# 2.2操控行为——靠近、离开、抵达

本文解释封装各种操控行为的SteeringBehaviors类，然后分别介绍每个操控行为。

## SteeringBehavior类

SteeringBehavior类包含了不同操控行为的处理代码，主要的功能就是根据被设置的不同操控行为，在Calculate方法中调用这些处理代码，并将求得的操控力返回给智能体。代码如下：

namespace Silverlight3dApp

{

 public class SteeringBehavior

 {

 private enum behavior\_type

 {

 none = 0x00000,

 seek = 0x00002,

 flee = 0x00004,

 arrive = 0x00008,

 };

 // 检测指定的行为是否被设置

 private bool On(behavior\_type bt) { return (\_flags & bt) == bt; }

 // 包含当前激活行为的标志

 private behavior\_type \_flags;

 private Vehicle \_vehicle;

 private Vector2 \_steeringForce;

 public Vector2 Force { get { return \_steeringForce; } }

 public SteeringBehavior(Vehicle agent)

 {

 \_vehicle = agent;

 }

 public Vector2 Calculate()

 {

 // 将操控力重置为0

 \_steeringForce = Vector2.Zero;

 if (On(behavior\_type.seek ))

 {

 \_steeringForce = Seek(\_vehicle.World.PositionCrosshair);

 }

 if (On(behavior\_type.flee))

 {

 \_steeringForce = Flee(\_vehicle.World.PositionCrosshair);

 }

 if (On(behavior\_type.arrive))

 {

 \_steeringForce = Arrive(\_vehicle.World.PositionCrosshair, \_deceleration);

 }

 Global.Truncate(ref \_steeringForce,\_vehicle.MaxForce);

 return \_steeringForce;

 }

 // 开启或关闭操控行为的方法

 public void FleeOn() { \_flags |= behavior\_type.flee; }

 public void SeekOn() { \_flags |= behavior\_type.seek; }

 public void ArriveOn() { \_flags |= behavior\_type.arrive; }

 public void FleeOff() { if (On(behavior\_type.flee)) \_flags ^= behavior\_type.flee; }

 public void SeekOff() { if (On(behavior\_type.seek)) \_flags ^= behavior\_type.seek; }

 public void ArriveOff() { if (On(behavior\_type.arrive)) \_flags ^= behavior\_type.arrive; }

 public bool isFleeOn() { return On(behavior\_type.flee); }

 public bool isSeekOn() { return On(behavior\_type.seek); }

 public bool isArriveOn() { return On(behavior\_type.arrive); }

 //------------------------------- Seek -----------------------------------

 //

 // 给定一个目标，这个行为会返回一个使交通工具靠近目标的力

 //------------------------------------------------------------------------

 private Vector2 Seek(Vector2 TargetPos)

 {

 // …下面具体介绍

 }

 // 小于这个距离智能体不会出现Flee行为

 private float panicDistanceSq = 100.0f \* 100.0f;

 //----------------------------- Flee -------------------------------------

 //

 // 与Seek行为正相反

 //------------------------------------------------------------------------

 private Vector2 Flee(Vector2 TargetPos)

 {

 //…下面具体介绍

 }

 // Arrive行为通过这个值得到智能体到底目标位置所希望的时间

 public enum EnumDeceleration:int { slow = 3, normal = 2, fast = 1 };

 private EnumDeceleration \_deceleration = EnumDeceleration.normal;

 public EnumDeceleration Deceleration

 {

 get { return \_deceleration; }

 set { \_deceleration = value; }

 }

 // 用于调整减速的程度

 const float DecelerationTweaker = 0.3f;

 //--------------------------- Arrive -------------------------------------

 //

 // 这个行为与Seek相似，但能使智能体慢慢减速到目标位置

 //------------------------------------------------------------------------

 private Vector2 Arrive(Vector2 TargetPos,EnumDeceleration deceleration)

 {

 //…下面具体介绍

 }

 // 绘制辅助线段等信息

 public void RenderAids()

 {

 // 绘制表示操控力的红色线段

 if (\_vehicle.World.ShowSteeringForce)

 {

 Vector2 F = \_steeringForce;

 ShapeRender.AddLine(\_vehicle.Pos, \_vehicle.Pos + F,new Color (255,0,0));

