2
Patrolling

Patrolling is a simple extension to pathfinding. Instead of just having a single target in mind, we might have two or more points. We might go back and forth between them, or travel in a never-ending loop.

In this chapter, you will learn about:

- How patrolling works
- Patrolling in Quick Path, React, and RAIN AI packages
- Getting to know more about behavior trees
- Creating patrols that go to different points in a level by not always following the same path

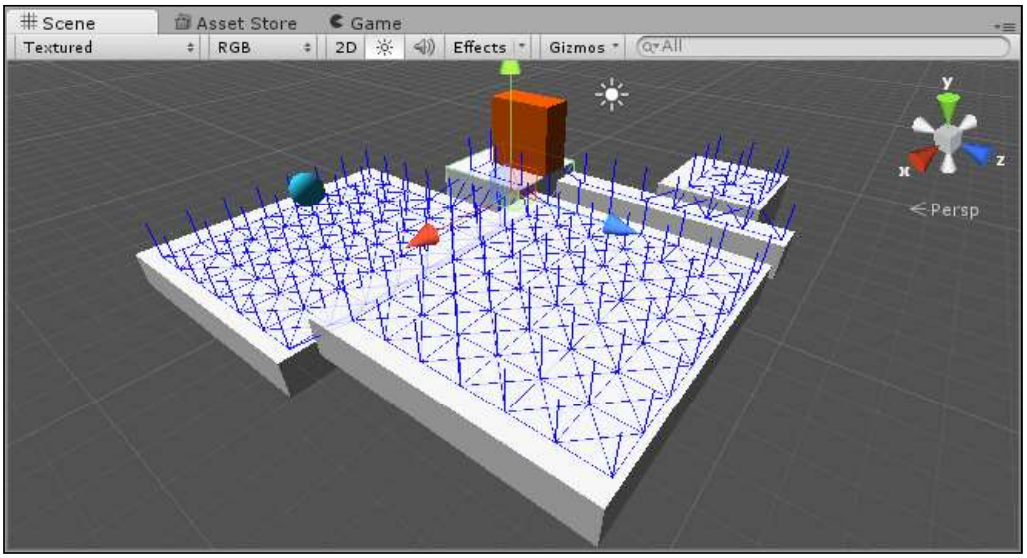
Patrolling is a way to get an object from point A to point B and then to point C, and so on. Pathfinding is still required to get from one waypoint to another, but here, we daisy-chain them into a larger, more meaningful path.

Quick Path AI

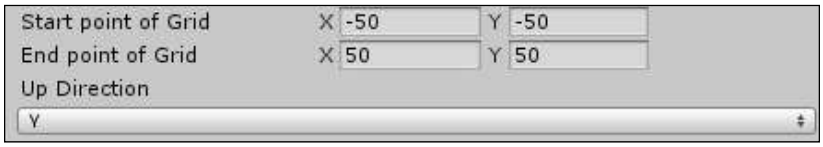
Quick Path is back again, with built-in capabilities to handle patrol. With its simple approach to AI, only a few straightforward steps are needed to get a scene finished. Here is a breakdown of these steps:

- Making the world ready for patrol
- Setting up the patrol script

We'll start by expanding on our world from *Chapter 1*, *Pathfinding* the quick path demo. Stretch out a couple of the blocks to make a larger surface area. Then, click on the terrain object (the parent of all the cubes forming the terrain), and in the **Inspector**, click on the **Bake** button. You can see what happens next in the following screenshot:



If the **Bake** function isn't covering all the areas, you'll need to check its grid dimensions. *X* remains the same in the world space, but *Y* is actually the *Z* axis. You might need to increase or decrease these numbers to cover everything in the scene:



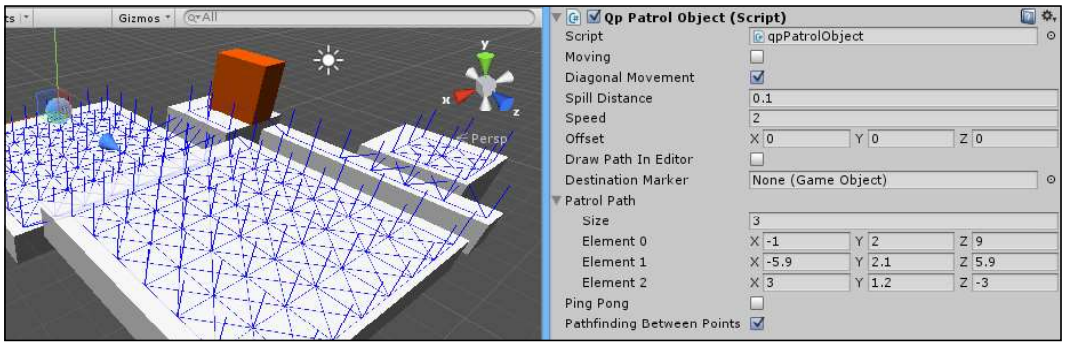
Quick Path converts the values of the *Y* or *Z* axis values internally. By default, it is set to *Y* as the *Up/Down* axis, but you can change this with the **Up Direction** parameter.

Now that we have a larger surface area to work with, we'll get the NPC object set up to patrol. If the object still has the following script, you will need to remove it, but then add the Quick Path patrol script. Then, the first thing to do in the **Inspector** panel for the patrol script is change the speed to `2`, as its default of 10 was rather fast.

Next, set **Spill Distance** to `0.1`. **Spill Distance** is how close you have to be to a waypoint before it picks the next waypoint as your target. If **Ping Pong** is checked, the NPC will stop at the end of its path and backtrack. If it is unchecked, at the end of its path, it will target the first position and start over.

Pathfinding Between Points is an option that helps it navigate around obstacles between waypoints. If your path is already clear, then you can keep this option off and save extra processing.

Finally, we have **Patrol Path**, which houses the waypoints for the NPC to travel on. Increase this to `3`, and then set the waypoints. A trick to figure out the values is to move your NPC in the scene to the waypoint positions that you want and then copy its position to one of the waypoints. So, select three points for your NPC to travel on. You can refer to the following screenshot for the settings:



Now, your game is ready to run, with your NPC navigating a course that you just set up.

React AI

React AI doesn't come equipped with a patrol script, so we provided one. We'll start with this behavior tree script and look at how it works and how to use it. Here are the steps to reproduce it:

1. Create a patrol script.
2. Create a patrol AI.
3. Set up the NPC patrol.

To start with, we've provided a script for you to use. In it, I started with the last script for pathfinding, and then I extended it to use a similar configuration to the patrol path in Quick Path's patrol script. Here are a couple of key points about this script:

- It is based on the `FollowThePlayer` script from the previous chapter.
- You can find the code in the book's contents at `\Scripts\React AI\Patrol.cs`.

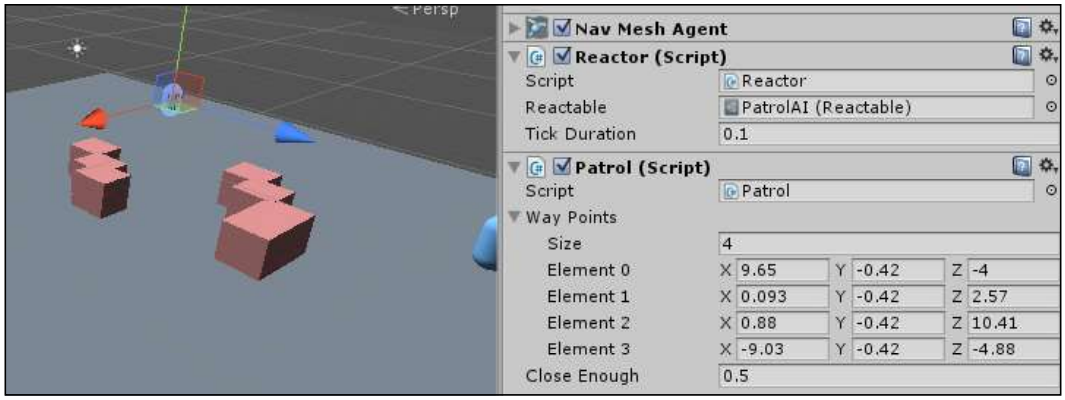
- It stores a public array of `Vector3`, so the **Inspector** UI can allow designers to set the waypoints.
- Instead of the target being a player, it is set to the next waypoint in the list. Once we are close enough, it selects the next waypoint. **Close Enough** is the float field that allows the inspector to find it.
- If there are no waypoints left to select, it starts over at the first waypoint.

