How it works...

These behaviors rely on `Seek` and `Flee` and take into consideration the target's velocity in order to predict where it will go next; they aim at that position using an internal extra object.

Arriving and leaving

Similar to `Seek` and `Flee`, the idea behind these algorithms is to apply the same principles and extend the functionality to a point where the agent stops automatically after a condition is met, either being close to its destination (arrive), or far enough from a dangerous point (leave).

Getting ready

We need to create one file for each of the algorithms, `Arrive` and `Leave`, respectively, and remember to set their custom execution order.

How to do it...

They use the same approach, but in terms of implementation, the name of the member variables change as well as some computations in the first half of the `GetSteering` function:

1. First, implement the `Arrive` behaviour with its member variables to define the radius for stopping (target) and slowing down:

```
using UnityEngine;
using System.Collections;

public class Arrive : AgentBehaviour
{
    public float targetRadius;
    public float slowRadius;
    public float timeToTarget = 0.1f;
}
```

2. Create the `GetSteering` function:

```
public override Steering GetSteering()
{
    // code in next steps
}
```

3. Define the first half of the `GetSteering` function, in which we compute the desired speed depending on the distance from the target according to the radii variables:

   ```
   Steering steering = new Steering();
   Vector3 direction = target.transform.position - transform.
   position;
   float distance = direction.magnitude;
   float targetSpeed;
   if (distance < targetRadius)
       return steering;
   if (distance > slowRadius)
       targetSpeed = agent.maxSpeed;
   else
       targetSpeed = agent.maxSpeed * distance / slowRadius;
   ```

4. Define the second half of the `GetSteering` function, in which we set the steering value and clamp it according to the maximum speed:

   ```
   Vector3 desiredVelocity = direction;
   desiredVelocity.Normalize();
   desiredVelocity *= targetSpeed;
   steering.linear = desiredVelocity - agent.velocity;
   steering.linear /= timeToTarget;
   if (steering.linear.magnitude > agent.maxAccel)
   {
       steering.linear.Normalize();
       steering.linear *= agent.maxAccel;
   }
   return steering;
   ```

5. To implement `Leave`, the name of the member variables changes:

   ```
   using UnityEngine;
   using System.Collections;

   public class Leave : AgentBehaviour
   {
       public float escapeRadius;
       public float dangerRadius;
       public float timeToTarget = 0.1f;
   }
   ```

6. Define the first half of the `GetSteering` function:

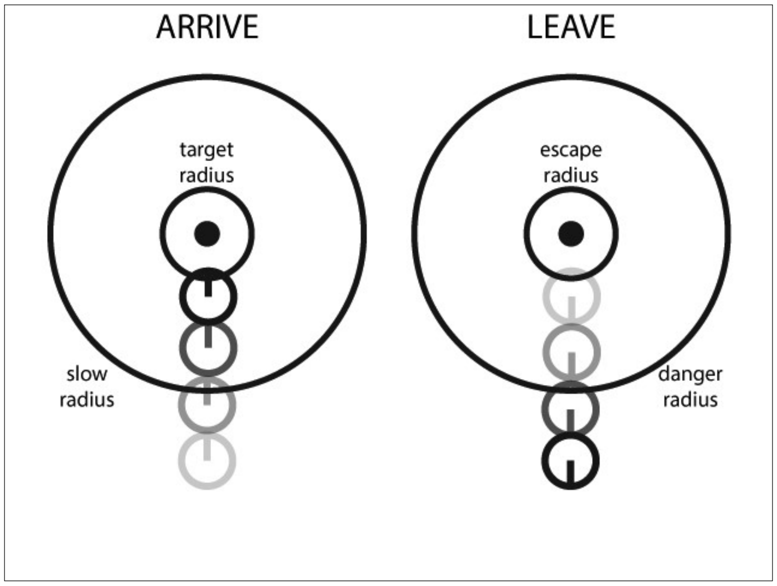
   ```
   Steering steering = new Steering();
   Vector3 direction = transform.position - target.transform.
   position;
   float distance = direction.magnitude;
   ```

```
    if (distance > dangerRadius)
        return steering;
    float reduce;
    if (distance < escapeRadius)
        reduce = 0f;
    else
        reduce = distance / dangerRadius * agent.maxSpeed;
    float targetSpeed = agent.maxSpeed - reduce;
```

7. And finally, the second half of `GetSteering` stays just the same.

How it works...

After calculating the direction to go in, the next calculations are based on two radii distances in order to know when to go full throttle, slow down, and stop; that's why we have several `if` statements. In the `Arrive` behavior, when the agent is too far, we aim to full-throttle, progressively slow down when inside the proper radius, and finally to stop when close enough to the target. The converse train of thought applies to `Leave`.



A visual reference for the Arrive and Leave behaviors