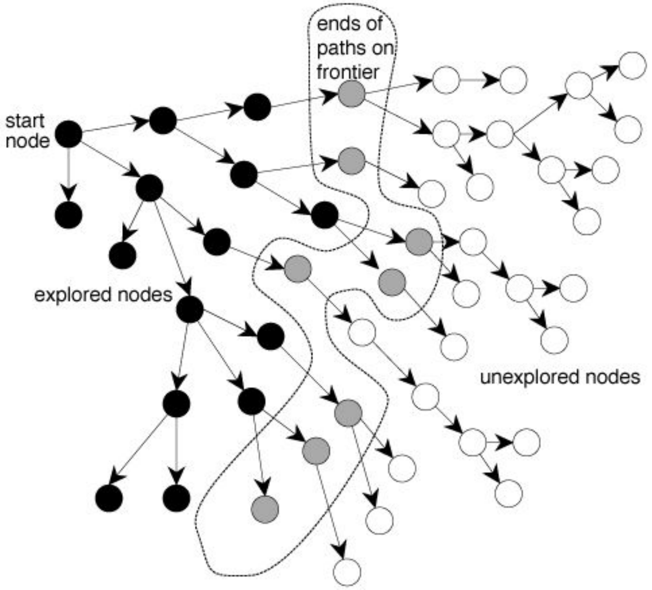
AI Course

Dr. Mürsel Taşgın

Game Playing

Game Playing

- Game playing is an application of informed searches
- Game playing can be considered as a graph search problem
    - Each node of the graph is a state in the game
- The goal nodes are the states when we win the game

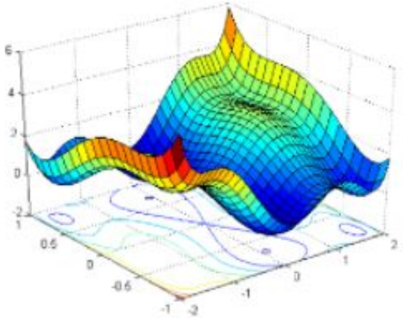


Game Playing - *Issues*

- Game-playing research includes the subjects:

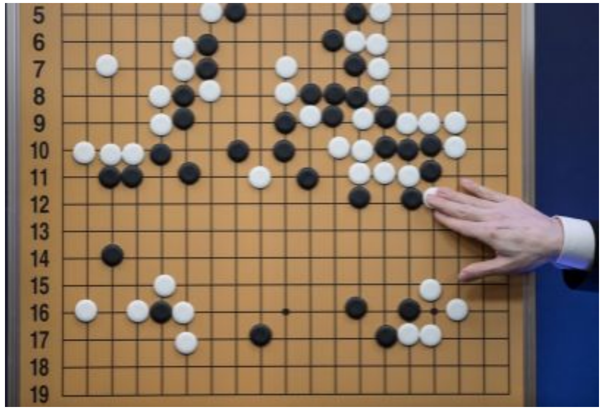
  - How to make the best use of time to reach a goal?

  - How to make a good decision, when reaching optimal decisions is impossible?

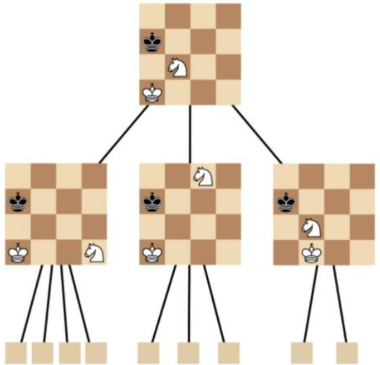


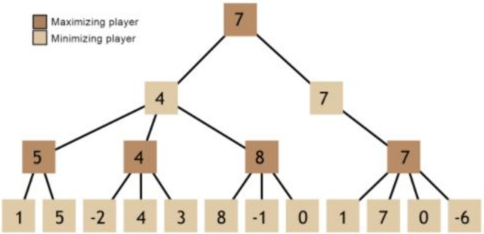
Game Playing vs Seach

- In the normal search problems, the search algorithm tries to find the **best** path to the goal state



