5
Crowd Control

In this chapter, you will learn how to build large crowds into your game. Instead of having the crowd members wander freely, like we did in the previous chapter, we will control the crowds better by giving them directions on what to do. This material will be useful for a wide range of game use cases, such as planning soldier attacks by groups or directing flows of traffic in a car game.

In this chapter, you will learn about:

- Crowd-steering behaviors
- Using the Fame Crowd Simulation API to manage crowds
- Exploring ANT-Op to create more goal-directed crowds

An overview of crowd control

In *Chapter 4*, *Crowd Chaos*, we looked at creating crowds using wandering behaviors, where different crowd members worked individually to travel to different points. This works well for ambient crowds, but there was no working as a group. As there was no larger group-defined behavior or director managing crowds, our previous implementations required creating and configuring character AIs individually. Defining and configuring individual AIs is fine for smaller groups, but not practical when creating much larger crowds. In the demos in this chapter, we will look at crowds that work, or at least move, as a group. Moving AI characters in groups, also called flocks, has been a popular subject in AI for many years. The most popular system is called **Boids**, and it was designed in the 1980s by Craig Reynolds, a renowned computer graphics and AI developer, and the basic design is used in crowd AIs in most games today. In these systems, different simple steering behaviors are defined, such as moving to a target position or following a path, as well as behaviors to not collide with other agents or align to the same direction of nearby agents.

These simple behaviors are applied to large numbers of game characters (or in the original system, Boids), and when they run together, they move as groups the way you would expect them to. These simple behavior combinations give surprisingly realistic results considering how simple the individual steering behaviors are.

Steering algorithms are simple to implement, but instead of coding something from scratch, we will use Unity plugins. The two plugins we will be focusing on, the Fame Crowd Simulation API and ANT-Op, are not general-purpose AI systems like RAIN; instead, they are focused just on crowd management. This is a popular trend in the AI world. Sometimes, instead of a large AI system, it's best to look for specific tools for specific AI tasks. Don't worry about combining multiple AI subsystems while designing the AI for your games, as this can often give you the best result.

The Fame Crowd Simulation API focuses, as you might expect, on crowds with a system design similar to Boids. It allows you to customize different values for the various steering behaviors and has a GUI interface that makes forming and directing crowds very straightforward. ANT-Op is based on simulating ants. Looking at ants might sound strange, but ant simulation is actually a popular topic in the AI world, since ants work really well as a group. Both of these plugins are useful when creating controlled crowds.

More details about Craig Reynolds' original flocking and steering systems can be found in his papers online:

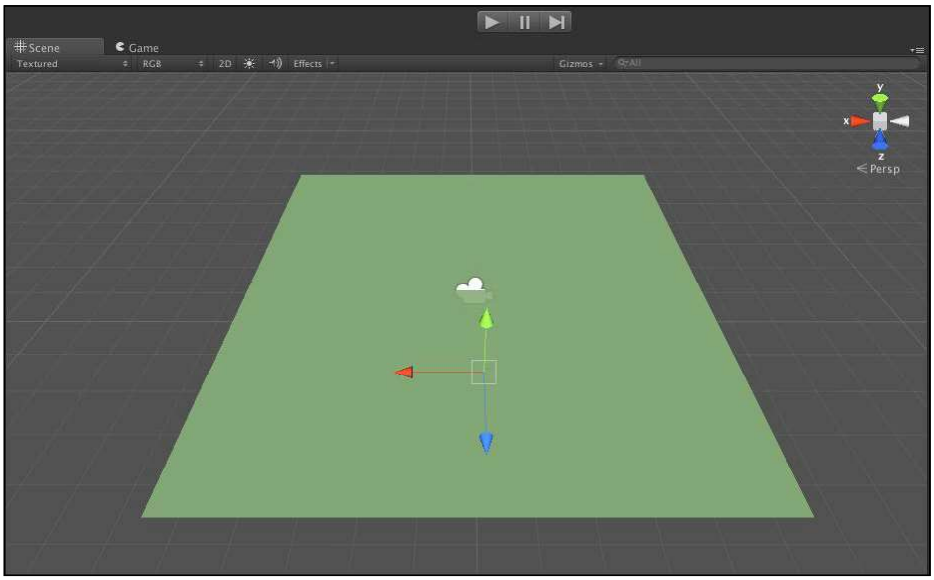
- *Steering Behaviors for Autonomous Characters*: `http://www.red3d.com/cwr/papers/1999/gdc99steer.html`
- *Flocks, Herds, and Schools: A Distributed Behavioral Model*: `http://www.red3d.com/cwr/papers/1987/boids.html`.

