DM842

Computer Game Programming: AI

Lecture 7

Decision Making

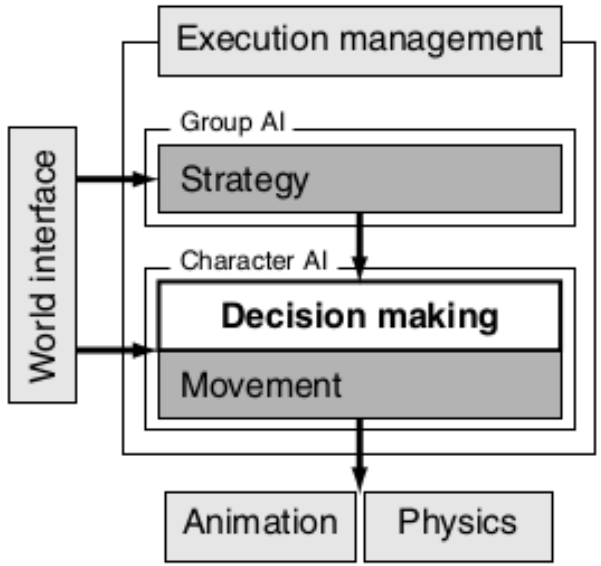
Christian Kudahl

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

1. Decision Making
    - Behavior Trees
    - Fuzzy Logic
    - Markov Systems
    - Goal-Oriented Behavior
    - Rule-Based Systems
    - BlackBoard Architectures

- state machines,
- decision trees
- behaviour trees
- fuzzy logic
- rule-based systems
- blackboard systems



Outline

1. Decision Making
    - Behavior Trees
    - Fuzzy Logic
    - Markov Systems
    - Goal-Oriented Behavior
    - Rule-Based Systems
    - BlackBoard Architectures

Outline

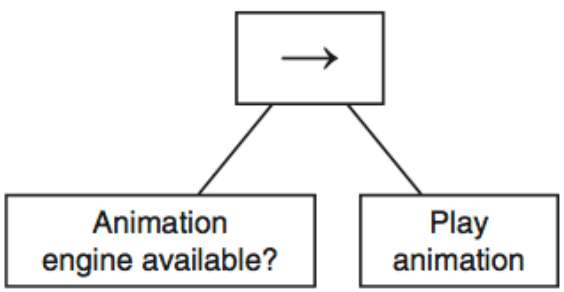
Resource Limitation

Limitations on resources: Examples:

- animation engine can only play one animation on each part of the skeleton at any time.
- no more than one audio sample per character at a time

Hence we need to check availability of resource:

1. By hard-coding the test in the behavior
2. By creating a Condition task to perform the test and using a Sequence
3. By using a Decorator to guard the resource



Decorators solution

- Semaphores are a mechanism for ensuring that a limited resource is not over subscribed.
- can cope with resources that aren't limited to one single user at a time.
- before using the resource, a piece of code must ask the semaphore if it can acquire it.
- when the code is done it should notify the semaphore that it can be released
- Most programming languages have libraries for semaphores removing the need to deal with low-level operating system primitives for locking.
- The Decorator returns its failure status code when it cannot acquire the semaphore. A select task higher up the tree will find a different action

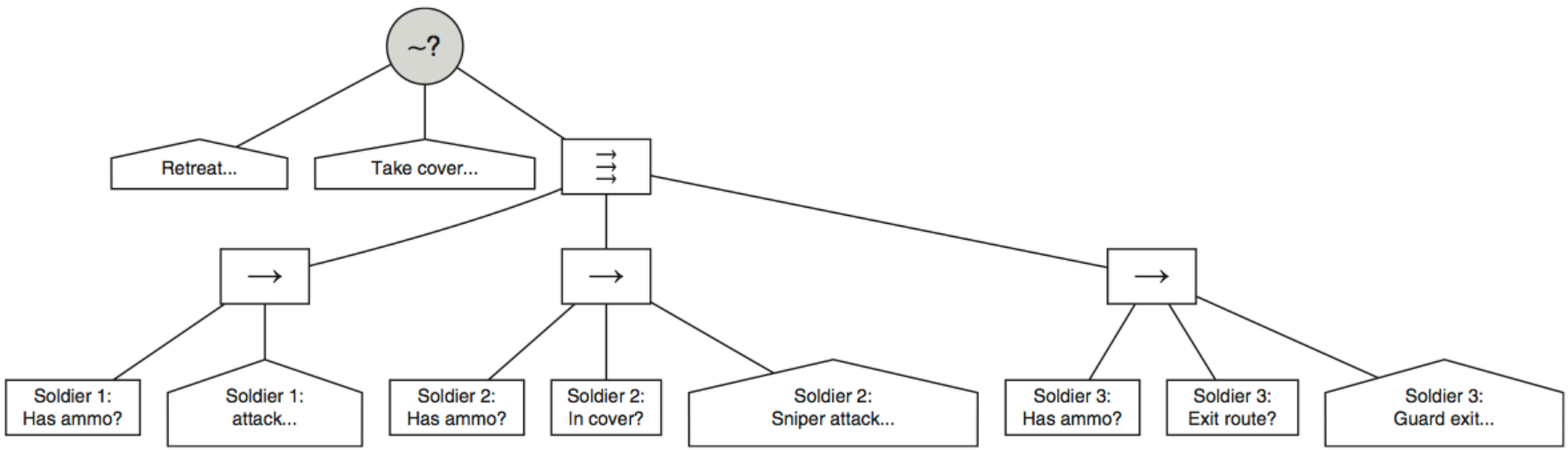
Concurrency, Timing

- So far we assumed only one task runs at a time.
- How do behavior trees work with respect to subsequent frames? how will it know what to do? Should we restart from the top of the tree every time?
- Concurrency: each behavior tree is running in its own thread. An Action can take seconds to carry out: the thread just sleeps while it is happening and wakes again to return True back to whatever task was above it in the tree. Since it can be highly wasteful to run lots of threads at the same time even on multi-core machines we may also need cooperative multitasking and scheduling algorithms.

