CS380
Artificial Intelligence for Games

Fuzzy Logic

Fuzzy Logic Applications

- Video games and simulation applications
- Climate control
- Quality of service
- Battery chargers
- Path finding
- Robotics

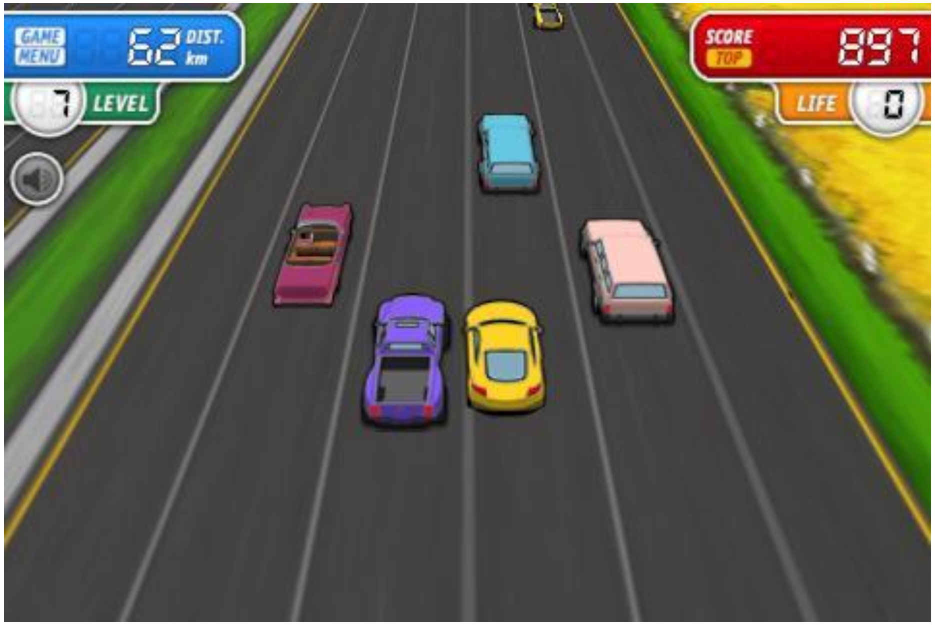
- 3D-Camera positioning and orientation
- Self-focusing camera
- Car breaking system
- Car breaks heating
- Camera white balance control
- Trains control
- Air conditioning and heating systems

What is Fuzzy Logic?

- Fuzzy logic is based on the observation that people make decisions based on imprecise and non-numerical information
  - IF weather is good AND distance is short THEN take a walk
  - How to make a computer to calculate “weather is good” and “distance is short”?
- Why not to use Boolean Logic with 0 and 1 only?

Example: 2D Racing Game

- Player car (PSpeed, ...) controlled by a human player
- Opponent cars (OSpeed, …) controlled by a computer



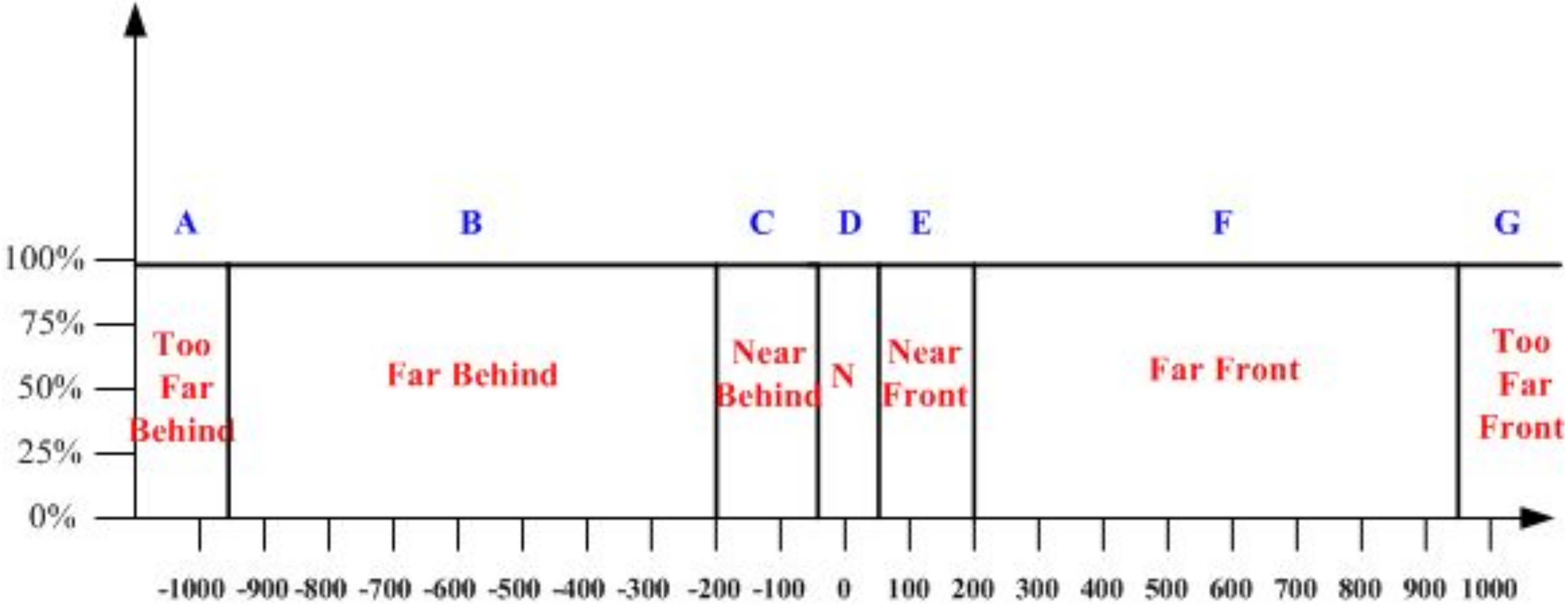
Non-Fuzzy Sets

Theory of Sets (Non-Fuzzy)

A **set** is a collection/range of related things or values:

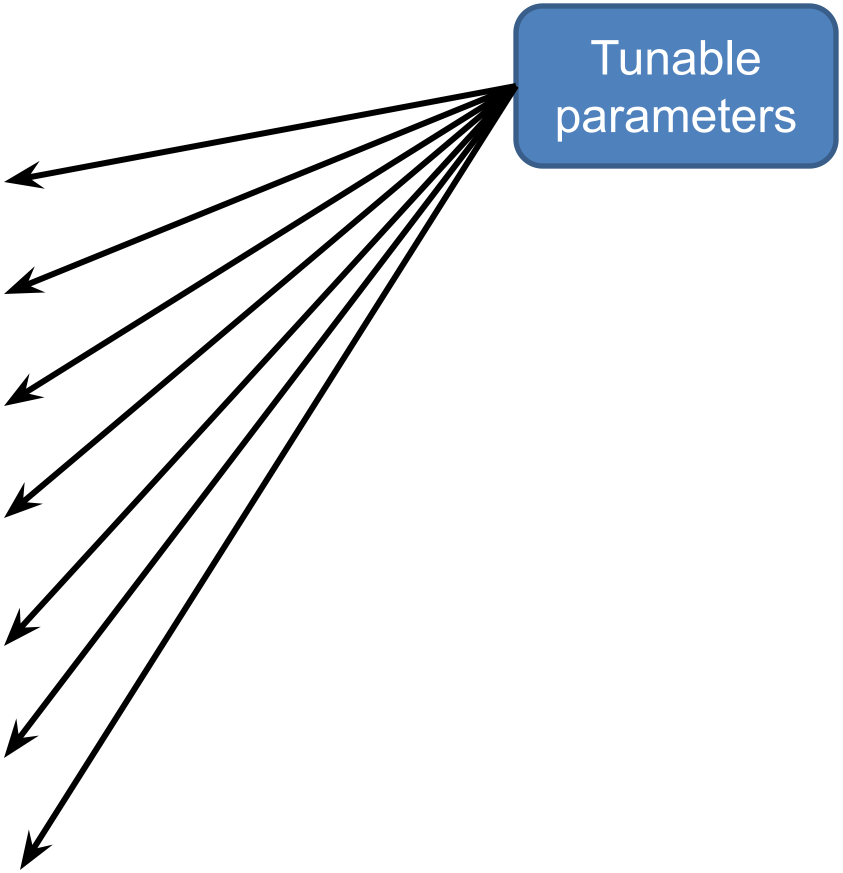
- Set1: Opponent car is **too far behind** player’s car
- Set2: Opponent car is **far behind** player’s car
- Set3: Opponent car is **near behind** player’s car
- Set4: Opponent car is **near** player’s car
- Set5: Opponent car is **near in front** of player’s car
- Set6: Opponent car is **far in front** of player’s car
- Set7: Opponent car is **too far in front** of player’s car

Theory of Sets (Non-Fuzzy)



Theory of Sets (Non-Fuzzy)

```
if (player is in front)
    distance*=-1;
if (distance < -950)
    OSpeed = PSpeed * 2.0
else if (distance < -200)
    OSpeed = PSpeed * 1.5
else if (distance < -50)
    OSpeed = PSpeed * 1.2
else if (distance < 50)
    OSpeed = PSpeed * 1.0
else if (distance < 200)
    OSpeed = PSpeed / 1.2
else if (distance < 950)
    OSpeed = PSpeed / 1.5
else // > 950
    OSpeed = PSpeed / 2.0
```



Tunable parameters

Fuzzy Sets

Fuzzy Sets

- A fuzzy set is a function $f(x)$ with domain $[0, 1]$, whose value denotes **Degree Of Membership (DOM)**, with 0 denoting that x is not a member and 1 denoting that x is definitely a member
- Car Example:

$f_{FAST}(30) = 0$

$f_{FAST}(200) = 0.5$

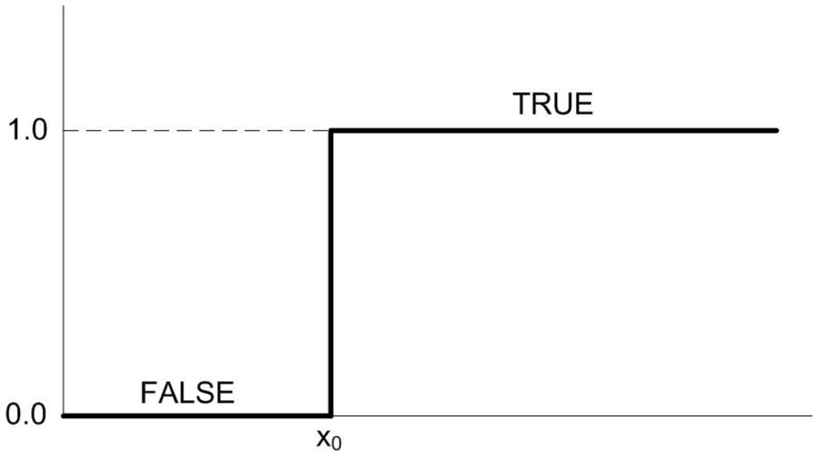
$f_{FAST}(447) = 1$

$f_{FAST\text{-}CAR}$(Benz Velo) = f_{FAST}(TOP-SPEED(Benz Velo)) = 0

$f_{FAST\text{-}CAR}$(Jaguar XK120) = f_{FAST}(Jaguar XK120)) = 0.5

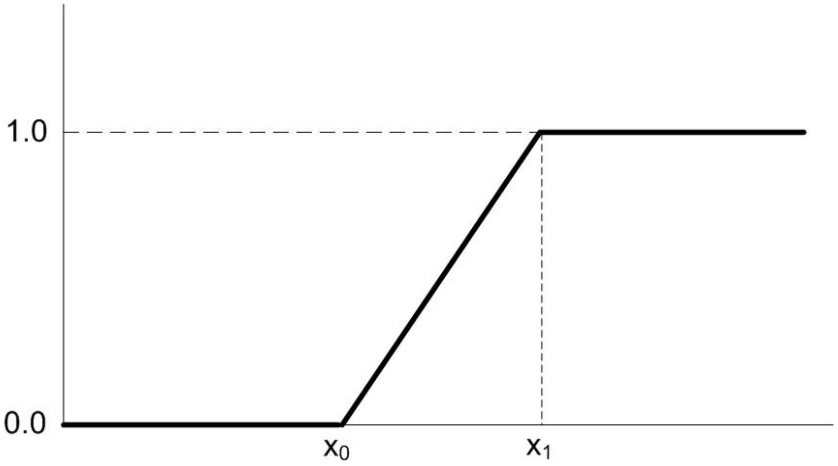
$f_{FAST\text{-}CAR}$(Agera RS) = f_{FAST}(TOP-SPEED(Agera RS)) = 1

Membership Function



Boolean logic membership function

$$f(x) = \begin{cases} 0 & \text{if } x \leq x_0 \\ 1 & \text{if } x > x_0 \end{cases}$$



