Non-AI and AI Techniques

- Non-AI techniques involve a series of instructions that are followed when a trigger is encountered. An information system that is built using decision-making techniques other than AI that include procedural or declarative knowledge. Procedural experience maps the process of decision making into the processes by which the issues are solved, or decisions are made.

- AI techniques are models built from sophisticated elements of the computational and mathematical models. Such models allow a computer or machine to measure tasks that are supposed to be performed by humans. E.g., Heuristics, Support Vector Machines, Artificial Neural Networks, Markov Decision Process, Natural Language Processing etc.

- In technical terms, An algorithm (Non-AI) is a set of instructions — a hardcoded recipe that the system executes line by line when it is triggered.

- AI on the other hand — which is trying to give "common sense" to computers that we humans use in our day to day life — is a group of algorithms (smart) that modify the present algorithm based on the previous inputs and the data processed using the inputs in an attempt to reduce the space and time complexity. Here the term "intelligence" is used to show the ability of the computer to change, adapt and grow based on new data.

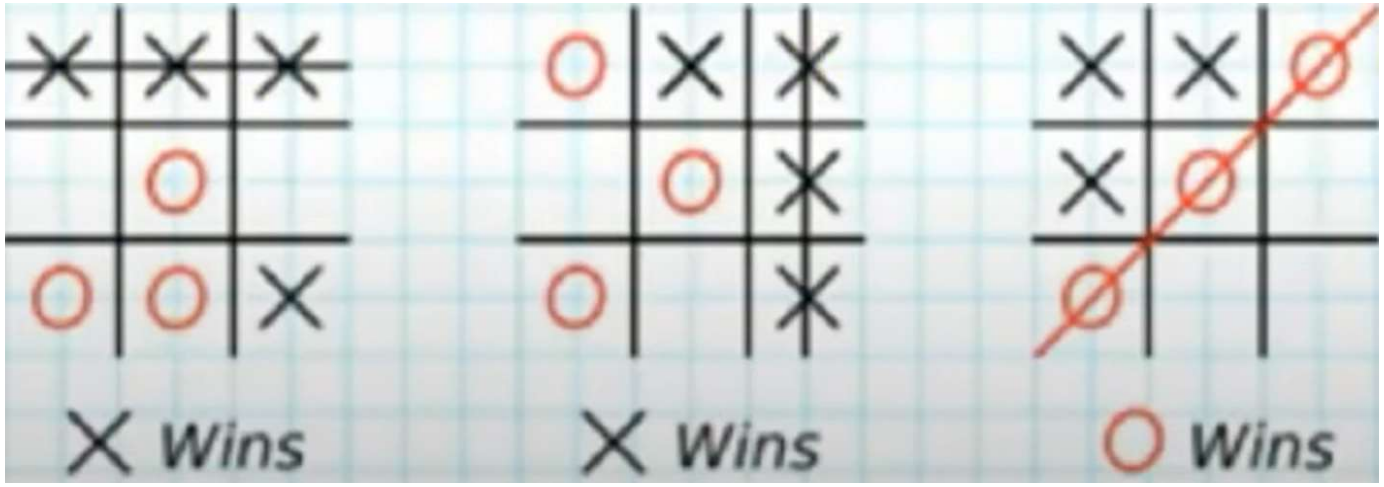
- The bottom line, non-AI technique is when the system keeps on following the defined set of rules to reach the solution and AI is when the system learns from its past, overcomes its mistakes and gives more optimal solutions to the problems.

Non-AI and AI Techniques (cont..)

Non-AI Machines	AI Machines
Smart machines which are not AI, do not require training data, they work on algorithms only.	AI machines are trained with data and algorithm.
Smart machines work on fixed algorithms and they always work with the same level of efficiency, which is programmed into them.	AI machines learn from mistakes and experience. They try to improvise on their next iterations.
Machines which are not AI cannot take decisions on their own.	AI machines can analyze the situation and can take decisions accordingly.
An automatic door in a shopping mall, seems to be AI-enabled, but it is built with only sensor technology	AI based drones capture the real-time data during the flight, processes it in real-time, and makes a humanin dependent decision based on the processed data.

Example: Tic-Tac-Toe

- Tic-Tac-Toe is a paper-pencil game of two-players **X** and **O**, who selects to marks a spaces on 3x3 grid.
- A player who succeeds in putting a 3 of their marks in a horizontal, vertical or diagonal line is a winner of the game.



Tic-Tac-Toe: Program 1 (Non-AI Technique)

- Data Structure BOARD: The Tic-Tac-Toe game consists of a nine-element vector called BOARD; it represents the numbers 1 to 9 in three rows. An element contains the value 0 for blank, 1 for X and 2 for O.

1	2	3
4	5	6
7	8	9

2-D Board

1	2	3	4	5	6	7	8	9

1-D Vector BOARD representing a board

- Data Structure MOVETABLE: A MOVETABLE vector consists of 19,683 elements (3^9) and is needed where each element is a nine-element vector. The contents of the vector are especially chosen to help the algorithm.

