Implementation of Minimax Search using Tic-Tac-Toe as an example domain

Minimax is an algorithm for decision rule used in the decision theory, game theory, statistics and philosophy for minimizing the possible loss for a worst case (maximum loss) scenario for any player. Originally formulated for two-player game theory, covering both the cases where players take alternate, it has also been extended to more complex games and to general decision-making in the presence of uncertainty about the future possibilities.

The Minimax algorithm is an algorithm for determining the optimal game strategy for finite two-person zero-sum games with perfect information about each and every step. These games are especially board games like Chess, tic tac toe in which both players always know the entire history of the game. Even for games with random influence as Backgammon allows the Minimax algorithm based on expected values ​​expand. Typically, but not exclusively, the minimax algorithm is applied to games with alternate train right.

A calculated value with the Minimax algorithm strategy is minimax strategy called utility value. It assures the player concerned the maximum winnning opportunities which can be achieved regardless of the playing style of the opponent player. This is formed from the minimax strategies of both players strategy pair forms a Nash equilibrium.

For non-zero-sum games, in which the defeat of the enemy coincides not necessarily with their own profit, the Minimax algorithm does not necessarily provide an optimal strategy.

Variants of the Minimax algorithm form the core element of gambling programs like a chess program. The increasing processing power of computers has now meant that even in such complex games like chess, most people can be beaten by the computer without any trouble now.

In game theory, a game tree is a directed graph whose nodes are positions in a game and whose edges are theie corresponding moves. The complete game tree for a game is the game tree starting at the initial position and containing all possible moves from each position. The complete tree is the same tree as that obtained from the extensive-form game representation graphically.

Initially, in our program we give a choice to the player to choose whom to start the game first. The computer takes the part of O and the player X.

-----Welcome to Tic Tac Toe game:------

You : X

Computer : O

Select the options:

1.You will start.

2.I will start

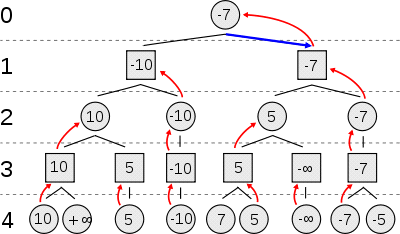
After giving the input "1" it asks for the coordinates from the player.

1

Enter the coordinates where you want to place X:

1 1

Then this move is placed on the board and the "applyMinimax Algorithm()" is called which creates the child nodes and displays it as it traverses through the tree.



The picture shows a simple tree with search depth 4. Player A is on the train.

The nodes of the levels 0 and 2 correspond to game situations where player A at the train. Here, the evaluation function of the child nodes of the profit for player A train will each maximized H. Selected and assigned the value of the parent node.

The nodes of level 1 and 3 correspond to game situations where player B on the train. Here, the evaluation function of the child nodes of the cheapest for player B train is respectively minimized, d. H. Selected and assigned the value of the parent node.

The algorithm starts at the bottom with the leaves, and then goes up to the root. In Level 3, the algorithm selects the minimum value of child nodes and assigns the parent node to minimize it (it is minimized). In Level 2 the largest respective child node is then the parent node assigned to maximize it (it is maximized). This is performed alternately until the root is reached. The root is assigned the value of the biggest child node. It then is the move, which is to be made.

Board:

-------

|X|X|O|

|O|X|X|

|O|X|O|

-------

Board:

-------

|X|-|O|

|O|X|X|

|O|X|O|

-------

Board:

-------

|X|-|-|

|O|X|X|

|O|X|O|

-------

Board:

-------

|-|X|-|

|O|X|X|

|O|X|O|

-------

Board:

-------

|O|-|X|

|O|X|X|

|O|X|O|

-------

Board:

-------

|X|O|X|

|O|X|X|

|O|X|O|

-------

Board:

-------

|-|O|X|

|O|X|X|

|O|X|O|

-------

Board:

-------

|-|-|X|

|O|X|X|

|O|X|O|

-------

Board:

-------

|-|-|-|

|O|X|X|

|O|X|O|

-------

Board:

-------

|-|-|-|

|-|X|X|

|O|X|O|

-------

Board:

-------

|-|-|-|

|-|X|-|

|O|X|O|

-------

Board:

-------

|-|-|-|

|-|X|-|

|-|X|O|

-------

Board:

-------

|-|-|-|

|-|X|-|

|-|-|O|

-------

The time taken for this step is:4157

The number of child nodes for this particular user input is55504

The current board after placing computers' move at:(0 0)

Board:

-------

|O|-|-|

|-|X|-|

|-|-|-|

-------

Enter the coordinates where you want to place X:

Also, the time taken for the computer to create these child nodes are printed using the:

**long** startTime = System.*currentTimeMillis*();

n=*ApplyMinMaxAlgorithm*(1,b);

**long** finishTime = System.*currentTimeMillis*();

**long** elapsedTime = finishTime - startTime;

The minimax algorithm is :

**public** **static** **int** ApplyMinMaxAlgorithm(**int** NextPlayer,TicTacToeBoard b)

{

**int** Score=0;

List<Coordinates> EmptyCoordinates=**new** ArrayList<>();

List<Integer> PointsAcquired=**new** ArrayList<>();

EmptyCoordinates=*FindEmptyCoordinates*(b);

**if**(*HasComputerWon*(b))

{

**return** 1;

}

**if**(*HasHumanWon*(b))

{

**return** -1;

}

**if**(*HasDraw*(b))

{

**return** 0;

}

**for**(**int** i=0;i<EmptyCoordinates.size();i++)

{

Coordinates C=**new** Coordinates(EmptyCoordinates.get(i));

TicTacToeBoard b1=**new** TicTacToeBoard();

