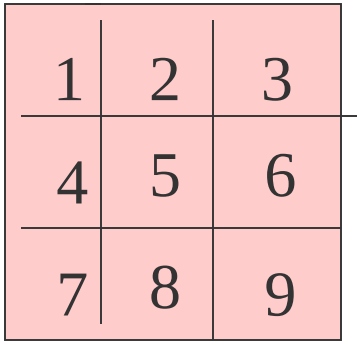
Tic-Tac-Toe

Program1

Board Position



Data Structures

Board A nine element vector representing the board.

An element contains the value 0 if corresponding square is blank, 1 if it is filled with an X or 2 if it is filled with O.

MoveTable

A large vector of 19,683 (3^9) elements, each element of which is a nine-element vector.

The Algorithm

To make a move, do the following

1. View the Vector Board as a ternary (base three) number. Convert it to a decimal number,
2. Use the number computed in step 1 as an index into the Movetable and access the vector stored there.
3. The vector selected in step 2 represents the way the board will look after the move that should be made. So set Board equal to that vector.

Disadvantages

- Takes lot of space.
- 3^9 entries are to be specified which is a difficult task and can have errors.
- Not possible to generalize from three dimension to four dimension.

The technique embodied in this program does not appear to meet any of the requirements of a good AI technique.

Program2

Data Structures

Board A nine element vector representing the board as described for program 1.

But

2 indicates blank

3 indicates X

5 indicates O

Turn An integer indicating which move of the game is about to be played.

1 indicates the first move, 9 the last.

Sub procedures

Make2 Tries to make 2 in a row

returns 5 if the center of the board is blank, that is, if Board(5) = 2

else returns non corner blank square (2,4,6 or 8)

Posswin(p)

Returns 0 if p can not win on the next move.

Otherwise returns the number of square that constitutes a winning move.

Go(n) Make a move to square n.

Board(n) = 3 if turn is odd

Board (n) = 5 if turn is even

increment turn

In odd number move, player playing with X plays

In even numbered move, player playing with O plays.

Turn1 Go(1) (upper left corner).

Turn2 If Board(5) is blank, Go(5) else Go(1).

Turn3 If Board(9) is blank, Go(9) else Go(3).

Turn4 If Posswin(X) is not 0, then Go(Posswin(X)) (i.e.block opponent's win),

else Go(Make2).

Turn5 If Posswin(X) is not 0 then Go(Posswin(X)) (i.e. win) else
if Posswin(O) is not 0 then Go(Posswin(O) (i.e. block win) else
if Board(7) is blank then Go(7) else Go(3),

Turn6 If Posswin(O) is not 0 then Go(Posswin(O) else
if Posswin(X) is not 0 then Go(Posswin(X) else
Go(Make2).

Turn7 If Posswin(X) is not 0 then Go(Posswin(X) else
if Posswin(O) is not 0 then Go(Posswin(O) else
go anywhere that is blank.

Turn8 If Posswin(O) is not 0 then Go(Posswin(O) else
if Posswin(X) is not 0 then Go(Posswin(X) else
go anywhere that is blank.

Turn9 same as Turn7.

Comments

- Less efficient in terms of time than first program since it has to check several conditions before making a move.
- More efficient in terms of space.
- Generalization of the domain's knowledge to a different domain ,such as, to three dimensional tic-tac-toe not possible.

Program2'

program identical to Program2 except for the representation of the board as follows:

8	3	4
1	5	9
6	7	2

•Numbering on the board produces a magic square. All the rows columns and diagonals add to 15. This simplifies the process of checking for a possible win.

Program3

BoardPosition

A structure containing a nine-element vector representing the board, a list of board positions that could result from the next move, and a number representing an estimate of how likely the board position is to lead to an ultimate win for the player to move.

Algorithm

Uses min-max strategy to be done later.

Advantage

•Possible to generalize

Production System

- A set of rules
- One or more databases / knowledge bases
  - Some part of the database may be invariant even though domain specific
  - some might be situation specific.
- A control strategy
  - specifies the order in which the rules will be compared to the database
  - resolves the conflicts that arise when several rules match at once.

