DRONACHARYA College of Engineering

INTELLIGENT SYSTEMS (CSE-303-F)

Section A

Tic Tac Toe Game playing strategies

Lecture 1

Tic–Tac–Toe game playing

- Two players
  - human
  - computer.
- The objective is to write a computer program in such a way that computer wins most of the time.
- Three approaches are presented to play this game which increase in
  - Complexity
  - Use of generalization
  - Clarity of their knowledge
  - Extensibility of their approach
- These approaches will move towards being representations of what we will call AI techniques.

Tic Tac Toe Board- (or **Noughts and crosses, Xs and Os**)

It is two players, *X* and *O*, game who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three respective marks in a horizontal, vertical, or diagonal row wins the game.

1	2	3
4	5	6
7	8	9

positions

Approach 1

- Data Structure
  - Consider a Board having nine elements vector.
  - Each element will contain
    - 0 for blank
    - 1 indicating X player move
    - 2 indicating O player move
  - Computer may play as X or O player.
  - First player who so ever is always plays X.

Move Table MT

- MT is a vector of 3^9 elements, each element of which is a nine element vector representing board position.
- Total of 3^9 (19683) elements in MT

Index	Current Board position	New Board position
0	000000000	000010000
1	000000001	020000001
2	000000002	000100002
3	000000010	002000010
:		
:		

Algorithm

- To make a move, do the following:
  - View the vector (board) as a ternary number and convert it to its corresponding decimal number.
  - Use the computed number as an index into the MT and access the vector stored there.
    - The selected vector represents the way the board will look after the move.
  - Set board equal to that vector.

Comments

- Very efficient in terms of time but has several <u>disadvantages.</u>
  - Lot of space to store the move table.
  - Lot of work to specify all the entries in move table.
  - Highly error prone as the data is voluminous.
  - Poor extensibility
    - 3D tic-tac-toe = 3^{27} board position to be stored.
  - Not intelligent at all.

Approach 2

- **Data Structure**
  - **Board**: A nine-element vector representing the board: B[1..9]
  - Following conventions are used

2	-	indicates blank
3	-	X
5	-	O

  - **Turn**: An integer

1	-	First move
9	-	Last move

Procedures Used

- *Make_2* → Tries to make valid 2
  - Make_2 first tries to play in the center if free and returns 5 (square number).
  - If not possible, then it tries the various suitable non corner square and returns square number.
- ***Go(n)*** ← makes a move in square 'n' which is blank represented by 2.

Procedure - PossWin

- ***PossWin (P)*** → Returns
  - 0, if player P cannot win in its next move,
  - otherwise the number of square that constitutes a winning move for P.
- Rule
  - If PossWin (P) = 0 {P can not win} then find whether opponent can win. If so, then block it.

Strategy used by PosWin

- ***PosWin*** checks one at a time, for each rows /columns and diagonals as follows.

  - If 3 * 3 * 2 = 18 then player X can win
  - else if 5 * 5 * 2 = 50 then player O can win

- These procedures are used in the algorithm on the next slide.

Algorithm

- Assumptions
  - The first player always uses symbol X.
  - There are in all 8 moves in the worst case.
  - Computer is represented by C and Human is represented by H.
  - Convention used in algorithm on next slide
    - If C plays first (Computer plays X, Human plays O) - **Odd moves**
    - If H plays first (Human plays X, Computer plays O) - **Even moves**
    - For the sake of clarity, we use C and H.

Algo - Computer plays first – C plays odd moves

- **Move 1:** Go (5)
- **Move** 2: *H plays*
- **Move 3:** If B[9] is blank, then Go(9) else Go(3) ***{make 2}***
- **Move 4:** *H plays*
- **Move 5:** ***{By now computer has played 2 chances}***
  - If PossWin(C) then ***{won}*** Go(PossWin(C))
  - else ***{block H}*** if PossWin(H) then Go(PossWin(H)) else if B[7] is blank then Go(7) else Go(3)
- **Move 6**: *H plays*
- **Moves 7 & 9 :**
  - If PossWin(C) then ***{won}*** Go(PossWin(C))
  - else ***{block H}*** if PossWin(H) then Go(PossWin(H)) else Go(Anywhere)
- **Move 8:** *H plays*

