Adversarial Search and Game-Playing

Note: this material was originated from the slides provided by Prof. Padhraic Smyth

Typical assumptions

- Two agents(game players) whose actions are alternate
- Utility values for each agent are the opposite of the other
  - If a utility value increases for one player, then it decreases for the other player
  - This property creates the adversarial situation
- In game theory terms:
  - Deterministic: The result of any action is expectable
  - turn-taking: There are two players whose actions must alternate
  - zero-sum games: For example, if one player wins a game of chess by +1, then the other player necessarily loses -1.
  - perfect information: all current game states are fully observable

Search versus Games

- (Just) Search – no adversary 적대적인 탐색을 하지 않는다
  - Solution is a method for finding goal or a path to goal
  - Some Heuristics techniques can find *optimal* solution
  - Evaluation function is an estimate of cost from start to goal through a given node
  - Examples: path finding, 8-puzzle

- Games – adversary
  - Solution is strategy that specifies move for every possible opponent reply
  - In many cases, time limits force an *approximate* solution
  - Utility function is an evaluation about "goodness" of game state
  - Examples: chess, checkers, Othello, Go

Game Setup

- Two players(MAX and MIN) exist

- MAX moves first and they take turns until the game is over
  - Winner gets award, loser gets penalty.

- Games as search: 4 components
  - Initial state: e.g. board configuration of chess
  - Successor function: list of legal moves at the current game state
  - Terminal test: Is the game finished?
  - Utility function: Gives numerical value of terminal states. E.g. win (+1), lose (-1) and draw (0) in tic-tac-toe or chess

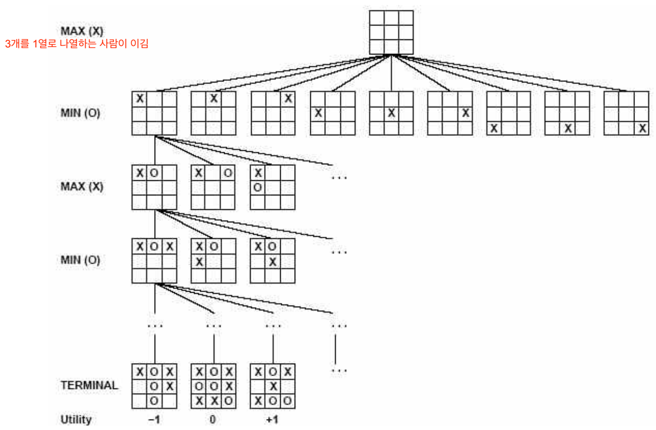
- So, MAX uses a usually very big search tree to determine next move

Size of search trees is estimated approximately

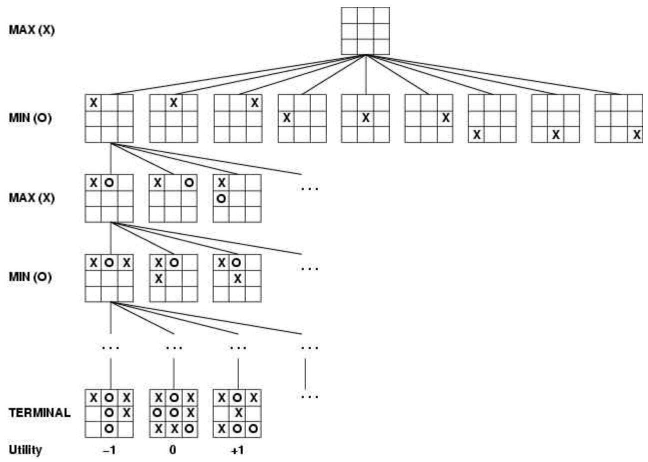
- b = branching factor
- d = the number of legal moves performed by both players
- Search tree is $O(b^d)$
- In Chess
  - b ~ 35
  - d ~100
    - search tree is ~ 10^{154} (extremely big !!!)
    - completely impractical to search this
- Game-playing emphasizes being able to make optimal decisions in a finite amount of time

이걸 배우면 알파고도 배울 수 있당

Partial Game Tree for Tic-Tac-Toe



Game tree (2-player, deterministic, turns)



An important problem: How can we search this big tree to find the optimal move?

