Université Concordia University

COMP 476

Advanced Game Development

Session 2

Movement AI – Delegated Behaviours

Based on AI for G, Millington Chapters 3 (Adapted from Dr. Feven's Slides)

Lecture Overview

- Delegated Behaviors
- Separation
- Collision Avoidance
- Combining Steering Behaviors
- Jumping
- Coordinated Movement

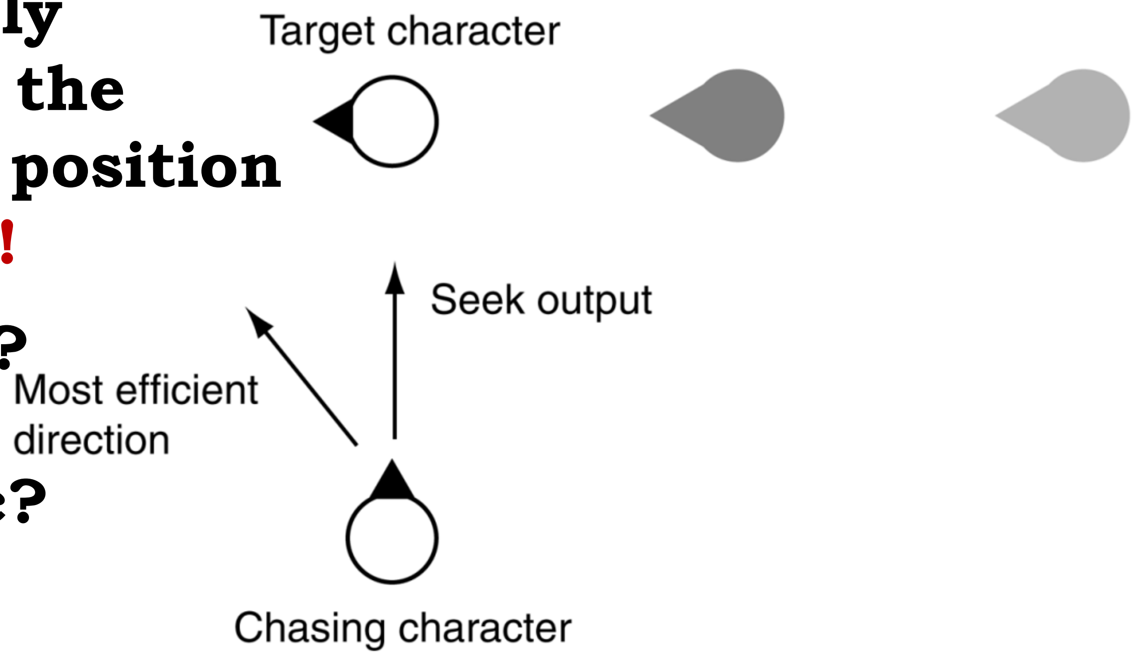
Université Concordia University

Delegated Behaviors

- More complex behaviours that make use of the basic fundamental steering behaviours
- Basic idea is to: ***calculate a target*** (typically position or orientation), and then **delegate to one of the fundamental behaviours** to calculate the steering.
- Turns out that **Seek**, **Align** and **Velocity Matching** are the only fundamental behaviours that need be used
- Many delegated behaviours can then in turn be used as the basis of another delegated behaviour

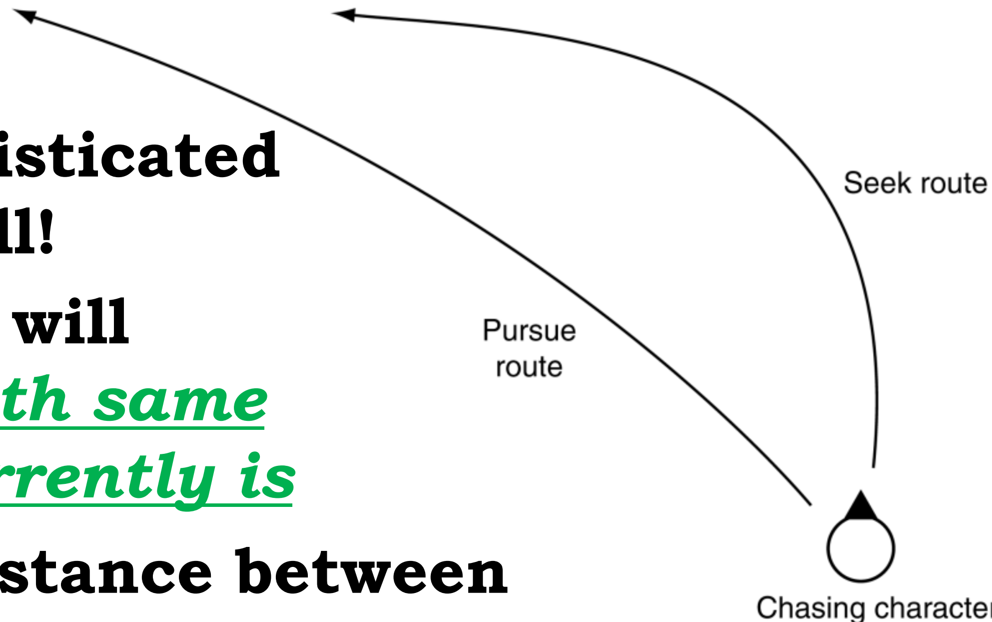
Pursue

- **When seeking a moving target, constantly moving towards the target's current position is not sufficient!**



- **Going in circles? Inefficient? Look unrealistic?**
- **Instead of aiming at its current position, *how about predicting where it will be at some time in the future, and aim towards that point*?**

Pursue

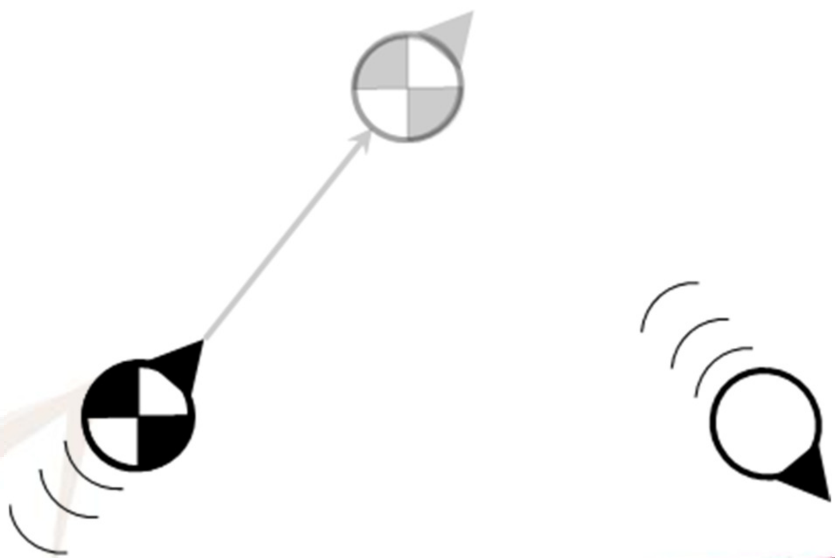


- Does not need sophisticated algorithms – Overkill!
- **Assumption**: Target will *continue moving with same velocity (v_t) as it currently is*
- Work out current distance between character and target ($d=|p_t - p_c|$), and how long it takes to get there at max velocity (d/v_m)
- *Use this time interval as prediction time*
- Calculates future position of target based on the assumption ($p_t + v_t(d/v_m)$)
- *Use new position as target for Seek*

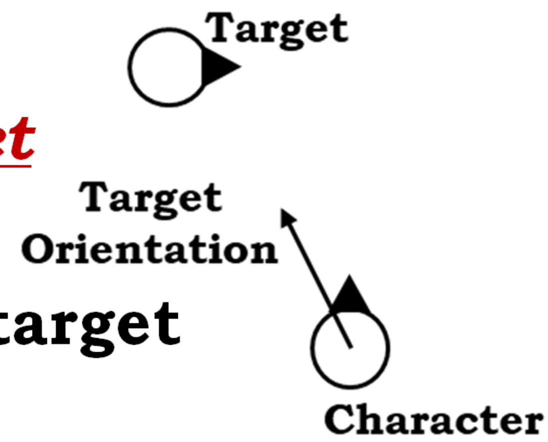
Université Concordia University

Evade

- Simply the ***opposite behaviour to Pursue***
- Instead of delegating to Seek, ***delegate the predicted future position of the target to Flee***
- E.g., evade the future predicted position (in gray) of the target.



