Quest 6 - Steps

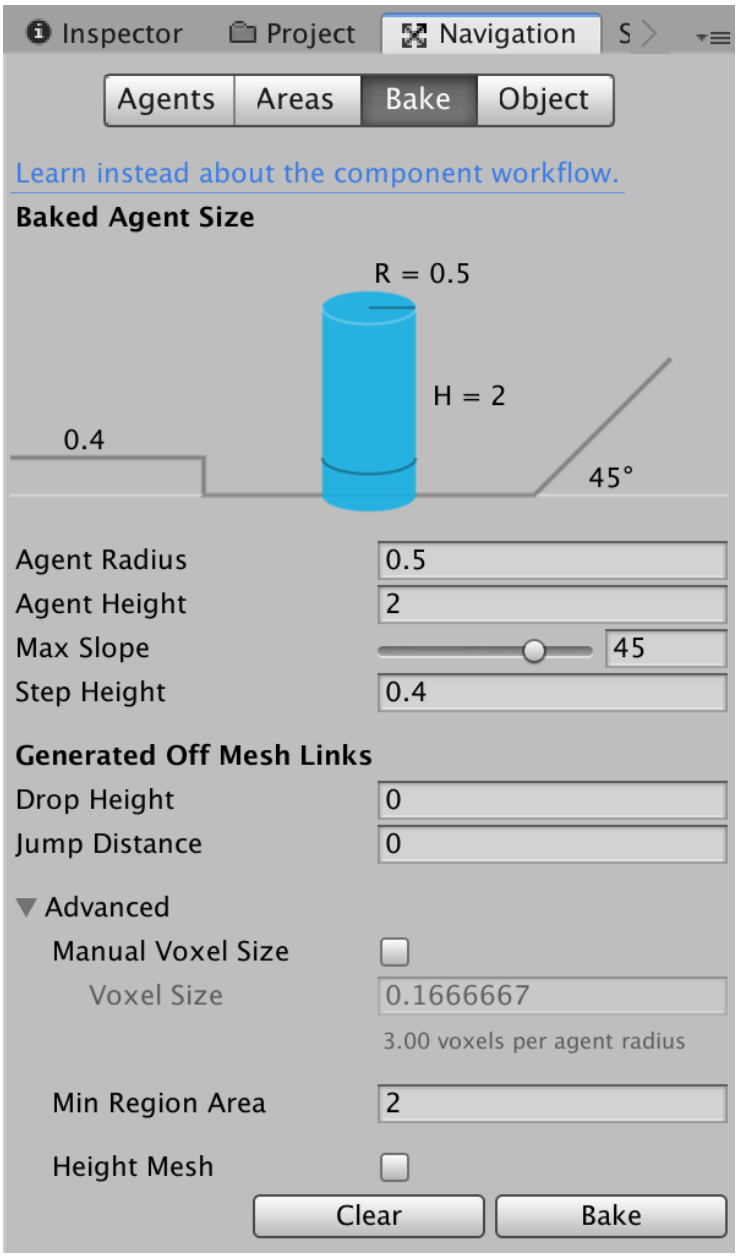
1) Create the Terrain

You will need the PlayerController functionality of Q5 and a Terrain to use a playing field.

2) Bake the NavMesh

The NavMesh is a computation of what is navigable by Unity's built-in pathfinding AI.

Open Window->AI->Navigation and select the object(s) you want to be navigable by AI. Default settings are fine unless you are already certain of alternative character sizes.



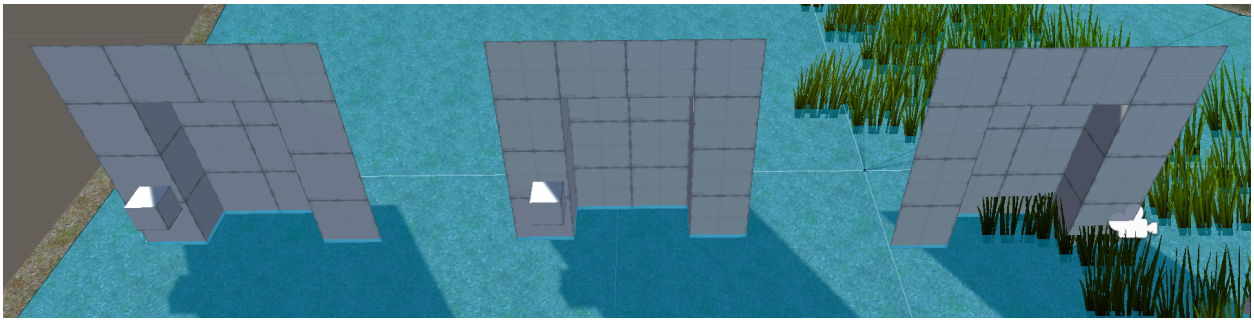
With the navigable object(s) selected, click Bake and wait for computation to finish.