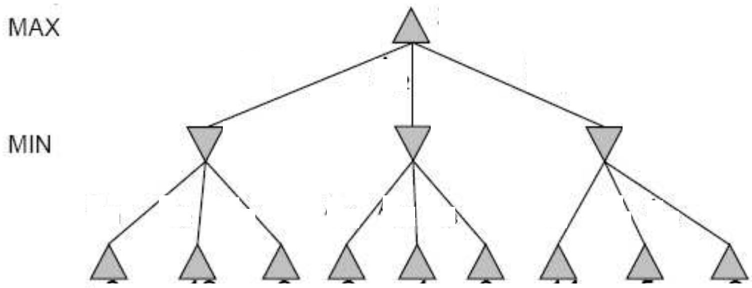
Minimax strategy상대방이 두는 노드를 min node

- Find the optimal *strategy* for MAX assuming an infallible MIN opponent
  - Need to compute this all the down the tree
- Assumption: Both players play optimally! → infallible palyers!
- Given a game tree, the optimal strategy can be determined by using the minimax value of each node:

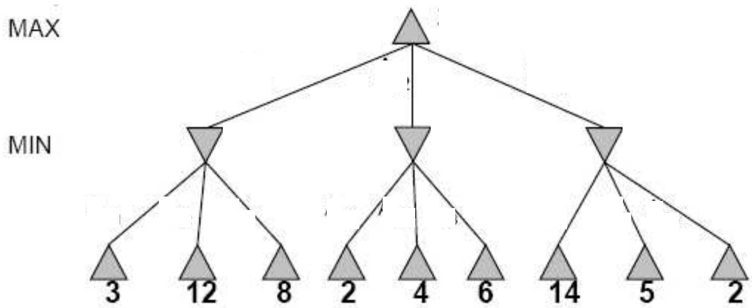
MINIMAX-VALUE(*n*)=

$$\text{MINIMAX-VALUE}(n) = \begin{cases} \text{UTILITY}(n), & \text{If } n \text{ is a terminal} \\ \max_{s \in successors(n)} \text{MINIMAX-VALUE}(s), & \text{If } n \text{ is a max node} \\ \min_{s \in successors(n)} \text{MINIMAX-VALUE}(s), & \text{If } n \text{ is a min node} \end{cases}$$

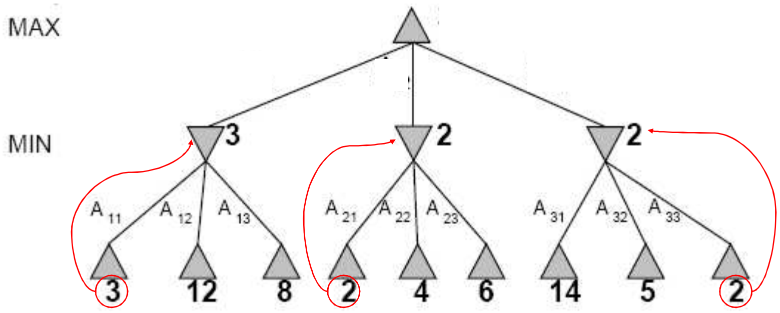
Two-Ply Game Tree



Two-Ply Game Tree



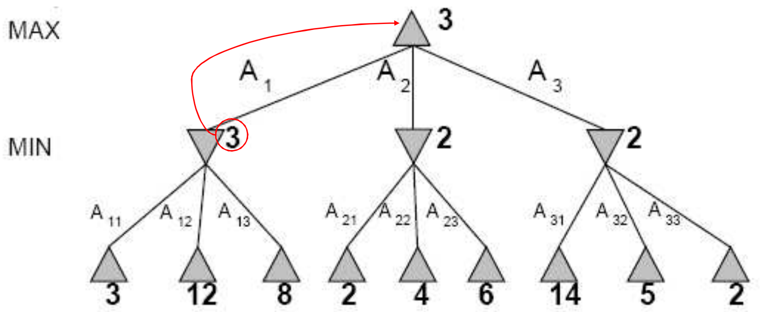
Two-Ply Game Tree



Two-Ply Game Tree

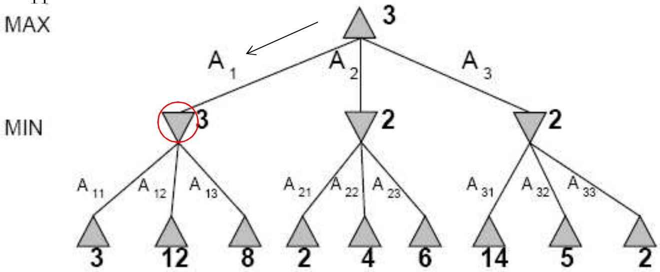
Minimax maximizes the utility for the worst-case outcome for max

The minimax decision: Just move to the leftmost successor!