Grade membership function

$$f(x) = \begin{cases} 0 & \text{if } x \leq x_0 \\ \frac{x-x_0}{x_1-x_0} & \text{if } x_0 < x < x_1 \\ 1 & \text{if } x \geq x_1 \end{cases}$$

To Find the Right OSpeed (Fuzzy)

1. Define the Fuzzy Sets (similar to the previous sets)

2. Find the current distance between opponent’s car and player’s car (as before)

3. Call the Fuzzy Set member function to get the DOM for each set

4. Use the value found in step 3 in a Fuzzy Control Equation to find the speed needed to be applied to the opponent car

To Find the Right OSpeed (Fuzzy)

1. Define the Fuzzy Sets (similar to the previous sets)

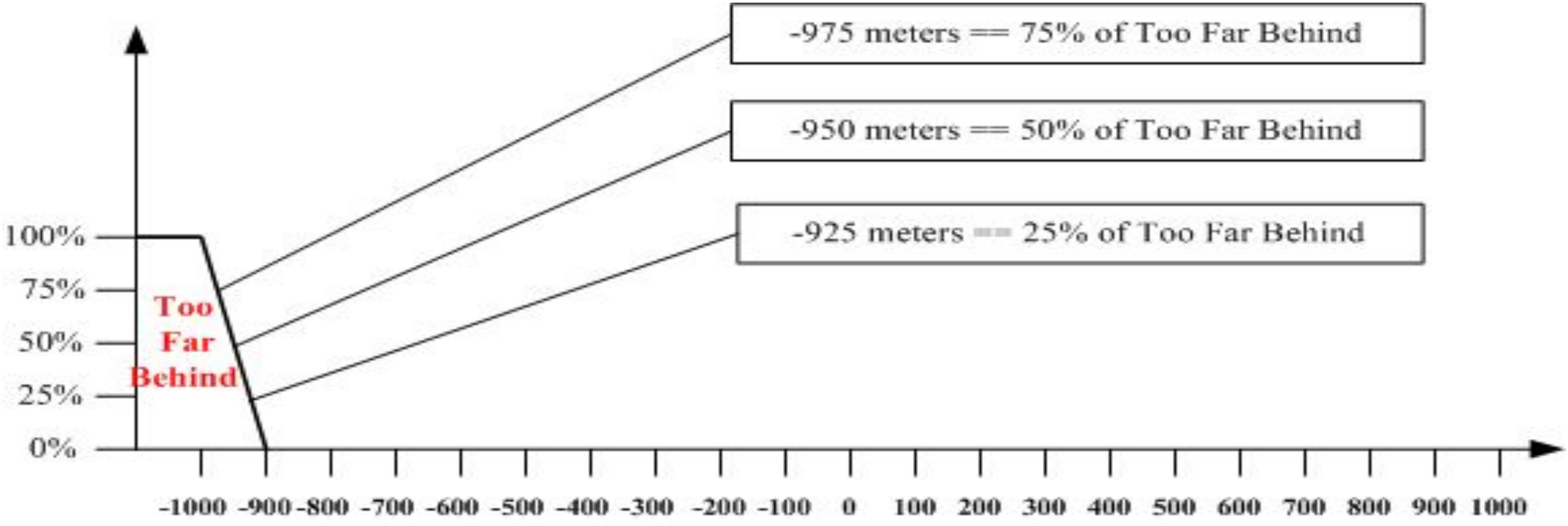
2. Find the current distance between opponent’s car and player’s car (as before)

3. Call the Fuzzy Set member function to get the DOM for each set

4. Use the value found in step 3 in a Fuzzy Control Equation to find the speed needed to be applied to the opponent car

2D Racing Game: Fuzzy Set

- Set1: Opponent car is too far behind player's car.



-975 meters == 75% of Too Far Behind

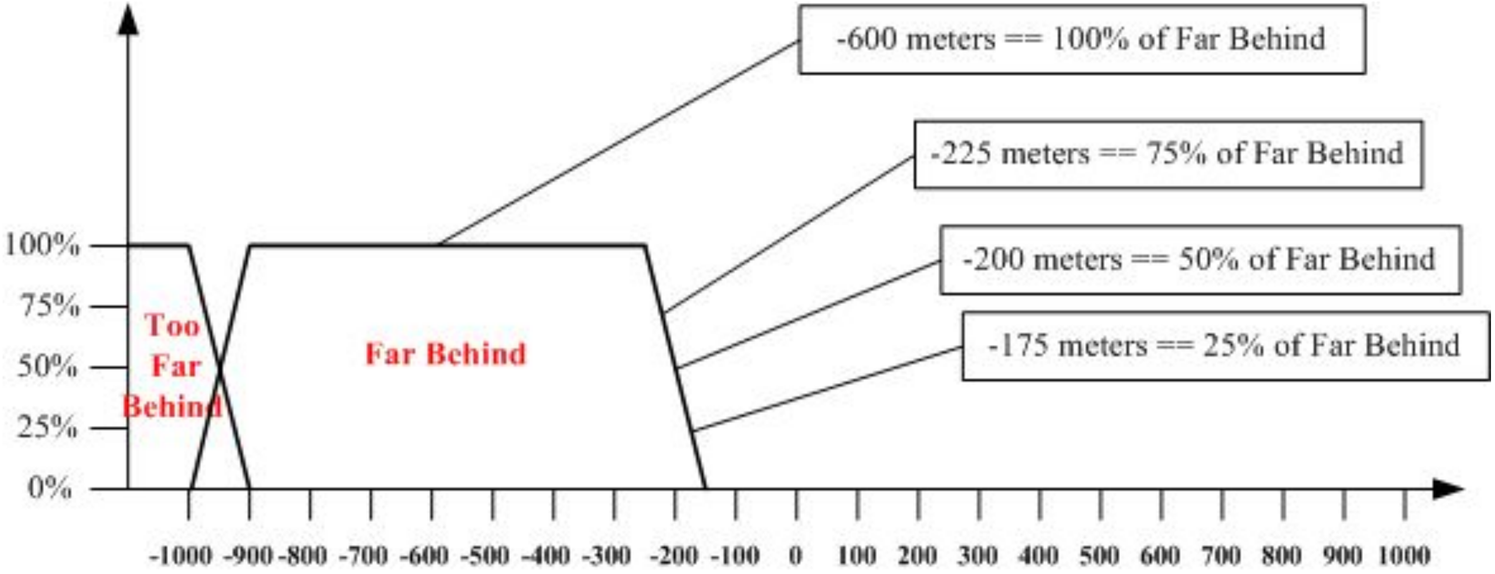
-950 meters == 50% of Too Far Behind

-925 meters == 25% of Too Far Behind

$$f(d) = \begin{cases} 1 & \text{if } d < -1000 \\ \frac{-900-d}{100} & \text{if } -1000 \leq d \leq -900 \\ 0 & \text{if } d > -900 \end{cases}$$

2D Racing Game: Fuzzy Set

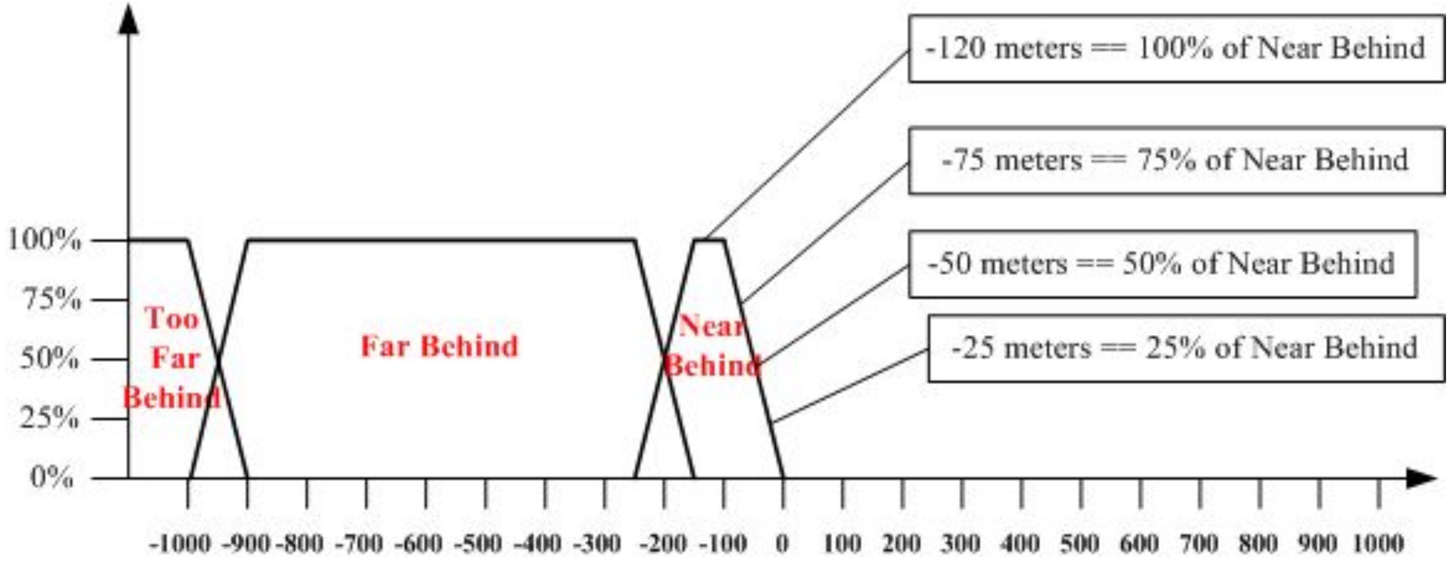
- Set2: Opponent car is far behind player's car.



$$f(d) = \begin{cases} \frac{d+1000}{100} & \text{if } -1000 \leq d \leq -900 \\ 1 & \text{if } -900 < d < -250 \\ -\frac{d+150}{100} & \text{if } -250 \leq d \leq -150 \\ 0 & \text{if } d > -150 \text{ or } d < -1000 \end{cases}$$

2D Racing Game: Fuzzy Set

- Set3: Opponent car is near behind player's car



-120 meters == 100% of Near Behind

-75 meters == 75% of Near Behind

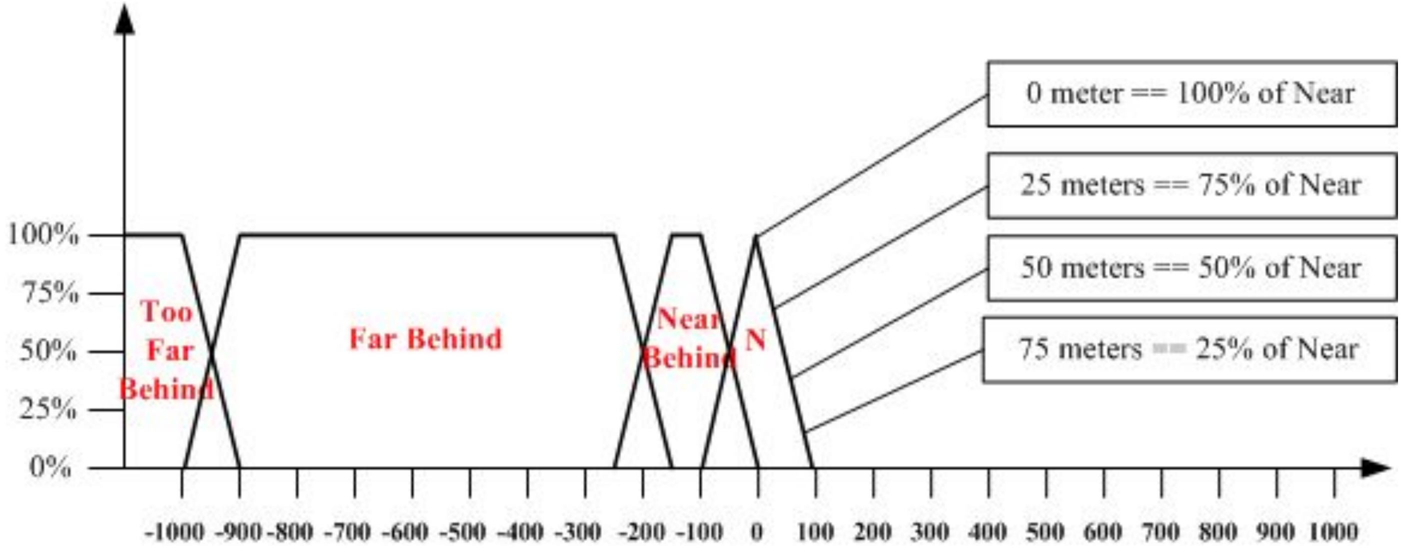
-50 meters == 50% of Near Behind

-25 meters == 25% of Near Behind

$$f(d) = \begin{cases} \frac{d+250}{100} & \text{if } -250 \leq d \leq -150 \\ 1 & \text{if } -150 < d < -100 \\ \frac{-d}{100} & \text{if } -100 \leq d \leq 0 \\ 0 & \text{if } d < -250 \text{ or } d > 0 \end{cases}$$

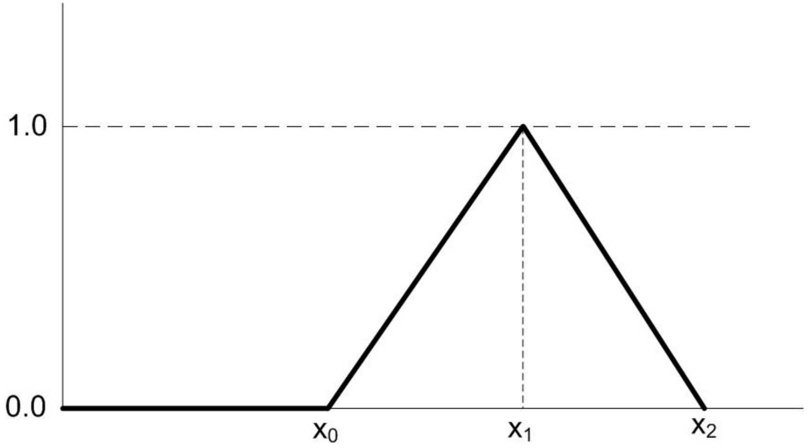
2D Racing Game: Fuzzy Set

- Set4: Opponent car is near player’s car.