 }

 }

 }

}

## Seek（靠近）

seek操控行为返回一个操控智能体到达目标位置的力。这很容易用程序实现。代码如下（注意，vehicle指向拥有SteeringBehaviors类的Vehicle类）：

//------------------------------- Seek -----------------------------------

//

// 给定一个目标，这个行为会返回一个使交通工具靠近目标的力

//------------------------------------------------------------------------

private Vector2 Seek(Vector2 TargetPos)

{

 Vector2 DesiredVelocity = Vector2.Normalize(TargetPos - \_vehicle.Pos) \* \_vehicle.MaxSpeed;

 return (DesiredVelocity - \_vehicle.Velocity);

}

首先计算预期的速度。这个速度是智能体在理想化情况下到达目标位置所需的速度。它是从智能体指向目标的向量，大小为智能体的最大速度。

该方法返回的操控力是所需要的力，当把它加到智能体当前速度向量上就得到预期的速度。所以，你可以简单地从预期速度中减去智能体的当前速度，见下图。



计算seek行为的向量。虚线向量加到当前速度后就可以产生预期结果。

通过运行可执行文件，你可以看到这个行为。单击鼠标左键，改变目标的位置。从中可以看到，智能体穿过目标，然后转向并再次靠近。穿过的次数取决于MaxSpeed和MaxForce的比率。你可以通过滑动条改变这两个值的大小。

seek行为在各种情况中都能派上用场，许多其他操控行为都会使用到它。

## Flee（离开）

flee和seek相反。flee产生一个操控智能体离开的力，而不是产生靠近智能体的力。代码如下：

private Vector2 Flee(Vector2 TargetPos)

{

 // 如果目标在“恐慌距离”之内才会发生Flee行为

 if (Vector2.DistanceSquared(\_vehicle.Pos, TargetPos) > panicDistanceSq)

 {

 return Vector2.Zero ;

 }

 Vector2 DesiredVelocity = Vector2.Normalize(\_vehicle.Pos - TargetPos) \*\_vehicle.MaxSpeed;

 return (DesiredVelocity - \_vehicle.Velocity);

}

由此可见，它们唯一的区别是DesiredVelocity是用指向相反方向的向量计算的（是\_vehicle.Pos - TargetPos，而不是TargetPos -\_vehicle.Pos）。

通过额外的几行代码，我们可以调整一下flee，使其只有当交通工具进入目标的一定范围内才产生离开力。从上面代码可知，到目标的距离是以平方计算的，这是为了省去计算平方根的开销。

## Arrive（抵达）

seek行为对于让1个智能体向正确方向移动很有用，但是很多情况是希望智能体能缓慢地停在目标位置。如你所看到的，seek行为不能很好地慢慢停下来。Arrive行为是一种操控智能体慢慢减速直至停在目标位置的行为。

这个函数接受两个参数：目标位置，一个枚举类型Deceleration的值。

enum EnumDeceleration:int { slow = 3, normal = 2, fast = 1 };

arrive通过这个值得到智能体到达目标位置所希望的时间。然后我们可以计算出在给定的时间内必须以多大的速度运行才能到达目标位置。接下来，计算过程就像seek那样了。

// 用于调整减速的程度

const float DecelerationTweaker = 0.3f;

//--------------------------- Arrive -------------------------------------

//

// 这个行为与Seek相似，但能使智能体慢慢减速到目标位置

//------------------------------------------------------------------------

private Vector2 Arrive(Vector2 TargetPos,EnumDeceleration deceleration)

{

 Vector2 ToTarget = TargetPos - \_vehicle.Pos;

 // 计算到目标位置的距离

 float dist = ToTarget.Length();

 if (dist > 0)

 {

 // 给定减速程度，计算能到达目标位置所需的速度

 float speed = dist / ((int)deceleration \* DecelerationTweaker);

 // 确保这个速度不超过最大值

 speed = Math.Min (speed, \_vehicle.MaxSpeed);

 // 下面的处理和Seek一样，除了不需要归一化ToTarget向量，

 // 因为我们已经计算了它的长度: dist

 Vector2 DesiredVelocity = ToTarget \* speed / dist;

 return (DesiredVelocity - \_vehicle.Velocity);

 }

 return Vector2.Zero ;

}

在了解arrive之后，让我们看一下示例程序。从中可以发现，当交通工具离目标很远时，arrive行为就和seek一样，只有当交通工具接近目标时，才有减速的效果。