The Fame Crowd Simulation API

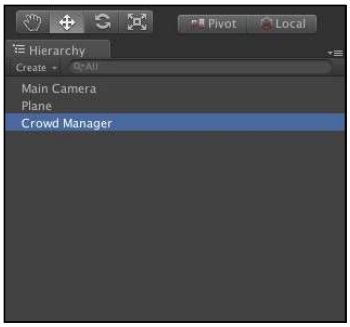
The Fame Crowd Simulation API by TechBizXccelerator is available at Unity Asset Store under the name Crowd Simulator API for $45 at the time of writing this book. It allows you to create groups and customize steering behaviors. For our crowd demo, we will create a demo with many spaceships traveling in a group.

Setting up a scene with Fame

To create our demo, first create a new scene with a plane as the ground. Like most of the AI plugins in this book, Fame Crowd Simulation also supports Unity's built-in terrain system, but for this demo, a basic ground plane will be fine. Fame also uses a singleton pattern that has one class that manages everything for crowd management, so create an empty GameObject and call it **Crowd Manager**. Then, import Fame if it is not already in your project, and attach the `FameManager` script from `Fame Assets/FameScripts/FameManager.cs` to the **Crowd Manager** GameObject. This initial setup is shown in the following screenshot:



The following should be the hierarchy:



This is our basic Fame scene setup with a **Crowd Manager** object.

The **FameManager** object will be our main point of interaction with Fame, and we can interact with it from anywhere in our game code.

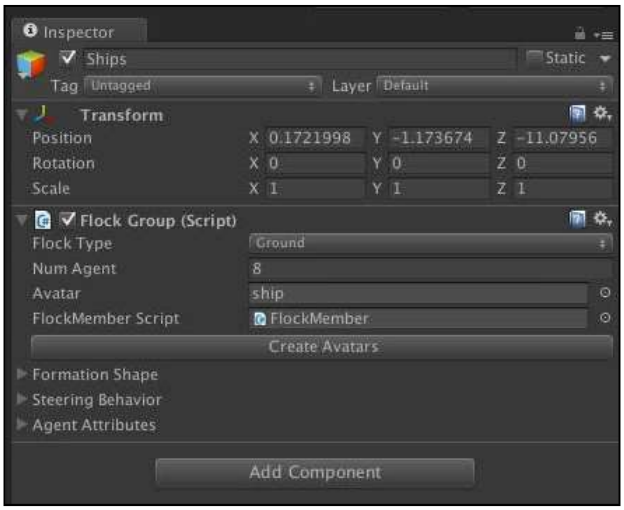
Unlike the other systems we saw where we create the AI characters individually to define our crowds, in Fame, we will define a group of characters, also called a flock. Add the ship model from *Chapter 3*, *Behavior Trees*, and then create another empty GameObject and call it `Ships`. Then, attach the `FlockGroup.cs` script to it from `Assets/FrameScripts`.

To avoid the warning about not having a terrain in **FAME settings,** disable **Enable Terrain Effect** in **FameManager** and **Get Info From Terrain GameObject** in Fame terrain.

