

# Introduction to Artificial Intelligence

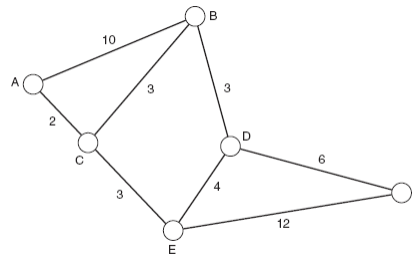
## Unit # 5

## Acknowledgement

- The slides of this lecture have been taken from the lecture slides of CS307 – “Introduction to Artificial Intelligence” by Dr. Sajjad Haider.

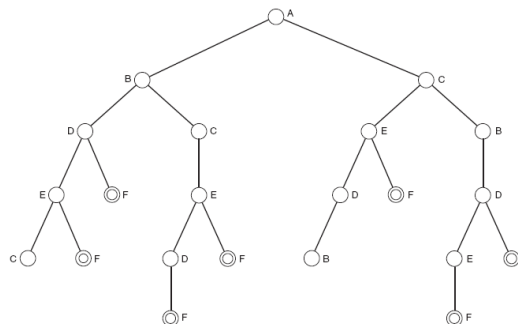
## The Modified Traveling Salesman Problem

- Each node represents a town, and the vertices between nodes represent roads that join towns together.
- A is the starting node, and F is the goal node.
- Each vertex is labeled with a distance, which shows how long that road is.
- The aim of this problem is to find the shortest possible path from city A to city F.



## Search Tree for the Modified TSP

- 1 A,B,D,E,F
- 2 A,B,D,F
- 3 A,B,C,E,D,F
- 4 A,B,C,E,F
- 5 A,C,E,F
- 6 A,C,B,D,E,F
- 7 A,C,B,D,F



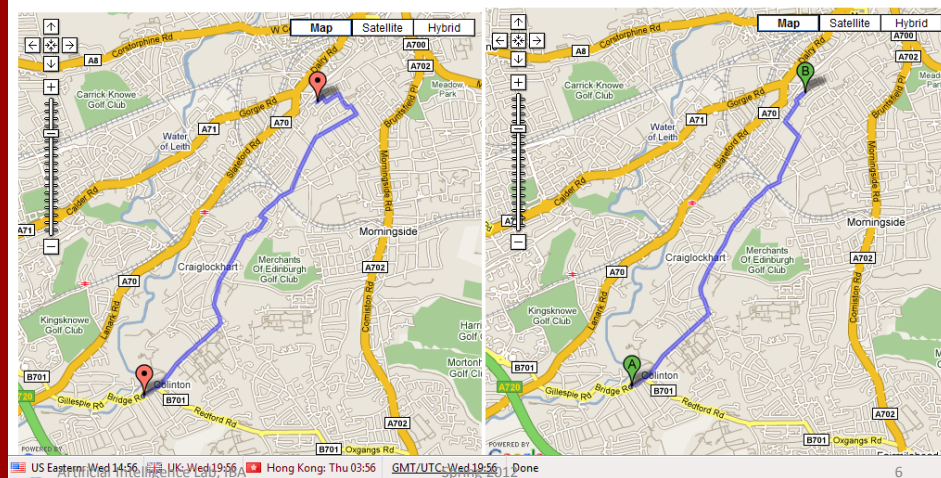
# Search Demo

[http://www.cs.rochester.edu/u/kautz/Mazes/search\\_algorithm\\_demo.htm](http://www.cs.rochester.edu/u/kautz/Mazes/search_algorithm_demo.htm)

# Result

A star algorithm

Google Map API



## Informed vs. Uniformed Search

- A search method or heuristic is **informed** if it uses additional information about nodes that have not yet been explored to decide which nodes to examine next.
- If a method is not informed, it is **uninformed** or **blind**.
- Best-first search is an example of informed search, whereas breadth-first and depth-first search are uninformed or blind.
- The more informed a search method is, the more efficiently it will search.

