**Game Playing**

<https://www.engineeringenotes.com/term-paper/game-playing/term-paper-on-game-playing-artificial-intelligence/34750>

**1. Introduction to Game Playing:**

The Federation of American Scientists have suggested that games teach skills which employers want analytical thinking, team building, multitasking and problem solving under duress. Unlike the humans, the games never lose patience.

Game-playing has played an important role in the history of artificial intelligence (AI). The earlier attempts at the game-playing computer program predate the field of AI. Even Babbage, the 19th century computer architect, desired to program his analytic engine to play chess though he did not succeed. Later on Dowden (1953) built a machine which could play tic-tac-toe. But nowadays computers compete at grand master level.

Claude Shanon, one of the pioneers of science of information and computing wrote a paper in 1950, in which he described mechanism which could be used in a program to play chess. Alan Turing, the father of AI, described a chess-playing program, though he could not build it.

In the early 1960s, Arthur Samuel wrote programs which could play checkers. His program even learnt from its mistake and improved its performance. Game-playing is a search technique with a difference. While we are doing our best to find the solution our adversary is trying to impede.

This search technique is a mixture of reasoning and creative flair and has epitomised the best of human intelligence. **The constrained environment of the game-playing with the clearly formulated rules is far more conductive to computation than the confused and open problems of everyday life.** One must plan well ahead and guess what our opponent’s moves will be.

**These ‘moves’ remain a ‘guess’ until we have made a move and, our opponent has responded. But alas! then its too late. The steps cannot be retraced, unlike in the case of working out to fill out a magic square, wherein one can back track and choose a different solution path. In the game playing, once the choice is made there is no going back.**

Mathematical Game theory, a branch of economics, views any multi-agent environment as a ‘game’ provided that the impact of each agent on others is “significant” regardless of whether the agents are cooperative or competitive.

**In AI, games are usually of a rather specialised kind-what game theorists call deterministic, turn-taking, two-player, zero-sum, games of perfect information. In simple language this means that the game is deterministic and has fully observable environments in which there are two agents whose actions must alternate and in which utility values at the end of game are always equal and opposite**.

In order to illustrate how the game-playing cycle works let us consider a game tree which belongs to a particular sort of the game — a deterministic or non- probabilistic, or absolute, open, two-person, turn taking, zero sum.

**These characteristics are explained below:**

**i. Two persons:**

No third party and no team playing on your or adversary side is assumed.

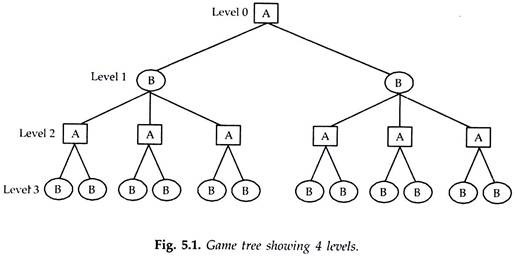
**ii. Turn taking:**

The players get alternative moves as opposed to a game where they can take multiple moves, based on the speed of play.

**iii. Zero sum:**

When one player wins, the other loses.

The possible game states can be organised into trees or graphs with the nodes linked by moves (arcs). The branches are labeled with the players who can make the choice between them. In a game tree alternate layers will be controlled by different players as shown in the Fig. 5.1.

[](https://www.engineeringenotes.com/wp-content/uploads/2018/01/clip_image0021.jpg)

In the Fig. 5.1, is shown a simple game tree showing 4 levels, starting with level, 0, in which player A has two options. In both states of level 1, player B has three options and in level 2 each state of player A has either 1 or 2 options. The options available to player B in level 3 are not shown.

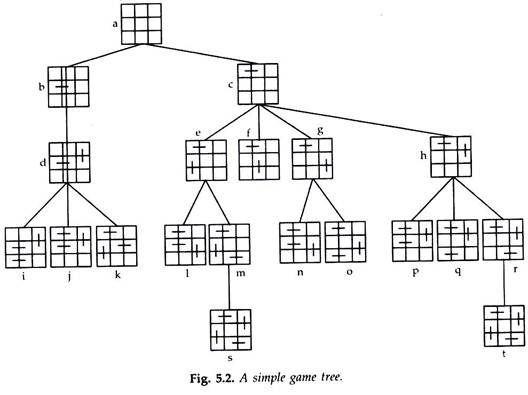
Like any deterministic problem, the game tree can be very big and typically has large branching factors. Even a trivial game like tic-tac-toe (or naught and crosses) has a game tree’s size too big to be drawn; it is usually possible only to examine a portion of the total state space.

