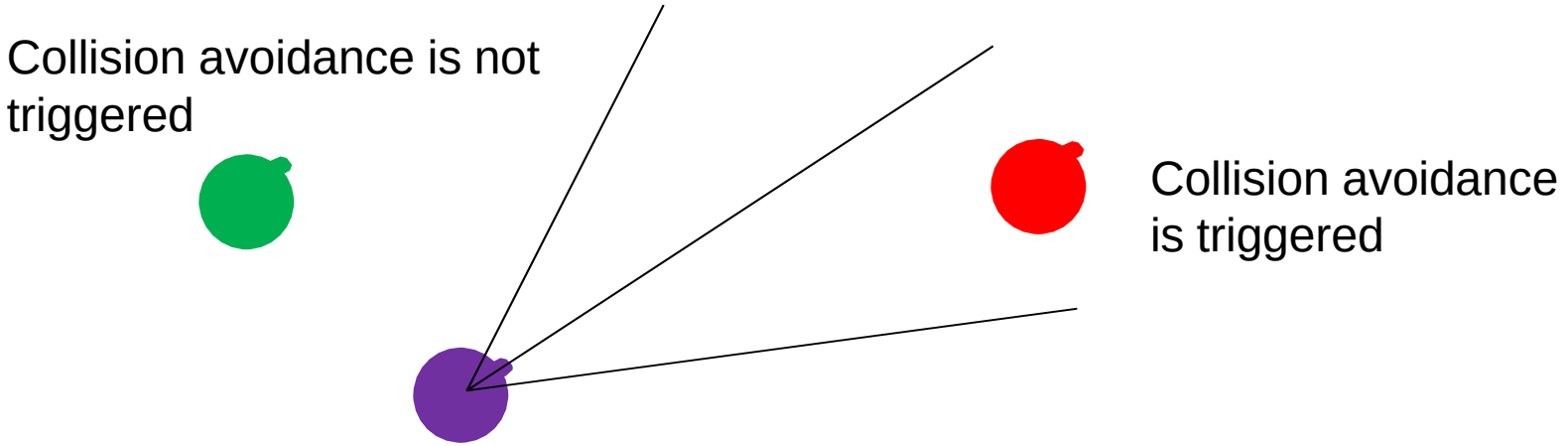
Dynamic Movement

- Collision Avoidance
  - Can use variation of evade or separation behavior
  - Typically only include characters that are within a cone of orientation of character



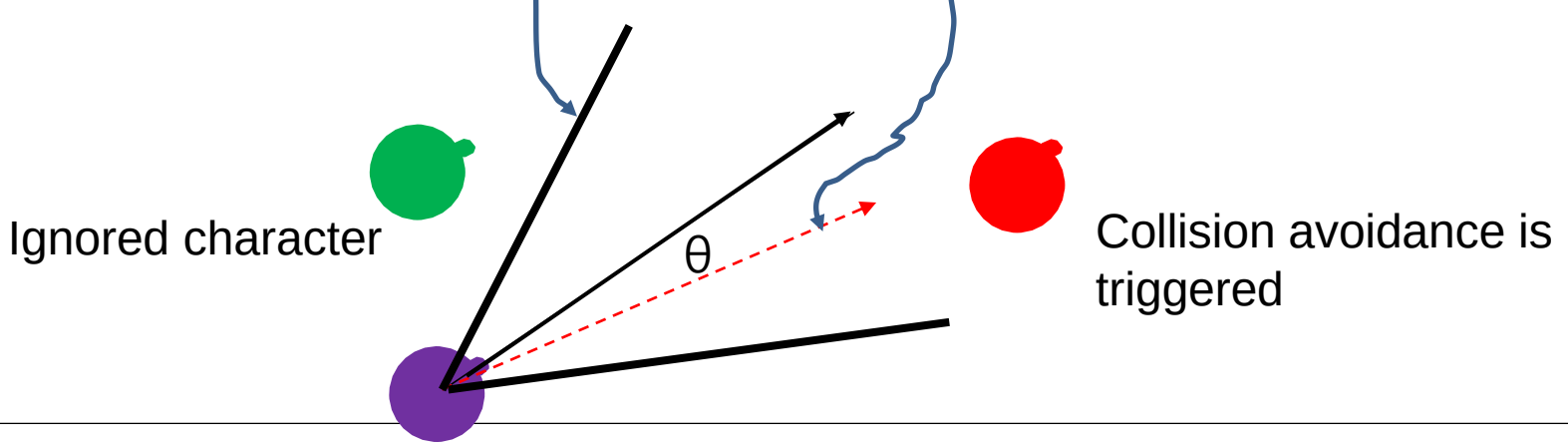
Dynamic Movement

- Collision Avoidance
  - cone check:

orientation.asVector() · direction ➔
cos θ 로서 θ는 두 벡터 사이 각도

```
if orientation.asVector() . direction > coneThreshold:
  # do the evasion
else:
  # return no steering
```

direction is the direction between the behavior's character and the potential collision point

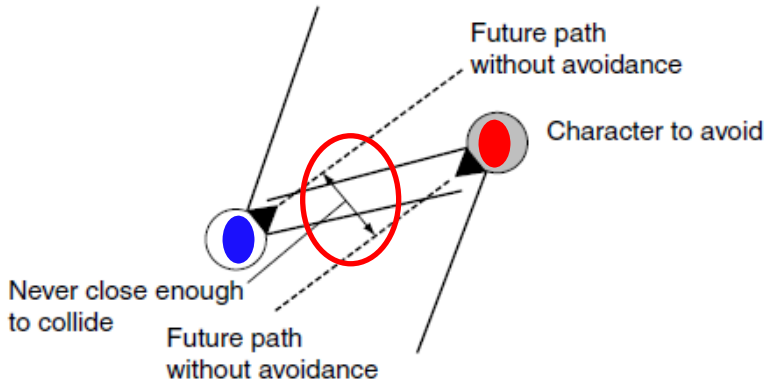


Dynamic Movement

- Collision Avoidance
  - Evasion behavior triggered within a cone does not work **for large crowds of characters** ← average position and speed of all characters in the cone or the position and speed of the closest character.
  - "**Panic" reaction for collison avoidance**

