Artificial Intelligence Programming

Utility and Decision Making

Chris Brooks

Department of Computer Science

University of San Francisco

22-2: Making decisions

- At this point, we know how to describe the probability of events occuring.
  - Or states being reached, in agent terms.
- Knowing the probabilities of events is only part of the battle.
- Agents are really interested in maximizing performance.
- Often, performance can be captured by *utility*.
- Utility indicates the relative value of a state.

22-3: Types of decision-making problems

- Single-agent, deterministic, full information, episodic
  - We've done this with the reflex agent
- Single-agent, deterministic, full information, sequential
  - We can use search here.
- Single-agent, stochastic, partial information, episodic
- Single-agent, stochastic, partial information, sequential
- multiple-agent, deterministic, full information, episodic

22-4: Types of decision-making problems

- Single-agent, deterministic, full information, episodic
  - We've done this with the reflex agent
- Single-agent, deterministic, full information, sequential
  - We can use search here.
- Single-agent, stochastic, partial information, episodic
  - We can use utility and probability here
- Single-agent, stochastic, partial information, sequential
  - We can extend our knowledge of probability and utility to a Markov decision process.
- multiple-agent, deterministic, full information, episodic (or sequential)
  - This is game theory

22-5: Expected Utility

- In episodic, stochastic worlds, we can use expected utility to select actions.
- An agent will know that an action can lead to one of a set S of states.
- The agent has a utility for each of these states.
- The agent also has a probability that these states will be reached.
- The *expected utility* of an action is:
- $\sum_{s \in S} P(s)U(s)$
- The principle of maximum expected utility says that an agent should choose the action that maximizes expected utility.

22-6: Example

- Let's say there are two levers.
  - Lever 1 costs \$1 to pull. With $p = 0.4$, you win \$2. With $p = 0.6$ you win nothing.
  - Lever 2 costs \$2 to pull. With $p = 0.1$ you win \$10. with $p = 0.9$ you lose \$1 (on top of the charge to pull).
- Should you a) pull lever 1 b) pull lever 2 c) pull neither?

22-7: Example

- EU(lever 1) = 0.4 * 1 + 0.6 * -1 = -0.2
- EU(lever 2) = 0.1 * 8 + 0.9 * -3 = 5.3
- EU(neither) = 0
- Lever 2 gives the maximum EU.

22-8: Example: Vegas

- The typical roulette wheel has 38 numbers. (1-36, plus 0 and 00).
- 1-36 are either red or black.
- The payoff for betting on a single number is 35:1
  - In other words, if the number you picked comes up, you win $35. Otherwise, you lose $1.
- What is the expected utility of betting on a single number?

22-9: Example: Vegas

- The typical roulette wheel has 38 numbers. (1-36, plus 0 and 00).
- 1-36 are either red or black.
- The payoff for betting on a single number is 35:1
  - In other words, if the number you picked comes up, you win \$35. Otherwise, you lose \$1.
- What is the expected utility of betting on a single number?
- $\frac{1}{38}35 + \frac{37}{38} - 1 = -0.052$

22-10: Example: Vegas

- The typical roulette wheel has 38 numbers. (1-36, plus 0 and 00).
- 1-36 are either red or black.
- The payoff for betting on a single number is 35:1
  - In other words, if the number you picked comes up, you win $35. Otherwise, you lose $1.
- What if you decide to “spread the risk” and bet on two numbers?

22-11: Example: Vegas

- The typical roulette wheel has 38 numbers. (1-36, plus 0 and 00).
- 1-36 are either red or black.
- The payoff for betting on a single number is 35:1
  - In other words, if the number you picked comes up, you win $35. Otherwise, you lose $1.
- What if you decide to “spread the risk” and bet on two numbers?
- $\frac{2}{38}34 + \frac{36}{38} - 2 = -0.105$

22-12: Example: Vegas

- The typical roulette wheel has 38 numbers. (1-36, plus 0 and 00).
- 1-36 are either red or black.
- The payoff for betting on color is 1:1
  - In other words, if you bet 'red' and a red number comes up, you win $1. Otherwise, you lose $1.
- What is the expected utility of betting on 'red'?