**2. A Simple Game Tree:**

In order to demonstrate a complete game tree, consider game of Placing Dominoes. Take a squared board such as a chess board. Each player in turn places a domino which covers exactly two squares.

One player always places pieces right to left, and the other always places them top to bottom. The player who cannot place a piece loses. The complete tree when placed on 3 × 3 board is shown in Fig. 5.2. Some equivalent states have not been shown, for example, there are two states similar to b and four similar to c. (six in all).

Arisha and Bobby are the two adversaries, and Arisha plays first and places her pieces left to right. Consider board position j. This is a win for Arisha, as it is Bobby’s turn to play and there is no way to play a piece top to bottom (moves from i to k are similar). Position (s or t) is a win for Bobby, as neither player can play a piece, although it is Arisha’s turn to play.

[](https://www.engineeringenotes.com/wp-content/uploads/2018/01/clip_image002-41.jpg)

**Important features of the game search from this tree are:**

The leaves of the tree are given scores of + 1 (win for Arisha, Bobby loses) or -1 (win for Bobby – Arisha loses). Here search tree is complete but more often than not full complete game tree cannot be expanded. In such cases some useful heuristics are applied.

Left hand branch of the search tree is quite simple. If Arisha makes this move, she wins as she has only one move (apart from equivalent ones) and from this move she wins whatever state she acquires. The right-hand branch is rather more interesting and knowledgeable. So let us- study it in a little detail.

**Consider Node M:**

From this Bobby has one possible move, but this leads to a win for him (and loss for Arisha). This position (m) should be regarded a win for Bobby and could be labeled -1. From position e Arisha has two choices, either to play l-a win for her or to play m-a defeat. If Arisha is sensible enough, she will play l. Making this sort of reasoning we can mark the tree nodes as win or lose for Arisha.

In a win-lose game either there will be a way that the first player can always win or alternatively the second player will always be able to force a win. This game is a first-player win game; Arisha is a winner. If draws are also allowed then there is a third alternative that the two good players should always be able to prevent each, other from winning-all games are draws. This is the case for noughts and crosses and to some extent true in chess.

**The reason that, chess is more interesting to play than the game of noughts and crosses no one knows and even if it were true that in theory the first player would always win; the limited ability to look ahead means that this does not happen in practice. But in games like chess, the combination explosion is inherent.**

**(1) The average branching factor is 35,**

**(2) In an average, each player might make 50 moves,**

**(3) So in order to examine the complete game tree, we would have to examine 35100or 10154 state spaces (although search graph has ‘only’ about 1040 distinct nodes).**

So a straight forward search of the game tree is not at all possible. A player will not be able to make his/her first move in the life time of his/her opponent or even computer would take 10105 centuries to generate the full chess game tree, assuming that a successor would be generated in approximately a nanosecond. Games also penalise in efficiency severely.

**3. Additional Pruning of Game Tree:**

The minimax and negmax procedures specify how static evaluation function value gets propagated upward through the game tree in order to select the optimum move. The minimax and negmax procedures themselves are heuristic only in the sense that they select the optimum move based on the heuristic function, *f*j.

The minimax procedure by itself does not prune away any of the search space of the game tree. A number of heuristics are available for additional pruning of the game tree to permit a reasonable number of plies to be searched in a reasonable time.

