# 2.4 操控行为——追逐、逃避

## Pursuit（追逐）

当智能体要拦截一个可移动的目标时，pursuit行为就变得很有用。当然它能向目标的当前位置靠近，但是这不能制造出智能的假象。想象你还是个小孩，在操场上玩“单脚抓人”游戏。当想抓住某人时，你不会直接移向他们的当前位置（有效地靠近他们）。你预测他们的未来位置，然后移向那个偏移位置，其间通过不断调整来缩短差距。下图演示了这一行为。



**计算追逐操控行为的向量，虚线向量加到当前速度后就可以产生预期结果**

pursuit函数的成功与否取决于追逐者预测逃避者的运动轨迹有多准。这可以变得很复杂，所以做个折衷以得到足够的效率但又不会消耗太多的时钟周期。

追逐者可能会碰到一种提前结束（enables an early out）的情况：如果逃避者在前而，几乎面对智能体，那么智能体应该直接向逃避者当前位置移动。这可以通过点积快速算出。在示例代码中，逃避者朝向的反方向和智能体的朝向必须在20°内（近似）才被认为是面对着的。

一个好的预测的难点之一就是决定预测多远。很明显，这个“多远”应该正比于追逐者与逃避者的距离，反比于追逐者和逃避者的速度。一旦这“多远”确定了，我们就可以估算出追逐者seek的位置。让我们看一下这个行为的代码：

//------------------------------ Pursuit ---------------------------------

//

// 这个行为产生一个力，操控智能体朝向逃避者

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

private Vector2 Pursuit(Vehicle evader)

{

 // 如果逃避者位于追逐者之前且正对（本例中正对的意思是朝向的夹角不超过约18度），

 // 那么只需Seek逃避者的当前位置

 Vector2 ToEvader = evader.Pos - \_vehicle.Pos;

 float RelativeHeading = Vector2.Dot (\_vehicle.Heading,evader.Heading);

 if ( (Vector2.Dot (ToEvader,\_vehicle.Heading) > 0) && (RelativeHeading < -0.95)) //acos(0.95)=18 degs

 {

 return Seek(evader.Pos);

 }

 // 如果不是正对则需要预测逃避者的位置。

 // 预测的时间正比于两者的距离，反比于两者的速度之和

 float LookAheadTime = ToEvader.Length() / (\_vehicle.MaxSpeed + evader.Speed);

 // 十字光标表示预测位置

 \_vehicle.World.PositionCrosshair = evader.Pos + evader.Velocity \* LookAheadTime;

 // Seek逃避者的预测位置

 return Seek(evader.Pos + evader.Velocity \* LookAheadTime);

}

在pursuit行为的示例程序中，一个小的交通工具正在被一个大的追逐。十字准线位置正是我们预测的逃避者的下一个位置。（逃避者用wander操控行为实现运动）。

我们通过把目标传给相关方法来为追逐者设置猎物。示例程序中有两个智能体，一个追逐，一个徘徊，就像这样：

prey.Steering.WanderOn();

predator.Steering.PursuitOn(prey);

下面让我们讨论pursuit的相反行为——evade（逃避）。

## Evade（逃避）

除了逃避者远离预测的位置这一点，evade几乎和pursuit一样。

//----------------------------- Evade ------------------------------------

//

// 与pursuit类似，唯一相反的是智能体是在逃避追逐者的预测位置

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

private Vector2 Evade(Vehicle pursuer)

{

 Vector2 ToPursuer = pursuer.Pos - \_vehicle.Pos;

 // 只有在在ThreatRange范围内才会产生Evade行为

 float ThreatRange = 100.0f;

 if (ToPursuer.LengthSquared() > ThreatRange \* ThreatRange)

 return Vector2.Zero ;

 // 预测的时间正比于两者的距离，反比于两者的速度之和

 float LookAheadTime = ToPursuer.Length() / (\_vehicle.MaxSpeed + pursuer.Speed);

 // 十字光标表示预测位置

 \_vehicle.World.PositionCrosshair = pursuer.Pos + pursuer.Velocity \* LookAheadTime;

 // Flee追逐者的预测位置

 return Flee(pursuer.Pos + pursuer.Velocity \* LookAheadTime);

}

注意，这次没有必要检查面向方向。