What happens if MIN does not play optimally?

- Definition of optimal play for MAX assumes MIN plays optimally(infalliblely):
  - So, Minmax strategy maximizes worst-case outcome for MAX
- But even though MIN does not play optimally, MAX will do even better
  - Because MAX will be better if MIN moves to A_{12} or A_{13} instead of A_{11}



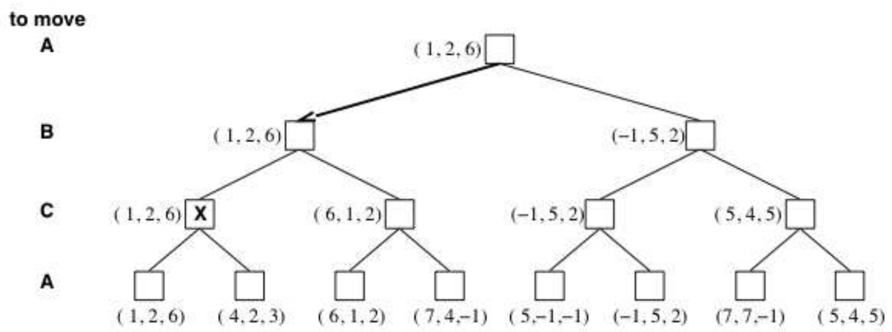
Minimax Algorithm

- Complete depth-first exploration of the game tree
- Assumptions:
  - Max depth = d, b legal moves at each node
  - E.g., Chess: d ~ 100, b ~35

Criterion	Minimax
Time	☹ $O(b^d)$
Space	$O(bd)$ ☺

Multiplayer games

- Games allow more than two players
- Single minimax values become vectors
  - e.g. for three palyers, just use a vector (UtilityForA, UtilityForB, UtilityForC>



Aspects of multiplayer games

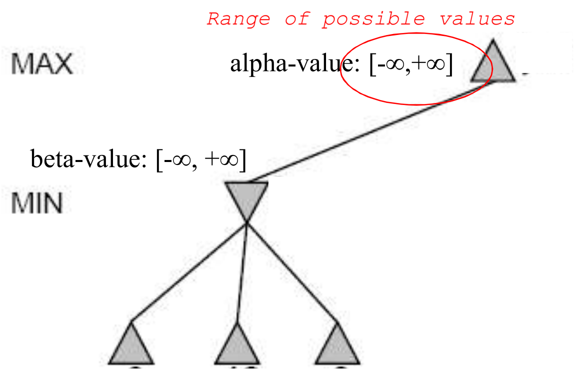
- Previous slide (standard minimax analysis) assumes that each player operates to maximize only their own utility
- In practice, players make alliances
  - e.g, C strong, A and B both weak
  - May be best for A and B to attack C together rather than each other
- If game is not zero-sum (i.e., utility(A) = - utility(B) then alliances can be useful even with 2 players
  - e.g., both cooperate to maximize the sum of the utilities
  - This makes minimax strategy so complicated!

Practical problem with minimax search

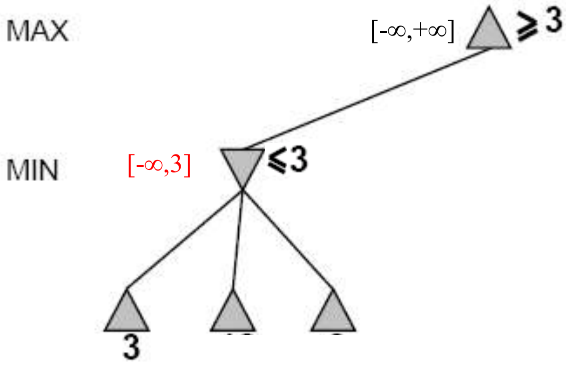
- Number of game states is exponential in the number of moves.
  - Solution: Do not search every node of a possible extremely big game tree => pruning !
    - Remove branches that do not influence final decision
- Revisit the example from the next slide …