*CopyBoard*(b,b1);

**if**(NextPlayer==1)

{

*EditBoard*(b1,C,NextPlayer);

**int** lowest=*ApplyMinMaxAlgorithm*(2,b1);

PointsAcquired.add(lowest);

Collections.*sort*(PointsAcquired, Collections.*reverseOrder*());

Score=PointsAcquired.get(0);

++*count*;

*DisplayBoard*(b1);

}

**if**(NextPlayer==2)

{

*EditBoard*(b1,C,NextPlayer);

**int** highest=*ApplyMinMaxAlgorithm*(1,b1);

PointsAcquired.add(highest);

Collections.*sort*(PointsAcquired);

Score=PointsAcquired.get(0);

++*count*;

*DisplayBoard*(b1);

}

b.CoordinateClass.add(**new** CoordinateClass(C,Score));

}

**return** Score;

}

This makes use of various functions to check whether the current board position is won by anubody or not by checking diagonally, ertically and horizontaly.

**public** **static** **int** HasAnybodyWon(TicTacToeBoard b)

{

**if**(*HasComputerWon*(b))

**return** 0;

**if**(*HasHumanWon*(b))

**return** 0;

**if**(*HasDraw*(b))

**return** 0;

**return** -1;

}

**public** **static** **boolean** HasComputerWon(TicTacToeBoard b)

{

**for**(**int** i=0;i<3;i++)

**for**(**int** j=0;j<3;j++)

**if** ((b.Contents[0][0] == b.Contents[1][1] && b.Contents[0][0] == b.Contents[2][2] && b.Contents[0][0] == 1) || (b.Contents[0][2] == b.Contents[1][1] && b.Contents[0][2] == b.Contents[2][0] && b.Contents[0][2] == 1))

{

**return** **true**;

}

**for** (**int** i = 0; i < 3; ++i)

{

**if** (((b.Contents[i][0] == b.Contents[i][1] && b.Contents[i][0] == b.Contents[i][2] && b.Contents[i][0] == 1)|| (b.Contents[0][i] == b.Contents[1][i] && b.Contents[0][i] == b.Contents[2][i] && b.Contents[0][i] == 1)))

{

**return** **true**;

}

}

**return** **false**;

}

**public** **static** **boolean** HasHumanWon(TicTacToeBoard b)

{

**for**(**int** i=0;i<3;i++)

**for**(**int** j=0;j<3;j++)

**if** ((b.Contents[0][0] == b.Contents[1][1] && b.Contents[0][0] == b.Contents[2][2] && b.Contents[0][0] == 2)|| (b.Contents[0][2] == b.Contents[1][1] && b.Contents[0][2] == b.Contents[2][0] && b.Contents[0][2] == 2))

{

**return** **true**;

}

**for** (**int** i = 0; i < 3; ++i)

{

**if** (((b.Contents[i][0] == b.Contents[i][1] && b.Contents[i][0] == b.Contents[i][2] && b.Contents[i][0] == 2)|| (b.Contents[0][i] == b.Contents[1][i] && b.Contents[0][i] == b.Contents[2][i] && b.Contents[0][i] == 2)))

{

**return** **true**;

}

}

**return** **false**;

}

**public** **static** **boolean** HasDraw(TicTacToeBoard b)

{

**for**(**int** i=0;i<3;i++)

**for**(**int** j=0;j<3;j++)

{

**if**(b.Contents[i][j]==0)

**return** **false**;

}

**return** **true**;

}

An ideal evaluation function assigns a position the value +1 to when player A wins, and the value -1 if player B wins, and 0 in a draw. Can one of all game positions the search tree up to the maximum depth to build (up to the end position, where you can see who wins), the algorithm makes a perfect match in all cases. However, in practice the complete structure of a search tree only for very simple games like is Tic-Tac-Toe is possible.

In almost all other games, this is too computationally expensive. Therefore, one is content to construct the search tree only to a search level (horizon). The evaluation function is modified, very good match for positions obtained very high values, very good match for positions B get very low scores. To determine the values ​​one uses heuristics to estimate.

All the scores calculated will be stored in an object of class "b.CoordinateClass" along with the coordinates on the tic tac toe board. It takes the coordinate which will minimize the players' points and places its move at that particular point. The computer tries ti minimize the players' points and the player inturn will be trying to maximize his points.

**for** (**int** i=0;i<b.CoordinateClass.size(); i++)

{

**if** (n < b.CoordinateClass.get(i).Score)

{

n=b.CoordinateClass.get(i).Score;

efficient=i;

}

}

b.Contents[NewC.x][NewC.y]=1;

System.***out***.println("\nThe current board after placing computers' move at:("+NewC.x+" "+NewC.y+")");

*DisplayBoard*(b);

In this way the game moves on until all the positions on the board are filled resulting in either a win for the computer or numan,or a draw match.

The time taken for this step is:0

The number of child nodes for this particular user input is3

The current board after placing computers' move at:(0 1)

Board:

-------

|O|O|O|

|X|X|O|

|X|-|X|

-------

Yayy..!!! I won.

Wanna play another game???

1. Yes

2. No

Board:

-------

|O|X|O|

|O|X|X|

|X|O|X|

-------

The time taken for this step is:0

The number of child nodes for this particular user input is0

Match is a draw! Nobody wins...

Wanna play another game???

1. Yes

2. No

The corresponding code for the above mechanism is:

**if** (*HasComputerWon*(b))

{

System.***out***.println("Yayy..!!! I won.");

}

**else** **if** (*HasHumanWon*(b))

{

System.***out***.println("Winner:Human");

}

**else** **if**(*HasDraw*(b))

{

System.***out***.println("Match is a draw! Nobody wins... ");

}

}