a **Locomotion System** and save the scene as a new working scene, as follows:     1. Open the 06-Locomotion-1 scene (**File | Open Scene**.) 2. Select the **XR Rig** game object in **Hierarchy**. 3. Add a **Locomotion System** component (in **Inspector**, **Add Component**, search locomotion **| Locomotion System**.) 4. Now, we can add a Teleportation Provider to the **XR Rig** (in **Inspector**, **Add**   **Component**, search teleport | **Teleportation Provider**.) |

5.

Save it to a new working scene, 07-Locomotion-4 (

**File | Save As**

.)

That's the Provider side of the equation. Now, we can add some teleport interactables to the

scene. Let's add one area and several anchors.

Add a teleport area by dragging the TeleportArea prefab to the back-left corner of the play area,

as follows:

1.

From the

**Project**

window, locate the

**TeleportArea**

prefab (this can be found in

Assets/Prefabs/.)

2.

Drag it into the scene

3.

Set its

**Scale**

to (0.5,

1

,

0.5)

and

**Position**

to (-3,

0

,

3.)

Press

**Play**

and point with the hand controller at the

**TeleportArea**

, it highlights the

**HighlightFence**

around its perimeter, and the

**TeleportReticle**

is displayed at the target

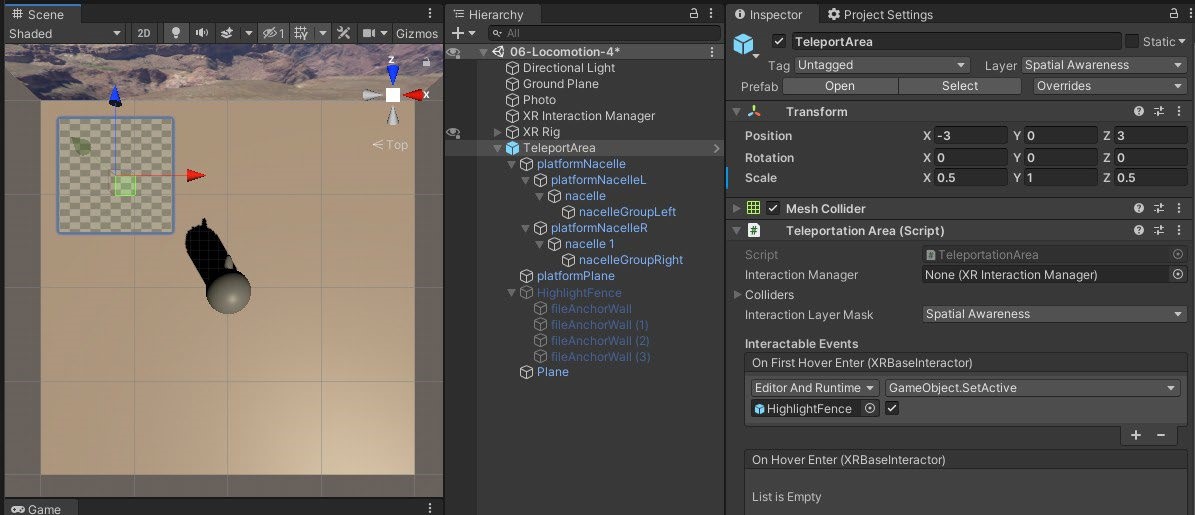
position.

Squeeze the grip button (or whichever button is configured to activate the teleport), and the

**XR**

**Rig**

is moved to the selected location.



Make three separate

**TeleportAnchor**

instances in three quadrants of the ground plane:

1.

From the

**Project**

window, locate the

**TeleportAnchor**

prefab (this can be found in

Assets/Prefabs/.)

2.

Drag it into the scene and set its

**Position**

to (3,

0

,

3)

and

**Rotation**

,

(0

-135

,

0.)

3.

Add another at

**Position**

(-3

,

0

,

-3)

and

**Rotation**

(0

,

45

,

0.)

4.

And another at

**Position**

(3

,

0

,

-3)

and

**Rotation**

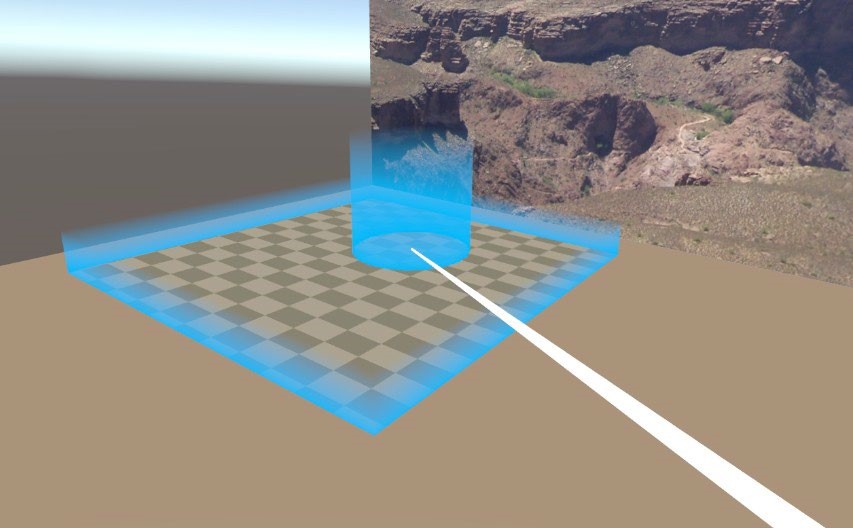
(0

,

-45

,

0.)



**Restricting Interaction to a Specific Layer**

When your scene contains many interactable objects, you will want to limit the work required to

select teleportation interactables by placing them on a specific layer.