**Plausible Move Generator**:

**Certain moves are unproductive or dangerous because they obviously lead away from engagement or to an immediate loss of a piece with no hope of trapping the other player. These moves can be avoided by the substitution of a plausible move generator in place of the legal move generator.**

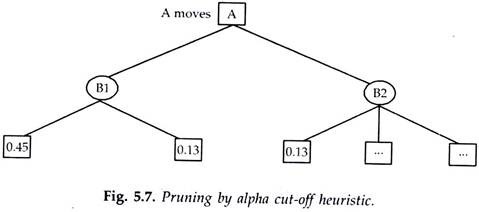
**If, for instance, the average branching from a node in chess could be cut from 30 to 10 by means of a plausible move generator, the number of nodes requiring evaluation in a 6 ply search could be cut from 729 million to just 1 million. The plausible move generator is equivalent to a beam search in which only a limited number of promising nodes are expanded at each ply.**

**Alpha-Beta Cut-Off**:

The minimax procedure is a depth-first 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 the value can then be passed up the path one level at a time.

This search procedure gets modified, to a new search strategy, called alpha-beta pruning, when minimax procedure works under two bounds, one for each player. It requires the maintenance of two threshold values, one representing the lower bound on the value that a maximising node may ultimately be assigned (call it alpha) and another representing an upper bound on the value that a minimising node may be assigned (call it beta).

So the alpha-beta cut-off heuristic is a form of branch-and-bound technique. In Fig. 5.7, we show the application of the alpha cut-off and in Fig. 5.8 that of the beta cut-off.

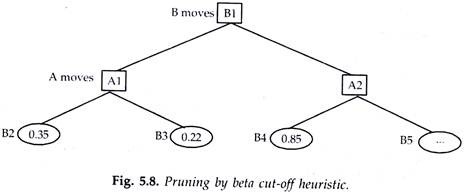
[](https://www.engineeringenotes.com/wp-content/uploads/2018/01/clip_image004-30.jpg)

α is defined as the lower limit (bound) on the value of the maximiser accordingly the value of alpha will assume the largest of the minimum values generated from expanding successive B nodes at a given level (ply). Once a successor node from any B node at that level has value less than alpha, that node and all its successors may be pruned.

In Fig. 5.7, it may be noted that as the minimax search evaluates nodes in ply 2 (bottom level), it can stop once the *f*j = 0.13 node is computed. The argument is that Bobby will select 0.45 if Arisha selects move B1. (Alpha is set to 0.45 by the expansion of the B1 nodes).

If Arisha selects move B2, B2 will select the node/evaluating to 0.13 or smaller. Therefore, Arisha must select move B1 independently of the \*\*\* node values. Therefore, these nodes and their successor may be pruned from the game tree.

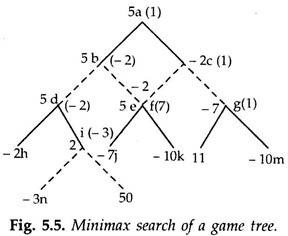
A similar branch-and-bound argument may be used to generate the beta cut-off. Beta is defined as the upper limit (bound) on the value of the minimiser. By examining nodes resulting from expanding nodes in an A ply we note that in Fig. 5.8, beta will be set to a value of 0.35. Any additional A nodes in ply 1 which have any successors evaluating to more than 0.35 will automatically be pruned from the game tree by the beta cut-off.

[](https://www.engineeringenotes.com/wp-content/uploads/2018/01/clip_image006-29.jpg)

The use of alpha-beta pruning can greatly narrow the search and offer the opportunity of searching much more deeply into the game tree with the same number of evaluations. As a result, this heuristic is almost universally applied, at least as an outer limit on the width of the search. It is also clear, after a little thought, that alpha is a bound which continues to get larger as higher values of the evaluation function are found at a given level.

The general statement is that alpha is the largest of the minimum values of *f*j for the sons of Bn. Similarly, the value of beta will decrease as more nodes are opened at the 2-ply level of Fig. 5.8. The general statement is that beta will be the smallest of the maximum values of *f*j for the sons of An.

The same can be explained by applying branch and bound technique on the minimax procedure. Referring, again to Fig. 5.5, imagine we are considering moves from d. h has score – 2. Now look at node i, its child n has score -3. So before we look at O with score 5, we know that the minimax score of i will not be more than – 3, as Bobby will be minimising. Thus, Arisha would be foolish to choose i as h is going to be better than i whatever score O has.