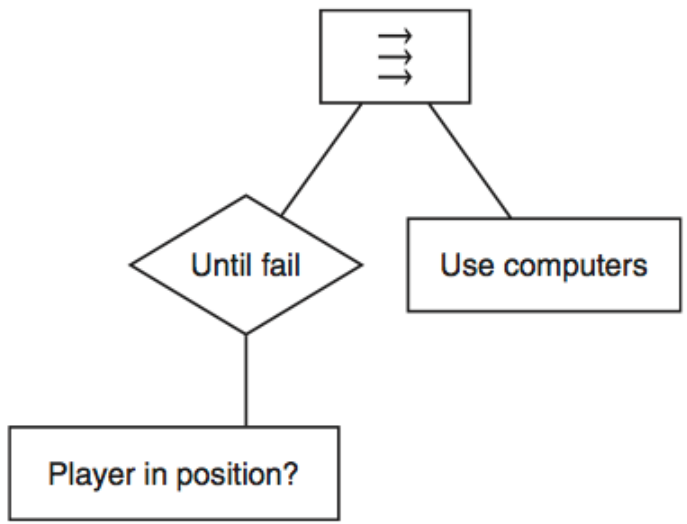
Parallel tasks

- Parallel task similar to the Sequence task.
- it has a set of child tasks, and runs them until one of them fails, in which case it return failure.
  If all child tasks complete successfully, the Parallel task returns with success.
- children are run simultaneously
- when one thread fail all other threads are terminated.
- tasks need to be able to receive a termination message and clean up themselves after termination

parallel tasks, application example

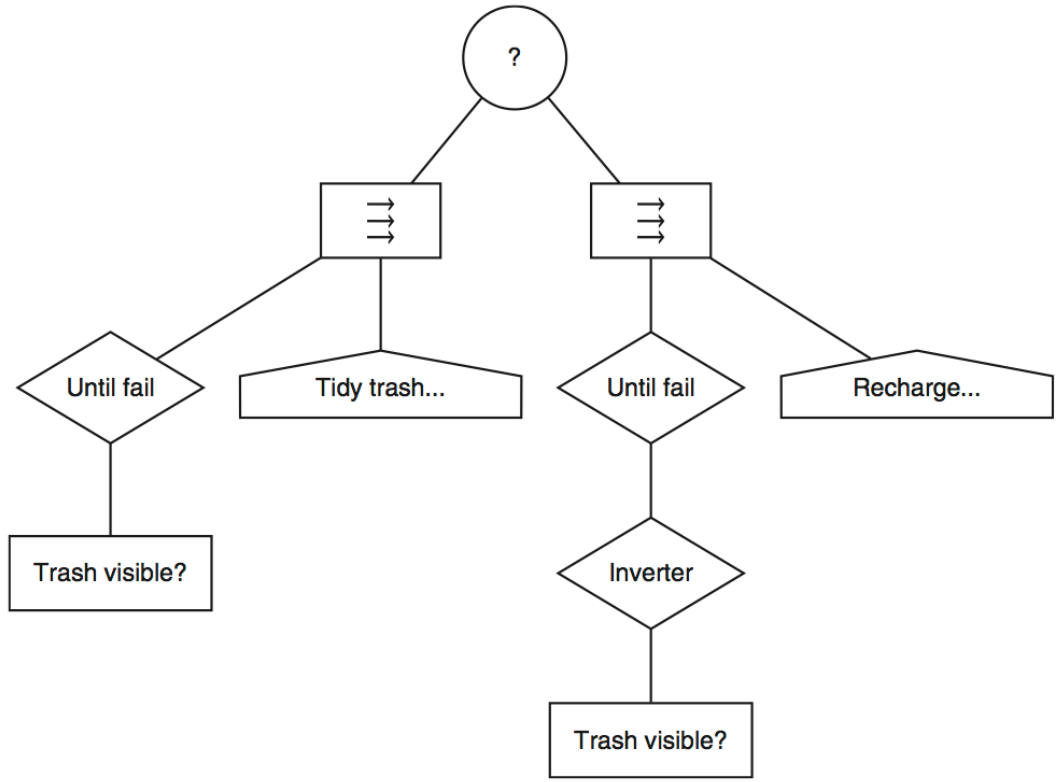


Using Parallel blocks to make sure that Conditions hold



Using Parallel blocks to make sure that Conditions hold

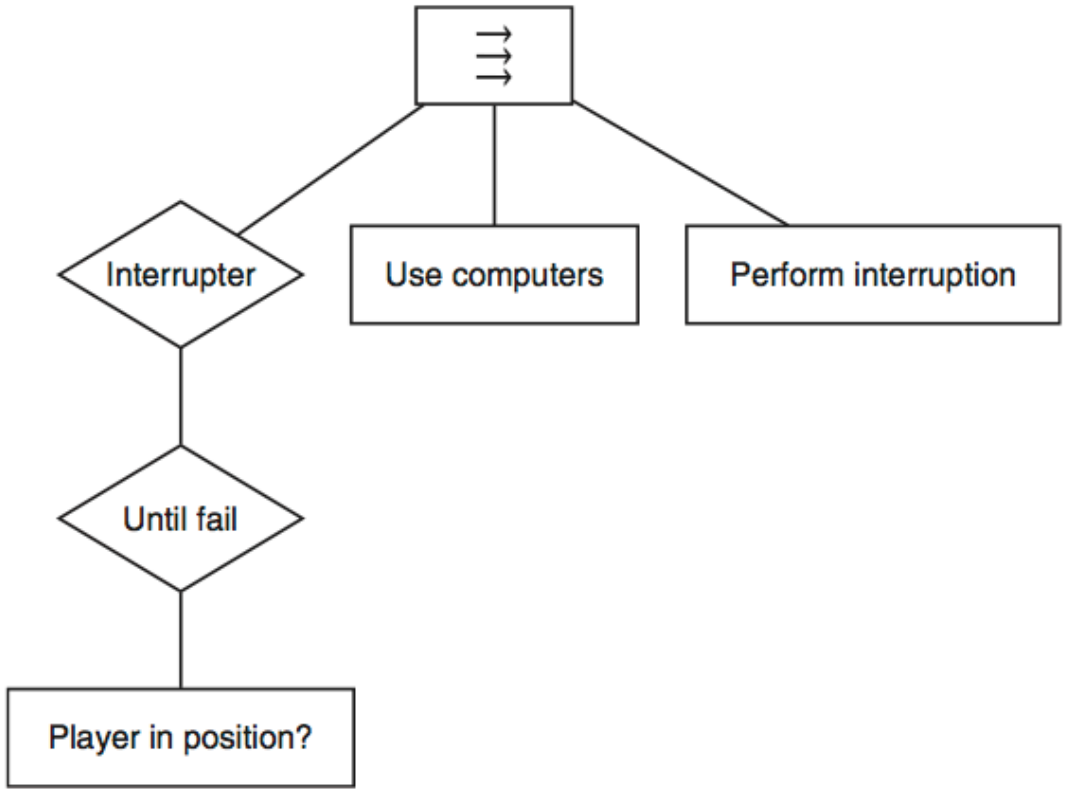
achieve state machines and ability to switch tasks when events occur