Alpha-Beta Pruning Example

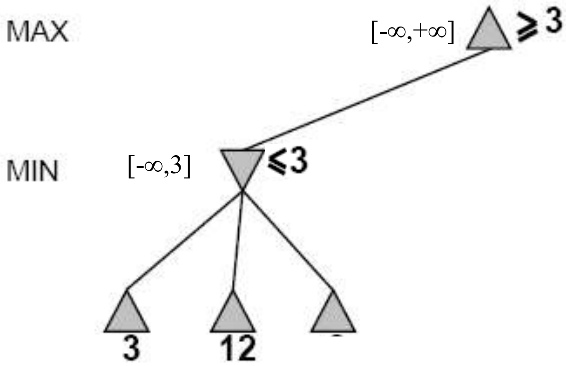
Do DF-search until first leaf



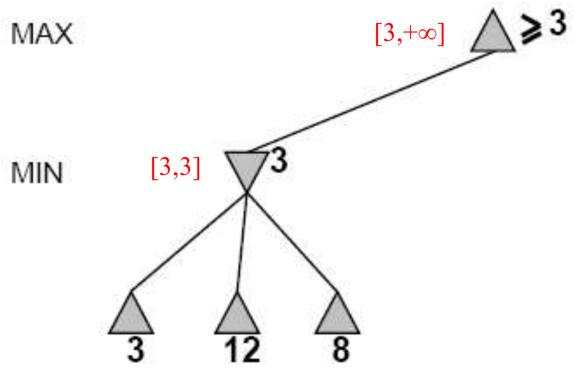
Alpha-Beta Pruning Example (continued)



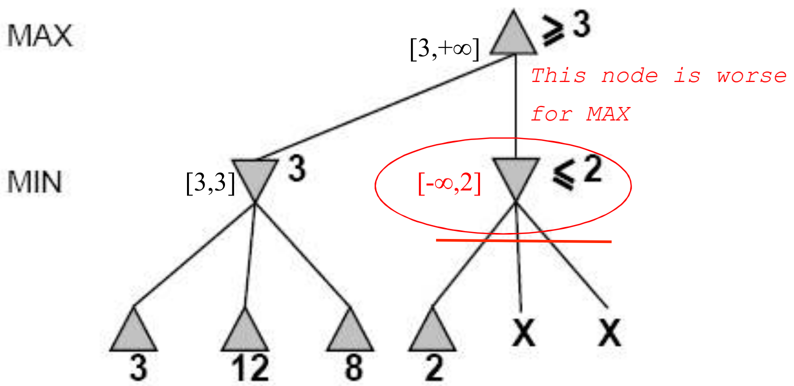
Alpha-Beta Pruning Example (continued)



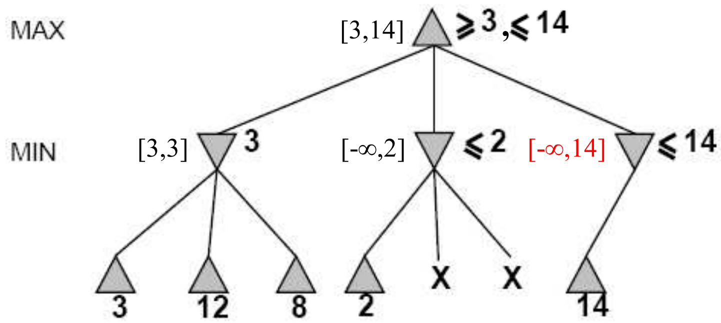
Alpha-Beta Pruning Example (continued)



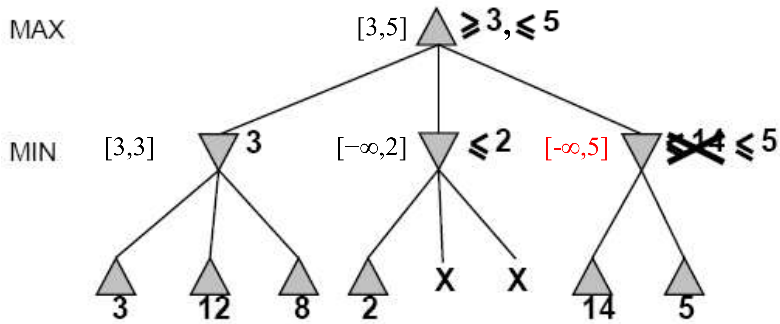
Alpha-Beta Pruning Example (continued)