Face



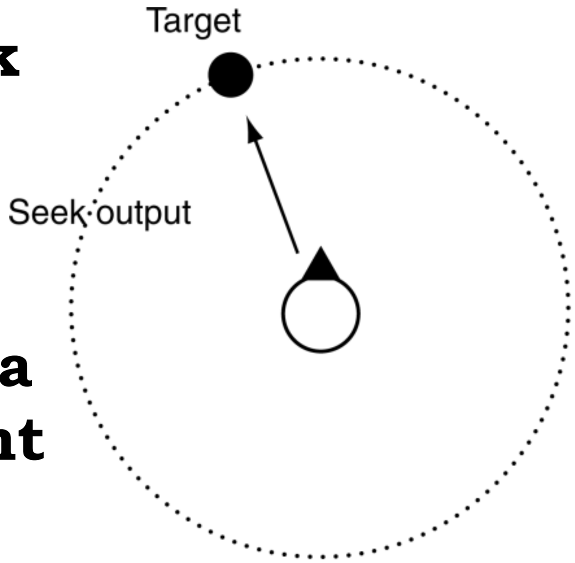
- Makes character ***look at its target***
- ***Delegates to Align behaviour to perform rotation***, but calculates target orientation first
- Target orientation generated from relative position of target to character
  - **Terminology trap**: the target orientation is desired orientation of the character (so it can "face" its target), not the orientation of the target (e.g., ***we don't care what the orientation of a target itself is***)

Looking Where You're Going

- To enable character to ***face in the direction it is moving***
- Using ***Align***, give the character angular acceleration to make it face the right way while moving – this method causes gradual facing change (more natural)
- Method of implementation is similar to Face behaviour except for the target orientation which in this case is calculated using (***the direction of***) ***the current velocity*** of character

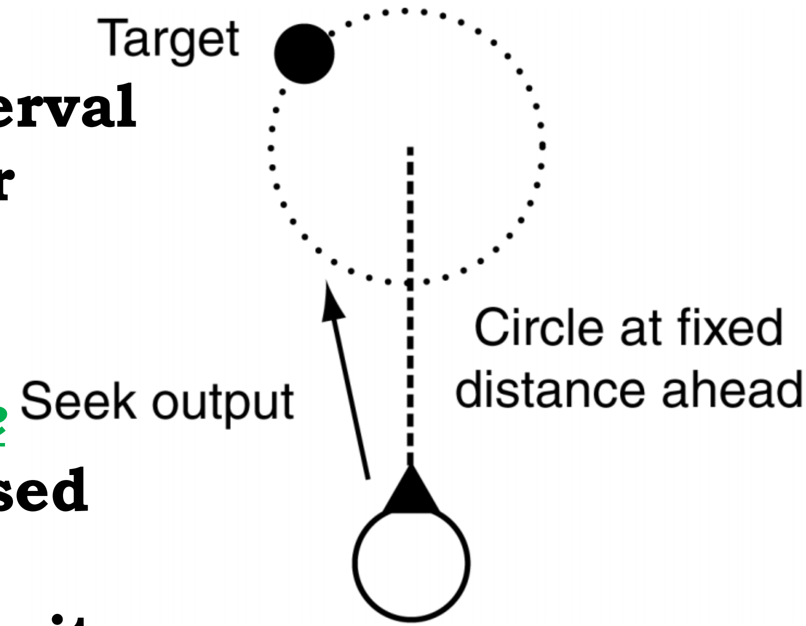
Wander

- In *Kinematic* version, direction of character is perturbed by a random amount of rotation each time it was run. Result = **Erratic (twitchy) rotation**
- This can be smoothed by making orientation of the character indirectly reliant on random numbers.
- OR, think of it as a delegated Seek behaviour
- **Idea #1**: *Constrain the target to a circle around the character* (the target is moved around the circle a little, by a uniform random amount within an limited arc)



Wander

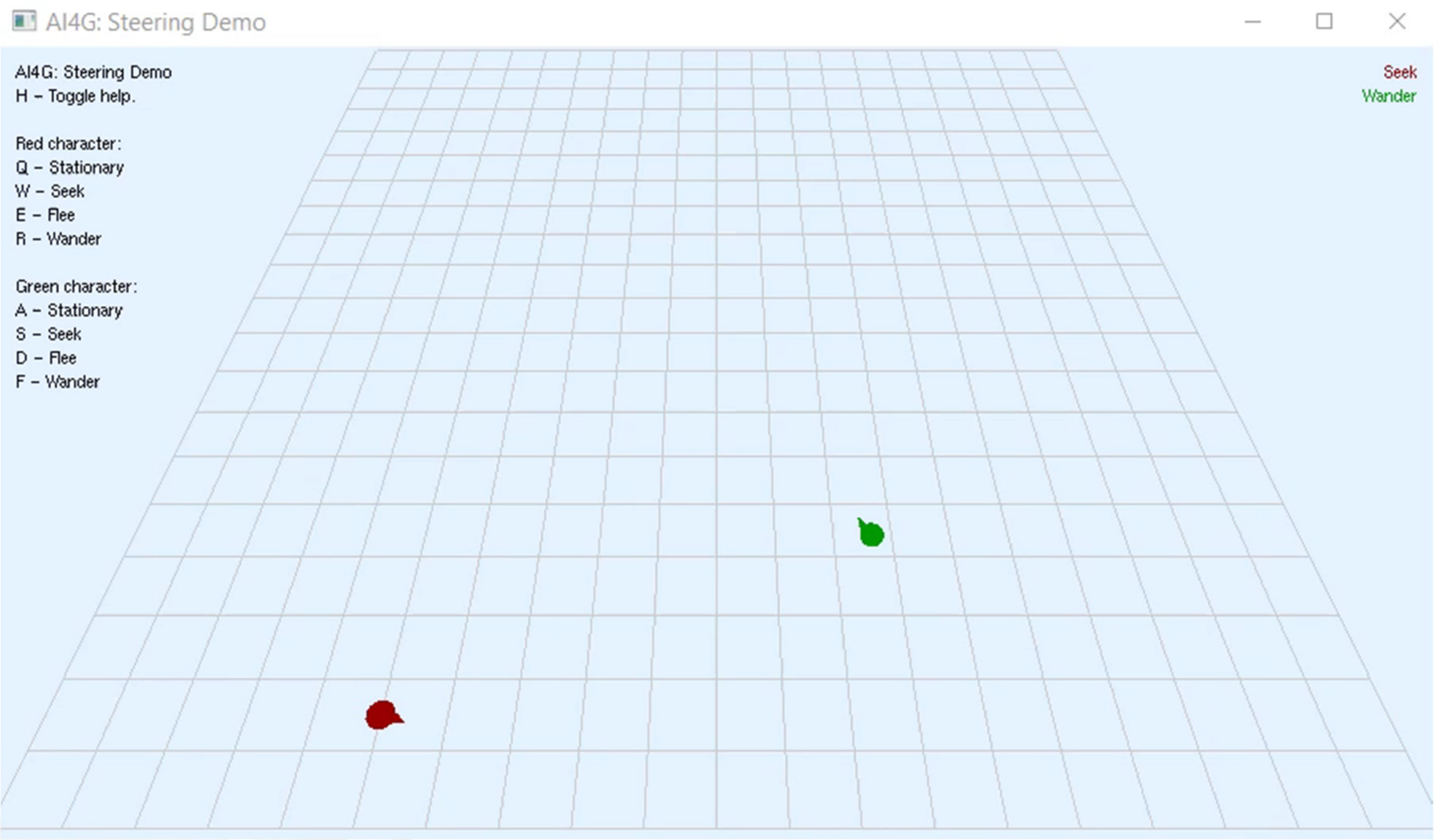
- Idea #2: Improve it by ***moving the circle out in front of the character and shrink it down***
- **A maximum "wander rate"** can be used to constrain the random numbers to an interval within the previous wander direction – ***to prevent too much erratic rotation***



