CSE 573: Artificial Intelligence Winter 2019

Adversarial Search

Hanna Hajishirzi

Based on slides from Dan Klein, Luke Zettlemoyer

Many slides over the course adapted from either Stuart Russell or Andrew Moore

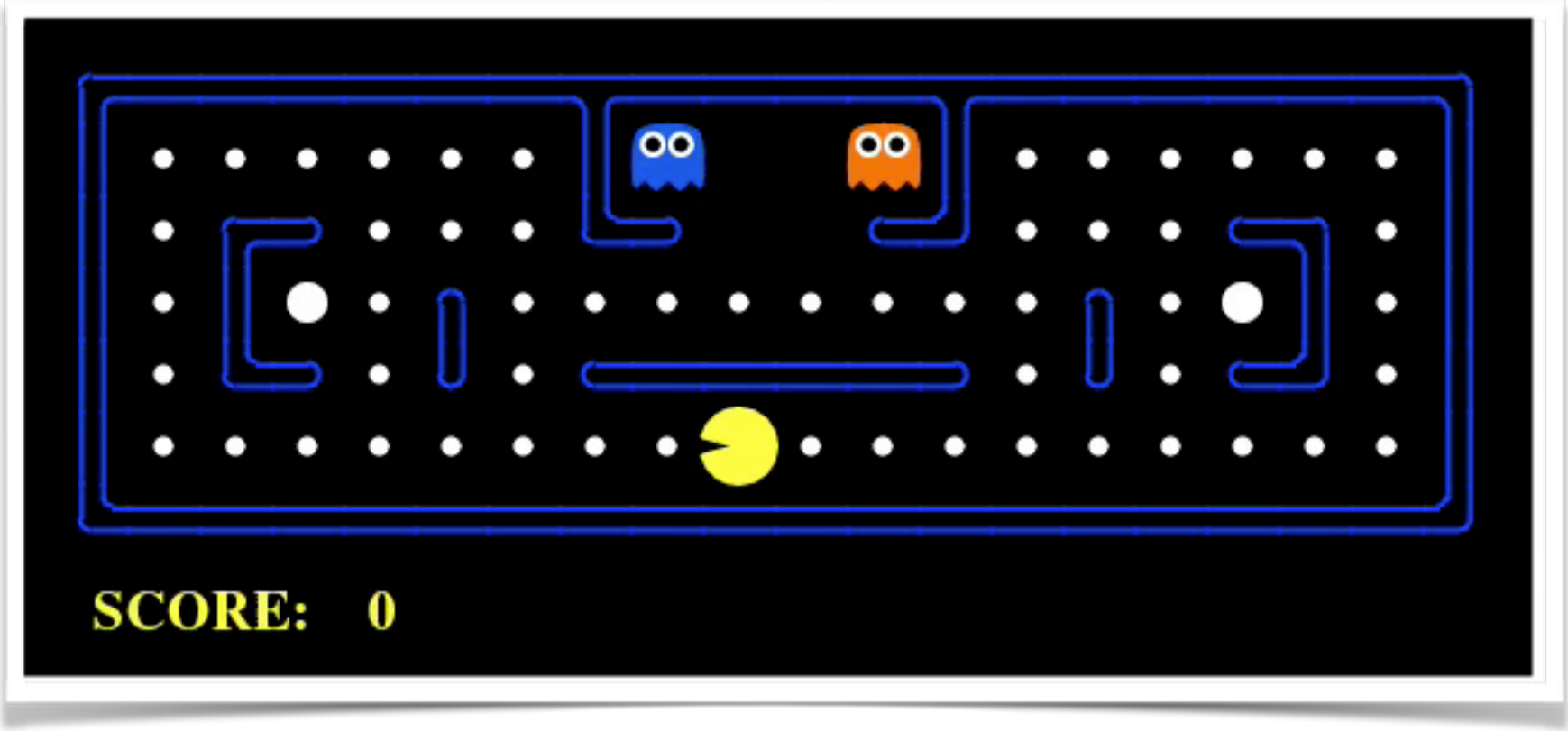
Outline

- Adversarial Search
  - Minimax search
  - α-β search
  - Evaluation functions

Game Playing State-of-the-Art

- **Checkers:** Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions. 2007: Checkers is now solved!
- **Chess:** Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue examined 200 million 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.
- **Othello:** Human champions refuse to compete against computers, which are too good.
- **Go:** DeepMind AlphaGo beat European champions (2 dan) in 2015 and (9 dan) in 2016 using deep reinforcement learning. In go, $b > 300$, previously most programs use pattern knowledge bases to suggest plausible moves, along with aggressive pruning.
- **Pacman:** unknown