The areas highlighted in turquoise represent the area that can be navigated by an AI agent of the shape defined in your bake settings.

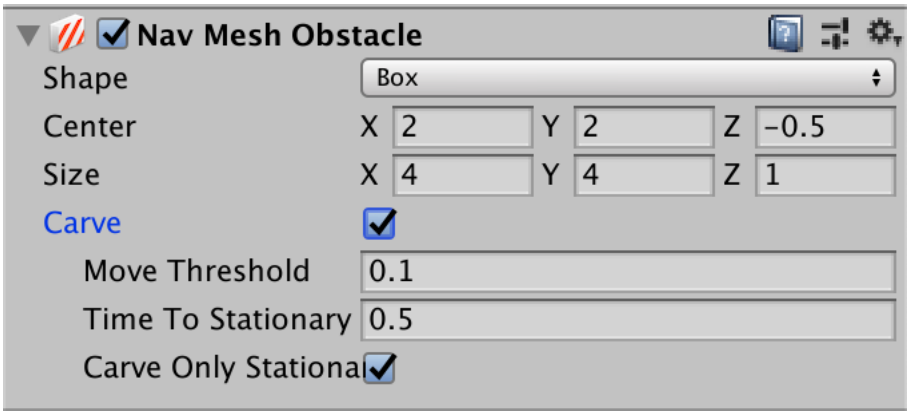
If necessary, you may need to adjust your terrain and repeat this step to rebake a new NavMesh.

3) Setup NavMesh Obstructions

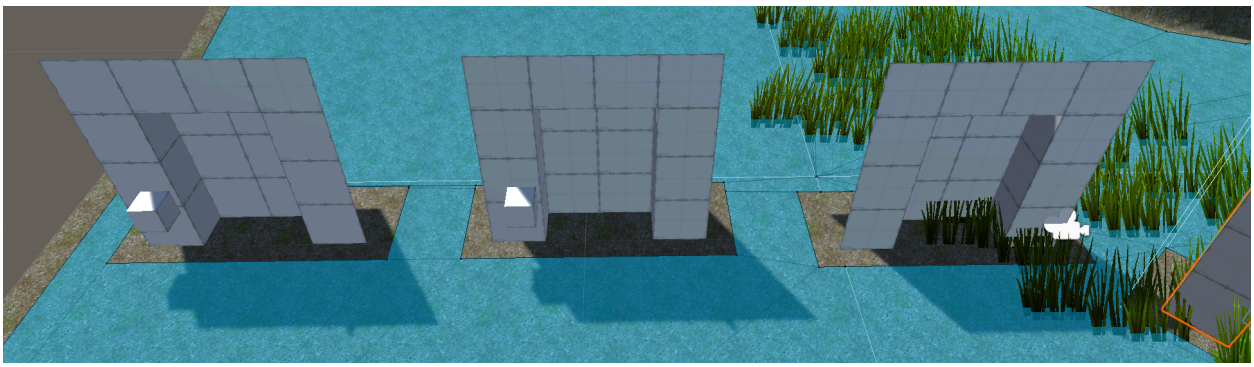
You may have objects that should obstruct the NavMesh but are currently marked as navigable.



We must mark these objects as obstructions by attaching a NavMeshObstacle component to the parent-most object containing the meshrenderer.