22-13: Example: Vegas

- The typical roulette wheel has 38 numbers. (1-36, plus 0 and 00).
- 1-36 are either red or black.
- The payoff for betting on color is 1:1
  - In other words, if you bet 'red' and a red number comes up, you win $1. Otherwise, you lose $1.
- What is the expected utility of betting on 'red'?
- $\frac{18}{38}1 + \frac{20}{38} - 1 = -0.052$

22-14: Regarding Preferences

- In order for MEU to make sense, we need to specify some (hopefully reasonable) constraints on agent preferences.
- Orderability. We must be able to say that A is preferred to B, B is preferred to A, or they are equally preferable. We cannot have the case where A and B are incomparable.
- Transitivity. If $A \prec B$ and $B \prec C$, then $A \prec C$.
- Continuity. If $A \prec B \prec C$, then there is a scenario where the agent is indifferent to getting B and having a probability p of getting A and $1 - p$ chance of getting C.

22-15: Rational Preferences

- Monotonicity. If two actions A and B have the same outcomes, and I prefer A to B, I should still prefer A if the probability of A increases.
- Decomposability. Utilities over a sequence of actions can be decomposed into utilities for atomic events.
- These preferences are (for the most part) quite reasonable, and allow an agent to avoid making foolish mistakes.

22-16: Utility, Money, and Risk

- Utility comes from economics
  - Money is often used as a substitute for utility.
- Preferences for money behave oddly when dealing with small or large amounts.
- For example, you will often take more chance with small amounts, and be very conservative with very large amounts.
- This is called your *risk profile*
  - convex - risk-seeking
  - concave, risk-averse
- Typically, we say that we have a *quasilinear* utility function regarding money.

22-17: Gathering information

- When an agent has all the information it can get and just needs to select a single action, things are straightforward.
  - Find the action with the largest expected utility.
- What if an agent can choose to gather more information about the world?
- Now we have a sequential decision problem:
  - Should we just act, or gather information first?
  - What questions should we ask?
    - Agents should ask questions that give them useful information.
    - “Useful” means increasing expected utility.
  - Gathering information might be costly, either in time or money.

22-18: Example

- An agent can recommend to a prospecting company that they buy one of n plots of land.
- One of the plots has oil worth C dollars; the others are empty.
- Each block costs C/n dollars.
- Initially, agent is indifferent between buying and not buying. (why is that?)

22-19: Example

- Suppose the agent can perform a survey on a block that will indicate whether that block contains oil.
- How much should the agent pay to perform that survey?
- $P(oil-in-block) = 1/n$. $P(\neg oil-in-block) = (n-1)/n$
  - If oil found, buy for C/n. Profit = $C - C/n = (n-1)C/n$
  - If oil not found, buy a different block.
  - Probability of picking the oil block is now: $1/(n-1)$ Expected Profit: $C/(n-1) - C/n = C/(n*(n-1))$.
- So, the expected profit, given the information is:
- $\frac{1}{n}\frac{(n-1)C}{n} + \frac{n-1}{n}\frac{C}{n(n-1)} = \frac{C}{n}$
- So the company is willing to pay up to C/n (the value of the plot) for the test. This is the *value of that information*.

22-20: Value of Perfect Information

- Let's formalize this.
- We find the best action α in general with:
- $EU(\alpha) = max_{a \in actions} \sum_{i \in outcomes} U(i)P(i|a)$
- Let's say we acquire some new information E.
- Then we find the best action with:
- $EU(\alpha|E) = max_{a \in actions} \sum_{i \in outcomes} U(i)P(i|a, E)$
- The value of E is the difference between these two.

22-21: Value of Perfect Information

- However, before we do the test, we don't know what E will be.
- We must average over all possible values of E.
- $VPI(E) = (\sum_{j \in values_E} P(E = j)EU(\alpha|E = j)) - EU(\alpha)$
- In words, consider the possibility of each observation, along with the usefulness of that observation, to compute the expected information gain from this test.
- In general, information will be valuable to the extent that it changes the agent's decision.

