Note

Optimizing Code

In the preceding code there are an awful lot of calls to `GetComponent()`. This can be quite heavy, processing wise. Instead you should declare a variable at the top of the code, then call GetComponent only once in the `Start()`, like this:

```
Animation anim;

void Start()
{
    anim=this.GetComponent<Animation>();
}
```

Then whenever you want to gain access to the animation anywhere else in the code you simply write:

```
anim.Play("run");
```

This waypoint system is used in the next section after the development of a self-motivated character is explained. However before we get to that we will examine another A* using mechanism that is built into Unity, the Navigation Mesh or NavMesh.

Unity Hands On

Setting up and using a NavMesh

Step 1: Download the *Chapter Five/NavmeshStationStarter.zip* and open it up in Unity.

Step 2: Align the view in the Scene window to look something similar to Figure 5.11.

FIG 5.11 Suggested alignment of Scene window for this exercise.

Step 3: Select the Main Camera in the Hierarchy. Then from the main menu, select GameObject > Align With View from the menu so that your camera is looking down when you press play.
Step 4: Create two new materials, red and blue, for the different agents so that we can tell them apart.
Step 5: Make sure you have the Navigation window open in your project. To do this, from the main menu select Window > AI > Navigation.
Step 6: Hold down the Shift key and select an area similar to Figure 5.12 for the agents to navigate.
Step 7: With your chosen area selected, in the Inspector, click on the Static drop-down list and select Navigation Static as shown in Figure 5.13.
Step 8: Next select the Navigation tab as shown in Figure 5.14 and its Bake tab and then click on the Bake button.

You should end up with something like Figure 5.15 where the navigable area turns blue.

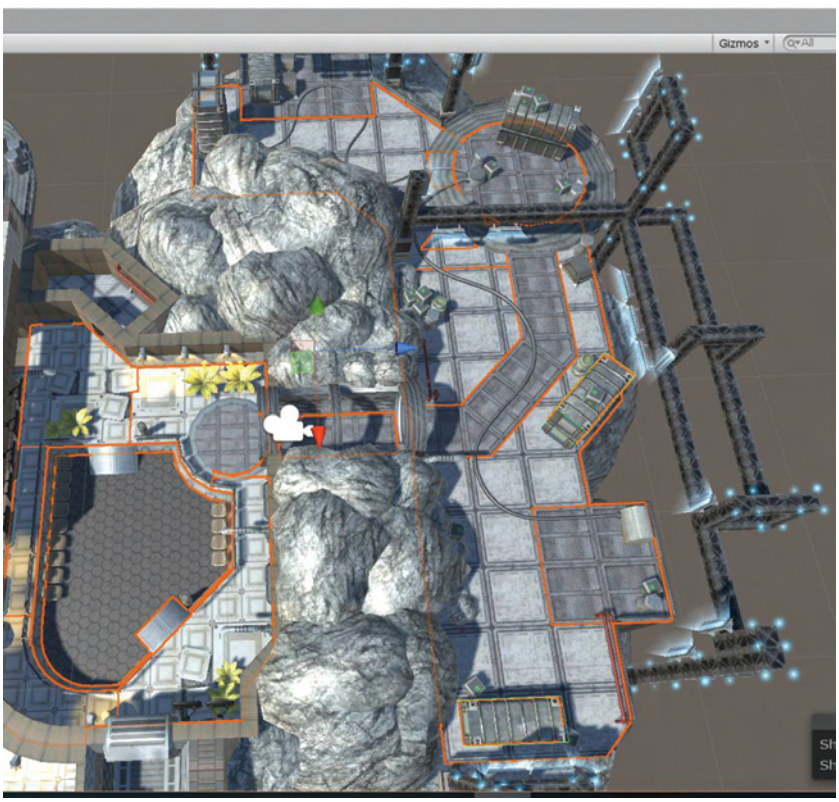

FIG 5.12 Area to select for agent navigation.