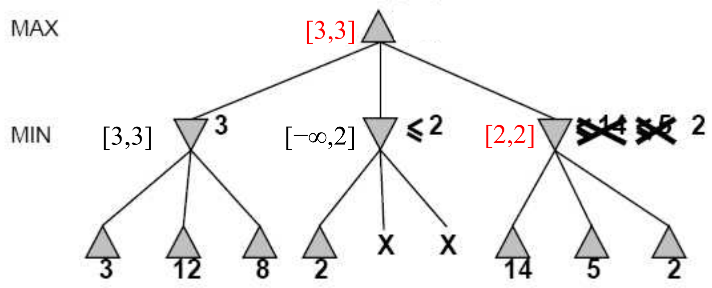
Alpha-Beta Pruning Example (continued)



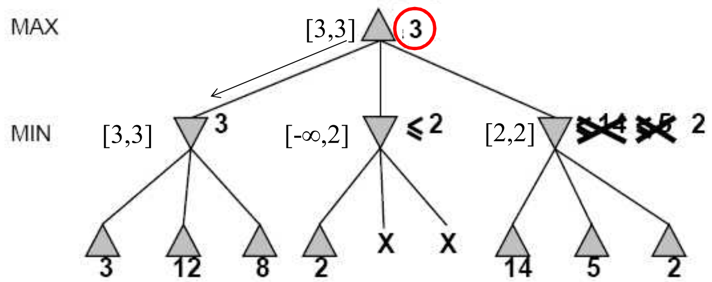
Alpha-Beta Pruning Example (continued)



Alpha-Beta Pruning Example (continued)



Alpha-Beta Pruning Example (continued)



Alpha-beta Pruning Algorithm - Summary

- Depth first search – only considers nodes along a single path at any time

  alpha-value = highest-value choice we have found at any choice point along the DFS path for MAX

  beta-value = lowest-value choice we have found at any choice point along the DFS path for MIN

- Update values of α–value and β–value during each search and prunes remaining branches as soon as the value is known to be worse than its anscester's α–value or β-value for MAX or MIN

Effectiveness of Alpha-Beta Pruning Search

- Worst-Case: $O(b^d)$
  - branches are very specially ordered so that no pruning takes place. In this case, alpha-beta pruning search gives no improvement over minimax search

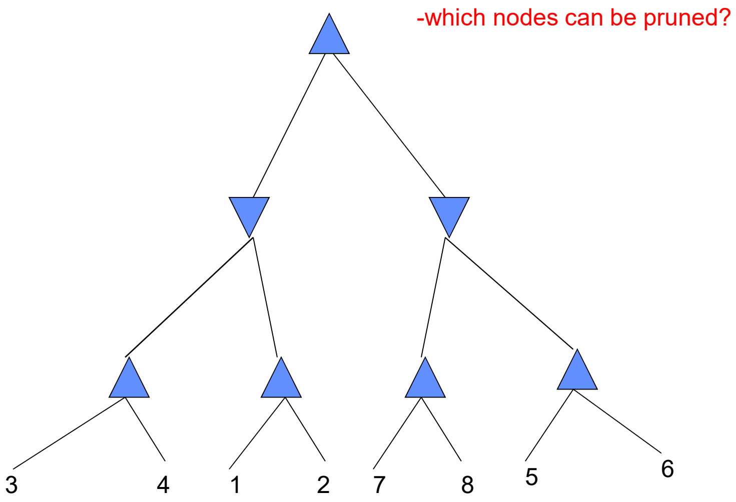
- Best-Case complexity: $O(b^{(d/2)})$
  - each player's best move is the left-most alternative so that every possible pruning takes place

- In practice, average performance is closer to the best-case one rather than worst-case
  - It means that alpha-beta pruning often get $O(b^{(d/2)})$ rather than $O(b^d)$ in practice
  - this is the same as having a branching factor of sqrt(b),
    - since $(\text{sqrt}(b))^d = b^{(d/2)}$
  - e.g., in chess go from b ~ 35 to b ~ 6
    - this permits much deeper search in the same amount of time
    - It means that alpha-beta pruning makes it possible for a player to move more optimally than a simple minimax search

Additional Comments about Alpha-Beta Pruning

- Alpha-beta pruning produces the same results as minimax algorithm
- In the best case, most of entire subtrees can be pruned
- Sometimes, good move *ordering* improves effectiveness of alpha-beta pruning

