Recap
We’ve seen two kinds of search strategies so far:

✧ Uninformed search
  - Breadth-first search
  - Depth-first search

✧ Informed search
  - Uniform cost search
  - A* search
Adversarial Search
So far, we have only considered one-player games. What happens when we add another player?
In competitive multiplayer games, we have to consider our opponent’s possible actions, as well as our own.

We call this adversarial search.
Game Playing State-of-the-Art

- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!

- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

- **Go:** 2016: Alpha GO defeats human champion. Uses Monte Carlo Tree Search + neural network to learn evaluation function.

- **Go + Chess + Shogi:** 2017: Alpha Zero learns all 3 games using reinforcement learning to play against itself.

Slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley
Many different kinds of games!

Axes:
- Deterministic or stochastic?
- One, two, or more players?
- Zero sum?
- Perfect information (can you see the state)?

Want algorithms for calculating a strategy (policy) which recommends a move from each state
Deterministic Games

- Many possible formalizations, one is:
  - States: $S$ (start at $s_0$)
  - Players: $P=\{1…N\}$ (usually take turns)
  - Actions: $A$ (may depend on player / state)
  - Transition Function: $S \times A \rightarrow S$
  - Terminal Test: $S \rightarrow \{t,f\}$
  - Terminal Utilities: $S \times P \rightarrow \mathbb{R}$

- Solution for a player is a policy: $S \rightarrow A$
Zero-Sum Games

- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition

- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, and more are all possible
Single-Agent Trees

Slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley
Value of a State

Slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley
Adversarial Game Trees

Pacman's

Ghost torn

Slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley
Minimax Values

States Under Agent’s Control:

-8  -5

States Under Opponent’s Control:

-10  +8
Tic-Tac-Toe Game Tree

Slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley
Adversarial Search (Minimax)

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result

- Minimax search:
  - A state-space search tree
  - Players alternate turns
  - Compute each node’s minimax value: the best achievable utility against a rational (optimal) adversary

Minimax values: computed recursively

Terminal values: part of the game
Minimax Implementation

**Def max-value(state):**
initialize \( v = -\infty \)
for each successor of state:
\( v = \max(v, \min-value(successor)) \)
return \( v \)

**Def min-value(state):**
initialize \( v = +\infty \)
for each successor of state:
\( v = \min(v, \max-value(successor)) \)
return \( v \)

\[ V(s) = \max_{s' \in \text{successors}(s)} V(s') \]

\[ V(s') = \min_{s \in \text{successors}(s')} V(s) \]
def value(state):
    if the state is a terminal state: return the state’s utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)

def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v

def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
Minimax Example

---

The diagram illustrates a minimax example with Pacman and Ghost.

- Pacman has a decision node with 3, 12, 8, 2, 4, 6, and 14 as possible scores.
- Ghost has a decision node with 2, 14, 5, and 2 as possible scores.

The tree structure shows the possible outcomes for each move, with the goal of maximizing the score for Pacman and minimizing the score for Ghost.
Minimax Properties

Optimal against a perfect player. Otherwise?
Video of Demo Min vs. Exp (Min)

Slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley
Video of Demo Min vs. Exp (Exp)
Another Demo

Diagram:

- A
  - B
    - E
      - M
    - F
      - 10
  - C
    - G
      - 3
    - H
      - 6
    - I
      - 9
  - D
    - J
      - 10
    - K
      - 1
    - L
      - 2
Minimax Summary

- Rank final game states by their final scores (for tic-tac-toe or chess: win, draw, loss).

- Rank intermediate game states by whose turn it is and the available moves.
  - If it's X's turn, set the rank to that of the maximum move available. If a move will result in a win, X should take it.
  - If it's O's turn, set the rank to that of the minimum move available. If a move will result in a loss, X should avoid it.
Efficiency
Minimax Efficiency

- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: $O(bm)$

- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?
Game Tree Pruning
Pruning

- **Key idea**: give up on paths when you realize that they are worse than options you’ve already explore.
- Track the maximum score that the minimizing player (beta) can get
- Track the minimum score that the maximizing player (alpha) can get
- Whenever the maximum score that beta can get becomes less than the minimum score that alpha can get, the maximizing player can stop searching down this path, because it will never be reached.
Minimax Example

[Diagram of a minimax tree with nodes labeled A, B, C, D, E, F, G containing values 3, 5, 6, 9, 1, 2, 0, -1.]

Alpha-Beta Pruning

- **General configuration (MIN version)**
  - We’re computing the MIN-VALUE at some node $n$
  - We’re looping over $n$’s children
  - $n$’s estimate of the children’s min is dropping
  - Who cares about $n$’s value? MAX
  - Let $a$ be the best value that MAX can get at any choice point along the current path from the root
  - If $n$ becomes worse than $a$, MAX will avoid it, so we can stop considering $n$’s other children (it’s already bad enough that it won’t be played)
- **MAX version is symmetric**
**Alpha-Beta Implementation**

\[\alpha: \text{MAX's best option on path to root}\]
\[\beta: \text{MIN's best option on path to root}\]

---

**def max-value(state, \(\alpha\), \(\beta\)):**

1. Initialize \(v = -\infty\)
2. For each successor of state:
   - \(v = \max(v, \text{value}(\text{successor}, \alpha, \beta))\)
   - If \(v \geq \beta\) return \(v\)
3. If \(v \leq \alpha\) return \(v\)
4. \(\alpha = \max(\alpha, v)\)
5. Return \(v\)

**def min-value(state, \(\alpha\), \(\beta\)):**

1. Initialize \(v = +\infty\)
2. For each successor of state:
   - \(v = \min(v, \text{value}(\text{successor}, \alpha, \beta))\)
   - If \(v \leq \alpha\) return \(v\)
3. \(\beta = \min(\beta, v)\)
4. Return \(v\)
This pruning has **no effect** on minimax value computed for the root!