Algo - Human plays first – C plays even moves

- **Move 1:** *H plays*
- **Move** 2: If B[5] is blank, then Go(5) else Go(1)
- **Move 3:** *H plays*
- **Move** 4: ***{By now H has played 2 chances}***
  - If PossWin(H) then ***{block H}*** Go (PossWin(H))
  - else Go (Make_2)
- **Move 5:** *H plays*
- **Move** 6: ***{By now both have played 2 chances}***
  - If PossWin(C) then ***{won}*** Go(PossWin(C))
  - else ***{block H}*** if PossWin(H) then Go(PossWin(H)) else Go(Make_2)
- **Moves 7 & 9 :** *H plays*
- **Move** 8: ***{By now computer has played 3 chances}***
  - If PossWin(C) then ***{won}*** Go(PossWin(C))
  - else ***{block H}*** if PossWin(H) then Go(PossWin(H)) else Go(Anywhere)

Complete Algorithm – Odd moves or even moves for C playing first or second

- **Move 1:** go (5)
- **Move** 2: If B[5] is blank, then Go(5) else Go(1)
- **Move 3:** If B[9] is blank, then Go(9) else Go(3) ***{make 2}***
- **Move** 4: ***{By now human (playing X) has played 2 chances}*** If PossWin(X) then ***{block H}*** Go (PossWin(X)) else Go (Make_2)
- **Move 5:** ***{By now computer has played 2 chances}*** If PossWin(X) then ***{won}*** Go(PossWin(X)) else ***{block H}*** if PossWin(O) then Go(PossWin(O)) else if B[7] is blank then Go(7) else Go(3)
- **Move** 6: ***{By now both have played 2 chances}*** If PossWin(O) then ***{won}*** Go(PossWin(O)) else ***{block H}*** if PossWin(X) then Go(PossWin(X)) else Go(Make_2)
- **Moves 7 & 9 :** ***{By now human (playing O) has played 3 chances}*** If PossWin(X) then ***{won}*** Go(PossWin(X)) else ***{block H}*** if PossWin(O) then Go(PossWin(O)) else Go(Anywhere)
- **Move** 8: ***{By now computer has played 3 chances}*** If PossWin(O) then ***{won}*** Go(PossWin(O)) else ***{block H}*** if PossWin(X) then Go(PossWin(X)) else Go(Anywhere)

Comments

- Not as efficient as first one in terms of time.
- Several conditions are checked before each move.
- It is memory efficient.
- Easier to understand & complete strategy has been determined in advance
- Still can not generalize to 3-D.

Approach 3

- Same as approach 2 except for one change in the representation of the board.
  - Board is considered to be a magic square of size 3 X 3 with 9 blocks numbered by numbers indicated by magic square.
- This representation makes process of checking for a possible win more simple.

Board Layout – Magic Square

- Board Layout as magic square. Each row, column and diagonals add to 15.

Magic Square

8	3	4
1	5	9
6	7	2

Strategy for possible win for one player

- Maintain the list of each player's blocks in which he has played.
  - Consider each pair of blocks that player owns.
  - Compute difference D between 15 and the sum of the two blocks.
  - If $D < 0$ or $D > 9$ then
    - these two blocks are not collinear and so can be ignored
    - otherwise if the block representing difference is blank (i.e., not in either list) then a move in that block will produce a win.

Working Example of algorithm

- Assume that the following lists are maintained up to 3rd move.
- Consider the magic block shown in slide 18.
  - First Player X (Human)

8	3

  - Second Player O (Computer)

5

Working – contd..

- Strategy is same as in approach 2
  - First check if computer can win.
    - If not then check if opponent can win.
    - If so, then block it and proceed further.
- Steps involved in the play are:
  - First chance, H plays in block numbered as 8
  - Next C plays in block numbered as 5
  - H plays in block numbered 3
  - Now there is a turn of computer.

Working – contd..