However, now we need to create the user endpoint. Right-click on a folder in the **Project** tab and choose **Create | Reactable**. Name the reactable `PatrolAI`. In its **Inspector** UI, add the patrol script as one of the behaviors. Next, right-click on the **PatrolAI** asset and select **Edit Reactable**.

In the reactable, right-click on the root element and select **AddBranch | Sequence**. A sequence repeats all the steps in an order. Under the **Sequence** option, right-click and navigate to **Add | Leaf | Action**. Assuming that you only added the patrol script to its behaviors, it should automatically select **Patrol.Go** as its action. You can add notes to each step to help write a better story of what the AI is doing. When it is this simple, it does not matter so much, but many AIs will become more complex.

Next, the NPC needs to be set up to use this new patrol AI. Find the NPC in the previous chapter's React AI project. You'll need to remove the following AI that was on the NPC before, or create a new NPC. If you create a new NPC, do not forget to add the NavMesh agent so that it can navigate.

Add two components to the NPC: **Reactor** and **Patrol**. In the **Reactor** component, you will need to set the **Reactable** value to the **Patrol AI** asset that we created earlier. Then, in the patrol script, add some waypoints. Like we did for the Quick Path patrol script, we need to set the **Vector** locations for each of the waypoints.



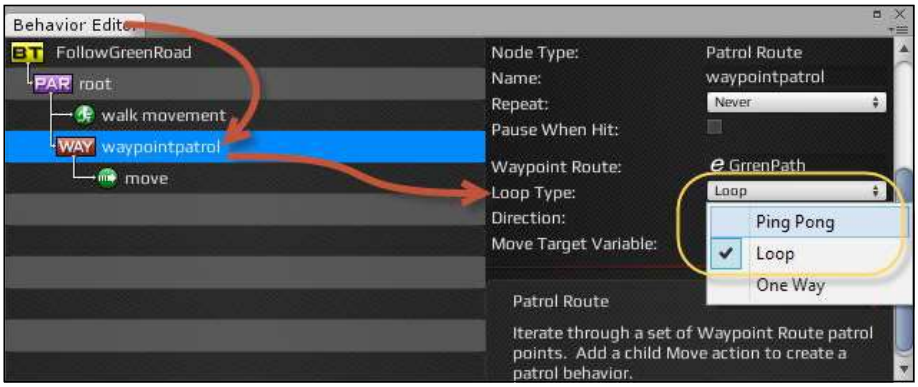
A tip to get exact numbers is to just move the NPC to the waypoint positions you want and then copy the position of the NPC to one of the waypoints.

Now your game should have a character who patrols from point to point.

RAIN AI

RAIN has this section put together pretty well. In reality, we only have one small section to change from the pathfinding demo, especially because the pathfinding demo had actually turned off the patrol feature.

