Lecture Note : Game Algorithm 1
Movement 중

Dynamic Movement

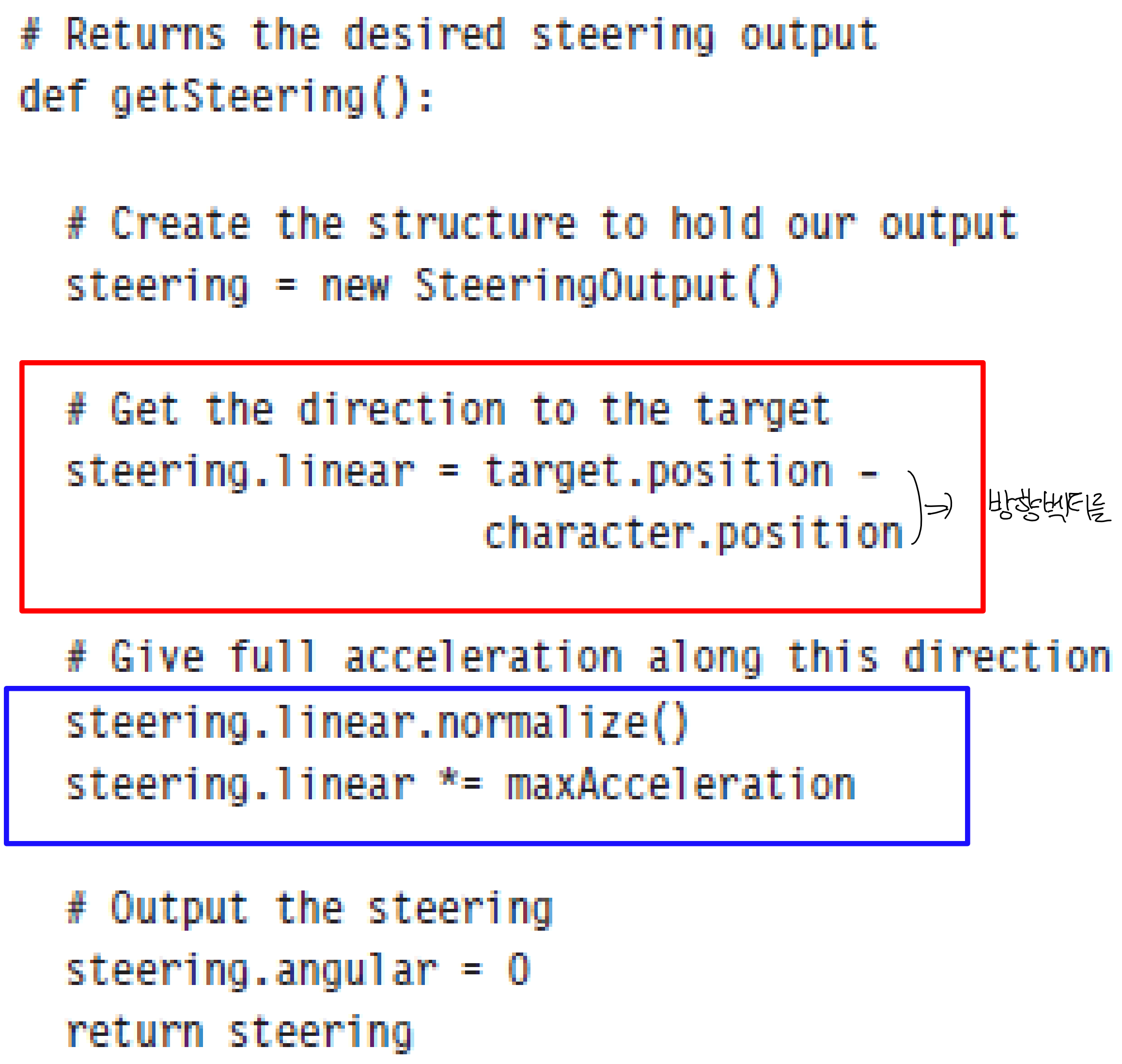
가속, 감속, 회전가속도
을 구함.

캐릭터위치 → target
까지 가는 가속도 구하기

Δt를 이용해 속도구하고
캐릭터 이동

- Seek Algorithm
  - Seek will approach target at maximum speed

    가속도값이 너무 커지는 것을 방지
  - Will probably miss it by a little bit
    - Use a target radius to stop when close enough
  - Need an approach algorithm
    - Used when close to target
    - Triggered when within approach radius of target
    - Calculates desired speed based on **time to target** and **distance to target**



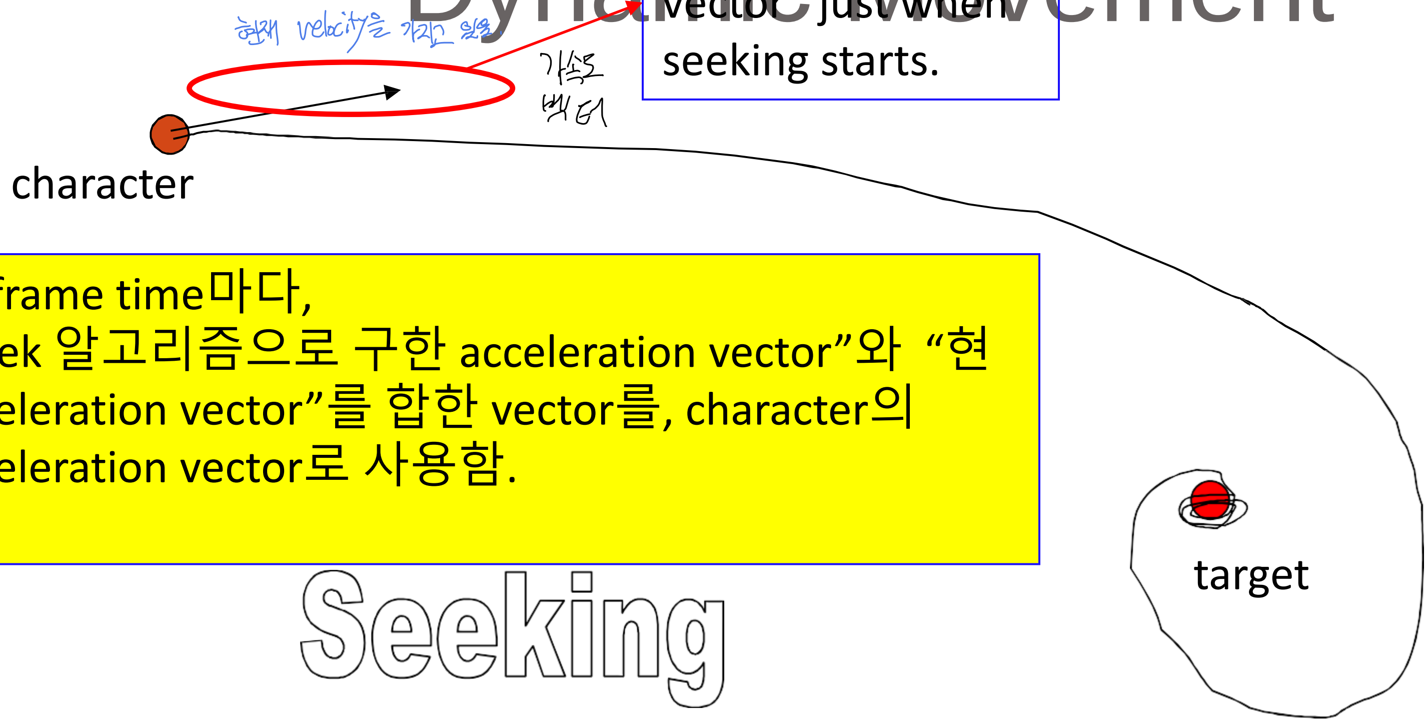
```
# Returns the desired steering output
def getSteering():

  # Create the structure to hold our output
  steering = new SteeringOutput()

  # Get the direction to the target
  steering.linear = target.position -
                    character.position

  # Give full acceleration along this direction
  steering.linear.normalize()
  steering.linear *= maxAcceleration

  # Output the steering
  steering.angular = 0
  return steering
```

⇒ 방향벡터를 얻게됨.