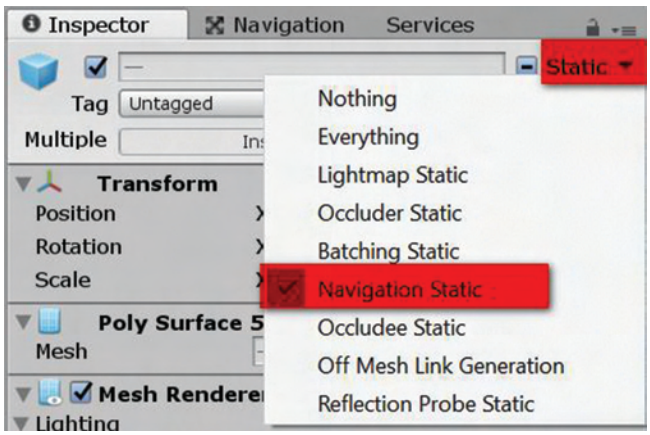

FIG 5.13 Setting mesh areas to navigation static.

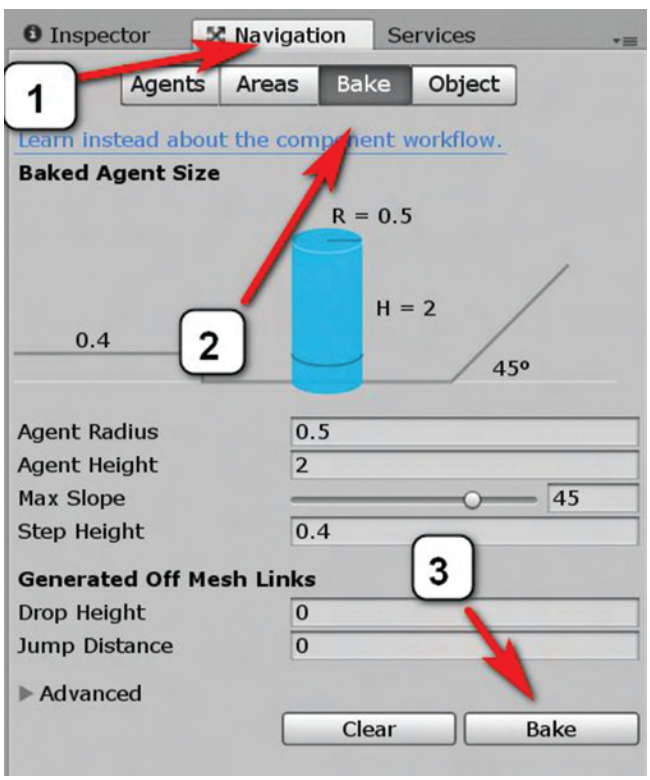

FIG 5.14 The navigation tab.

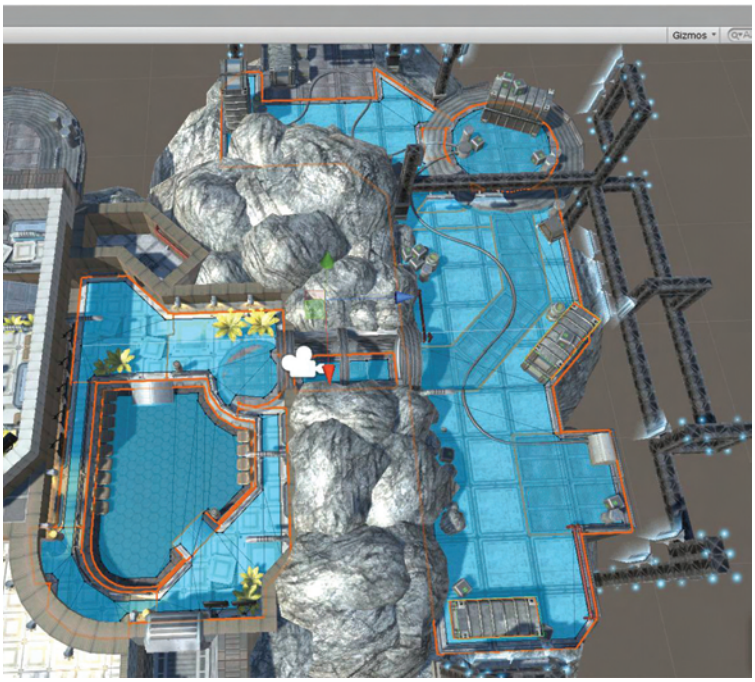

FIG 5.15 The appearance of a navigable area in the Scene.