## Summary: Search

- Depth-first search and breadth-first search are extremely commonly used and well understood exhaustive search methods.
- A search method is **complete** if it will always find a solution if one exists. A search method is **optimal** (or admissible) if it always finds the best solution that exists.
- Heuristics can be used to make search methods more informed about the problem they are solving. A heuristic is a method that provides a better guess about the correct choice to make at any junction that would be achieved by random guessing.
- One heuristic is more informed than another heuristic if a search method that uses it needs to examine fewer nodes to reach a goal.

## More Search..... 😊

### Greedy Algorithm

(Source: Wikipedia)

- Greedy algorithms can be characterized as being 'short sighted', and as 'non-recoverable'. They are ideal only for problems which have 'optimal substructure'. Despite this, greedy algorithms are best suited for simple problems.
- **Greedy choice property**
- We can make whatever choice seems best at the moment and then solve the subproblems that arise later. The choice made by a greedy algorithm may depend on choices made so far but not on future choices or all the solutions to the subproblem. It iteratively makes one greedy choice after another, reducing each given problem into a smaller one. In other words, a greedy algorithm never reconsiders its choices.

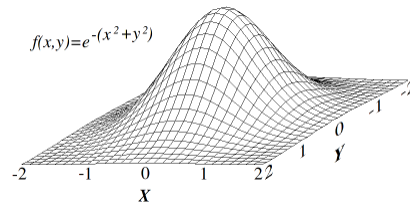
## Hill Climbing

(Source: Wikipedia)

- In [computer science](#), **hill climbing** is a [mathematical optimization](#) technique which belongs to the family of [local search](#). It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by [incrementally](#) changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found.
- For example, hill climbing can be applied to the [traveling salesman problem](#). It is easy to find an initial solution that visits all the cities but will be very poor compared to the optimal solution. The algorithm starts with such a solution and makes small improvements to it, such as switching the order in which two cities are visited. Eventually, a much shorter route is likely to be obtained.
- Hill climbing is good for finding a [local optimum](#) (a good solution that lies relatively near the initial solution) but it is not guaranteed to find the best possible solution (the [global optimum](#)) out of all possible solutions (the [search space](#)).

## Local vs Global Optimum?

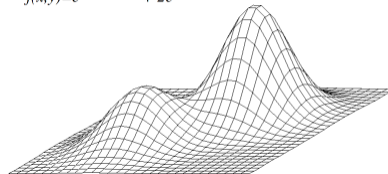
What is the optimum point?



## Local vs Global Optimum?

What do you think about the optimum point now?

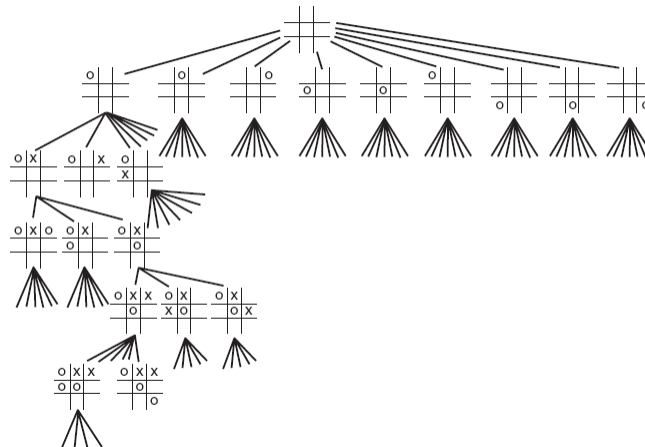
$$f(x,y) = e^{-(x^2+y^2)} + 2e^{-((x-1.7)^2+(y-1.7)^2)}$$



## Game Tree

- Many two-player games can be efficiently represented using trees, called **game trees**.
- A game tree is an instance of a tree in which the root node represents the state before any moves have been made, the nodes in the tree represent possible states of the game (or **positions**), and arcs in the tree represent moves.
- It is usual to represent the two players' moves on alternate levels of the game tree, so that all edges leading from the root node to the first level represent possible moves for the first player, and edges from the first level to the second represent moves for the second player, and so on.

