**Lab 3: Using Gaze-Based Control**

# 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 3 Files are available on LZ

Let's make sure we have a scene with an XR camera rig by following these steps:

1. Open the Diorama screen we created earlier using **File | Open Scene** (or double- click the scene file in the **Project** window's Scenes/ folder).
2. Add an **XR Rig** using the XR-Rig Prefab.
3. Save the scene with a new name using **File | Save As**.

# Artificially intelligent Ethan

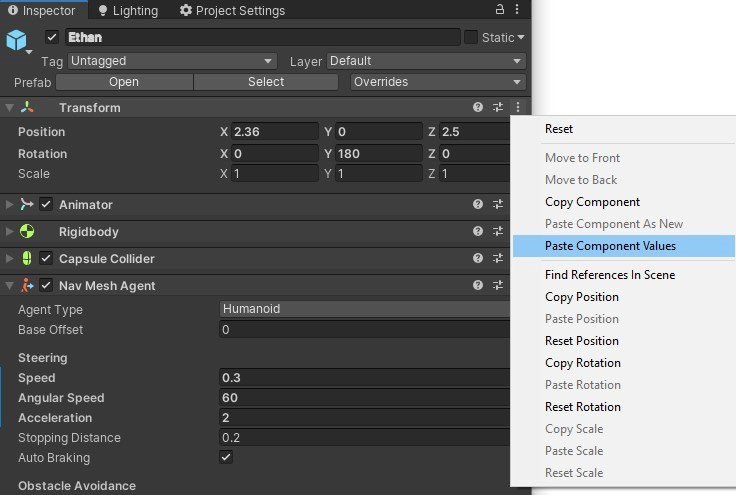
Replace the ThirdPersonController prefab that we used initially with Unity's AI character, AIThirdPersonController. We'll do this by inserting the AI version into the scene, copying the transform from the old character, and pasting those values into the new one. Follow these steps to do so:

1. Start by opening the Unity project and Diorama scene we created in Chapter 2,
2. *Understanding Unity, Content, and Scale*.
3. In the **Project** window, open the Standard Assets/Characters/

ThirdPersonCharacter/Prefabs folder and drag AIThirdPersonController into the scene. Name it Ethan.

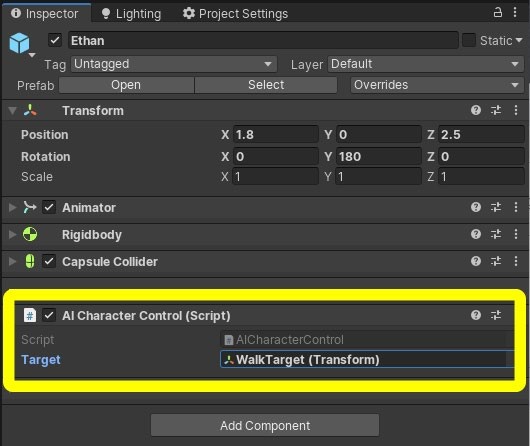
1. In the **Hierarchy** window, select the previous ThirdPersonController (the old Ethan), if present. Then, in **Inspector**, choose the three-dot icon to the upper right of the **Transform** panel and select **Copy Component**.
2. Select the new Ethan object from the **Hierarchy** and on its **Inspector Transform**, choose the three-dot icon and select **Paste Component Values**.
3. Now, you can delete the old ThirdPersonController object from the scene (in Hierarchy, **right-click | Delete**).