22-22: Example

- Imagine that you are on a game show and are given a choice of three possible doors to open.
- If you open door number 1, you will win $10.
- If you open door number 2, you have a 50% change of winning nothing, and a 50% chance of winning $25.
- If you open door number 3, you have a 20% of winning $20, and an 80% chance of winning $10.
- Which door should you choose?

22-23: Example

- EU(door1) = 10
- EU(door2) = 0.5 * 0 + 0.5 * 25 = 12.5
- EU(door3) = 0.2 * 20 + 0.8 + 10 = 12
- Door 2 is best.

22-24: Example

- Now, suppose that the host offers to tell you honestly what you'll get if you open door number 2. How much would you pay for this information?

22-25: Example

- There are two cases: either door 2 pays 0 or it pays 25.
- If it pays 25, we should choose it. This happens 50% of the time.
- If it pays 0, we should choose door3, which pays 12. This happens 50% of the time.
- So, our utility will be: 0.5 * 25 + 0.5 * 12 = 18.5
- Our EU before we asked was 12.5, so the information is worth 6.

22-26: Dealing with multiple agents

- Value of information shows how an agent can rationally 'look ahead' in a partially observable world.
- Looking ahead is also valuable in multi-agent environments.
  - Anticipate your opponent's moves, or his reaction to your moves, or his reaction to your reaction to his reaction ...

22-27: Multi-agent Environments

- Competitive environments
  - Chess, checkers, go, ...
  - Auctions, online markets
- Cooperative Environments
  - Agents on a soccer team
  - Rovers surveying Mars
- And in-between cases
  - Two agents making deliveries

22-28: Game Theory

- *Game Theory* is the branch of mathematics/economics that deals with interactions between multiple agents.
- Prescribes methods for determining optimal behavior.
- Distinctions:
  - Games of perfect information: Agents have access to all knowledge about the world.
    - We'd call this a fully observable environment.
    - Board games without randomness, paper-scissors-rock
  - Games of imperfect information: An agent must make inferences about the world or its opponent.
    - We call this a partially observable environment.
    - Games of chance, some auctions, most real-world interactions

22-29: Game Theory

- Game theory assumes:
  - Perfect rationality - agents will always do the right thing.
  - Unlimited computation - agents are always able to determine the right thing to do.
- As we've seen, these assumptions may not make sense in some cases.
- However, we'll still use some ideas from game theory to help decide what to do.
- So how does an agent determine what do do in a multiagent environment?