## Game Tree of Tic-Tac-Toe



## Search in Game Trees

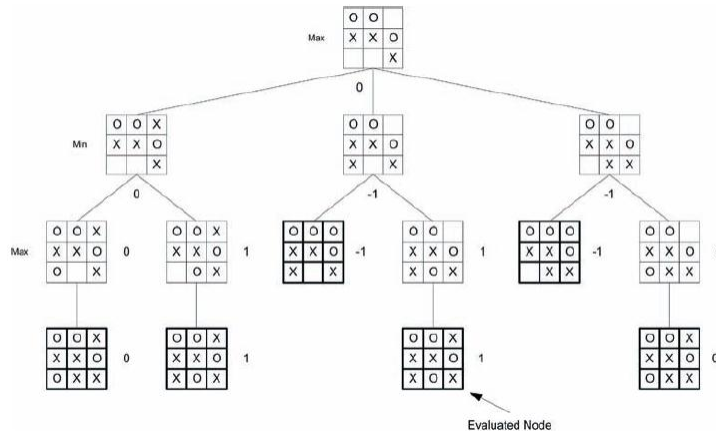
- One approach to playing a game might be for the computer to use a tree search algorithm such as depth-first or breadth-first search, looking for a goal state (i.e., a final state of the game where the computer has won).
- For Tic-Tac-Toe, the branching factor of the root node is 9 because there are nine squares in which the computer can place its first nought.
- It can be inefficient for a computer to exhaustively search the tree because it has a maximum depth of 9 and a maximum branching factor of 9, meaning there are approximately  $9 \times 8 \times 7 \dots 2 \times 1$  nodes in the tree, which means more than 350,000 nodes to examine.
- The [Deep Blue chess computer](#) which defeated [Kasparov](#) in 1997 would typically search to a depth of between six and sixteen plies to a maximum of forty plies in some situations.

## Evaluation Function

- For a computer to use this tree to make decisions about moves in a game of tic-tac-toe, it needs to use an **evaluation function**, which enables it to decide whether a given position in the game is good or bad.
- If we use exhaustive search, then we only need a function that can recognize a win, a loss, and a draw.
- Then, the computer can treat “win” states as goal nodes and carry out search in the normal way.



## Minimax Algorithm for Tic-Tac-Toe



Artificial Intelligence Lab, IBA

Spring 2012

17

## Ply and Move

- In discussing game trees, we use the concept of **ply**, which refers to the depth of the tree.
- When a computer evaluates a game tree to ply 5, it is examining the tree to a depth of 5. The 4th ply in a game tree is the level at depth 4 below the root node.
- Because the games we are talking about involve two players, sequential plies in the tree will alternately represent the two players. Hence, a game tree with a ply of 8 will represent a total of eight choices in the game, which corresponds to four moves for each player.
- It is usual to use the word *ply* to represent a single level of choice in the game tree, but for the word *move* to represent two such choices—one for each player (Source: wikipedia)

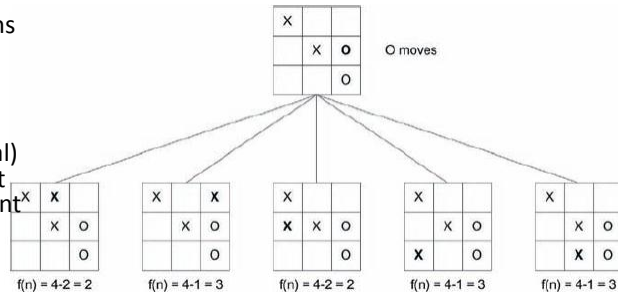
Artificial Intelligence Lab, IBA

Spring 2012

18

## Static Evaluation Function for Tic-Tac-Toe

- The static evaluation function is defined as the number of possible win positions not blocked by the opponent minus the number of possible win positions (row, column, and diagonal) for the opponent not blocked by the current player:



- $f(n) = \text{win\_positions} - \text{lose\_positions}$

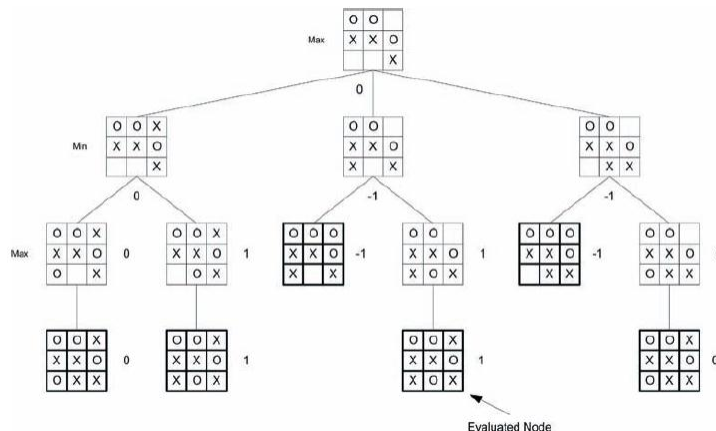
## Minimax Algorithm

- The minimax algorithm is a useful method for simple two-player games. It is a method for selecting the best move given an alternating game where each player opposes the other working toward a mutually exclusive goal.
- In a two-person game, you must assume that your opponent has the same knowledge that you do and applies it as well as you do. So at each stage of the game you must assume your opponent makes the **best available move**. This is the basis of the minimax procedure.

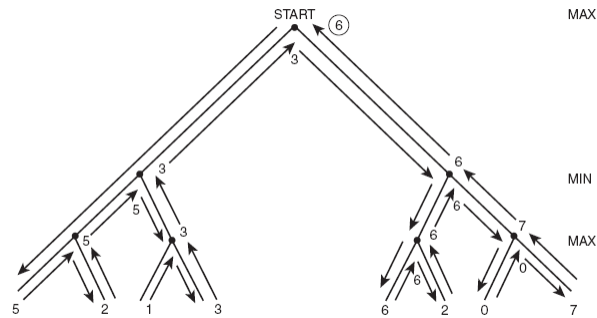
## MiniMax Algorithm

- It is assumed that a suitable static evaluation function is available, which is able to give an overall score to a given position.
- In applying Minimax, the static evaluator will only be used on leaf nodes, and the values of the leaf nodes will be filtered up through the tree, to pick out the best path that the computer can achieve.
- The principle behind Minimax is that a path through the tree is chosen by assuming that at its turn (a **max node**), the computer will choose the move that will give the highest eventual static evaluation, and that at the human opponent's turn (a **min node**), he or she will choose the move that will give the lowest static evaluation.
- In other words, we assume that each player makes the move next that benefits them the most.