[](https://www.engineeringenotes.com/wp-content/uploads/2018/01/clip_image005_thumb2.jpg)

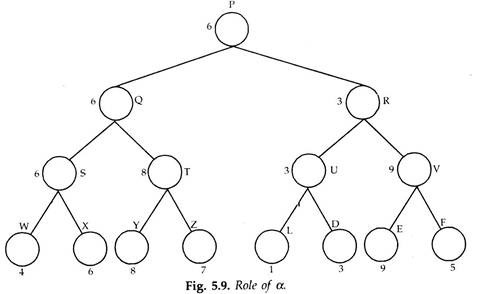
This can also be substantiated by referring to Fig. 5.2. Imagine, we are trying to find the move from position C. We have evaluated e and its children as well as *f*; and are about to look at the children of g and h. From Bobby’s point of view (minimisation) *f* is the best so far.

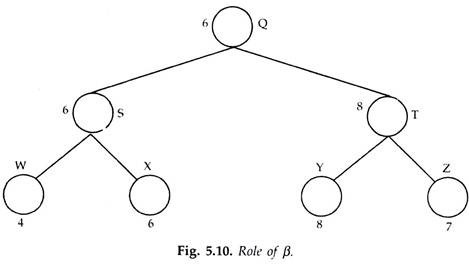
Now as soon as we look at node n we can see that the minimax score for g will be at least ‘1’ as Arisha will play to maximise; so there is no reason to examine node O. Similarly, having seen node t, nodes p and q can be skipped. In fact, if we look a bit further up we can see that even less search is required. Position b has a minimax score of ‘V.

As soon as we have seen that node *f* has a score’—1′. We know that Bobby could choose this path and that the minimax score of C is at most -1. Thus, nodes g and h could be ignored completely. This process of pruning by α – β heuristics depend upon carrying around a best-so-far value for Arisha’s choice and worst-so-far value for Bobby’s choice.

**Illustration of α – β Cut-Off:**

The combined role of α – β cut-off can be fully appreciated by taking a skeleton game tree (Fig. 5.9 and 5.10).

[](https://www.engineeringenotes.com/wp-content/uploads/2018/01/clip_image008-19.jpg)

[](https://www.engineeringenotes.com/wp-content/uploads/2018/01/clip_image010-14.jpg)

**Role of α:**

Consider the tree up to three levels in Fig. 5.9. The static evaluation function generator has assigned values which are given for leaf nodes. By shifting back the values Q takes minimum of 6 and 8 and R the minimum of 3 and 9.

P being maximiser, will obviously opt for Q. Now to P, the maximiser, a value of 6 is assured by moving to node Q. This value is compared with that of value at R. P, being the maximiser would follow any path whose value is greater than or at least equal to 6. Hence, this value of 6, being least that a maximiser node can obtain is set as the value of α.

This value of α is used as a reference point. Any node whose value is greater than a is acceptable and all nodes whose values are less than this value are rejected. From the above tree R has the value of 3, which is less than the cut-off value. Hence the entire tree under R is totally rejected.

**Role of β:**

Consider now the tree Fig. 5.10, which is a portion of the above tree in Fig. 5.9, with Q as the root. Q is a minimiser. S and T are maximisers, the maximum values of their children are backed up as their static evaluation function values.

Q being minimiser will move to S. The value of S (6) is used as reference called β, the maximum value a minimiser can be assigned. Any node whose value is more than this value are not acceptable, hence the tree under T is pruned.

**The two golden rules that are used in the α-β cut-off strategy are:**

i. R: for maximisers, if the static evaluation function value formed at any node is less than the α-value, reject it.

ii. R: for minimisers, if the static evaluation function value found at an node is greater than the β-value, reject it.

**Effectiveness of α-β Algorithm:**

For a uniform tree whose depth is d and branching factor is b the minimax algorithm evaluates all the nodes which corresponds to examining bd nodes. For the modified mini-max algorithm (α-β cut-off), the number of nodes to be examined is (best case and tree is balanced).