22-30: Optimal Strategies

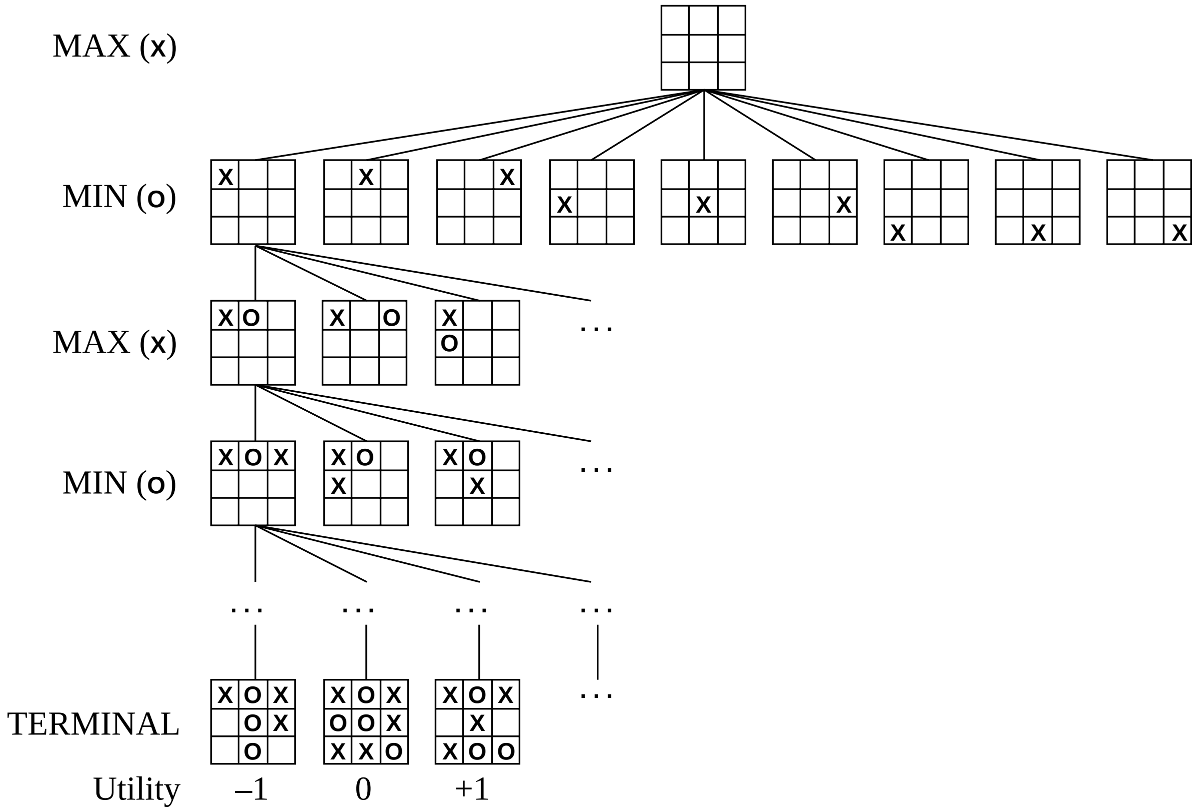
- We'll start with the simplest sort of game.
  - Two players
  - Perfect information
  - Zero-sum (one wins, one loses) - this is also called *perfectly competitive*
- We can define this game as a search problem.

22-31: Optimal Strategies

- Two players: max and min
  - Initial state: game's initial configuration
  - Successor function - returns all the legal moves and resulting configurations.
  - Terminal test - determines when the game is over.
  - Utility function - amount 'won' by each player,
    - In zero-sum games, amounts are +1 (win), -1 (loss), 0 (draw)

22-32: Game Trees

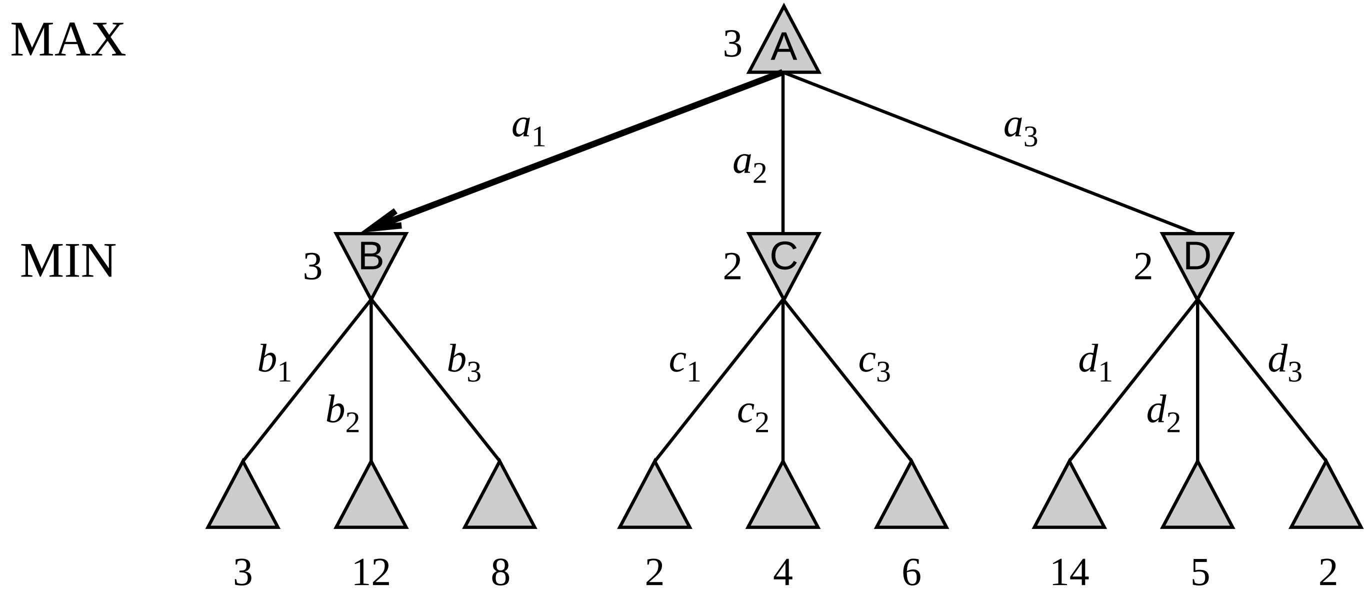
- We can then (in principle) draw the *game tree* for any game.



