COMP 4701
Introduction to Game Development

Işık University

Basic AI and Enemy Behavior

Programming an intelligent enemy is no easy task, and often goes hand in hand with long working hours and frustration.

However, Unity has **built-in features**, components, and classes we can use to design and implement AI systems in a more user-friendly way.

In this chapter, we'll focus on the following topics:

- The Unity navigation system
- Working with a navigation mesh
- Navigation agents
- Procedural programming and logic
- Taking and dealing damage
- Adding a loss condition
- Refactoring and keeping it DRY

Contents

- Navigating 3D space in Unity
- Moving enemy agents
- Enemy game mechanics
- Refactoring and keeping it DRY

Navigating 3D space in Unity

When we talk about navigation in real life, it's about how to get from point A to point B.
Navigating around virtual 3D space is largely the same, but how do we account for the experiential knowledge we humans have accumulated since the day we first started crawling?

Everything from walking on a flat surface to climbing stairs and jumping off of curbs is a skill we learned by doing; how can we possibly program all that into a game?

We'll learn what navigation components Unity has to offer.

Navigation components

Unity has its navigation system and delivering components that we can use to govern for playable and non-playable characters.
The following components are included with the new Unity AI Navigation package and has complex features already built in:

• A **NavMeshSurface** is essentially a map of the walkable surfaces in a given level; the NavMeshSurface component itself is created from the level geometry in a process called baking. Baking a NavMeshSurface into your level creates a unique project asset that holds the navigation data.

• If a **NavMeshSurface** is the level map, then a **NavMeshAgent** is the moving piece on the board. Any object with a NavMeshAgent component attached will automatically avoid other agents or obstacles it comes into contact with.

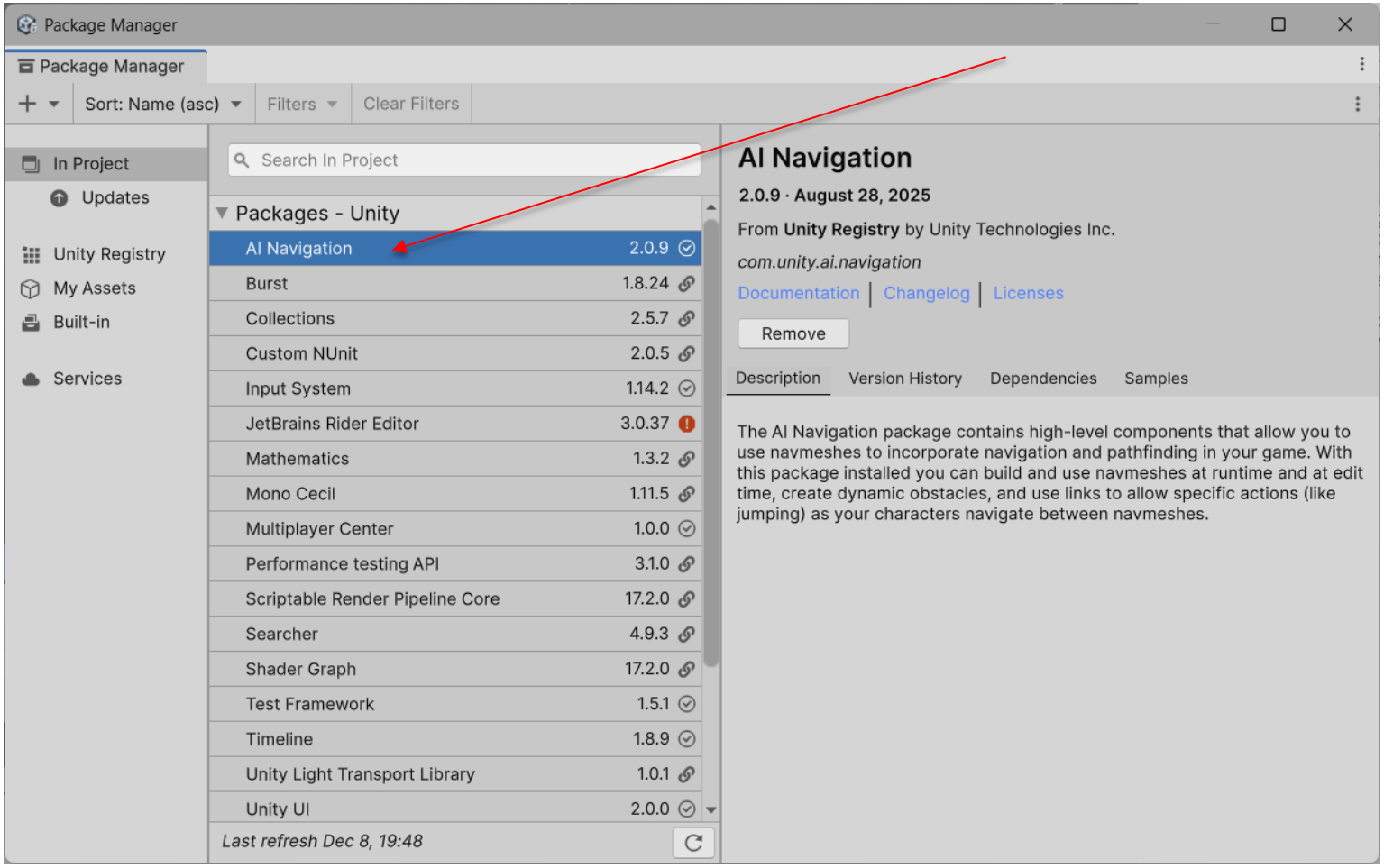
• The navigation system needs to be aware of any moving or stationary objects in the level that could cause a NavMeshAgent to alter its route. Adding **NavMeshObstacle** components to those objects lets the system know that they need to be avoided.

We'll be focusing on adding a **NavMeshSurface** to our level, setting up the **Enemy** Prefab as a **NavMeshAgent**, and getting the **Enemy** Prefab to move along a predefined route in a seemingly intelligent way.

You can find more information on the new Navigation system at https://docs.unity3d.com/Packages/com.unity.ai.navigation@1.1/manual/NavigationSystem.html

Check the AI Navigation package whether it is installed:

1. Go to **Window** > **Package Management** > **Package Manager**

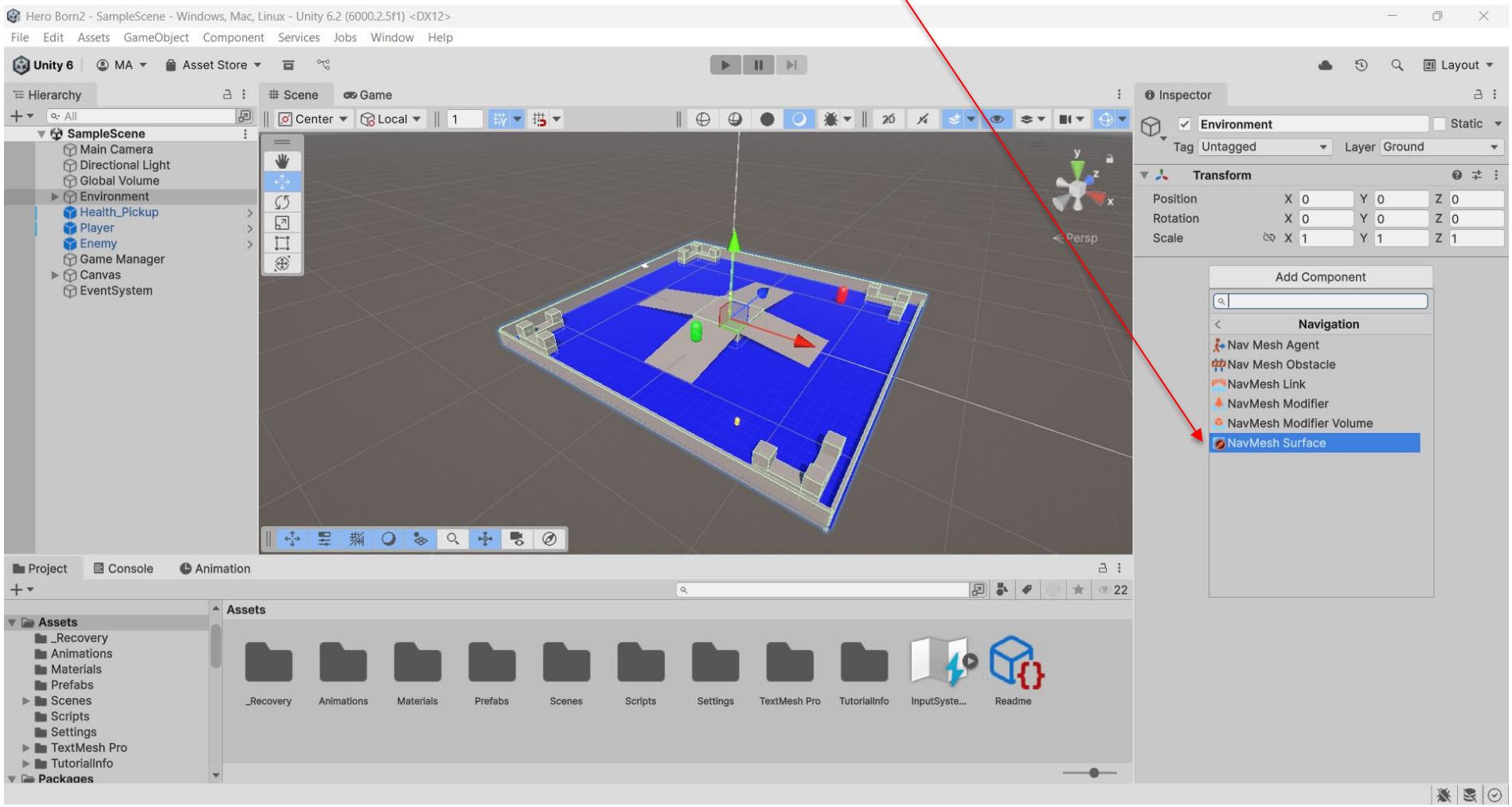


2. If it is not installed, Click the **+ sign** in the upper-left corner and select **Add package by name**:

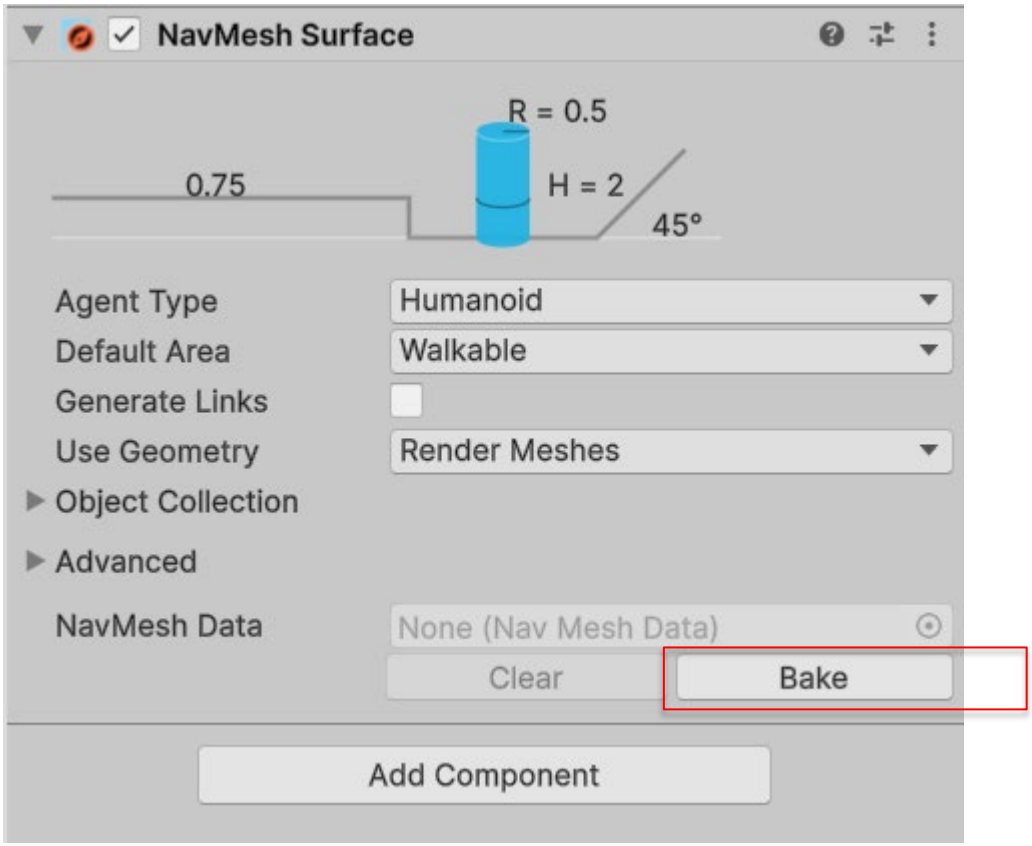
3. Enter `com.unity.ai.navigation` and click **Add**:

Our first task in setting up an “intelligent” enemy is to create a **NavMeshSurface** over the arena’s walkable areas. Let’s set up and configure our level’s NavMeshSurface:

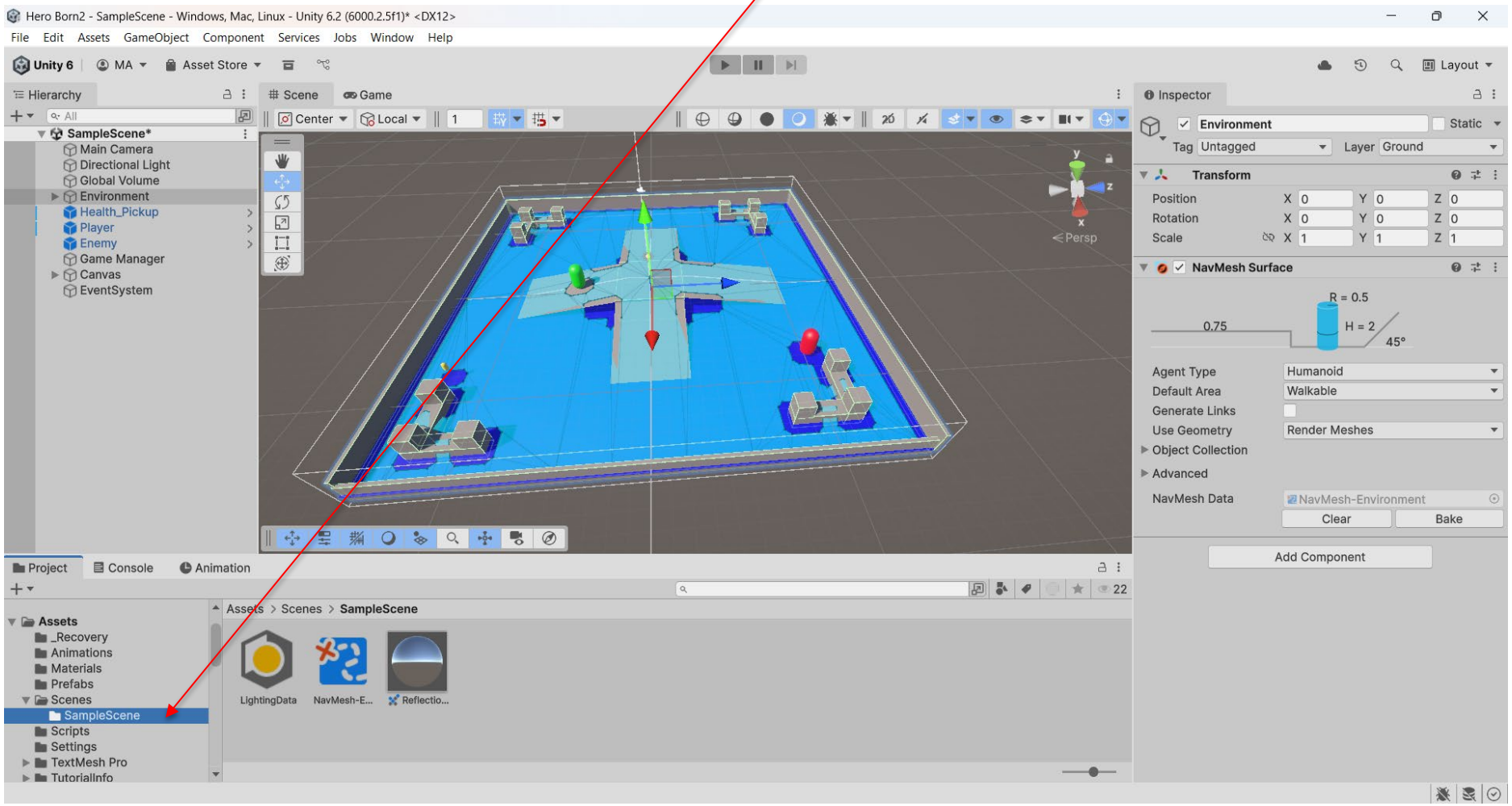
1. Select the **Environment** GameObject, click on **Add Component** in the **Inspector** window, and choose **Navigation** > **NavMeshSurface**:



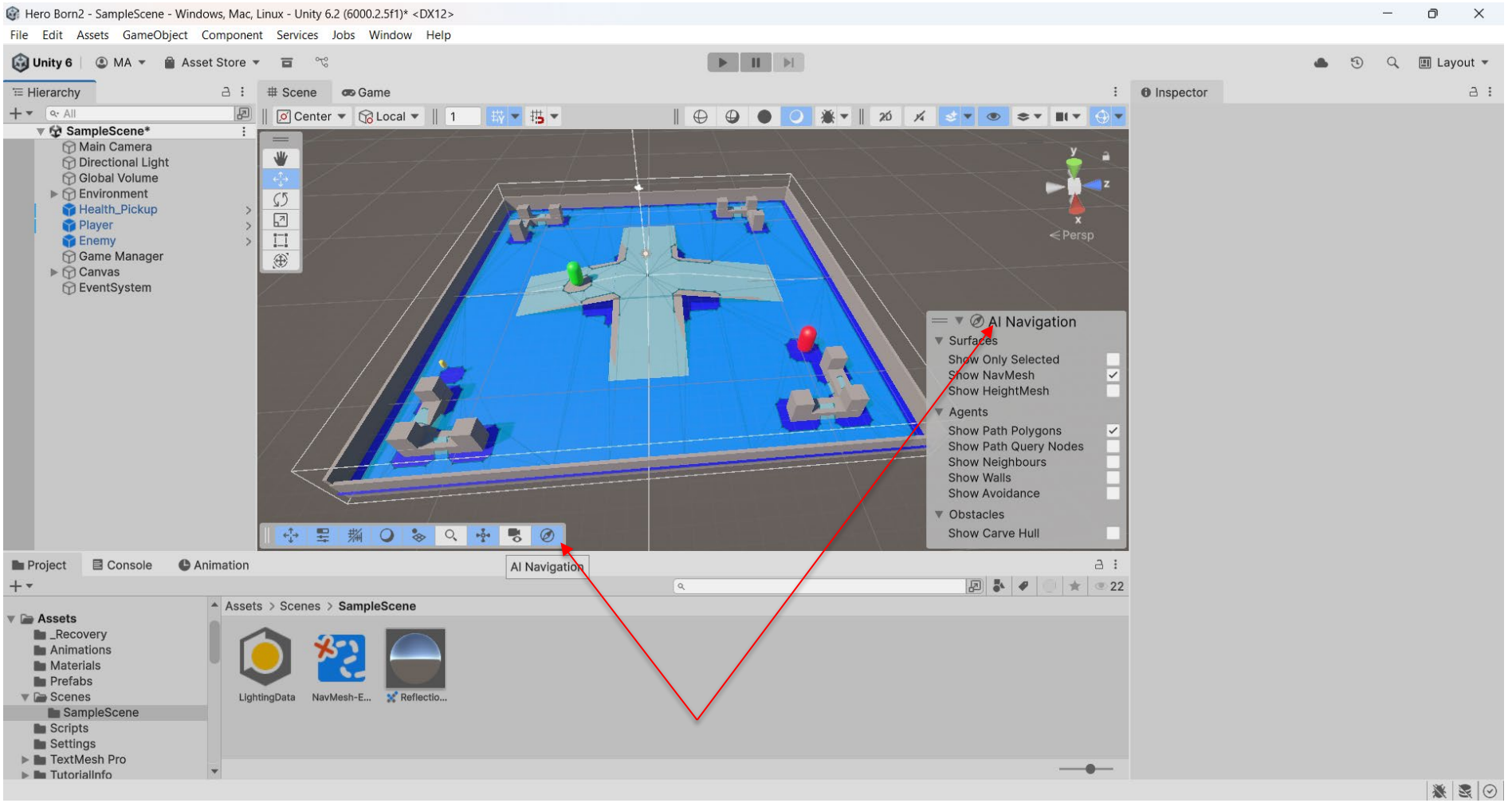
2. In the **Inspector**, leave everything set to the default values and click **Bake** in the **NavMeshSurface** component:



3. Once baking is finished, you'll see a new **SampleScene** folder inside the **Scenes** folder with our new navigation mesh data:



Our newly baked **NavMeshSurface** is now setup for us to interact with and a **light blue** overlay in has been added to show all the walkable surfaces a **NavMeshAgent** component attached:

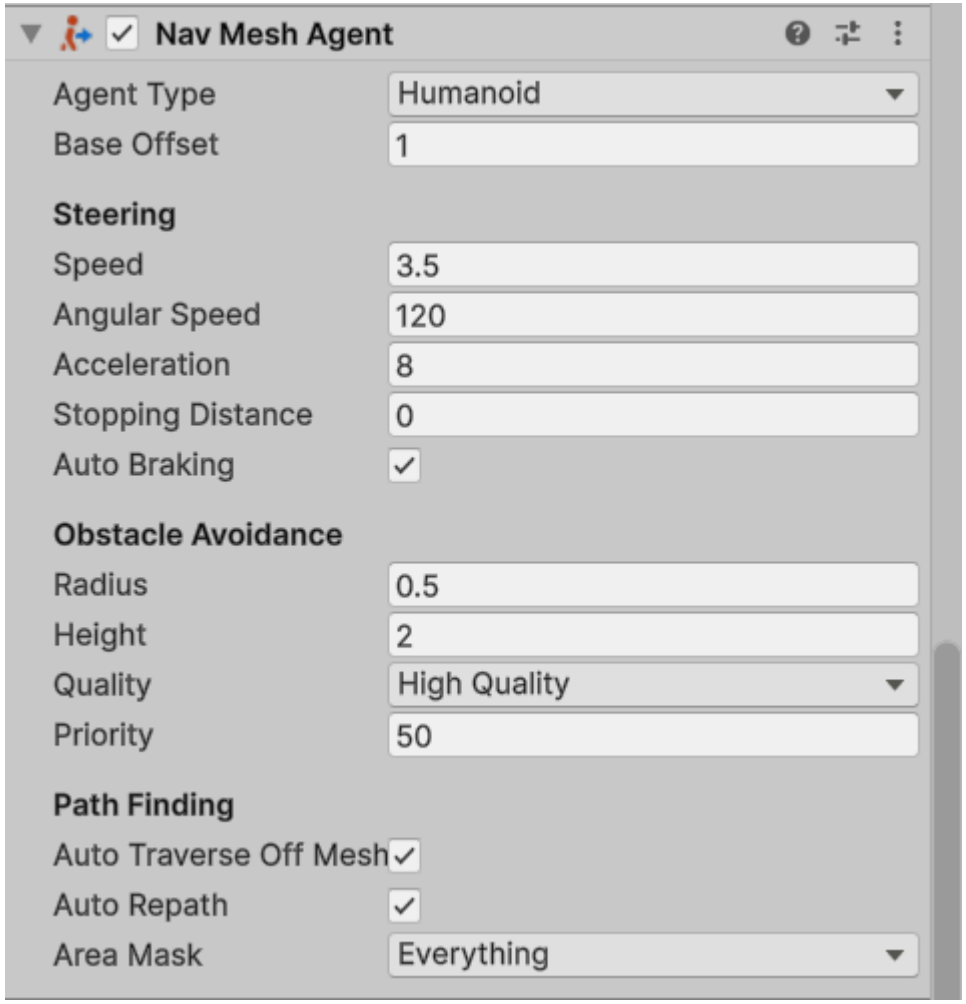


The **NavMeshSurface** is covering the entire surface. Our next task is to add get the **Enemy** walking around using the built-in navigation system!

Setting up enemy agent

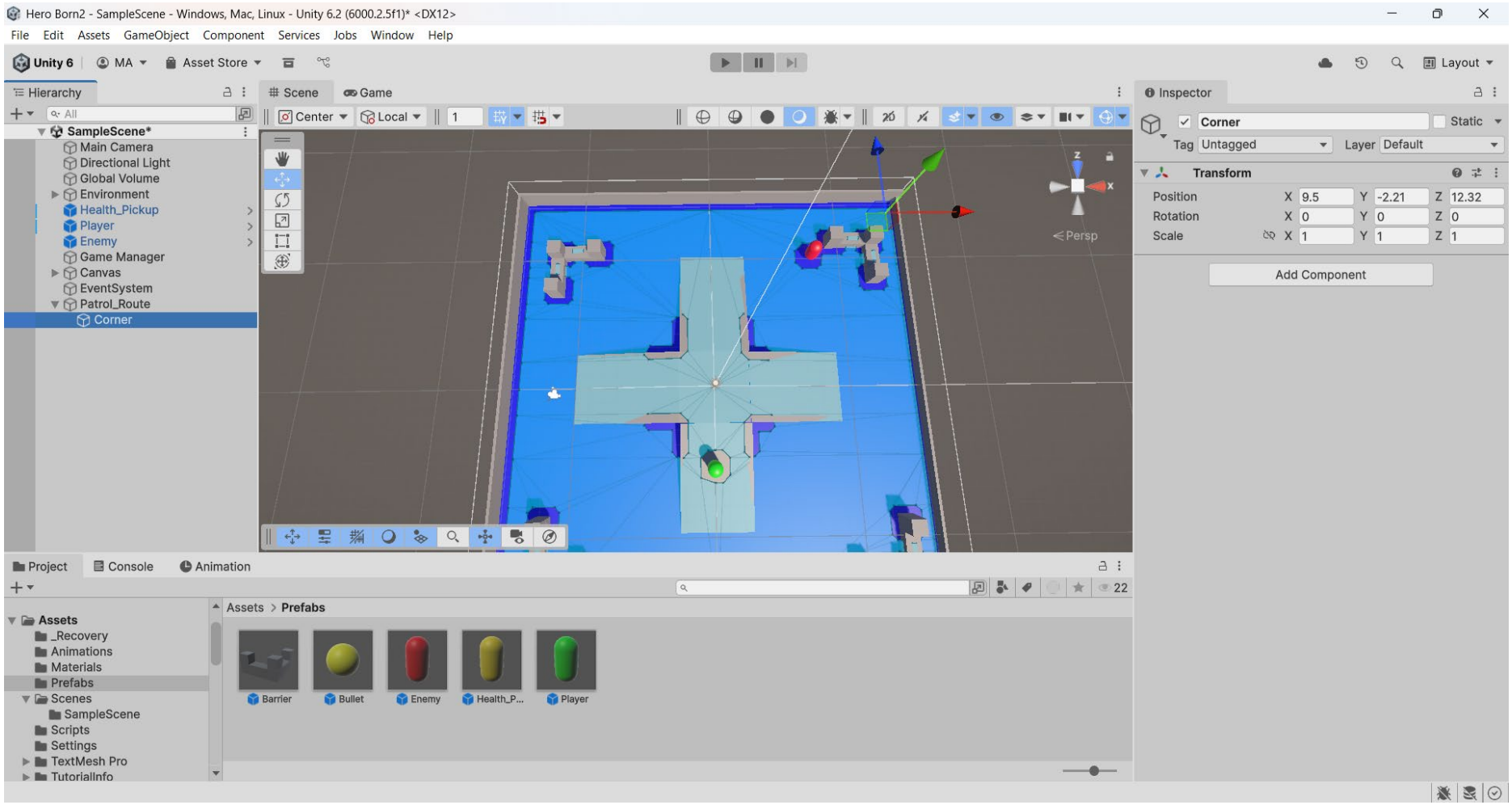
Let's register the **Enemy** Prefab as a **NavMeshAgent**:

1. In the **Prefabs** folder, select the **Enemy** Prefab, click **Add Component** in the **Inspector** window, and search for **Navigation** > **NavMeshAgent**:

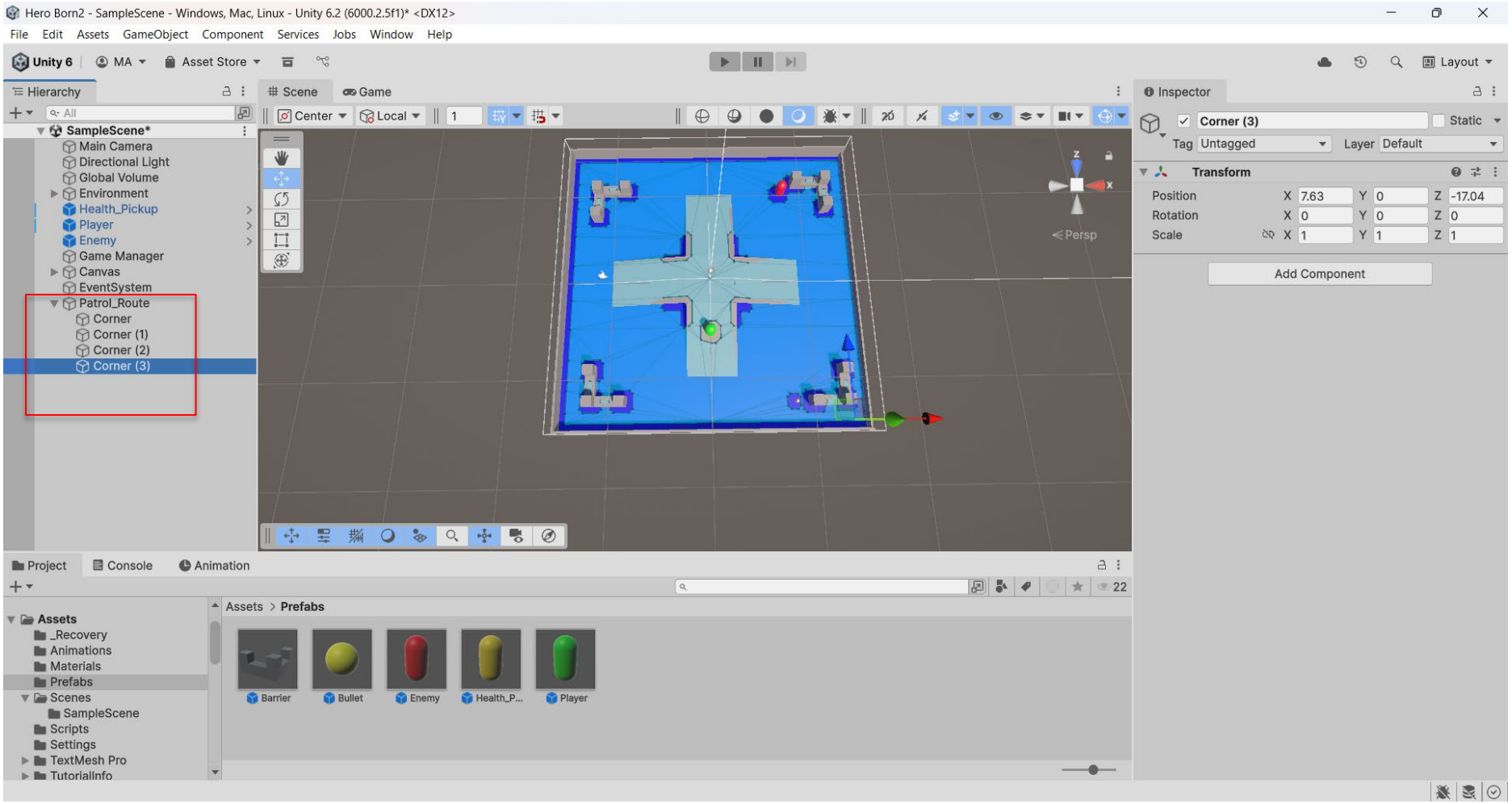


2. Click **+** > **Create Empty** from the **Hierarchy** window and name the GameObject `Patrol_Route`:

• Select `Patrol_Route`, click **+** > **Create Empty** to add a child GameObject, and name it `Corner`. Position `Corner` in one of the corners of the level —make sure there's enough room between the barriers and the wall on each side to let the enemy walk past:



3. Duplicate `Corner` to create three more empty child objects in `Patrol_Route` and position them in the remaining corners of the level to form a square:



Adding a **NavMeshAgent** component to the **Enemy** tells the **NavMeshSurface** component to take notice and register it as an object that has access to its autonomous navigation features.

Creating the four empty GameObjects in each corner of the level lays out the simple route we want our enemies to eventually patrol; grouping them in an empty parent object makes it easier to reference them in code and makes for a more organized **Hierarchy** window.

All that's left is the code to make the enemy walk the patrol route, which you'll add in the next section.

Moving enemy agents

Our patrol locations are set and the **Enemy** Prefab has a **NavMeshAgent** component, but now we need to figure out how to reference those locations and get the enemy moving on its own.
To do that, we'll first need to talk about an important concept in the world of software development: procedural programming.

Procedural programming

Any task that executes the same logic on one or more sequential objects is the perfect candidate for procedural programming.
You already did a little procedural programming when you debugged arrays, lists, and dictionaries with `for` and `foreach` loops. Each time those looping statements were executed, you performed the same call to `Debug.Log()`, iterating over each item sequentially.

The idea now is to use that skill to get a more useful outcome.
One of the most common uses of procedural programming is adding items from one collection to another, often modifying them along the way. This works great for our purposes since we want to reference each child object in the `Patrol_Route` parent and store them in a list.

Referencing the patrol locations

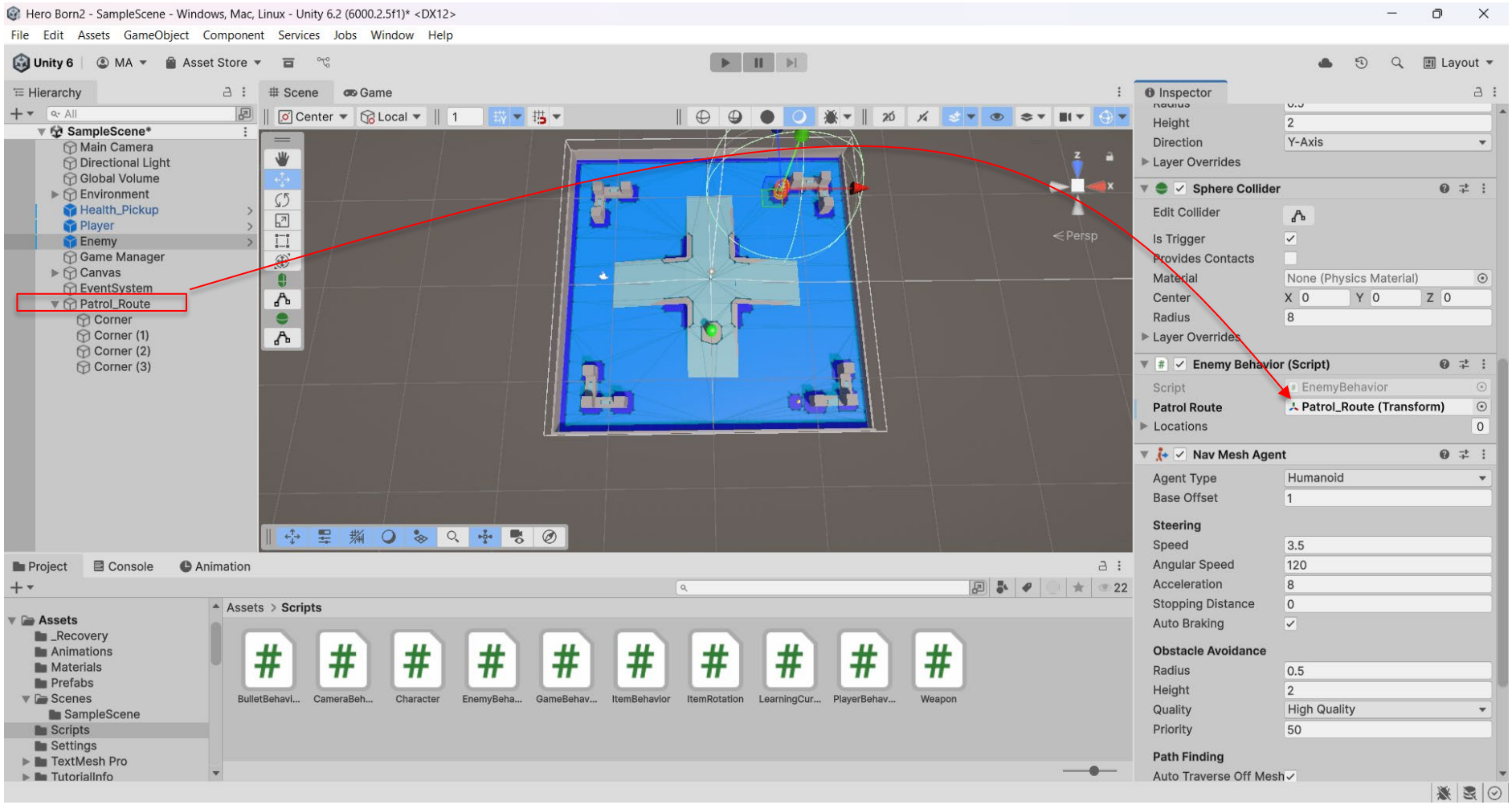
Now that we understand the basics of procedural programming, it's time to get a reference to our patrol locations and assign them to a usable list:

1. Add the following code to `EnemyBehavior`:

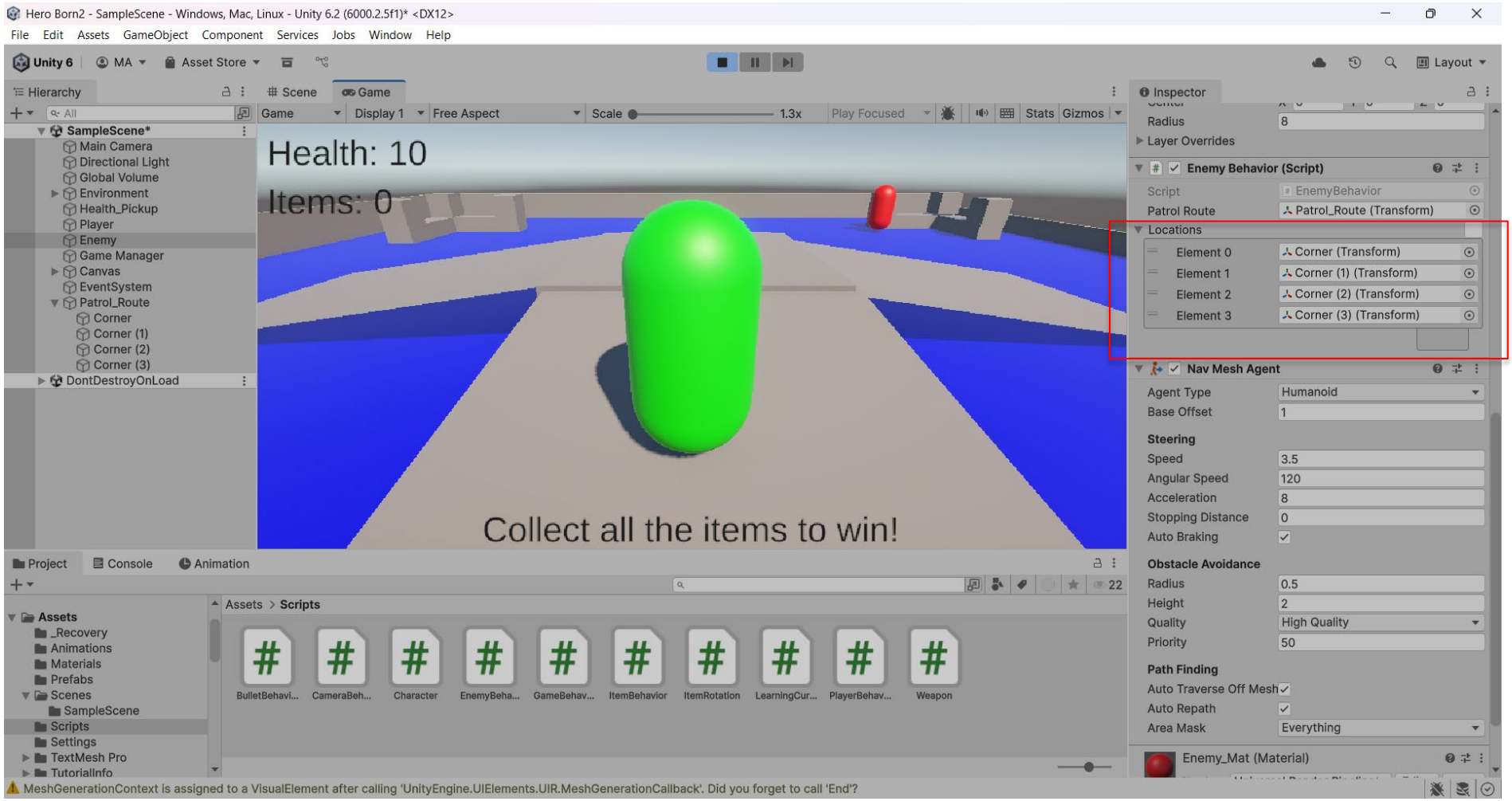
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (1 asset reference) | 0 references
public class EnemyBehavior : MonoBehaviour
{
    //1
    public Transform PatrolRoute;
    //2
    public List<Transform> Locations;

    Unity Message | 0 references
    private void Start()
    {
        //3
        InitializePatrolRoute();
    }

    //4
    1 reference
    void InitializePatrolRoute()
    {
        //5
        foreach(Transform child in PatrolRoute)
        {
            //6
            Locations.Add(child);
        }
    }
```

```
    Unity Message | 0 references
    private void OnTriggerEnter(Collider other)
    {
        if(other.name == "Player")
        {
            Debug.Log("Player detected - attack!");
        }
    }

    Unity Message | 0 references
    private void OnTriggerExit(Collider other)
    {
        if(other.name == "Player")
        {
            Debug.Log("Player out of range, resume patrol");
        }
    }
}
```

2. Select `Enemy` in the **Hierarchy** window and drag the `Patrol_Route` object from the **Hierarchy** window onto the **Patrol Route** variable in `EnemyBehavior`:



3. Hit the arrow icon next to the **Locations** variable in the **Inspector** window and **run** the game to see the list populate:



Let’s break down the code:

```
//1
public Transform PatrolRoute;
//2
public List<Transform> Locations;

Unity Message | 0 references
private void Start()
{
    //3
    InitializePatrolRoute();
}
```

1. First, it declares a variable for storing the `Patrol_Route` empty parent GameObject.

2. Then, it declares a `List` variable to hold all the child `Transform` components in `Patrol_Route`.

3. After that, it uses `Start()` to call the `InitializePatrolRoute()` method when the game begins.