Index	Current Board Position	Next Board Position
0	000000000	000010000
1	000000001	020000001
2	000000002	000100002
3	000000010	002000010
:		
19682		

Tic-Tac-Toe: Program 1 (cont..)

- The algorithm: (Assumption: 1st player is X.)
  1. View the vector as a ternary number. Convert it to a decimal number.
  2. Use the decimal number as an index in MOVETABLE and access the vector.
  3. Set BOARD to this vector indicating how the board looks after the move.

- This approach is capable in time, but it has several disadvantages.
- Disadvantages:
  - It takes more space to store MOVETABLE.
  - It has a lot of work to specify all entries in MOVETABLE.
  - It requires stunning effort to calculate the decimal numbers.
  - It is highly error-prone due to huge volume of data.
  - This method is specific to this 2-D game and cannot be completed for more dimensions like 3-D for which 3^{27} board positions would have to be stored.

Tic-Tac-Toe: Program 2

- Data Structure BOARD: The structure of the data is as before but we use 2 for a blank, 3 for an X and 5 for an O.
- Data Structure TURN: An integer variable called TURN indicates 1 for the first move and 9 for the last.

- The algorithm consists of three sub procedures:
  - MAKE2: Returns 5 if the center square is blank(i.e., if BOARD[5]=2, then return 5); otherwise, it returns any blank non corner square, i.e., 2, 4, 6 or 8.

1	2	3
4	5	6
7	8	9

  - POSSWIN(p): Returns 0 if player p cannot win on the next move and otherwise returns the number of the square that gives a winning move. It operates by checking, one at a time, each of the rows, columns and diagonals. It checks each line using products of values of its squares together. A product 3*3*2 = 18 gives a win for X, 5*5*2=50 gives a win for O, and the winning move is the holder of the blank.

  - GO(n): Makes a move to square n. It sets BOARD[n] to 3 if TURN is odd or 5 if TURN is even. It also increments TURN by 1.

Tic-Tac-Toe: Program 2 (cont..)

- The algorithm has a built-in-strategy for each move it may have to make. It makes odd-numbered moves if it is playing X and even-numbered moves if it is playing O. The strategy for each turn is as follows:
  - TURN=1: Go(1) (Upper left corner).
  - TURN=2: If Board[5] =2(blank), Go(5), else Go(1).
  - TURN=3: If Board[9] =2(blank), Go(9), else Go(3).
  - TURN=4: If POSSWIN(X)≠0, then Go(POSSWIN(X)) [i.e. block opponent's win], else Go(Make2).
  - TURN=5: If POSSWIN(X)≠0, then Go(POSSWIN(X)) [i.e. win], else if POSSWIN(O)≠0, then Go(POSSWIN(O)) [i.e. block opponent's win], else if BOARD[7] =2(blank), Go(7), else Go(3). [Here, program is trying to make a fork.]
  - TURN=6: If POSSWIN(O)≠0, then Go(POSSWIN(O)), else if POSSWIN(X)≠0, then Go(POSSWIN(X)), else Go(Make2).
  - TURN=7: If POSSWIN(X)≠0, then Go(POSSWIN(X)), else if POSSWIN(O)≠0, then Go(POSSWIN(O)), else go anywhere that is blank.
  - TURN=8: If POSSWIN(O)≠0, then Go(POSSWIN(O)), else if POSSWIN(X)≠0, then Go(POSSWIN(X)), else go anywhere that is blank.
  - TURN=9: Same as TURN 7.
- This algorithm checks several conditions before making each move so, takes longer time but it is more efficient in storage which compensates for its longer time. It depends on the programmer's skill. Also, it cannot be generalized to different domains like 3-D Tic-Tac-Toe.

Tic-Tac-Toe: Program 2’

- It is identical to Program 2 except for the one change in the board representation.
- Data Structure BOARD: The numbering of the BOARD produces a magic square: all row, columns and diagonals sum up to 15.

8	3	4
1	5	9
6	7	2

- It simplifies the process of checking of a possible win.
- The algorithm:
  - In addition to marking the BOARD as moves are made, keep a list for each player, of squares in which he/she has played.
  - To check a possible win for one player, consider each pair of squares owned by that player and compute the difference between 15 and sum of the two squares. If the difference is < 0 (negative) or > 9, then the original two squares are not collinear and so can be ignored.
  - Otherwise, if the square representing the difference is blank, a move there will produce win
  - or check if the opponent is winning, block his/her win.
- Since, no player can have more than 4 squares at a time, there will be many fewer squares examined using this scheme than using approach in Program 2.
- This shows how the choice of representation can have impact on the efficiency on problem-solving program.