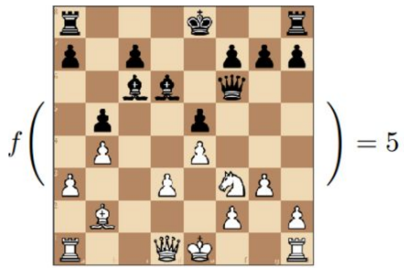
- In a game, the opponents make moves in turns
- Therefore, in a game, in one step we want to maximize a value, while in the next step we want to minimize it
- An algorithm named *min-max* algorithm is used for game playing



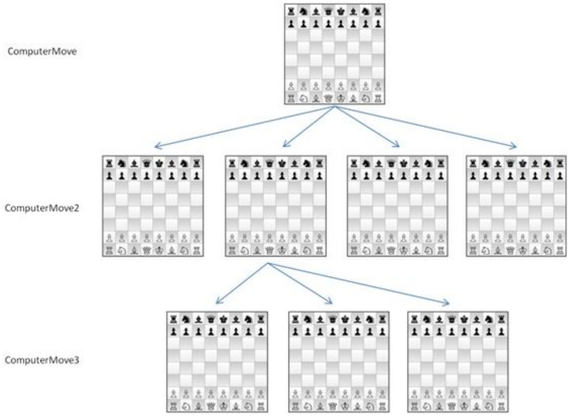


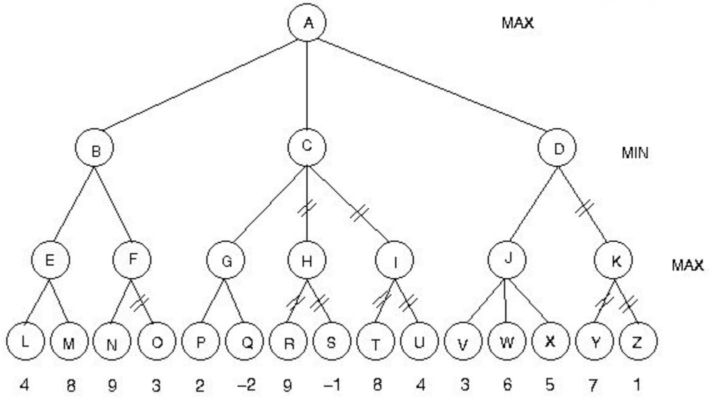
Game Playing – *Min-Max Search*

- Key idea: At each step choose the action which *minimizes* our *maximum* possible "loss" from making a particular move.
- For each move;
  - look ahead as many steps as our computing power will allow
  - examine all the possible moves our opponent could make in each of their future turns
  - make the move to minimize the possible maximum loss (in opponent’s move)
- Evaluation function: Takes the current state of the game and calculates a real-number value for the state



Garry Kasparov vs. Deep Blue





Game Playing – *Perfect Decision in 2-person Games*

- A game can be formally defined as a kind of search problem with the following components:
  - The **initial state**, which includes the game position and an indication of whose move it is
  - A set of **legal moves** that a player can make
  - A **terminal test**, which determines when the game is over
  - A **utility function** which gives a numeric value for each state of the game

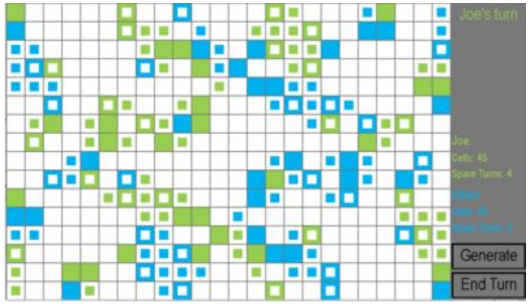
Game Playing – *Perfect Decision in 2-person Games (cont.)*

- In a normal search problem, we follow larger node values which means we are at a better state (close to win) to reach the goal □ (MAX step)