Requirements of a good control strategy

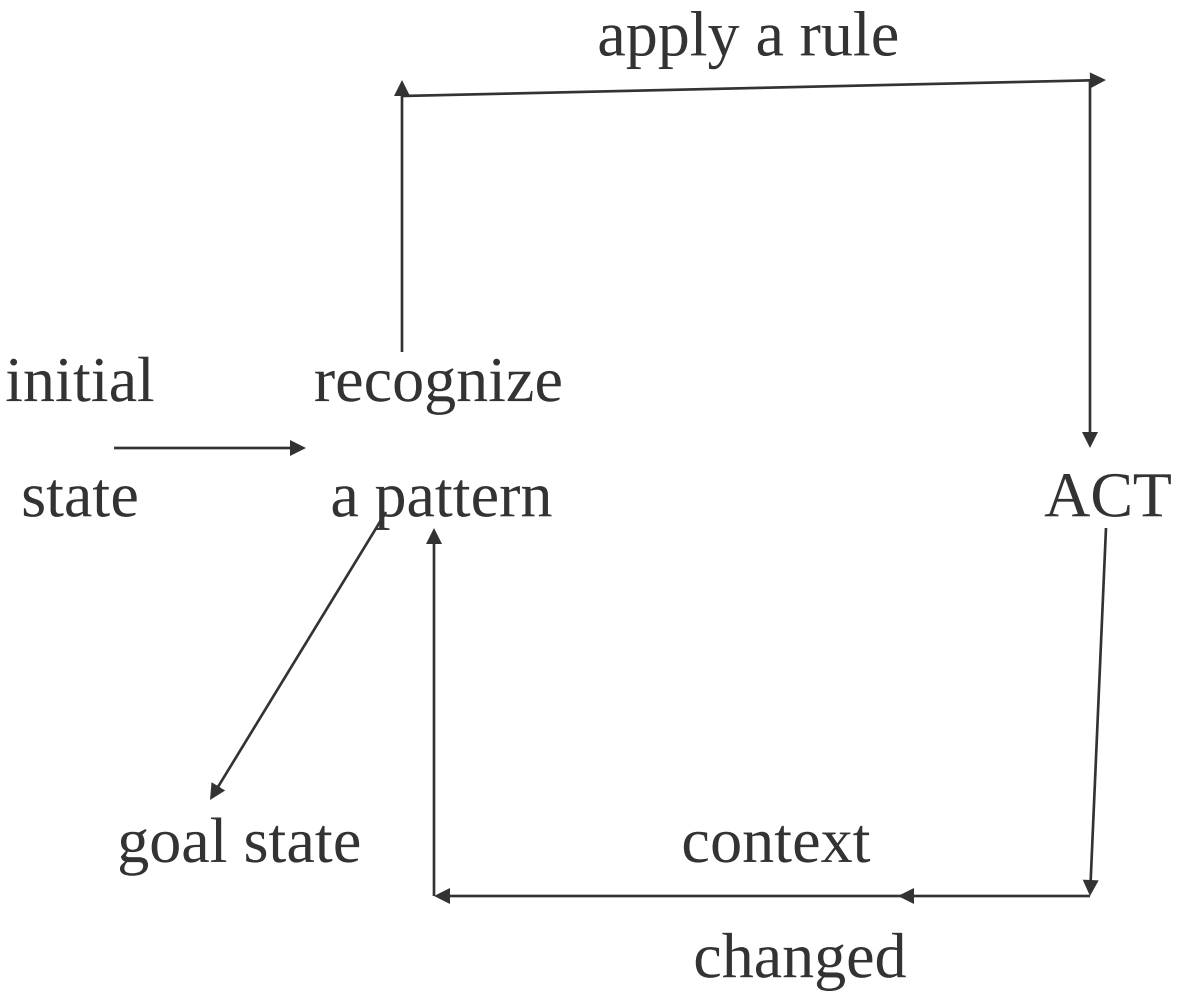
- It cause motion.

  The control strategies that do not cause motion will never lead to a solution.

- It is systematic.

  It corresponds to the need for global motion (over the course of several steps) as well as for local motion(over the course of single step).

Design for a Production System



Water-jug problem

Database

- 4 unit jug
  - full - 4
  - empty - 0
  - some - >0 , < 4
- 3 unit jug
  - full - 3
  - empty - 0
  - some - >0 , < 3

Assumptions

- unlimited supply of water
- water may be spilled
- water may be poured from one jug to the other.