The **Transform** context menu where you can copy/paste the values that were used in these steps can be seen in the following screenshot:



The Ethan object has Nav Mesh Agent and AI Character Control components. Nav Mesh Agent has parameters for how Ethan will move around the scene. On the other hand, AI Character Control takes a target object where Ethan will walk to. Let's populate this, as follows:

1. Add an empty game object to the Hierarchy (**GameObject | Create Empty**) and rename it WalkTarget.
2. Reset its **Transform** values to position (0,0,0) (**Transform | right-click | Reset**).
3. Select Ethan and drag WalkTarget into the **Target** property in the **Inspector** panel's **AI Character Control** component, as shown here:



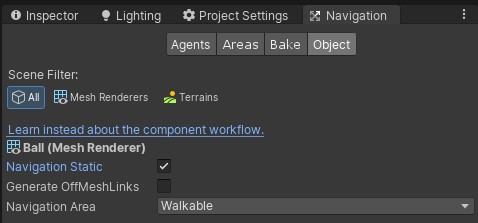
We now have an AI character in the scene (Ethan), an empty game object that will be initially used as a navigation target (WalkTarget) in the center of our scene, and we told AI Character Controller to use this target object. When we run the game, wherever WalkTarget is, Ethan will go there.

# The NavMesh

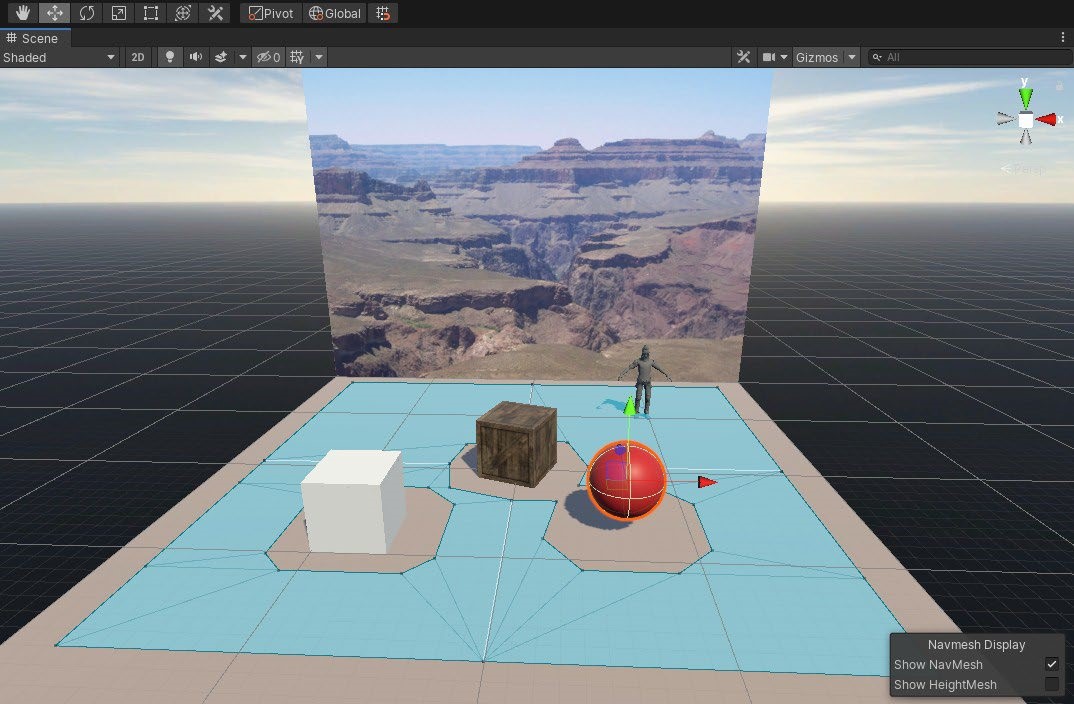
Ethan cannot just go walking around without being told where he's allowed to roam! We need to define a *navmesh*, a simplified geometrical plane that enables a character to plot its path around obstacles.

In our scene, Ethan is a navmesh agent (he has a Nav Mesh Agent component attached). The *navmesh* says where he's allowed to walk. We'll create a navmesh by marking the **static objects** in the scene that should **block navigation** and then **bakin**g the navmesh. To do this, follow these steps:

1. Open the **Navigation** window (**Window | AI | Navigation**). The **Navigation** window is shown in the following screenshot.
2. Select its **Object** tab.
3. Select **Ground Plane** in **Hierarchy**. Then, in the **Navigation** window, check the **Navigation Static** checkbox, as shown in the following screenshot. (Alternatively, you can use the object's **Inspector** window **Static** dropdown list.)
4. Repeat *step 3* for each of the objects that should get in his way; for example, in my scene, these are **Crate**, **Ball**, and **UnitCube**.
5. In the **Navigation** window, select the **Bake** tab.
6. Unfold the **Advanced** options and check the **Height Mesh** checkbox (this will prevent Ethan from hovering above the ground).
7. Then, click on the **Bake** button at the bottom of the panel:



The **Scene** view should now show a blue overlay where the navmesh is defined, as shown in the following screenshot:



To test this out, follow these steps:

1. Ensure that the **Game** window's **Maximize on Play** setting is not selected.
2. Click on the **Play** button (the triangle at the top of the editor).
3. In the **Hierarchy** window, select the WalkTarget object and ensure that the **Translate** gizmo is active in the **Scene** panel (you can use the *W* key shortcut on the keyboard).
4. Now, drag the red (**X-axis**) and/or the blue (**Z-axis**) arrow handles onto the WalkTarget object to move it around the floor plane. As you do, Ethan should follow!
5. Click on **Play** again to stop Play mode.

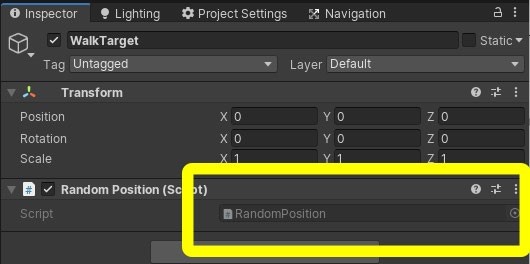
Now, we have Ethan, the nav agent, a navmesh defined in the scene, and a target that Ethan automatically walks toward. Next, we'll write a script that moves the WalkTarget object to random places.

# Scripting a Random Walk Target

Our goal is to make Ethan walk to random positions in the scene.

Attach the script to the WalkTarget object.

1. Select the WalkTarget object in the **Hierarchy** window.
2. In its **Inspector** panel, click on the **Add Component** button.
3. Select **New Script** (you may need to scroll down to find it).
4. Name it RandomPosition.
5. Click on **Create and Add**.
6. This should create a script component on the WalkTarget object. Double-click on the RandomPosition script slot, as shown here, and open it in your code editor (for example, Visual Studio):



We want to move the WalkTarget object to a random location so that Ethan will head in that direction, wait a few seconds, and move the WalkTarget object again.

The RandomPosition.cs script looks like this. Type or paste the following code into Visual Studio for the RandomPosition.cs file:

using System.Collections;

using UnityEngine;

public class RandomPosition : MonoBehaviour { void Start() {

StartCoroutine(RePositionWithDelay());

}

IEnumerator RePositionWithDelay() { while (true) {

SetRandomPosition();

yield return new WaitForSeconds(5f); }

}

void SetRandomPosition() { float x = Random.Range(-5.0f, 5.0f); float z = Random.Range(-5.0f, 5.0f);

Debug.Log("X,Z: " + x.ToString("F2") + ", " +

z.ToString("F2"));

transform.position = new Vector3(x, 0.0f, z); }

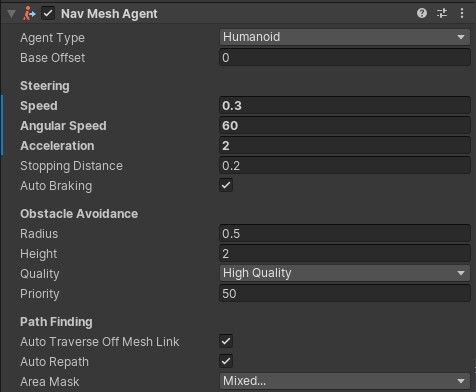
}

Save the script in Visual Studio (**File | Save**).