    even though collision is not imminent

    when algorithm ignores velocity and speed of other characters

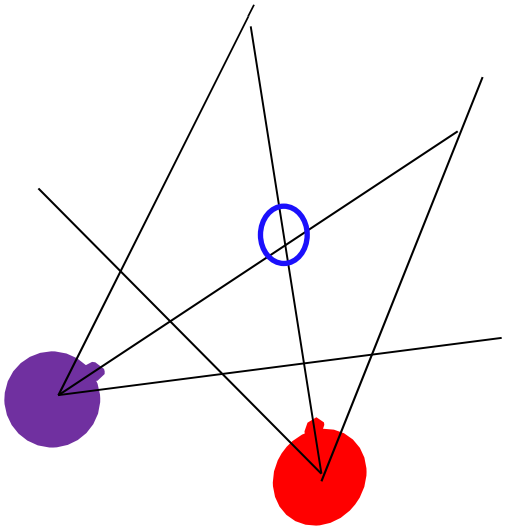


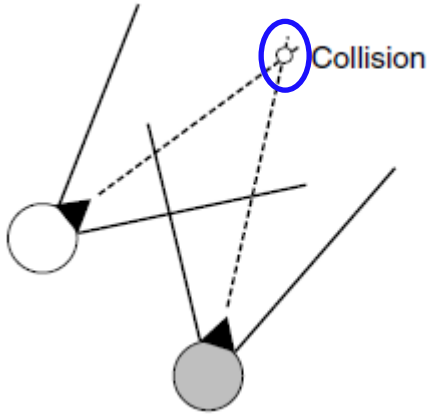
Two in-cone characters who will not collide

Dynamic Movement

- Collision Avoidance
  - Out-of-cone characters can collide

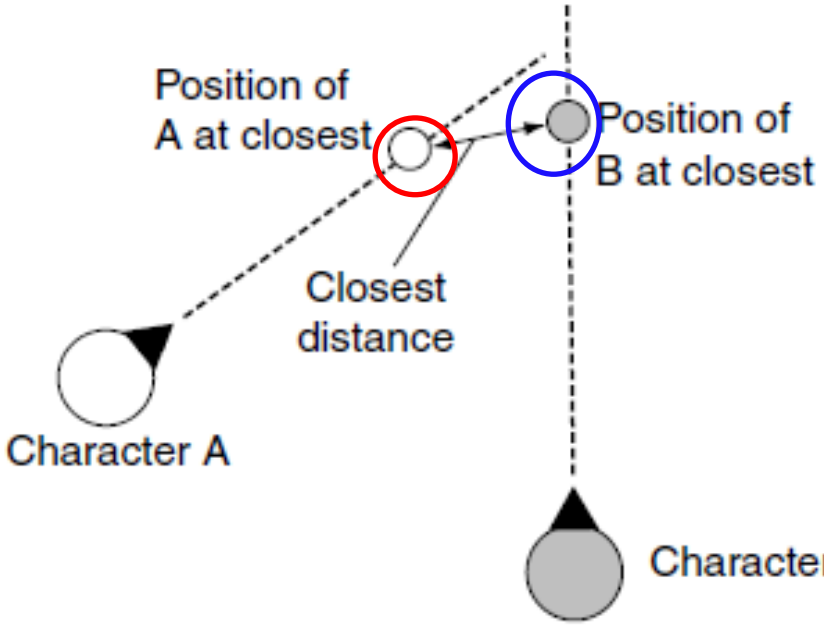
VO velocity obstacle
RVO





Dynamic Movement

- Collision Avoidance (using collision prediction)
  - Calculate closest approach between characters in the future.
  - Determine whether distance at this point is less than a threshold distance before triggering evasive behavior



Collision avoidance is triggered?

character c 와 target t 가 가장 가까워질 때, 그 시점과 거리는?

The time of closest approach is given by

$$t_{\text{closest}} = -\frac{d_p . d_v}{|d_v|^2},$$

where d_p is the current relative position of target to character

$$d_p = p_t - p_c$$

and d_v is the relative velocity:

$$d_v = v_t - v_c.$$

$Q = (P_c + v_c{*}t) - (P_t + v_t{*}t)$

|Q|를 최소로 하는 t ➔ $t_{closest}$

The position of character and target at the time of closest approach can be calculated:

$$p'_c = p_c + v_c t_{\text{closest}},$$

$$p'_t = p_t + v_t t_{\text{closest}}.$$

Dynamic Movement

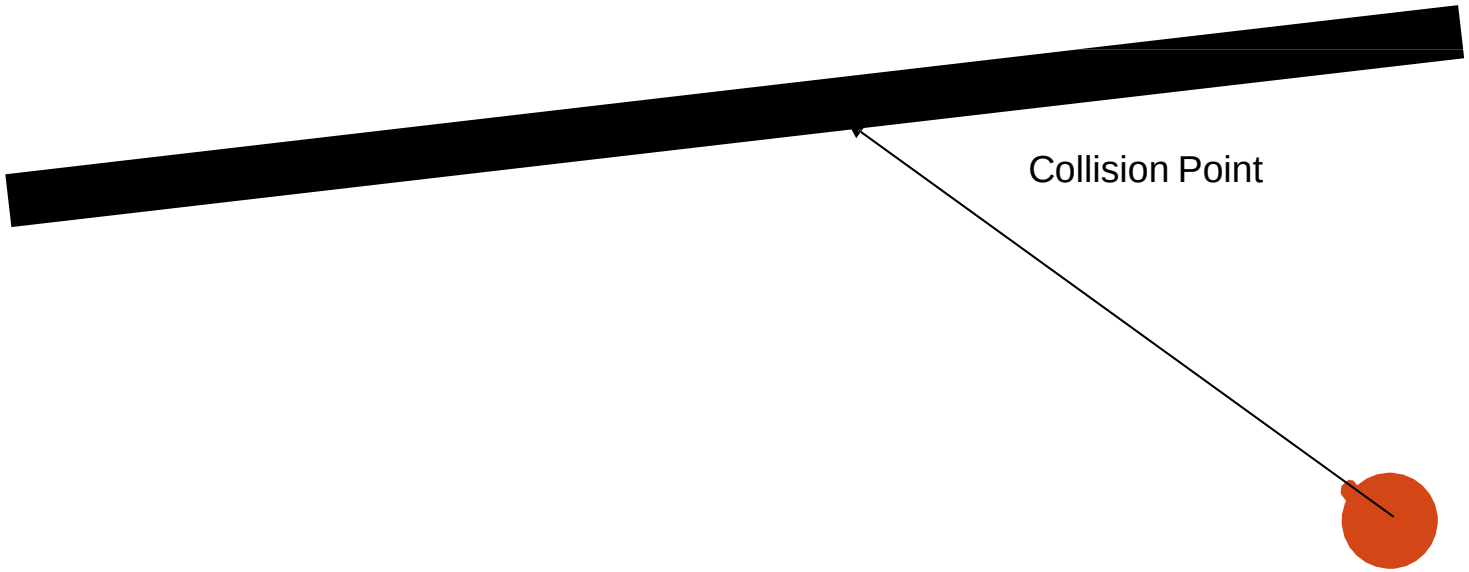
- Collision Avoidance
  - Avoiding colliding with coordinated groups of characters
    - Average velocity / position does not work well
    - Instead: Calculate character in group with whom collision is most likely