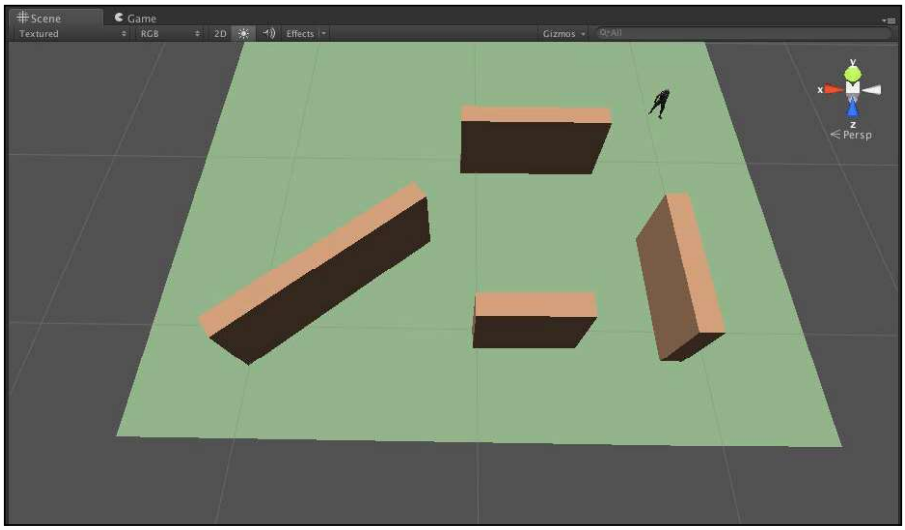
Start with the project for RAIN AI from *Chapter 1*, *Pathfinding*. From the menu, navigate to **RAIN | Behavior Tree Editor**. From the editor, select **FollowGreenRoad**. Under **Sequence** is a patrol route node called **waypointpatrol**; select it. Finally, we have a property called **Loop Type**. Presently, it is on **One Way**, which stops at the last waypoint. You can switch it to **Ping Pong** or **Loop**, as shown in the following screenshot:



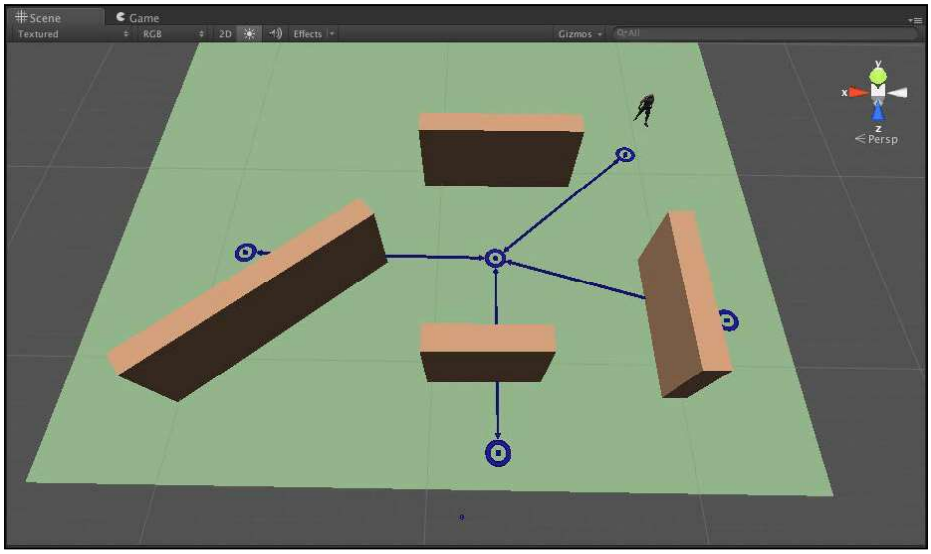
Ping Pong bounces you back and forth on the path, while **Loop** connects the last waypoint to the first to start over.

This works when creating a typical patrolling behavior, where a character loops along a path. However, what if we want to have a character patrol an area by walking around back and forth to different points without always following the same route? In RAIN AI, we can do this by using a waypoint network instead of a waypoint graph and updating our behavior tree to randomly pick different points in the level to go to.

To illustrate this, create a new scene, and like in our current patrol example, add a character and some blocks and create a navigation mesh. Separate the blocks a bit so that we can add different paths in between them. You can refer to the next screenshot to view this setup:

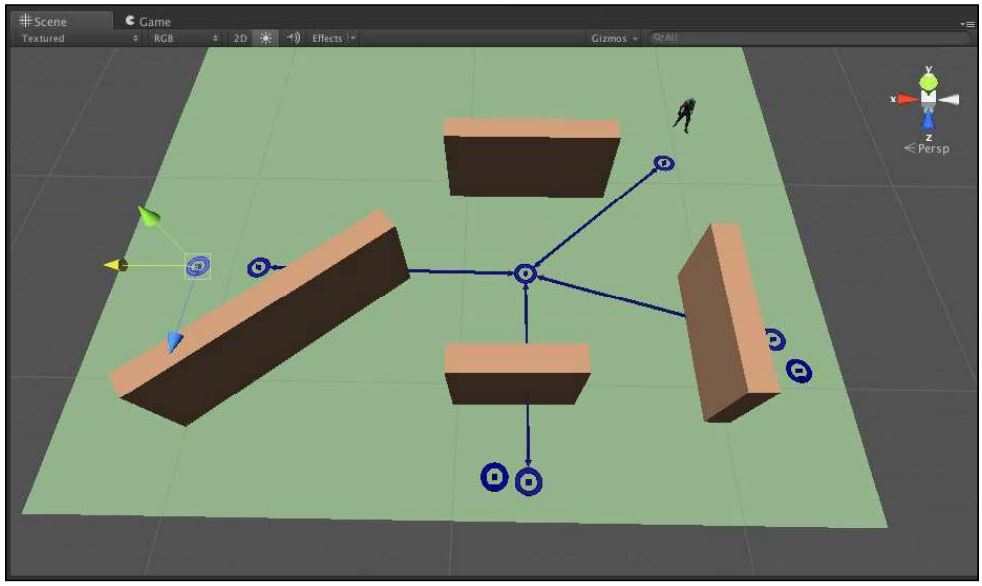


In this demo, we will have the character walk to different points outside the walls, but when patrolling, the character won't go in a circle outside the walls; instead, it will always walk through the middle. To do this, we will need a waypoint network similar to the one shown in the following screenshot:

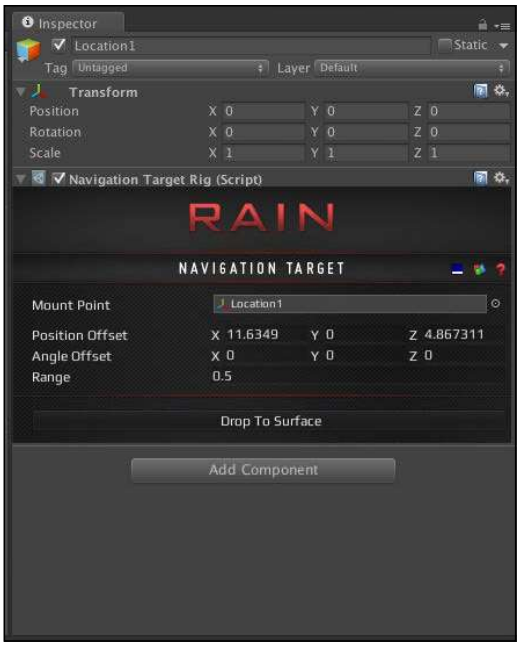


To add a waypoint, navigate to **RAIN | Create Waypoint Network**. Then, set up the network similar to how you set up a waypoint route, by creating different points. However, unlike a waypoint route, with a waypoint network you can also connect different points. To connect two waypoints, select them by pressing *Ctrl + Shift* and left-clicking the mouse and then click on **Connect** in the **RAIN Waypoint Network** component menu. Connect the points in a *plus sign* shape as illustrated in the previous screenshot. With this network, to walk from the side of one wall to another, the character will always need to walk through the middle of the scene.

The network waypoint describes how a character should walk to different spots on a level, but it actually doesn't contain the different points we can tell the AI character to go to. If we want to tell our character to go to a specific location, we need what RAIN calls a navigation target. A navigation target is just an object that contains a point in the scene that we can use with the rest of the AI system. You can create navigation targets by navigating to **RAIN | Create Navigation Target** and place them like you would place a waypoint. Create three navigation targets and place them on the side of three walls. We will follow a convention used in other RAIN examples and name the navigation targets `Location1`, `Location2`, and `Location3`, as shown in the following screenshot:



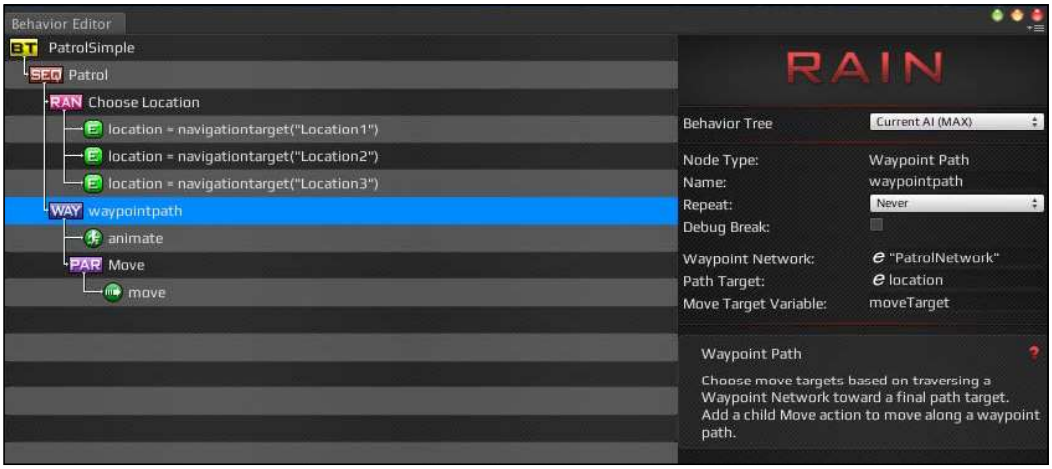
The Inspector panel should look like the following screenshot:



This is all of the scene setup that we need to specify routes and locations for the character to walk. However, we will need to customize the behavior tree to randomly choose different points to patrol to.

Create a new behavior tree for our character and call it `Patrol`. Open the behavior tree editor, and below the new root node, create a **Random** node by right-clicking on the root and navigating to **Create** | **Decisions** | **Random**. This creates a node that randomly selects one of its children to execute. Don't worry too much about how the different nodes work in the RAIN behavior tree for now; we will go into more detail about them in the next chapter. For now, create three expression nodes as children of the **Random** node by right-clicking on **Random** and navigating to **Create** | **Actions** | **Expression**. An expression node allows us to execute a single statement, which is called an expression in RAIN. Rename the expression nodes `Choose Location 1`, `Choose Location 2`, `Choose Location 3`. Then, in the expression field for the nodes, set the first to `location = navigationtarget(Location1)`, and do the same for the other location expression nodes, using numbers 2 and 3. These expression nodes create a variable location that is a randomly determined navigation target that we can use as a target to walk to. The setup should look like this:

All that is left is to add nodes to walk to the target. Right-click on the root node and navigate to **Create | Decisions | Waypoint Path** (not waypoint patrol like last time). In the waypoint path node, set the **Waypoint Network** field to `PatrolNetwork` (with quotes) to tell it which network to use. Set the **Path Target** field to location (without quotes), which is the variable we stored our random target to walk to. Finally, set up the rest of the tree as shown earlier, with an animation node and a child `Move` node. The final setup should look as shown in the following screenshot:



If you run the project now, the character will randomly patrol the area of the level by randomly walking from one navigation target location to the next and always walking through the middle of the level.

Summary

We were able to get patrolling operational in all three AIs. Each AI had its own approach.

For patrol as well as pathfinding, Quick Path had very few steps. If your need is mostly pathfinding, it does exactly what it claims to do easily. As for React, it does not have patrol out of the box, as we saw in the last chapter. However, it was not difficult to create a script that operates inside its behavior tree AI editor. This is a powerful system that you can use to allow designers to easily access and apply your awesome scripts, but you have to be comfortable programming to build the pieces for it. For RAIN, which made this about as easy as a big red button. With one setting changed, we changed the pathfinding AI into a patrolling AI. RAIN comes equipped with a huge variety of prebuilt character controlling; we looked at how to use a different waypoint system, a waypoint network, to give variety to our character when patrolling.

This concludes our chapters on setting up basic character pathfinding and movement. In later chapters, we will look in more detail at different aspects of this, such as animation and creating navigation meshes. In the next chapter, we will look at customizing our characters by investigating behavior trees in more detail. We will also learn to create more advanced setups using behavior trees.