**"Zombie-izing" Ethan!**

Adjust the navmesh's steering parameters, as shown in the following steps:

1. Select Ethan in **Hierarchy**.
2. In the **Inspector** window, in the **Nav Mesh Agent** component's **Steering** parameters, set the following values:



Press the **Play** button again. He has slowed down. That's better. Ethan now walks like a zombie, but he doesn't really look like one yet:



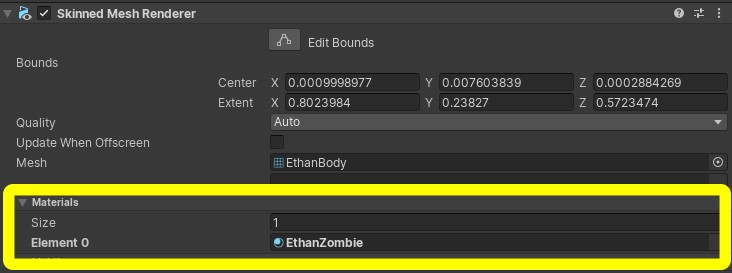
1. In the **Hierarchy** window, unfold the **Ethan** game object and select **EthanGlasses.**
2. In the **Inspector** window, uncheck the enable checkbox to disable and hide the glasses. Next, we'll give him some zombie skin.



**Adding a Zombie Material**

To give our Ethan zombie some decaying skin, let's make a copy of his body's material to work with. To do that, follow these steps:

1. In the **Hierarchy** window, select the **EthanBody** object.
2. In the **Inspector** window, click the material (for example, **EthanGray**), which can be found under **Skinned Mesh Renderer | Materials | Element 0**. That'll highlight the material file in the **Project** window.
3. In **Project**, duplicate the material (with the material file selected, press *Ctrl + D* on Windows, or *Cmd + D* on Mac).
4. Rename the copy EthanZombie.
5. Again, ensure **EthanBody** is selected in **Hierarchy**.
6. Next, drag the **EthanZombie** material from the **Project** window onto the **Materials | Element 0** slot, as shown in the following screenshot:



Adjust his body's base color from gray to white by following these steps:

1. In the **EthanBody** inspector, unfold the **EthanZombie** material.
2. Click the **Base Map** color swatch (right of the **Base Map** label).
3. Select white (FFFFFF).

Add a zombie face texture to Ethan's face. Use the texture image named EthanZombie.png that's included with this code:

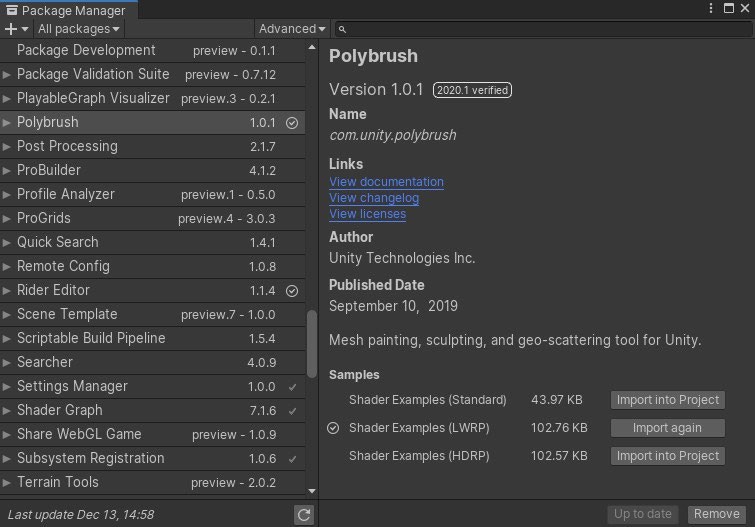
1. From the main menu, select **Assets | Import New Asset....**
2. Navigate to the files folder with the assets that came with this Lab.
3. Select the EthanZombie.png file and click **Import**. For tidiness, ensure that it resides in the Assets/Textures folder. (Alternatively, you can just drag and drop the file from Windows Explorer into the **Project** window.)
4. With **EthanBody** selected in **Hierarchy**, drag the **EthanZombie** texture from **Project** onto the **Base Map** texture chip (left of the **Base Map** label).



