MODULE 4

****

SYLLABUS

Module 4 (6Hrs 15%Marks)

Advanced Search: Heuristics in Games, Design of good heuristic-an example.
Min-Max Search Procedure, Alpha Beta pruning,

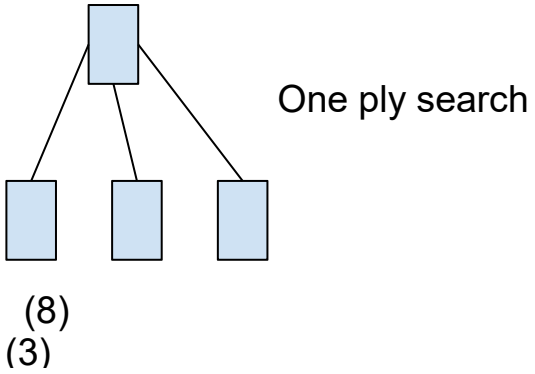
****

GAME PLAYING

In many environments, there are multiple agents. In this, a given agent must consider the actions of other agents. If the agents are competing with each other, then such an environment is called a competitive environment. Competitive environments in which the agents goals are in conflict, results in problems known as games.
Consider the game of chess. If one player wins the game of chess, then utility function value is +1. in this, the opposite player loses. His utility function value is -1. if the game ends in a draw, the utility function value is 0. Like this, in AI, games are turn taking, 2 player, zero sum games.
For AI researchers, the nature of games makes them an attractive subject for study. The state of a game is easy to represent and agents are limited to a small number of actions. Precise rules are there for making these actions. By 1950, chess playing program was developed by Zuse, Shannon, Wiener and Turing. After that, systems for playing checkers, Othello, Backgammon and Go were developed. Games are interesting because they are too hard to solve. For example, chess has around 35100 states. It is impossible for a computer to generate all these states. Therefore game playing research has brought a number of interesting ideas on how to make the best possible use of time.

The MinMax Search Procedure

The MinMax search procedure is a **depth first, depth-limited** search procedure. The idea is to start at the current position and use the **plausible-move generator to generate** the set of possible successor position. Now we can apply the **static evaluation function** to those functions and simply choose the best one. After doing so we can back that value upto the starting position to represent our evaluation of it. The starting position is exactly as good for us as the position generated by the best move we can make next. Here we assume that the static evaluation function returns large values to indicate good situations for us, so our goal is to maximize the value of the static evaluation function of the next board position.



It assumes a static evaluation function that returns values ranging from -10 to 10, with 10 indicating win for us, -10 indicating win for opponent and 0 indicates even match.

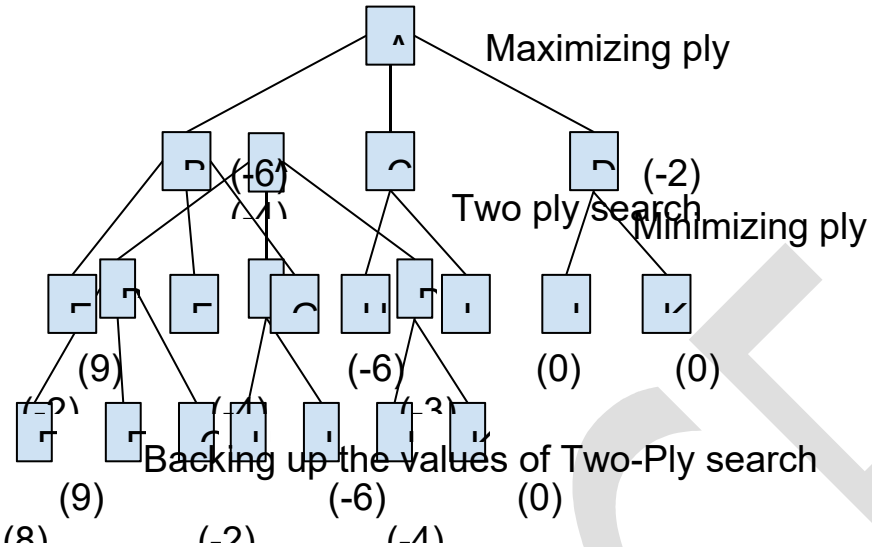
Since our goal is to maximize the value of heuristic function, we choose to move to B. Backing B's value upto A, we can conclude that A's value is 8, since we know we can move to a position with a value of 8.