Values of intermediate nodes might be wrong
- Important: children of the root may have the wrong value
- So the most naïve version won’t let you do action selection

Good child ordering improves effectiveness of pruning

With “perfect ordering”:
- Time complexity drops to $O(b^{m/2})$
- Doubles solvable depth!
- Full search of, e.g. chess, is still hopeless...

This is a simple example of **metareasoning** (computing about what to compute)
Alpha-Beta Quiz

\[
\max_v v(a) \leq 10
\]

\[
v(h) = 2
\]
Resource Limits

- **Problem:** In realistic games, cannot search to leaves!

- **Solution:** Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Replace terminal utilities with an evaluation function for non-terminal positions

- **Example:**
  - Suppose we have 100 seconds, can explore 10K nodes/sec
  - So can check 1M nodes per move
  - α-β reaches about depth 8 - decent chess program

- Guarantee of optimal play is gone
- More plies makes a BIG difference
- Use iterative deepening for an anytime algorithm
Video of Demo Thrashing (d=2)
Why Pacman Starves

- **A danger of replanning agents!**
  - He knows his score will go up by eating the dot now (west, east)
  - He knows his score will go up just as much by eating the dot later (east, west)
  - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
  - Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!
Why Pacman Starves

- A danger of replanning agents!
  - He knows his score will go up by eating the dot now (west, east)
  - He knows his score will go up just as much by eating the dot later (east, west)
  - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
  - Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!
Evaluation Functions
Evaluation Functions

- Evaluation functions score non-terminals in depth-limited search

- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

- e.g. $f_1(s) = (\text{num white queens} - \text{num black queens})$, etc.

$$Eval(s) = w_1f_1(s) + w_2f_2(s) + \ldots + w_nf_n(s)$$
Evaluation for Pacman
Video of Demo Smart Ghosts (Coordination)

Slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley
Video of Demo Smart Ghosts (Coordination) - Zoomed In
Depth Matters

- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- An important example of the tradeoff between complexity of features and complexity of computation
Video of Demo Limited Depth (2)
Video of Demo Limited Depth (10)
Synergies between Evaluation Function and Alpha-Beta?

- **Alpha-Beta:** amount of pruning depends on expansion ordering
  - Evaluation function can provide guidance to expand most promising nodes first (which later makes it more likely there is already a good alternative on the path to the root)
    - (somewhat similar to role of A* heuristic)

- **Alpha-Beta:** (similar for roles of min-max swapped)
  - Value at a min-node will only keep going down
  - Once value of min-node lower than better option for max along path to root, can prune
  - Hence: IF evaluation function provides upper-bound on value at min-node, and upper-bound already lower than better option for max along path to root THEN can prune
Expectimax
Stochastic Transition Model

In our search algorithms so far, the transition model was deterministic and described the outcome of each action in each state.
Stochastic Transition Model

In our search algorithms so far, the transition model was deterministic and described the outcome of each action in each state.

The transition function is sometimes written as $T(s, a, s')$, or explicitly as a probability:

$$p(s' | s, a)$$

Transitions are **Markovian**: the probability of arriving in $s'$ only depends on $s$ and not the history of earlier states.

[Andrey Markov (1856-1922)]
Many reasons that outcomes are unpredictable:
- Explicit randomness: rolling dice
- Unpredictable opponents: the ghosts respond randomly
- Actions can fail: when moving a robot, wheels might slip

[Demo: min vs exp (L7D1,2)]
Video of Demo Minimax vs Expectimax (Min)
Bonus material
Other Game Types
Mixed Layer Types

- E.g. Backgammon
- Expectiminimax
  - Environment is an extra “random agent” player that moves after each min/max agent
  - Each node computes the appropriate combination of its children
Example: Backgammon

- Dice rolls increase $b$: 21 possible rolls with 2 dice
  - Backgammon $\approx$ 20 legal moves
  - Depth 2 = $20 \times (21 \times 20)^3 = 1.2 \times 10^9$

- As depth increases, probability of reaching a given search node shrinks
  - So usefulness of search is diminished
  - So limiting depth is less damaging
  - But pruning is trickier...

- Historic AI: TDGammon uses depth-2 search + very good evaluation function + reinforcement learning:
  world-champion level play

- 1st AI world champion in any game!
Multi-Agent Utilities

- What if the game is not zero-sum, or has multiple players?

- Generalization of minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and competition dynamically...

Slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley
Maximum Expected Utility

- Why should we average utilities? Why not minimax?

- Principle of maximum expected utility:
  - A rational agent should choose the action that maximizes its expected utility, given its knowledge

- Questions:
  - Where do utilities come from?
  - How do we know such utilities even exist?
  - How do we know that averaging even makes sense?
  - What if our behavior (preferences) can’t be described by utilities?
The Axioms of Rationality

Orderability
\[(A \succ B) \lor (B \succ A) \lor (A \sim B)\]

Transitivity
\[(A \succ B) \land (B \succ C) \Rightarrow (A \succ C)\]

Continuity
\[A \succ B \succ C \Rightarrow \exists p \ [p, A; 1 - p, C] \sim B\]

Substitutability
\[A \sim B \Rightarrow [p, A; 1 - p, C] \sim [p, B; 1 - p, C]\]

Monotonicity
\[A \succ B \Rightarrow (p \geq q \iff [p, A; 1 - p, B] \geq [q, A; 1 - q, B])\]

Theorem: Rational preferences imply behavior describable as maximization of expected utility