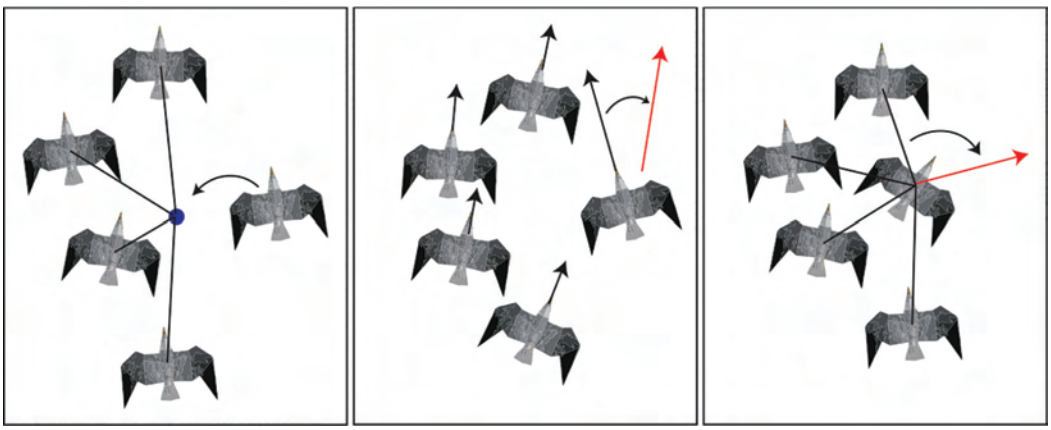

FIG 5.20 Flocking rules: (a) Move towards average group position, (b) align heading with average group heading, (c) avoid others.

Flocking creates a moving group with no actual leader. The rules can be increased to take into consideration moving toward a goal position or the effect of wind. These rules are used to create a flock of birds in the following workshop.

Unity Hands On

Flocking

Step 1: Download *Chapter Five/flockStarter.unitypackage* from the website and open the *testFlock* scene. Play. You will see a radar showing the position of seagulls created with the globalFlock script attached to the camera. To move the camera, use the arrow keys. This will allow you to look at parts of the flock. Currently the seagulls do not move. In the globalFlock script, 100 seagulls are being created. If you want more, change the value where it currently states 100. If your computer starts to slow down, you may need to reduce the flock size.

Step 2: Open the flock script and add the code in Listing 5.15.

Listing 5.15 Starting the flock code

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

public class Flock: MonoBehaviour {
    float speed = 0.001f;
    float rotationSpeed = 5.0f;
    Vector3 averageHeading;
    Vector3 averagePosition;

    float neighbourDistance = 2.0f;
```

```
    void Start()
    {
        speed = Random.Range(0.1f, 1f);
    }

    void Update()
    {
        transform.Translate(0, 0, Time.deltaTime
            * speed);
    }
}
```

Step 3: Note the flock script is attached to the *SeagullPrefab* in the Project.
Step 4: Play. Each seagull will have a random speed setting and will be flying in a straight line. You can use the arrow keys to follow the birds.
Step 5: Let us apply the first flocking rule to the population. Modify the flock script as shown in Listing 5.16.

Listing 5.16 Script to make seagulls move to the average location of their neighbors

```
    float speed = 0.001f;
    ...
    void Update()
    {
        if (Random. Range(0, 5) < 1)
            ApplyRules();

        transform.Translate(0, 0, Time.deltaTime
            * speed);
    }
    void ApplyRules()
    {
        GameObject[] gos;
        gos = GameObject.FindGameObjectsWithTag("Seagull");
        Vector3 vcentre = Vector3.zero;
        Vector3 vavoid = Vector3.zero;
        float gSpeed = 0;
        float dist = 0;
        int groupSize = 0;

        foreach (GameObject go in gos)
        {
            if (go != this.gameObject)
            {
                dist = Vector3.Distance(go.transform.
                    position, this.transform.position);
```

```
            if (dist <= neighbourDistance)
            {
                vcentre += go.transform.position;
                groupSize++;
            }
        }
    }
    if (groupSize != 0)
    {
        vcentre = vcentre / groupSize;

        var direction = vcentre - transform.position;
        if (direction != Vector3.zero)
            transform.rotation = Quaternion.
            Slerp(transform.rotation,
                Quaternion. LookRotation(direction),
                rotationSpeed * Time.deltaTime);
    }
}
```

Step 6: Play. Small flocks will form in the population. In the script, Random.Range is used to apply the rules about one in five game loops. This ensures that not all the birds have the rules that run each game loop. If this happens, the application runs very slowly and gets slower the more birds you have. Note that the average position is also only determined for neighboring birds, not the whole population. This is determined by the neighbourDistance variable. If you make this value larger, you will get one big flock.
Step 7: Currently, because the birds have random speeds, birds will eventually break away from their flocks because they cannot keep up or are going too fast. To keep them together, the second rule is applied. Birds in the flock match the average speed. To achieve this, modify the flock script to that in Listing 5.17.

Listing 5.17 Applying an average speed to a flock

```
void ApplyRules()
{
    ...
    foreach (GameObject go in gos)
    {
        if (go != this.gameObject)
        {
            dist = Vector3.Distance(go.transform.
                position, this.transform.position);
            if (dist <= neighbourDistance)
            {
                vcentre += go.transform.position;
```

```
                groupSize++;
                gSpeed = gSpeed + go.
                    GetComponent<Flock>().speed;
            }
        }
    }

    if (groupSize != 0)
    {
        vcentre = vcentre / groupSize;
        speed = gSpeed / groupSize;