- but unnatural: events cause a change of action, rather than the lack of the event allows the lack of a change of action.
- state-based behaviors are hard to model!
  eg: a character who needs to respond to external events
  eg: interrupting a patrol route to go into hiding or to raise an alarm

Inter-behavior communication

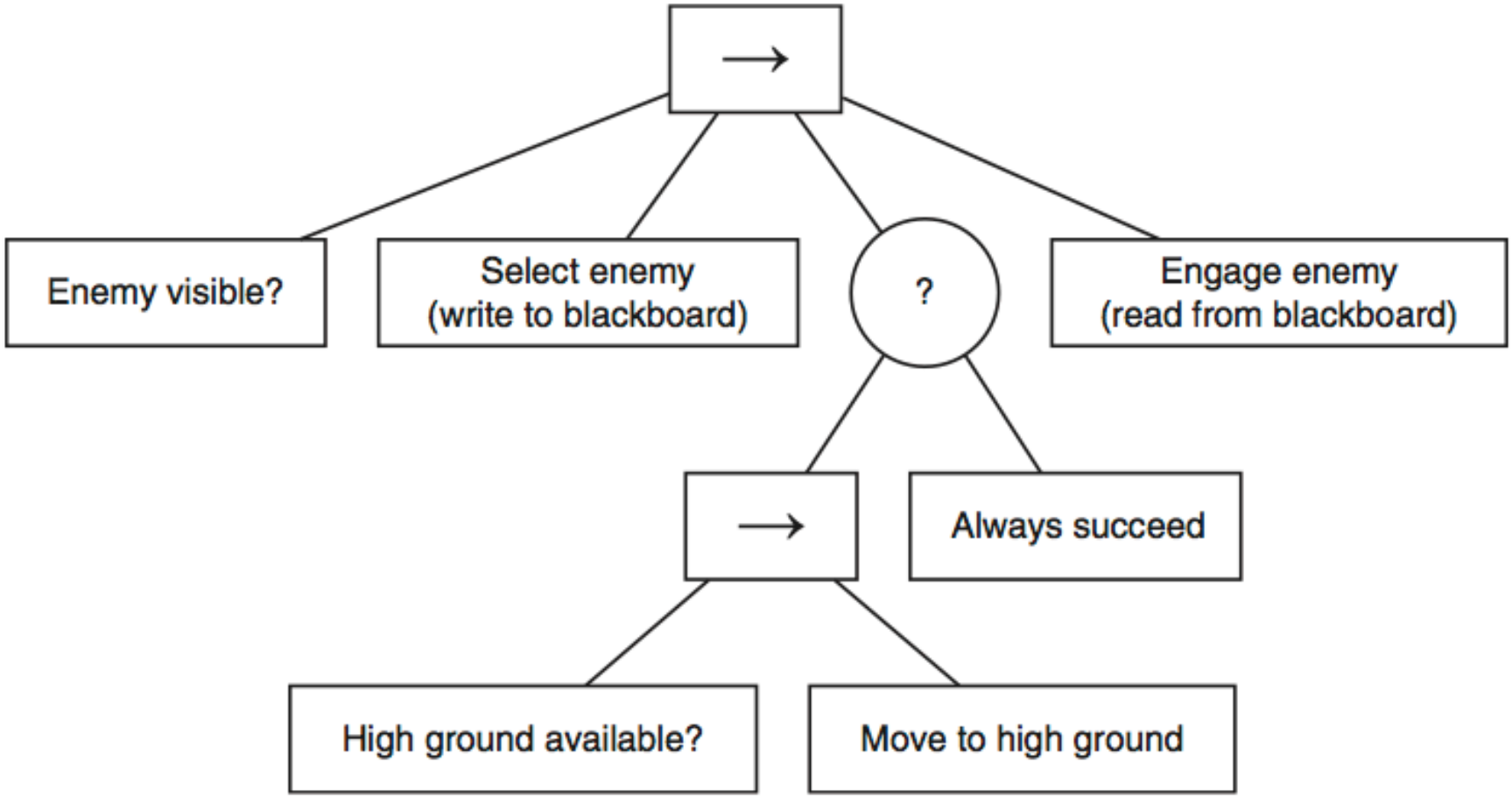
Interrupter decorators



We want data to pass between behavior trees, but we dont want data into tasks as parameters to their run method. Else each task needs to know what arguments its child tasks take and how to find these data.

Data

- Decouple the data that behaviors need from the tasks themselves.
- `blackboard`: external data store for all the data that the behavior tree needs. Tasks can then query the `blackboard` for data.
- we can write tasks that are still independent of one another but can communicate when needed.
- we can still handle data privacy by introducing hierarchies of `blackboards`. Tasks that are roots of Subtrees create their own `blackboards` (similar to scopes in many programming languages).
- tasks can communicate by writing and reading from the `blackboard` rather than calling methods.



The tasks should be written so that, if the blackboard had no target, then the task fails, and the behavior tree can look for something else to do.