`FlockGroup` is the main class to hold a group of characters, and it has several settings we can customize:

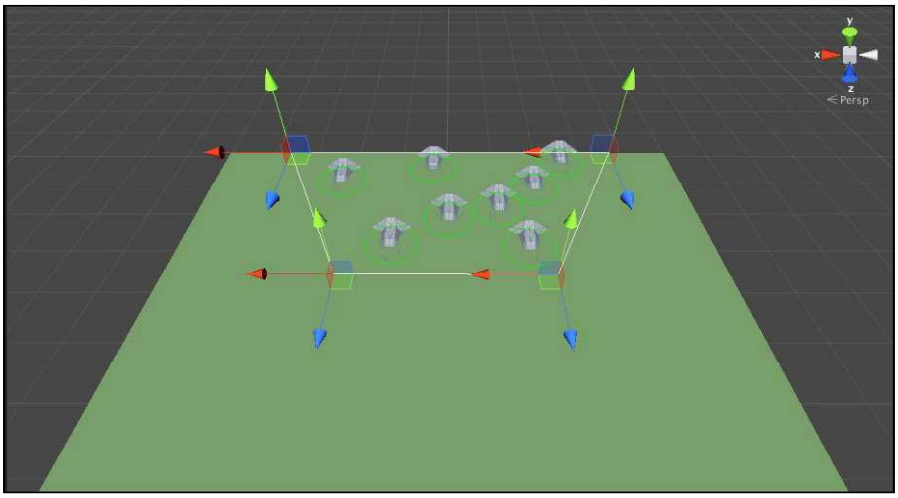
- **Flock Type**: This is **Ground** or **Air** and defines whether the AI characters will move on a plane (ground) or move in all three axes (air). If set to **Air**, characters will ignore any terrain. You might think we would set our ships to air, but we want them to all hover at the same height, so keep this set to ground.
- **Num Agent**: This is the number of agents (or AI characters) in the group. This can be a very high number as Fame is efficient, but for our demo, we will set this to 8.
- **Avatar**: This is the GameObject set to be the individual members of the crowd. For our demo, this is the ship model.
- **FlockMemberScript**: You can create a custom script to define how the members of your crowd will act. For this demo, we will keep things simple and use Fame's default `FlockMemberScript.cs` script.

The following screenshot shows our Fame **FlockGroup** settings for the ship group:



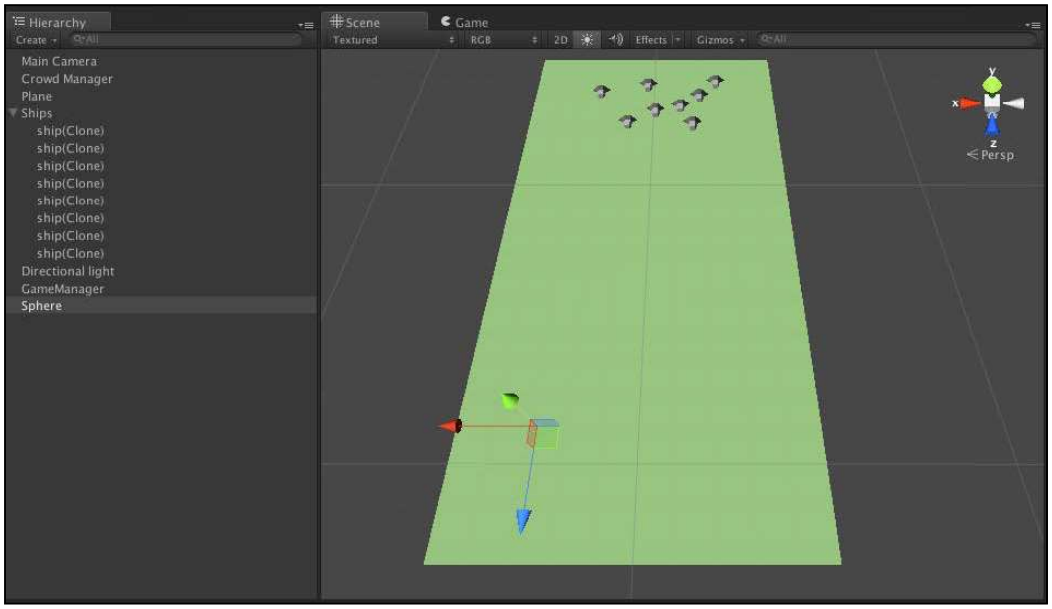
Setting up a group

Next, we need to create the initial formation we want our group to be in. With the **Ships** GameObject selected, you will see three connected gizmos in the Unity 3D view that represent the shape of the initial group. Go to **Formation Shape** in the **Inspector** panel for **Ships** and click on **Add Point**. This creates a fourth point for the shape of the group. Arrange the points into a pyramid-like shape and click on **Create Avatars**. A set of ships in a group will be created:



This is our group setup that shows the control points that define the boundary for our group. You can see our ship group shape with eight ships has been created.