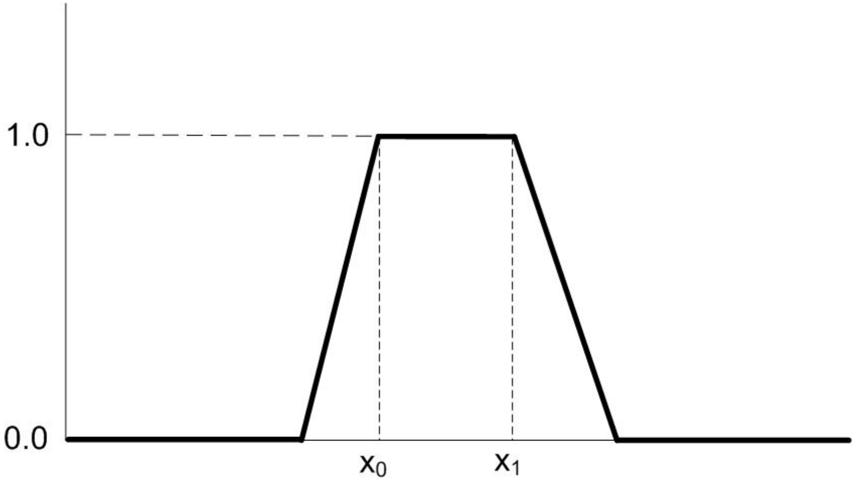


$$f(d) = \begin{cases} \frac{d+100}{100} & \text{if } -100 \leq d \leq 0 \\ 1 & \text{if } d = 0 \\ -\frac{d-100}{100} & \text{if } 0 \leq d \leq 100 \\ 0 & \text{if } d < -100 \text{ or } d > 100 \end{cases}$$