- Utilities for terminal states are from the perspective of player 1 (max).

22-33: Finding a strategy

- Max cannot just search forward and find the path that leads to a winning outcome.
- Max must take the fact that min is also trying to win into account.
- Consider the following even simpler search tree:

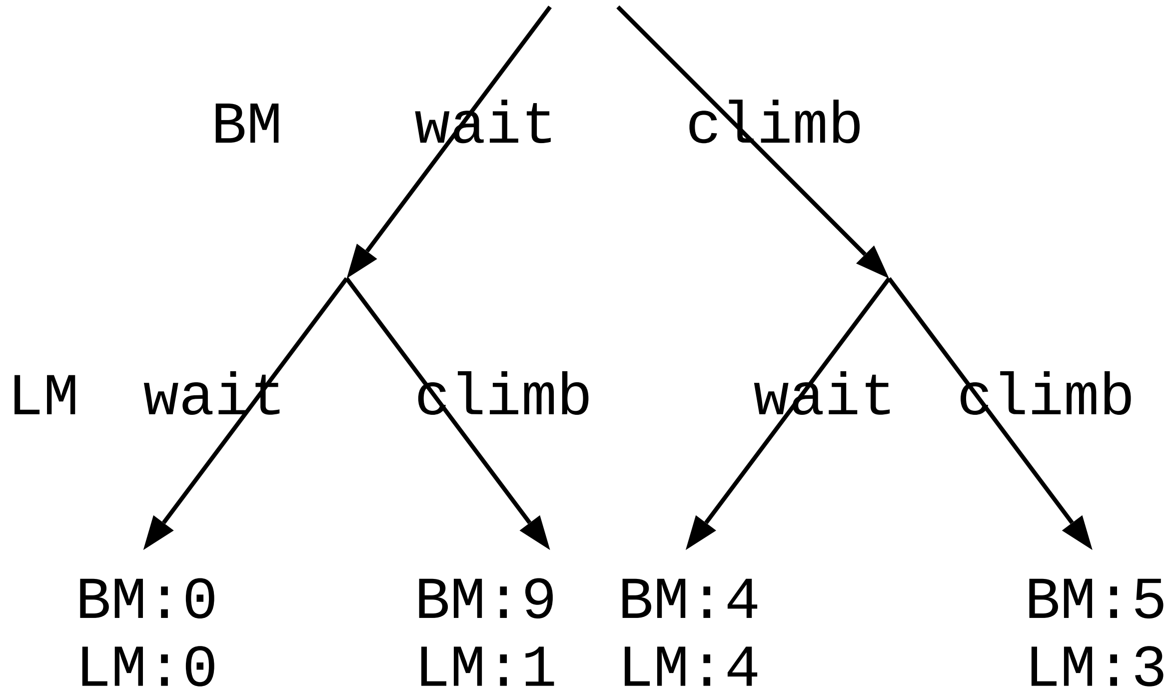


22-34: Single-stage games

- This approach is easiest to see with a single-stage game.
- Consider the big-monkey/little-monkey problem:
- There is 10 Cal worth of fruit in the tree.
- Big Monkey spends 2 Cal getting fruit, Little Monkey 0.
- If neither gets fruit, both get 0.
- If both climb, BM gets 7, LM gets 3.
- If BM climbs and LM waits, BM gets 6, LM 4.
- If LM climbs and BM waits, BM gets 9, LM gets 1.