Creating a group object was really easy with Fame. Now, we need to set up a way for our crowd to move. We will have the ships move across the plane to a target point. Set the *Z* scale of the plane to a larger value, create a sphere object to be our target position, and place it across the plane. The screen should look as follows:



You can see a group of ships with a target placed.

When we send the group, or flock, the command to move to the target location, it will move the position of the flock to match it, which isn't necessarily the center of the flock. In this demo, the center of the flock is the upper-left corner from the camera view, which is why the target sphere is offset to the upper-left corner.

Now that we have a target element in the scene, we need to give Fame a command to move the group to it. Fame supports path following, but as most of the time we will want our groups to move dynamically, such as chasing a player, we will look at how to move the crowd by code. We have a lot of ways to set up a script to control the flock, but we'll take the simplest route and use a game manager. Create another empty GameObject and call it **GameManager**. Then, create a new script, `GameManager.cs`, and attach it to the **GameManager** object.

Add the following code to it:

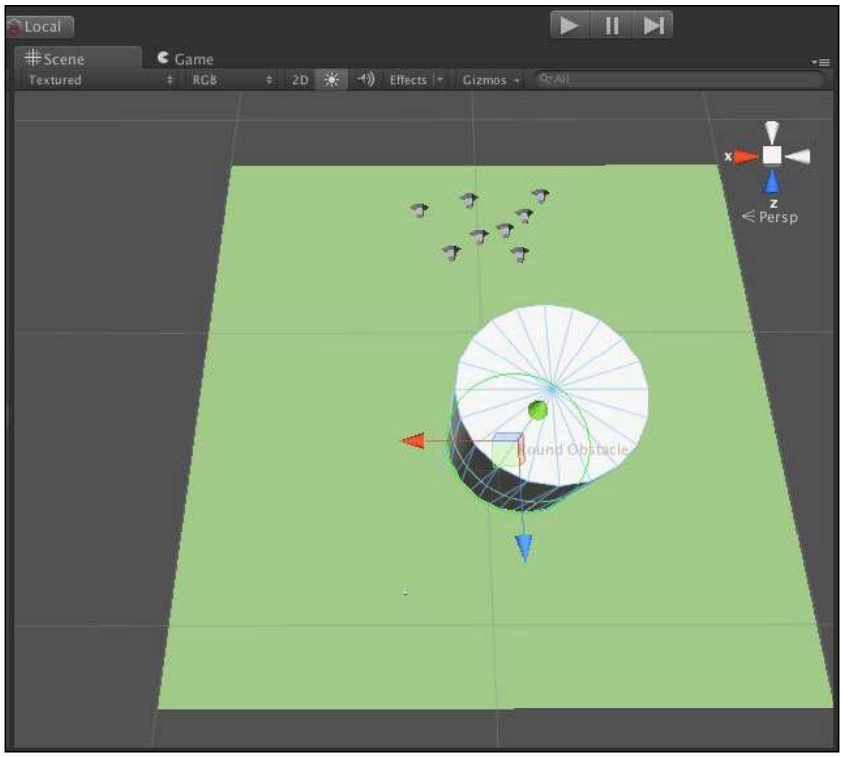
```
using UnityEngine;
using System.Collections;

public class GameManager : MonoBehaviour {

  public Transform target;

  void Start () {

  }

  void Update () {
    FlockGroup group = FameManager.GetFlockGroup(1); // Get first
flock

    group.MoveGroupAbs(target.position);
  }
}
```

In this code, we have a public variable for the target for the flock. In the Unity editor, drag the **Sphere** target object to this field to set its value. Then, in the update, we just get the flock group from Fame manager (which is a singleton class with static methods, so we don't need a reference to it). Then, we just use `MoveGroupAbs` that tells the group to move to a specific location. (There is also a method called `MoveGroup` that takes in an offset instead of an absolute position you can use if you want the crowd to move in a general direction instead of to a specific point.) If you run the demo now, the ships will travel down the ground to the target point. As you can see, setting up a crowd and giving directions for it to move is very easy and straightforward with Fame.