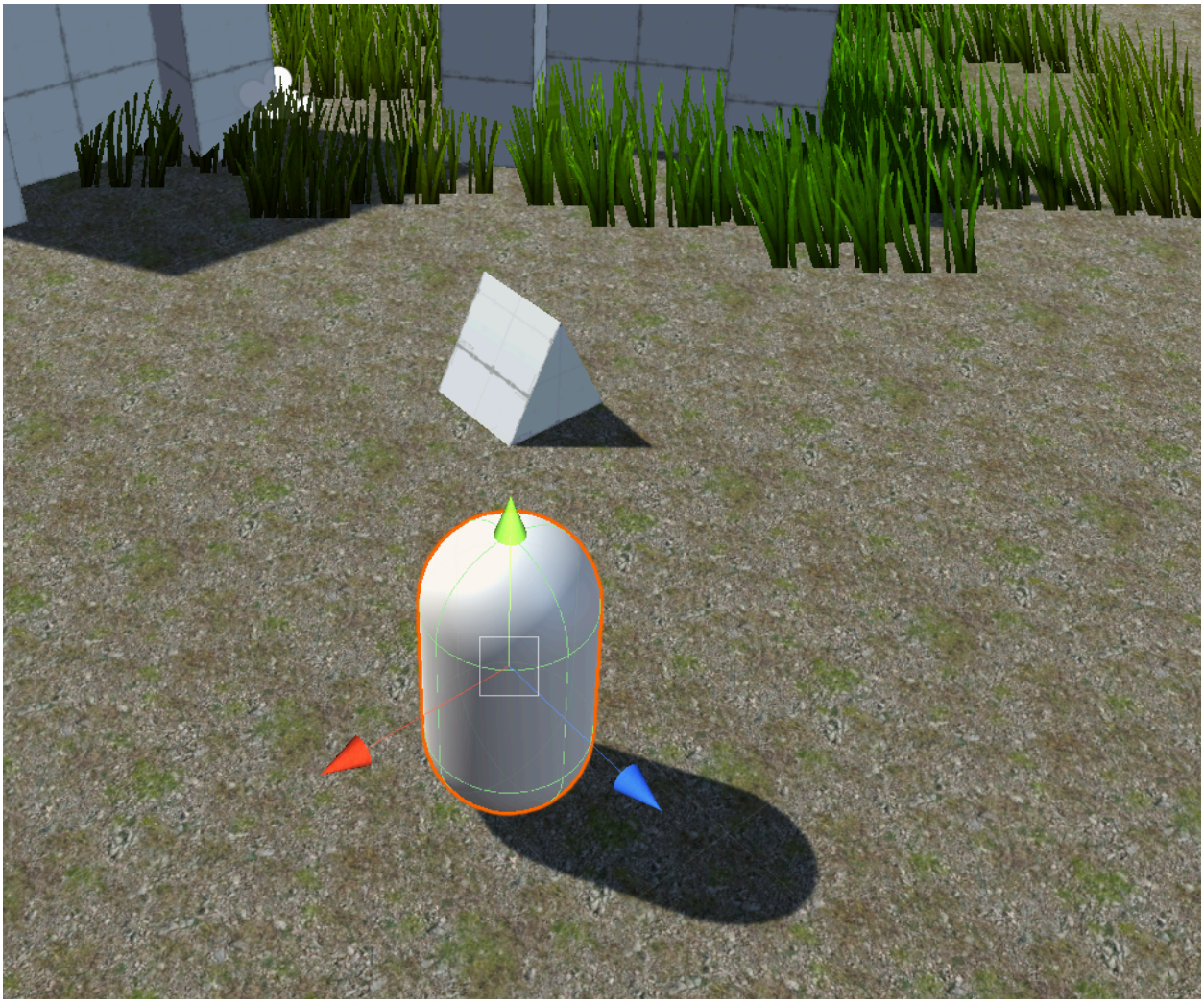


The Carve setting must be checked as well.



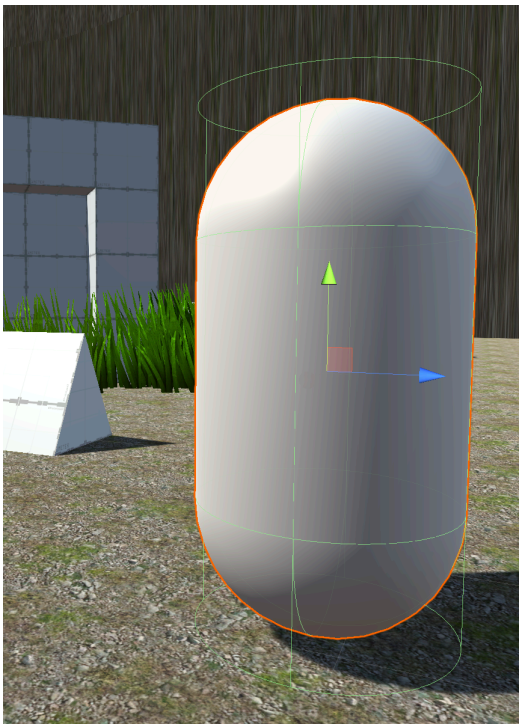
4) Create Enemy #1

Create a capsule-shaped object to represent your first enemy, and attach an Enemy.cs component.

