DM810

Computer Game Programming II: AI

Lecture 10

Decision Making

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Resume

- Decision trees
- State Machines
- Behavior trees
- Fuzzy logic

Outline

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 positions. Moving to position 1 implies the transition **M**
Moving to other positions would have similar matrices. If we stay the transition might increase safety

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

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

- first-order Markov processes: current vector states depend only on previous vector values.
- conservative Markov process ensures that the sum of the values in the state vector does not change over time $\rightsquigarrow$ transition matrix is a stochastic matrix
- Markov chain is a Markov process which has a discrete (finite or countable) state-space.
- time discrete Markov chains
- stationary Markov processes (or time-homogeneous): the transition matrix is the same at each step
- steady-state $\pi = \mathbf{M}\pi$ via eigenvector of the matrix
- Different transition matrices represent different events in the game, and they update the state vector accordingly.

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 (real) 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.
- in shooting games the actions give a list of motives they can satisfy
- actions can be in fact sequences of actions

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
- time to location does not belong to action:
  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
- array of world models $k + 1$
- best sequence of actions so far and its discomfort value
- 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
- It may still be necessary to split the search by an execution management in order not to compromise frame rates.

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
- but with actions there might be infinite sequences hence: 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

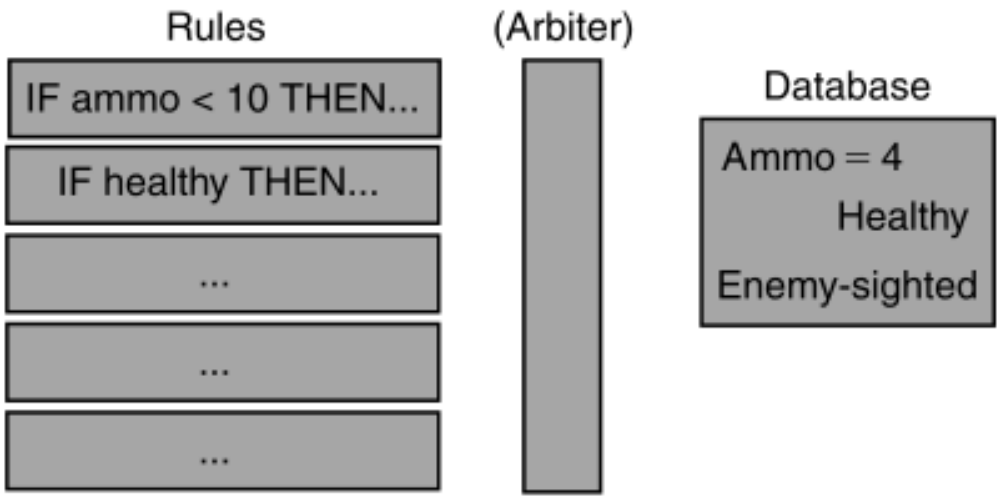
Motives may require intermediate actions to be fulfilled.

- Action-Based Signals: requires then 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
```

```
(Captain's-weapon (Rifle (Ammo 36)))
(Johnson's-weapon (Machine-Gun (Ammo 229)))
```

or

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

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))
```

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

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

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

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

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

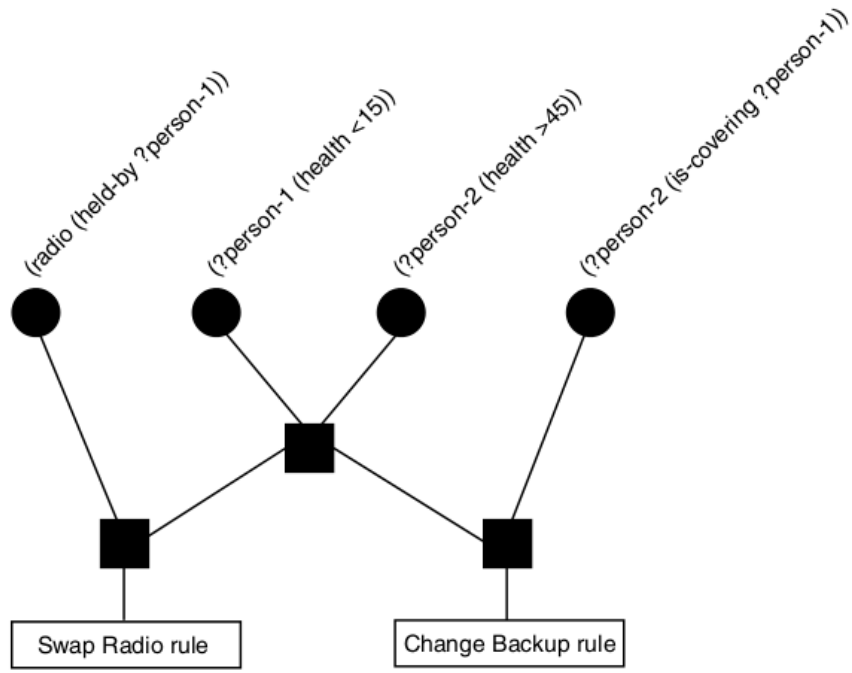
Computational issues, but unification can be done in $O(nm)$, n data, m clauses