## Minimax Algorithm for Tic-Tac-Toe



## Example of Minimax Algorithm

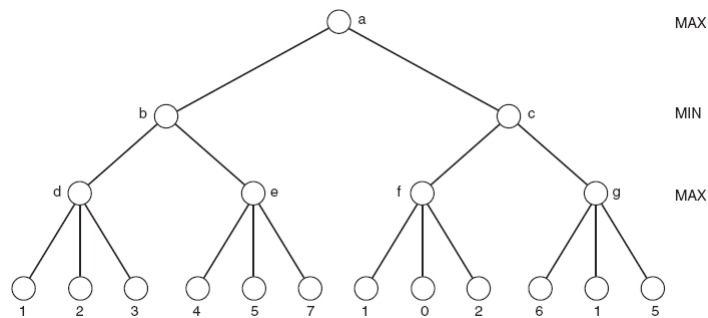


Artificial Intelligence Lab, IBA

Spring 2012

23

## Example I

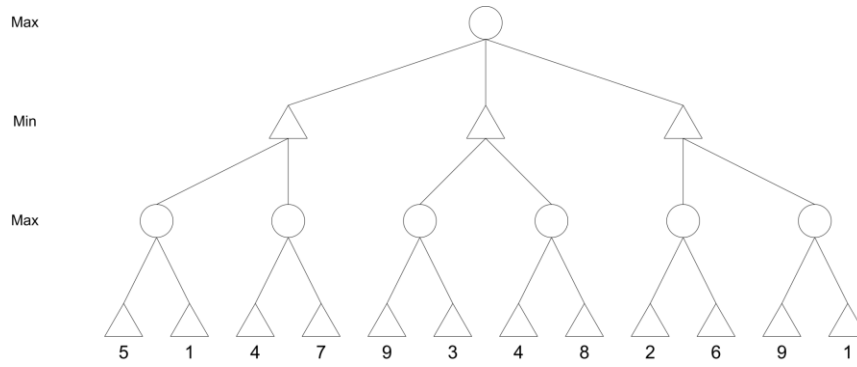


Artificial Intelligence Lab, IBA

Spring 2012

24

## Example II

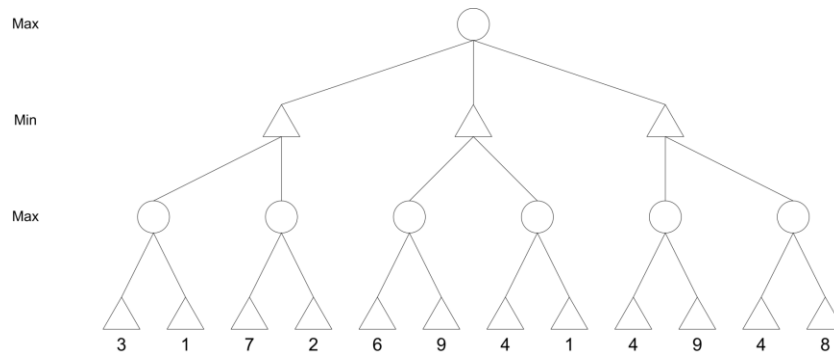


Artificial Intelligence Lab, IBA

Spring 2012

25

## Example III



Artificial Intelligence Lab, IBA

Spring 2012

26

## Lookahead and Horizon Effect

- Minimax is a very simple algorithm and is unsuitable for use in many games, such as chess, where the game tree is extremely large.
- In such cases, **bounded lookahead** is very commonly used and can be combined with Minimax.
- The idea of bounded lookahead is that the search tree is only examined to a particular depth. All nodes at this depth are considered to be leaf nodes and are evaluated using a static evaluation function.
- When we employ lookahead strategy, we suffer from what is called the **horizon effect**.
- When we can't see beyond the horizon, it becomes easier to make a move that looks good now, but leads to problems later as we move further into this subtree.

## Representation of Tic-Tac-Toe board

O	O	X
X	X	O
O	X	X

How to represent it in memory ?

## Minimax Optimization

Which are the factors that effect the feasibility of Minimax algorithm?

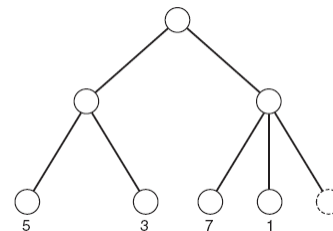
- Depth of tree
- Branching factor

While there are  $3^9$  unique Tic-Tac-Toe board configurations, there are actually many fewer valid boards as an early win (prior to filling up the board) makes all successor boards invalid.

		X
X	X	O
		O

## Alpha-Beta Pruning

- Using alpha–beta pruning, it is possible to remove sections of the game tree that are not worth examining, to make searching for a good move more efficient.
- The principle behind alpha–beta pruning is that if a move is determined to be worse than another move that has already been examined, then further examining the possible consequences of that worse move is pointless.