- In game search, MIN *(the opponent wants to win)* is also important

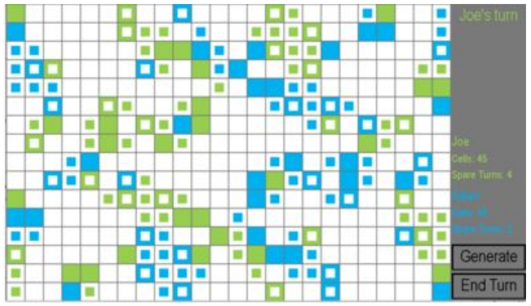
- Therefore we must find a *strategy* that will lead to a winning terminal state regardless of what MIN does

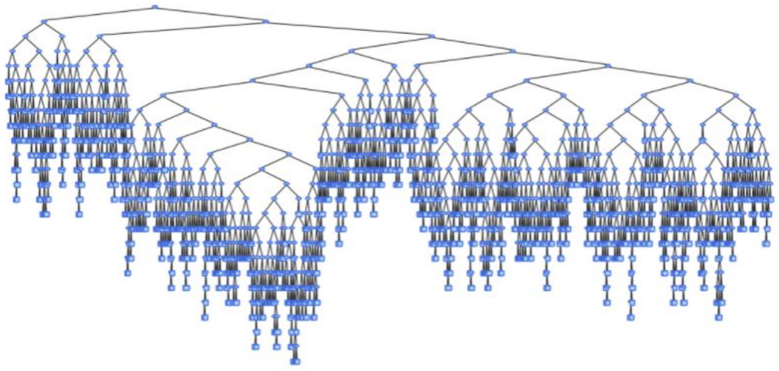
- This means we should choose the correct moves for MAX for each possible move of MIN



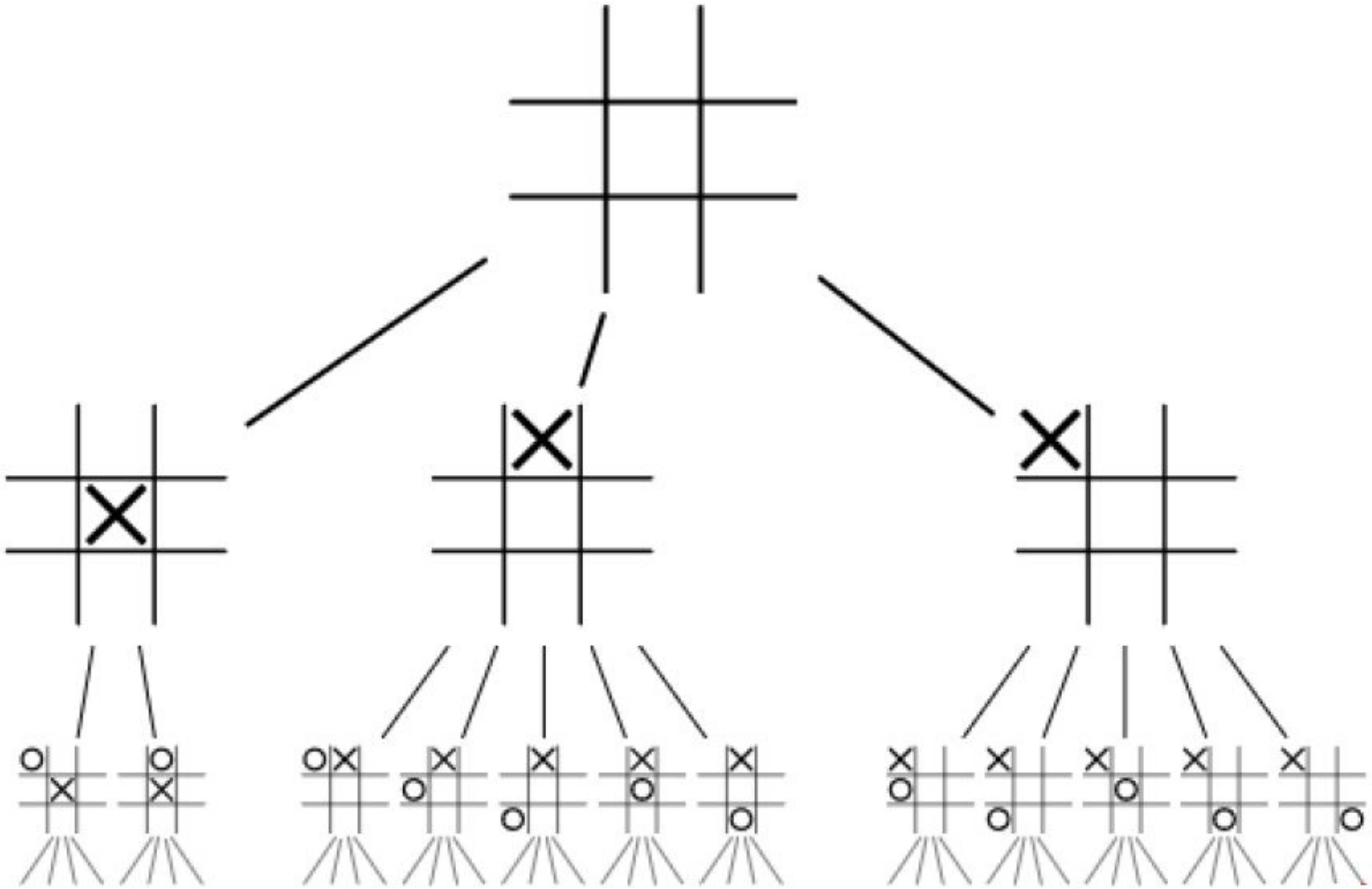
Game Playing – *Perfect Decision in 2-person Games (cont.)*