- ***Face or Look Where You're Going*** behaviours can be used to align the character's orientation to the direction it is moving in

DEMO

Wander



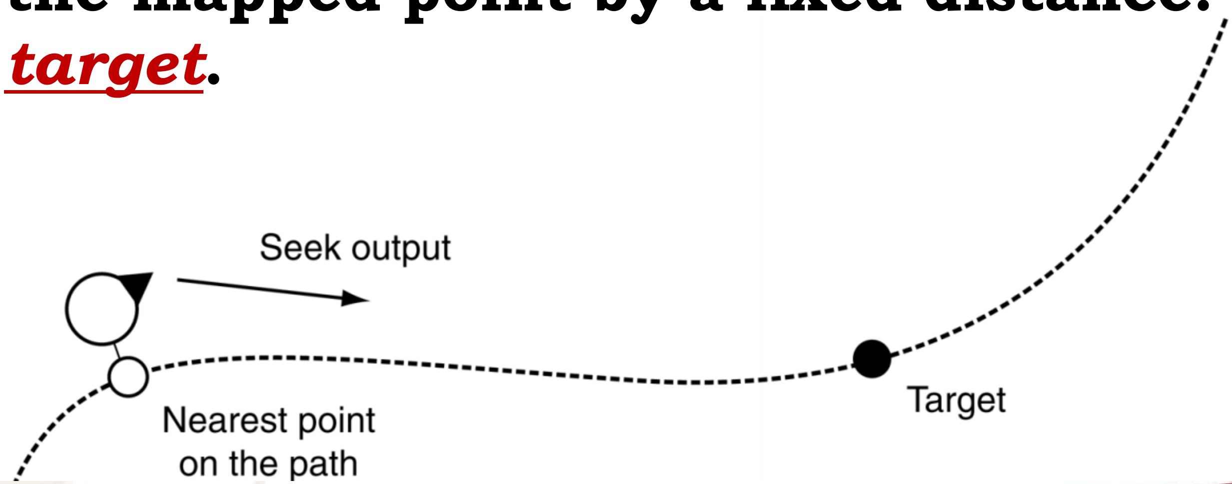
https://github.com/idmillington/aicore

Path Following

- Steering behaviour that takes a whole path as a target
- Move along path ***in one direction***
- Since the target position is always moving forward along the path, we shouldn't need to worry about catching up to it (**so no need to delegate to Arrive**)
- A Delegate behaviour:
  - Calculates ***position of target based on current character location and shape of path***
  - Hands over its target to Seek

Path Following

- 2 stages:
  - Current character position is mapped to nearest point along path. Curved paths or paths with many line segments can increase computation complexity.
  - Target is selected further along the path than the mapped point by a fixed distance. ***Seek the target***.



Université Concordia University

Path Following

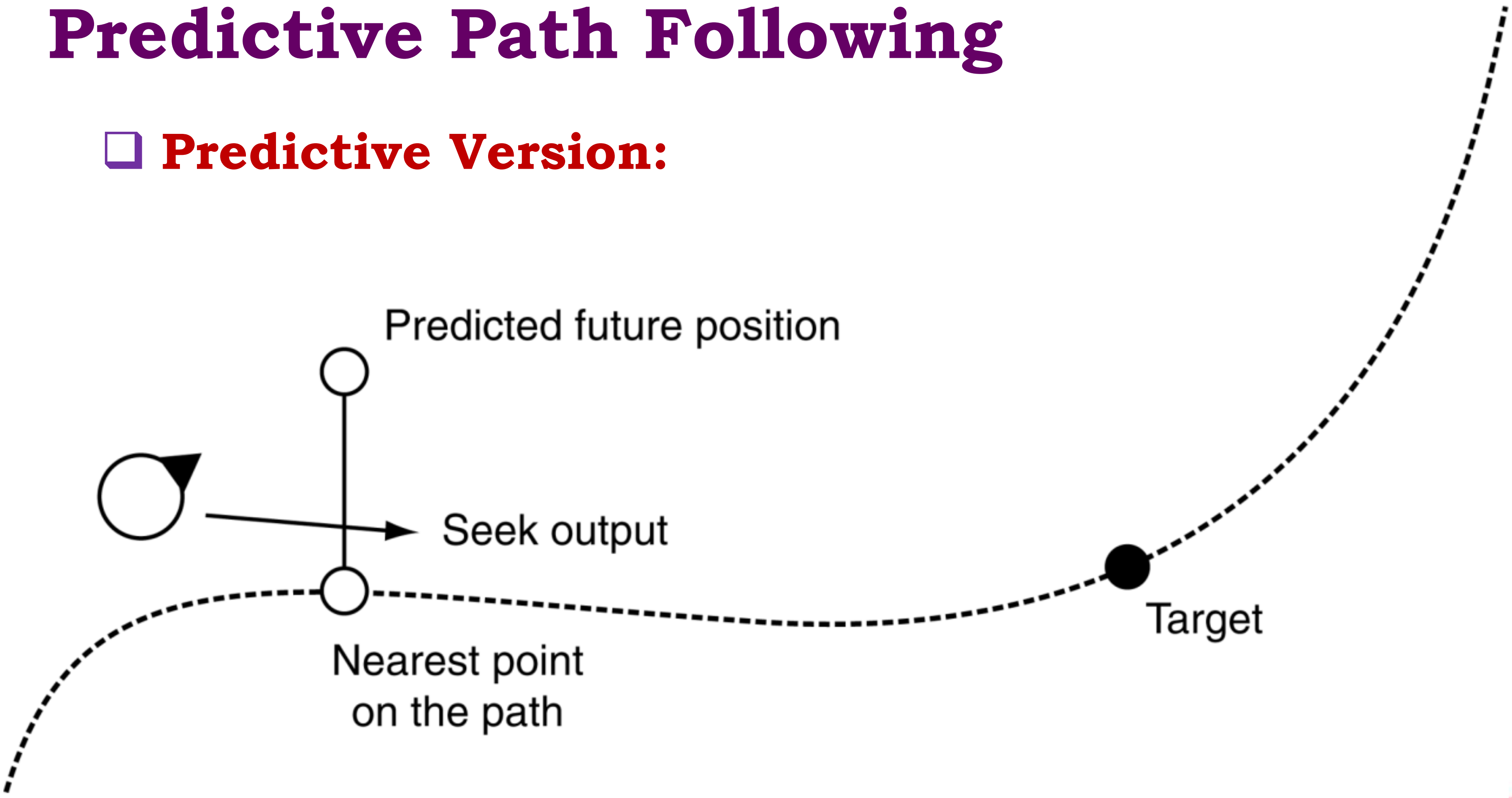
Predictive Path Following

- **Predictive Version**:
  - ***Predict where the character will be in a short time***
  - **Map this predicted point to the nearest point on the path. Target is selected further along the path than the mapped point by a fixed distance. This is *the candidate target for Seek*.**
  - **If the new candidate target has not been placed farther along the path than it was at the last frame, then the target is changed so that it is further along the path.**