## Working of Alpha-Beta Pruning

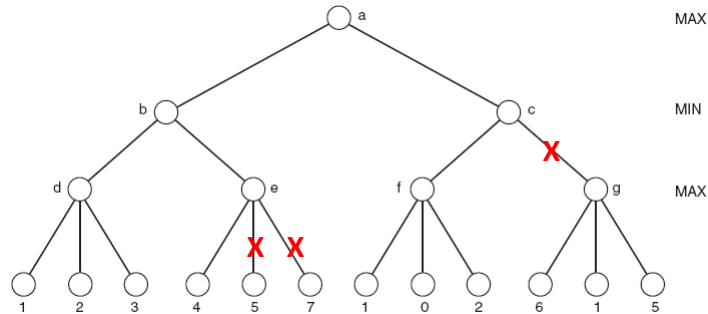
- During the depth-first search of the game tree, we calculate and maintain two variables *alpha* and *beta*.
- The alpha variable defines the best move that can be made to maximize (our best move) and the beta variable defines the best move that can be made to minimize (the opposing best move).
- While we traverse the tree, if alpha is ever greater than or equal to beta, then the opponent's move forces us into a worse position.
- In this case, we avoid evaluating this branch any further.

## Working of Alpha-Beta Pruning

- The algorithm maintains two values, alpha and beta, which represent the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of respectively.
- Initially alpha is negative infinity and beta is positive infinity.
- As the recursion progresses the "window" becomes smaller. When beta becomes less than alpha, it means that the current position cannot be the result of best play by both players and hence need not be explored further.



## Example I (Revisited)



Only 7 nodes (out of 12) explored with alpha-beta pruning.

## Working of Alpha-Beta for Example I

Step	Node	Alpha	Beta	Notes
1	a	$-\infty$	$\infty$	Alpha starts as $-\infty$ and beta starts as $\infty$ .
2	b	$-\infty$	$\infty$	
3	d	$-\infty$	$\infty$	
4	d	1	$\infty$	
5	d	2	$\infty$	
6	d	3	$\infty$	At this stage, we have examined the three children of d and have obtained an alpha value of 3, which is passed back up to node b.
7	b	$-\infty$	3	At this min node, we can clearly achieve a score of 3 or better (lower). Now we need to examine the children of e to see if we can get a lower score.
8	e	$-\infty$	3	

## Working of Alpha-Beta for Example I (Cont'd)

8	e	$-\infty$	3	
9	e	4	3	CUT-OFF. A score of 4 can be obtained from the first child of e. Min clearly will do better to choose d rather than e because if he chooses e, max can get at least 4, which is worse for min than 3. Hence, we can now ignore the other children of e.
10	a	3	$\infty$	The value of 3 has been passed back up to the root node, a. Hence, max now knows that he can score at least 3. He now needs to see if he can do better.
11	c	3	$\infty$	
12	f	3	$\infty$	We now examine the three children of f and find that none of them is better than 3. So, we pass back a value of 3 to c.
13	c	3	3	CUT-OFF. Max has already found that by taking the left-hand branch, he can achieve a score of 3. Now it seems that if he chooses the right-hand branch, min can choose f, which will mean he can only achieve a score of 2. So cut-off can now occur because there is no need to examine g or its children.

Spring 2012

35

## Alpha-beta pruning

```

alpha-beta(player,board,alpha,beta)
  if(game over in current board position)
    return winner
  children = all legal moves for player from this board
  if(max's turn)
    for each child
      score = alpha-beta(other player,child,alpha,beta)
      if score > alpha then
        alpha = score (we have found a better best move)
      if alpha >= beta then return alpha (cut off)
      return alpha (this is our best move)
  else (min's turn)
    for each child
      score = alpha-beta(other player,child,alpha,beta)
      if score < beta then
        beta = score (opponent has found a better worse move)
      if alpha >= beta then
        return beta (cut off)
      return beta (this is the opponent's best move)