Step 9: Make sure nothing is selected in the hierarchy. Right click in it and select 3D Object/Capsule. Rename the capsule RedAgent. Create a red material and add it to the capsule. Place it somewhere over to the right in the open. With the RedAgent still selected, press the Tag drop-down list in the Inspector and then Add Tag as shown in Figure 5.16.

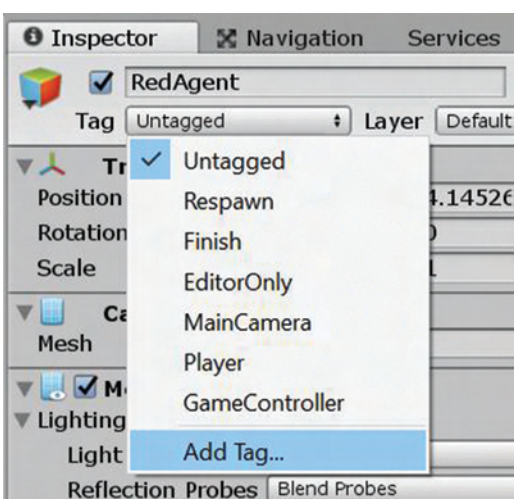

FIG 5.16 Creating and adding a tag.

Under Tags (List is Empty). Select the plus button and enter "ai" and hit save. Again, select the RedAgent in the hierarchy and the Tags dropdown list where you can now select "ai" as a tag.

Step 10: Create a new C# Script named *AIControl.cs* and enter the code in Listing 5.8. Attach to redAgent

Listing 5.8 AI controller code

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

public class AIControl : MonoBehaviour {

    public NavMeshAgent agent;

    // Use this for initialization
    void Start () {
        agent = this.GetComponent<NavMeshAgent>();
    }
}
```

Step 11: Save the script and, back in Unity, add an empty object to the Hierarchy called AgentManager. Create a second C# script named *AgentManager.cs* and add the code in Listing 5.9.

Listing 5.9 Agent management code

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

public class AgentManager : MonoBehaviour {

    GameObject[] agents;

    // Use this for initialization
    void Start () {
        agents = GameObject.FindGameObjectsWithTag("ai");
    }

    // Update is called once per frame
    void Update () {
        if (Input.GetMouseButtonDown(0)) {
            RaycastHit hit;
```

```
            if(Physics.Raycast(Camera.main.
                ScreenPointToRay(Input.mousePosition), out
                hit, 100)) {
                foreach(GameObject a in agents) {
                    a.GetComponent<AIControl>().agent.
                        SetDestination(hit.point);
                }
            }
        }
    }
}
```

Step 12: Again, save your code and head back into Unity.
Step 13: With the RedAgent selected, head over to the Inspector and scroll down to the bottom and select Add Component. Enter *Nav Mesh Agent* in the search box and click to select. When you press play, you will now be able to click anywhere on the previously baked area and your agent should now walk to the chosen spot. It does this using A* to calculate the path from one location to another using the NavMesh as a reference of points for where it is allowed to travel.
Step 14: You can adjust the agent's speed in the Inspector in the Nav Mesh Agent component section you added. Set it to about 10 as shown in Figure 5.17.

