Exercise – Steering Behaviours – Part 2

Exercise 1:

Based on the lecture notes, Implement a Pursue behaviour that implements the IBehaviour class/interface.

```
dir = (agent.target.position + agent.target.velocity) – agent.position
forceToApply = normalise(dir) * agent.maxVelocity – agent.velocity
//etc...
```

Exercise 2:

Based on the lecture notes, Implement an Evade behaviour that implements the IBehaviour class/interface.

```
//can you figure out how to get the inverse direction from the Persue example?
dir = ?
forceToApply = normalise(dir) * agent.maxVelocity – agent.velocity
//etc...
```

Exercise 3:

Based on the lecture notes, Implement an Arrival behaviour that implements the IBehaviour class/interface. There are a couple ways to approach this:

1) This can be done by directly manipulating the agent's velocity, though this may make it difficult later on to combine the forces of multiple steering behaviours.

```
//seek force calculation
force = normalise(agent.target.position – agent.position) * agent.maxVelocity
force -= agent.velocity

//arrival multiplier calculation
distance = magnitude(agent.position – agent.target.position)
mul = min(distance / agent.arriveRadius, 1)

agent.velocity += force * mul
```

2) Alternatively we can use a more complicated technique for an arrive behaviour that determines a force to be applied to the agent, just like most other steering behaviours.

```
//direction and distance to target
direction = normalised(agent.target.position - agent.position);
distance = magnitude(agent.position - agent.target.position);

//calculate seek force
force = normalised(agent.target.position - agent.position) * agent.maxVelocity;
```

```
//scalar representing distance from target (1 on radius edge, 0 at target, else > 1)
scalar = min(distance / arriveRadius, 1);

//if within radius
if (scalar < 1)
{
    //scale seek force down according to distance from target (as 0-1 scalar)
    force *= scalar;

    //(-1 - 1) scalar for how close our heading is with the direction to target
    d = dot(direction, normalised(agent.velocity));
    //force for pushing agent away from target to slow down (full force if
    //heading directly towards target, less if not, inverse if heading away)
    //force is based on current speed
    resistance = -(normalised(agent.velocity)) * magnitude(agent.velocity) * d * 2;

    //add resistance force
    force += resistance;
}

//sub own velocity to get change in heading
forceToApply = force – agent.velocity;
//etc...
```

Exercise 4:

Alter the IBehavior object to take in a weight value when created and have it return forces that are scaled by this weight value.

- Weight 1.0 = full force
- Weight 0.5 = half force
- Weight 0.25 = quarter force

Now give your agent the ability to store a collection of IBehaviors and have each influence the agent's movement by collecting the forces from all behaviours and applying them to the agent's velocity.

```
force = zero
foreach(IBehaviour b in behaviors)
    force += b.Update()
velocity += force
//etc...
```

Alternatively, you can have the IBehaviour take in a weight value as an argument for the Update function, though this may slightly complicate the code for assigning weighting values.

Challenge Exercise:

Implement an Avoid behaviour that works with circles. Try to write this as a force that can be combined with the forces from other steering behaviours, rather than directly manipulating and agent's velocity or position.

- See this tutorial for help getting started.

References:

- A great tutorial for revision on the Arrival behaviour can be found here.
- A great tutorial for revision on the Pursue and Evade behaviours can be found here.
- A great tutorial for revision on combining steering behaviours can be found here.
- A great tutorial for revision on the Avoidance behaviour can be found here.