Adversarial Search



Game Playing

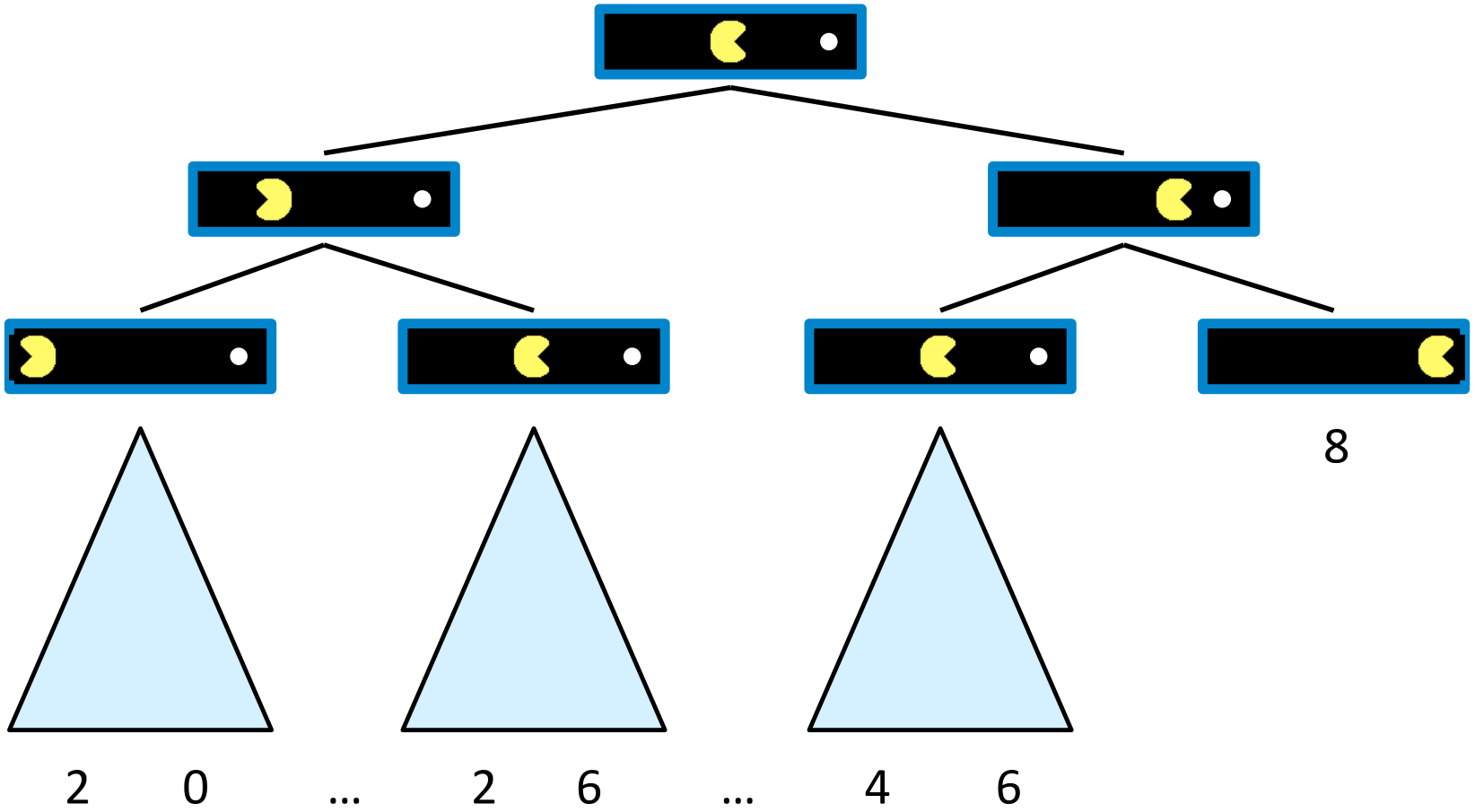
- Many different kinds of games!
- Choices:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Perfect information (can you see the state)?
- Want algorithms for calculating a strategy (policy) which recommends a move in each state

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon, monopoly
imperfect information	stratego	bridge, poker, scrabble, nuclear war

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 x A → S
  - Terminal Test: S → {t,f}
  - Terminal Utilities: S x P → R
- Solution for a player is a policy: S → A