Path Following

Predictive Path Following

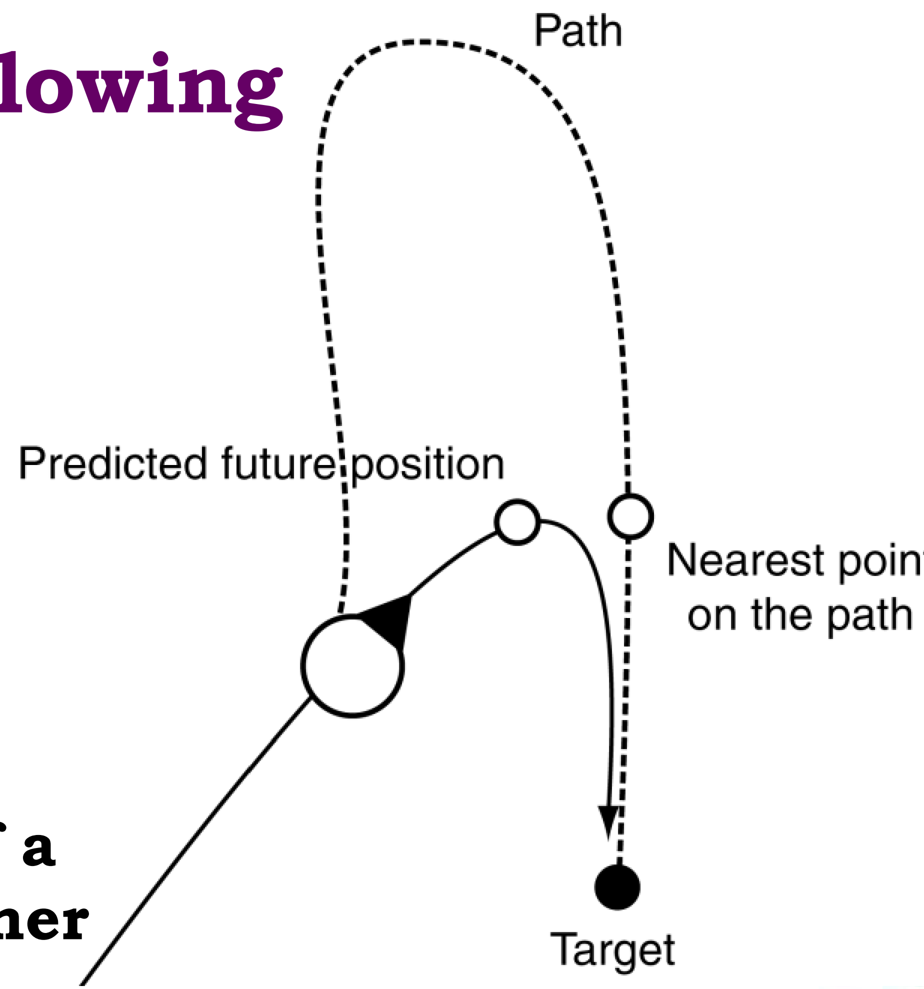
- **Predictive Version:**



Path Following

Predictive Path Following

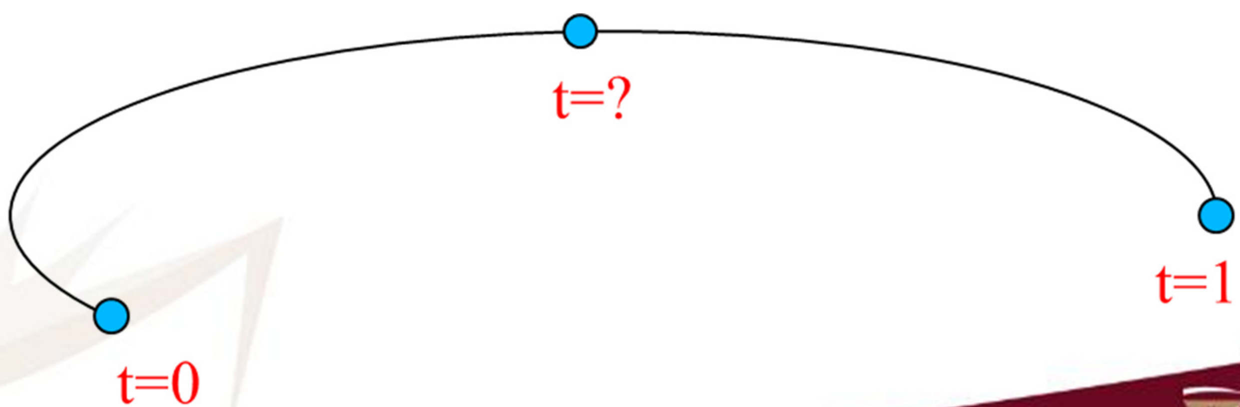
- **Upside**: ***Smoother for complex paths*** **with sudden direction changes**
- **Downside**: **Cutting-corner behaviour – Character *may miss a whole section of the path* if two sections of a path come close together**



Path Following

How to Construct Path?

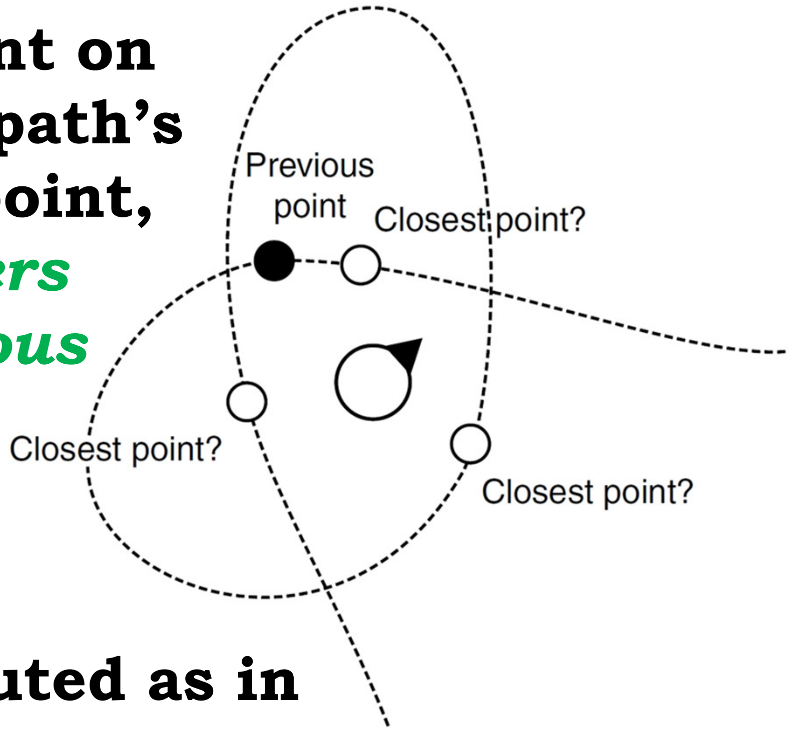
- For ease of use in graphics/rendering systems, paths are ***normally represented using a single parameter*** (normally floating-point, constrained to a range) that increases monotonically along the path (can be seen as distance along path). Known as **a parametric curve** (e.g., Bézier curves)



Path Following

Keeping Track of the Parameter

- When mapping the character's position to the nearest point on the path to determine the path's parameter at the mapped point, we *only consider parameters that are close to the previous parameter*.
- This technique is called *coherence*.
- Then even paths as convoluted as in the example to the right are easily handled.