```
//4
1 reference
void InitializePatrolRoute()
{
    //5
    foreach(Transform child in PatrolRoute)
    {
        //6
        Locations.Add(child);
    }
}
```

4. Next, it creates `InitializePatrolRoute()` as a private utility method to procedurally fill `Locations` with `Transform` values:

- Remember that not including an access modifier makes variables and methods `private` by default

5. Then, we use a `foreach` statement to loop through each child GameObject in `PatrolRoute` and reference its `Transform` component:

- Each `Transform` component is captured in the local `child` variable declared in the `foreach` loop

```
//4
1 reference
void InitializePatrolRoute()
{
    //5
    foreach(Transform child in PatrolRoute)
    {
        //6
        Locations.Add(child);
    }
}
```

6. Finally, we add each sequential `child Transform` component to the list of locations using the `Add()` method as we loop through the child objects in `PatrolRoute`:

- This way, no matter what changes we make in the **Hierarchy** window, `Locations` will always be filled in with all the `child` objects under the `Patrol_Route` parent.

While we could have assigned each location GameObject to `Locations` by dragging and dropping them directly from the **Hierarchy** window into the **Inspector** window, it's easy to lose or break these connections; making changes to the location object names, object additions or deletions, or project updates can all throw a wrench in a class's initialization.

It's much safer, and more readable, to procedurally fill GameObject lists or arrays in the `Start()` method.

Due to that reasoning, I also tend to use `GetComponent()` in the `Start()` method to find and store component references attached to a given class instead of assigning them in the **Inspector** window.

However, in situations where components you're looking for might be in other child objects or nested in complex Prefabs, it may be easier to drag-and-drop the components directly in the **Inspector** window.

Moving the enemy

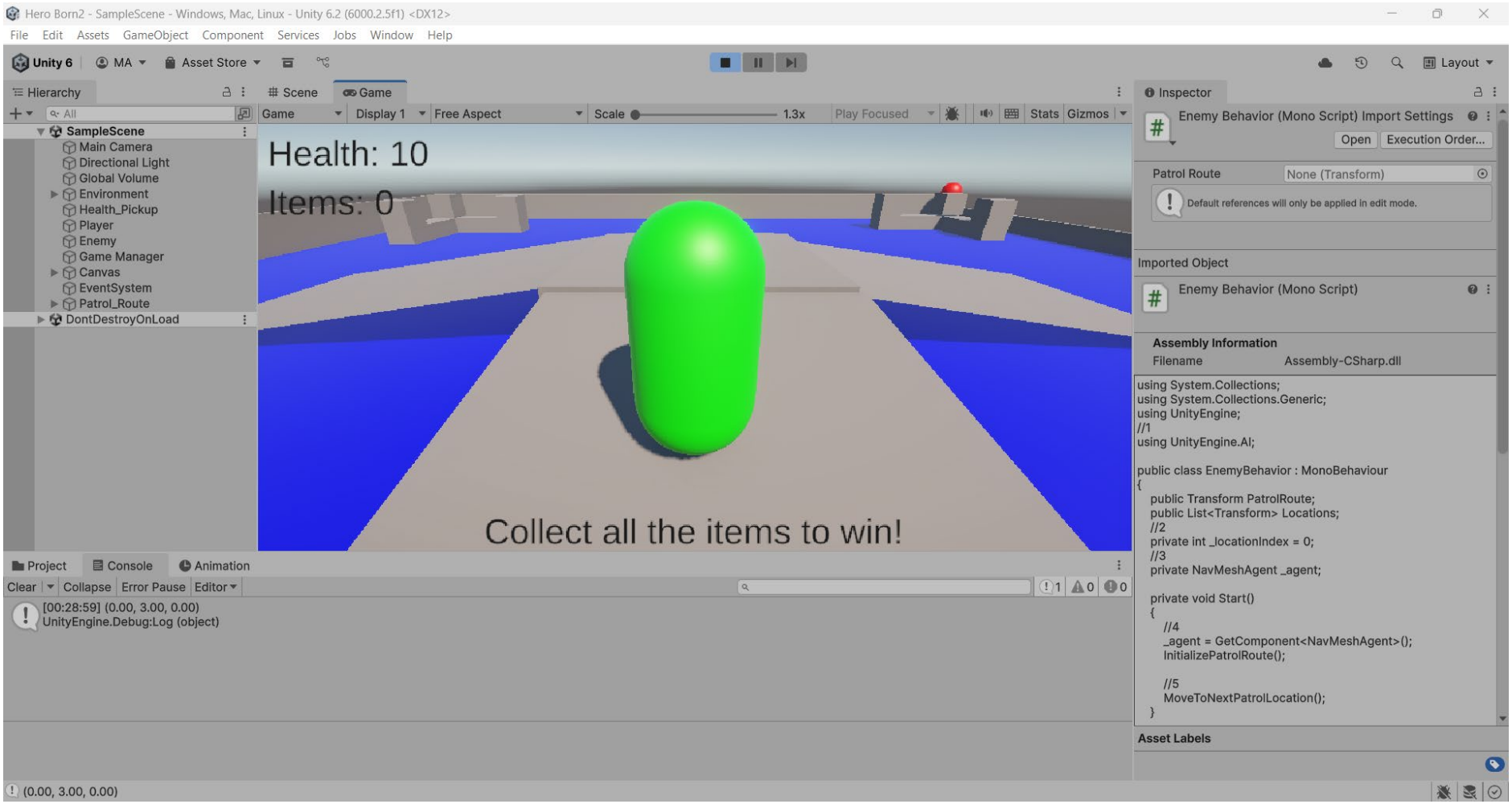
Now, we need the enemy object to follow the patrol route we laid out.

With a list of patrol locations initialized on `Start()`, we can grab the **Enemy** **NavMeshAgent** component and set its first destination.

Update `EnemyBehavior` with the following code and hit **Play**:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//1
using UnityEngine.AI;

Unity Script (1 asset reference) | 0 references
public class EnemyBehavior : MonoBehaviour
{
    public Transform PatrolRoute;
    public List<Transform> Locations;
    //2
    private int _locationIndex = 0;
    //3
    private NavMeshAgent _agent;

    Unity Message | 0 references
    private void Start()
    {
        //4
        _agent = GetComponent<NavMeshAgent>();
        InitializePatrolRoute();

        //5
        MoveToNextPatrolLocation();
    }

    1 reference
    void InitializePatrolRoute()
    {
        foreach(Transform child in PatrolRoute)
        {
            Locations.Add(child);
        }
    }
```

```
    1 reference
    void MoveToNextPatrolLocation()
    {
        //6
        _agent.destination = Locations[_locationIndex].position;
    }

    Unity Message | 0 references
    private void OnTriggerEnter(Collider other)
    {
        if(other.name == "Player")
        {
            Debug.Log("Player detected - attack!");
        }
    }

    Unity Message | 0 references
    private void OnTriggerExit(Collider other)
    {
        if(other.name == "Player")
        {
            Debug.Log("Player out of range, resume patrol");
        }
    }
}
```


Hero Born2 - SampleScene - Windows, Mac, Linux - Unity 6.2 (6000.2.5f1) <DX12>
File Edit Assets GameObject Component Services Jobs Window Help
Unity 6
MA
Asset Store
Layout
Hierarchy
SampleScene
Main Camera
Directional Light
Global Volume
Environment
Health_Pickup
Player
Enemy
Game Manager
Canvas
EventSystem
Patrol_Route
DontDestroyOnLoad
Scene
Game
Display 1
Free Aspect
Scale
1.3x
Play Focused
Stats
Gizmos
Health: 10
Items: 0
Collect all the items to win!
Inspector
Enemy Behavior (Mono Script) Import Settings
Open
Execution Order...
Patrol Route
None (Transform)
Default references will only be applied in edit mode.
Imported Object
Enemy Behavior (Mono Script)
Assembly Information
Filename
Assembly-CSharp.dll
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//1
using UnityEngine.AI;
public class EnemyBehavior : MonoBehaviour
{
public Transform PatrolRoute;
public List<Transform> Locations;
//2
private int _locationIndex = 0;
//3
private NavMeshAgent _agent;
private void Start()
{
//4
_agent = GetComponent<NavMeshAgent>();
InitializePatrolRoute();
//5
MoveToNextPatrolLocation();
}
Asset Labels
Project
Console
Animation
Clear
Collapse
Error Pause
Editor
[00:28:59] (0.00, 3.00, 0.00)
UnityEngine.Debug:Log (object)
(0.00, 3.00, 0.00)

Let's break down the code:

```
//1
using UnityEngine.AI;

Unity Script (1 asset reference) | 0 references
public class EnemyBehavior : MonoBehaviour
{
    public Transform PatrolRoute;
    public List<Transform> Locations;
    //2
    private int _locationIndex = 0;
    //3
    private NavMeshAgent _agent;
```

1. First, it adds the `UnityEngine.AI using` directive so that `EnemyBehavior` has access to Unity's navigation classes, in this case, `NavMeshAgent`.

2. Then, it declares a variable to keep track of which patrol location the enemy is currently walking toward. Since `List` items are zero-indexed, we can have the **Enemy** Prefab move between patrol points in the order they are stored in `Locations`.

3. Next, it declares a variable to store the **NavMeshAgent** component attached to the **Enemy** GameObject. This is `private` because no other classes should be able to access or modify it.

```
Unity Message | 0 references
private void Start()
{
    //4
    _agent = GetComponent<NavMeshAgent>();
    InitializePatrolRoute();

    //5
    MoveToNextPatrolLocation();
}
```

4. After that, it uses `GetComponent()` to find and return the attached **NavMeshAgent** component to the agent.

5. Then, it calls the `MoveToNextPatrolLocation()` method on `Start()`.

```
1 reference
void MoveToNextPatrolLocation()
{
    //6
    _agent.destination = Locations[_locationIndex].position;
}
```

6. Finally, it declares `MoveToNextPatrolLocation()` as a private method and sets `_agent.destination`:

- `destination` is a `Vector3` position in 3D space
- `Locations[_locationIndex]` grabs the `Transform` item in `Locations` at a given index
- Adding `.position` references the `Transform` component's `Vector3` position

Now, when our scene starts, locations are filled with patrol points and `MoveToNextPatrolLocation()` is called to set the destination position of the **NavMeshAgent** component to the first item at `_locationIndex 0` in the list of locations.

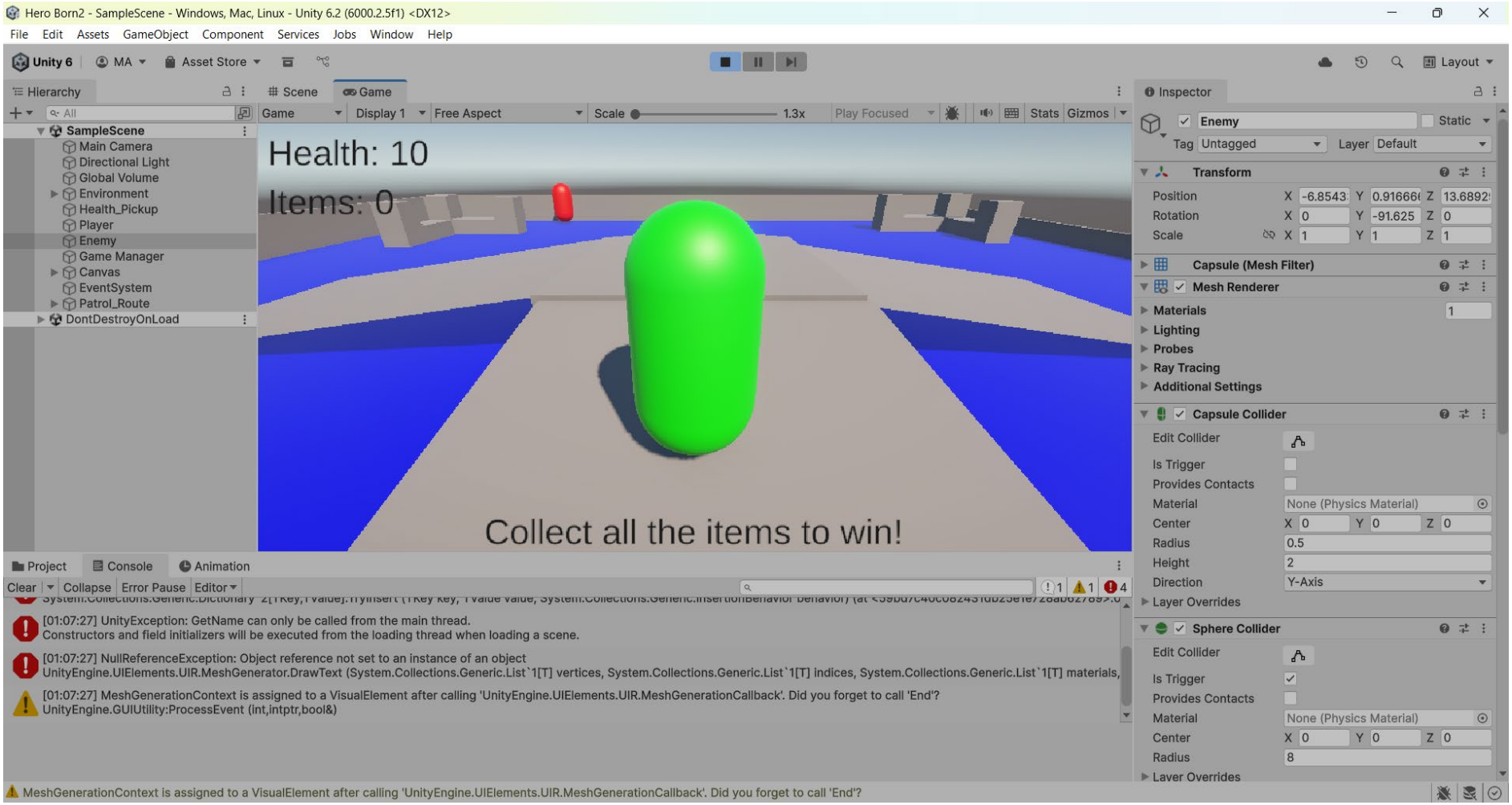
The next step is to have the enemy object move from the first patrol location to all the other locations in sequence.

Our enemy moves to the first patrol point just fine, but then it stops. What we want is for it to continually move between each sequential location, which will require additional logic in `Update()` and `MoveToNextPatrolLocation()`.
Let's create this behavior.

Add the following code to `EnemyBehavior` and hit **Play**:

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

Unity Script (1 asset reference) | 0 references
public class EnemyBehavior : MonoBehaviour
{
    public Transform PatrolRoute;
    public List<Transform> Locations;
    private int _locationIndex = 0;
    private NavMeshAgent _agent;

    Unity Message | 0 references
    private void Start()
    {
        _agent = GetComponent<NavMeshAgent>();
        InitializePatrolRoute();

        MoveToNextPatrolLocation();
    }

    Unity Message | 0 references
    private void Update()
    {
        //1
        if(_agent.remainingDistance < 0.2f && !_agent.pathPending)
        {
            //2
            MoveToNextPatrolLocation();
        }
    }
```

```
    1 reference
    void InitializePatrolRoute()
    {
        foreach(Transform child in PatrolRoute)
        {
            Locations.Add(child);
        }
    }

    2 references
    void MoveToNextPatrolLocation()
    {
        //3
        if (Locations.Count == 0)
            return;

        _agent.destination = Locations[_locationIndex].position;
        //4
        _locationIndex = (_locationIndex + 1) % Locations.Count;
    }

    Unity Message | 0 references
    private void OnTriggerEnter(Collider other)
    {
        if(other.name == "Player")
        {
            Debug.Log("Player detected - attack!");
        }
    }

    Unity Message | 0 references
    private void OnTriggerExit(Collider other)
    {
        if(other.name == "Player")
        {
            Debug.Log("Player out of range, resume patrol");
        }
    }
}
```


Hero Born2 - SampleScene - Windows, Mac, Linux - Unity 6.2 (6000.2.5f1) <DX12>
File Edit Assets GameObject Component Services Jobs Window Help
Unity 6
Hierarchy
SampleScene
Main Camera
Directional Light
Global Volume
Environment
Health_Pickup
Player
Enemy
Game Manager
Canvas
EventSystem
Patrol_Route
DontDestroyOnLoad
Scene
Game
Health: 10
Items: 0
Collect all the items to win!
Inspector
Enemy
Tag Untagged
Layer Default
Transform
Position
Rotation
Scale
Capsule (Mesh Filter)
Mesh Renderer
Materials
Lighting
Probes
Ray Tracing
Additional Settings
Capsule Collider
Edit Collider
Is Trigger
Provides Contacts
Material
Center
Radius
Height
Direction
Y-Axis
Layer Overrides
Sphere Collider
Project
Console
Animation
Clear
Collapse
Error Pause
Editor
[01:07:27] UnityException: GetName can only be called from the main thread.
Constructors and field initializers will be executed from the loading thread when loading a scene.
[01:07:27] NullReferenceException: Object reference not set to an instance of an object
UnityEngine.UIElements.UIR.MeshGenerator.DrawText (System.Collections.Generic.List`1[T] vertices, System.Collections.Generic.List`1[T] indices, System.Collections.Generic.List`1[T] materials,
[01:07:27] MeshGenerationContext is assigned to a VisualElement after calling 'UnityEngine.UIElements.UIR.MeshGenerationCallback'. Did you forget to call 'End'?
UnityEngine.GUIUtility:ProcessEvent (int,intptr,bool&)
MeshGenerationContext is assigned to a VisualElement after calling 'UnityEngine.UIElements.UIR.MeshGenerationCallback'. Did you forget to call 'End'?

Let's break down the code:

```
Unity Message | 0 references
private void Update()
{
    //1
    if(_agent.remainingDistance < 0.2f && !_agent.pathPending)
    {
        //2
        MoveToNextPatrolLocation();
    }
}
```

1. First, it declares the `Update()` method and adds an `if` statement to check whether two different conditions are true:

- `remainingDistance` returns how far the **NavMeshAgent** component currently is from its set destination, so we're checking if that is less than `0.2`

- `pathPending` returns a `true` or `false` Boolean, depending on whether Unity is computing a path for the **NavMeshAgent** component

2. If `_agent` is very close to its destination, and no other path is being computed, the `if` statement returns `true` and calls `MoveToNextPatrolLocation()`.