Implementation Issues

A. Construction
   Levels of abstraction

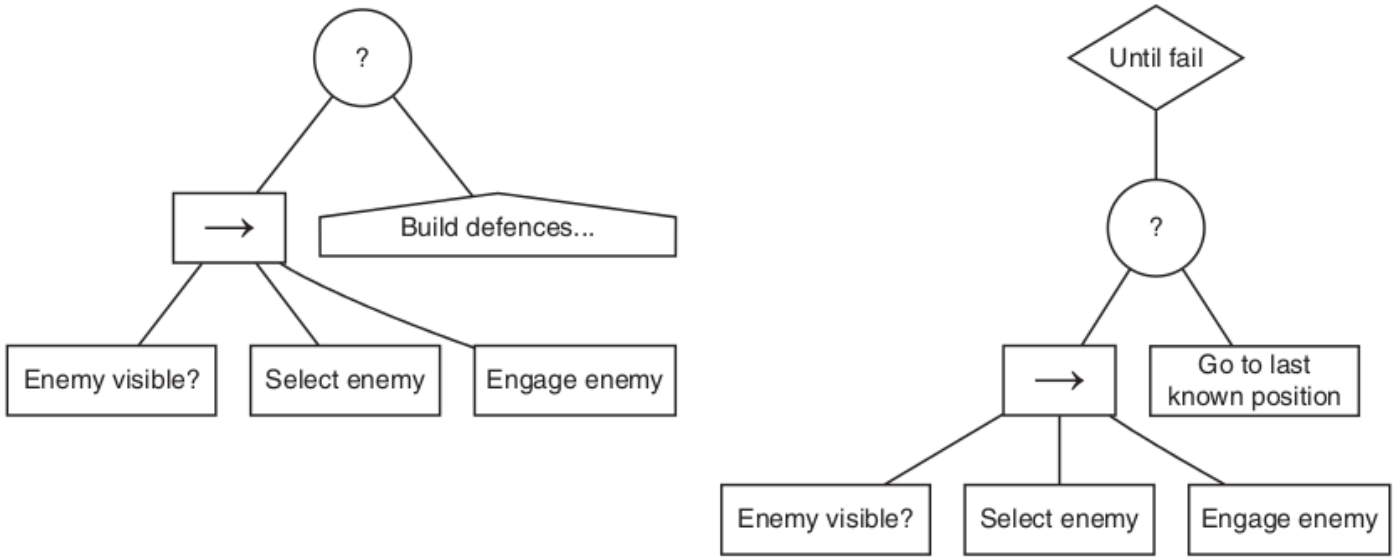
   1. classes abstract concepts about how to achieve some task we saw in pseudo-code.
   2. instances of these classes arranged in a behavior tree.
   3. the behavior tree needs is instantiated for a particular character at a particular time.

   Use a cloning operation to instantiate trees for characters.
   use behavior tree as an "archetype"; Any time we need an instance of that behavior tree we take a copy of the archetype and use the copy; each task has a clone method that makes a copy of itself.

B. Reuse behavior trees for multiple characters

```
Enemy Character (goon):
model = ‘‘enemy34.model’’
texture = ‘‘enemy34-urban.tex’’
weapon = pistol-4
behavior = goon-behavior
```

C. use sub-trees multiple times in different contexts



store partial sub-trees in the behavior tree library

Outline

Fuzzy Logic

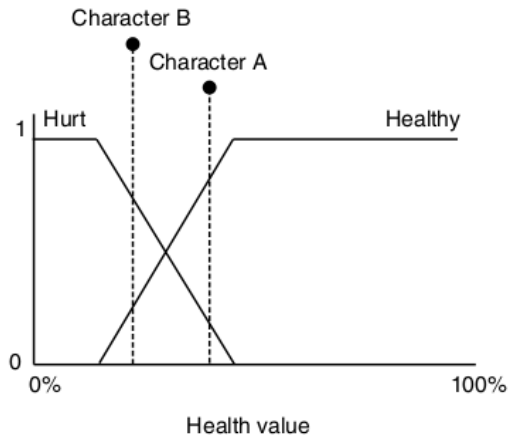
- So far decisions based on `true` and `false`.
- Fuzzy logic includes a range of degrees for decisions.
- It is popular in games to represent any kind of uncertainty

Fuzzy sets

- predicates: sentence about the world are true or false
- classical sets: either belongs to the set or not
- fuzzy sets: everything can partially belong to the set according to a numeric value called degree of membership.
  sets without sharp boundaries
- in fuzzy logic predicate have a value.
- degree of membership: $[0, .., 255] \subset \mathbf{N}$ or $[0, 1] \subset \mathbf{R}$
- everything can be a member of multiple sets at same time

Fuzzification

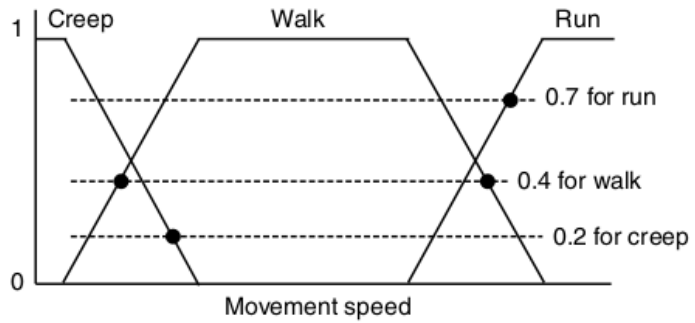
- Fuzzification: turning regular data into degrees of membership
- Eg: membership function: a function that maps the input value (hit points) to a degree of membership, for each fuzzy set.



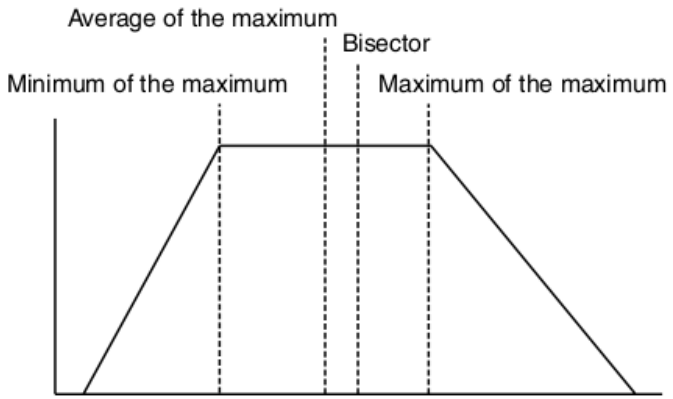
- their values don't need to add up to 1, although in most cases it is convenient if they do.
- for Boolean values pre-determined membership values for each relevant set.

Defuzzification

- Defuzzification: turning a set of membership values into a single output value.



1. Highest Membership: Eg: 0.7 $\rightsquigarrow$ run + precomputed value:



$\rightsquigarrow$ 0 creep, 0 walk, 1 run $\equiv$ 0.33 creep, 0.33 walk, 0.34 run.