22-35: Single-stage games

- What if Big Monkey chooses first?



- If BM waits, LM will climb. BM gets 9
- If BM climbs, LM will wait. BM gets 4.
- BM will wait.

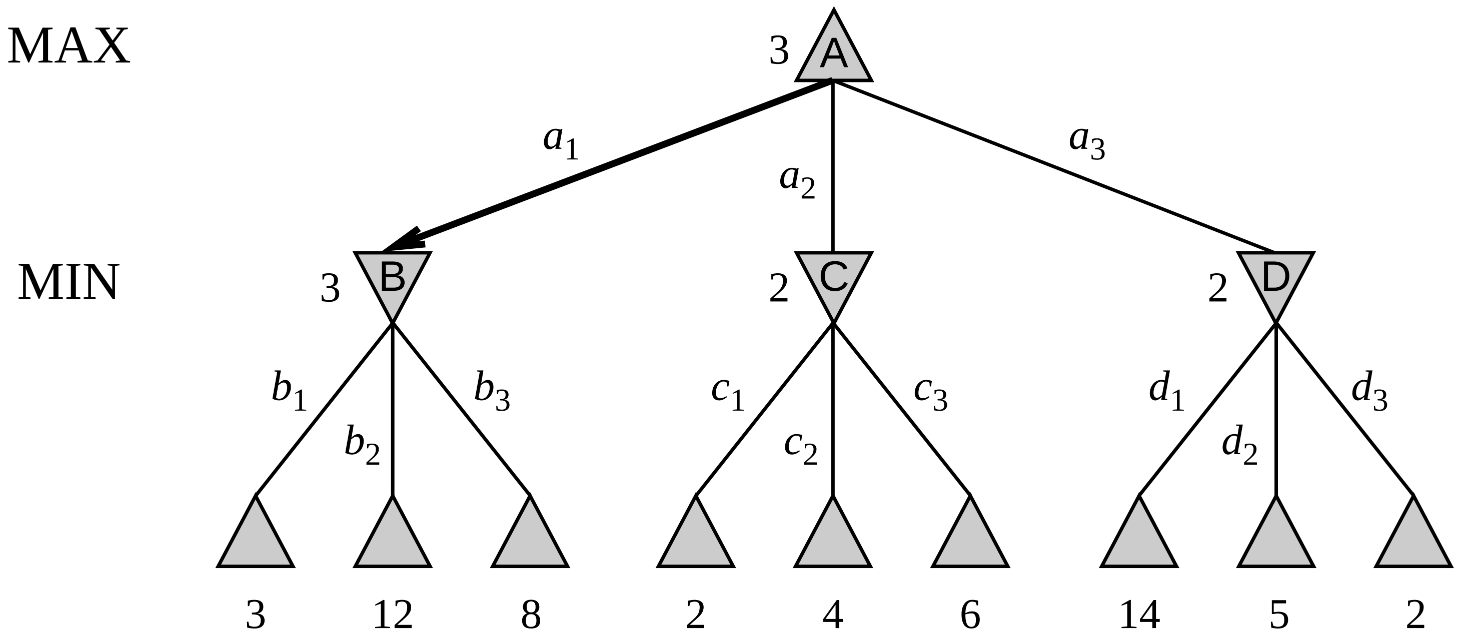
22-36: Single-stage games

- This analysis only works if the monkeys choose sequentially.
- If they choose simultaneously, things can become more complicated.
- Each must consider the best action for each opposing action.
- We'll focus on sequential games.

22-37: Dealing with multiple stages

- To deal with games where many actions are taken, begin at the leaves and compute their payoff.
- The minimax value of a node is the payoff (from that player's perspective) assuming both players play optimally from that point on.
- Computed recursively.
- Begin with terminal nodes.- are known.
- Minimax-value (node) =
  - max(children) if it's the max player's turn
  - min(children) if it's the min player's turn.