- Seek Algorithm

```
INPUT: target and character position
OUTPUT: steering update
void steering.update(Target & tar, Character & ch)
{
  this.linear = tar.position – ch.position;
  if(this.linear.length() > MAXACCELERATION)
         this.linear *=
MAXACCELERATION/this.linear.length();
}
```

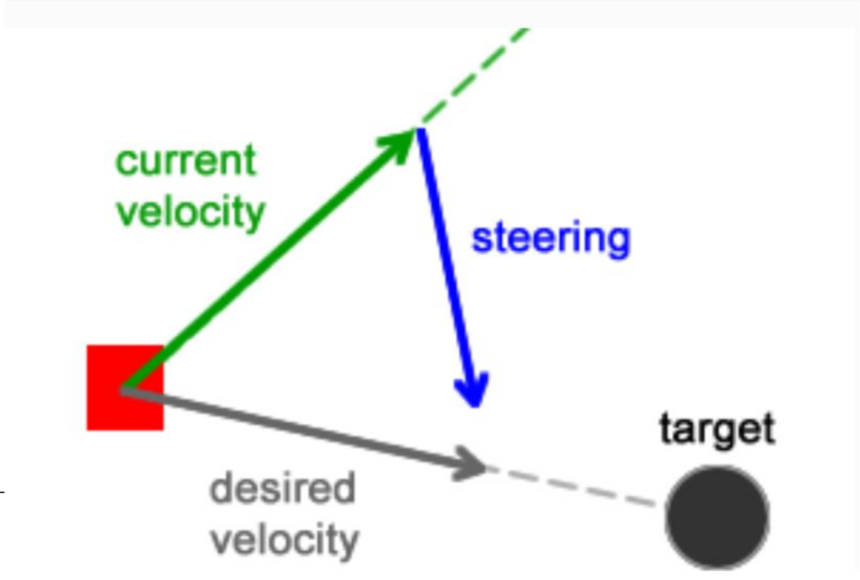
Dynamic Movement



the acceleration vector just when seeking starts.

매 frame time마다,
“Seek 알고리즘으로 구한 acceleration vector”와 “현 acceleration vector”를 합한 vector를, character의 acceleration vector로 사용함.

Seeking



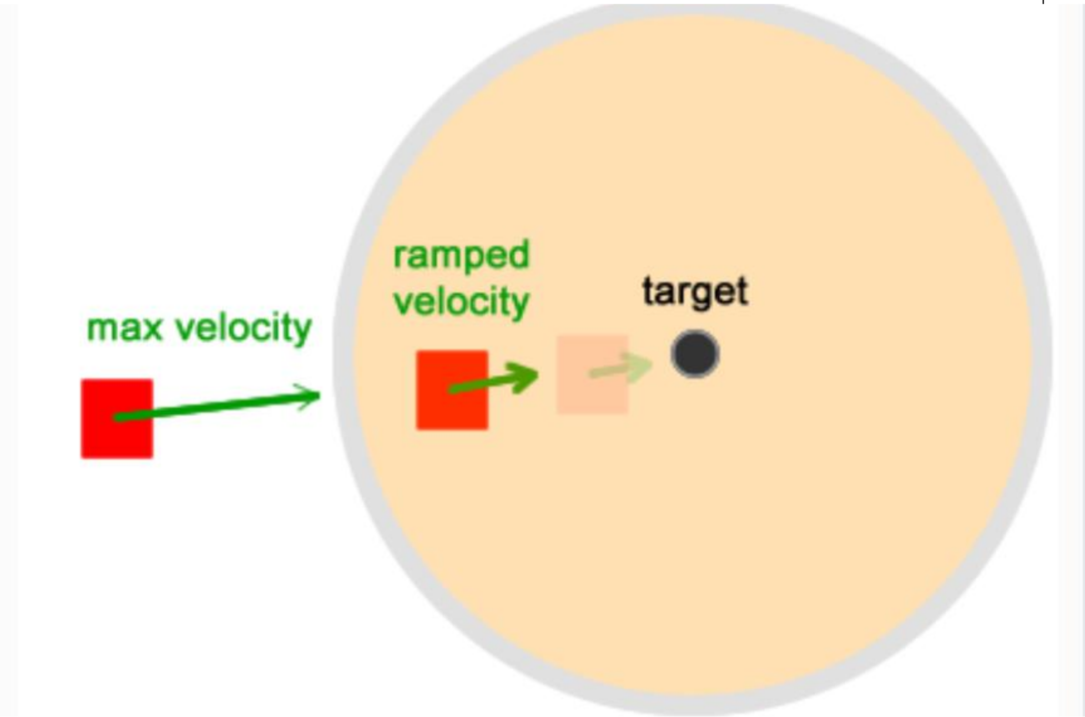
Dynamic Movement

- Arrival algorithm

가까우면 천천히 멀리있을경우는 빠르게

  - Define radius within which character slows down
    - Calculate optimal speed to target

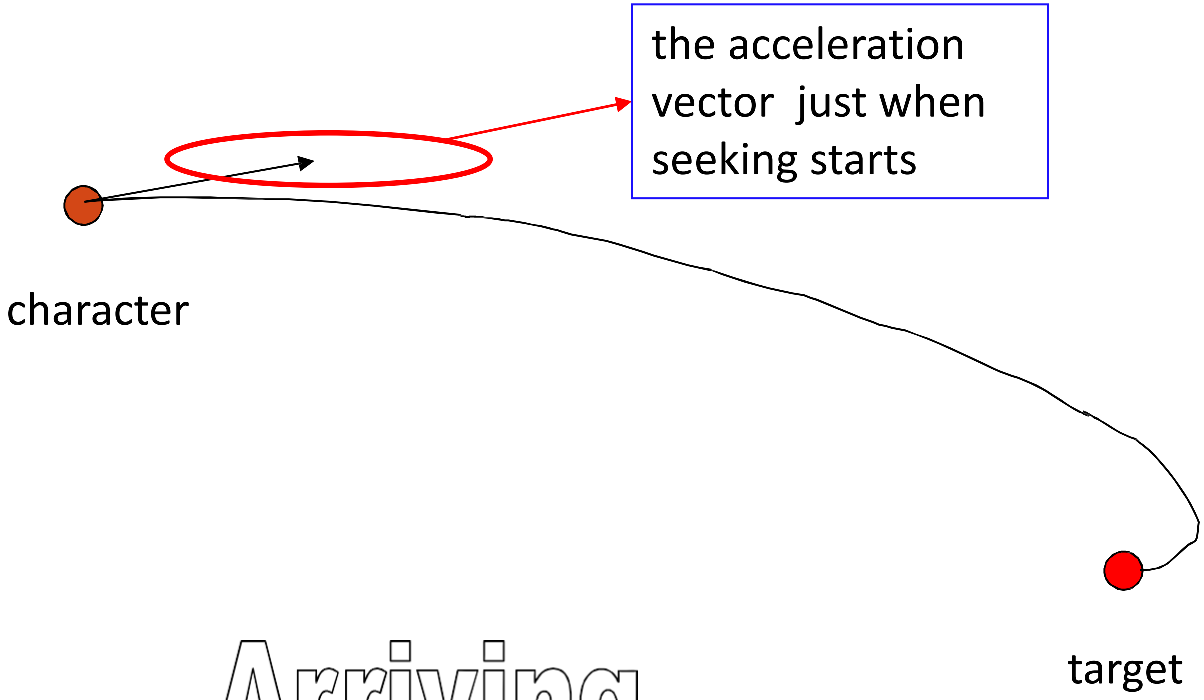
      If current speed is slower, accelerate towards target
    - If current speed is faster, accelerate away from the target
  - Many algorithms used do not have a target radius



```
INPUT: target and character position
OUTPUT: steering update
void Steering::update(Target & tar, Character & ch, double&
radius, double& timeToTarget)
{
                 target               character
  Vector2D dir = tar.position - ch.position;
  double distance = dir.Length();
  if (distance < Epsilon)
   Zero(); // close enough, stop steering
  double targetSpeed;
  if (distance > radius) targetSpeed = maxSpeed;
  else targetSpeed = maxSpeed * ( distance / radius);
  Vector2D targetVelocity = (dir/distance)*targetSpeed
  this.linear = targetVelocity - ch.velocity;
  this.linear/= timeToTarget;//linear를 timeToTarget로조정
  if(this.linear.Length()>MAXACCELERATION)
    this.linear /= MAXACCELERATION/linear.Length();
  this.angular = 0;   }
```