- In Perfect Decision using MIN-MAX Search we do:
  - Create the whole tree
  - Assign a value to each node using the utility function
  - Start from the goal node continue backward toward the root, one layer at a time using MAX-MIN values
  - Finally, at the root point, the *sequence to the goal* is found
  - This is called the perfect min-max decision, because it maximizes the utility under the assumption that the opponent will play perfectly to minimize it
  - It also assumes we can create the whole tree





Game Playing – *Tic-Tac-Toe Example*



Game Playing – Imperfect decisions

- The min-max algorithm assumes that the program has time to search all the way to the goal *(usually not practical)*

- Instead of using the utility function, the program should cut off the search earlier and apply a heuristic evaluation function to the leaves of the tree.

- We should alter min-max in two ways:
  1. The utility function is replaced by an evaluation function (EVAL)
  2. The terminal test is replaced by a cutoff test (CUTOFF)

Game Playing – *Utility Function, Evaluation Function*

- When we search in state space, we use the score of each state
- This score shows how close we are to the goal
- If the state space graph is small, we can calculate exact value of the score using Utility Function
- If the graph is too large, we use an estimation of the score
- The estimation is given by Evaluation Function which is a heuristic function

Game Playing – *Searching in Large State Spaces*

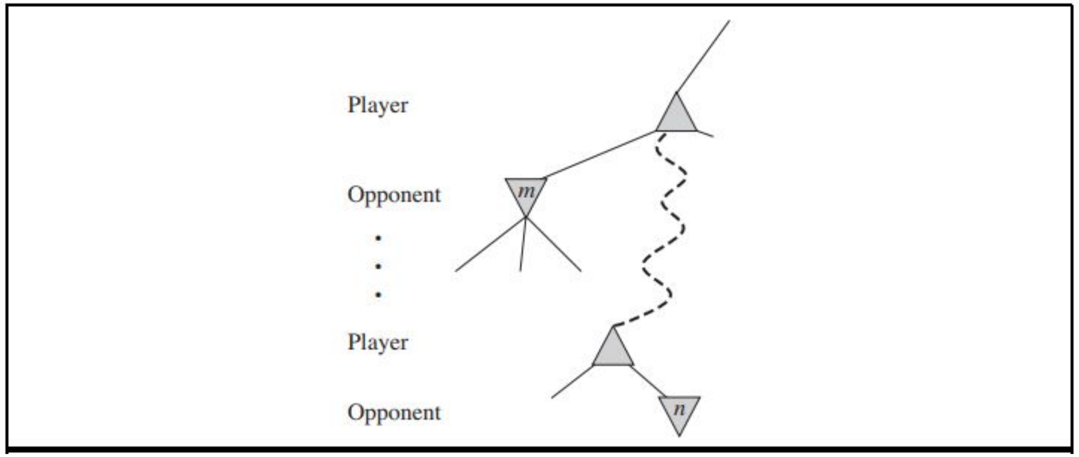
- If the state space is large, we cannot expand all nodes
- Therefore, search will be limited to a few levels
- For example, in chess game we cannot search until a goal node (win node)