Dynamic Movement

- Obstacle and Wall Avoidance
  - Simple obstacles are represented by bounding spheres
  - More complicated objects such as walls, houses, cliffs, etc cannot be represented by bounding spheres
  - Characters use a different obstacle avoidance algorithm

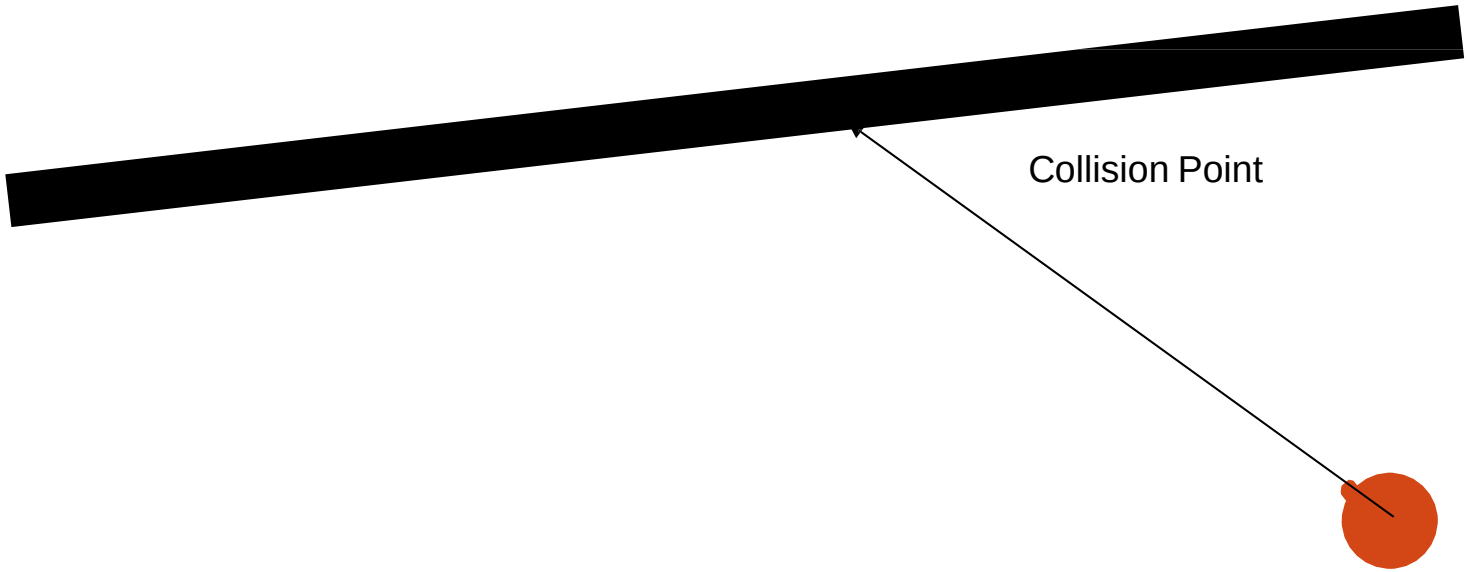
Dynamic Movement

- Obstacle and Wall Avoidance
  - Character casts out a single limited distance ray