Arrival 기능(즉, target에 가까이 다가가면 speed 를 줄임)을 가진 Seek Algorithm을 적용한 결과 예시

Dynamic Movement



the acceleration vector just when seeking starts

character

target

Arriving

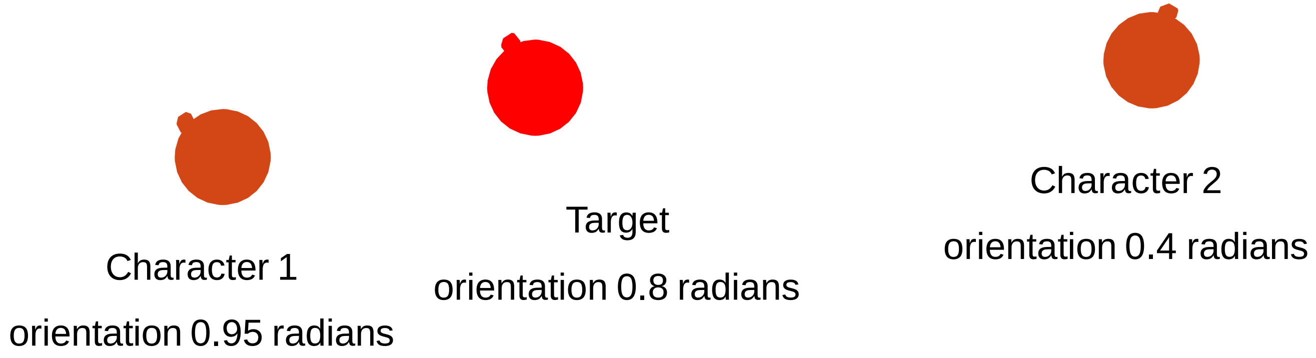
Dynamic Movement

- Seeking a moving target
    - Current Seek / Arrive algorithm will follow a dog's curve
    - More realistic behavior
        - **Calculate position of target some time in the future** based on current position and velocity
        - This fits the intelligent agent model since it used information that can be perceived

Character나 target이
쳐다보는 (facing) 방향

Dynamic Movement

- Aligning (Orientation Matching)
  - Match orientation of character to that of another character
  - Careful because orientation is a value modulo 2π, so difference is misleading
  - Subtract character orientation from target orientation and change to be between Aπ and +π

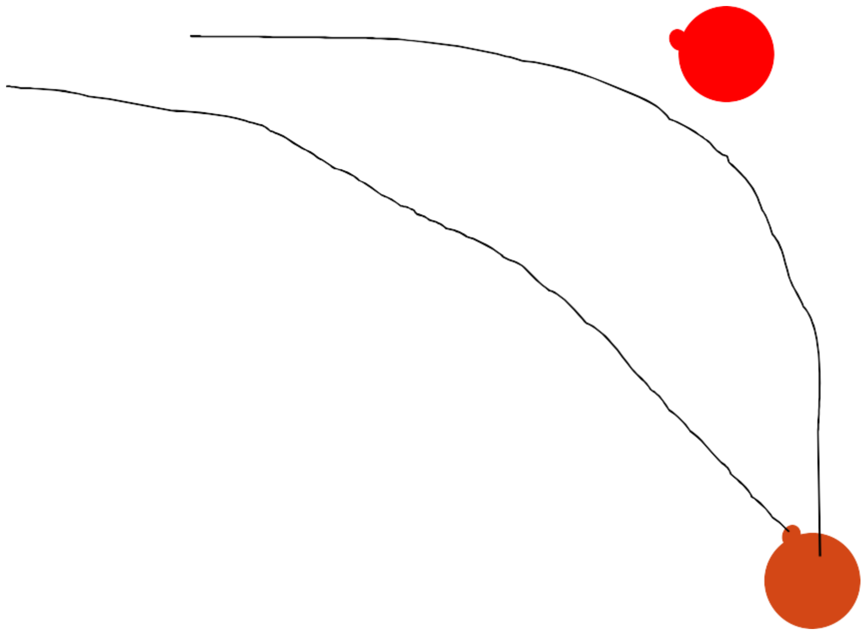


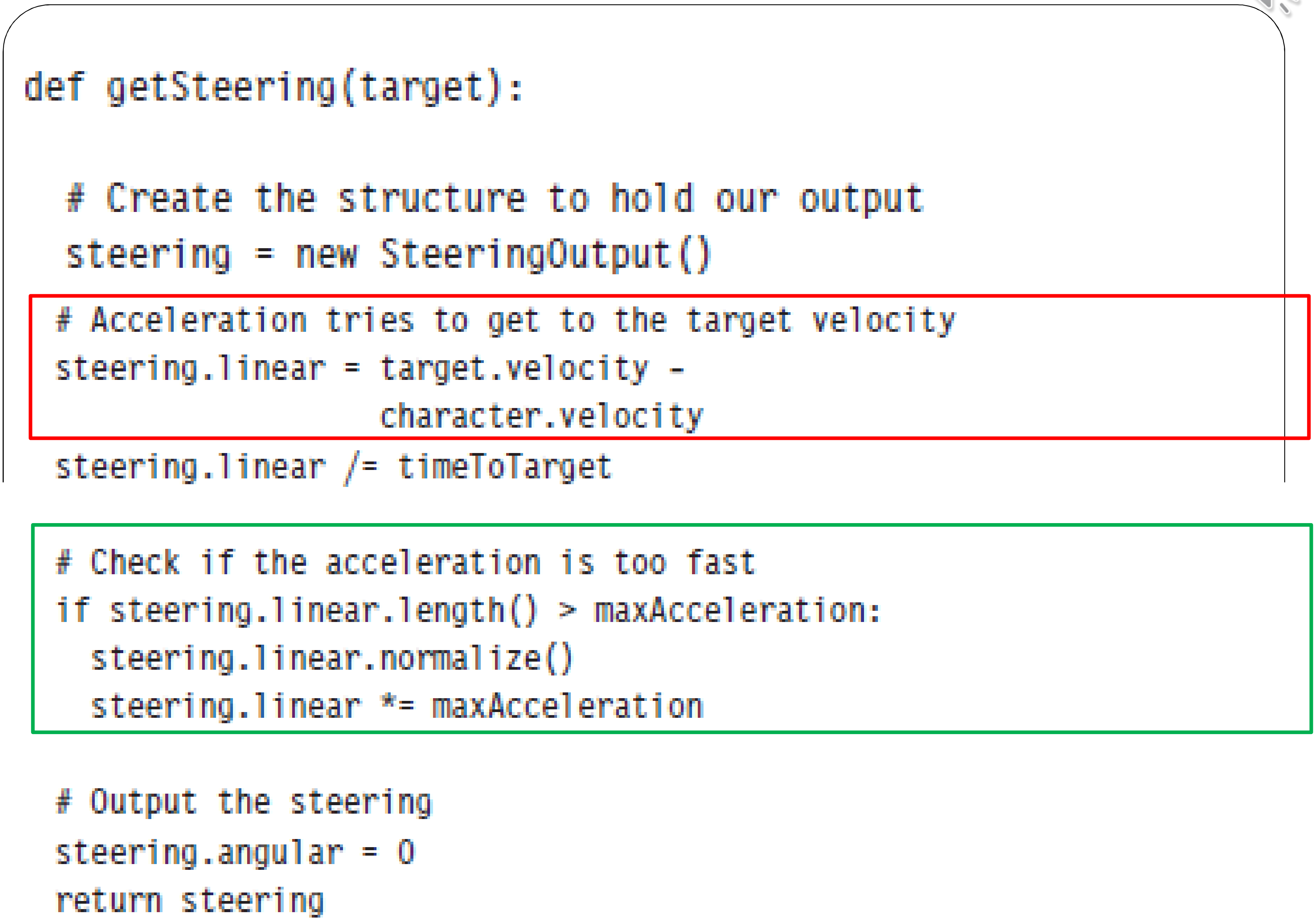
```
rotation = target.orientation - character.orientation

rotation = mapToRange(rotation) # Map the result to
                                # the $(-\pi, \pi)$ interval
rotationSize = abs(rotation)
if rotationSize > slowRadius:
    targetRotation = maxRotation
else:
   targetRotation = maxRotation * (rotationSize/slowRadius)

targetRotation *= rotation / rotationSize
steering.angular = targetRotation - character.rotation
steering.angular /= timeToTarget
```

