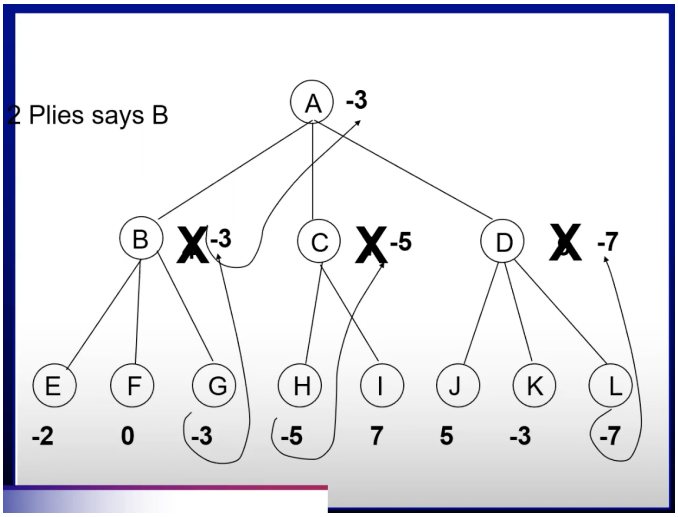
Day 13: Minimax Search & Knowledge-Search Tradeoff

- Game Tree
  - Graph abstraction of a game's state space, from the current player's POV
  - Usually not stored all in memory, just visited by a program in some order
  - Each node is labeled either by static evaluation or values "backed-up" from below
- Knowledge-Search Tradeoff
  - Foundational Principle of AI: The more you know, the less you have to search
  - In a minimax game, the "knowledge" static evaluation of how good a position is.
  - Perfect knowledge would be the same as fully unrolling the tree, which is impossible for complex games
  - The more plies you search, the better the player. Up until you bottom out, at which point you've just fully unrolled the tree.
  - **NB:** If you're looking at 2 plies down, then what you're actually trying to do is maximize your score on the 2nd ply, ignoring the 1st ply:



- Branch and Bound Algorithmic Technique
  - When exploring multiple paths, use knowledge of known paths to prune other branches.
    - If you can prove that Branch A can't possibly be better than Branch B, don't bother searching it.
  - Specialized case for games: Alpha/Beta Pruning
    - If we're guaranteed to score by making move A, don't bother searching move B, if we know that any descendent of move B will score less than A.
    - In theory, returns the same result as minimax but does so more efficiently because it doesn't expand paths that can't change the outcome.
    - Alpha is the greatest lower bound on my score (worst case for me)
    - Beta is the least upper bound on your score (best case for you)
    - Alpha/Beta are used recursively in a flip-flop fashion for alternating layers of the tree
    - *This still doesn't make a lot of sense to me*
- Complications of Games
  - In games where you exchange value, the heuristics are bumpy

    - To solve, you make assumptions to smooth out the heuristic
    - "Google 'quiescence'"
- Horizon Effect
    - No matter how deep you search, there could be a loss just beyond the horizon
    - Solution: do a deeper secondary search on a few final candidate moves
- Iterative Deepening: fusion of depth-first search and breadth-first search
    - First search to depth 1, then depth 2
    - Combined with caching, it's the same perf as DFS (?) and is very commonly used
- Big problem for AI: "Silo" problem: AI built for one thing is often completely useless for anything else.