Cutting Off Search

- The most common approach to control the search is to set a fixed depth limit
- The depth is chosen so that the amount of time used, will not exceed what the rules of the game allow
- A slightly better approach is to apply iterative deepening
- When time runs out, the program returns the move selected by the deepest completed search

Game Playing – *Alpha-Beta Pruning*

- Sometimes searching all branches of a tree is not possible (time is limited)
- In these cases we can cut some of the branches
- The remaining branches are searched in deeper levels
- One of the algorithms to cut (prune) branches of a tree is **Alpha-Beta Pruning**



If m is better than n for Player, we will never get to n

Game Playing – Alpha-Beta Pruning

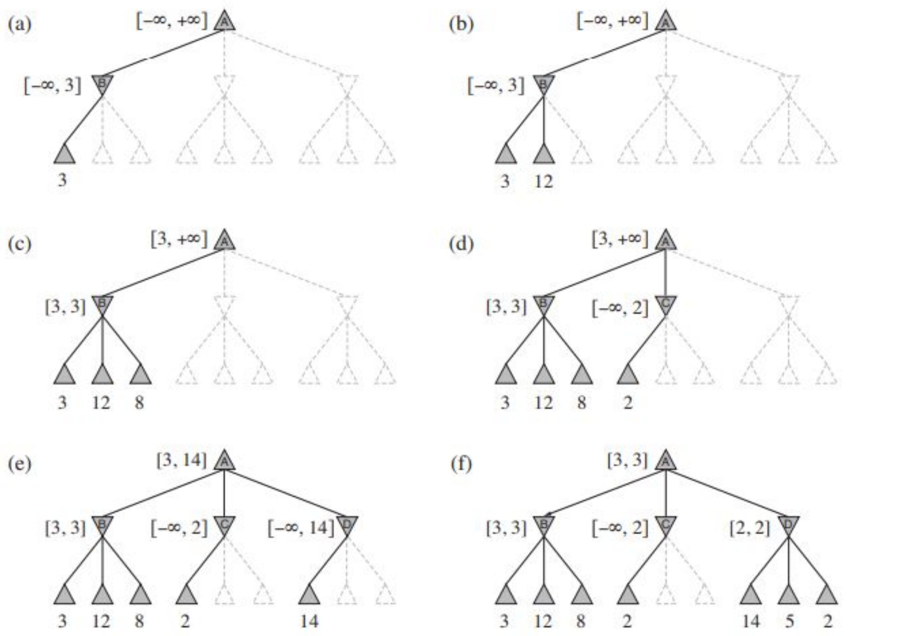
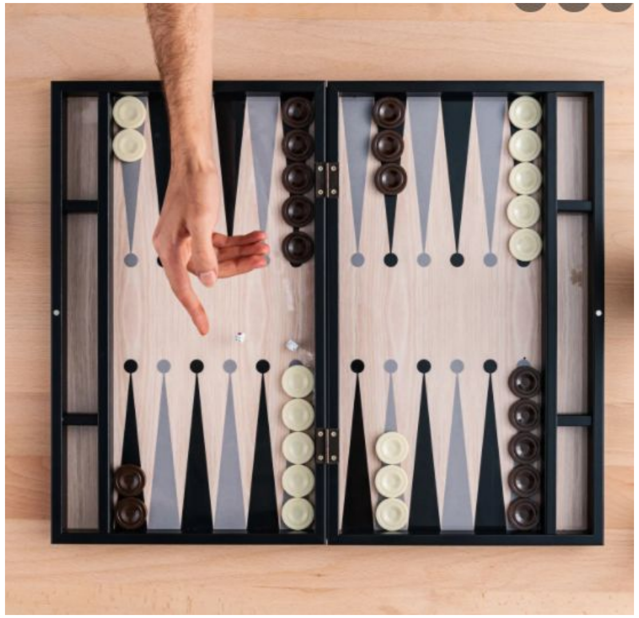