- Strategy by computer: Since H has played two turns and C has played only one turn, C checks if H can win or not.
  - Compute sum of blocks played by H
    - S = 8 + 3 = 11
    - Compute D = 15 – 11 = 4
    - Block 4 is a winning block for H.
    - So block this block and play in block numbered 4.
    - The list of C gets updated with block number 4 as follows:

H $\boxed{8 \quad 3}$ C $\boxed{5 \quad 4}$

Contd..

- Assume that H plays in block numbered 6.
- Now it's a turn of C.
  - C checks, if C can win as follows:
    - Compute sum of blocks played by C
    - S = 5 + 4 = 9
    - Compute D = 15 – 9 = 6
    - Block 6 is not free, so C can not win at this turn.
  - Now check if H can win.
    - Compute sum of new pairs (8, 6) and (3, 6) from the list of H
    - S = 8 + 6 = 14
    - Compute D = 15 – 14 = 1
    - Block 1 is not used by either player, so C plays in block numbered as 1

Contd..

- The updated lists at 6th move looks as follows:
  - First Player H

8	3	6

  - Second Player C

5	4	1

- Assume that now H plays in 2.
- Using same strategy, C checks its pair (5, 1) and (4, 1) and finds bock numbered as 9 {15-6 = 9}.
- Block 9 is free, so C plays in 9 and win the game.

Comments

- This program will require more time than two others as
  - it has to search a tree representing all possible move sequences before making each move.
- This approach is extensible to handle
  - 3-dimensional tic-tac-toe.
  - games more complicated than tic-tac-toe.

3D Tic Tac Toe (Magic cube)

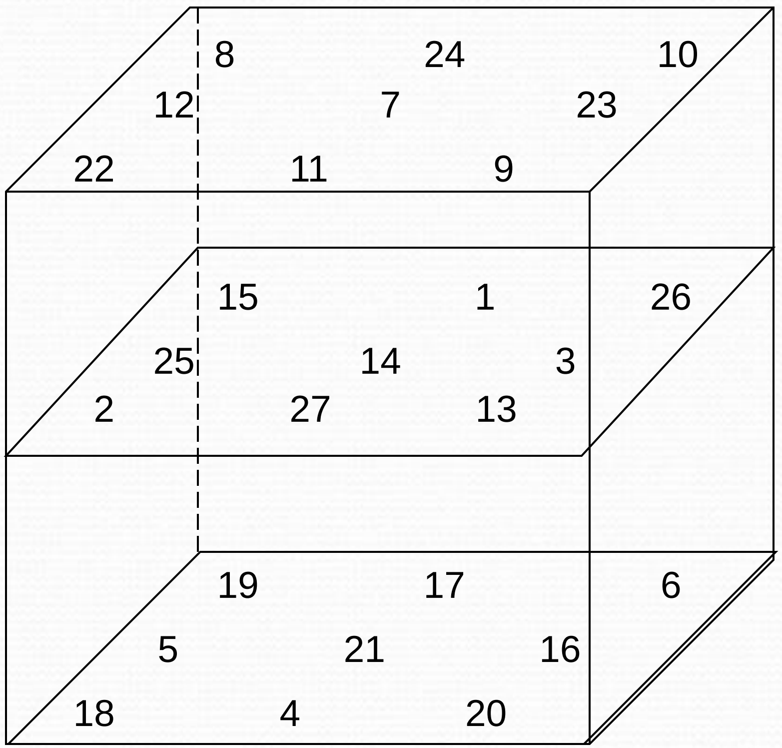
- All lines parallel to the faces of a cube, and all 4 triagonals sum correctly to 42 defined by

$$S = m(m^3 + 1)/2 \text{ , where } m=3$$

- No planar diagonals of outer surfaces sum to 42. so there are probably no magic squares in the cube.

8	24	10	15	1	26	19	17	6
12	7	23	25	14	3	5	21	16
22	11	9	2	27	13	18	4	20

8	24	10		15	1	26		19	17	6
12	7	23		25	14	3		5	21	16
22	11	9		2	27	13		18	4	20



- Magic Cube has 6 outer and 3 inner and 2 diagonal surfaces
- Outer 6 surfaces are not magic squares as diagonals are not added to 42.
- Inner 5 surfaces are magic square.