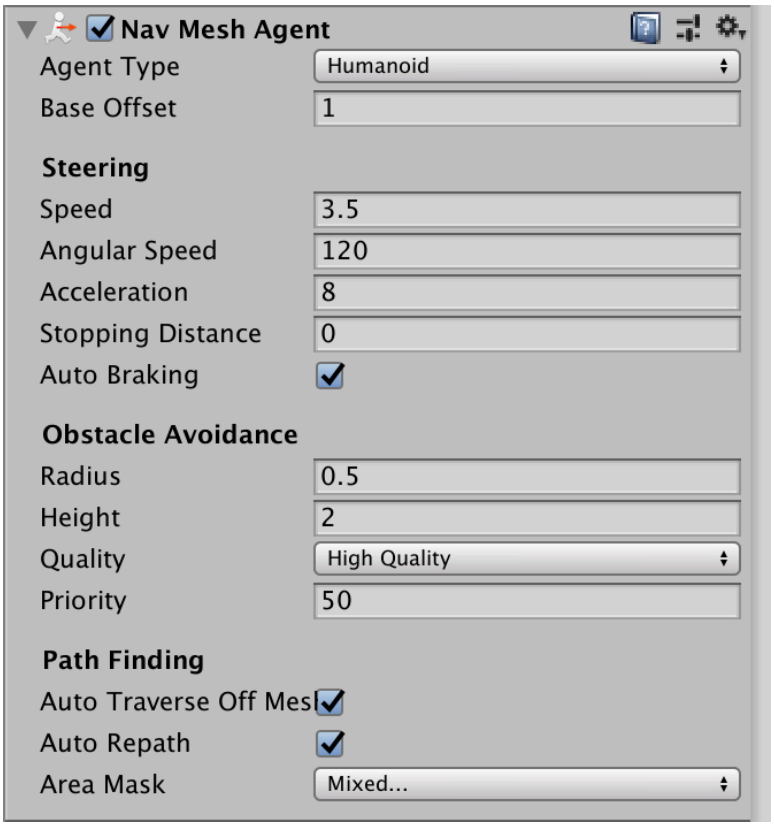


Add a NavMeshAgent component to your Enemy. This component allows your enemy to move according to AI Pathfinding computations.

(In other projects, if an object has BOTH NavMeshAgent and Rigidbody, the Rigidbody must be set to Kinematic, otherwise these two components will unpredictably compete with each other.)



Adjust the NavMeshAgent settings so that the cylindrical outline roughly encases the character bounds.



5) Program Enemy Pathing

The Enemy script will pass a destination to the NavMeshAgent. In this example, we are updating the destination every frame in case the target is moving.

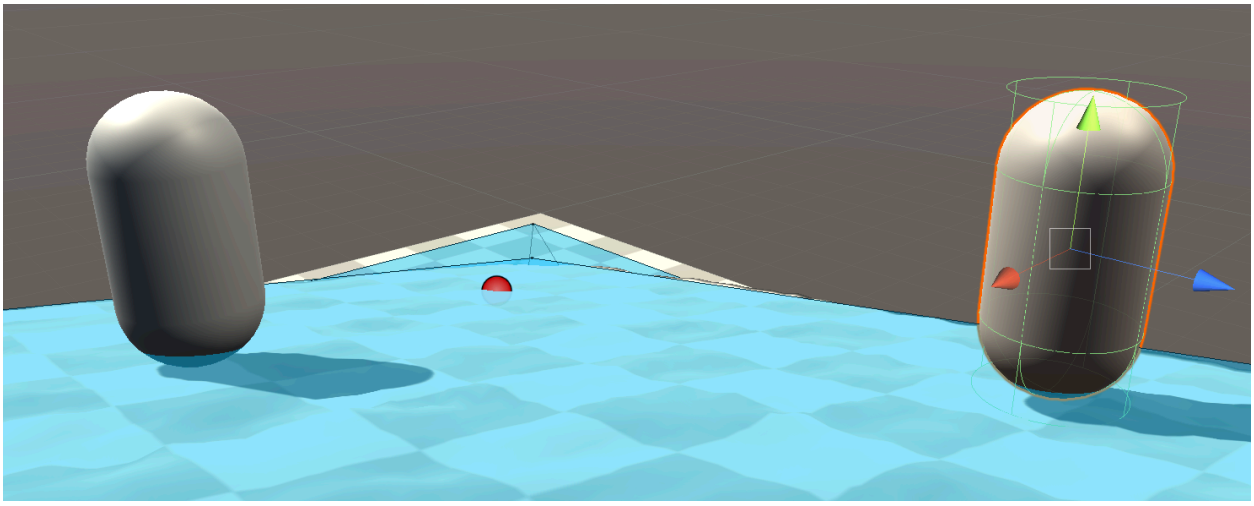
```
using UnityEngine.AI;

public class Enemy : MonoBehaviour
{
    // Outlets
    NavMeshAgent navAgent;

    // Configuration
    public Transform target;

    // Methods
    void Start() {
        navAgent = GetComponent<NavMeshAgent>();
    }

    void Update() {
        if(target) {
            navAgent.destination = target.position;
        }
    }
}
```