22-38: Example



- Terminal utilities are given.
- Recurse up to the 'Min' layer (or *ply*)
  - Min player will choose actions b_1, c_1, d_3.
- Recurse again to 'Max' layer.
  - Max player will choose action a_1.

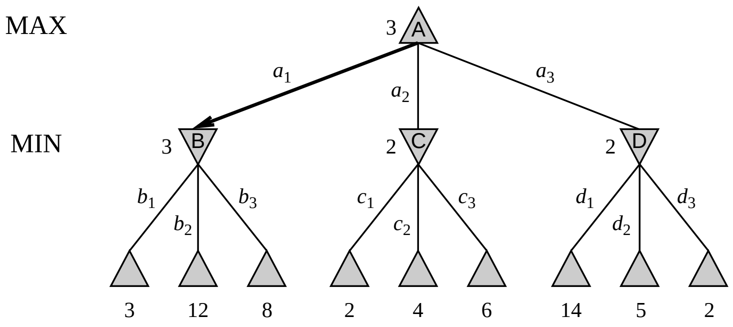
22-39: Minimax search

- Perform a depth-first traversal of the search tree.
- Once the values for a node's children have been computed, the value for that node can then be computed.
- Retain only the best action sequence so far.
- This will find the optimal strategy
- Problem:
  - Totally impractical for large games.
- Serves as a basis fore more practical approaches.

22-40: Alpha-beta pruning

- Recall that *pruning* is the elimination of branches in the search tree without expansion.
    - You can rule out the possibility that the solution is down one branch.
- Consider the previous example again, remembering that non-terminal node values are generated via depth-first search.

22-41: Alpha-beta pruning



- After traversing the left branch, we know that node B has a value of 3
  - The Min player will pick 3 over 12 and 8
  - More generally, the min player will pick the lowest value.
- As soon as we find a 2 down the center Branch, we can prune the other children.
  - Since Min would pick the smallest value down this branch, C can have a value of *at most* 2.
  - Node B already has a value of 3, so Max would prefer this.
- Still must traverse the third branch.

22-42: Alpha-beta pruning

- Intuition behind alpha-beta:
- Consider a node n - if the player considering n can make a better move at n *or one of its parents*, then n will never be reached.
- Call Alpha the best (max) value at a choice point
- Call beta the lowest (min) value at a choice point.

22-43: Alpha-beta pruning

- Alpha-beta is very sensitive to the order in which nodes are enqueued.
- What if nodes B and C were swapped in our previous example?
- Heuristics can be used to reorder the tree and search promising branches first.
- Problem: Alpha-beta still needs to descend to a terminal node before states can be evaluated.

22-44: Evaluation Functions

- For large problems (like chess) you need to be able to estimate the value of non-terminal states.
- Construct a heuristic known as an *evaluation function* that estimates the terminal utility of that state.
- How to build such a function?
  - Should order states according to the utility of their terminals.
  - Should be fast.
  - Should be strongly correlated with actual chances of winning.

22-45: Evaluation Functions

- Evaluation functions introduce uncertainty.
- You’d like to have your function “guess correctly” most of the time.
- Place states in categories or classes; states in a category win X% of the time.
  - This is a *classification* problem.
- Alternatively, construct a domain-specific value for a state.
  - In chess, give pieces point values, assign weights to squares controlled.
  - How to get these numbers?
    - Ask an expert
    - Run minimax offline on different board configurations.

22-46: Cutting off search

- Evaluating non-terminal nodes is fine, but we still need to anticipate future moves.
- When can we cut off search and have a good idea of a branch's value?
- Approach 1: Use a fixed depth of lookahead.
- Approach 2: Use iterative deepening until our time to make a decision runs out.
- Both of these approaches are vulnerable to the *horizon problem*
  - Things look good at this state, but will go horribly wrong next turn.