[](https://www.engineeringenotes.com/wp-content/uploads/2018/01/clip_image0025-1.png)

[](https://www.engineeringenotes.com/wp-content/uploads/2018/01/clip_image0045.png)

But there is a need to observe a restrain-the formula is valid for only the special case in which a tree is perfectly arranged. As such the above formula is an approximation; if there were a way of arranging the tree with the best moves on the left, clearly there would be no point in using α-β purning. However, this does not mean that the exercise of α-β have been fruitless—it establishes the lower bound on the number of static evaluations which would be needed in a real game.

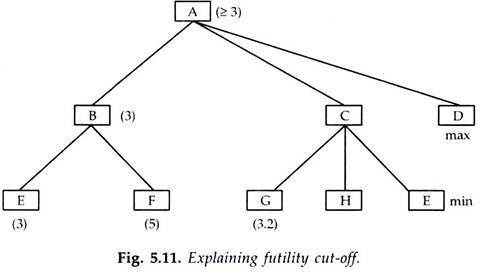
It is a lower bound which may or may not be close to the real result, depending on how well the moves are, in fact, arranged. The real result must lie somewhere between the worst case for which static values must be somewhere between the worst case for which static values must be computed for all bd leaf nodes and the best case, for which static values must be computed for approximately 2bd/2 leaf nodes. In practice, the number of static evaluations come nearer to the best case than the worst case.

Still the amount of work required becomes impossibly large with increasing depth. The α-β procedure merely wins a temporary reprieve from the effects of the explosive exponential growth and the procedure does not prevent the explosive growth.

**Futility Cut-Off Limit:**

The α-β procedure can also be used to cut-off additional paths which at best show only slight improvement over paths which have already been explored.

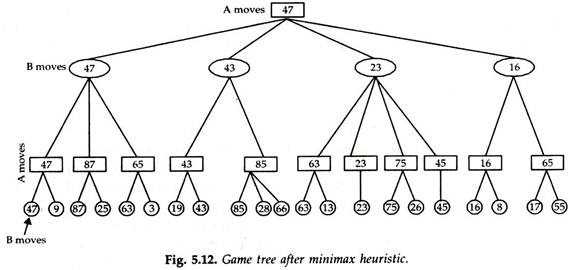
Consider the minimax situation shown in Fig. 5.11. After examining node G, we can say that if the move is along C the value of static evaluation function is 3.2 but the move toward B is guaranteed and has a score of 3. Since 3.2 is only very slightly better than 3, in the scale 0-10 we should disband exploration of C. We can thus devote more time to exploring other parts of the tree which could be more useful.

[](https://www.engineeringenotes.com/wp-content/uploads/2018/01/clip_image014-14.jpg)

This limit of the static evaluation function is called futility cut-off limit and helps in terminating the exploration of a sub tree which offers limit possibility for improvement over other known parts. This concept of futility limit was imbibed in depth-first search.

**Alpha-Beta Cut-Off Example**:

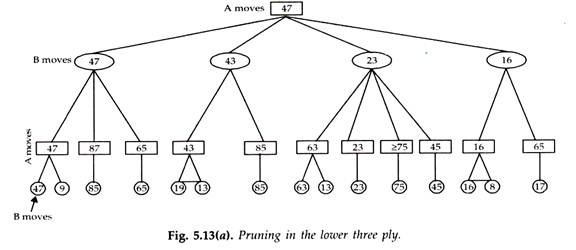
**Consider the following 3-ply game tree which has been completely evaluated with the minimax procedure, (assuming root as zeroth level). (Fig.5.12).**

[](https://www.engineeringenotes.com/wp-content/uploads/2018/01/clip_image016-14.jpg)

The first step in applying the alpha-beta heuristic is to evaluate the B-A-B structures in the lowest three ply (Ply 1, 2, 3) for each family (sons and grandsons) of Bn at Ply 1. Beta is set equal to 47 after evaluating nodes (47, 9). Since no smaller maximum occurs in the leftmost 6 nodes, no further ratcheting downwards of beta occurs in this set, but nodes (25,3) are eliminated without evaluation.