Seek/arrival 알고리즘에서
velocity → rotation
position → orientation
linear → angular
radius → slowRadius

Dynamic Movement

- Velocity Matching
  - Change velocity of character to target velocity
  - Check for acceleration limits



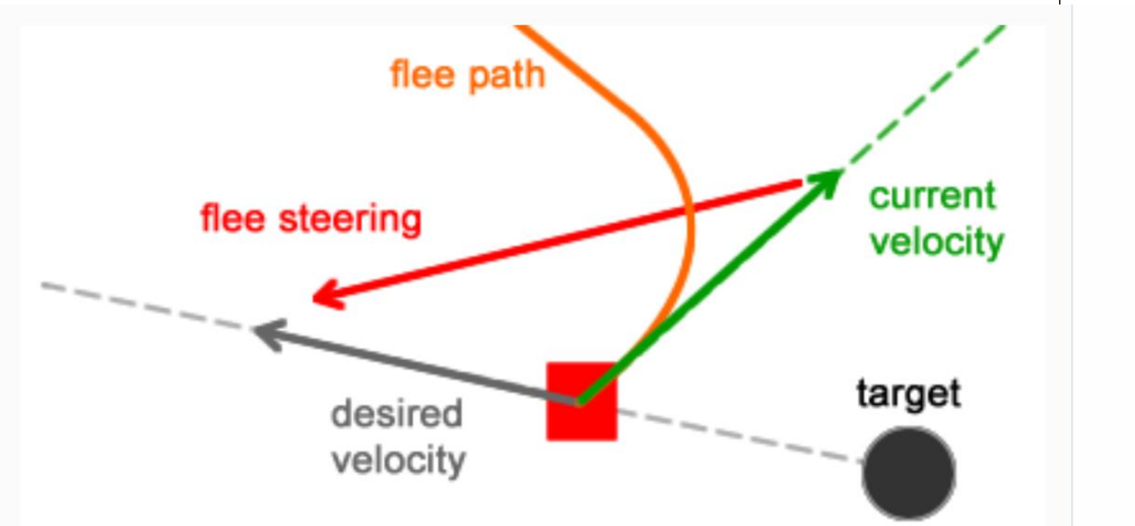


```
def getSteering(target):

  # Create the structure to hold our output
  steering = new SteeringOutput()

  # Acceleration tries to get to the target velocity
  steering.linear = target.velocity -
                    character.velocity
  steering.linear /= timeToTarget

  # Check if the acceleration is too fast
  if steering.linear.length() > maxAcceleration:
    steering.linear.normalize()
    steering.linear *= maxAcceleration

  # Output the steering
  steering.angular = 0
  return steering
```

Dynamic Movement

- Flee
  - Opposite from Seek but no need to take of arriving

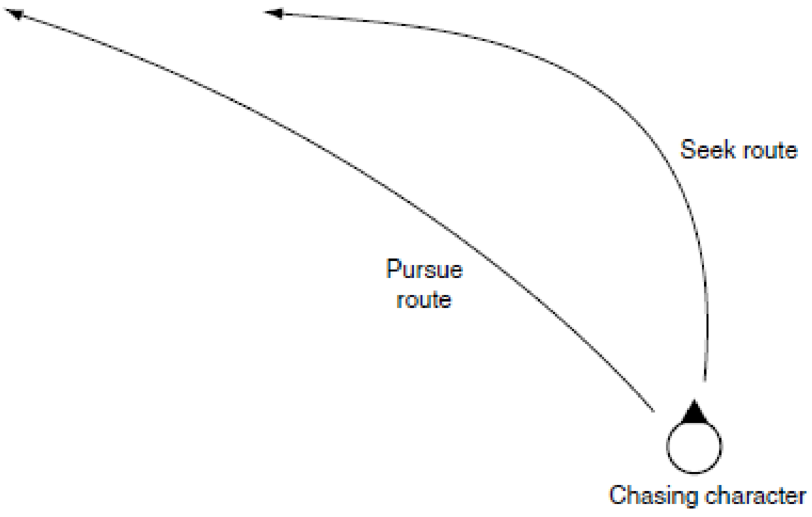
Flee 알고리즘:
앞에서 살펴본 Seek 알고리즘 내에서. velocity vector나 linear (가속도) vector의 부호 (즉 vector의 방향)을 바꾸도록 코드를 적절히 수정을 하면 될 것 임.