The default objects we've been using may be on any layer (

**Everything**

)

or could be already set

up to

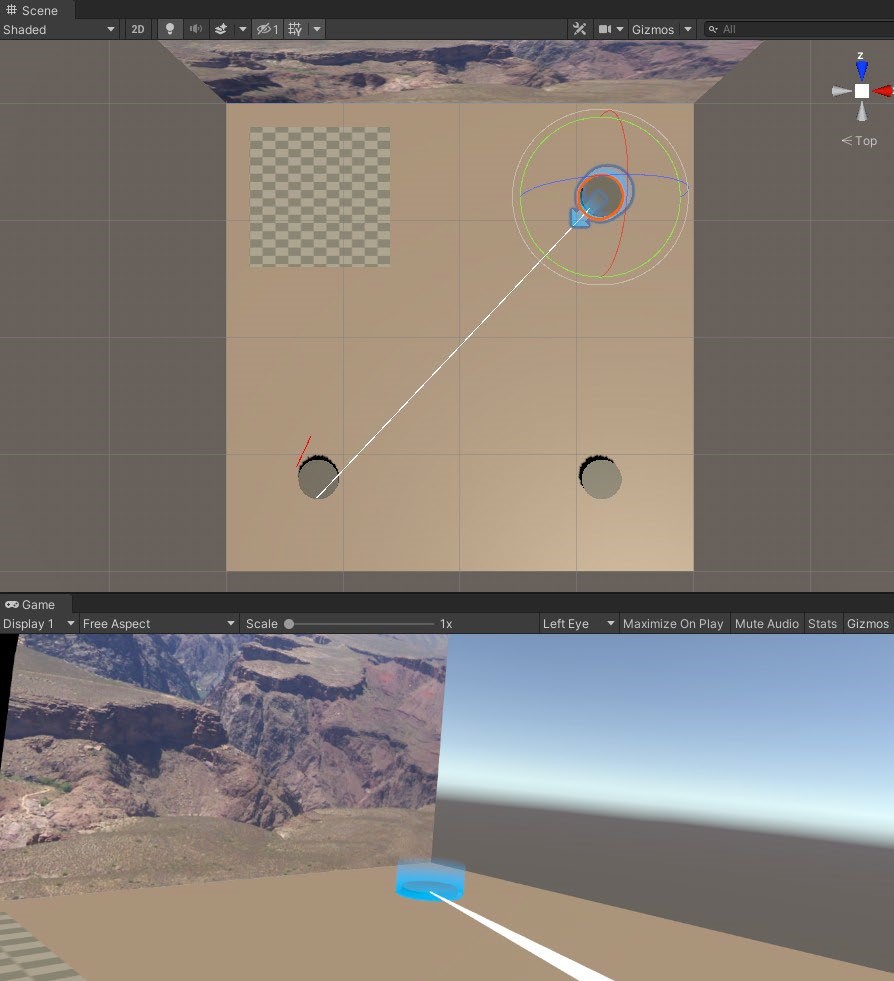
use

**Layer 31**

, named Spatial

Awareness or Teleport by XRI. If not, you can create a

layer now, using the following steps:



1.

In the

**Layer**

dropdown in the top-right of the

**Inspector**

, choose

**Add Layer...**

or

**Edit Layers...**

.

2.

This opens the

**Tags & Layers**

window. Unfold the

**Layers**

list.

3.

Choose an undefined layer slot. It can be

**User Layer 31**

or another one. Type the name

Teleport.

4.

Select the

**TeleportArea**

GameObject again in

**Hierarchy**

to restore its

**Inspector**

view.

5.

Set its

**Layer**

value to Teleport and its

**Teleport Area | Interaction Layer Mask**

,

**Ray interactors for Teleportation**

We were introduced to the

**XR Ray Interactor**

in Chapter 5

*, Interacting with your Hands*

, as it.

For teleportation use the Projectile Curve. For example, the following steps will render a projectile

curve that drops off near the end instead of a straight-line ray:

1.

Select the

**RightHand Controller**

in

**Hierarchy**

.

2.

In

**Inspector**

, set its

**Line Type**

to

**Projectile Curve**

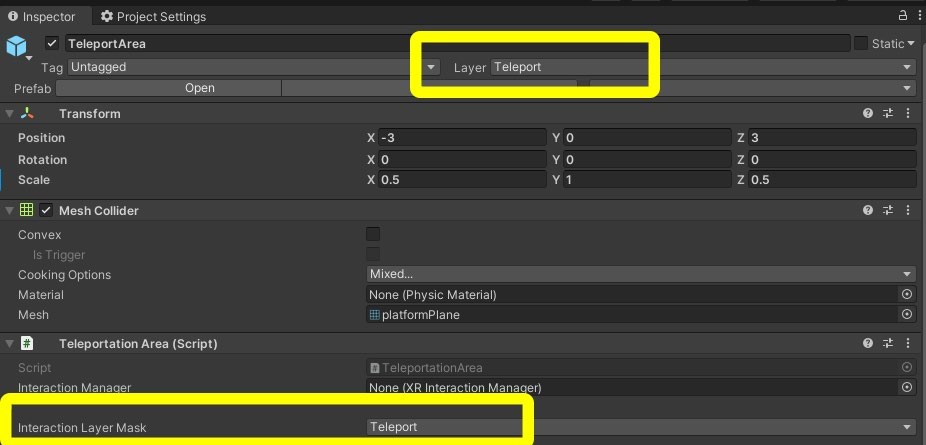
.

3.

Set its

**Velocity**

to 10.



**Switching between Interactors**

The

**XRRig\_Demo**

object includes a script named

ControllerManager

that allows the

player to switch between hand controllers with the press of a button.

mechanics related to locomotion, teleportation, VR comfort, and motion sickness.

END OF LAB 6