The static evaluation function is not completely accurate, we would like to carry the search further ahead than One Ply. For eg, in a chess game in which we are in the middle of piece exchange. After our move, the situation would appear to be very good but if we look one move ahead we will see that our pieces also gets captured and so the situation is not as favourable as it seemed. So we would like to look ahead to see what will happen to each of the new game position at the next move which will be made by opponent. Instead of applying the static evaluation function to each of the positions that we just generated we apply the plausible move generator generating a set of successor positions for each position.

If we want to stop at two ply lookahead, we could apply the static evaluation function to each of these positions. But now we must take into account that the opponent gets to choose which successor moves to make and thus which terminal value should be backed up to next level.

Suppose we made move B then the opponent must choose among moves E, F,G. The opponents goal is to minimize the value of valuation function, so he or she can be expected to choose move F. this means that if we make move B, the actual position in which we will end up one move later i very bad for us. This is true even though a possible configuration is that represented by node E, which is very good for us. But at

this level we are not ones to move we will not get to choose it.



Result of Propagating new values up the tree

At the level representing the opponents choice the minimum value was chosen and backed up, At the level representing our choice, the maximum value was chosen.

Once the value from the second ply are backed up, it become clear that the correct move for us to make at first level is C(-2), since there is nothing the opponent can do from there to produce a value worse than -2. this process can be repeated for as many ply as time allows and more accurate evaluations that are produced can be used to choose the correct move at the top level.

The alteration of maximizing and minimizing at alternative ply when evaluations are being pushed back up corresponds to the opposing strategies of the two players and gives this method the name MINMAX.

Two auxiliary procedures that are specific to game being played.

1. **MOVEGEN(Position,Player):** the plausible- move generator, which returns a list of nodes representing the moves that can be made by Player in Position. Eg, in Chess Program we call two players PLAYER-ONE and PLAYER-TWO
2. **STATIC(Position, Player):** the static evaluation function, which returns a number representing the goodness of position from the standpoint of player.

A critical issue in the design of the MINMAX procedure is when to stop the recursion and simply call the static evaluation function.

Factors that influence this decision

- Has one side won?
- How many ply have we already explored?
- How promising is the path?
- How much time is left?
- How stable is the configuration?

DEEP_ENOUGH(Position, Depth)

Which is assumed to evaluate all of these factors and to return **TRUE if the search should be stopped at the current level** and FALSE if otherwise. It will take two parameters, Position and Depth. It will ignore its position parameter and simply return TRUE if its depth parameter exceeds a constant cutoff value.

Two Results (VALUE, PATH)

- The backed up value of the path it choose
- the path itself

Initial Call

Three parameters

- a board position
- the current depth of the search and
- player to move

PLAYER ONE -> MINMAX(CURRENT,0,PLAYER_ONE)

PLAYER TWO -> MINMAX(CURRENT,0,PLAYER_TWO)

Algorithm: MINMAX(Position, Depth, Player)

1. If DEEP_ENOUGH(Position, Depth) then return the structure
   VALUE=STATIC(Position, Player)

   PATH=NIL

   This indicate that there is no path from this node and that its value is that determined by static evaluation function

2. Otherwise, generate one more ply of the tree by calling the function MOVE_GEN(Position, Player) and setting SUCCESSORS to the list it returns
3. If the SUCCESSORS is empty, then there are no moves to be made so return the same structure that would have been returned if DEEP_ENOUGH had returned true
4. If SUCCESSORS is not empty, then examine each element in turn and keep track of the best one. This is done as follows. Initialize the BEST_SCORE to minimum value that the STATIC can return. It will be updated to reflect the best score that can be achieved by an element of SUCCESSORS. For each element do the following
   a. Set RESULT_SUCC to
      MINMAX(SUCC, Depth+1,OPPOSITE(Player))