# Painting models with Polybrush

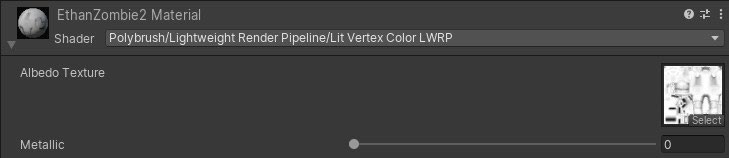
Install Polybrush into our project and import its shaders for the Render Pipeline we're using.:

1. Open Package Manager (**Window | Package Manager**).
2. Filter the list to **All Packages** (the drop-down list in the top left of the window).
3. Type polybrush into the search area and with the package selected, press **Install**.
4. Also, press the **Shader Examples (LWRP)'s Import into Project** button:



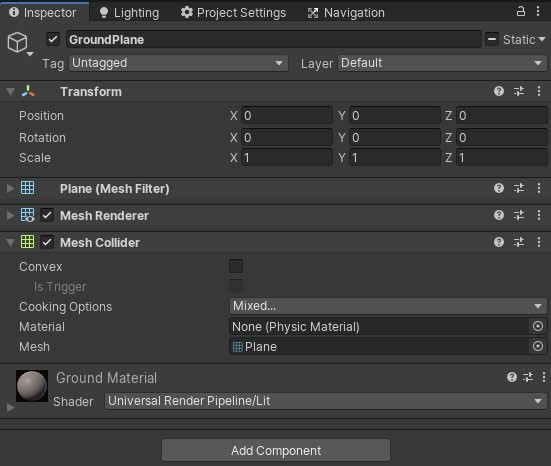
Edit the EthanBody game object's material. Use the EthanZombie.png texture image,

1. With **EthanBody** selected, set the material's **Shader** to **Polybrush | Lightweight Render Pipeline | Lit Vertex Color LWRP**.
2. Drag the **EthanZombie** texture onto the material's **Albedo Texture** chip:



1. Open the **Polybrush** window (**Tools | Polybrush | Polybrush Window**).
2. Ensure **EthanBody** is the currently selected object in **Hierarchy**.
3. In the **Polybrush** window, adjust the brush radius to **Outer Radius**: 0.01 and **Inner Radius**: 0.
4. Select the middle tab icon for **Paint Vertex Colors on Meshes**.
5. Select a paint color.
6. Begin painting on the model.

10



**Going Where I'm looking at**

For our next objective, instead of being random, we'll send Ethan to wherever you look. As you know, the VR camera object in your scene moves with your head.

# The LookMoveTo script

1. Select the WalkTarget object in **Hierarchy**.
2. In its **Inspector**, click on the **Add Component** button and select **New Script** called LookMoveTo.
3. Click on **Create And Add**.
4. Disable or remove the **RandomPosition** component by unchecking its checkbox, or **rightclick | Remove Component**.

using UnityEngine;

using System.Collections;

public class LookMoveTo : MonoBehaviour { public GameObject ground;

void Update() {

Transform camera = Camera.main.transform; Ray ray;

RaycastHit hit; GameObject hitObject;

Debug.DrawRay(camera.position, camera.rotation \*

Vector3.forward \* 100.0f);

ray = new Ray(camera.position, camera.rotation \*

Vector3.forward); if (Physics.Raycast(ray, out hit)) { hitObject = hit.collider.gameObject;

if (hitObject == ground) {

Debug.Log("Hit (x,y,z): " + hit.point.ToString("F2")); transform.position = hit.point;

}

}

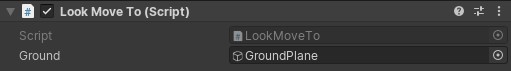
}

}

Save the script

1. With WalkTarget selected, in **Inspector**, the LookMoveTo script component now has a field for the GroundPlane object.
2. From the **Hierarchy** window, select and drag the GroundPlane game object onto the **Ground** field.

**Save** the scene



Then, click the **Play** button. Ethan should follow our gaze (at his own pace).