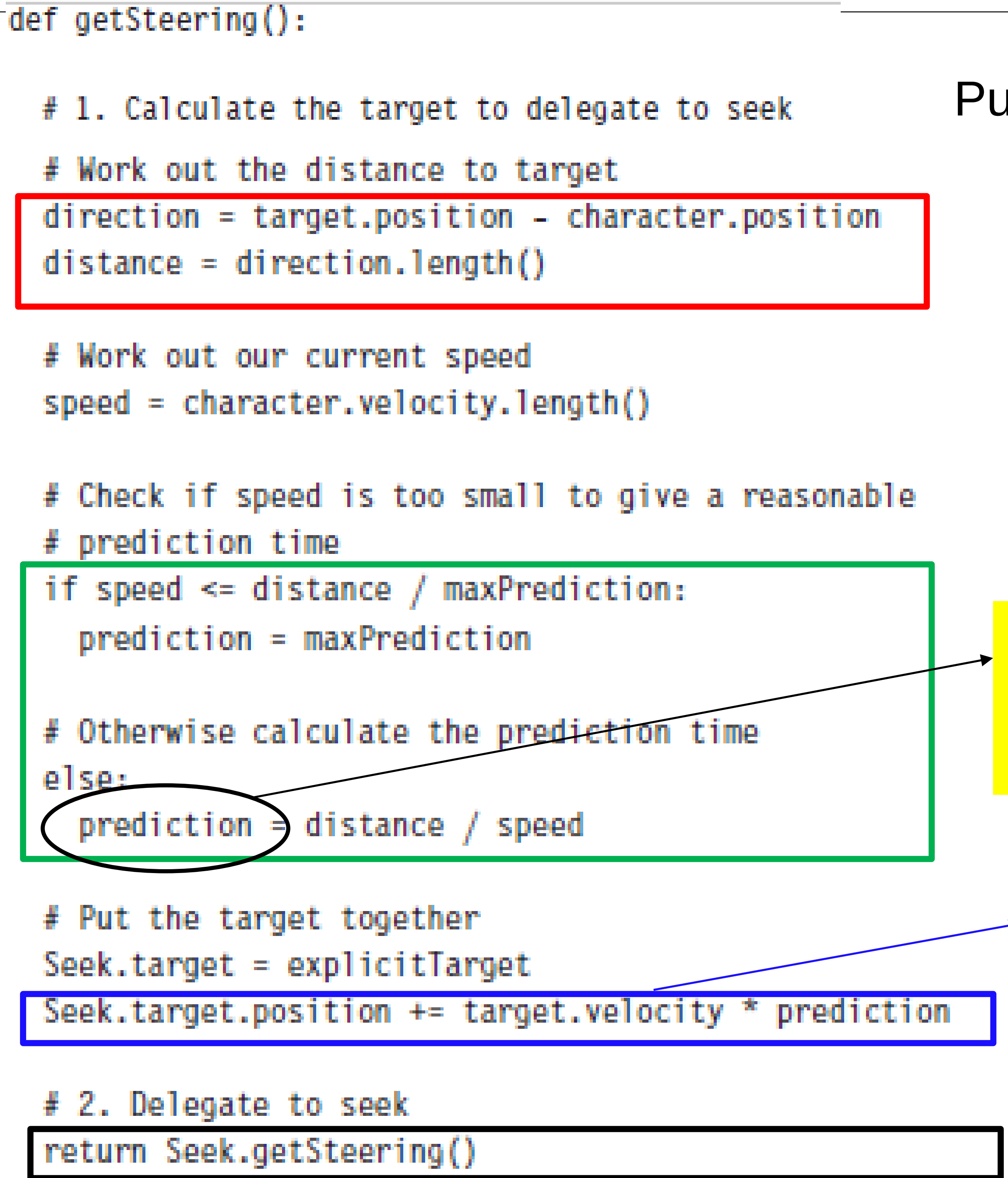
Dynamic Movement

- Delegated Behavior
  - Built from simpler movement components
    - Based on target and character position
  - Pursue
    - Seek, but aiming for future position of target

- Evade



Pursue Algorithm

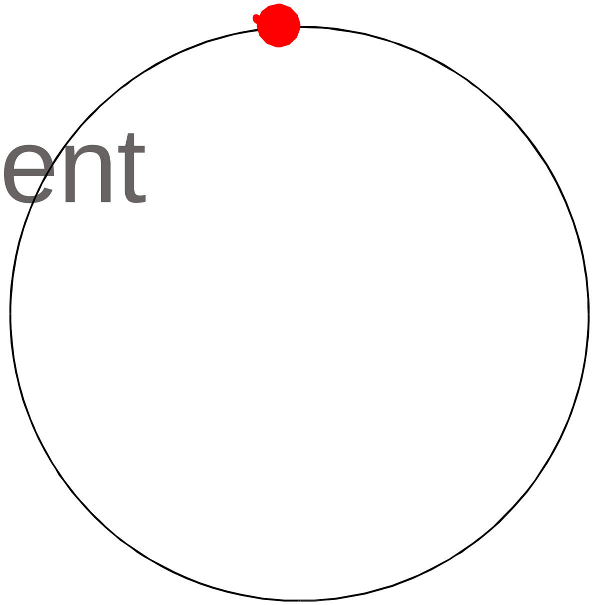


```
def getSteering():

  # 1. Calculate the target to delegate to seek

  # Work out the distance to target
  direction = target.position - character.position
  distance = direction.length()

  # Work out our current speed
  speed = character.velocity.length()

  # Check if speed is too small to give a reasonable
  # prediction time
  if speed <= distance / maxPrediction:
    prediction = maxPrediction

  # Otherwise calculate the prediction time
  else:
    prediction = distance / speed

  # Put the target together
  Seek.target = explicitTarget
  Seek.target.position += target.velocity * prediction

  # 2. Delegate to seek
  return Seek.getSteering()
```

distance 거리를
character의
현 속력으로 진행할 때
걸리는 시간

Target의 미래 위치를
계산

Dynamic Movement

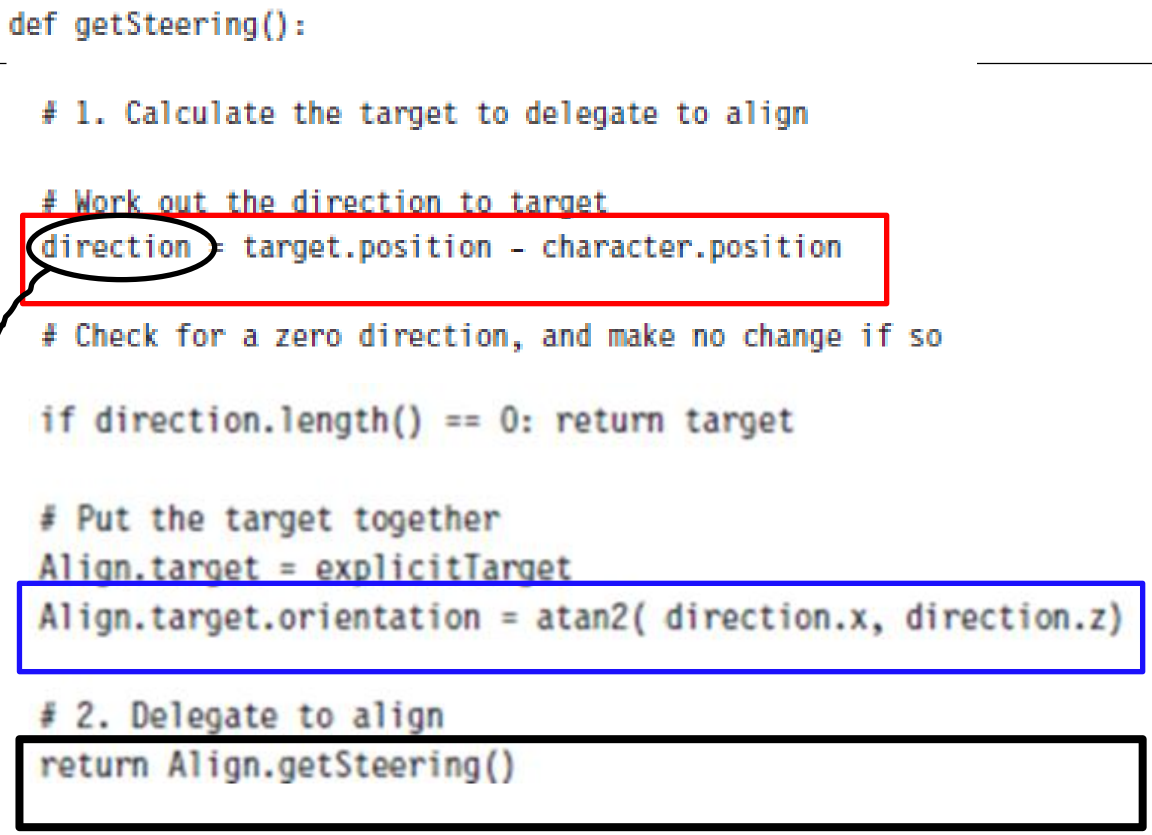


- Delegated Behavior
  - Facing

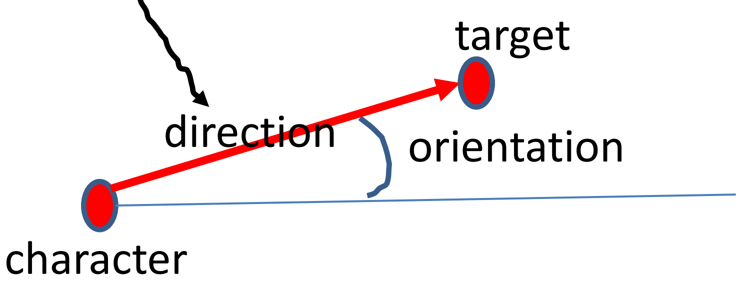
    쳐다보도록 만듦.