```
void MoveToNextPatrolLocation()
{
    //3
    if (Locations.Count == 0)
        return;

    _agent.destination = Locations[_locationIndex].position;
    //4
    _locationIndex = (_locationIndex + 1) % Locations.Count;
}
```

3. Here, we added an `if` statement to make sure that `Locations` isn't empty before the rest of the code in `MoveToNextPatrolLocation()` is executed:

- If `Locations` is empty, we use the `return` keyword to exit the method without continuing.

- `If` statements that only have one line of code can be written without any brackets, which can make them easier to write and read.

- This is referred to as **defensive programming**, and, coupled with refactoring, it is an essential skill to have in your arsenal as you move toward more intermediate C# topics. We will consider refactoring at the end of the chapter.

```
void MoveToNextPatrolLocation()
{
    //3
    if (Locations.Count == 0)
        return;

    _agent.destination = Locations[_locationIndex].position;
    //4
    _locationIndex = (_locationIndex + 1) % Locations.Count;
}
```

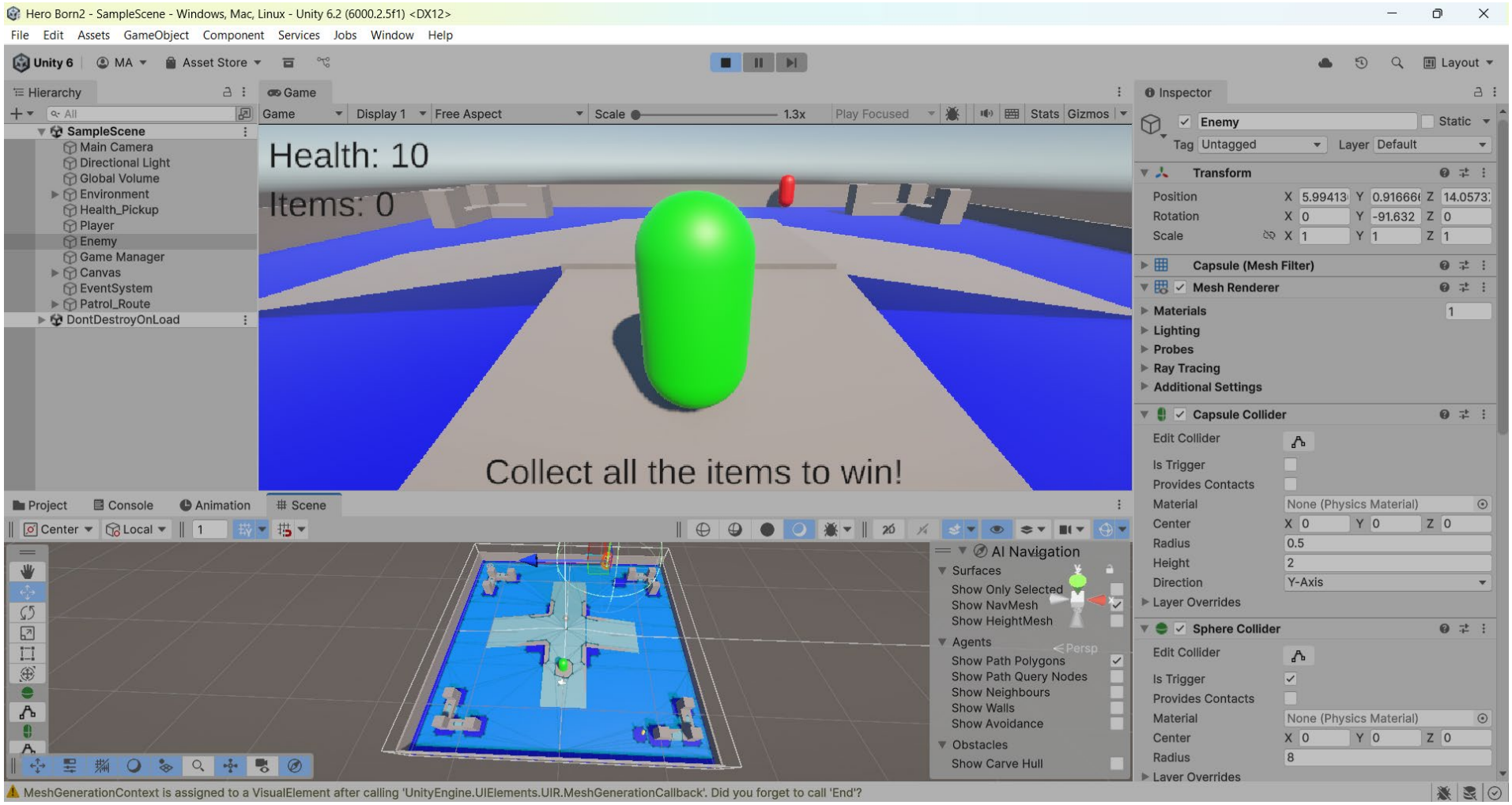
4. Then, we set `_locationIndex` to its current value, `+1`, followed by the modulo (`%`) of `Locations.Count`:

- This will increment the index from `0` to `4` and then restart it at `0` so that our **Enemy** Prefab moves in a continuous path.

- The modulo operator returns the remainder of two values being divided—2 divided by 4 has a remainder of 2 when the result is an integer, so 2 % 4 = 2. Likewise, 4 divided by 4 has no remainder, so 4 % 4 = 0.

We now need to check that the enemy is moving toward its set patrol location every frame in `Update()`; when it gets close, `MoveToNextPatrolLocation()` is fired, which increments `_locationIndex` and sets the next patrol point as the destination.

If you drag the **Scene** view down next to the **Console** window, as shown in the following screenshot, and hit **Play**, you can watch the **Enemy** Prefab walk around the corners of the level in a continuous loop:



The enemy now follows the patrol route around the outside of the map, but it doesn't seek out the player and attack when it's within a preset range. You'll use the **NavAgent** component to do just that in the next section.

Enemy game mechanics

Now that our enemy is on a continuous patrol circuit, it’s time to give it some interaction mechanics of its own; there wouldn’t be much risk or reward if we left it walking around with no way to act against us.

Seek and destroy: changing the agent's destination

We'll be focusing on switching the target of the enemies' **NavMeshAgent** component when the player gets too close and dealing damage if a collision occurs.
When the enemy successfully lowers the player's health, it will return to its patrol route until its next run-in with the player.

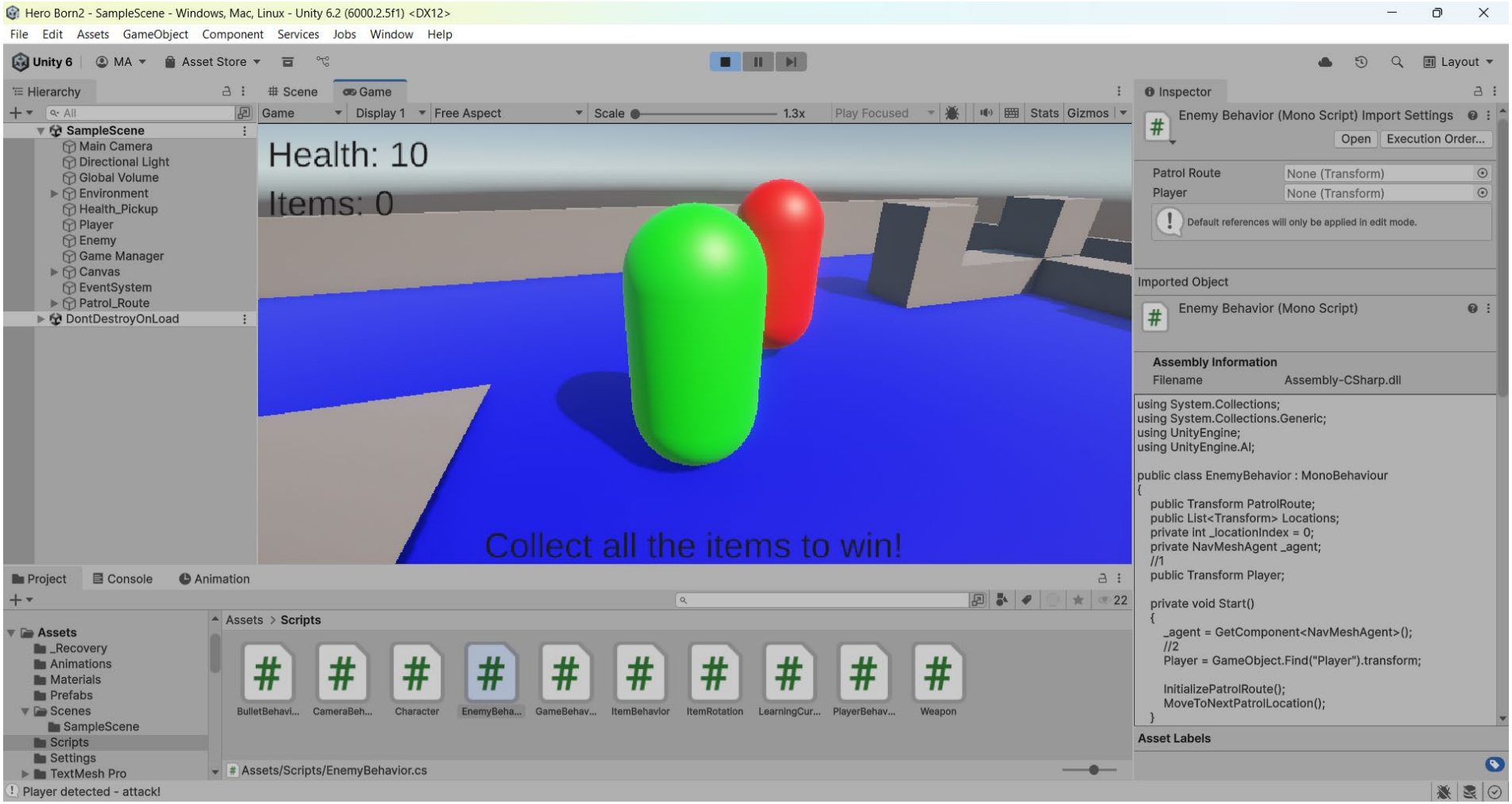
However, we're not going to leave our player helpless; we'll also add in code to track enemy health, detect when an enemy is successfully hit with one of the player's bullets, and when an enemy needs to be destroyed.

Now that the **Enemy** Prefab is moving around on patrol, we need to get a reference to the player's position and change the destination of **NavMeshAgent** if it gets too close.

Add the following code to `EnemyBehavior`:

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

Unity Script (1 asset reference) | 0 references
public class EnemyBehavior : MonoBehaviour
{
    public Transform PatrolRoute;
    public List<Transform> Locations;
    private int _locationIndex = 0;
    private NavMeshAgent _agent;
    //1
    public Transform Player;

    Unity Message | 0 references
    private void Start()
    {
        _agent = GetComponent<NavMeshAgent>();
        //2
        Player = GameObject.Find("Player").transform;

        InitializePatrolRoute();
        MoveToNextPatrolLocation();
    }

    Unity Message | 0 references
    private void Update()
    {
        if(_agent.remainingDistance < 0.2f && !_agent.pathPending)
        {
            MoveToNextPatrolLocation();
        }
    }
```

```
    void InitializePatrolRoute()
    {
        foreach(Transform child in PatrolRoute)
        {
            Locations.Add(child);
        }
    }

    2 references
    void MoveToNextPatrolLocation()
    {
        if (Locations.Count == 0)
            return;

        _agent.destination = Locations[_locationIndex].position;
        _locationIndex = (_locationIndex + 1) % Locations.Count;
    }

    Unity Message | 0 references
    private void OnTriggerEnter(Collider other)
    {
        if(other.name == "Player")
        {
            //3
            _agent.destination = Player.position;
            Debug.Log("Player detected - attack!");
        }
    }

    Unity Message | 0 references
    private void OnTriggerExit(Collider other)
    {
        if(other.name == "Player")
        {
            Debug.Log("Player out of range, resume patrol");
        }
    }
}
```


Hero Born2 - SampleScene - Windows, Mac, Linux - Unity 6.2 (6000.2.5f1) <DX12>
File Edit Assets GameObject Component Services Jobs Window Help
Unity 6
MA
Asset Store
Layout
Hierarchy
Scene
Game
Inspector
SampleScene
Main Camera
Directional Light
Global Volume
Environment
Health_Pickup
Player
Enemy
Game Manager
Canvas
EventSystem
Patrol_Route
DontDestroyOnLoad
Game
Display 1
Free Aspect
Scale
1.3x
Play Focused
Stats
Gizmos
Health: 10
Items: 0
Collect all the items to win!
Enemy Behavior (Mono Script) Import Settings
Open
Execution Order...
Patrol Route
None (Transform)
Player
None (Transform)
Default references will only be applied in edit mode.
Imported Object
Enemy Behavior (Mono Script)
Assembly Information
Filename
Assembly-CSharp.dll
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
public class EnemyBehavior : MonoBehaviour
{
public Transform PatrolRoute;
public List<Transform> Locations;
private int _locationIndex = 0;
private NavMeshAgent _agent;
//1
public Transform Player;
private void Start()
{
_agent = GetComponent<NavMeshAgent>();
//2
Player = GameObject.Find("Player").transform;
InitializePatrolRoute();
MoveToNextPatrolLocation();
}
Asset Labels
Project
Console
Animation
Assets
_Recovery
Animations
Materials
Prefabs
Scenes
SampleScene
Scripts
Settings
TextMesh Pro
Assets > Scripts
BulletBehavi...
CameraBeh...
Character
EnemyBeha...
GameBehav...
ItemBehavior
ItemRotation
LearningCur...
PlayerBehav...
Weapon
Assets/Scripts/EnemyBehavior.cs
Player detected - attack!