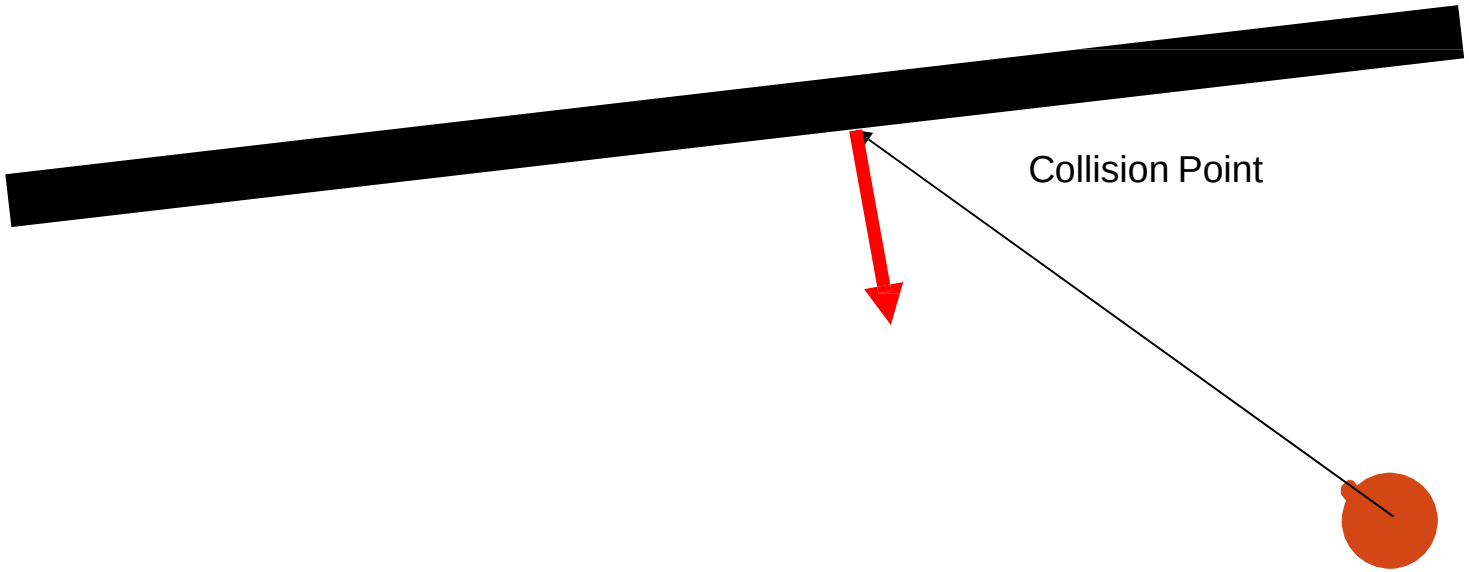
Dynamic Movement

- Obstacle and Wall Avoidance
  - Calculate Collision Point



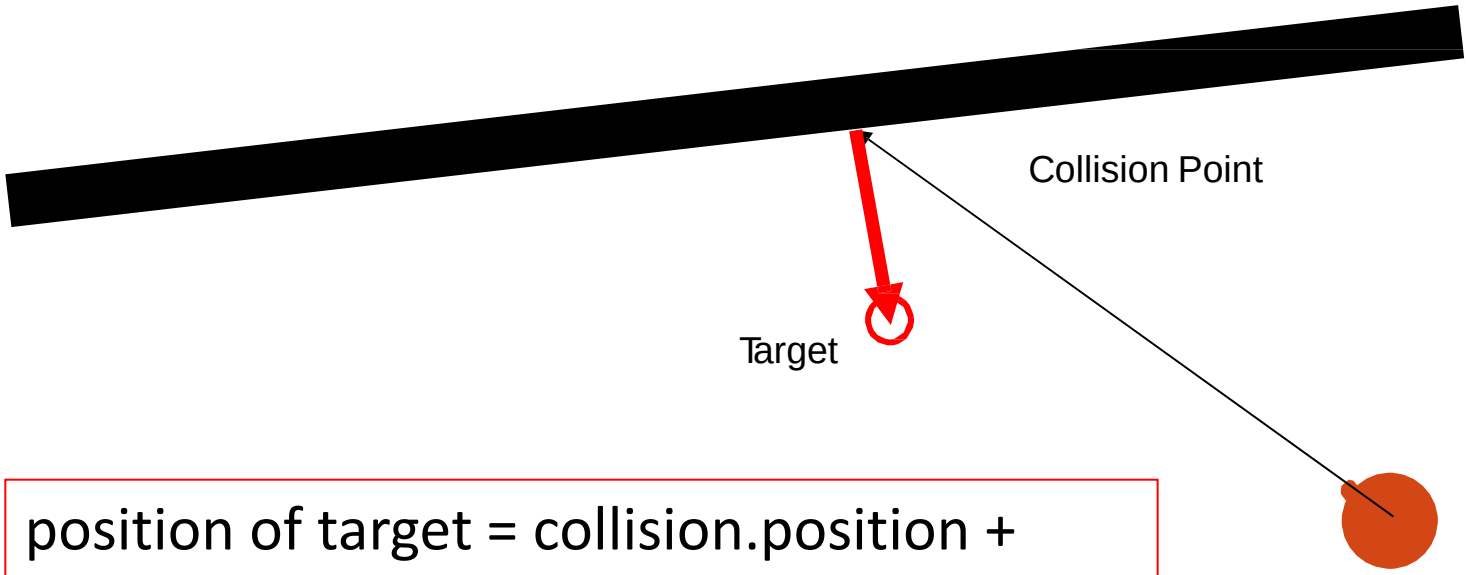
Dynamic Movement

- Obstacle and Wall Avoidance
  - Calculate normal of obstacle at collision point



Dynamic Movement

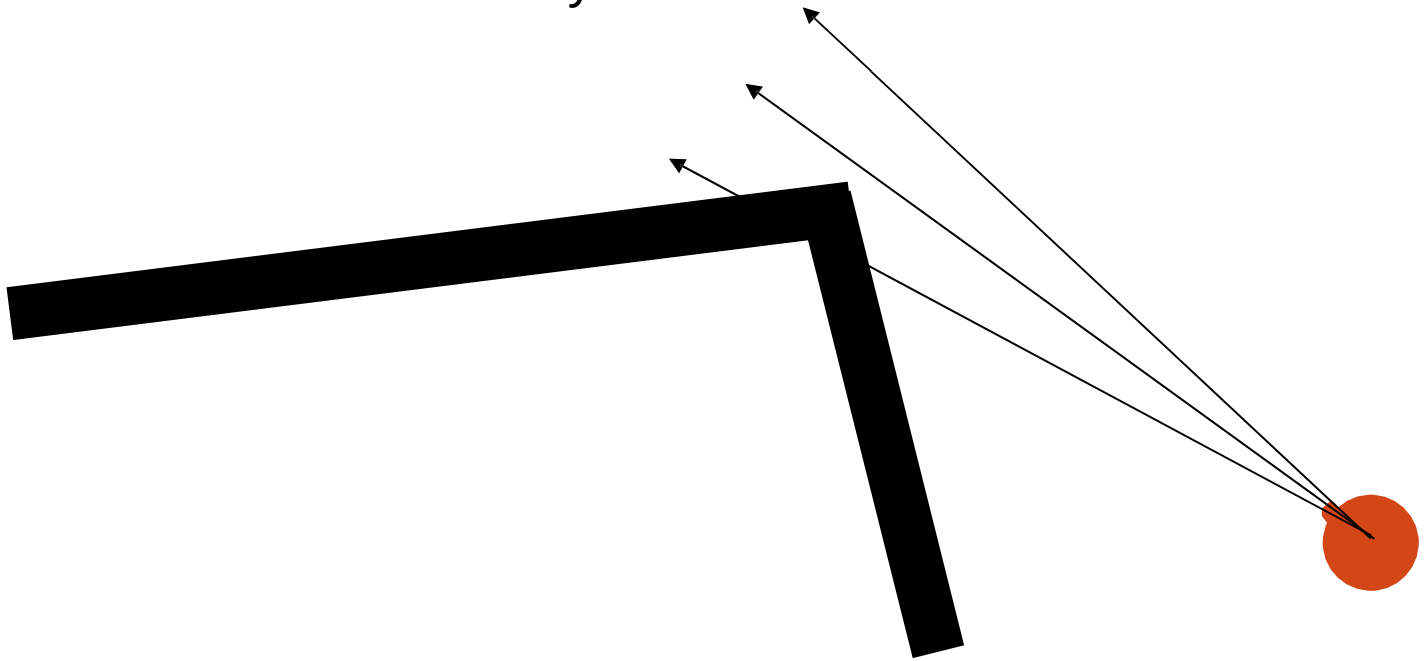
- Obstacle and Wall Avoidance
  - Set target on normal at avoidance distance
  - Go to Seek



position of target = collision.position + collision.normal * avoidDistance

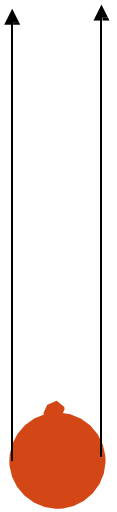
Dynamic Movement

- Obstacle and Wall Avoidance
  - Ray casting can be very expensive
  - Single ray might not detect collision
    - Use three rays instead