Modify your script first by adding the following code:

private Transform camera;

void Start() { camera = Camera.main.transform;

}

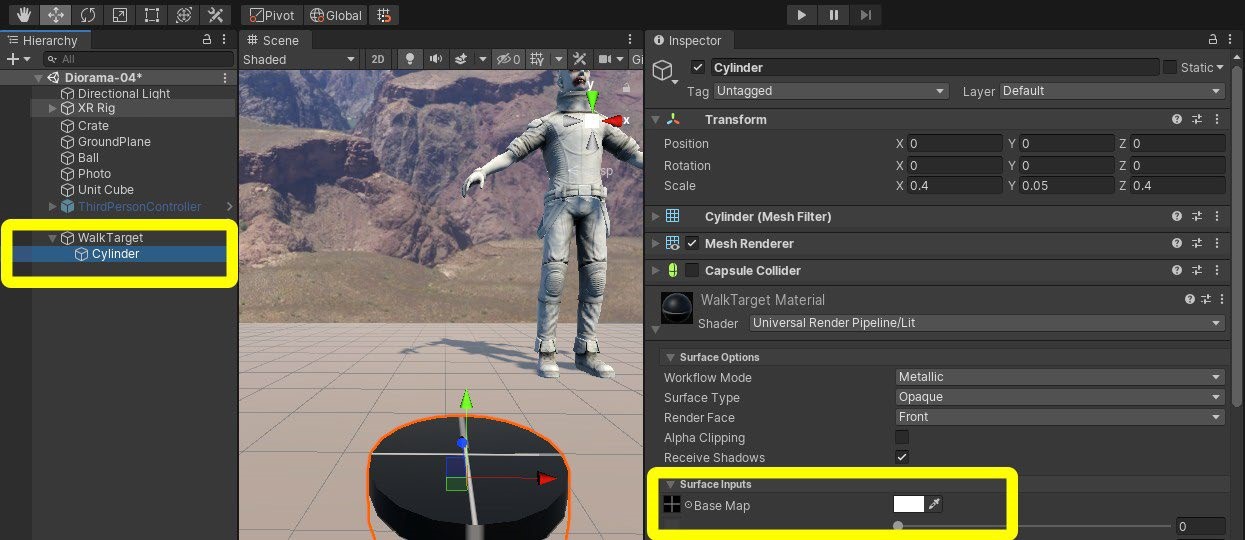
Remove the line of code that declares camera as a local variable in Update

# Adding a feedback cursor object

Add a cursor to the scene.

1. In **Hierarchy**, right-click the **WalkTarget** object and select **3D Object | Cylinder**. This will create a cylindrical object parented by WalkTarget. Rename it CursorDisk.
2. Reset its transform (in **Inspector**, **right-click Transform | Reset**).
3. Change its **Scale** to (0.4, 0.05, 0.4). This will create a flat disk with a diameter of 0.4.
4. Disable its **Capsule Collider** by unchecking that checkbox. We don't need to use its physics.
5. As a performance optimization, in **Mesh Renderer**, you can also disable **Cast Shadows**, **Light Probes**, and **Reflection Probes** (set them to **Off**).

Now, click **Play** again. You will see that the cursor disk follows your gaze.



So far, we've got Ethan to follow where we're looking by moving the **WalkTarget** object to a position on **GroundPlane** that's determined by raycasting from the camera and seeing where it intersected that plane.

# Observing through Obstacles

You may have noticed that the cursor seems to get *stuck* when we slide our gaze over other solid objects, such as the crate and ball. That's because the **physics engine** has determined which object is hit first, never getting to the ground plane.

Replace the body of Update() with the following code:

void Update()

{

Transform camera = Camera.main.transform;

Ray ray;

RaycastHit[] hits;

GameObject hitObject;

Debug.DrawRay(camera.position, camera.rotation \*

Vector3.forward \* 100.0f);

ray = new Ray(camera.position, camera.rotation \*

Vector3.forward);

hits = Physics.RaycastAll(ray);

for (int i = 0; i < hits.Length; i++)

{

RaycastHit hit = hits[i];

hitObject = hit.collider.gameObject; if (hitObject == ground)

{

Debug.Log("Hit (x,y,z): " + hit.point.ToString("F2")); transform.position = hit.point;

}

}

}

On calling RaycastAll, we get back a list of hits. Then, we loop through each one looking for a ground hit anywhere along the path of the ray vector. When you press **Play**, our cursor will trace along the ground, regardless of whether there's another object in between.