Adding obstacles to Fame

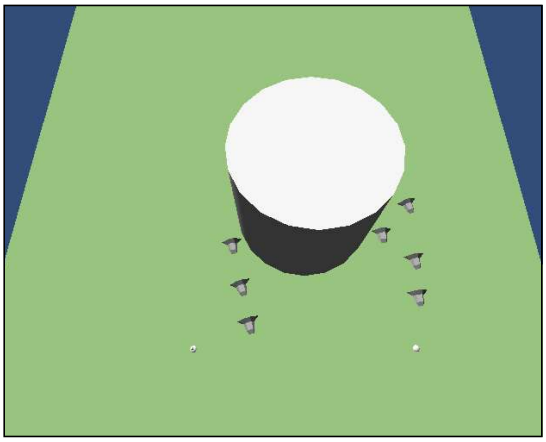
Next, let's add an obstacle inbetween the ships and their target. Increase the size of the ground plane by increasing its *X* scale. Then, add a cylinder to the middle of the scene to be an obstacle for the ships. To have this obstacle recognized by Fame, add the Fame obstacle script, `FameObstacle.cs`, to it. Fame allows two types of obstacles: round (circles) and 2D polygons. You specify the polygon ones, the same as we did for group shapes, by modifying control points. For our obstacle, we just need it to match the radius of the cylinder. In this example, the cylinder has a scale of **50**, which makes its radius 25, so set the obstacle's **radius** to `25`.

The demo should now look like the following screenshot:



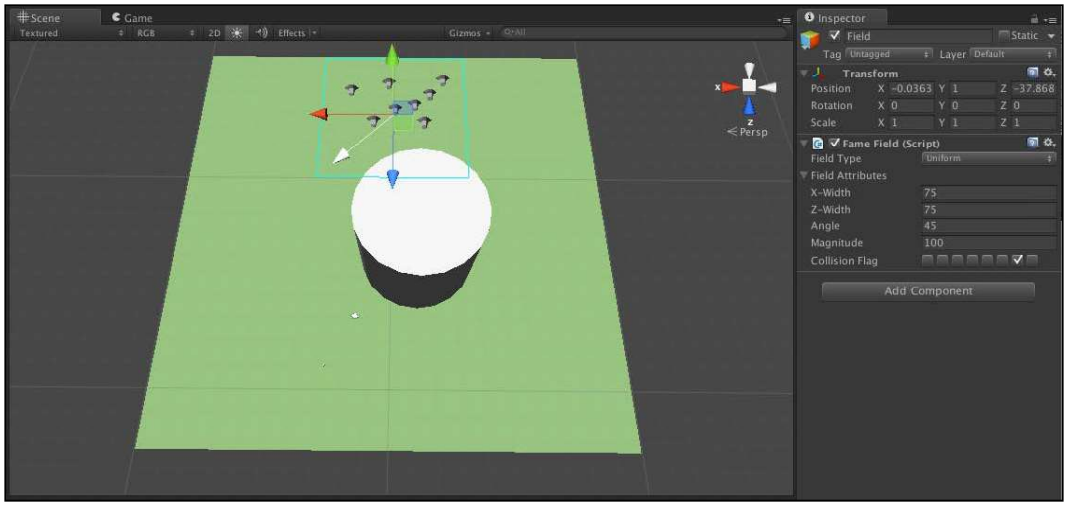
This is the obstacle setup for our ship group.