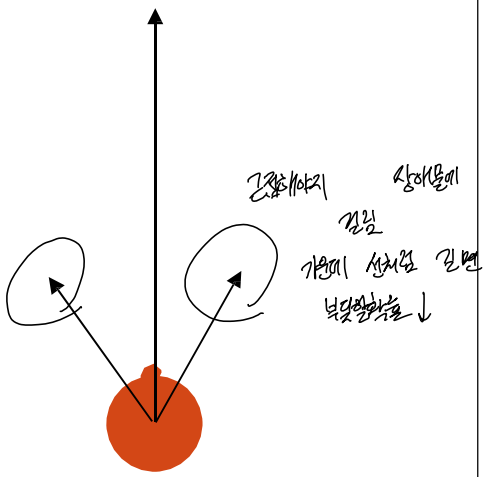
Dynamic Movement

- Obstacle and Wall Avoidance
  - Several configurations tried for rays
    - **Parallel side rays**
    - Central ray with short whiskers
    - Whiskers only
    - Single ray only



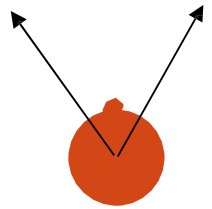
Dynamic Movement

- Obstacle and Wall Avoidance
  - Several configurations tried for rays
    - Parallel side rays
    - **Central ray with short whiskers**
    - Whiskers only
    - Single ray only



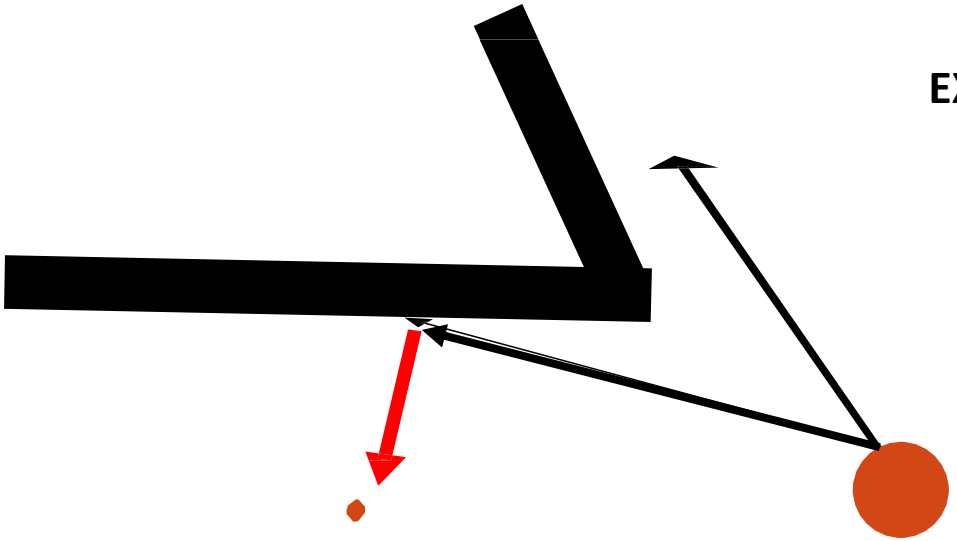
Dynamic Movement

- Obstacle and Wall Avoidance
  - Several configurations tried for rays
    - Parallel side rays
    - Central ray with short whiskers
    - **Whiskers only**
    - Single ray only



Dynamic Movement

- Obstacle and Wall Avoidance
  - Corner Trap Problem



EX) the corner trap problem for multiple rays

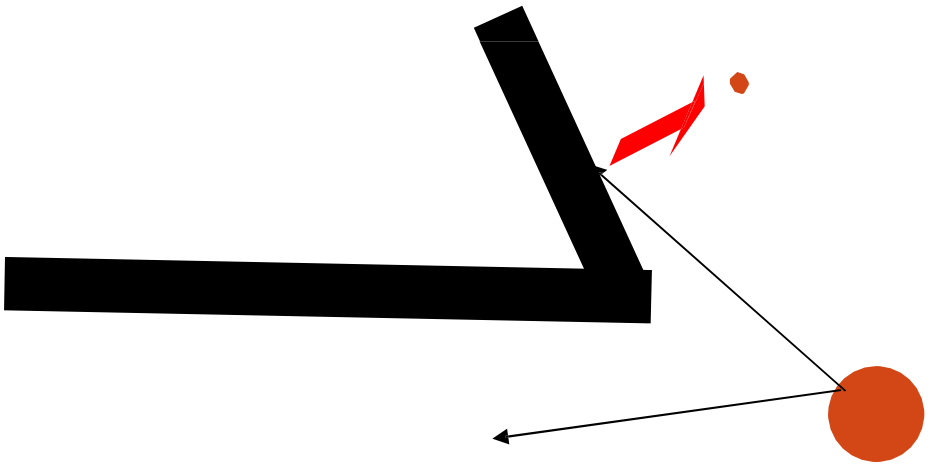
Left ray detects collision

Right ray detects no collision

Whisker based algorithm sets target to the right

Dynamic Movement

- Obstacle and Wall Avoidance
  - Corner Trap
  - Algorithm steers character straight towards the corner



Now right ray detects collision

Left ray detects no collision

Algorithm sets target to the right

➔ 다음에는 ?