Let's break down the code:

```
//1
public Transform Player;

Unity Message | 0 references
private void Start()
{
    _agent = GetComponent<NavMeshAgent>();
    //2
    Player = GameObject.Find("Player").transform;

    InitializePatrolRoute();
    MoveToNextPatrolLocation();
}
```

1. First, it declares a `public` variable to hold the `Player` capsule's `Transform` value.

2. Then, we use `GameObject.Find("Player")` to return a reference to the **Player** object in the scene:

• Adding `.transform` directly references the object's `Transform` value in the same line.

```
private void OnTriggerEnter(Collider other)
{
    if(other.name == "Player")
    {
        //3
        _agent.destination = Player.position;
        Debug.Log("Player detected - attack!");
    }
}
```

3. Finally, we set `_agent.destination` to the player's `Vector3` position in `OnTriggerEnter()` whenever the player enters the enemies' attack zone that we set up earlier with a Collider component.

If you play the game now and get too close to the patrolling enemy, you'll see that it breaks from its path and comes straight for you. Once it reaches the player, the code in the `Update()` method takes over again and the **Enemy** Prefab resumes its patrol.

Lowering player health

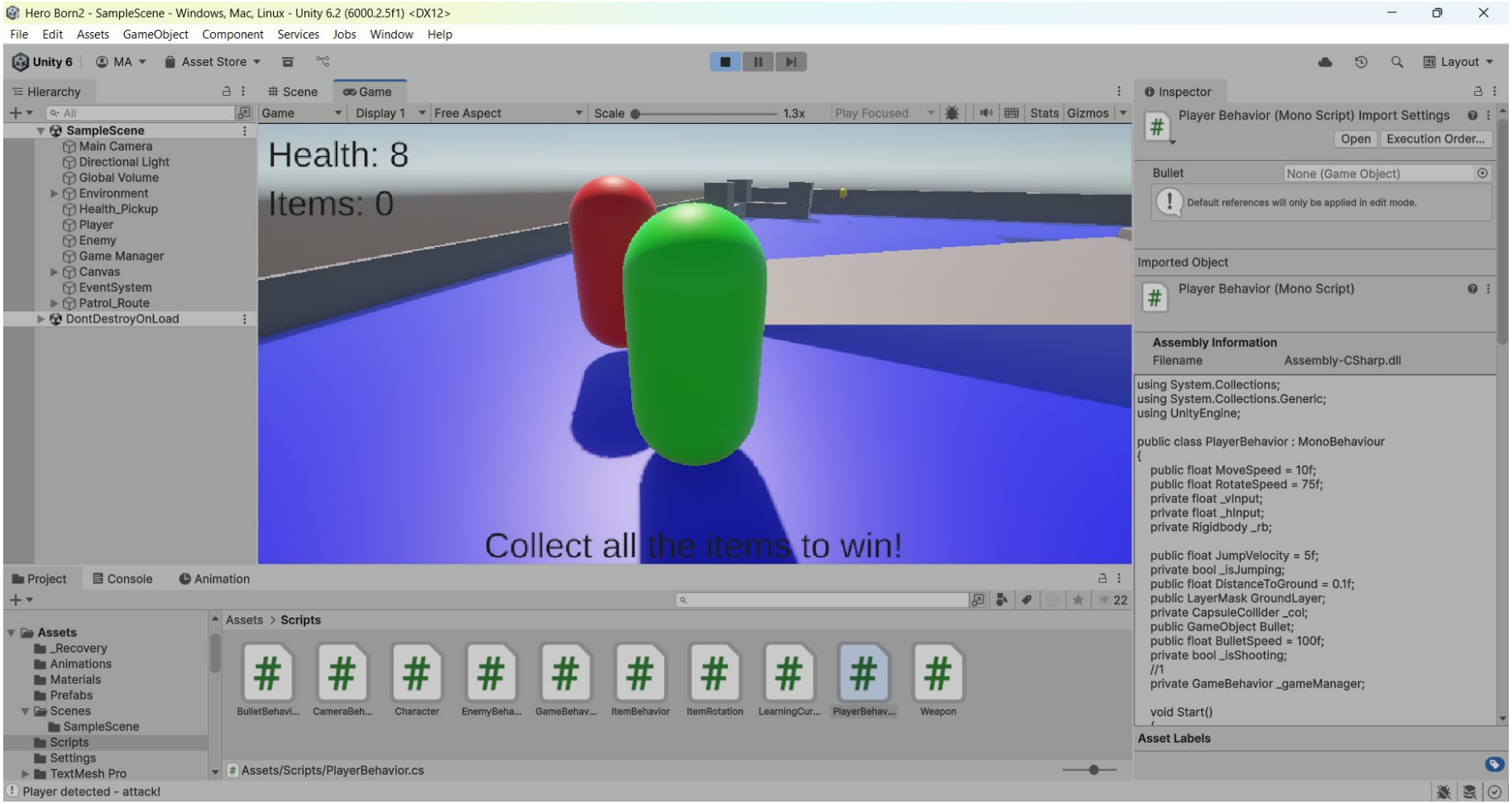
We still need the enemy to be able to hurt the player in some way.

While our enemy mechanic has come a long way, it's still anti-climactic to have nothing happen when the **Enemy** Prefab collides with the player Prefab.
To fix this, we'll tie in the new enemy mechanics with the game manager.

Update `PlayerBehavior` with the following code and hit **Play**:

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

Unity Script (1 asset reference) | 0 references
public class PlayerBehavior : MonoBehaviour
{
    public float MoveSpeed = 10f;
    public float RotateSpeed = 75f;
    private float _vInput;
    private float _hInput;
    private Rigidbody _rb;

    public float JumpVelocity = 5f;
    private bool _isJumping;
    public float DistanceToGround = 0.1f;
    public LayerMask GroundLayer;
    private CapsuleCollider _col;
    public GameObject Bullet;
    public float BulletSpeed = 100f;
    private bool _isShooting;
    //1
    private GameBehavior _gameManager;

    Unity Message | 0 references
    void Start()
    {
        _rb = GetComponent<Rigidbody>();
        _col = GetComponent<CapsuleCollider>();
        //2
        _gameManager = GameObject.Find("Game Manager").GetComponent<GameBehavior>();
    }
    Unity Message | 0 references
    void Update()
    {
        _vInput = Input.GetAxis("Vertical") * MoveSpeed;
        _hInput = Input.GetAxis("Horizontal") * RotateSpeed;

        _isJumping |= Input.GetKeyDown(KeyCode.J);
        _isShooting |= Input.GetKeyDown(KeyCode.Space);
    }
```

```
    void FixedUpdate()
    {
        if(IsGrounded() && _isJumping)
        {
            _rb.AddForce(Vector3.up * JumpVelocity, ForceMode.Impulse);
        }
        _isJumping = false;

        Vector3 rotation = Vector3.up * _hInput;
        Quaternion angleRot = Quaternion.Euler(rotation * Time.fixedDeltaTime);
        _rb.MovePosition(this.transform.position + this.transform.forward * _vInput * Time.fixedDeltaTime);
        _rb.MoveRotation(_rb.rotation * angleRot);

        if(_isShooting)
        {
            GameObject newBullet = Instantiate(Bullet, this.transform.position + new Vector3(0, 0, 1), this.transform.rotation);
            Rigidbody BulletRB = newBullet.GetComponent<Rigidbody>();
            BulletRB.linearVelocity = this.transform.forward * BulletSpeed;
        }
        _isShooting = false;
    }
    1 reference
    private bool IsGrounded()
    {
        Vector3 capsuleBottom = new Vector3(_col.bounds.center.x,
            _col.bounds.min.y, _col.bounds.center.z);
        bool grounded = Physics.CheckCapsule(_col.bounds.center, capsuleBottom, DistanceToGround,
            GroundLayer, QueryTriggerInteraction.Ignore);
        return grounded;
    }
    //3
    Unity Message | 0 references
    private void OnCollisionEnter(Collision collision)
    {
        //4
        if(collision.gameObject.name == "Enemy")
        {
            //5
            _gameManager.HP -= 1;
        }
    }
}
```


Hero Born2 - SampleScene - Windows, Mac, Linux - Unity 6.2 (6000.2.5f1) <DX12>
File Edit Assets GameObject Component Services Jobs Window Help
Unity 6
Hierarchy
SampleScene
Main Camera
Directional Light
Global Volume
Environment
Health_Pickup
Player
Enemy
Game Manager
Canvas
EventSystem
Patrol_Route
DontDestroyOnLoad
Scene
Game
Health: 8
Items: 0
Collect all the items to win!
Inspector
Player Behavior (Mono Script) Import Settings
Open
Execution Order...
Bullet
None (Game Object)
Default references will only be applied in edit mode.
Imported Object
Player Behavior (Mono Script)
Assembly Information
Filename
Assembly-CSharp.dll
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class PlayerBehavior : MonoBehaviour
{
public float MoveSpeed = 10f;
public float RotateSpeed = 75f;
private float _vInput;
private float _hInput;
private Rigidbody _rb;
public float JumpVelocity = 5f;
private bool _isJumping;
public float DistanceToGround = 0.1f;
public LayerMask GroundLayer;
private CapsuleCollider _col;
public GameObject Bullet;
public float BulletSpeed = 100f;
private bool _isShooting;
//1
private GameBehavior _gameManager;
void Start()
Asset Labels
Project
Console
Animation
Assets
_Recovery
Animations
Materials
Prefabs
Scenes
SampleScene
Scripts
Settings
TextMesh Pro
Assets > Scripts
BulletBehavi...
CameraBeh...
Character
EnemyBeha...
GameBehav...
ItemBehavior
ItemRotation
LearningCur...
PlayerBehav...
Weapon
Assets/Scripts/PlayerBehavior.cs
Player detected - attack!

Let's break down the code:

```
//1
private GameBehavior _gameManager;

Unity Message | 0 references
void Start()
{
    _rb = GetComponent<Rigidbody>();
    _col = GetComponent<CapsuleCollider>();
    //2
    _gameManager = GameObject.Find("Game Manager").GetComponent<GameBehavior>();
}
```

1. First, it declares a `private` variable to hold the reference to the instance of `GameBehavior` we have in the scene.

2. Then, it finds and returns the `GameBehavior` script that's attached to the `Game Manager` object in the scene:

• Using `GetComponent()` on the same line `as GameObject.Find()` is a common way to cut down on unnecessary lines of code.

```
//3
Unity Message | 0 references
private void OnCollisionEnter(Collision collision)
{
    //4
    if(collision.gameObject.name == "Enemy")
    {
        //5
        _gameManager.HP -= 1;
    }
}
```

3. Since our player is the object being collided with, it makes sense to declare `OnCollisionEnter()` in `PlayerBehavior`.

4. Next, we check for the name of the colliding object; if it's the **Enemy** Prefab, we execute the body of the `if` statement.

5. Finally, we subtract `1` from the public `HP` variable using the `_gameManager` instance.

Collisions between two objects work both ways, so you could also put this code in the `EnemyBehavior` script and look for a collision with the Player object.
Remember, we're using our big Sphere Collider to detect when the Player is in range, but here we're using the Capsule Collider to detect collisions when the player gets hit (GameObjects can have multiple colliders).

Whenever the enemy now tracks and collides with the player, the game manager will fire the set property on **`HP`**.

The UI will update with a new value for player health, which means we have an opportunity to put in some additional logic for the loss condition later on.

For now, let's move on to detecting when bullets hit our enemies and potentially take them out of commission.

Detecting bullet collisions

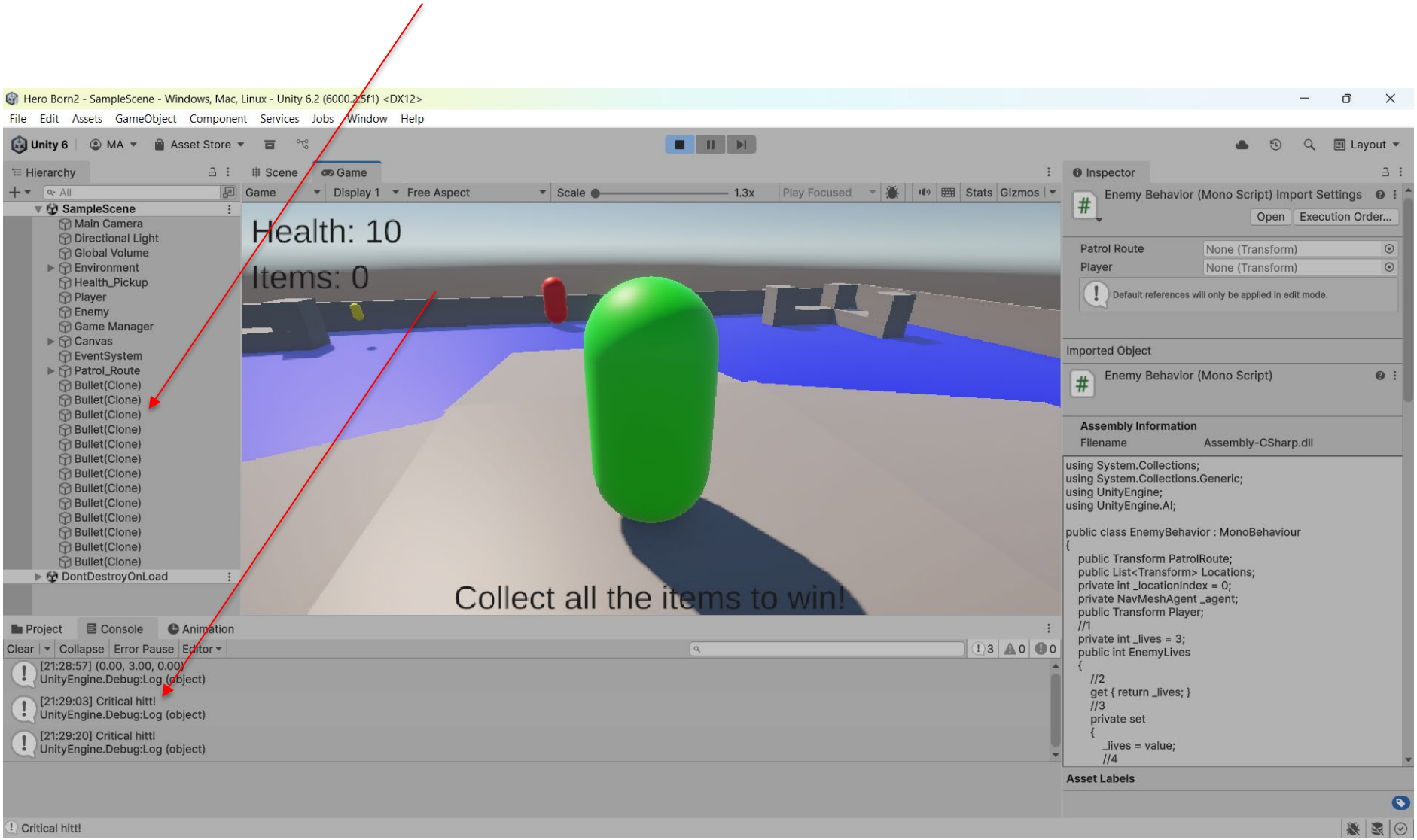
Now that we have our loss condition, it's time to add a way for our player to fight back and survive enemy attacks.

Open up `EnemyBehavior` and modify it with the following code:

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

Unity Script (1 asset reference) | 0 references
public class EnemyBehavior : MonoBehaviour
{
    public Transform PatrolRoute;
    public List<Transform> Locations;
    private int _locationIndex = 0;
    private NavMeshAgent _agent;
    public Transform Player;
    //1
    private int _lives = 3;
    1 reference
    public int EnemyLives
    {
        //2
        get { return _lives; }
        //3
        private set
        {
            _lives = value;
            //4
            if(_lives <= 0)
            {
                Destroy(this.gameObject);
                Debug.Log("Enemy down.");
            }
        }
    }

    Unity Message | 0 references
    private void Start()
    {
        _agent = GetComponent<NavMeshAgent>();
        Player = GameObject.Find("Player").transform;

        InitializePatrolRoute();
        MoveToNextPatrolLocation();
    }
```

```
Unity Message | 0 references
private void Update()
{
    if(_agent.remainingDistance < 0.2f && !_agent.pathPending)
    {
        MoveToNextPatrolLocation();
    }
}

1 reference
void InitializePatrolRoute()
{
    foreach(Transform child in PatrolRoute)
    {
        Locations.Add(child);
    }
}

2 references
void MoveToNextPatrolLocation()
{
    if (Locations.Count == 0)
        return;

    _agent.destination = Locations[_locationIndex].position;
    _locationIndex = (_locationIndex + 1) % Locations.Count;
}
```

```
    Unity Message | 0 references
    private void OnTriggerEnter(Collider other)
    {
        if(other.name == "Player")
        {
            _agent.destination = Player.position;
            Debug.Log("Player detected - attack!");
        }
    }

    Unity Message | 0 references
    private void OnTriggerExit(Collider other)
    {
        if(other.name == "Player")
        {
            Debug.Log("Player out of range, resume patrol");
        }
    }

    //5
    Unity Message | 0 references
    private void OnCollisionEnter(Collision collision)
    {
        if(collision.gameObject.name == "Bullet(Clone)")
        {
            //6
            EnemyLives -= 1;
            Debug.Log("Critical hitt!");
        }
    }
}
```