# Making a look-to-kill system

* Looking at Ethan hits him with our line-of-sight laser gun.
* Sparks are emitted when the gun hits its target.
* After 3 seconds of being hit, Ethan is killed.
* When he's killed, Ethan explodes and then he respawns at a new location.

write a script, named KillTarget, which will kill Ethan when the user stares at him. Then, we'll add some visual effects, using the Unity particle system, to show when Ethan is being hit and killed.

# The KillTarget Script

Attach the script to a new empty **GameManager** object by performing the following steps:

1. Create an empty game object and name it GameManager.
2. Using **Add Component**, attach a new C# script to it named KillTarget.

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

public class KillTarget : MonoBehaviour { public GameObject target; public ParticleSystem hitEffect; public GameObject killEffect; public float timeToSelect = 3.0f; public int score;

Transform camera;

private float countDown;

void Start() { camera = Camera.main.transform; score = 0;

countDown = timeToSelect;

}

void Update()

{ bool isHitting = false;

Ray ray = new Ray(camera.position, camera.rotation \* Vector3.forward);

RaycastHit hit;

if (Physics.Raycast(ray, out hit))

{ if (hit.collider.gameObject == target)

{ isHitting = true;

}

}

if (isHitting)

{ if (countDown > 0.0f)

{

// on target

countDown -= Time.deltaTime; // print (countDown); hitEffect.transform.position = hit.point; if (hitEffect.isStopped)

{ hitEffect.Play();

}

} else

{

// killed

Instantiate(killEffect, target.transform.position, target.transform.rotation);

score += 1;

countDown = timeToSelect; SetRandomPosition();

}

} else

{

// reset

countDown = timeToSelect; hitEffect.Stop();

}

}

void SetRandomPosition() { float x = Random.Range(-5.0f, 5.0f); float z = Random.Range(-5.0f, 5.0f);

target.transform.position = new Vector3(x, 0.0f, z); }

}

Save the script and open the Unity Editor.

# Adding Particle Effects

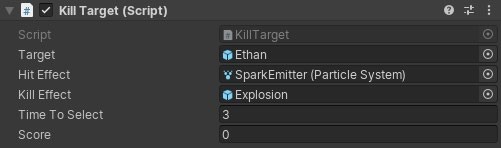
1. Select **GameManager** from **Hierarchy** and go to the **Kill Target (Script)** pane in **Inspector**.
2. Drag the **Ethan** object from **Hierarchy** onto the **Target** field.
3. From the main menu bar, navigate to **GameObject** | **Effects** | **Particle System** and name it SparkEmitter.
4. Reselect **GameManager** and drag **SparkEmitter** onto the **Hit Effect** field.
5. Select **SparkEmitter** from the **Hierarchy** panel. You'll see it playing in the **Scene** window.
6. In its **Inspector**, under **Particle System**, set the following values:
   * **Start Size**: 0.15
   * **Start Color**: Pick a red/orange color
   * **Start Lifetime**: 0.3
   * **Start Speed**: 0.3
   * **Max Particles**: 50
7. Under **Emission**, set **Rate over Time**: 100
8. Under **Shape**, set **Shape**: **Sphere** and **Radius**: 0.01

This will spray a bunch of fire-colored particles at the hit location when Ethan is hit. 1. In the **Project** window, find the Explosion prefab in Standard



Assets/ParticleSystems/Prefabs.

2. Drag the Explosion prefab onto the **GameManager | Kill Target | Kill Effect** field.



When Ethan is shot, the hitEffect particle system is activated. After 3 seconds (or whatever value you set in the TimeToSelect variable), his *health* is depleted, the explosion effect is instantiated, the score is incremented, and he respawns at a new location