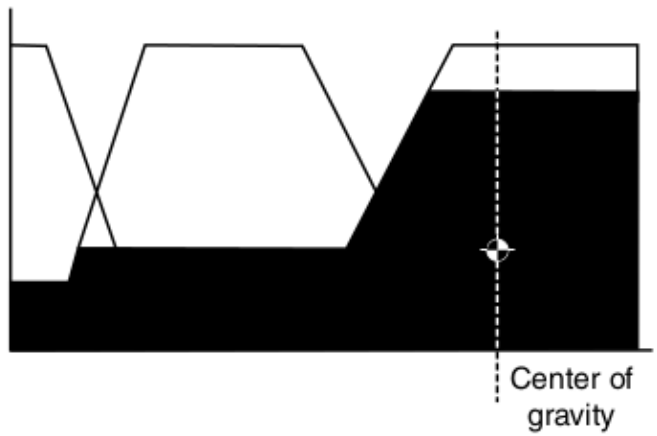
2. Blending Based on Membership:
   eg: 0.33 creep, 0.33 walk, 0.34 run $\mapsto$ (0.33$\times$ characteristic creep speed) + (0.33$\times$ characteristic walk speed) + (0.34$\times$ characteristic run speed).

- blend of minima: Smallest of Maximum, or Left of Maximum (LM).
  blend of the maxima: Largest of Maximum (LM), Right of Maximum.
  blend of the average values: Mean of Maximum (MoM).

- allows predefined values

- often the preferred approach because quick to calculate

3. Center of Gravity: crop functions, and calculate center of mass (requires integration). Cannot be precomputed.



IEEE version doesn't crop each function before calculating its center of gravity and so can be precomputed $\rightsquigarrow$ blending

Defuzzification to a Boolean Value: cut-off value

Fuzzy Logic Reasoning

It might be partially raining (membership of 0.5) and slightly cold (membership of 0.2).

What is the value of the compound statement such as "it is raining AND cold"?

Expression	Equivalent	Fuzzy Equation
AND		$m_{(A \text{ AND } B)} = \min\{m_A, m_B\}$
OR		$m_{(A \text{ OR } B)} = \max\{m_A, m_B\}$
NOT		$m_{\text{NOT } (A)} = 1 - m_A$
XOR	NOT(B) AND A OR NOT(A) AND B	$m_{(A \text{ XOR } B)} = \max\{\min\{m_A, 1 - m_B\} \min\{1 - m_A, m_B\}\}$

(corresponds to first order logic when $m_A = \{0, 1\}$ and $m_B = \{0, 1\}$)

It may be reasonable but also not: if $T(Funny(Christian)) = 0.4$ then $T(Funny(Christian) \wedge \neg(Funny(Christian)) = 0.4$

Fuzzy Rules

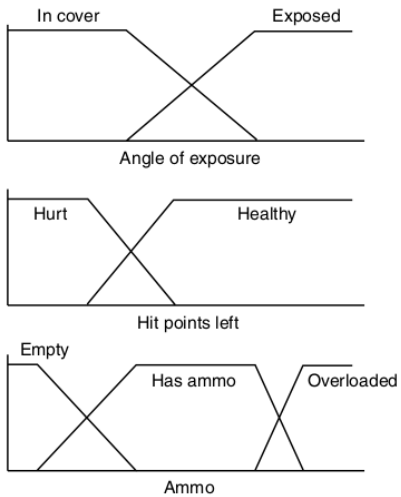
from known membership of certain fuzzy sets
to membership values for other fuzzy sets.

Example: Should brake if close to the corner AND traveling fast

$$m_{(\text{should brake})} = \min\{m_{(\text{close to the corner})}, m_{(\text{traveling fast})}\}$$

Fuzzy Control – Block Format Rules

- also used to build industrial controllers that take action based on a set of inputs
- it can be used to determine if transitions in a state machine should fire, or the activity at the states
- based on AND rules



input 1 state AND ... AND input *n* state THEN out state

Hurt AND In cover AND Empty THEN brake
Healthy AND Exposed AND Overloaded THEN accelerate
...

needs rules for each combination of inputs.
Here 12 rules (2 x 2 x 3).

- For each rule calculate degree of membership for output state taking the minimum degree of membership for input states in that rule (AND).
- Output: maximum output from any of the applicable rules.

Example

with only two inputs:

corner-entry AND going-fast THEN brake
corner-exit AND going-fast THEN accelerate
corner-entry AND going-slow THEN accelerate
corner-exit AND going-slow THEN accelerate

We might have the following degrees of membership:

Corner-entry: 0.1
Corner-exit: 0.9
Going-fast: 0.4
Going-slow: 0.6

Then the results from each rule are

$$\begin{aligned} \text{Brake} &= \min(0.1, 0.4) = 0.1 \\ \text{Accelerate} &= \min(0.9, 0.4) = 0.4 \\ \text{Accelerate} &= \min(0.1, 0.6) = 0.1 \\ \text{Accelerate} &= \min(0.9, 0.6) = 0.6 \end{aligned}$$

then take max per state (0.1;0.6), and defuzzificay or keep numerical value.

Complexity? How can we speed up the computation procedure?

Fuzzy Control – Combs

Alternative fuzzy control method based on rules of the form:

$$\text{a AND b ENTAILS c}$$

$$\text{a AND b ENTAILS c} \equiv \text{(a ENTAILS c) OR (b ENTAILS c)} \equiv \begin{array}{l}\text{IF } a \text{ THEN } c \\ \text{IF } b \text{ THEN } c\end{array}$$

$$\text{IF } a_1 \text{ AND } \ldots \text{ AND } a_n \text{ THEN } c \qquad \begin{array}{l}\text{IF } a_1 \text{ THEN } c \\ . \\ . \\ . \\ \text{IF } a_n \text{ THEN } c.\end{array}$$

From having rules involving all possible combinations of states to a simple set of rules with only one state in the IF-clause and one in the THEN-clause

but

IF corner-entry AND going-fast THEN brake
IF corner-exit AND going-fast THEN accelerate

IF corner-entry THEN brake
IF going-fast THEN brake
IF corner-exit THEN accelerate
IF going-fast THEN accelerate

This is an inconsistent set of rules $\rightsquigarrow$ one rule can be decomposed, more than one rule cannot

One has to take care of defining only consistent rules
more restrictive but less heavy

corner-entry AND going-fast THEN brake
corner-exit AND going-fast THEN accelerate
corner-entry AND going-slow THEN accelerate
corner-exit AND going-slow THEN accelerate

could be expressed as:

corner-entry THEN brake
corner-exit THEN accelerate
going-fast THEN brake
going-slow THEN accelerate

With inputs of:

Corner-entry: 0.1
Corner-exit: 0.9
Going-fast: 0.4
Going-slow: 0.6

the block format rules give us results of:

$$\text{Brake} = 0.1$$
$$\text{Accelerate} = 0.6$$

while Combs method gives us:

$$\text{Brake} = 0.4$$
$$\text{Accelerate} = 0.9$$

Outline

Markov Processes

- dynamic numerical values associated to state to represent level of risk
- state vector: each position in the vector corresponds to a single state and has a value.
  Often with random variables values are probability of events and sum up to one.
- values in the state vector change according to the action of a transition matrix.