    - Makes character look at a target
      - Calculates target orientation
      - Calls align
  - Wandering
    - Steering in random directions gives jerky behavior
    - Instead:
      - Define a target on a (wide) circle around character or far away
      - Move target slowly
      - Call seek

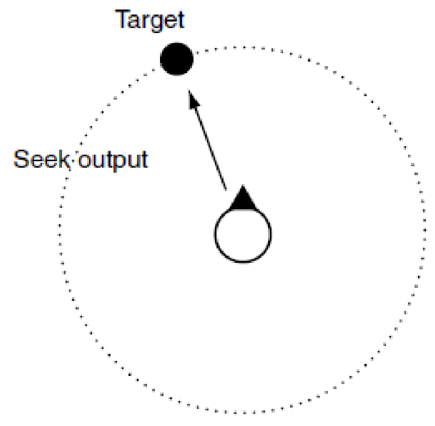
Facing Algorithm



```
def getSteering():

    # 1. Calculate the target to delegate to align

    # Work out the direction to target
    direction = target.position - character.position

    # Check for a zero direction, and make no change if so

    if direction.length() == 0: return target

    # Put the target together
    Align.target = explicitTarget
    Align.target.orientation = atan2( direction.x, direction.z)

    # 2. Delegate to align
    return Align.getSteering()
```

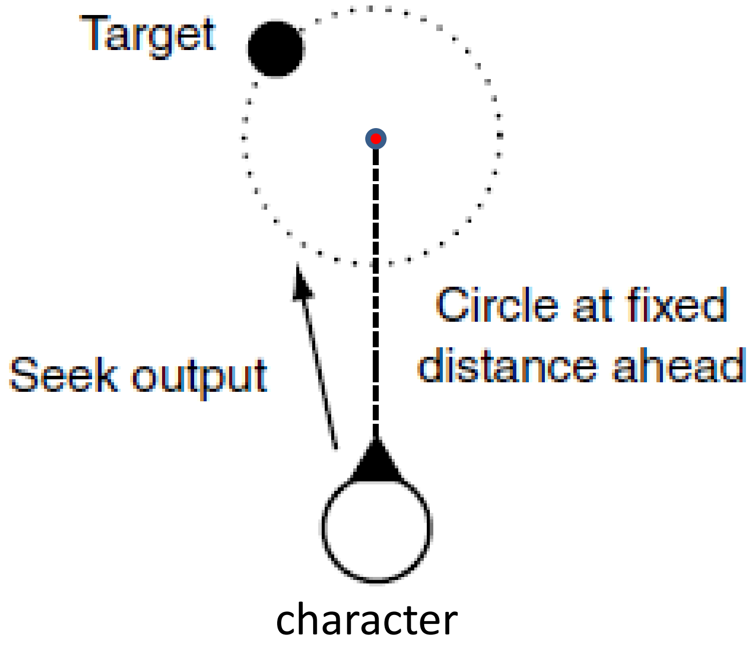


direction은 벡터
orientation은 각도



The kinematic wander as a seek

Target은 화면에 실제 보이 지는 않음



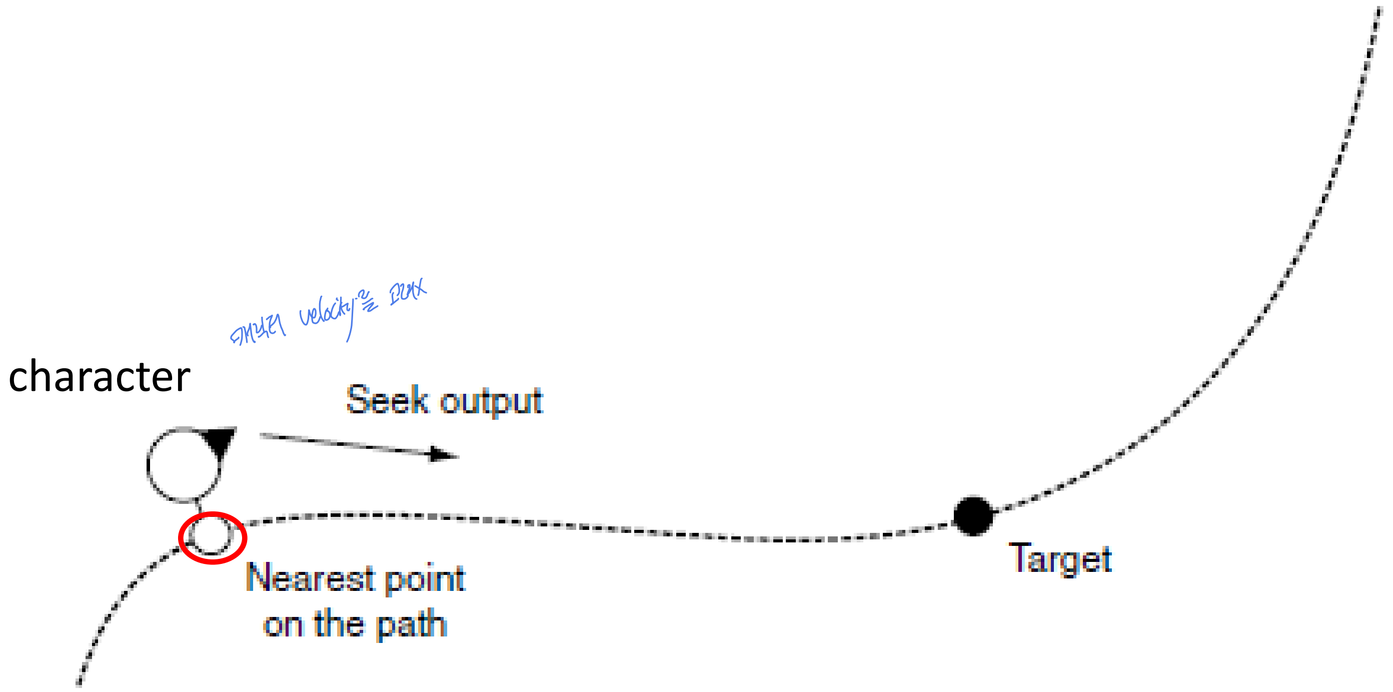
Dynamic Movement

- Path Following
  - Steering behavior not towards a point, but to **a path**
  - Implemented as delegated behavior:
    - Define (moving) target on path
    - Use seek

Dynamic Movement

- Path Following Implementation I
  - Define (moving) target on path
    - Step 1) Find nearest point on path to character
      - (Already difficult)
    - Step 2) Place target ahead of nearest point on path
    - Step 3) Seek