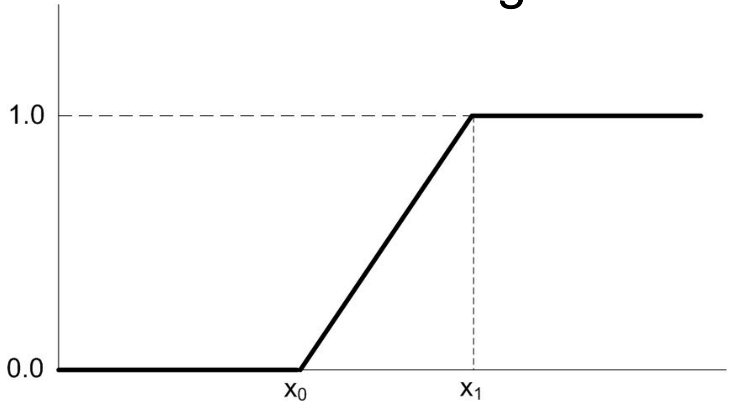
Common Shapes of Fuzzy Set



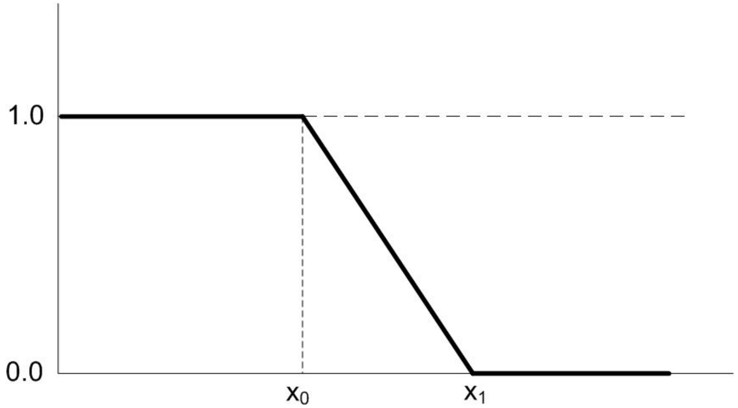
Triangle



Trapezoid

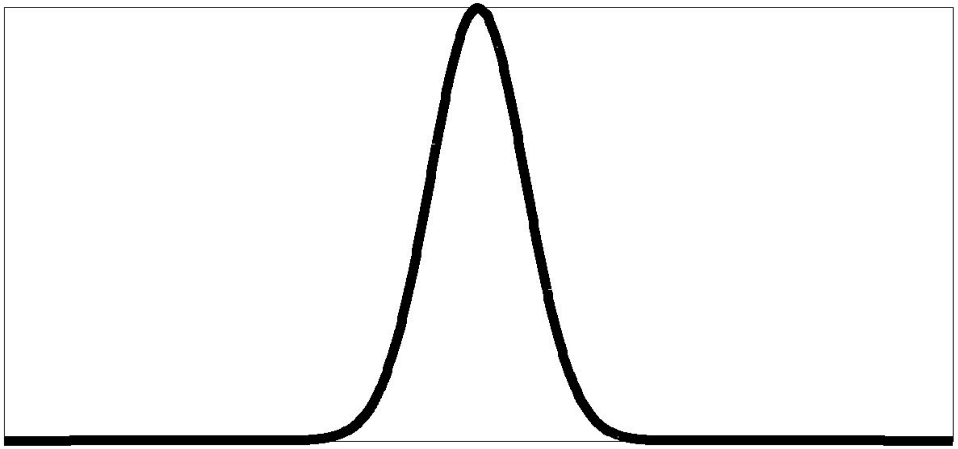


Right Shoulder



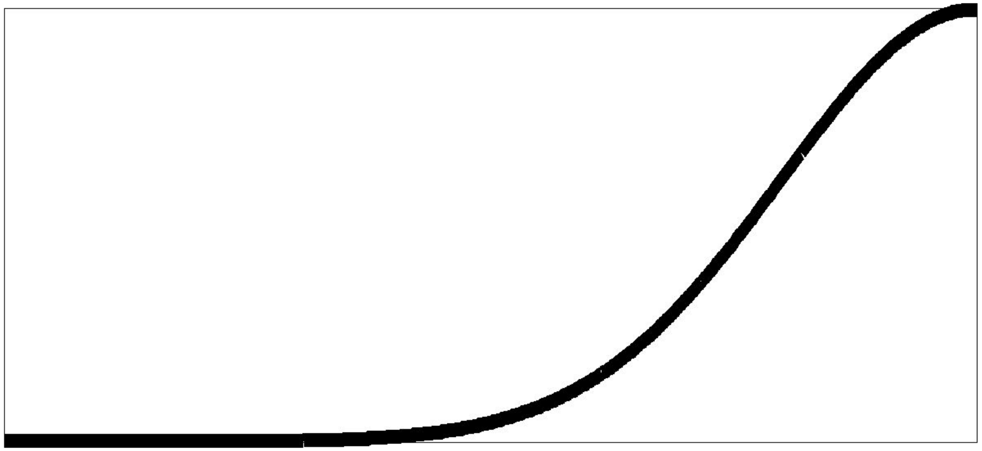
Left Shoulder

Other Membership Functions



Distribution graph

$$f(x) = a \exp\left(-\frac{(x-b)^2}{2c^2}\right)$$

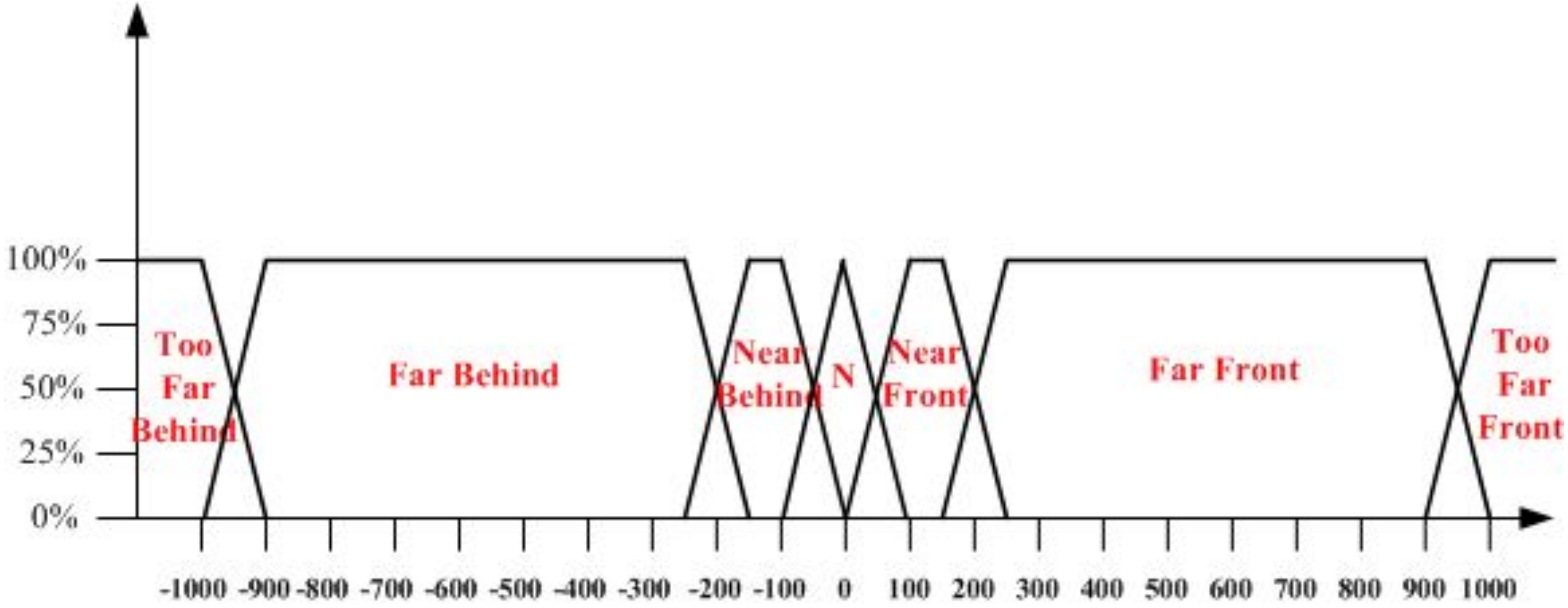


Logistic curve

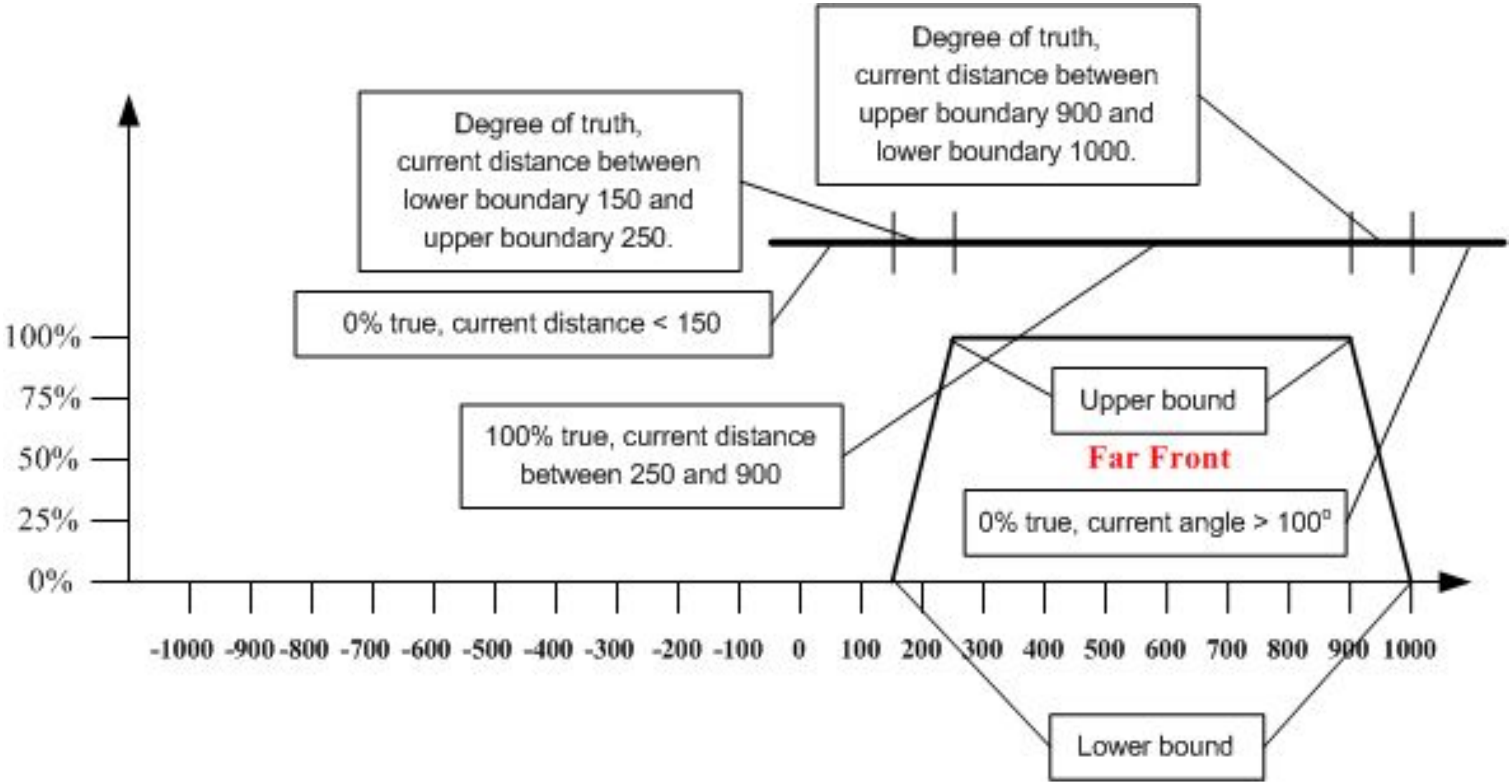
$$f(x) = \frac{L}{1 + \mathrm{e}^{-k(x-x_0)}}$$

Fuzzy Variables

- A fuzzy variable is a mapping from an abstract space (sets) onto the real line
- Example: Distance for 2D Racing Game



Fuzzy Variables



Degree of truth, current distance between lower boundary 150 and upper boundary 250.

Degree of truth, current distance between upper boundary 900 and lower boundary 1000.

0% true, current distance < 150

100% true, current distance between 250 and 900

Upper bound

Far Front

0% true, current angle > 100°

Lower bound

Fuzzy Variables

- Setting up collections of fuzzy sets for a given input variable is a matter of judgment and trial and error.
  - Try using more or less fuzzy sets
  - Try different shapes of each fuzzy set

- Rule of thumb: Each set should overlap its neighbor by around 25%

To Find the Right OSpeed (Fuzzy)

1. Define the Fuzzy Sets (similar to the previous sets)

2. Find the current distance between opponent’s car and player’s car (as before)

3. Call the Fuzzy Set member function to get the DOM for each set

4. Use the value found in step 3 in a Fuzzy Control Equation to find the speed needed to be applied to the opponent car

Distance Computation

```
distance =
  GetDistance(player.position,
  opponent.position);

  if (opponent is behind of player)
    distance *= -1;
```

To Find the Right OSpeed (Fuzzy)

1. Define the Fuzzy Sets (similar to the previous sets)

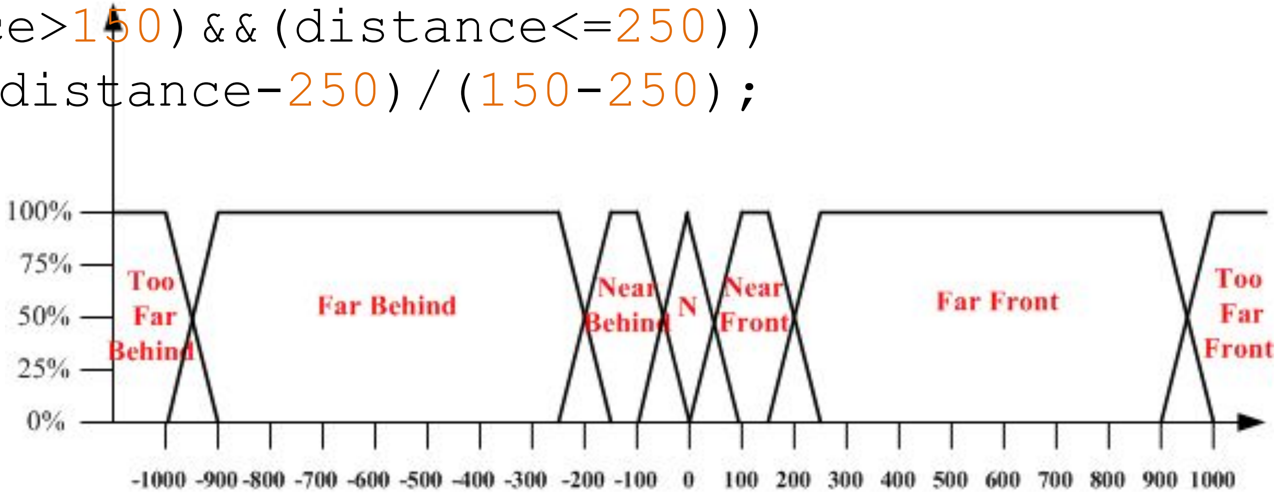
2. Find the current distance between opponent's car and player's car (as before)

3. Call the Fuzzy Set member function to get the DOM for each set

4. Use the value found in step3 in a Fuzzy Control Equation to find the speed needed to be applied to the opponent car

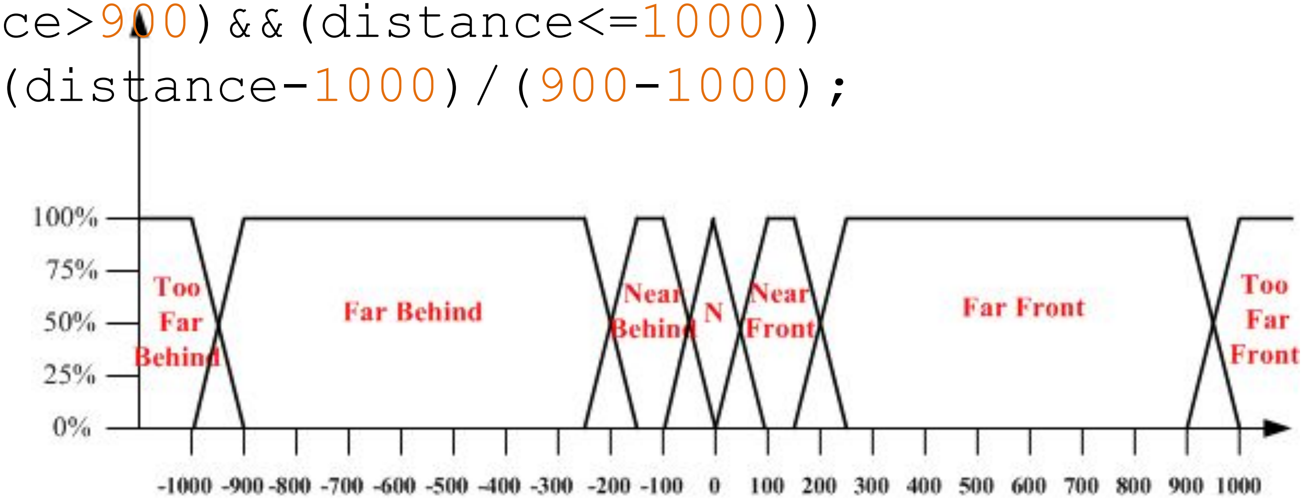
2D Racing Game: Member Function

```
float DOM_NearFront(float distance)
{
    if (distance<=0)
        return 0;
   if ((distance>0)&&(distance<=100))
        return (distance-0)/(100-0)
   if ((distance>100)&&(distance<=150))
        return 1;
   if ((distance>150)&&(distance<=250))
        return (distance-250)/(150-250);
   return 0;
}
```



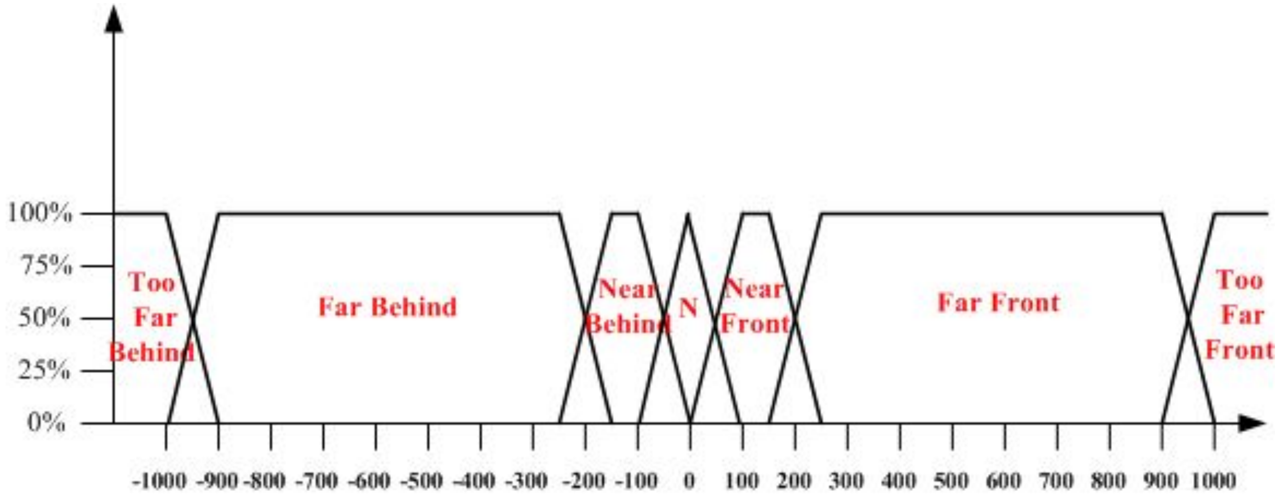
2D Racing Game: Member Function

```
float DOM_FarFront(float distance)
{
    if (distance<=150)
        return 0;
   if ((distance>150)&&(distance<=250))
        return (distance-150)/(250-150);
   if ((distance>250)&&(distance<=900))
        return 1;
   if ((distance>900)&&(distance<=1000))
        return (distance-1000)/(900-1000);
   return 0;
}
```



2D Racing Game: Member Function

```
float DOM_TooFarFront(float distance)
{
   if (distance>1000)
       return 1;
   if (distance<900)
       return 0;
   return (distance-900)/(1000-900);
}
```



To Find the Right OSpeed (Fuzzy)

1. Define the Fuzzy Sets (similar to the previous sets)

2. Find the current distance between opponent’s car and player’s car (as before)

3. Call the Fuzzy Set member function to get the DOM for each set

4. Use the value found in step 3 in a **Fuzzy Control Equation** to find the speed needed to be applied to the opponent car

Fuzzy Control Equations

Traditional Boolean Logic Operators

x	y	x AND y	x OR y
0	0		
0	1		
1	0		
1	1		

x	NOT *x*
0	
1	

Fuzzy Logic Operators

Boolean Logic	Fuzzy Logic
NOT x	1 – x
x AND y	min(x, y)
x OR y	max(x, y)

Boolean Logic V.S. Fuzzy Logic

- Boolean logic:
  - If (0 or 1) then result = 1

    // the result is 100% executed
  - If (0 and 1) then result = 0

    // the result is 0% executed

- Fuzzy logic:
  - If (0.2 or 0.77) then result = 0.77

    // the result is 77% executed
  - If(0.2 and 0.77) then result = 0.2

    // the result is 20% executed

Fuzzy Logic Operators

Examples:

- FAST-CAR(Porsche-944) = 0.9
- **NOT** FAST-CAR(Porsche-944) =
  1 – FAST-CAR(Porsche-944) = 0.1
- PRETENTIOUS-CAR(Porsche-944) = 0.7
- FAST-CAR(Porsche-944) **AND**
  PRETENTIOUS-CAR(Porsche-944) = 0.7
- FAST-CAR(Porsche-944) **AND**
  ¬FAST-CAR(Porsche-944) = 0.1
- FAST-CAR(Porsche-944) **OR**
  ¬FAST-CAR(Porsche-944) = 0.9