Dynamic Movement

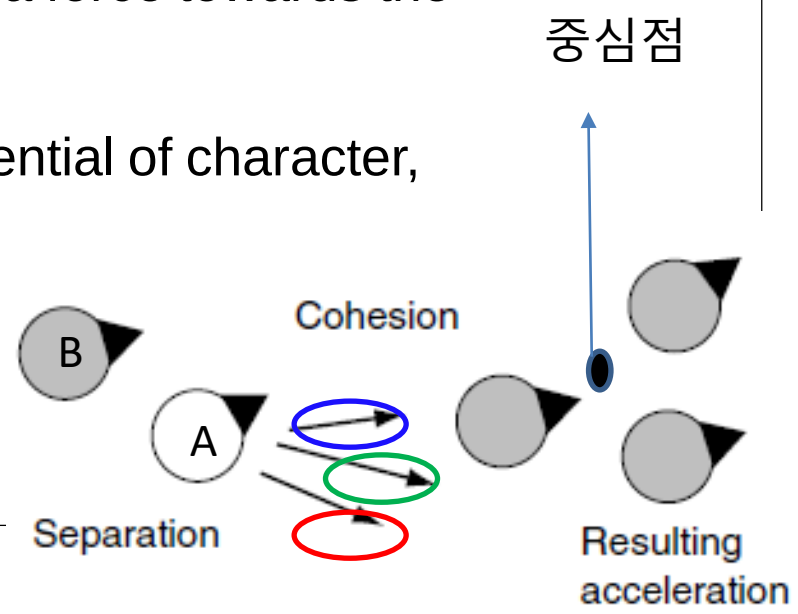
- Obstacle and Wall Avoidance: Corner Trap
  - Wide enough fan angle can avoid the problem
    - **But**, wide enough fan angle does not allow characters to walk through a door
    - Solutions:
      - Get level designers to make doors wide enough for AI characters
      - Use adaptive fan angles:
        - Increase fan angles if collisions are detected
        - Decrease fan angles if collisions are not detected
  - Run corner trap avoidance
    - If character moves erratically (left/right ray detect alternatively collisions), declare one ray to be the winner

Combining Steering Behavior

- Blending
  - Execute all steering behaviors
  - Combine results by calculating a compromise based on weights
    - Example: Flocking based on separation and cohesion
- Arbitration
  - Selects one proposed steering

Combining Steering Behavior

- Blending
  - Example: A group of rioting characters that need to act as a unified mob
    - Characters need to stay by the other
    - Characters need to keep a safe distance from each other in order not to bump into each other
  - For each character, calculate accelerations based on separation and cohesion
    - For the moment, define **cohesion** as a force towards the projected center of the group
    - Add both accelerations
    - Ensure that acceleration is within potential of character, otherwise crop acceleration

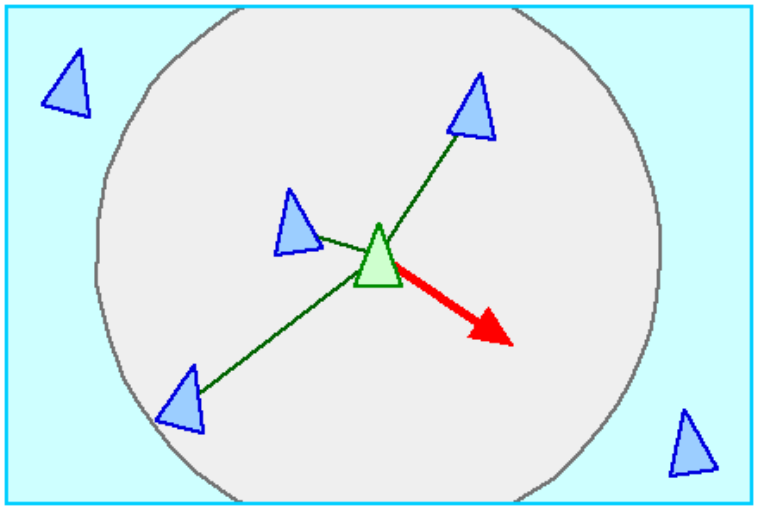


Combining Steering Behavior

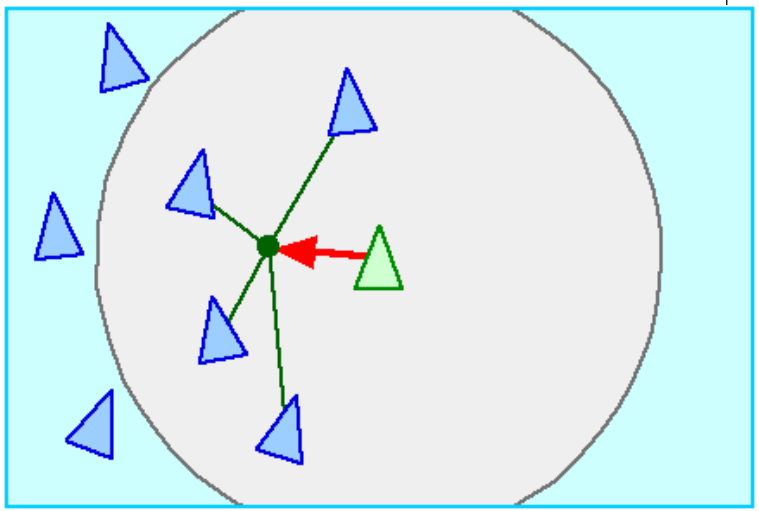
- Blending Weights
  - Blending weights do not need to add up to one
  - Blending weights can evolve
    - Learning algorithms
  - Trial and Error fixed parameters

Combining Steering Behavior

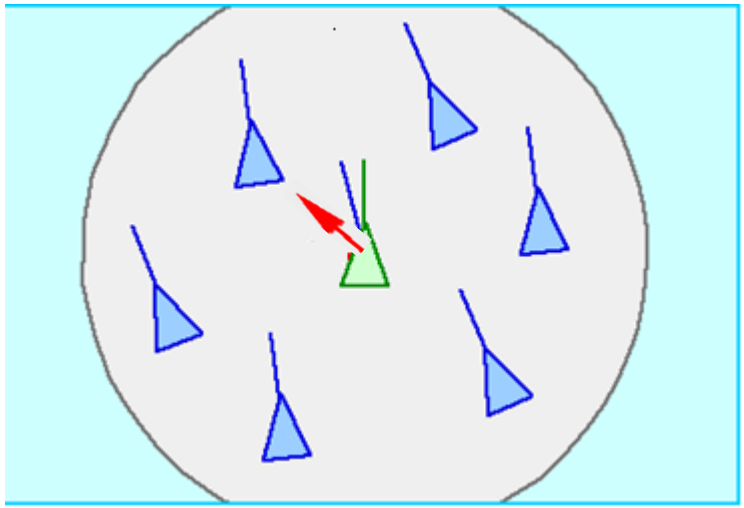
- Flocking and Swarming Algorithm
  - Craig Reynold's "boids"
    - Simulated birds
    - Blends three steering mechanisms
      1. Separation
         - Move away from other birds that are too close
      2. Cohesion
         - Move to center of gravity of flock
      3. Alignment
         - Match orientation and velocity
    - Equal Weights for simple flocking behavior