Set the Player as the target of the enemy, so that the enemy follows you during gameplay.

Playtest your game and ensure that the Enemies travel toward the player's position as best as the terrain allows.

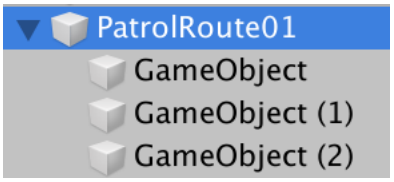
6) More Intelligent Enemy Patrolling and Tracking

Duplicate your Enemy. This second enemy will be assigned a patrol route, and it will only chase the player if the player is within a threshold distance. It will also forget about the player and return to its patrol route if the player escapes.



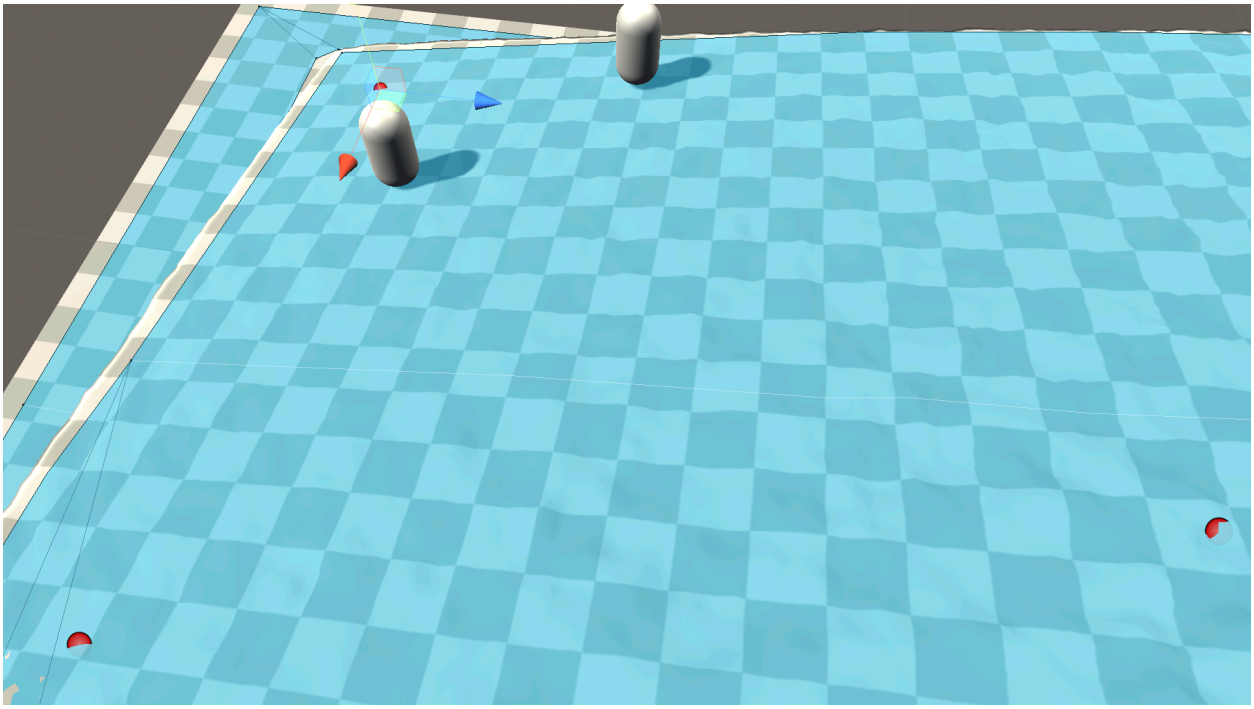
7) Create a Patrol Route

Create an empty game object and rename it PatrolRoute. The children of this object will be positioned as patrol points throughout the map.