Fuzzy Rules

- Rules are used within Fuzzy Logic systems to infer an output based on input variables
- Form:
  - IF <antecedent> THEN <consequent>

Fuzzy Rules

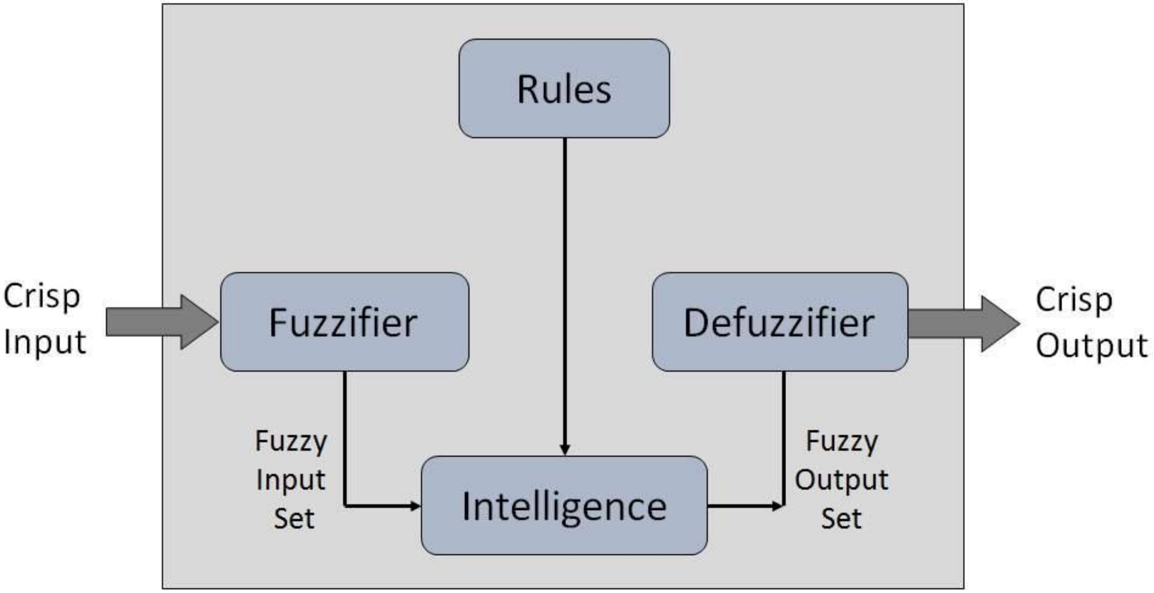
- Enemy AI character:
  - Strength = { weak, normal, strong }
  - Distance = { close, far }
  - State = { hide, wander, attack }
  - (Strength, Distance) => State
- Exact Input:
  - **Strength:** a value
  - **Distance:** a value
- Fuzzy Input:
  - **Strength**: weak, normal, and strong degree of truth.
  - **Distance**: close and far degree of truth

Fuzzy Rules

- IF close OR weak THEN hide
- IF close AND (normal OR strong) THEN attack
- IF far AND (weak OR normal) THEN wander
- IF far AND strong THEN attack

Fuzzy Inference System

- It fuzzifies some variable's input and defuzzifies the result to give an exact variable output
  - Fuzzification
  - Rules Evaluation
  - Aggregation
  - Defuzzification



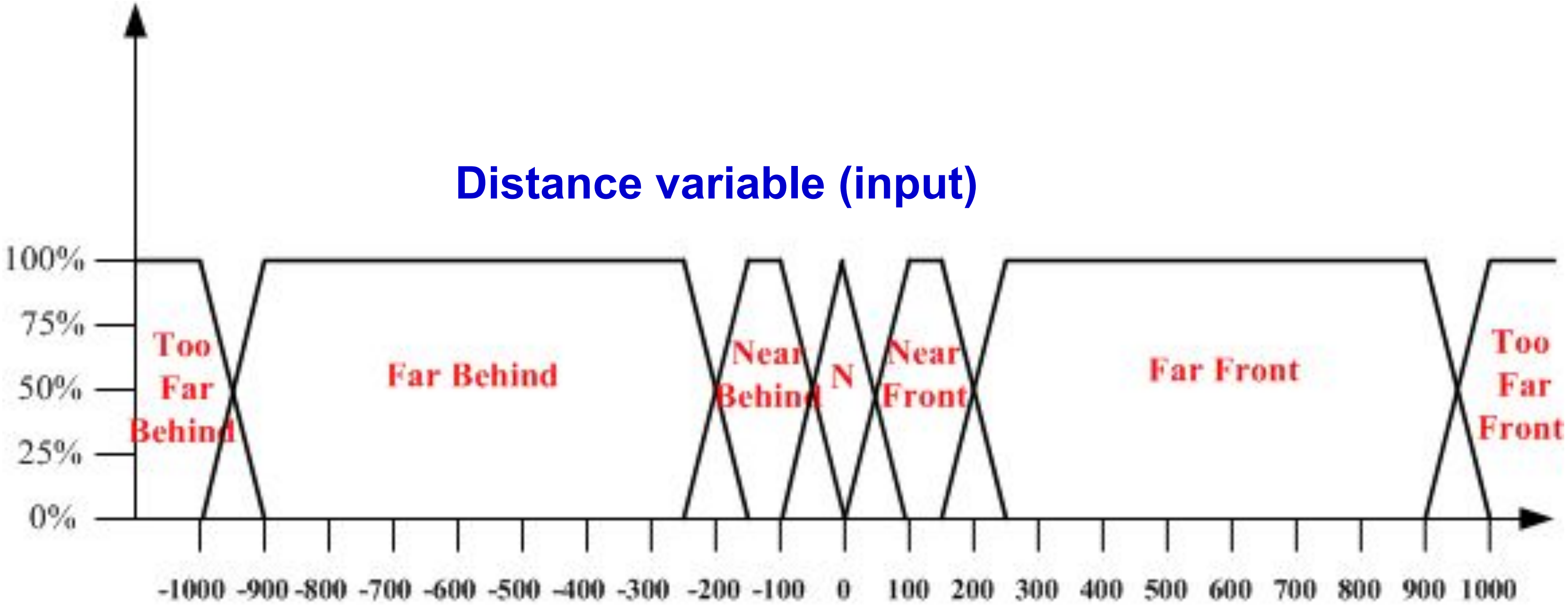
Inference: Fuzzification

- For each input value of a different variable V_1, V_2,...,V_n we can get fuzzy inputs: S_1, S_2,…,S_n, where S_i is the number of the sets in V_i.

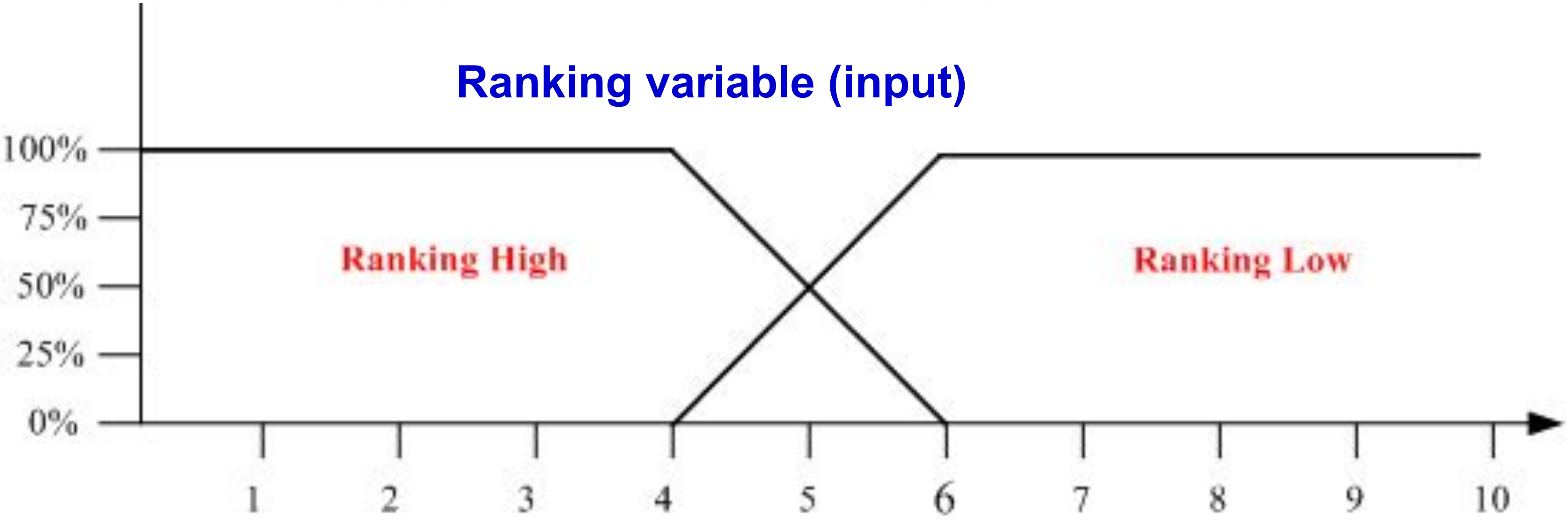
- 2D Racing Game
  - Objective: update the speed of the opponent car: OSpeed
  - Inputs: relative distance, ranking position of the player

Fuzzification

Distance variable (input)

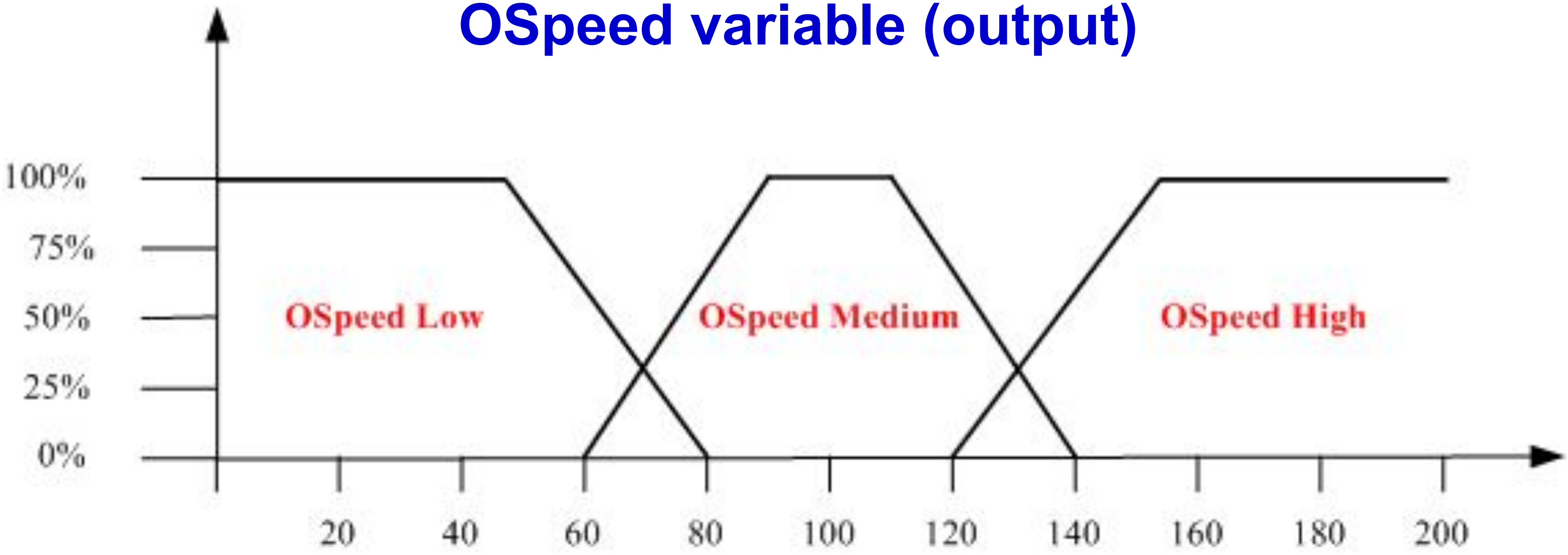


Ranking variable (input)



Fuzzification

OSpeed variable (output)



Rules Evaluation

- In this step, we will set some rules (conditions) to set the opponent’s speed OSpeed

Rules Construction

1. **IF** (Far Front **OR** Too Far Front) **AND** Ranking Low **THEN** OSpeed Low

2. **IF** Near **OR** Near Behind **OR** Near Front **THEN** OSpeed Medium

3. **IF** (Far Behind **OR** Too Far Behind) **AND** Ranking High **THEN** OSpeed High