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.

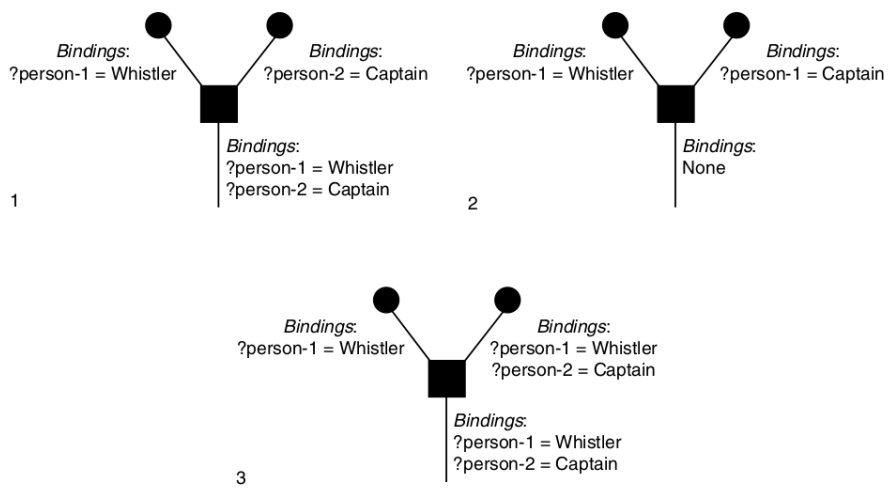
```
Swap Radio Rule:
IF
  (?person-1 (health < 15))
  AND
  (radio (held-by ?person-1))
  AND
  (?person-2 (health > 45))
THEN
  remove(radio (held-by ?person-1))
  add(radio (held-by ?person-2))

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.

Other Approaches

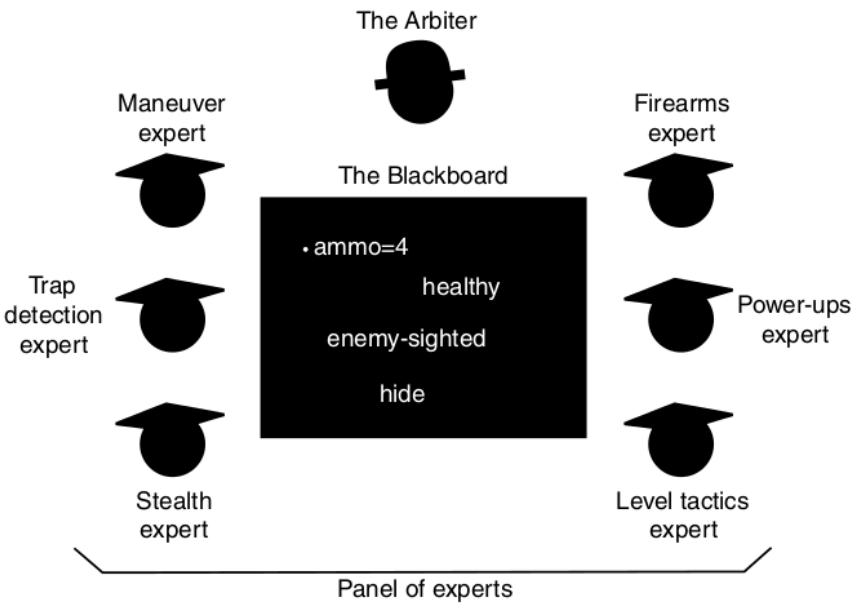
- examining ways to combine different decision makers together,
- script behaviors directly from code,
- execute actions that are requested from any decision making algorithm

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)