22-47: Quiescence

- We want to apply our evaluation function only to search paths that are *quiescent*
  - Aren't changing wildly.
  - Don't stop in the middle of a sequence of captures, for example.
- If our estimate of the state is changing drastically as we expand, we need to search further down that branch to get an accurate estimate.
- We still may not avoid the horizon problem
- There may be a move that is ultimately unavoidable, but can be prolonged indefinitely.
  - Search must be able to distinguish between delaying and avoiding an outcome.

22-48: Performance improvements

- May choose to focus search more deeply down moves that are clearly better than other.
- Can also store visited states
  - Avoid repeated paths to the same state
  - Avoid symmetric states
- High-performance game-playing systems will typically have a large dictionary of pre-computed endgames.
- Once a game reaches this state, optimal moves are read from a hashtable.

22-49: Deep Blue

- Deep Blue is a great example of the combination of modern computing power and AI techniques.
- 1997: Deep Blue defeats Garry Kasparov, clearly the best human chess player.
    - This has been an AI goal since the 50s.
- Deep Blue is a 32-Node RS/6000 supercomputer
- Software written in C
- Uses alpha-beta search

22-50: Deep Blue

- Specialized hardware for computing evaluation functions.
- Evaluates 200,000,000 states/second
  - Evaluates 100-200 billion states in 3 minutes (the time alloted)
  - Average branching factor in chess is 35, average game is 50 moves
  - State space is around 10^{70}
- Large amount of specialized domain knowledge from humans used to construct accurate evaluation functions.
- Maintains a database of opening moves used by grandmasters in previous games.
- Also a large 'book' of endgames.

22-51: Deep Blue

- This is nothing like how humans play chess.
  - Kasparov claims to consider 2-3 states per second.
  - Humans seem to use pattern-matching to recognize good moves.
- Deep blue is intelligent in the sense that it *acts rationally*, but not that it thinks or acts like a human.

22-52: Dealing with chance

- Extending minimax to deal with chance is straightforward.
- Add a “chance” player between max and min who can affect the world.
- Each player must now act to maximize *expected* payoff.
- Evaluation becomes more difficult, and the search tree explodes.
- There is also a distinction between random events (such as dice rolls) and deterministic but unknown variables (such as cards in an opponent’s hand).
  - In the second case, we can deduce information to help us eliminate uncertainty.

22-53: Other game-playing AI successes

- Checkers.
  - By 1994, Chinook could defeat Dr. Marion Tinsley, who'd been world checkers champion for 40 years.
- Othello
  - World champion defeated in 1997.
  - It's generally acknowledged that Othello is "small enough" to play optimally.
- Backgammon
  - TD-Gammon ranked among top three players in the world.
  - Doesn't use alpha-beta - uses shallow search with a very accurate evaluation function learned by playing against itself.

22-54: Other game-playing AI successes

- Go.
  - Still open - computers can beat amateurs, but not top professionals.
  - Challenges: no hierarchy of pieces, hard to focus search on board sections, branching factor of 361
- Bridge
  - GIB competes with human world champions (12th out of 35)
  - Uses a set of scripts to plan bidding strategies
  - Learns generalized rules.

22-55: AI and games

- Minimax and gametree search is great for solving sequential-move games.
- This works well for chess, checkers, etc, but is not very relevant to Halo.
- What are the AI problems for modern gaming?

22-56: AI and games

- Until recently, “AI” behavior in games was typically pretty primitive.
  - Emphasis was on graphical rendering
  - Very few cycles left for AI.
  - Assumption was that “suspension of disbelief” would be enough.
- NPC behavior typically governed by simple rules or finite state machines.
- Now, graphics cards are powerful enough to provide an opportunity to incorporate AI into the game itself.

22-57: AI needs in gaming

- Modern gaming could use:
  - Adaptive behavior
  - Rich, human-like interaction (NPCs with their own motivations/goals)
  - A “middle ground” between scripted stories and completely open-ended worlds.
  - Intelligent group/unit behavior.
  - Better AI opponents.

22-58: Promising games for adding AI

- Quake II/Half-life - allows you to write a DLL that governs agent behavior
- Descent 3 - also allows a DLL
- Age of Kings - allows you to edit opponent's strategic rules
- FreeCiv - lets you modify the rules governing AI behavior.