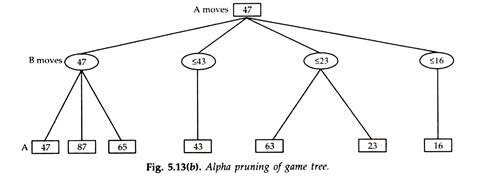
As the evaluations proceed into the (19 – 66) series of nodes, beta is set to 43 after evaluating nodes (19, 43). No ratcheting of beta occurs, and nodes (28, 66) are eliminated. Within the series (63 – 45) beta starts at 63 after nodes (63, 13) and ratchets downward to 23 after node (23), thus eliminating node (26). In the series (16-55), beta starts at 16 after nodes (16, 8) and eliminates evaluation of node (55).

Six of the 21, 3-ply nodes are eliminated from the search and evaluation process by the beta pruning heuristic. Note the ‘≥’ signs resulting from reduced information available at the An nodes at ply 3. This uncertainty is no way effects the final result of the pruned search. The tree after applying beta pruning is shown in Fig. 5.13 (a).

[](https://www.engineeringenotes.com/wp-content/uploads/2018/01/clip_image018-12.jpg)

Now, let us assume that we had opened the game tree down only two plies and had computed the evaluation function/ to give the An shown in ply 2 of Fig. 5.13(b). Let’s see what pruning the alpha cut-off would provide? After nodes (47, 87, 65) are evaluated, alpha is set to 47, and no further ratcheting of alpha occurs (since no larger minimum of sons of Bn occurs).

After opening node 43 in ply 2, the alpha cut-off of 47 tells us to propagate “≤ 43” upward to Bn and thereby eliminating the node which would evaluate to (85). Similarly, after evaluating nodes (63, 23) we can propagate “≤ 23” up to Bn and so on. The results are shown in Fig. 5.13 (b).

[](https://www.engineeringenotes.com/wp-content/uploads/2018/01/clip_image019.jpg)

It is obvious that alpha pruning eliminates 4 nodes which would have evaluated to (85, 75, 45, 55). After the left three nodes are evaluated, alpha is set to 47 and does not ratchet upward in this example.

The final result of alpha-beta pruning has been to eliminate 6 of 21 nodes (applying the beta cut-off at ply 3) or 4 of 11 nodes (applying alpha cut-off at ply 2) of the game tree in Fig. 5.12. It should be noted that the efficiency of alpha-beta pruning depends critically on the ordering of the nodes at the lower levels of the tree.

In the worst possible case, there is no improvement over normal minimax evaluation. In the best possible case, the number of nodes requiring evaluation is 2√N where, N is the number of nodes in the normal minimax tree.

The alpha-beta cut-off algorithm was first introduced in chess program MacHack by Greenblatt et.al (1967). It has been an important tool in essentially all significant chess and checkers programs since then.

The minimax and α-β heuristics presented here, assumed a fixed level in which breadth first search is conducted, depending upon the computing power of the computing machine.

**4. More Refinements in Game Playing:**

In addition to α-β procedure, there are a few more refinements to the minimax procedures which can still improve its performance. A number of heuristics are available for additional pruning of the game tree to permit a reasonable number of plies to be searched in a reasonable time.

**These heuristics are described below:**

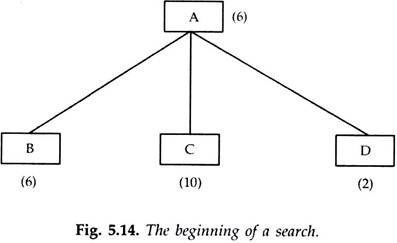
**Waiting for Quiescence**:

This is sophisticated cut-off test which decides when to stop searching deeper. The evaluation function should be applied only to positions which are quiescent that is unlikely to exhibit wild savings in value in the near future.

In chess, for example, positions in which favourable captures can be made are not quiescent for an evaluation function which just counts material. Non-quiescent positions can be expanded further until quiescent positions are reached.

This extra search is called a quiescence search, sometimes it is restricted to consider only certain types of moves, such as capture moves, which will quickly resolve the uncertainties in the position.