      This recursive call to MINMAX will actually carry out exploration of SUCC

   b. Set NEW_VALUE to -VALUE(RESULT_SUCC). This will cause it to reflect the merits of the position from the opposite perspective from that of the next lower level.
   c. If the NEW_VALUE>BEST_SCORE then we have found a successor that is better than any that have been examined so far. Record this by doing the following:
      i. Set BEST_SCORE to NEW_VALUE

ii. The best known path is now from CURRENT to SUCC and then on to the appropriate path down from SUCC as determined by recursive call to MINMAX. So set BEST_PATH to result of attaching SUCC to the front of PATH(RESULT_SUCC)

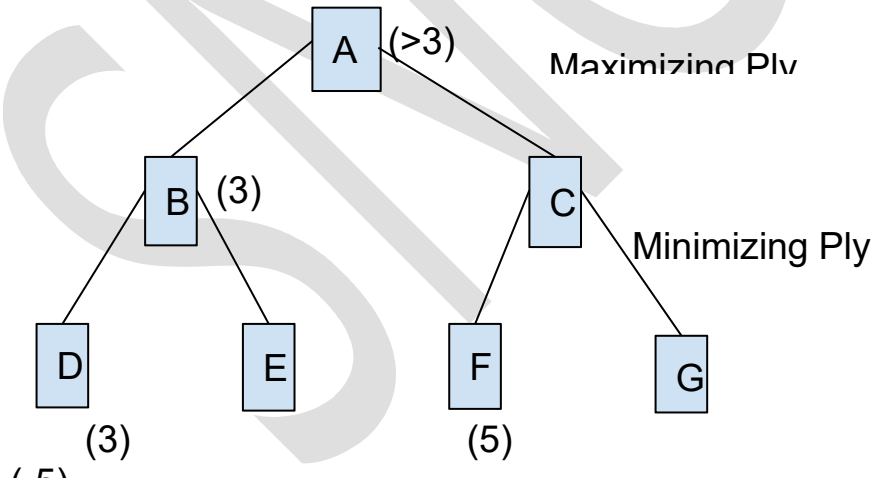
5. Now that all the successors have been examined we know the value of the Position as well as which path to take from it. So return the structure. VALUE=BEST_SCORE PATH=BEST_PATH

When the initial call to MINMAX returns, the best move from CURRENT is the first element on PATH. This MINMAX procedure is so simple but its performance can be improved by a few refinements.

Adding ALPHA-BETA Cut-Offs

The MINMAX procedure is a depth first search process. One path is explored as far as time allows, the static evaluation function is applied to the game positions at the last step of the path, and value can then be passed up the path one level at a time. One of the good things about depth first search is that its efficiency can be improved by using branch-and-bound techniques in which partial solution that are clearly worse than known solution can be abandoned early.