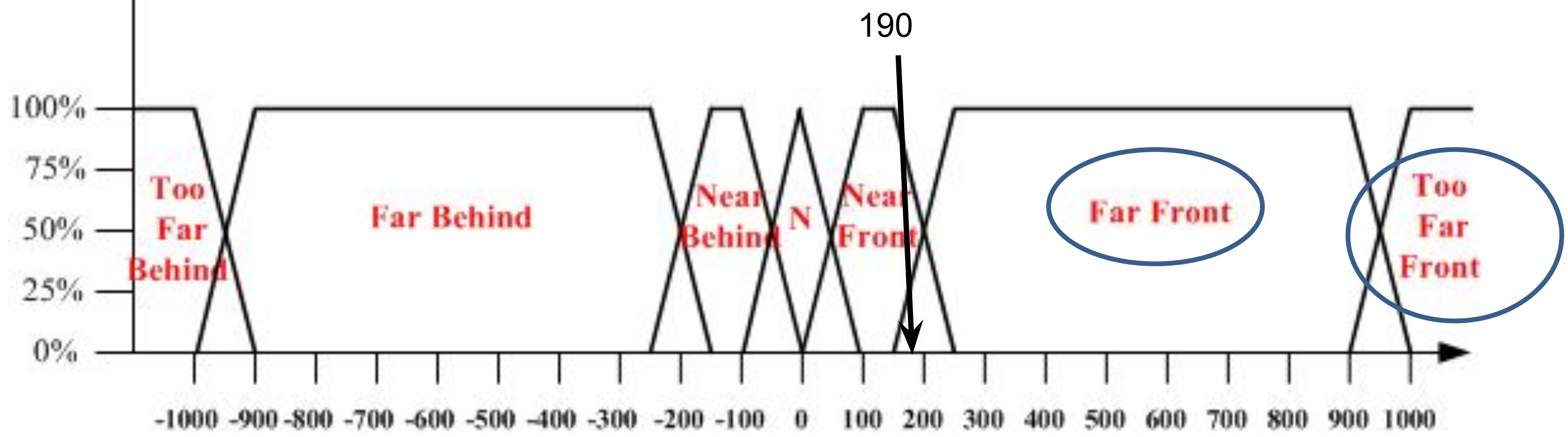
Rules Evaluation: 1st Rule

- Assume that the current input of the game is:
  - distance = 190
  - ranking = 5

- **IF** (Far Front **OR** Too Far Front) **AND** Ranking Low **THEN** OSpeed Low

- DOM(OSpeed Low) =
  (DOM(Far Front) OR DOM(Too Far Front))
  AND DOM(Ranking Low) =
  (0.4 OR 0.0) AND 0.5 = 0.4

Rules Evaluation: 1st Rule



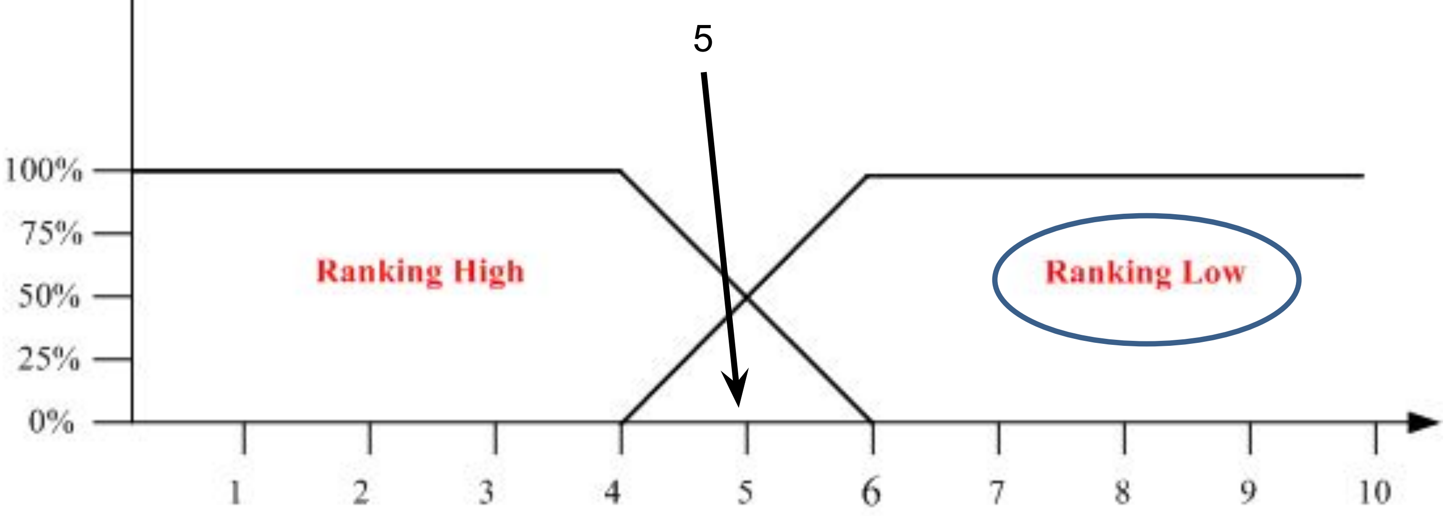
```
float DOM_FarFront(float distance){
   if (distance<=150)
       return 0;
   if ((distance>150)&&(distance<=250))
       return (distance–150)/(250–150);
   if ((distance>250)&&(distance<=900))
       return 1;
   if ((distance>900)&&(distance<=1000))
       return (distance-1000)/(900-1000);
   return 0;
}
```

```
float DOM _TooFarFront(float distance){
   if (distance>1000)
       return 1;
   if (distance<900)
       return 0;
   return (distance-900)/(1000–900);
}
```

DOM_FarFront(190) = 0.4

DOM_TooFarFront(190) = 0

Rules Evaluation: 1st Rule



```
float DOM_RankingLow (int rank) {
  if (rank <= 4)
    return 0;
  if ((rank>4)&&(rank<=6))
    return (rank-4)/2
  if (rank>6)
    return 1;
}
```

DOM(Ranking Low) = 0.5

Rules Evaluation: 1st Rule

- Assume that the current input of the game is:
  - distance = 190
  - ranking = 5

- **IF** (Far Front **OR** Too Far Front) **AND** Ranking Low **THEN** OSpeed Low

- DOM(OSpeed Low) =
  (DOM(Far Front) OR DOM(Too Far Front))
  AND DOM(Ranking Low) =
  (0.4 OR 0.0) AND 0.5 = 0.4

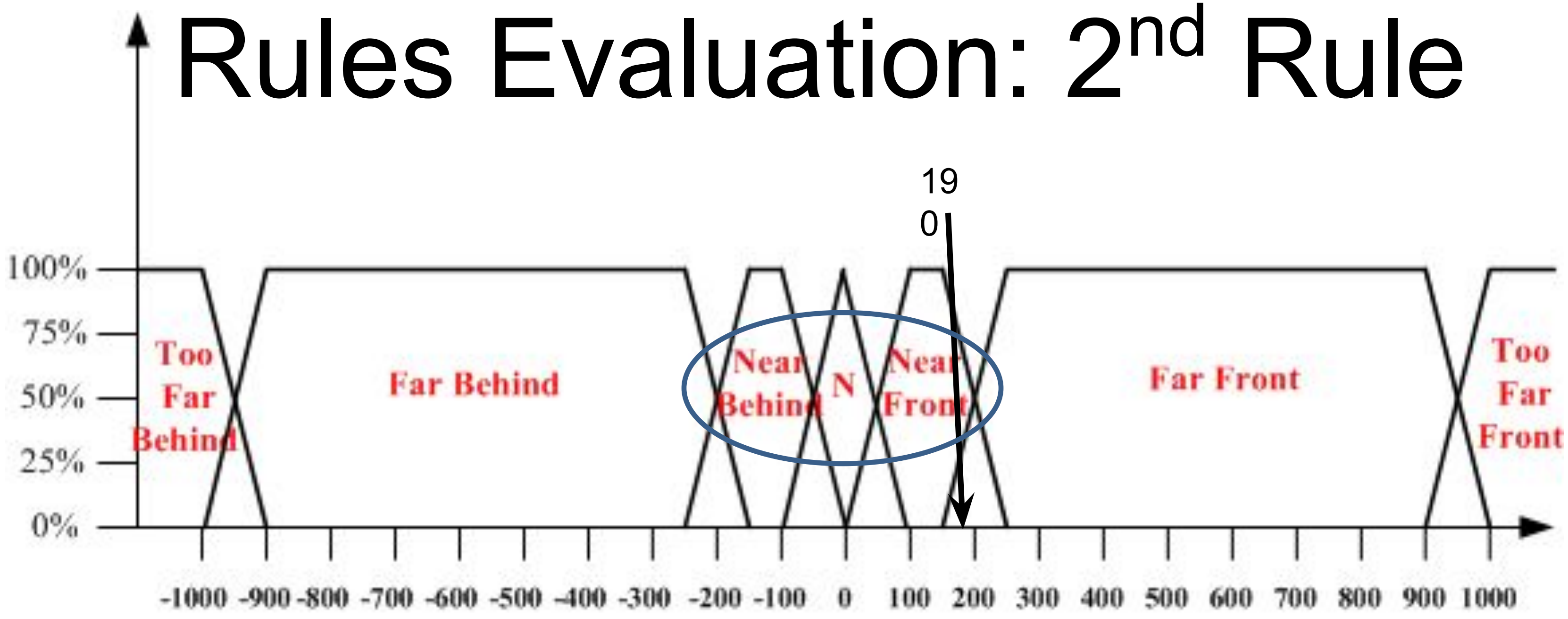
Rules Evaluation: 2nd Rule

- Assume that the current input is the same:
  - distance = 190
  - ranking = 5

- **IF** Near **OR** Near Behind **OR** Near Front **THEN** OSpeed Medium

- DOM(OSpeed Medium) =
  (DOM(Near) OR DOM(Near Behind)
  OR DOM(Near Front)) =
  (0 OR 0 OR 0.6) = 0.6

Rules Evaluation: 2nd Rule



```
float DOM_NearFront(float distance) {
   if (distance<=0)
      return 0;
   if ((distance>0)&&(distance<=100))
      return (distance-0)/(100–0)
   if ((distance>100)&&(distance<=150))
      return 1;
   if ((distance>150)&&(distance<=250))
      return (distance-250)/(150-250);
   return 0;
}
```

DOM(Near Front) = 0.6
DOM(Near Behind) = 0
DOM(Near) = 0

Rules Evaluation: 2nd Rule

- Assume that the current input is the same:
  - distance = 190
  - ranking = 5

- **IF** Near **OR** Near Behind **OR** Near Front **THEN** OSpeed Medium

- DOM(OSpeed Medium) =
  (DOM(Near) OR DOM(Near Behind)
  OR DOM(Near Front)) =
  (0 OR 0 OR 0.6) = 0.6

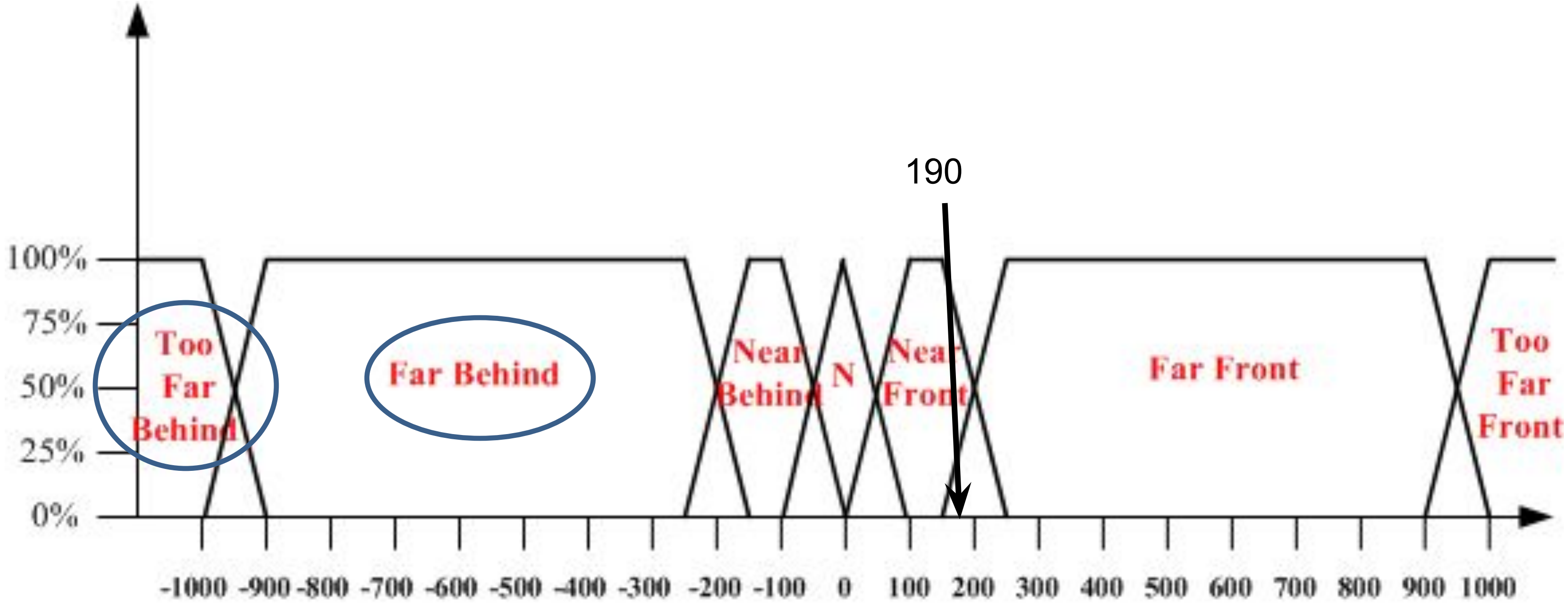
Rules Evaluation: 3nd Rule

- Assume that the current input is the same:
  - distance = 190
  - ranking = 5

- **IF** (Far Behind **OR** Too Far Behind) **AND** Ranking High **THEN** OSpeed High