Separation

- Commonly used for crowd simulations (where number of characters are ***heading roughly in the same direction***)
- Acts to keep characters from getting too close and crowded.
- Does not work ***when characters move across each others' path*** (collision avoidance behaviour should be used instead; we'll see this next)
- Most of the time, zero output in terms of movement!

Separation

- **Idea**: If behaviour detects another character closer than some threshold (a fixed distance away), it acts like "evade" to move away
- **Strength** of the "evade" movement ***is related to the distance from the target***
- **Acts the same way as a physical repulsive force**
- **If there are multiple characters within the threshold, steering is calculated for each in turn and then summed (such that result ≤ a_m).**
- **If the number of characters is huge, a spatial data structure (e.g., a BSP tree) can be used to find those characters that are closest**

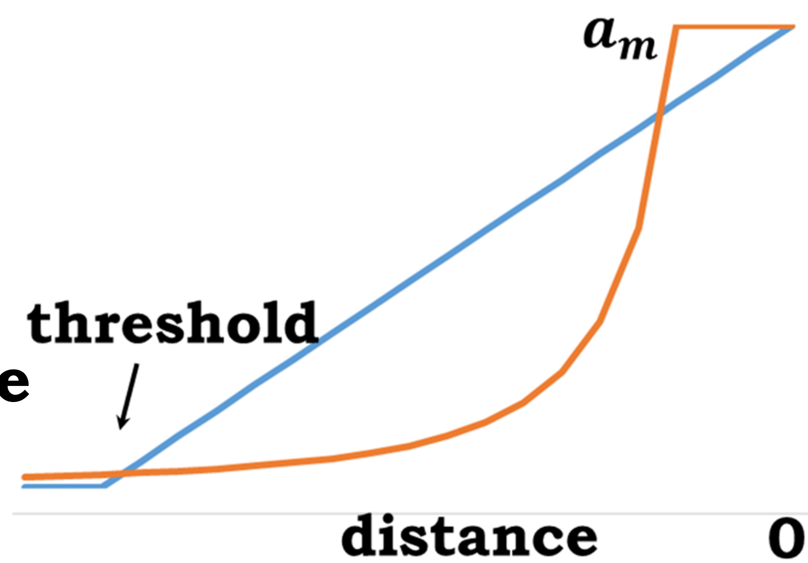
Separation

- Two Common Calculations

  strength = a_m * (threshold – distance)/threshold

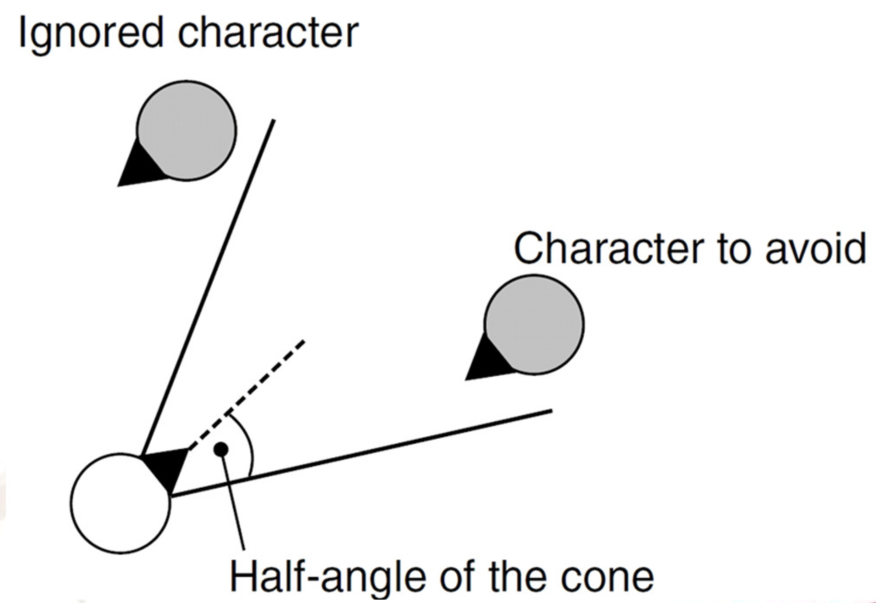
  strength = min(k / (distance * distance), a_m)

- For each case,
  - distance: distance between character and nearby neighbor
  - threshold: min distance for separation to occur
  - a_m: max acceleration
  - k: strength decay constant



Collision Avoidance

- **The goal is *to avoid collision between various moving characters* in the same space.**
- **A simple approach is to use *a variation of Evade or Separation behaviour*, which only engages if a target is within a cone in front of the character**