Set of Rules

Let X ,Y denote the contents of the two jugs under reference.

Rule1 (X,Y | X < 4) => (4,Y)

If the jug with the 4 unit capacity has less than 4 units of water, it may be filled.

Rule2 (X,Y | Y < 3) => (X,3) similar to rule1.

Rule3 (X,Y | X > 0) => (X- D,Y)

Pour out some water from 4 unit jug.

Rule4 (X,Y | Y > 0) => (X,Y - D) similar to rule3.

Rule5 (X,Y | X > 0) => (0,Y) empty 4 unit jug

Rule6 (X,Y | Y > 0) => (X,0) empty 3 unit jug

Rule7 (X,Y | X +Y >= 4 ^ Y > 0) => (4,Y - (4 - X))

Pour from 3 unit jug to 4 unit jug to fill it.

Rule8 (X,Y | X +Y >= 3 ^ X > 0) => (X - (3 - Y),3)

Pour from 4 unit jug to 3 unit jug to fill it.

Rule9 (X,Y | X +Y <= 4 ^ Y > 0) => (X + Y ,0)

Pour all water from 3 unit jug into 4 unit jug.

Rule10 (X,Y | X +Y <= 3 ^ X > 0) => (0 , X +Y)

Pour all water from 4 unit jug into 3 unit jug.

Solution 1

jug1	jug2	Rule applied	Conflict Set
0	0	2	Rule1, Rule2
0	3	9	Rule1, Rule9
3	0	2	Rule1,2,3,4,5,6,7,8
3	3	7	
4	2	5	
0	2	9	
2	0		

With regard to the control strategy, there are two major issues

•How should one obtain the set of applicable rules?

(pattern matching of the left hand side)

•How should the conflict set be resolved?

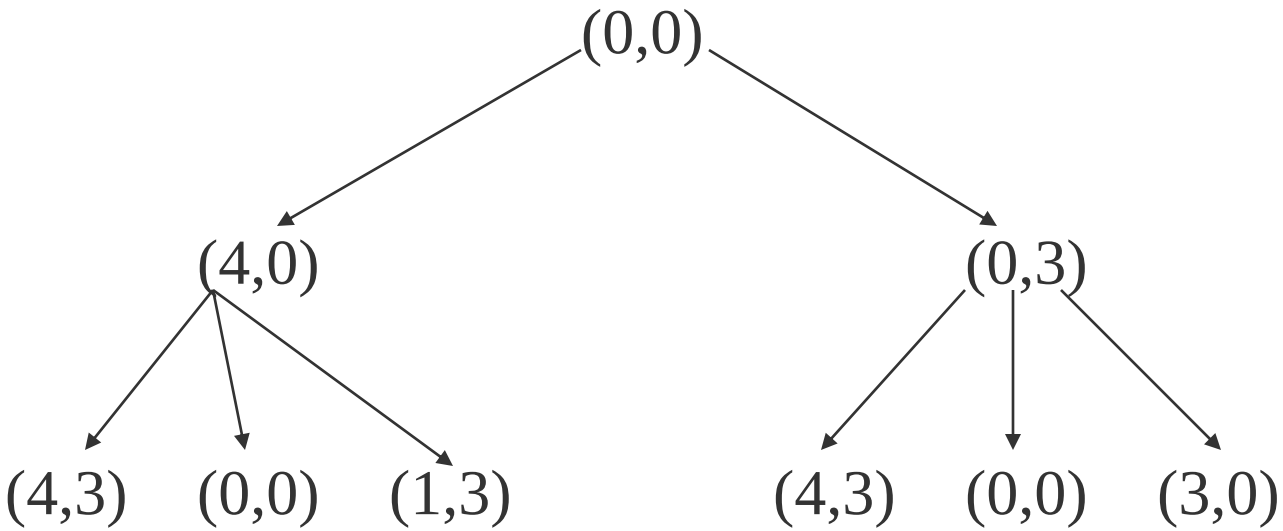
(based on the evaluation of a heuristic function)

The control strategy

must cause motion.

Should explore the solution space in a systematic manner.

A Search Tree for water-jug problem



A Search Graph for water-jug problem