If you run the demo now, the ships will go to their target and smoothly avoid the obstacle. You can see in the next screenshot how the ships avoid our obstacle:



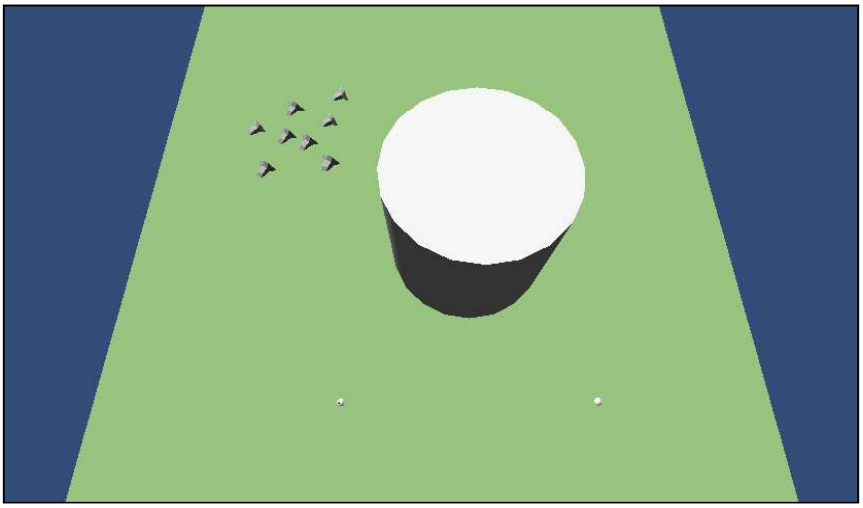
Adding vector fields to Fame

We just saw how to add obstacles to our Fame crowd scene. Right now, all of our ships split when avoiding the cylinder, about half to the left and the other half to the right. However, we want all of them to move to the left. To do this, we can vector fields to the scene. Vector fields are a popular way of adding directional *hints* to a scene, and they are defined areas that have directional vectors that are associated with them. When a character is inside the field, the vector helps move the character in the desired direction. Vector fields can be a powerful tool for level design and are easy to add to your scene. To see how easy it is to add them to the scene, add a new empty GameObject to the scene and name it `Field`. Then, attach the `FameField.cs` script from `Assets/Fame Assets/FameScripts` to it. The field can be rectangular, called **Uniform** or **Circular**. Select **Uniform** for the type and set **x** and **z widths** to **75**. Then, set the angle to **45** and the magnitude to **100**. The white arrow visualizes the angle for the vector field. Place the field over the initial positions for the ships, as shown:



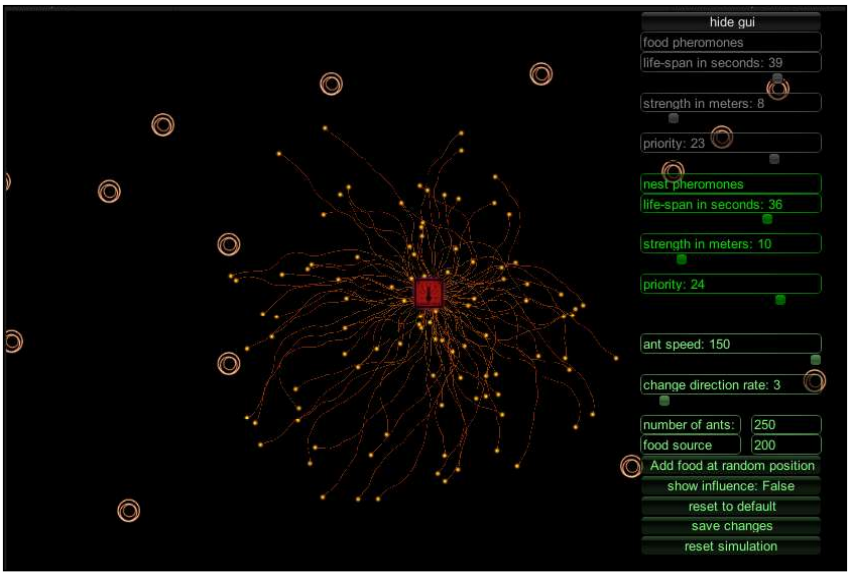
If you run the demo now, ships will all veer to the left before avoiding the cylinder. For larger levels, you can set up multiple vector fields; they are a good way to control the general movement of AI characters.