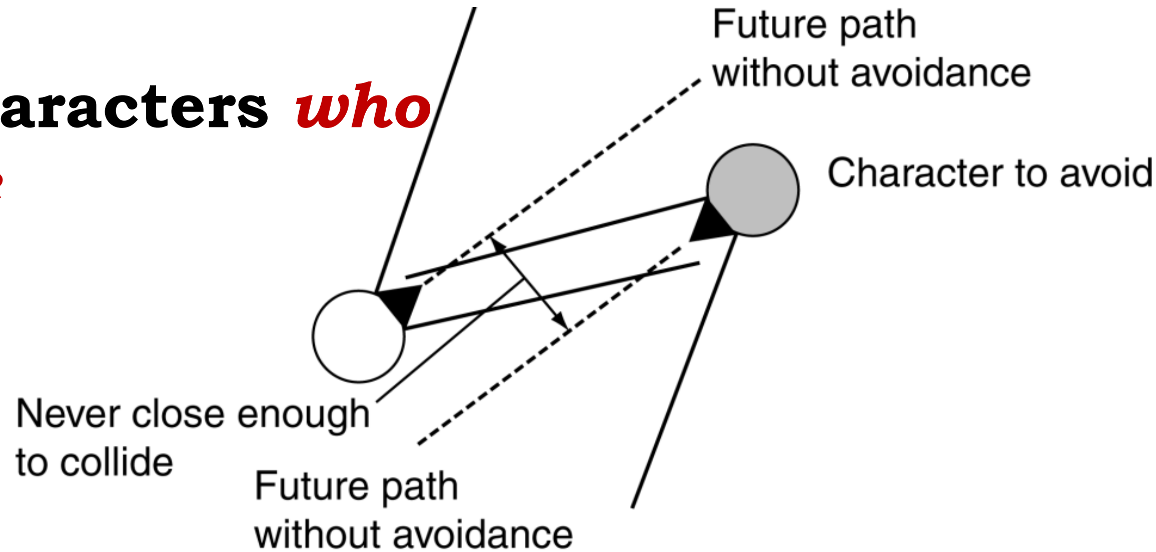
Collision Avoidance

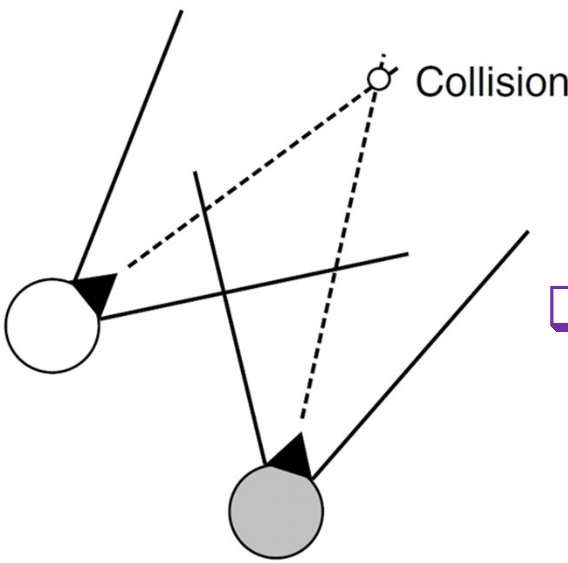
- If there are several characters in the cone, then the ***behaviour needs to avoid them all***.
- It is often sufficient to find the **average position and speed of all characters in the cone and evade that target**.
- Or, the ***closest character*** in the cone can be found and the rest ignored.
- Unfortunately, this latter approach, while simple to implement, ***doesn't work well with more than a handful of characters***

Collision Avoidance

Closest Character - Problems

- Two in-cone characters ***who will not collide***

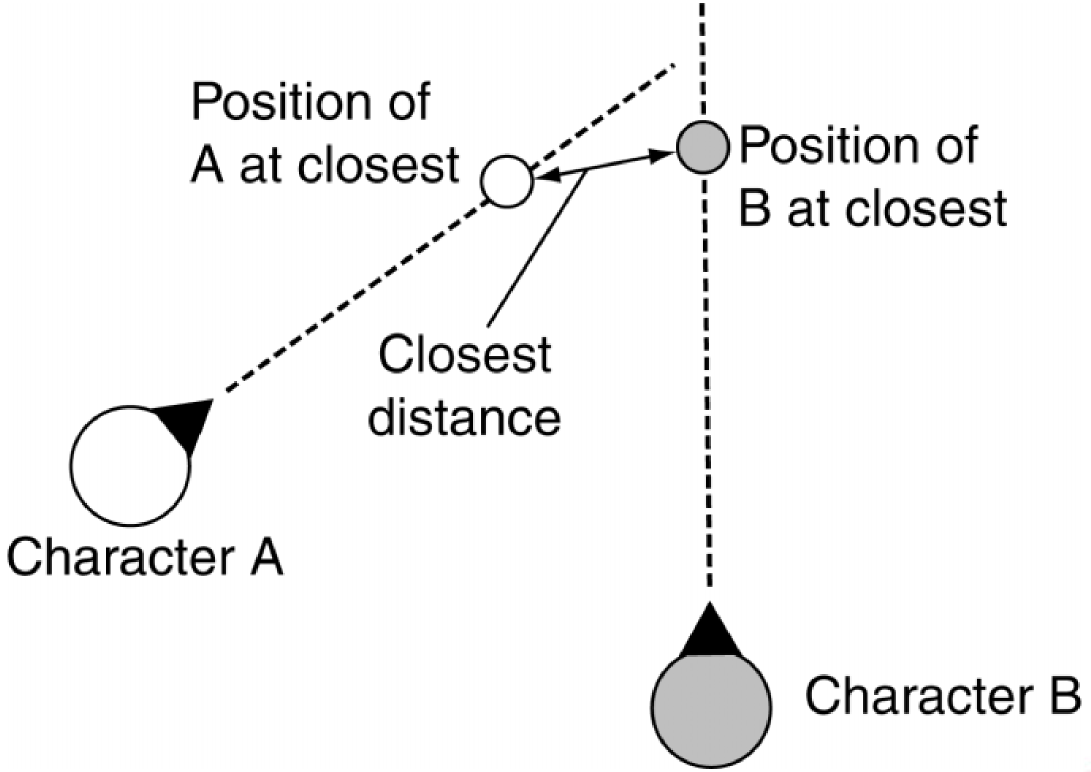




- Two out-of-cone characters ***who will collide***

Collision Avoidance

- **A better (cone-free) solution works out whether or not the characters will collide *<u>if they keep to their current velocity</u>*.**



Collision Avoidance

- Note that the closest approach will not normally be the same as the point where the future trajectories cross (***their velocities may differ***).
- Instead, we have to ***find the moment that they are at their closest***, use this to derive their separation, and check if they collide.
- The **time of closest approach** is given by

$$t_{\text{closest}} = \left(\frac{d_p}{|d_v|}\right) \cdot \left(-\frac{d_v}{|d_v|}\right) = -\frac{d_p \cdot d_v}{|d_v|^2}$$

where d_p is the current relative position of target to character, $d_p = pt - pc$, and d_v is the relative velocity: $d_v = vt - vc$.

Collision Avoidance

Time of Closest Approach

- If the time of closest approach is negative, then the character is *already moving away from the target*, and no action needs to be taken.
- The position of character and target at the time of closest approach can be calculated:

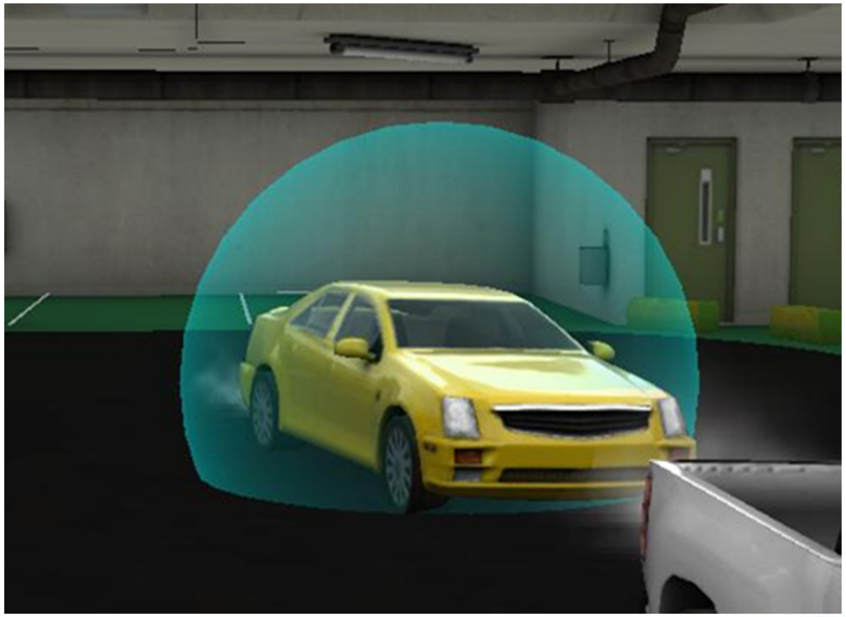
$$p'_c = p_c + v_c t_{\text{closest}}$$
$$p'_t = p_t + v_t t_{\text{closest}}$$

- We then use these positions as the basis of an *Evade behaviour*.
- If the character is going to hit the target with smallest $t_{closest}$ exactly, or is already colliding, use the Evade behaviour with current positions

