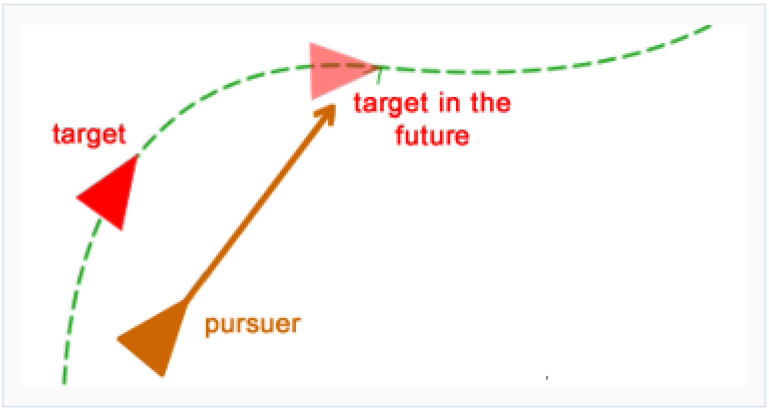
Pursue

A pursuit is the process of following a target aiming to *catch it*. It's important to note that the word "catch" makes all the difference here. If something is just following a target, all it has to do is repeat the target's movements and, as a consequence, it will be on its track.

When pursuing something, the pursuer must follow the target, but it also has to anticipate where the target will be in the near future. If you can predict (or estimate) where the target will be within the next few seconds, it's possible to adjust the current trajectory to avoid unnecessary routes:



Predicting the Future

As described in the first tutorials the movement is calculated using Euler integration:

```
position = position + velocity
```

As a direct consequence, if the current character's position and velocity are known it is possible to predict where it will be within the next `T` game updates. Assuming the character is moving in a straight line and the position we want to predict is three updates ahead (`T=3`), the character's future position will be:

```
position = position + velocity * T
```

The key for a good prediction is the right value for `T`. If the value is too high, it means the pursuer will end up chasing a ghost. If `T` is too close to zero, it means the pursuer is not actually pursuing the target, but it is just following it (no prediction).

If the prediction is calculated for every game frame, it works even if the target is constantly changing its direction. Every update a new "future position" is generated based on the character's current velocity vector (which also guides the movement direction).

Pursuing

The pursuit behavior works pretty much the same way seek does, the only difference is that the pursuer will not seek the target itself, but its position in the near future.

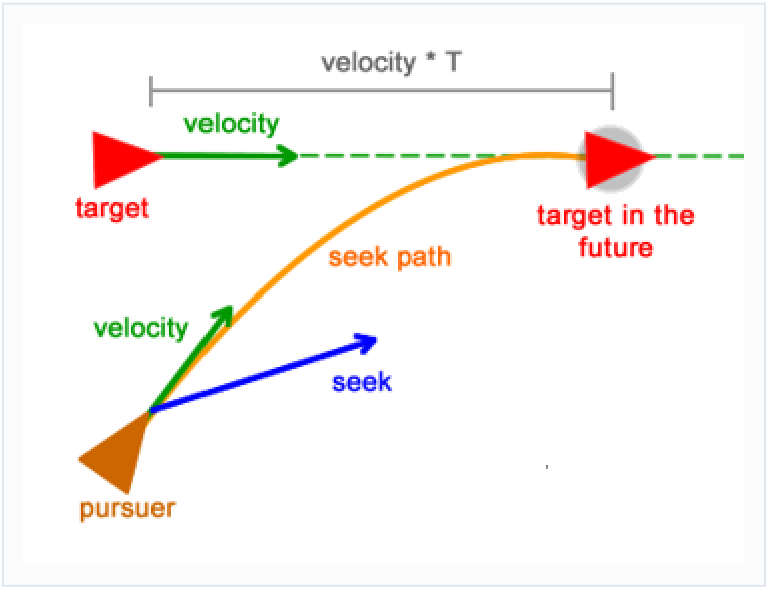
Assuming every character in the game is represented by a class named `Boid`, the following pseudo-code implements the basic idea behind the pursuit behavior:

```
public function pursuit(t :Boid) :Vector3D {
  T :int  = 3;
  futurePosition :Vector3D = t.position + t.velocity * T;
  return seek(futurePosition);
}
```

After the pursuit force calculation, it must be added to the velocity vector just like all previous steering forces:

```
public function update() :void {
  steering = pursuit(target)
  steering = truncate (steering, max_force)
  steering = steering / mass
  velocity = truncate (velocity + steering , max_speed)
  position = position + velocity
}
```

The figure below illustrates the process:



Improving Pursuit Accuracy

There is a problem when the value of `T` is constant: the pursuit accuracy tends to be poor when the target is close. This is because when the target is close, the pursuer will continue to seek the prediction of the target's position, which is `T` frames "away". That behavior would never happen in a real pursuit because the pursuer would know it is close enough to the target and should stop predicting its position.

There is a simple trick that can be implemented to avoid that and drastically improve the pursuit accuracy. The idea is very simple; instead of a constant value for `T`, a dynamic one is used:

```
T = distanceBetweenTargetAndPursuer / MAX_VELOCITY
```

The new `T` is calculated based on the distance between the two characters and the maximum velocity the target can achieve. In simple words the new `T` means *"how many updates the target needs to move from its current position to the pursuer position"*.

The longer the distance, the higher `T` will be, so the pursuer will seek a point far ahead the target. The shorter the distance, the lower `T` will be, meaning it will seek a point very close to the target. The new code for that is:

```
public function pursuit(t :Boid) :Vector3D {
  var distance :Vector3D = t.position - position;
  var T :int = distance.length / MAX_VELOCITY;
  futurePosition :Vector3D = t.position + t.velocity * T;
  return seek(futurePosition); }
```