Tic-Tac-Toe: Program 2’

- E.g. BORAD

8	3	4
1	5	9
6	7	2

TURN1

	X	

TURN2

O		
	X	

Turn3

O		X
	X	

TURN4

O		X
	X	
O		

TURN5

O		X
X	X	
O		

TURN6

O		X
X	X	
O		

TURN7

O		X
X	X	O
O		

and so on….

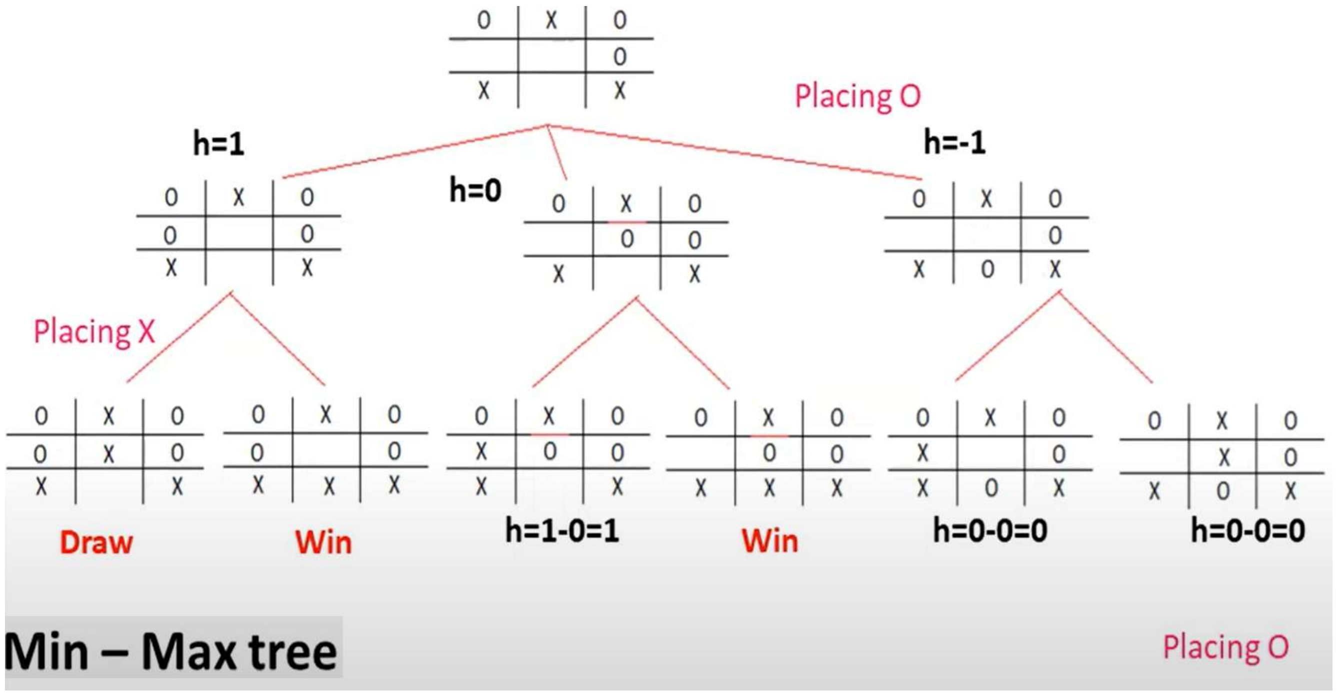
Tic-Tac-Toe: Program 3 (AI Technique)

- Data Structure BOARDPOSITION: The structure contains a nine-element vector, a list of board positions that could result from the next move and a number representing an estimation of how the board position leads to an ultimate win for the player to move.

- The algorithm looks ahead to make a decision on the next move by deciding which the most promising move or the most suitable move at any stage would be, selects the same and assign the rating of the best move to the current position.
- To decide which of a set of BOARD positions is best, do the following for each of them:
  - See if it is a win, If so, call the best by giving it the highest possible rating.
  - Otherwise, consider all the moves the opponent could make next. See which of them is the worst for us(by recursively calling this procedure). Assume the opponent will make that move. Whatever rating that move has , assign it to the node we are considering.
  - The best node is then the one with the highest rating.
- The algorithm will look ahead all possible moves to find out a sequence that leads to a win , if possible, in the shortest time.
- It attempts to maximize a likelihood of winning, while assuming that opponent will try to minimize that likelihood.

Tic-Tac-Toe: Program 3 (cont..)

E.g.
Heuristics Function H= P(X wins) - P(O wins)
H=0 is DRAW.
X chooses Maximum value of H.
O chooses Minimum value of H.



- Actually, this is most difficult to program by a good limit as it will need more time and computations to search a tree representing all possible move sequences prior to each move, but it is superior to other programs as this technique can be extended to in any game.
- It can be augmented by a variety of specific kinds of knowledge of games and reduce number searches by applying simple and reasonable algorithms.
- This method makes relatively fewer loads on the programmer in terms of the game technique, but the overall game strategy must be known to the adviser.