Collision Avoidance

Obstacle/Wall Avoidance

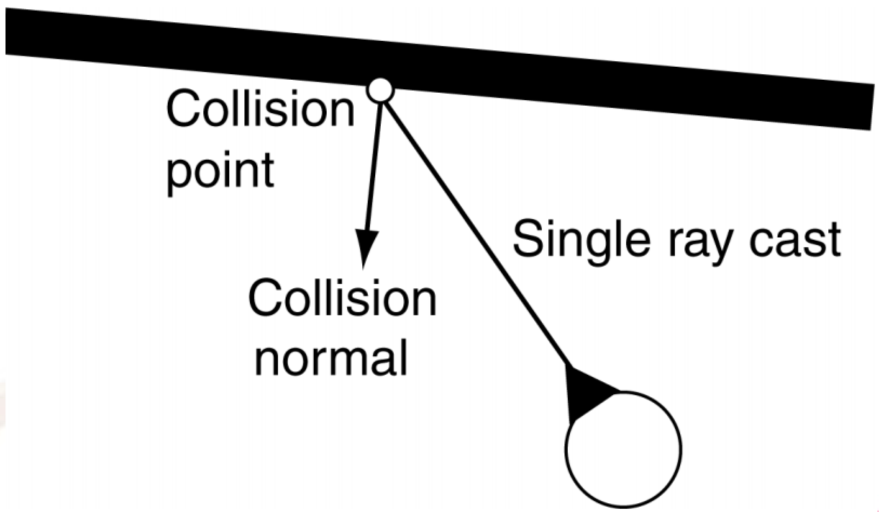
- **If The goal is to avoid collision between character and *unanimated obstacles or walls***
- **The collision avoidance in games frequently *assume that targets are spherical*.**
- **But the most common obstacles in the game, walls, *cannot be simply represented by bounding spheres at all***



Collision Avoidance

Obstacle/Wall Avoidance

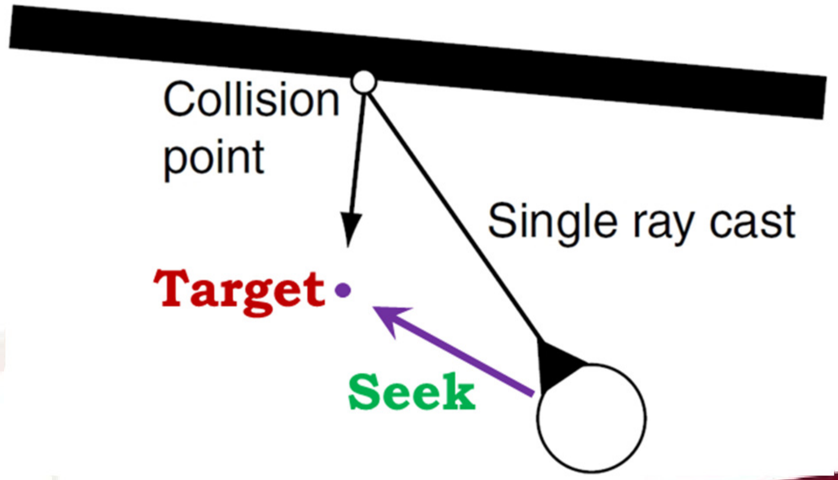
- The obstacle and wall avoidance behaviour *uses a different approach* to avoiding collisions. The moving character *casts one or more rays* out ***a short distance ahead of the character in the direction of its motion***.



Collision Avoidance

Obstacle/Wall Avoidance

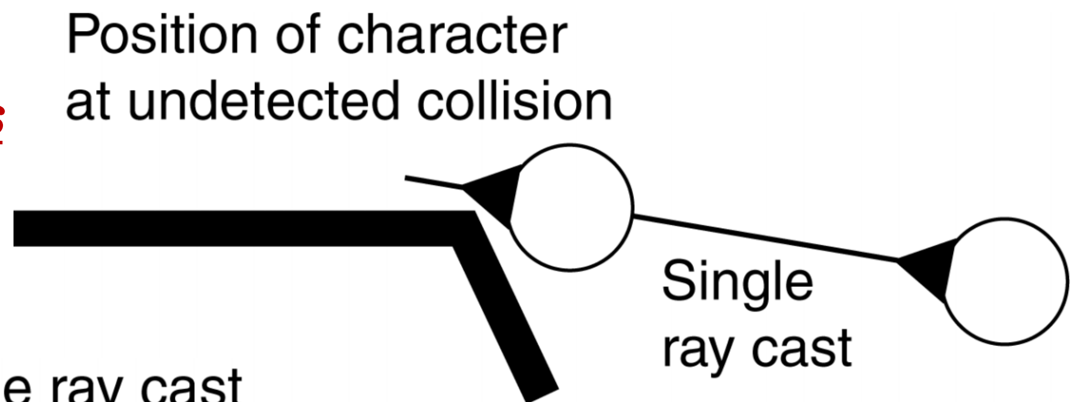
- **If these rays collide with an obstacle**
  - **The collision point and its normal are determined**
  - **A target location is created a fixed distance from the collision point in the direction of the normal,**
  - **and the character does a *basic Seek on this target***

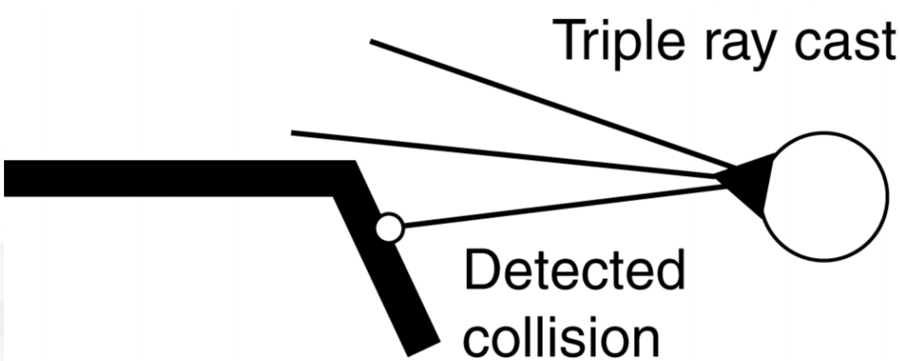


Collision Avoidance

Collision Detection Problems

- **In practice, detecting collisions with a single ray cast is <u>not a good solution</u>.**
- **Below, <u>*a one-ray character collides with a wall that it never detects*</u>.**