You can make patrol points easier to see and manage by changing their object icon. These icons are only visible during Scene view and do not affect gameplay.

Notice how the three patrol points have been positioned in a triangular route. Use Cmd+Shift (Mac) or Ctrl+Shift (Windows) to ensure the patrol points snap to the terrain and do not get stuck underground.

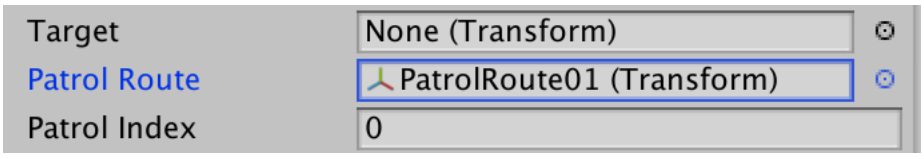


8) Patrol Logic

The Enemy.cs needs an outlet variable to tell it what object to use as a Patrol Route (whose children will be used as Patrol Points) and a state tracking variable to keep track of which Patrol Point we are currently tracking.

```
public Transform target;
public Transform patrolRoute;

// State Tracking
public int patrolIndex;
```

The Patrol Route needs to be assigned in the Inspector. Assign the parent object that holds all of the individual patrol points.



We will refine the tracking logic to cycle through the Patrol Points if a Patrol Route has been assigned.

```
void Update() {
    if(patrolRoute) {
        // Which patrol point is active?
        target = patrolRoute.GetChild(patrolIndex);

        // How far is the patrol point
        float distance = Vector3.Distance(transform.position, target.position);
        print("Distance: " + distance);

        // Target the next point when we are close enough
        if(distance <= 1.5f) {
            patrolIndex++;
            if(patrolIndex >= patrolRoute.childCount) {
                patrolIndex = 0;
            }
        }
    }
```

The threshold distance of 1.5 for switching patrol points may need to be adjusted depending on your terrain shape and the size of your character. In this code, I am printing distance to the console log to help you assess what that number should be.

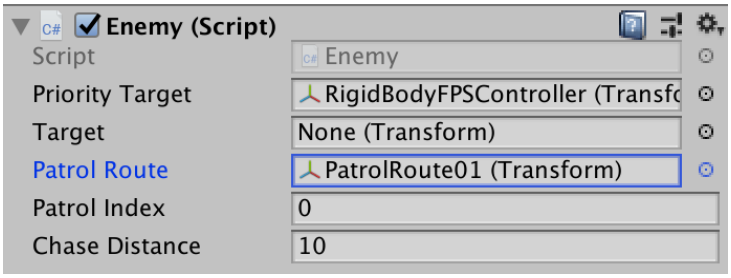
If we playtest the game now, you should see that this enemy patrols its route by cycling through each patrol point in sequence. It does not yet know when to follow the player.

9) Chase Logic

To have our enemy also chase the player, we need another outlet variable to define the player as a priority target and a configuration variable to define the range that will trigger a chase.

```
// Configuration
public Transform priorityTarget;
public Transform target;
public Transform patrolRoute;

// State Tracking
public int patrolIndex;
public float chaseDistance;
```

The player will be assigned as the priority target. (The target variable will change dynamically because of the patrol route.) A chase distance of 10 means if the player gets within 10 units of the enemy, they will be chased until they manage to get farther than 10 units again.



Finally, the patrol logic is refined with an extra condition that favors chasing the player if the distance conditions are met.

```
if(priorityTarget) {
    // Keep track of our priority target
    float priorityTargetDistance = Vector3.Distance(transform.position, priorityTarget.position);

    // If the priority target gets too close
    // Follow it and highlight ourselves
    if(priorityTargetDistance <= chaseDistance) {
        target = priorityTarget;
        GetComponent<Renderer>().material.color = Color.red;
    } else {
        GetComponent<Renderer>().material.color = Color.white;
    }
}

if(target) {
    navAgent.destination = target.position;
}
```

To help visualize the state of our enemy, we make it change its color to red when it is in chase mode.

10) Playtest

Playtest your game and ensure that Enemy #1 follows the player all the time, while Enemy #2 endlessly patrols its route and only follows the player when they get too close.