π represents the safety of four sniping positions.
Shooting from position 1 implies the transition $\mathbf{M}$.
Position 1 got less safe, but the others got more safe.

$$\pi = \begin{bmatrix} 1.0 \\ 0.5 \\ 1.0 \\ 1.5 \end{bmatrix} \qquad \mathbf{M} = \begin{bmatrix} 0.1 & 0.3 & 0.3 & 0.3 \\ 0.0 & 0.0 & 0.0 & 0.8 \\ 0.0 & 0.0 & 0.0 & 0.8 \\ 0.0 & 0.0 & 0.0 & 0.8 \end{bmatrix}$$

$$\pi' = \begin{bmatrix} 0.1 \\ 0.7 \\ 1.1 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.3 & 0.3 & 0.3 \\ 0.0 & 0.0 & 0.0 & 0.8 \\ 0.0 & 0.0 & 0.0 & 0.8 \\ 0.0 & 0.0 & 0.0 & 0.8 \end{bmatrix} \begin{bmatrix} 1.0 \\ 0.5 \\ 1.0 \\ 1.5 \end{bmatrix}$$

Markov State Machine

- states of the machine are numeric values
- state vector is changed by transition matrices at occurrence of events.
- transition matrices are triggered by conditions and apply to the whole machine
- default transition occurs if no other transition is triggered.
  it may be time dependent. Timer reset by other transitions.
- there are no states but only one vector state $\rightsquigarrow$ actions are activated only by transitions.

Outline

Goal Oriented Behavior

- So far we have focused on approaches that react on input
- here we make the character seem like it has goals or desires (eg, catch someone, stay alive)
- but needed some flexibility in its goal seeking
- To look human, characters need to demonstrate their emotional and physical state by choosing appropriate actions. They should eat when hungry, sleep when tired, chat to friends when lonely
  decision trees would have too many possibilities to consider

  better:

- goal-oriented behavior: set of actions from which to choose the best one that meets the character's internal goals.

Goals

- A character may have one or more goals, also called motives.
- Each goal has a number representing a level of importance aka insistence
- the insistence may vary during the game in a pattern typical for the specific goal
- the insistence determines which goal to focus on

Actions

- actions can be generated centrally, but it is also common for them to be generated by objects in the world.
  eg. empty oven adds an "insert raw food"; enemy adds an "attack me"

- actions are pooled in a list of options and rated against the motives of the char.

People simulating example:

Goal	Eat = 4
Goal	Sleep = 3
Action	Get-Raw-Food (Eat - 3)
Action	Get-Snack (Eat - 2)
Action	Sleep-In-Bed (Sleep - 4)
Action	Sleep-On-Sofa (Sleep - 2)

choose the most pressing goal
(the one with the largest insistence)
and find an action that provides it
with the largest decrease in insistence.

Side Effects and Overall Utility

Goal	Eat = 4
Goal	Bathroom = 3
Action	Drink-Soda (Eat - 2; Bathroom + 3)
Action	Visit-Bathroom (Bathroom - 4)

- discontentment of the character: high insistence leaves the character more discontent
- aim of the character is to reduce its overall discontentment level
- add together all the insistence values to give the discontentment of the character.

  better:
  scale insistence so that higher values contribute disproportionately high discontentment values, eg, square

Goal	Eat = 4	
Goal	Bathroom = 3	
Action	Drink-Soda (Eat - 2; Bathroom + 2)	$\rightsquigarrow$ Eat = 2, Bath. = 5: Disc. = 29
Action	Visit-Bathroom (Bathroom - 4)	$\rightsquigarrow$ Eat = 4, Bath. = 0: Disc. = 16

Timing

- The time it takes for an action enters also in the decision process.
- Actions expose their duration time.
- Time split in time to get to location + time to complete
- A heuristic such as "the time is proportional to the straight-line distance from the character to the object"
  calculated via path finding
- take into account the consequences of the extra time if possible to know.
  Example:

Goal	Eat = 4 changing at + 4 per hour	
Goal	Bathroom = 3 changing at + 2 per hour	
Action	Eat-Snack (Eat - 2) 15 minutes	$\rightsquigarrow$ Eat = 2, Bath. = 3.5 Disc. = 16.25
Action	Eat-Main-Meal (Eat - 4) 1 hour	$\rightsquigarrow$ Eat = 0, Bath. = 5 Disc. = 25
Action	Visit-Bathroom (Bathroom - 4) 15 minutes	$\rightsquigarrow$ Eat = 5, Bath. = 0 Disc. = 25

Planning

- actions are situation dependent, it is normal for one action to enable or disable several others.
- action sequences and resource consumptions must be taken into account Example:

Goal	Heal = 4
Goal	Kill-Ogre = 3
Action	Fireball (Kill-Ogre −2) 3 energy-slots
Action	Lesser-Healing (Heal −2) 2 energy-slots
Action	Greater-Healing (Heal −4) 3 energy-slots

  If char has 5 energy slots, then choosing Greater-Healing would leave without energy for further actions.

- Overall Utility GOA planning: allows characters to plan detailed sequences of actions that provide overall optimum fulfillment of their goals.

- Need a model of the game world: implemented as a list of differences from previous states
- k: maximum depth parameter that indicates how many moves to look-ahead
- exact search: depth first search in the search space of sequences of actions $\rightsquigarrow O(nm^k)$, n num. of goals; m num. of actions
- heuristic search: never consider actions that lead to higher discomfort values
- The problem is, we don't know what we are looking for!