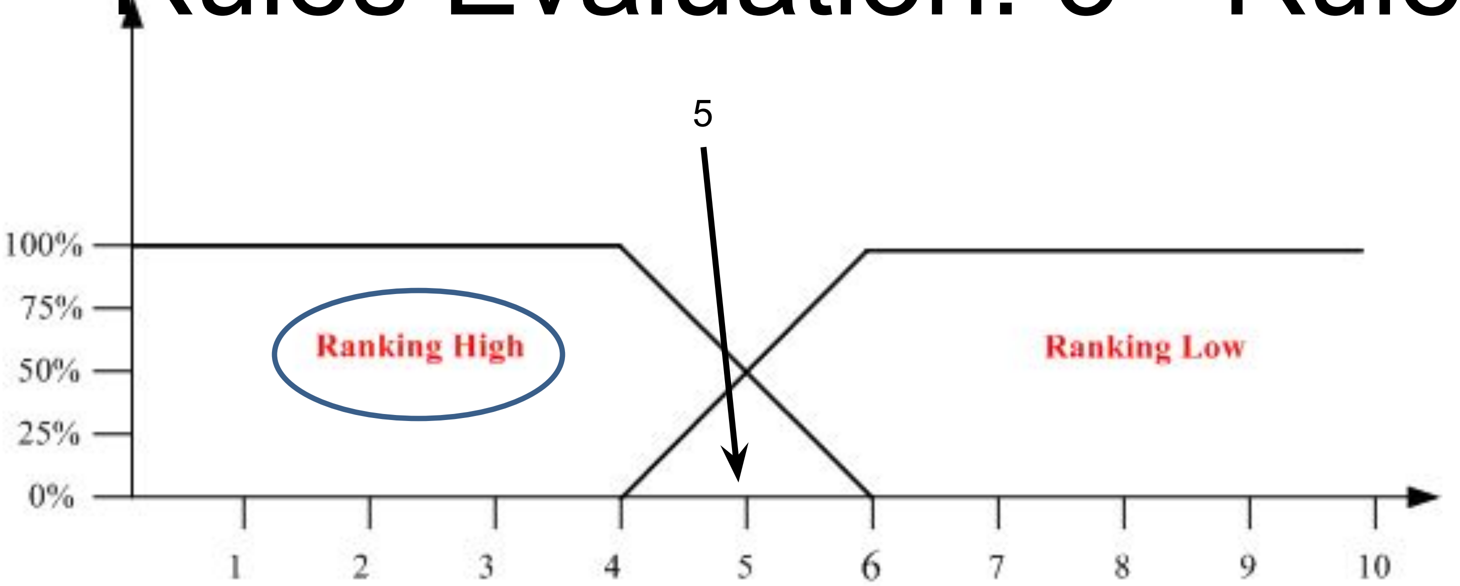
- DOM(OSpeed High) =
  (DOM(Far Behind) OR DOM(Too Far Behind))
  AND DOM(Ranking High) =
  (0 OR 0) AND 0.5 = 0

Rules Evaluation: 3rd Rule



DOM(Too Far Behind) = 0
DOM(Far Behind) = 0

Rules Evaluation: 3rd Rule



```
float DOM_RankingHigh(int rank){
    if (rank<=4)
        return 1;
    if ((rank>4)&&(rank<=6))
        return (6-rank)/2;
    if (rank>6)
        return 0;
}
```

DOM(Ranking High) = 0.5

Rules Evaluation: 3nd Rule

- Assume that the current input is the same:
  - distance = 190
  - ranking = 5

- **IF** (Far Behind **OR** Too Far Behind) **AND** Ranking High **THEN** OSpeed High

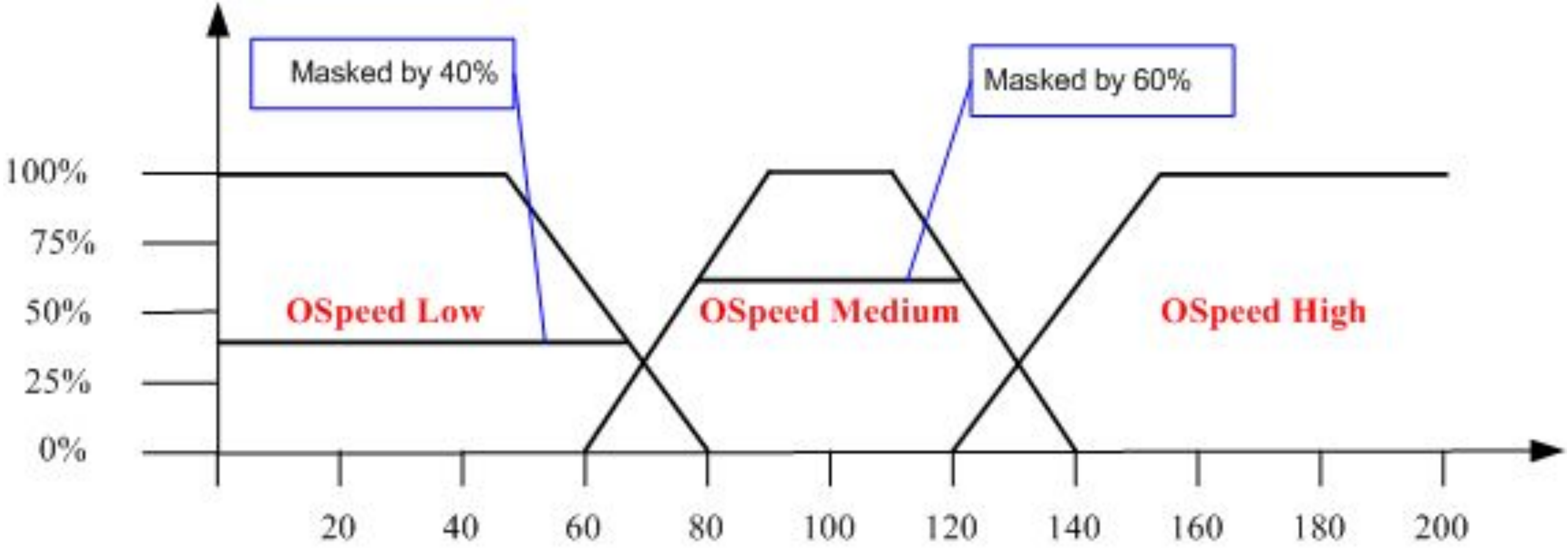
- DOM(OSpeed High) =
  (DOM(Far Behind) OR DOM(Too Far Behind))
  AND DOM(Ranking High) =
  (0 OR 0) AND 0.5 = 0

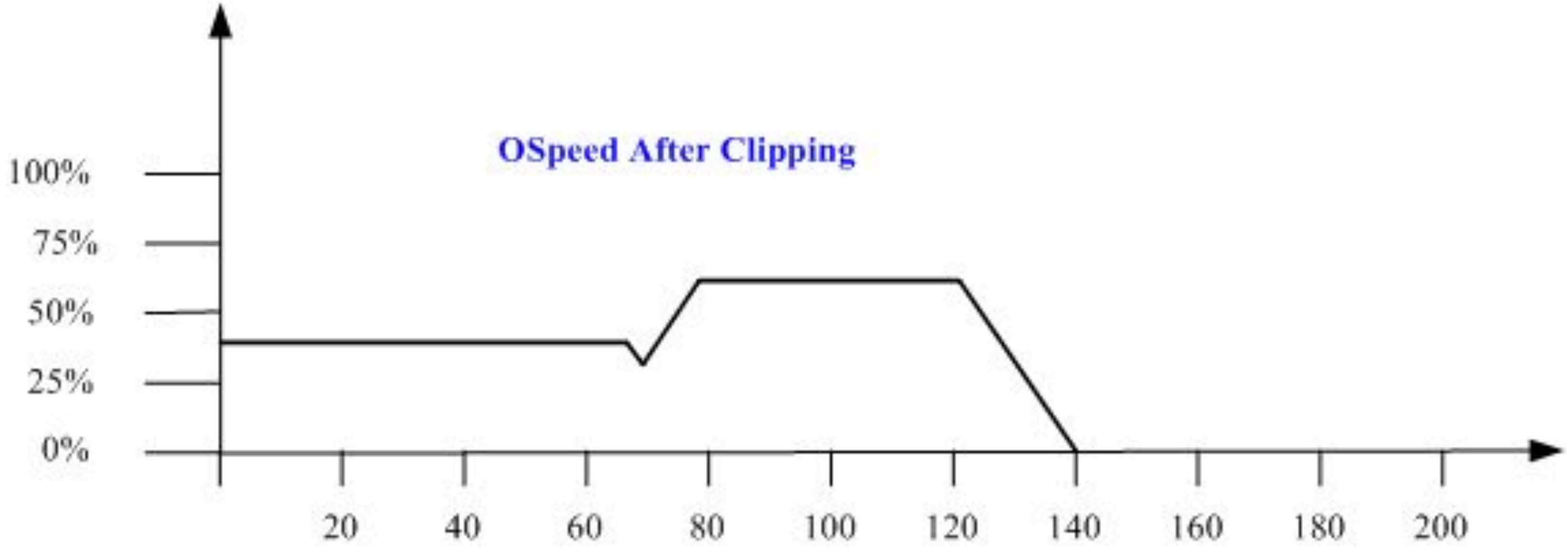
Inference: Aggregation

- What do we have now?
  - $\mathrm{DOM}(\mathrm{OSpeed\ Low}) = 0.4$
  - $\mathrm{DOM}(\mathrm{OSpeed\ Medium}) = 0.6$
  - $\mathrm{DOM}(\mathrm{OSpeed\ High}) = 0.0$