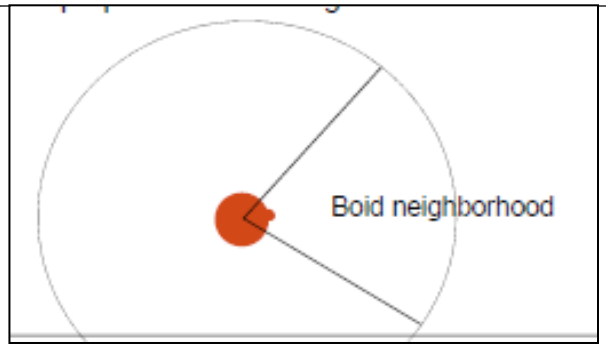
(a) separation

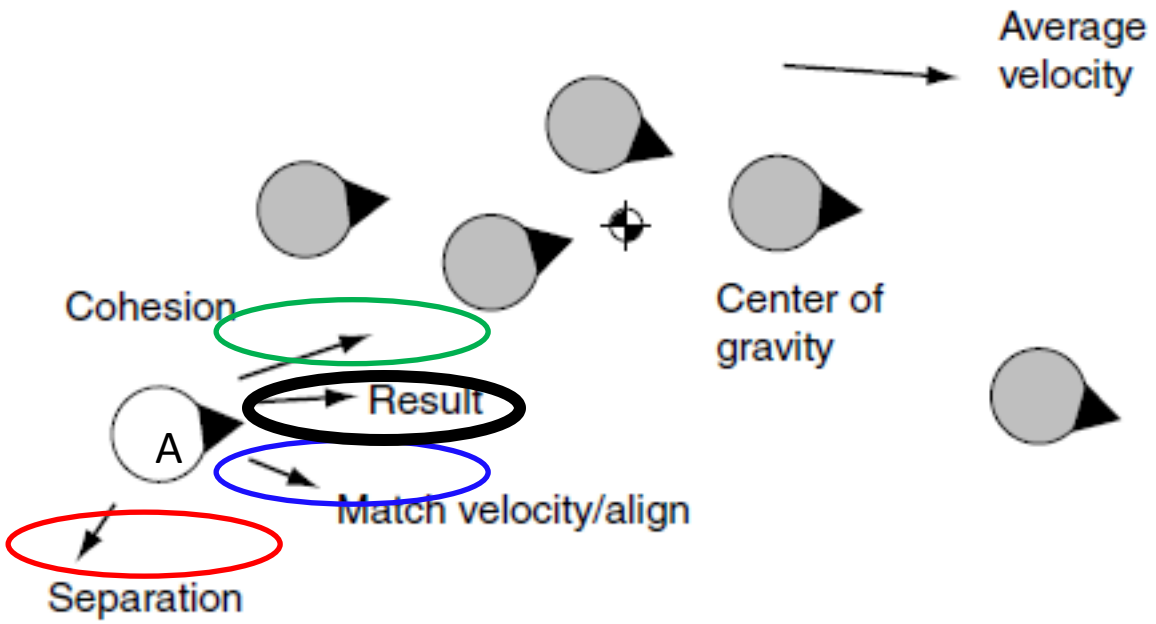


(b) cohesion



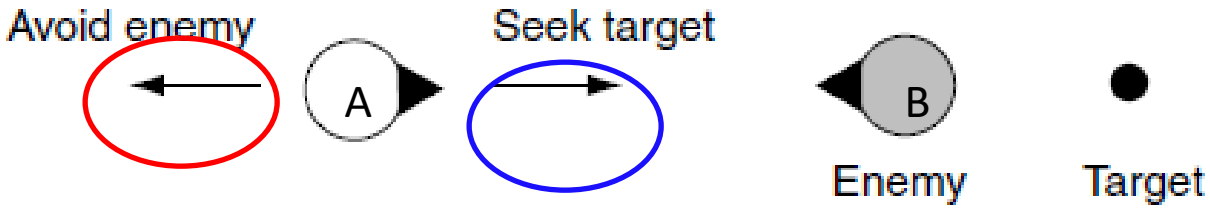
(c) alignment

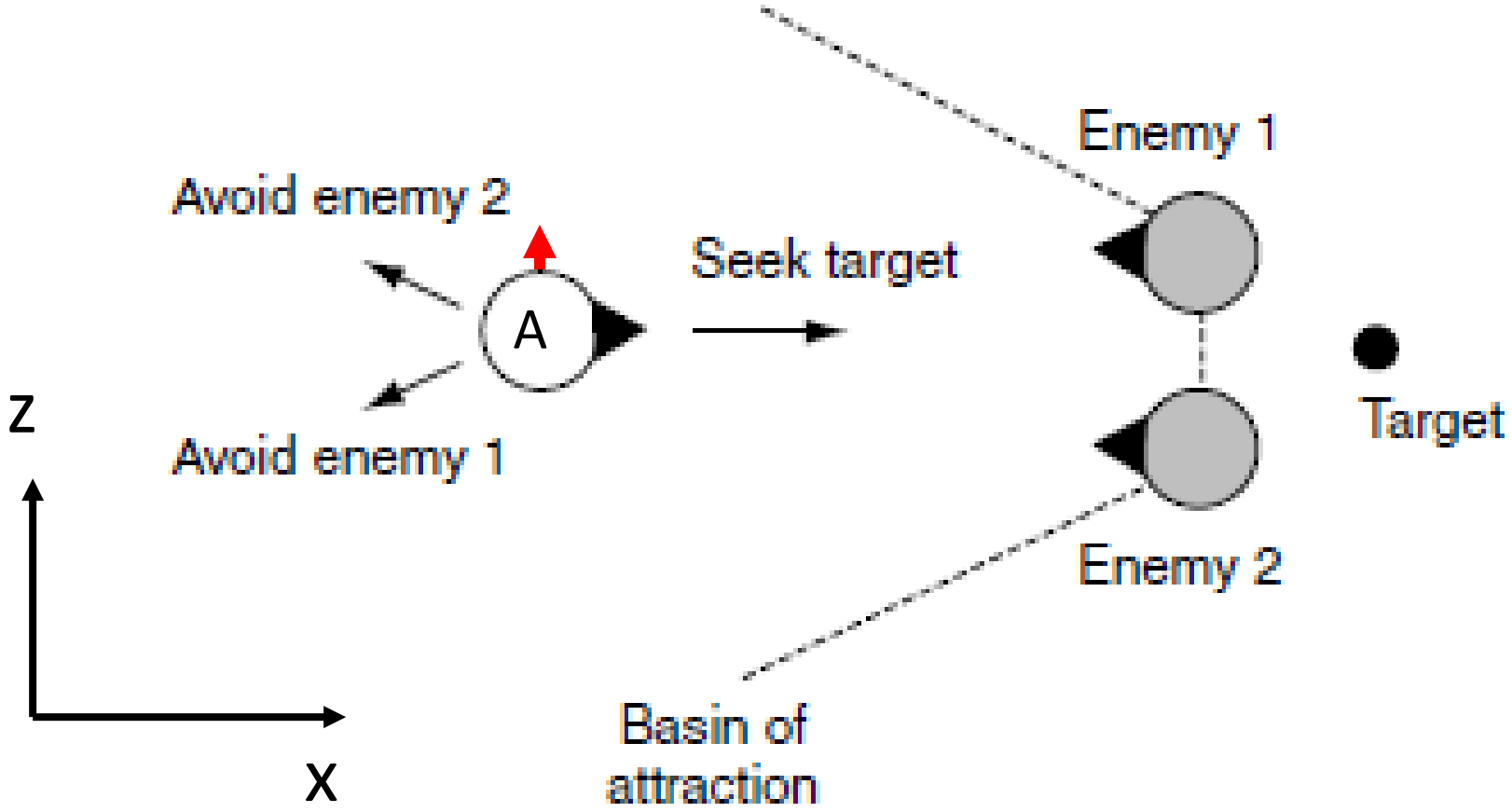

Boid neighborhood


Average
velocity
Cohesion
Center of
gravity
Result
A
Match velocity/align
Separation

Combining Steering Behavior

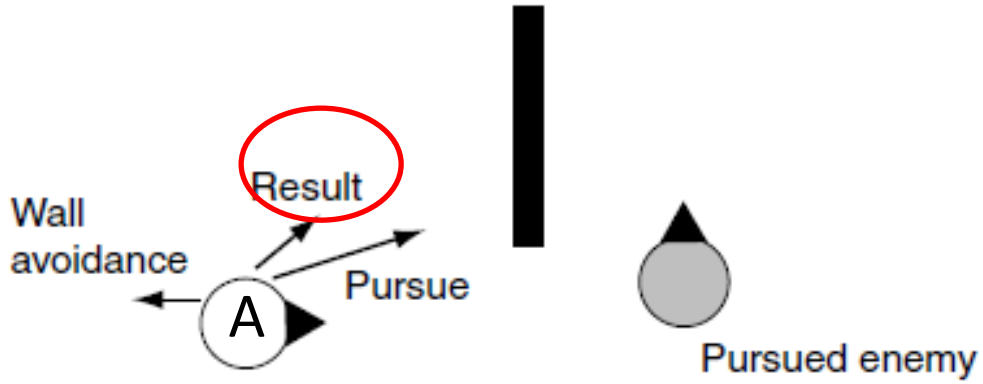
- Blending Problems
  - Characters can get stuck
    - Unstable equilibrium between “seek target” and “flee enemy” steering if enemy is between character and target
    - If enemy and target are stationary, small rounding errors usually let character flip laterally with increasing amplitude until character finally makes dash for target