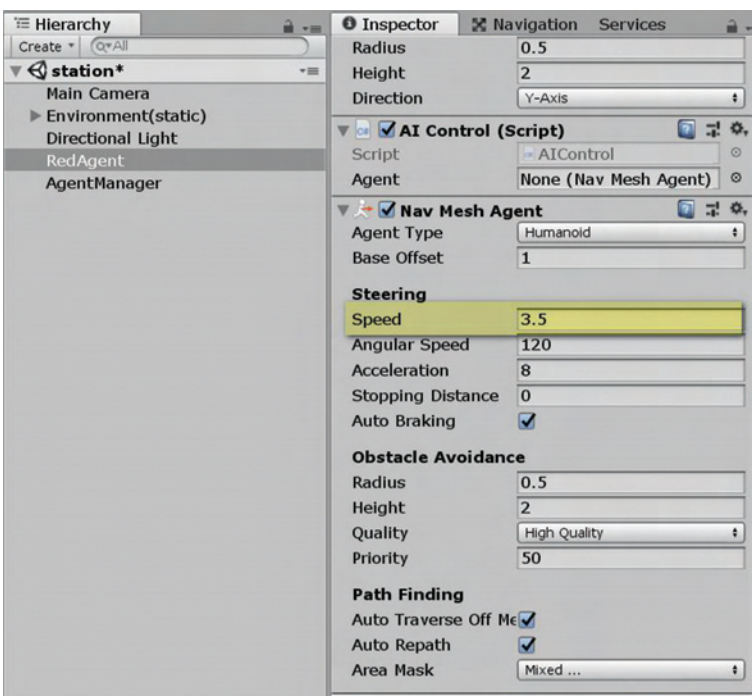

FIG 5.17 Setting the agent's speed.

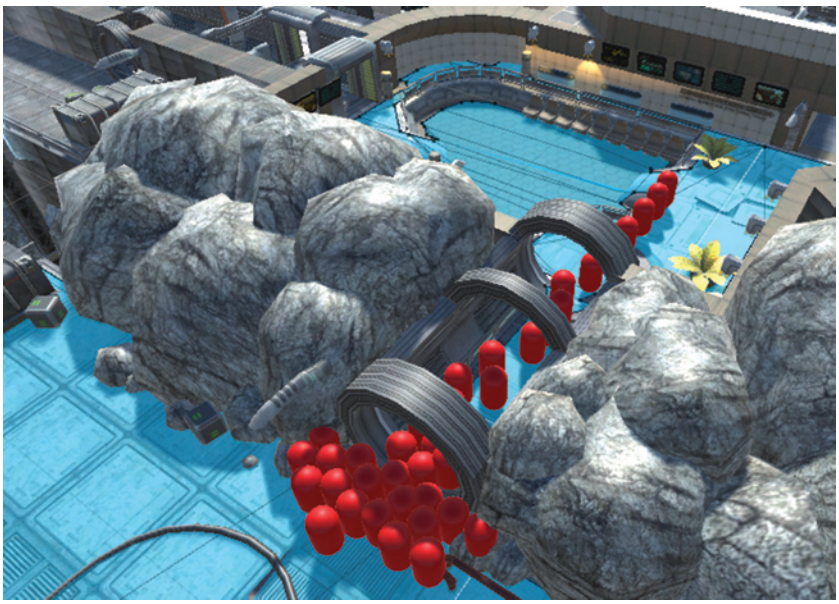

FIG 5.18 Multiple NavMesh agents moving across a mesh.

Step 15: Select the RedAgent in the Hierarchy and press CTRL + D several times to make several duplicates. They will all occupy the same world space until you hit play and tell them to move as demonstrated in Figure 5.18.

5.5 Finite State Machines

The most popular form of AI in games and NPCs is *nondeterministic automata*, or what are more commonly known as *finite state machines* (FSM). An FSM can be represented by a directed graph (digraph), where the nodes symbolize states and the directed connecting edges correspond to state transitions. States represent an NPC's current behavior or state of mind. Formally, an FSM consists of a set of states, *S*, and a set of state transitions, *T*. For example, an FSM might be defined as *S* = {*WANDER*, *ATTACK*, *PURSUE*} and *T* = {*out of sight*, *sighted*, *out of range*, *in range*, *dead*} as shown in Table 5.1.

In short, the state transitions define how to get from one state to another. For example, in Table 5.1, if the NPC is in state WANDER and it sights its opponent,

TABLE 5.1 State Transition for a FSM

	Transitions (*opponent is…*)				
State	***Out of Sight***	***Sighted***	***Out of Range***	***In Range***	***Dead***
WANDER	*WANDER*	*PURSUE*	—	—	—
PURSUE	*WANDER*	—	*PURSUE*	*ATTACK*	—
ATTACK	—	—	*PURSUE*	*ATTACK*	*WANDER*