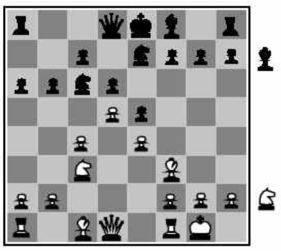
Example



Utility(or Evaluation) Functions

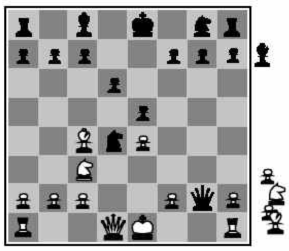
- A Utility(or Evaluation) Function:
  - estimates how good the current board configuration is for a player.
  - Typically, one figures how good it is for the player, and how good it is for the opponent, and subtracts the opponent's score from the player's score
  - Othello: Number of white pieces - Number of black pieces
  - Chess: (weighted sum of white pieces) – (weighted sum of black pieces)
- Typical values from -infinity (loss) to +infinity (win) or [-1, +1].
- If the board(game state) evaluation is X for a player at a time, it's -X for the opponent at that time
- Example:
  - Evaluating chess boards,
  - Checkers
  - Tic-tac-toe

Evaluation functions



Black to move

White slightly better



White to move

Black winning

For chess, typically *linear* weighted sum of features

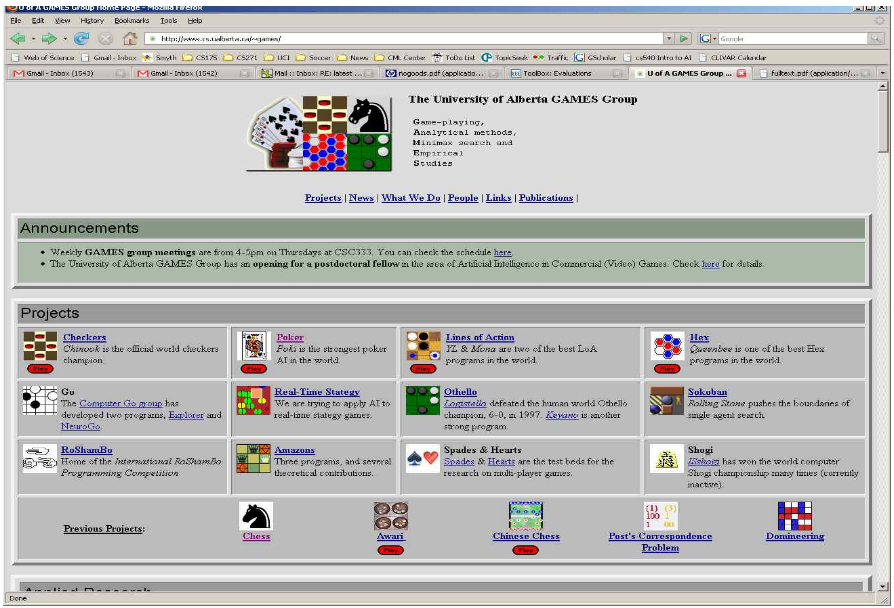
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

e.g., $w_1 = 9$ with
$f_1(s) =$ (number of white queens) – (number of black queens), etc.

Chapter 5, Sections 1–5 14

The State of Playing Games in the World

- Checkers:
  - Chinook(computer program) ended 40-year-reign of human world champion Marion Tinsley in 1994
- Chess:
  - Deep Blue(computer program based on alpha-beta search) defeated human world champion Garry Kasparov in a six-game match in 1997
- Othello:
  - human champions refuse to compete against computers: because computer programs are too much better than the human champions
- Go(바둑):
  - human champions refuse to compete against computers: computer programs are too much worse than the human champions
  - What does it means ? the size of game tree is too big to be searched (Imagine b > 300 and d >> 150 of Go, then 300^{150} (!)



See (e.g.) http://www.cs.ualberta.ca/~games/ for more information

Summary

- Game playing can be effectively modeled as a search problem
- Game trees represent alternate computer/opponent moves
- Evaluation functions estimate the quality of a given board configuration for the Max player
- Minimax is a procedure which chooses moves by assuming that the opponent will always choose the move which is best for them
- Alpha-Beta is a procedure which can prune large parts of the search tree and allow search to go deeper into game tree
- For many well-known games, computer algorithms based on the adversarial search out-perform human world champions