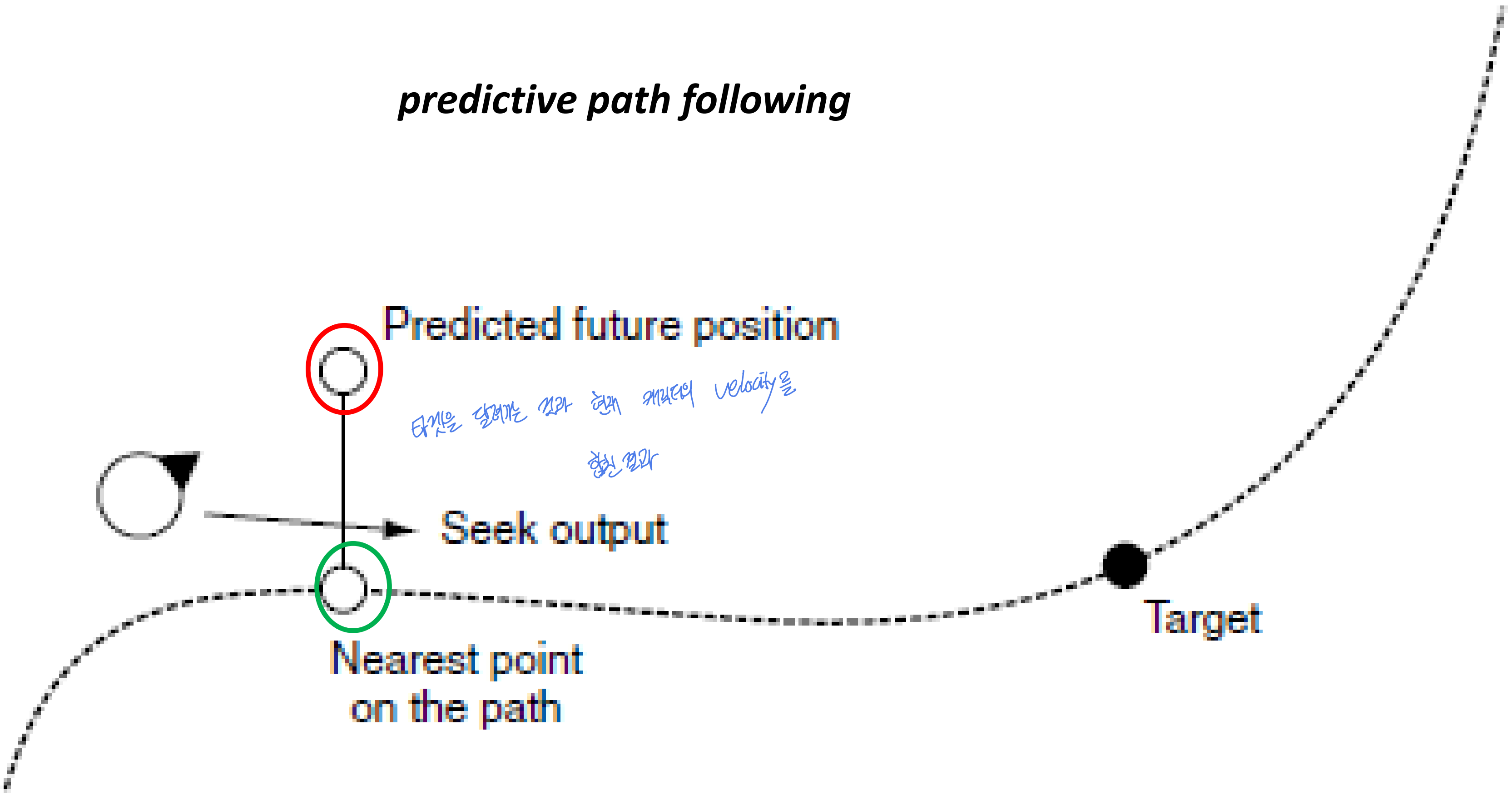
chasing the rabbit



Dynamic Movement

- Path Following Implementation II
  - Define (moving) target on path
    - Step 0) Find **a near future position** of character
    - Step 1) Find nearest point on path to a **near future position**
      - (Already difficult)
    - Step 2) Place target ahead of nearest point on path
    - Step 3) Seek

predictive path following

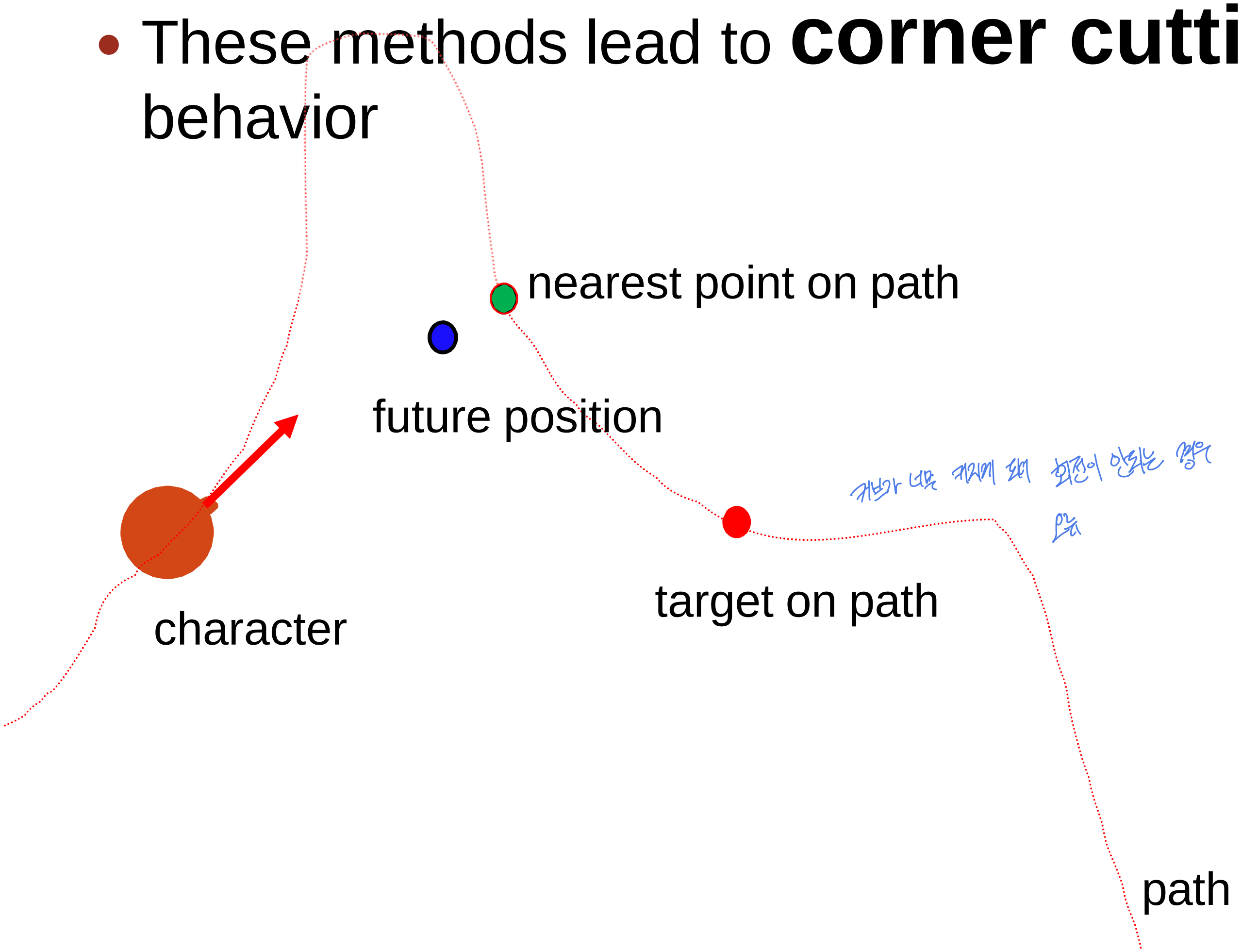


타겟을 달려가는 결과 현재 캐릭터의 velocity를 합친 결과

This implementation can appear smoother for complex paths with
sudden changes of direction.