GOAP with IDA*

- If we forget about discontentment, choose a single goal on the basis of its insistence, and want to find the best action sequence that leads to it, then we can use A*
- best: in total number of actions, in total duration, resource consumption
- assume that there is at least one valid route to the goal
  allow A* to search as deeply as needed
- In case it is not possible to reach our goal: consider iterative deepening A* (maximum search depth + the cut-off value)
- heuristic function that estimates how far a given world model is from the goal or $h = 0$.
- avoid considering same set of actions over and over in each depth-first search (ie, symmetries) $\rightsquigarrow$ transposition table, ie hash value of the world model (avoid chaining by replacing an entry if the current entry has a smaller number of actions associated with it)

Smelly GOAP

- Objects diffuse smells: eg: an oven: “I can provide food” smell,
  a bed “I can give you rest” smell

- characters follow the smell for the motive it is most concerned with
  fulfilling

- diffusion takes time to spread and the smell diminishes as one gets away
  from source
  characters can move in the direction of the greatest concentration of
  smell at each frame

- if three possible sources of food:
  compare: conventional GOAP uses pathfinder to find the easiest source
  vs
  smelly GOAP approach

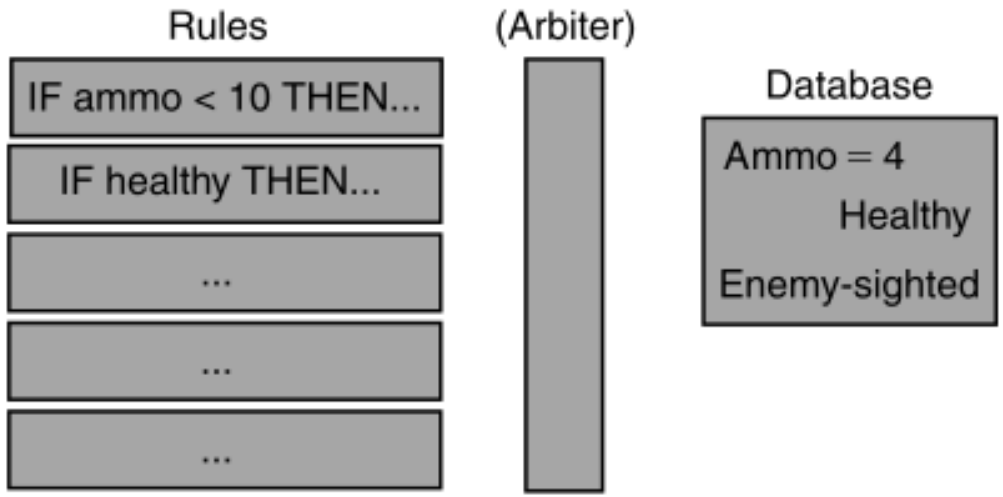
Motives may require intermediate actions to be fulfilled.

- Action-Based Signals: Requires conventional GOAP
- Character-Specific Signals: objects only emit signals if they are capable of being used by the character at that specific time. Signals diffusing around the game are now dependent on one particular character $\rightsquigarrow$ problem if large number of motives

Outline

Rule-based systems

- database containing knowledge + set of if-then rules (+ arbiter)



- inefficient and difficult to implement + similar behaviors can almost always be achieved by decision trees or state machines.

Example

Database:

```
Captain's health is 51
Johnson's health is 38
Sale's health is 42
Whisker's health is 15
Radio is held by
     Whisker
```

Condition-Actions rules:

```
IF Whisker's health < 15 AND Radio is held by Whisker
THEN Sale: pick up the radio
```

The character decides to `pick up the radio`, the game decides whether the action succeeds and the database needs update
Possible also to have actions that manipulate the database

Wild cards:

```
Anyone's health < 15 AND Anyone's health > 45
```

The rule-based system simply checks each of its rules to see if they trigger on the current database. The first rule that triggers is fired, and the action associated with the rule is run.

Reasoning carried out by forward chaining

Database

- Database consists of identifiers.
- conditions are matched with identifiers and their values
- hierarchical format. A Datum either holds a value or holds a set of Datum objects.

```
Captain's-weapon = rifle
Johnson's-weapon = machine-gun
Captain's-rifle-ammo = 36
Johnson's-machine-gun-ammo = 229
```

Example:

```
(
 Captain (Weapon (Rifle (Ammo 36) (Clips 2)))
         (Health 65)
         (Position [21, 46, 92])
)
```

```
(?anyone (Health 0-15))
```

Rule Arbitration

An arbiter policy decides which rules fire when more than one rule triggers.

- first applicable
  rules are provided in a fixed order, and the first rule in the list that triggers gets to fire.
  Rules are suspended until database changes (or particular Datum changes)
- last recently used
  When a rule fires, it is moved to the end of the list
- random rule*
  select at random among those that trigger
- more specific conditions:
  More specific rules should be preferred over more general rules (count ANDs)
- dynamic priority arbitration*
  based on dynamic priorities that returned by each rule on how important its action might be in the current situation.
  Eg: `Get health pack` When the character's health is high, the rule may return a low priority.

Unification

```
(?person (health 0-15))
AND
(Radio (held-by ?person))
```

if those are simply wild cards then we would match:

```
(Johnson (health 38))
(Sale (health 15)) # <=
(Whisker (health 25))
(Radio (held-by Whisker)) # <=
```

which is not what we want... we want the same person for both patterns

```
(Johnson (health 38))
(Sale (health 42))
(Whisker (health 15)) # <=
(Radio (held-by Whisker)) # <=
```

In unification, a set of wild cards are matched so that they all refer to the same thing.

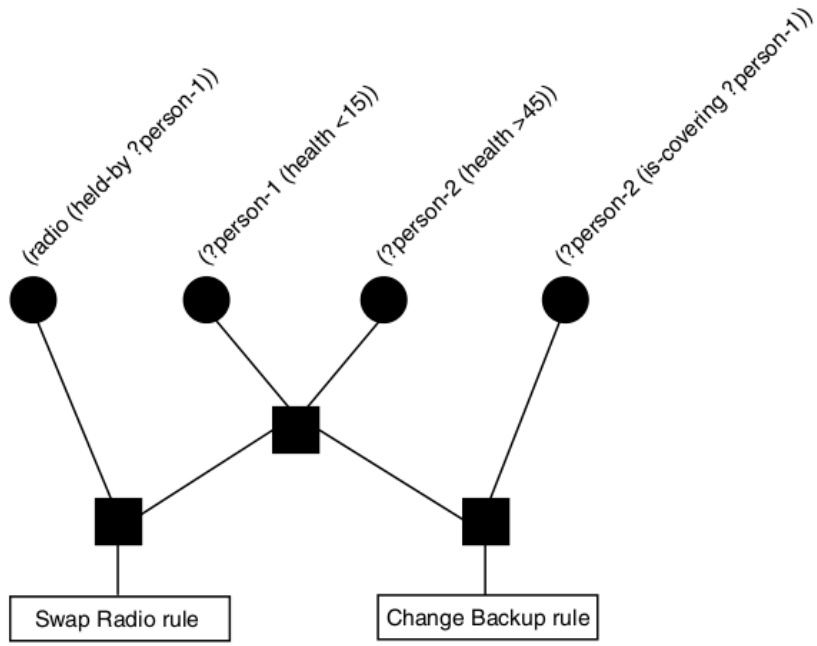
```
(Johnson (health ?value-1))
AND
(Sale (health ?value-2))
AND
?value-1 < ?value-2
```