```

Artificial Intelligence Lab, IBA

Spring 2012

36

## Alpha-beta pruning - Pseudocode

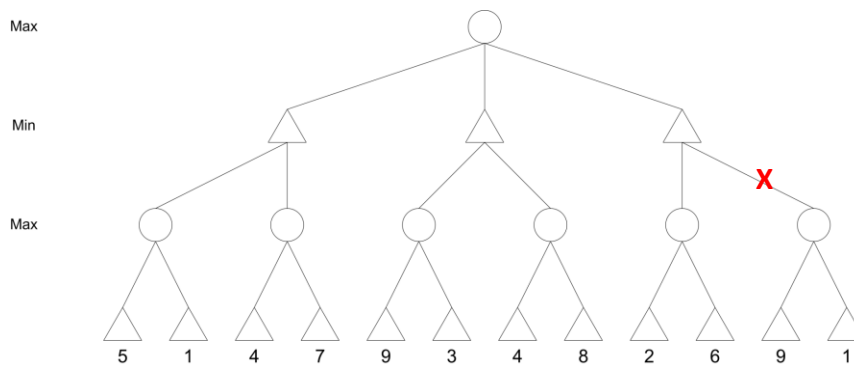
(Source: Wikipedia)

```

function alphabeta(node, depth,  $\alpha$ ,  $\beta$ , Player)
  if depth = 0 or node is a terminal node
    return the heuristic value of node
  if Player = MaxPlayer
    for each child of node
       $\alpha := \max(\alpha, \text{alphabeta}(\text{child}, \text{depth}-1, \alpha, \beta, \text{not}(\text{Player}))$ 
      if  $\beta \leq \alpha$ 
        break
      return  $\alpha$ 
      (* Beta cut-off *)
  else
    for each child of node
       $\beta := \min(\beta, \text{alphabeta}(\text{child}, \text{depth}-1, \alpha, \beta, \text{not}(\text{Player}))$ 
      if  $\beta \leq \alpha$ 
        break
      return  $\beta$ 
      (* Alpha cut-off *)
  (* Initial call *)
  alphabeta(origin, depth, -infinity, +infinity, MaxPlayer)

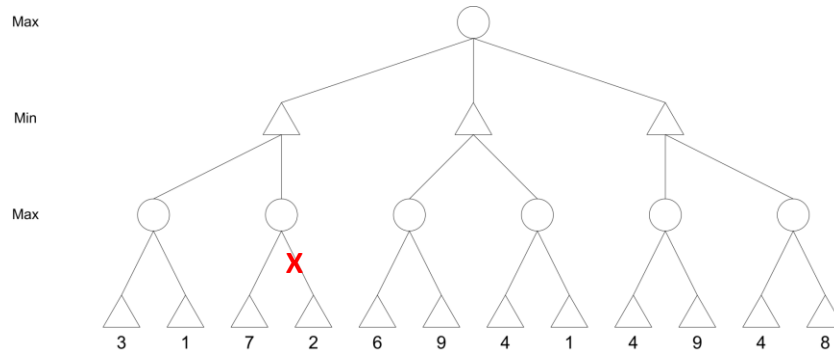
```

## Example II



Only 10 nodes (out of 12) explored with alpha-beta pruning.

## Example III

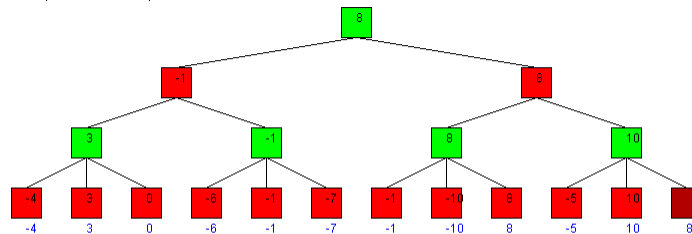


Only 11 nodes (out of 12) explored with alpha-beta pruning.

## Java Applet

- <http://www.ocf.berkeley.edu/~yoseni/extras/alphabeta/alphabeta.html>

Nodes completed: 18 / Nodes pruned: 1



## Advantages of Alpha-Beta Pruning

- The alpha–beta pruning method provides its best performance when the game tree is arranged such that the best choice at each level is the first one (i.e., the left-most choice) to be examined by the algorithm.
- With such a game tree, a Minimax algorithm using alpha–beta cut-off will examine a game tree to double the depth that a Minimax algorithm without alpha–beta pruning would examine in the same number of steps.