        ...
    }
}
```

Step 8: Play. Averaging of the speed will help keep the formed flocks together.
Step 9: Finally, adding the third rule will enable the birds to keep out of each other's way. Before changing the code, observe the current flocking movement. Once a bird is flying in a circular pattern within the flock, it stays with that pattern. Now change the flock script to that in Listing 5.18.

Listing 5.18 Adding avoiding behavior to flocking script

```
void ApplyRules()
    {
        ...

        foreach (GameObject go in gos)
        {
            if (go != this.gameObject)
            {
                dist = Vector3.Distance(go.transform.
                    position, this.transform.position);
                if (dist <= neighbourDistance)
                {
                    vcentre += go.transform.position;
                    groupSize++;
                    if(dist < 0.5)
                    {
                        vavoid = vavoid + (this.transform.
                            position - go.transform.
                            position);
                    }
                    gSpeed = gSpeed + go.
                        GetComponent<Flock>().speed;
                }
```

```
            }
        }

        if (groupSize != 0)
        {
            vcentre = vcentre / groupSize;
            speed = gSpeed / groupSize;
            Vector3 direction = (vcentre + vavoid) –
                transform.position;
            ...
        }
    }
```

Step 10: Play. Take a close look at the birds' behavior. You will notice that they now dart out of the way in a similar movement to what is observed in real flocking birds.

So far, the flocks created are reminiscent of crows or vultures circling in the sky around their next meal. This could be used for a dramatic effect in a game to point out the position of something sinister in the game environment. More often than not, flocks used for ambience tend to be traveling across the scene. To achieve this type of flocking, more rules can be added.
Step 11: To make the birds move across an endless sky, we can add a *wind* value. This is a vector indicating the direction of travel. Modify the flock script to that in Listing 5.19.

Listing 5.19 Directing flock to fly in a set direction

```
void ApplyRules()
    {
        ...
        Vector3 vavoid = Vector3.zero;
        float gSpeed = 0;
        Vector3 wind = new Vector3(1,0,1);
        float dist = 0;
        int groupSize = 0;

        ...

        if (groupSize != 0)
        {
            vcentre = vcentre / groupSize + wind;
            speed = gSpeed / groupSize;

            Vector3 direction = (vcentre + vavoid) -
                transform.position;
            ...
        }
    }
```

Step 12: Play. The added wind vector will cause the birds to fly along the same course. The birds will still flock together, but instead of circling they will form close-streamed groups.
Step 13: If you want the birds to fly across the screen to a particular location, a goal position can also be added to the rules. Modify the flock script to that in Listing 5.20.

Listing 5.20 Directing flock to fly to a specific location

```
//Flock.cs ---------
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Flock : MonoBehaviour {

    float speed = 0.001f;
    float rotationSpeed = 5.0f;
    Vector3 averageHeading;
    Vector3 averagePosition;

    float neighbourDistance = 2.0f;

    void Start()
    {
        speed = Random.Range(0.1f,1f);
    }

    void Update ()
    {
        if(Random.Range(0,5) < 1)
        ApplyRules();
        transform.Translate(0, 0, Time.deltaTime
            * speed);
    }

    void ApplyRules()
    {
        GameObject[] gos;
        gos = GameObject.FindGameObjectsWithTag("Seagull");

        Vector3 vcentre = Vector3.zero;
        Vector3 vavoid = Vector3.zero;
        float gSpeed = 0;

        Vector3 wind = new Vector3(1,0,1);
        Vector3 goalPos = globalFlock.goalPos;
```

GlobalFlock

```
        float dist = 0;

        int groupSize = 0;
        foreach (GameObject go in gos)
        {
            if(go != this.gameObject)
            {
                dist = Vector3.Distance(go.transform.
                    position, this.transform.position);
                if(dist <= neighbourDistance)
                {
                    vcentre += go.transform.position;
                    groupSize++;

                    if(dist < 0.5f)
                    {
                        vavoid = vavoid + (this.transform.
                            position - go.transform.
                            position);
                    }

                    gSpeed = gSpeed + go.
                        GetComponent<flock>().speed;
                }
            }
        }

        if(groupSize != 0)
        {
            vcentre = vcentre/groupSize + wind + (goalPos -
                this.transform.position);
            speed = gSpeed/groupSize;

            var direction = (vcentre + vavoid) - transform.
                position;
            if(direction != Vector3.zero)
            transform.rotation = Quaternion.
                Slerp(transform.rotation, Quaternion.
                LookRotation(direction), rotationSpeed
                * Time.deltaTime);

        }
    }
}

//globalFlock.cs -----------
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

GlobalFlock

```
public class globalflock: MonoBehaviour {

    public GameObject gull;

    static public Vector3 goalPos;

    void Start()
    {
        //create seagulls
        for(int i = 0; i < 100; i++)
        {
            Vector3 pos = new
            Vector3(Random.Range(-10,10),0,Random.
                Range(-10,10));
            Instantiate(gull, pos, Quaternion.identity);
        }

        goalPos = new Vector3(0,0,0);
    }

    void Update ()
    {
        if(Random.Range(0,10000) < 50)
        {
            goalPos = new
            Vector3(Random.Range(-1.0f,1.0f),0,
                Random.Range(-1.0f,1.0f));
        }

    }
}
```

Step 14: Play. The birds will form into groups flying toward a single goal location.

These flocking rules have been used in movies to create flocking characters. They were used in *Batman Returns* to create realistic swarms of bats and in *The Lion King* for a herd of wildebeest.

The rules can also be modified in games to develop intelligent group enemy behavior and optimize processing speeds. For example, in the workshop that introduced breadcrumb path finding, if there were hundreds of enemies, having them all process the breadcrumbs could prove computationally costly. Furthermore, if each one even had to run the A* algorithm, it could certainly slow down the performance. However, if just one *leader* followed the breadcrumbs, the rest could flock and follow that leader.