Hero Born2 - SampleScene - Windows, Mac, Linux - Unity 6.2 (6000.2.5f1) <DX12>
File Edit Assets GameObject Component Services Jobs Window Help
Unity 6
Hierarchy
SampleScene
Main Camera
Directional Light
Global Volume
Environment
Health_Pickup
Player
Enemy
Game Manager
Canvas
EventSystem
Patrol_Route
Bullet(Clone)
DontDestroyOnLoad
Scene
Game
Health: 10
Items: 0
Collect all the items to win!
Inspector
Enemy Behavior (Mono Script) Import Settings
Open
Execution Order...
Patrol Route
None (Transform)
Player
None (Transform)
Default references will only be applied in edit mode.
Imported Object
Enemy Behavior (Mono Script)
Assembly Information
Filename
Assembly-CSharp.dll
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
public class EnemyBehavior : MonoBehaviour
{
public Transform PatrolRoute;
public List<Transform> Locations;
private int _locationIndex = 0;
private NavMeshAgent _agent;
public Transform Player;
//1
private int _lives = 3;
public int EnemyLives
{
//2
get { return _lives; }
//3
private set
{
_lives = value;
//4
Asset Labels
Project
Console
Animation
Clear
Collapse
Error Pause
Editor
[21:28:57] (0.00, 3.00, 0.00)
UnityEngine.Debug:Log (object)
[21:29:03] Critical hit!
UnityEngine.Debug:Log (object)
[21:29:20] Critical hit!
UnityEngine.Debug:Log (object)
Critical hit!

Let's break down the code:

```
//1
private int _lives = 3;
1 reference
public int EnemyLives
{
    //2
    get { return _lives; }
    //3
    private set
    {
        _lives = value;
        //4
        if(_lives <= 0)
        {
            Destroy(this.gameObject);
            Debug.Log("Enemy down.");
        }
    }
}
```

1. First, it declares a `private int` variable called `_lives` with a `public` backing variable called `EnemyLives`. This will let us control how `EnemyLives` is referenced and set, just like in `GameBehavior`.

2. Then, we set the `get` property to always return `_lives`.

3. Next, we use `private set` to assign the new value of `EnemyLives` to `_lives` to keep them both in sync.

```
//1
private int _lives = 3;
1 reference
public int EnemyLives
{
    //2
    get { return _lives; }
    //3
    private set
    {
        _lives = value;
        //4
        if(_lives <= 0)
        {
            Destroy(this.gameObject);
            Debug.Log("Enemy down.");
        }
    }
}
```

4. Then, we add an `if` statement to check whether `_lives` is less than or equal to `0`, meaning that the enemy should be dead:

- When that's the case, we destroy the `Enemy GameObject` and print out a message to the console.

We haven't seen `private get` or `private set` before, but they can have access modifiers just like any other executable code. Declaring `get` or `set` as `private` means that only the parent class has access to their functionality.

```
//5
Unity Message | 0 references
private void OnCollisionEnter(Collision collision)
{
    if(collision.gameObject.name == "Bullet(Clone)")
    {
        //6
        EnemyLives -= 1;
        Debug.Log("Critical hitt!");
    }
}
```

5. Because `Enemy` is the object getting hit with bullets, it's sensible to include a check for those collisions in `EnemyBehavior` with `OnCollisionEnter()`.

6. Finally, if the name of the colliding object matches a bullet clone object, we decrement `EnemyLives` by `1` and print out another message.

Notice that the name we're checking for is `Bullet(Clone)`, even though our bullet Prefab is named `Bullet`. This is because Unity adds the `(Clone)` suffix to any object created with the `Instantiate()` method, which is how we made them in our shooting logic.

Now, the player can fight back when the enemy tries to take one of its lives by shooting it three times and destroying it.

Again, our use of the `get` and `set` properties to handle additional logic proves to be a flexible and scalable solution.

Updating the game manager

Our final task is to update the game manager with a loss condition.

To fully implement the loss condition, we need to update the manager class:

1. Open up `GameBehavior` and add the following code:

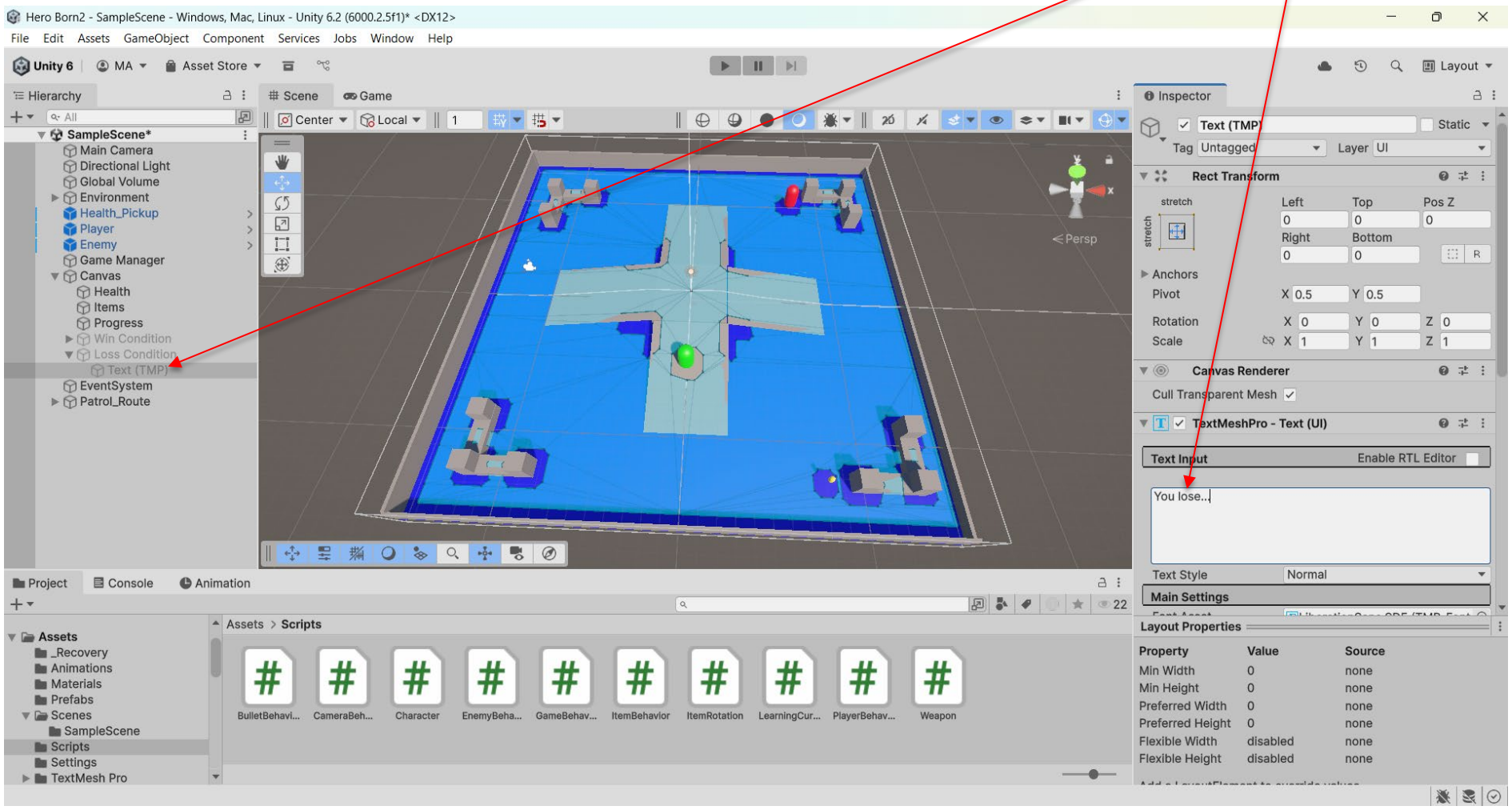
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
```

```
Unity Script (1 asset reference) | 4 references
public class GameBehavior : MonoBehaviour
{
    public int MaxItems = 4;
    public TMP_Text HealthText;
    public TMP_Text ItemText;
    public TMP_Text ProgressText;
    public Button WinButton;
    //1
    public Button LossButton;

    Unity Message | 0 references
    private void Start()
    {
        ItemText.text += _itemCollected;
        HealthText.text += _playerHP;
    }

    private int _itemCollected = 0;
    2 references
    public int Items
    {
        get { return _itemCollected; }
        set {
            _itemCollected = value;
            ItemText.text = "Items: " + Items;

            if(_itemCollected >= MaxItems)
            {
                ProgressText.text = "You have found all the itesm!";
                WinButton.gameObject.SetActive(true);

                Time.timeScale = 0f;
            }
            else
            {
                ProgressText.text = "Item found, only " + (MaxItems - _itemCollected) + " more to go!";
            }
        }
    }
```

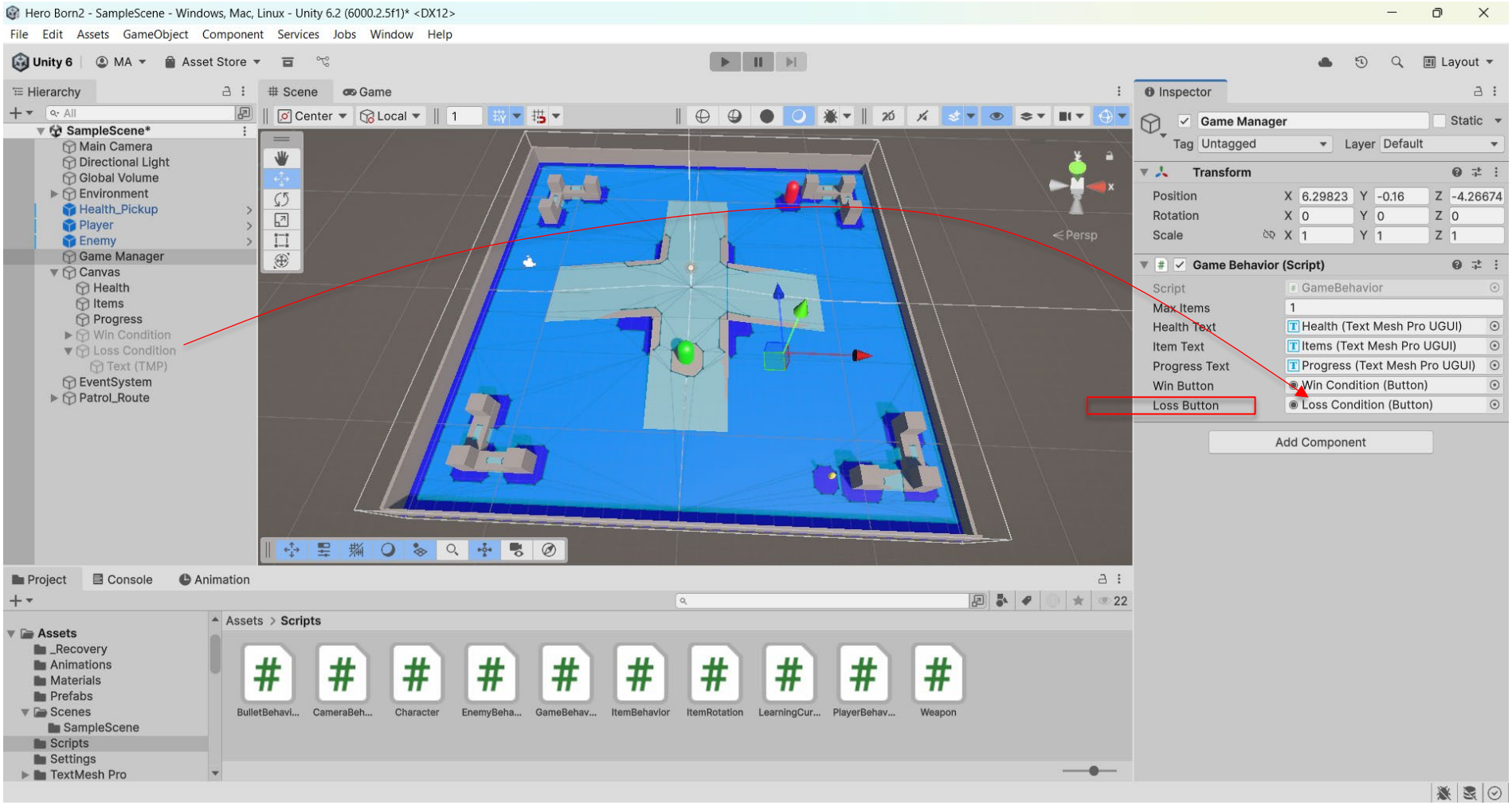
```
    0 references
    public void RestartScene()
    {
        SceneManager.LoadScene(0);
        Time.timeScale = 1f;
    }

    private int _playerHP = 10;
    2 references
    public int HP
    {
        get { return _playerHP; }
        set
        {
            _playerHP = value;
            HealthText.text = "Health: " + HP;
            //Debug.LogFormat("Lives: {0}", _playerHP);
            //2
            if(_playerHP <= 0)
            {
                ProgressText.text = "You want another life with that?";
                LossButton.gameObject.SetActive(true);
                Time.timeScale = 0;
            }
            else
            {
                ProgressText.text = "Ouch... that's got hurt.";
            }
        }
    }
}
```

2. In the **Hierarchy** window, right-click on **Win Condition**, choose **Duplicate**, and name it `Loss Condition`:

- Click the arrow to the left of **Loss Condition** to expand it, select the **Text (TMP)** object, and change the text to **You lose...**



3. Select **Game Manager** in the **Hierarchy** window and drag **Loss Condition** into the **Loss Button** slot in the **Game Behavior (Script)** component:



Let's break down the code:

```
//1
public Button LossButton;
```

1. First, we declare a new button that we want to show when the player loses the game.