This Fame demo showed you the basics you'll need to create organized crowds for most games: creating crowds, giving them directions to move, and adding obstacles and vector fields. Next, we will look at another specialized AI plugin, ANT-Op. You can see the ship group changing its direction because of the vector field. You can also notice how our crowd uses the vector fields to direct ship movement to the left:



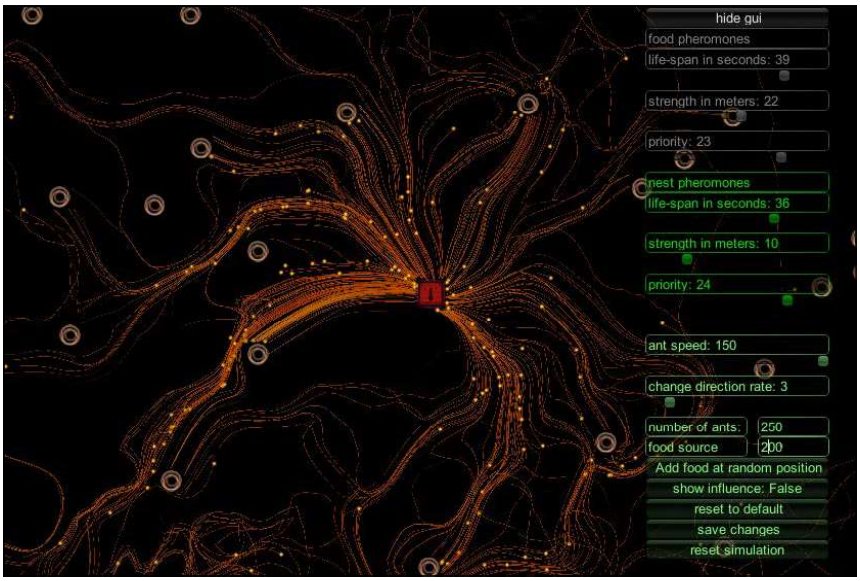
ANT-Op

Sometimes for your games, you might want to create crowd AIs that act in a very unique way, and instead of using an existing AI plugin, you might want to create a crowd AI from scratch. This would be done in the case of an ANT-Op AI. As we mentioned before, ant behavior is a popular topic in AI research and computer science in general. Initially, ants work independently and give off pheromones that are sensed by other ants to communicate messages. For example, when ants start searching for food, they give off pheromones as they search. When they find food, they give off different pheromones as they bring it back to the colony, which directs the next ants when searching for food. ANT-Op, by Gray Lake Studios, is available on Unity Asset Store for $75, and it simulates this ant food search process. Unlike the other AI plugins in this book, Ant-Op isn't really designed to be brought into an existing game; it's more of a technical demonstration that is a simulation you can use to see interesting AI at work and hopefully use it to inspire complex AI designs for your games. To start the demo, import ANT-Op and double-click on **Test Scene** to open it. The scene will initially be blank, but if you start the demo, you can see the simulation start. In the following screenshot, you can see ANT-Op in action:



You can see ANT-Op simulating an ant colony. The lines represent pheromones from the ants.

The options on the right are different settings for the simulation. You can play by changing the values, and these can be saved to an XML file for later reloading. Running the simulation provides a complex visualization, which you can see in the following screenshot:



Again, this probably isn't something you would bring into your game as is; it's to generate ideas for your AI. The source files can be reviewed for ideas on creating your own AI. Don't underestimate what you can learn by looking at AI simulations like this.

Summary

This concludes the last of two chapters on crowd AI. Where the previous chapter focused on defining crowds by defining wander behaviors for different characters individually, in this chapter we focused on defining groups as a whole. We discussed steering-based group design and looked at the Fame Crowd Simulation API that you can use to set up crowds easily, give them direction, and have them adjust steering based on other factors in the environment, obstacles, and vector fields. We then discussed defining your own crowd AI for more unique systems and looked at ANT-Op as an example of this. This should give you all the info you need to create all kinds of crowds for your games.

In the next few chapters, we will turn the focus to having AI characters interact with their environment. In *Chapter 6*, *Sensors and Activities*, we will look at having our characters sense things in the environment and react to them.