- How to aggregate them?
  - Two methods:
    - Clipping
    - Scaling

Aggregation: Clipping

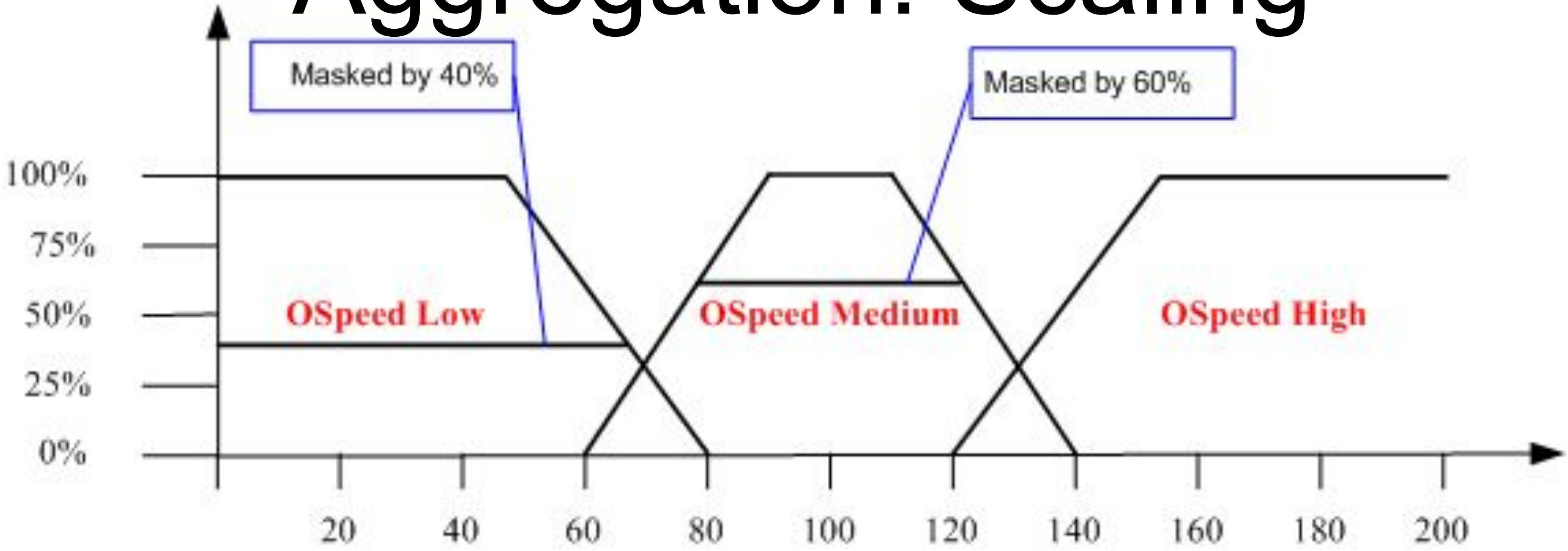


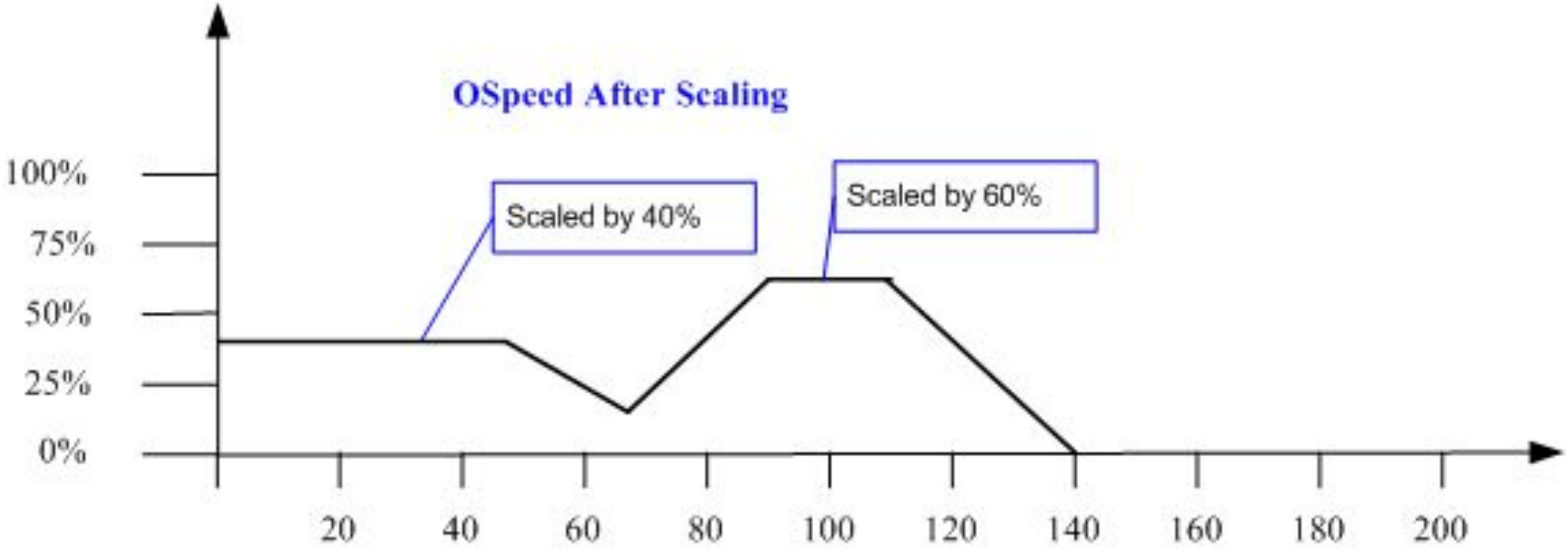


Aggregation: Clipping

- The top of the output Fuzzy sets is cut which makes some loses in the information
- However the clipping method has less computation complexities to generate an aggregated area to Defuzzify later
- Note: Is used only when applying the Centroid formula later

Aggregation: Scaling



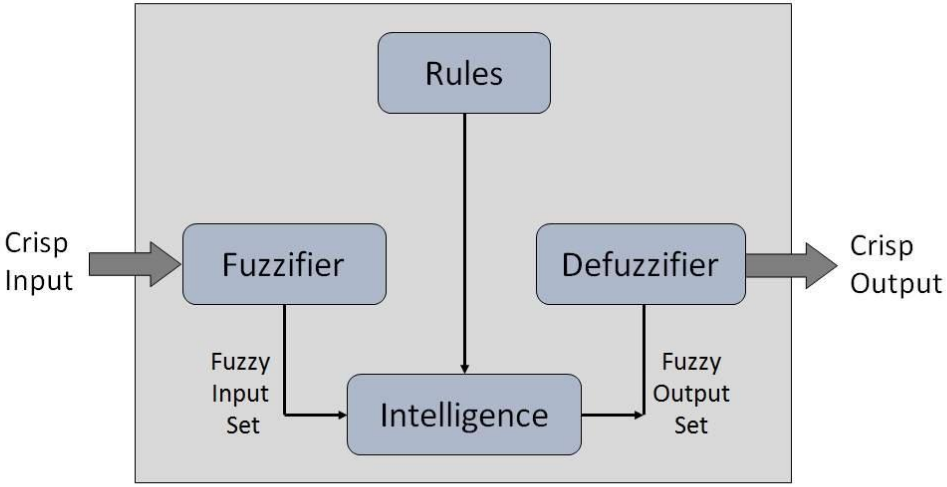


Aggregation: Scaling

- In the scaling method we multiply the original area of each set by the corresponding Fuzzy output which represents the scaling value
- Scaling method is a lesser used method than the Clipping method since it does more computation to preserve the shape of the area of each set
- This method loses less information after transformation and is more used in professional domains where accuracy is needed more
- Note: Is used only when applying the Centroid formula later

Inference: Defuzzification

- **Defuzzification** is the process of producing a quantifiable result in crisp logic, given fuzzy sets and corresponding DOMs.
- Examples:
  - Decide how much pressure to apply when given "Decrease Pressure (15%), Maintain Pressure (34%), Increase Pressure (72%)".



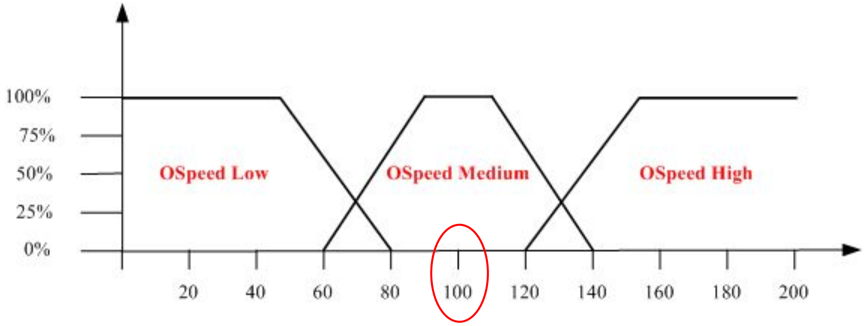
Inference: Defuzzification

- There are many different methods of defuzzification available, including the following:
  - Maximum value method
  - Singleton value method
  - Weighted average method (Singleton method extended)
  - Centroid method

Defuzzification: Maximum Value Method

Given a variable and DOMs:

DOM(OSpeed Low) = 0.4
DOM(OSpeed Medium) = 0.6
DOM(OSpeed High) = 0.0



1. Get the variable's set with maximum DOM
   => OSpeed Medium

2. **Result** is the representative value (ex: midpoint) of the found set
   => OSpeed = 100

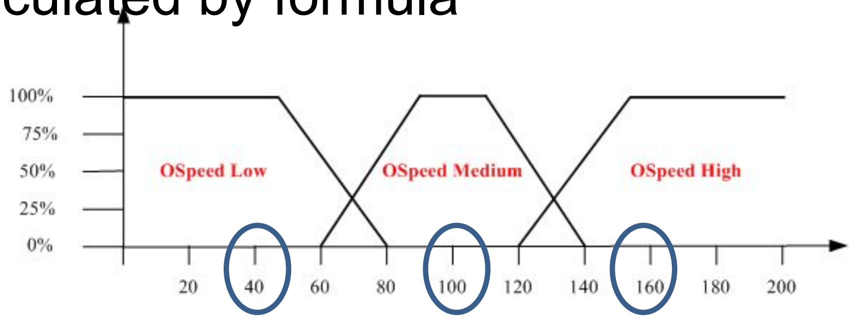
- This method is less accurate than other methods, but has a very fast computation
- It doesn't use the aggregation step done before

Defuzzification: Singleton Value Method

Given: Fuzzy sets S_i of a variable with DOMs and representative values (maxima or midpoint) K_i

Result of the defuzzification is calculated by formula

$$\frac{\sum_i(DOM(S_i) \times K_i)}{\sum_i DOM(S_i)}$$



Example:

$$\text{OSpeed} = \frac{0.4\times40 + 0.6\times100 + 0.0\times160}{0.4 + 0.6 + 0.0} = 76$$

- Method is less accurate but faster than the Centroid method coming next

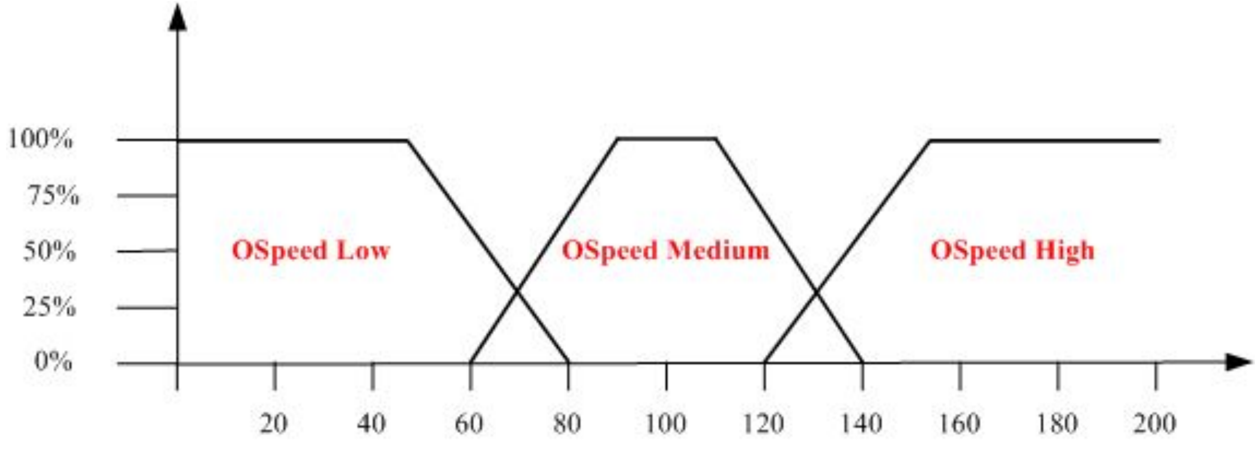
Defuzzification: Weighted Average Method

$W_1=(0+20+40+60)\times0.4+80\times0.0=48$

$W_2=60\times0.0+(80+100+120)\times0.6+140\times0.0=180$

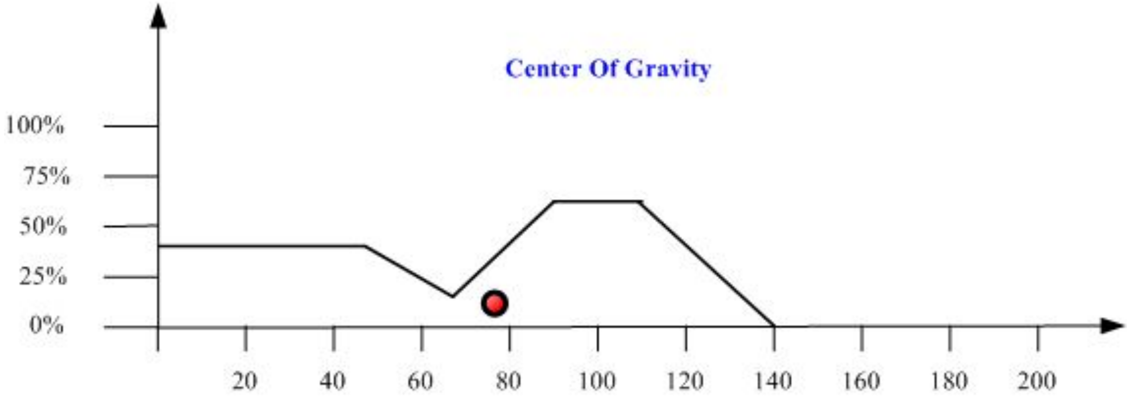
$W_3=(120+140+160+180+200)\times0.0=0$

$\text{OSpeed}=(W_1+W_2+W_3)/(0.4\times4+0.0+0.0+0.6\times3+0.0+0.0\times5)=67.059$



Defuzzification: Centroid Method

- Results of all rules are aggregated
- Compute the **center of gravity** for aggregated area
  - the unique point where the weighted relative position of the distributed mass sums to zero



- Ex: OSpeed=73 (by visual approximation)
- One of the most accurate methods but has high computational cost