Dynamic Movement

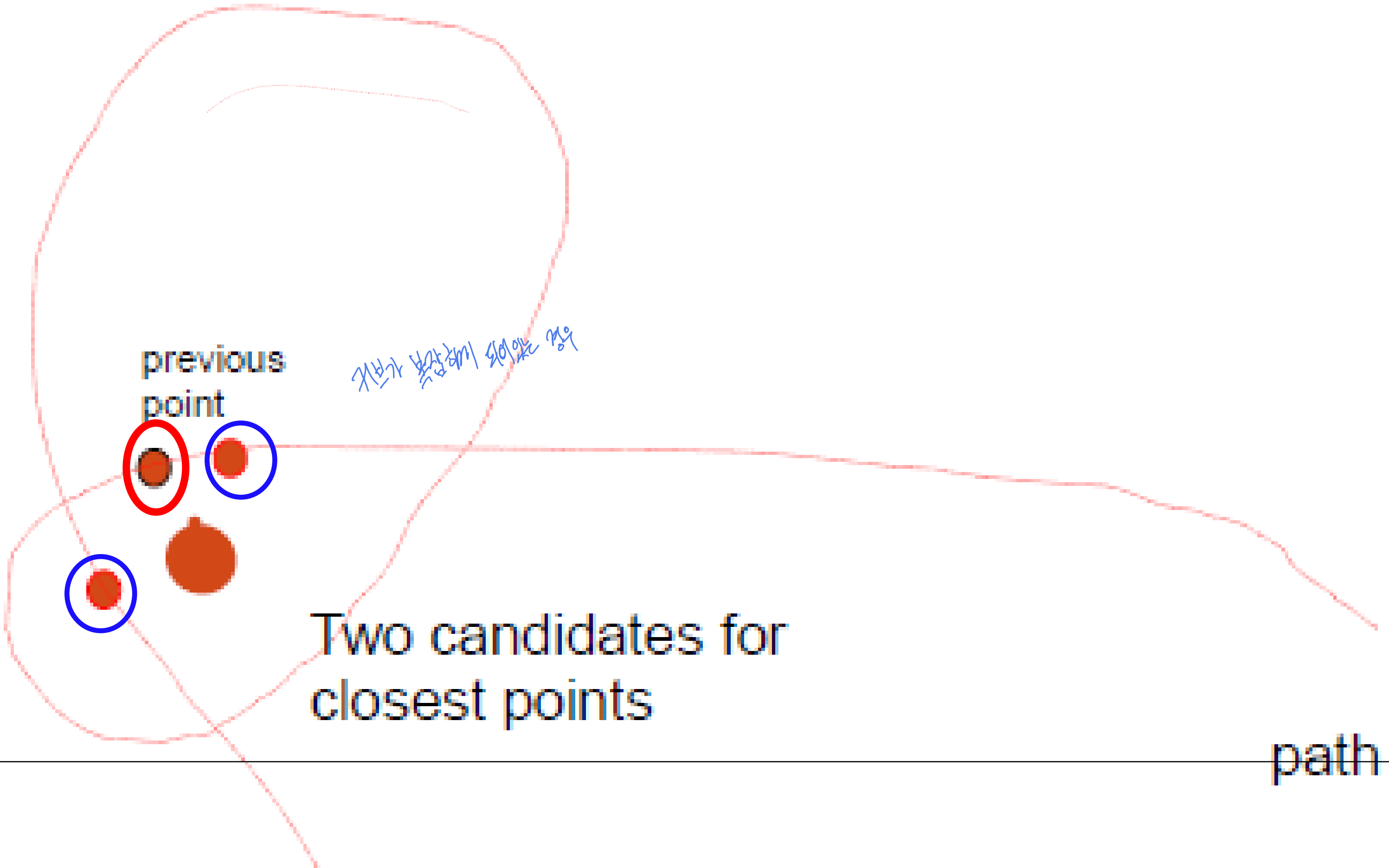
- Path Following
  - These methods lead to **corner cutting** behavior



커브가 너무 커지게 되어 회전이 안되는 경우 있음

Dynamic Movement

- Path Following
  - **Coherence problem** if path crosses itself (textbook p106)


previous point
커브가 복잡하게 되어있는 경우
Two candidates for closest points
path

Dynamic Movement

- Separation
  - Common in **crowd simulations** where many characters head into the same direction
  - Step 1) Identify close characters
  - Step 2) Move away (flee) from them
    - Method 1) Linear separation: Force of separation acceleration is proportional to distance

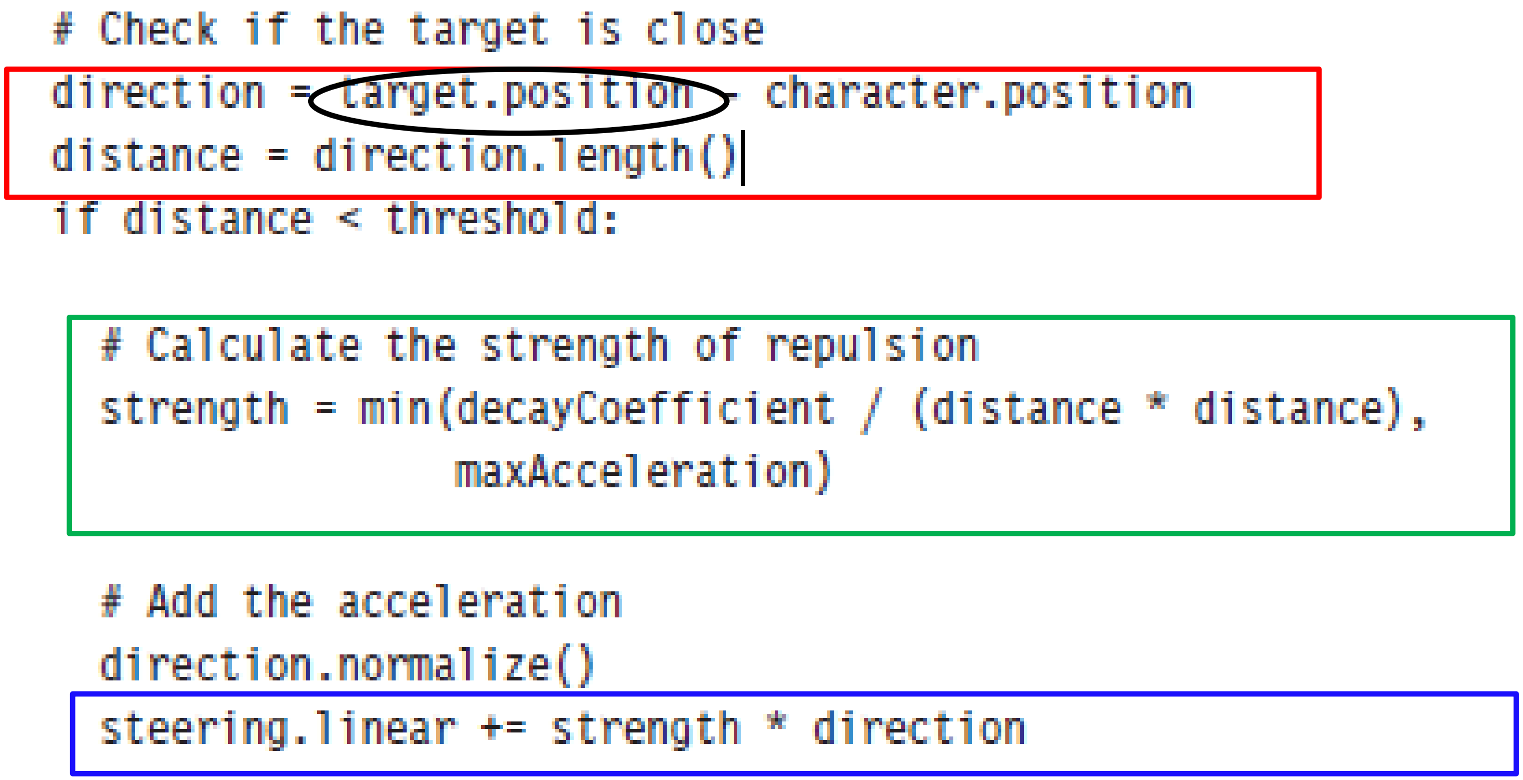
      strength = maxAcceleration * (threshold - distance) / threshold
    - Method-2) Inverse square law separation: Force of separation acceleration is inverse square of distance

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

maxAcceleration : acceleration vector의 (설정한) 최대 크기

threshold : 어떤 character에 대해, 가까운 character를 찾는데 기준이 되는 거리

Method-2를 이용한 separation algorithm



```
# Check if the target is close
direction = target.position - character.position
distance = direction.length()
if distance < threshold:

  # Calculate the strength of repulsion
  strength = min(decayCoefficient / (distance * distance),
                 maxAcceleration)

  # Add the acceleration
  direction.normalize()
  steering.linear += strength * direction
```

음성강의 종료