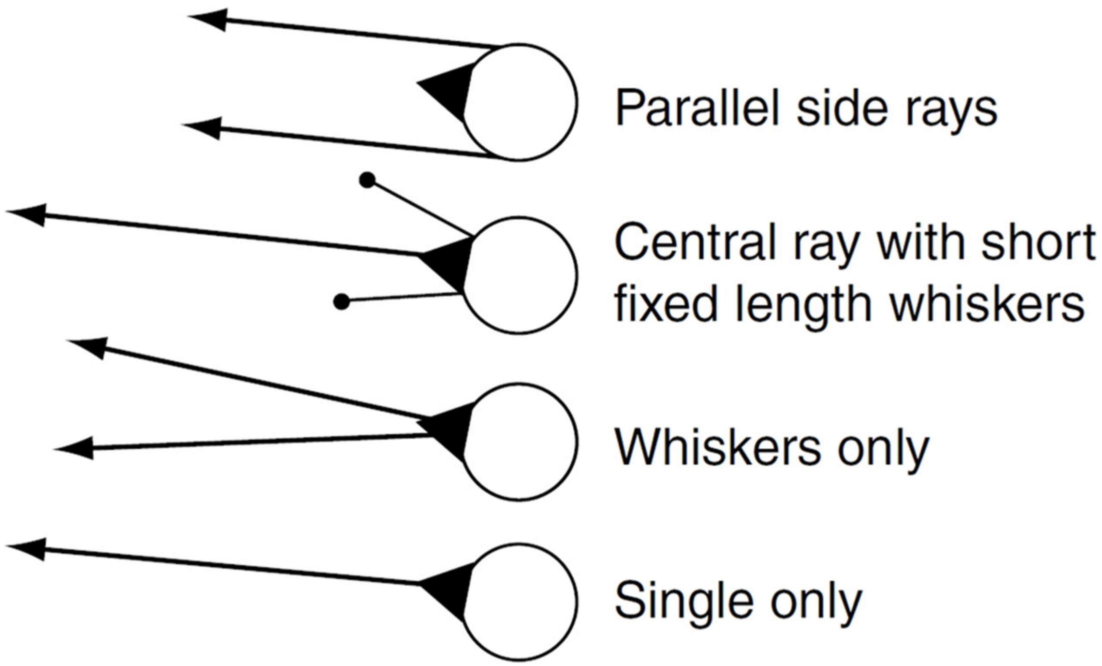


- **Typically, <u>*a character will need to have two or more rays*</u>.**

Collision Avoidance

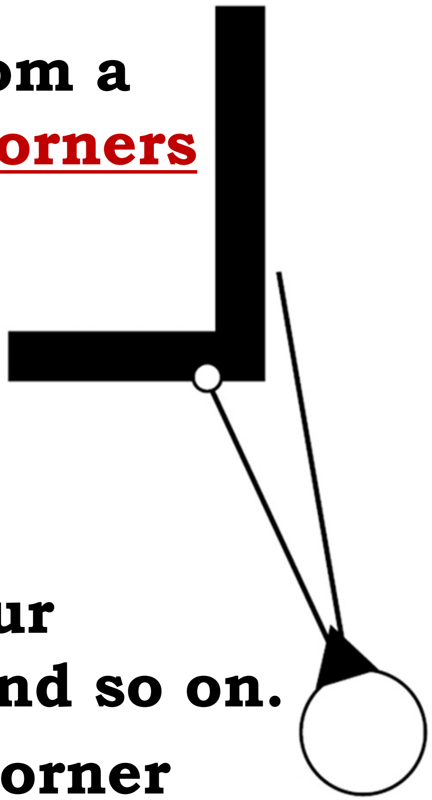
Basic Ray Configurations

- **Several basic ray configurations are *<u>used over and over for wall avoidance</u>*.**



Collision Avoidance

Corner Trap

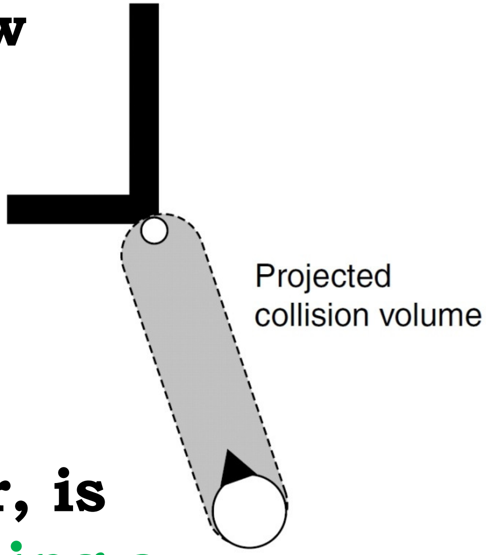


- Multi-ray wall avoidance can suffer from a crippling problem with acute angled corners
- In the example, the left ray is colliding with the wall. The steering behaviour will therefore ***turn it to the left*** to avoid the collision.
- Immediately, the right ray will then be colliding, and the steering behaviour will ***turn the character to the right***, and so on.
- Thus it will appear to home into the corner directly, until it slams into the wall. It will be unable to free itself from the trap.

Collision Avoidance

Avoiding Corner Traps

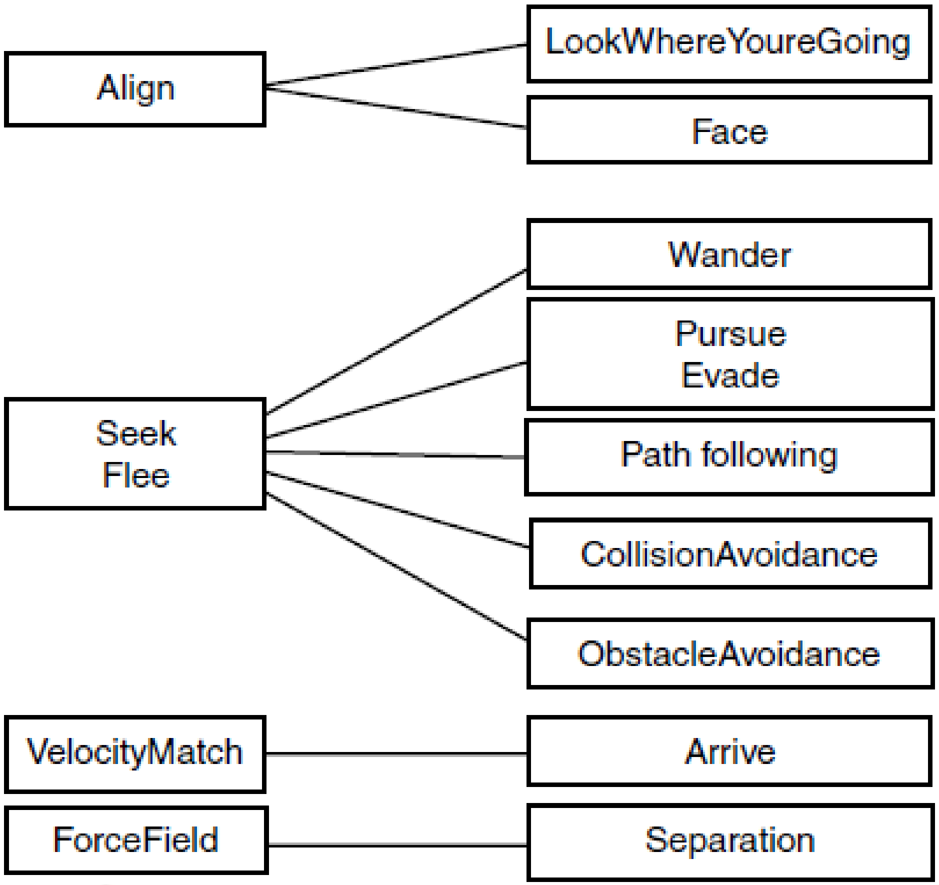
- A couple of strategies to avoid the corner trap:
  - A *fan structure, with a wide enough fan angle*, alleviates this problem. There is a trade-off between wide fans to avoid corners and narrow fans to navigate narrow corridors.
  - *Adaptive fan angles*: While no collisions, the fan angle is narrowed. If a collision is detected, then the fan angle is widened



- The only complete solution, however, is to perform the collision detection using a projected volume (extrusion) rather than a ray.

Steering Behaviors

Steering "Family Tree"



Steering Behaviors

Why do simple behaviours work?

- Steering behaviours effectively ***distribute their thinking over time***.
- Each decision they make is **very simple**, but because they are constantly reconsidering the decision, the overall effect is competent.
- If a behaviour makes a decision based, say, on an incorrect assumption about the trajectory of a target, the error will be quickly corrected over the next few times it runs.