Figure 5.5 Stages in the calculation of the optimal decision for the game tree in Figure 5.2. At each point, we show the range of possible values for each node. (a) The first leaf below B has the value 3. Hence, B, which is a MIN node, has a value of *at most* 3. (b) The second leaf below B has a value of 12; MIN would avoid this move, so the value of B is still at most 3. (c) The third leaf below B has a value of 8; we have seen all B's successor states, so the value of B is exactly 3. Now, we can infer that the value of the root is *at least* 3, because MAX has a choice worth 3 at the root. (d) The first leaf below C has the value 2. Hence, C, which is a MIN node, has a value of *at most* 2. But we know that B is worth 3, so MAX would never choose C. Therefore, there is no point in looking at the other successor states of C. This is an example of alpha–beta pruning. (e) The first leaf below D has the value 14, so D is worth *at most* 14. This is still higher than MAX's best alternative (i.e., 3), so we need to keep exploring D's successor states. Notice also that we now have bounds on all of the successors of the root, so the root's value is also at most 14. (f) The second successor of D is worth 5, so again we need to keep exploring. The third successor is worth 2, so now D is worth exactly 2. MAX's decision at the root is to move to B, giving a value of 3.

Game Playing – Games with luck element

- Some games combine luck and skill
- Backgammon is a typical example
- Dice are rolled at the beginning of a player's turn to determine the set of legal moves that is available to the player
- After rolling the dice, the player knows his/her possible moves
- The player does not know what the other player will roll (and his legal moves)
- That means he cannot construct a complete game tree
- A game tree in these games must include **chance nodes** in addition to MAX and MIN nodes



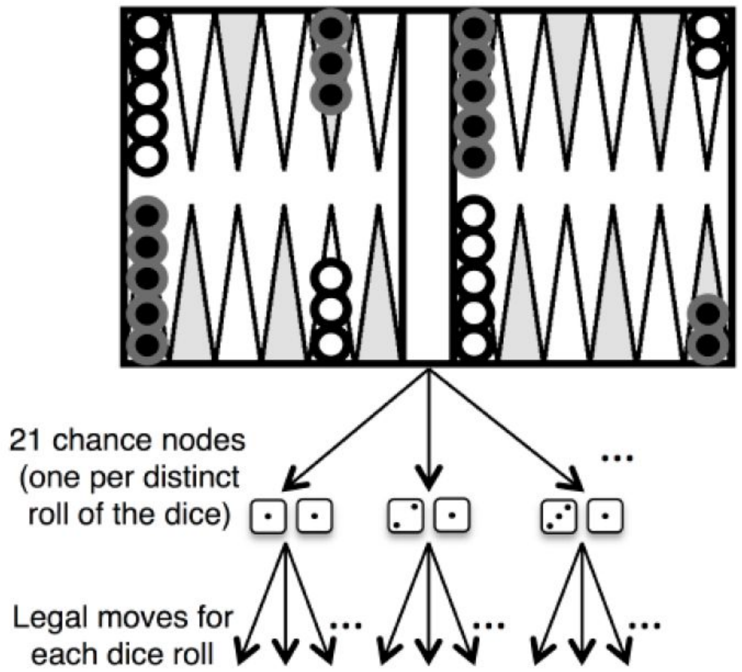
Game Playing – *Chance Nodes*

- In the backgammon example, white has rolled a 6-5, and has four possible moves

- Although white knows what his or her own legal moves are, white does not know what black is going to roll, and thus does not know what black's legal moves will be

- That means white cannot construct a complete game tree like we saw in chess and Tic-Tac-Toe

- A game tree in backgammon must include **chance nodes** in addition to MAX and MIN nodes.



Game Playing – *Summary*

- Searching in *Game Playing* is different than the general search in state space because our decision depends on the decision of our opponent

- In some games, we also have the chance element

- It is possible to cut some branches if we will never follow them. This makes search faster

- Search in game graphs is generally limited to a few levels