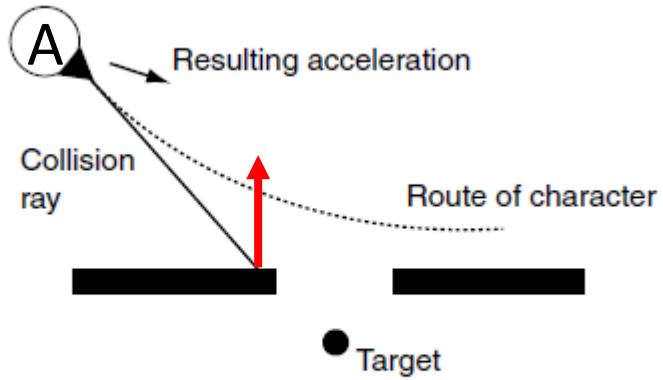
A stable equilibrium

Another blending problem:



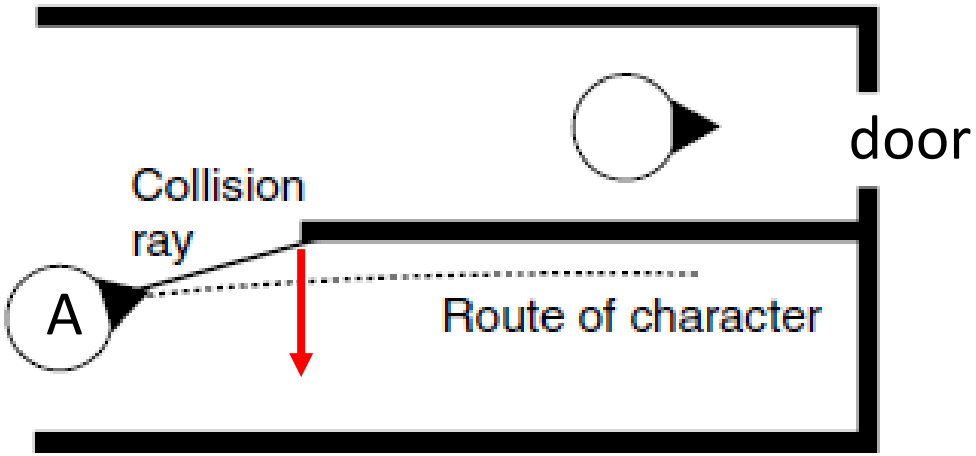
Can't avoid an obstacle and chase

Blending Problems



Missing a narrow doorway

Blending Problems



Long distance failure in a steering behavior

(arbitration)

- Priority Behavior
  - Can use a fixed order
  - Can use dynamic order
    - Different components return also a priority value
      - Collision avoidance might return square inverse of distance to obstacle
    - Select behavior based on priority value
      - Collision avoidance has no influence if collision is long away
      - But becomes hugely important just before a collision threatens

음성 강의 종료