```
//2
if(_playerHP <= 0)
{
    ProgressText.text = "You want another life with that?";
    LossButton.gameObject.SetActive(true);
    Time.timeScale = 0;
}
else
{
    ProgressText.text = "Ouch... that's got hurt.";
}
```

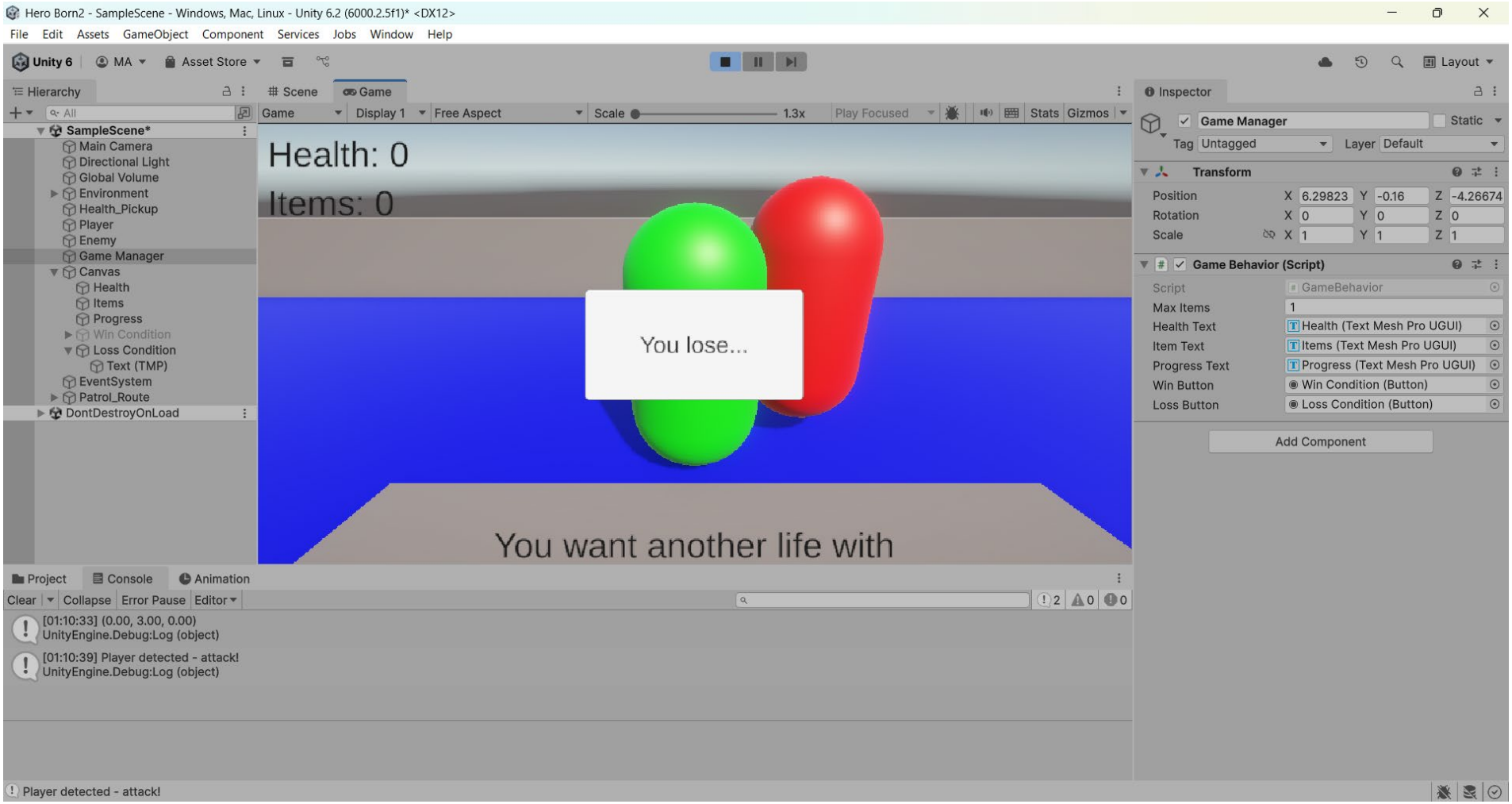
2. Then, we add in an `if` statement to check when `_playerHP` drops below `0`:

- If it's `true`, `ProgessText` and `Time.timeScale` are updated and the **Loss Condition** button is activated.
- If the player is still alive following an enemy collision, `ProgessText` shows a different message: **"Ouch… that's got to hurt."**

Now, change `_playerHP` to `1` in `GameBehavior.cs` and get the **Enemy** Prefab to collide with you and observe what happens.

```
private int _playerHP = 1 /*10*/;
```

You've successfully added a "smart" enemy that can damage the player and be damaged right back, as well as a loss screen through the game manager.


Hero Born2 - SampleScene - Windows, Mac, Linux - Unity 6.2 (6000.2.5f1)* <DX12>
File Edit Assets GameObject Component Services Jobs Window Help
Unity 6
Hierarchy
SampleScene*
Main Camera
Directional Light
Global Volume
Environment
Health_Pickup
Player
Enemy
Game Manager
Canvas
Health
Items
Progress
Win Condition
Loss Condition
Text (TMP)
EventSystem
Patrol_Route
DontDestroyOnLoad
Scene
Game
Display 1
Free Aspect
Scale
1.3x
Play Focused
Stats
Gizmos
Health: 0
Items: 0
You lose...
You want another life with
Inspector
Game Manager
Static
Tag Untagged
Layer Default
Transform
Position X 6.29823 Y -0.16 Z -4.26674
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Game Behavior (Script)
Script GameBehavior
Max Items 1
Health Text Health (Text Mesh Pro UGUI)
Item Text Items (Text Mesh Pro UGUI)
Progress Text Progress (Text Mesh Pro UGUI)
Win Button Win Condition (Button)
Loss Button Loss Condition (Button)
Add Component
Project
Console
Animation
Clear
Collapse
Error Pause
Editor
[01:10:33] (0.00, 3.00, 0.00)
UnityEngine.Debug:Log (object)
[01:10:39] Player detected - attack!
UnityEngine.Debug:Log (object)
Player detected - attack!

Refactoring and keeping it DRY

Don’t Repeat Yourself (DRY)

In practice, repeated code is part of programming life. Trying to avoid it by constantly thinking ahead will put up so many roadblocks in your project that it won't seem worthwhile carrying on.

A more efficient—and sane—approach to dealing with repeating code is to quickly identify it when and where it occurs and then look for the best way to remove it. This task is called refactoring, and our `GameBehavior` class could use a little of its magic right now.

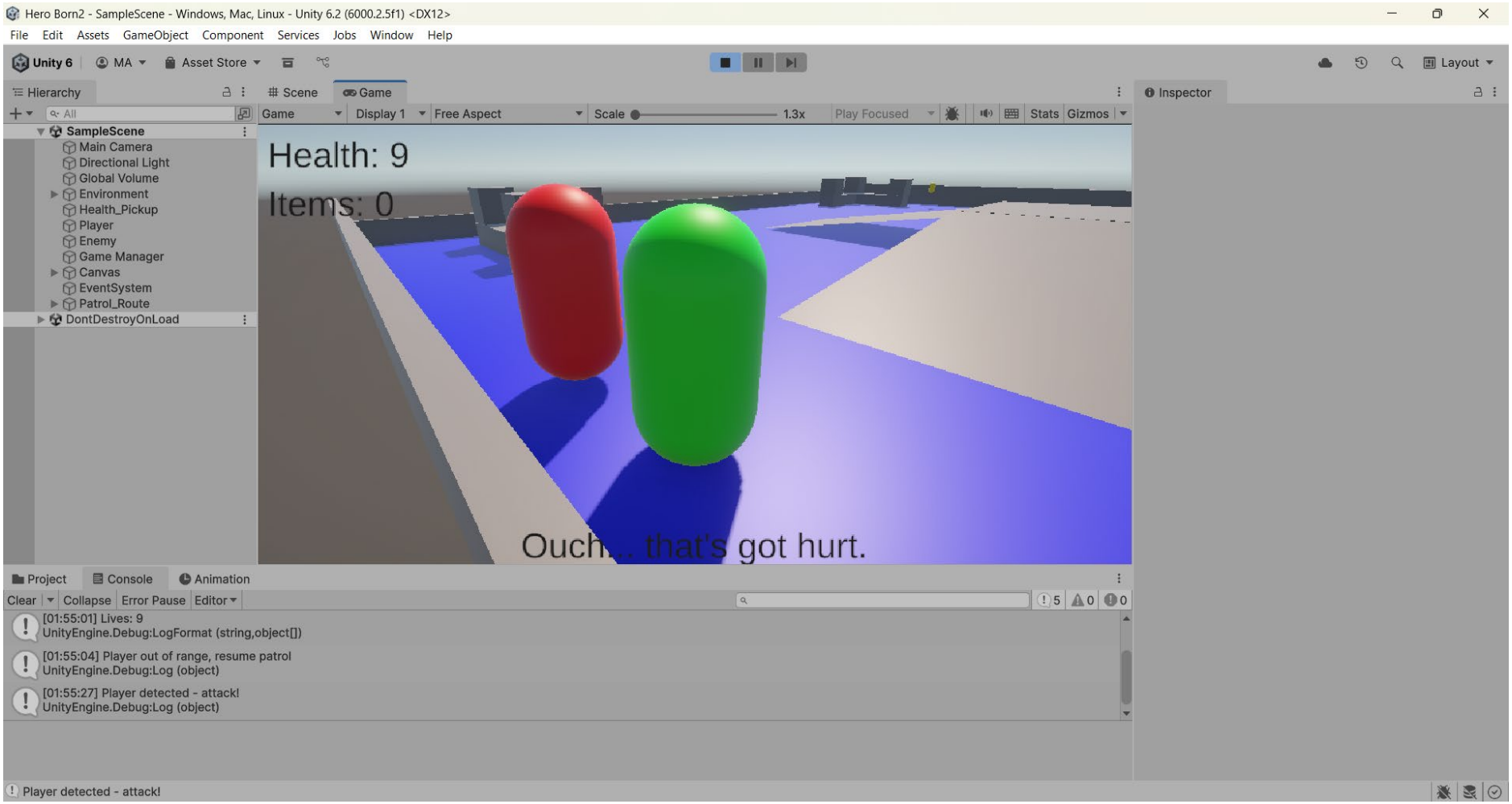
You may have noticed that we set the progress text and timescale in two separate places, but we could easily make ourselves a utility method to do this for us in a single place.

To refactor the existing code, you'll need to update `GameBehavior.cs` as follows:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

Unity Script (1 asset reference) | 4 references
public class GameBehavior : MonoBehaviour
{
    public int MaxItems = 4;
    public TMP_Text HealthText;
    public TMP_Text ItemText;
    public TMP_Text ProgressText;
    public Button WinButton;
    public Button LossButton;

    Unity Message | 0 references
    private void Start()
    {
        ItemText.text += _itemCollected;
        HealthText.text += _playerHP;
    }

    //1
    2 references
    public void UpdateScene(string updatedText)
    {
        ProgressText.text = updatedText;
        Time.timeScale = 0f;
    }
```

```
private int _itemCollected = 0;
2 references
public int Items
{
    get { return _itemCollected; }
    set {
        _itemCollected = value;
        ItemText.text = "Items: " + Items;
        //ItemText.text = "Items Collected: " + Items;

        if (_itemCollected >= MaxItems)
        {
            //ProgressText.text = "You have found all the itesm!";
            WinButton.gameObject.SetActive(true);
            //Time.timeScale = 0f;
            //2
            UpdateScene("You've found all the items!");
        }
        else
        {
            ProgressText.text = "Item found, only " + (MaxItems - _itemCollected) + " more to go!";
        }
    }
}

0 references
public void RestartScene()
{
    SceneManager.LoadScene(0);
    Time.timeScale = 1f;
}
```

```
    private int _playerHP = /*1*/ 10;
    2 references
    public int HP
    {
        get { return _playerHP; }
        set
        {
            _playerHP = value;
            HealthText.text = "Health: " + HP;
            //HealthText.text = "Player Health: " + HP;

            if (_playerHP <= 0)
            {
                //ProgressText.text = "You want another life with that?";
                LossButton.gameObject.SetActive(true);
                //Time.timeScale = 0;
                //3
                UpdateScene("You want another life with that?");
            }
            else
            {
                ProgressText.text = "Ouch... that's got hurt.";
            }
            Debug.LogFormat("Lives: {0}", _playerHP);
        }
    }
}
```


Hero Born2 - SampleScene - Windows, Mac, Linux - Unity 6.2 (6000.2.5f1) <DX12>
File Edit Assets GameObject Component Services Jobs Window Help
Unity 6
MA
Asset Store
Layout
Hierarchy
Scene
Game
Inspector
All
SampleScene
Main Camera
Directional Light
Global Volume
Environment
Health_Pickup
Player
Enemy
Game Manager
Canvas
EventSystem
Patrol_Route
DontDestroyOnLoad
Game
Display 1
Free Aspect
Scale
1.3x
Play Focused
Stats
Gizmos
Health: 9
Items: 0
Ouch... that's got hurt.
Project
Console
Animation
Clear
Collapse
Error Pause
Editor
[01:55:01] Lives: 9
UnityEngine.Debug:LogFormat (string,object[])
[01:55:04] Player out of range, resume patrol
UnityEngine.Debug:Log (object)
[01:55:27] Player detected - attack!
UnityEngine.Debug:Log (object)
Player detected - attack!

Let’s break down the code:

```
//1
2 references
public void UpdateScene(string updatedText)
{
    ProgressText.text = updatedText;
    Time.timeScale = 0f;
}
```

1. We declared a new method called `UpdateScene`, which takes in a string parameter that we want to assign to `ProgressText` and sets `Time.timeScale` to `0`.

```
if (_itemCollected >= MaxItems)
{
    //ProgressText.text = "You have found all the itesm!";
    WinButton.gameObject.SetActive(true);
    //Time.timeScale = 0f;
    //2
    UpdateScene("You've found all the items!");
}
```

2. We deleted our first instance of duplicated code and used our new method to update our scene when the game is won.

```
if (_playerHP <= 0)
{
    //ProgressText.text = "You want another life with that?";
    LossButton.gameObject.SetActive(true);
    //Time.timeScale = 0;
    //3
    UpdateScene("You want another life with that?");
}
```

3. We deleted our second instance of duplicated code and update the scene when the game is lost.

Summary

With that, our enemy and player interactions are complete.
We can dish out damage as well as take it, lose lives, and fight back, all while updating the on-screen GUI.

Our enemies use Unity's navigation system to walk around the arena and change to attack mode when within a specified range of the player.

Each GameObject is responsible for its behavior, internal logic, and object collisions, while the game manager keeps track of the variables that govern the game's state.

Lastly, we learned about simple procedural programming and how much cleaner code can be when repeated instructions are abstracted out into their methods.


unity