Rete

- uses a DAG to represent the matching. represents the patterns for all rules in a single data structure: the Rete
- pattern nodes represents a single pattern in one or more rules
- at each node we also store a complete list of all the facts in the database that match that pattern
- join nodes represent the AND operation
- each path through the graph represents the complete set of patterns for one rule
- key speed features of the Rete algorithm; it doesn't duplicate matching effort.

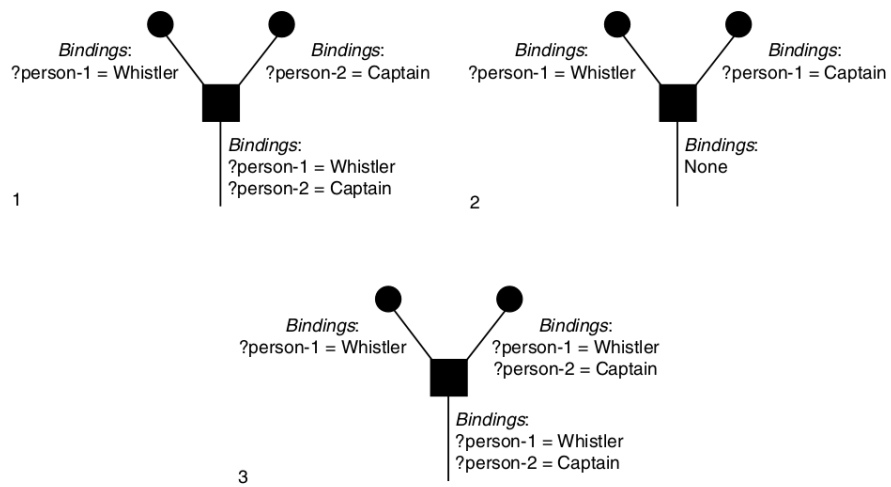
```
Swap Radio Rule:
IF
  (?person-1 (health < 15))
  AND
  (radio (held-by ?person-1))
  AND
  (?person-2 (health > 45))
THEN
  remove(radio (held-by ?person-1))
  add(radio (held-by ?person-2))

Change Backup Rule:
IF
  (?person-1 (health < 15))
  AND
  (?person-2 (health > 45))
  AND
  (?person-2 (is-covering ?person-1))
THEN
  remove(?person-2 (is-covering ?person-1)
        )
  add(?person-1 (is-covering ?person-2))
```



The algorithm:

- the database is fed into the top of the network.
- the pattern nodes try to find a match in the database: They find all the facts that match and pass them down to the join nodes. If the facts contain wild cards, the node will also pass down all the variable bindings.
- pattern nodes also keep a record of the matching facts they are given to allow incremental updating,
- join nodes makes sure that both of its inputs have matched and any variables agree



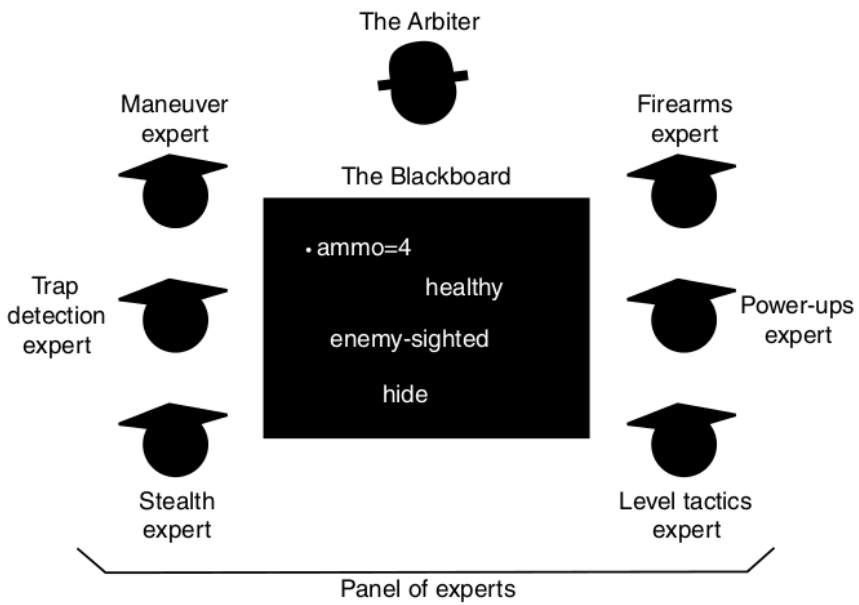
- the join node generates its own match list that contains the matching input facts it receives and a list of variable bindings.
- if multiple possible bindings, then it needs to work out all possible combinations of bindings that may be correct.

Outline

BlackBoard Architectures

- Mechanism for coordinating the actions of several decision makers.
- Each technique may be able to make suggestions as to what to do next, but the final decision can only be made if they cooperate.

  Example:
  target selector AI chooses a target, the movement AI moves into a firing position, and the ballistics AI calculates the firing solution



1. Experts look at the board and indicate their interest.

2. The arbiter selects an expert to have control (eg, by highest insistence value).

3. The expert does some work, possibly modifying the blackboard.

4. The expert voluntarily relinquishes control.

Actions are written on the blackboard and then passed to action execution

An action on the blackboard is only carried out if all relevant experts have agreed to it (just those who would be capable of finding a reason not to carry the action out)

Outline

1. Decision Making
    - Behavior Trees
    - Fuzzy Logic
    - Markov Systems
    - Goal-Oriented Behavior
    - Rule-Based Systems
    - BlackBoard Architectures