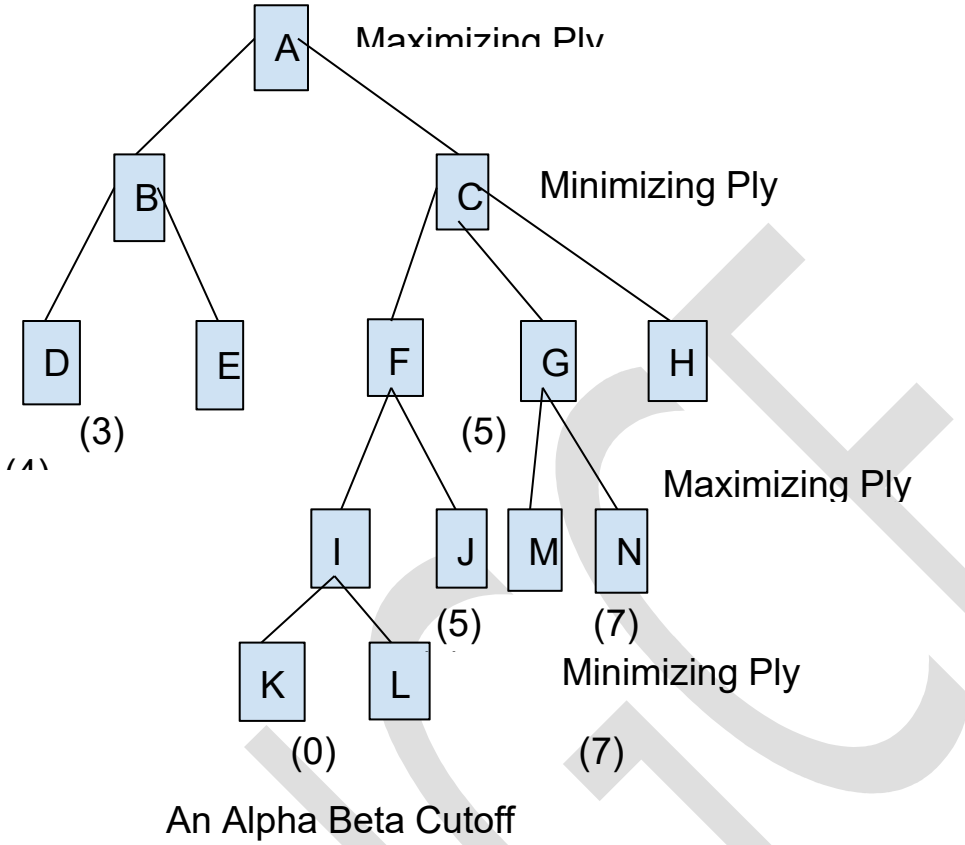
Just as it was necessary to modify our search procedure slightly to handle both maximizing and minimizing players, it is also necessary to modify the branch and bound strategy to include two bounds, one for each of the players. this modified strategy is called ***alpha-beta pruning***. It requires the maintenance of two threshold values, one representing a lower bound on the value that a maximizing node may ultimately be assigned(alpha) and another representing an upper bound on value that a minimizing node may be assigned(beta).



An Alpha Beta Cutoff

After examining node F, the opponent is guaranteed a score of -5 or less at C. But we are guaranteed a score of -5 or less at C. But we guaranteed a score of 3 or greater at node A which can achieve if we move to B. Any other move that produces a score of less than 3 is worse than move to B, we can ignore it. After examining only F we are sure that a move to C is worse regardless of the score of node G. Thus we need not bother to explore node G at all. Cutting out one node may not appear to justify the

expense of keeping track of the limits and checking them, but if we are exploring this tree to six ply, then we would have eliminated not a single node but an entire tree three ply deep.



In searching this tree, the entire subtree headed by B is searched and we discover that at A we can expect a score of at least 3. When this alpha value is passed down to F, it will enable us to skip the exploration of L. After K is examined, we see that I is guaranteed a maximum score of 0, which means that F is guaranteed a minimum of 0. But this is less than alpha's value of 3, so no more branches of I need to be considered. The maximizing player already knows not to choose to move to C and then to I since, if that move is made, the resulting score will be no better than 0 and a score of 3 can be achieved by moving to B instead.

After cutting off further exploration of I, J is examined yeilding a value of 5, which is assigned as the value of F. This value becomes the value of beta at node C. It indicates thatC guaranteed to get 5 or less .

Now we must expand y. First M is examined and it has a value of 7, which is passed back to y as its tentative value. But 7 is compared to beta(5). It is greater and the player whose turn it is at node C is trying to minimize. So that player will not choose y, which would lead to a score of at least 7, since there is an alternative move to F, which will lead to score of 5. Thus it is not necessary to explore any of the either branches of y.

.