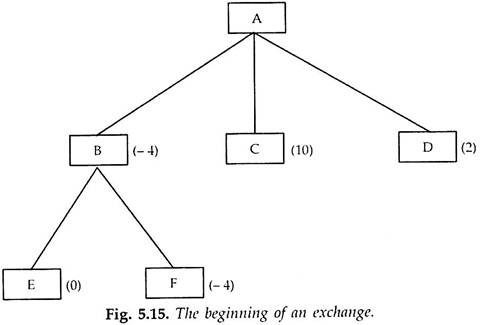
Consider the tree shown in Fig. 5.14 with nodes B, C, D created at the first ply from the root of the tree, A.

[](https://www.engineeringenotes.com/wp-content/uploads/2018/01/clip_image021.jpg)

Suppose that when node B is expanded one more level, the result is shown in Fig. 5.15. When we looked one move ahead, our estimate of the cost of B changed drastically (- 4 from 6). This might happen for example in the middle of a piece exchange.

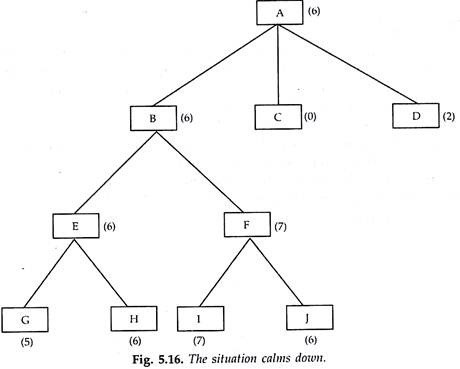
The opponent has significantly improved the immediate appearance of his/her position by initiating a piece exchange. If we stop exploring the tree at this level we assign the value – 4 to B and therefore decide that B is not a good move for the first player.

Should such a short-term measure actually influence our choice of move?

[](https://www.engineeringenotes.com/wp-content/uploads/2018/01/clip_image023.jpg)

No, this is too early to decide rather the search should be continued until no such drastic change occurs from one level to the next.

We may extend the tree in (Fig. 5.15) just by one more level and we might get a situation as shown in Fig. 5.16. The move to B looks again reasonable since value of evaluation function is 6, (which was at start) at A, since the other half of the place exchange has occurred. This is called waiting for quiescence.

[](https://www.engineeringenotes.com/wp-content/uploads/2018/01/clip_image025.jpg)

Thus, waiting for quiescence helps in avoiding the horizon effect in which an inevitable bad event can be delayed by various tactics until it does not appear in the mid the game tree which minimax explores. The effect may make a move look good tactics despite the fact that the move might be better if we delayed past the horizon.

The horizon effect can also influence a programs perception of good moves. Even with quiescence all fixed depth search programs are subject to subtle horizon effect.

As already stated computer chess-playing programs have to deal with a major problem called Horizon Effect, a term coined by Berliner (1977). These programs do look ahead search and if it is possible for the program to postpone the danger by dragging the situation into a branch of the tree which the algorithm explores, then the situation (original one) is out of the view of the program or horizon.

Horizon effect is more difficult to eliminate. It arises when the program is facing a move by the opponent which causes serious damage and is ultimately unavoidable. Consider the chess game in Fig. 5.17. Black is a head in material, but if white can advance its pawn from the seventh row to the eight the pawn will become a queen and create an easy win for white. Black can forestall this outcome for 14 ply by checking white with the rook, but inevitably the pawn will become a queen.

The problem with fixed-depth search is that it believes that these stalling moves have avoided the queen move—we say that the stalling moves push the inevitable queening move “over the search horizon” to place where it cannot be detected. In the Fig. 5.17, by a series of check by the Black rook forces the inevitable queening move by the white “over the horizon” and makes this position look like a win for Black, when it is really a win for White.

With the improvements in the hardware deeper search will be possible and it is expected that horizon effect will occur less frequently. The horizon effect can also be avoided with the use of singular extensions. A singular extension is a move which is “clearly better” than all other moves in a given position.

A singular extension search can go beyond the normal depth without incurring much cost because its branching factor is 1. (Quiescent search can be considered as a special type of singular extension). In Fig. 5.17, a singular extension search will find the eventual queening move, provided that Black’s checking moves and White’s king moves can be identified as “clearly better” than the alternatives.