Single-Agent Trees

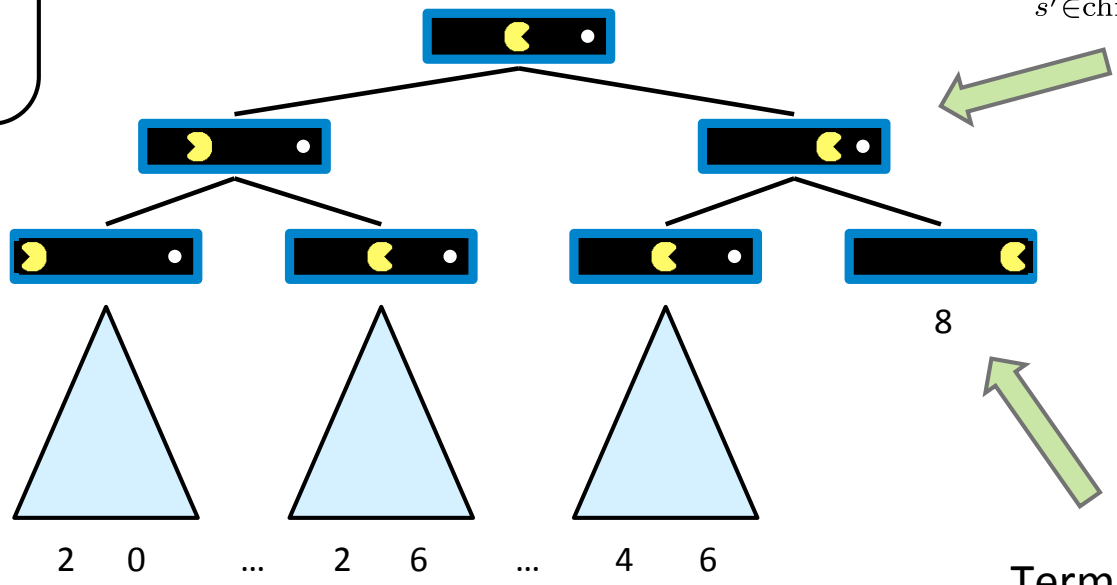


Value of States

Value of a state:
The best achievable outcome (utility) from that state

Non-Terminal States:

$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$

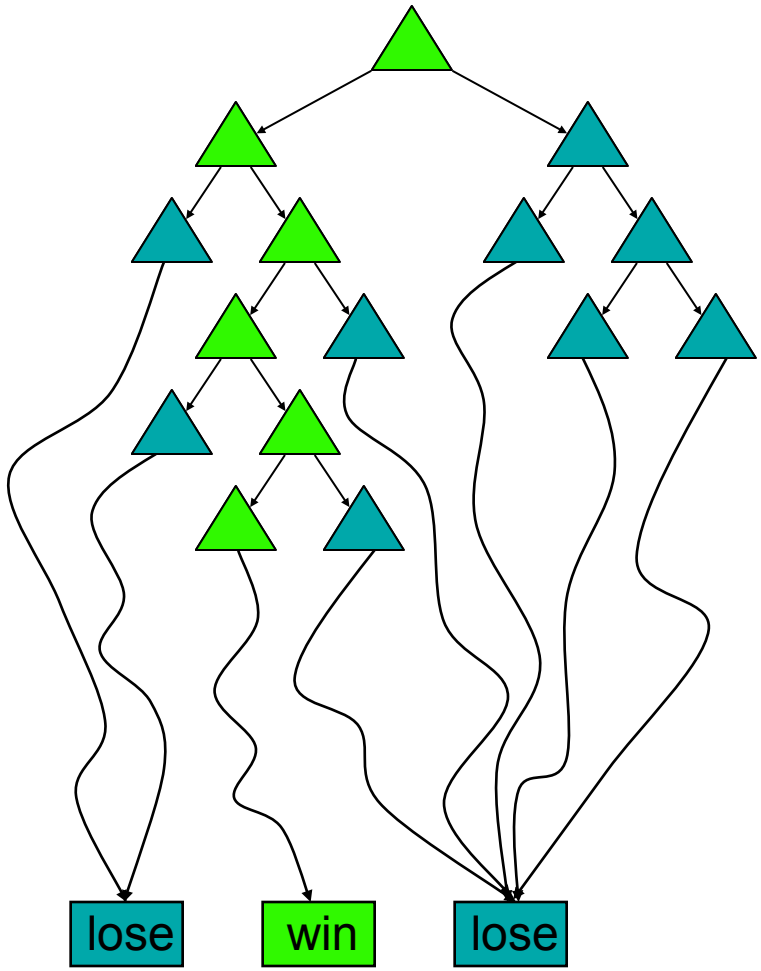


Terminal States:

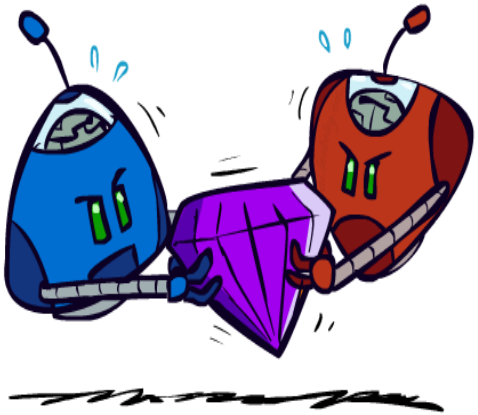
$$V(s) = \text{known}$$

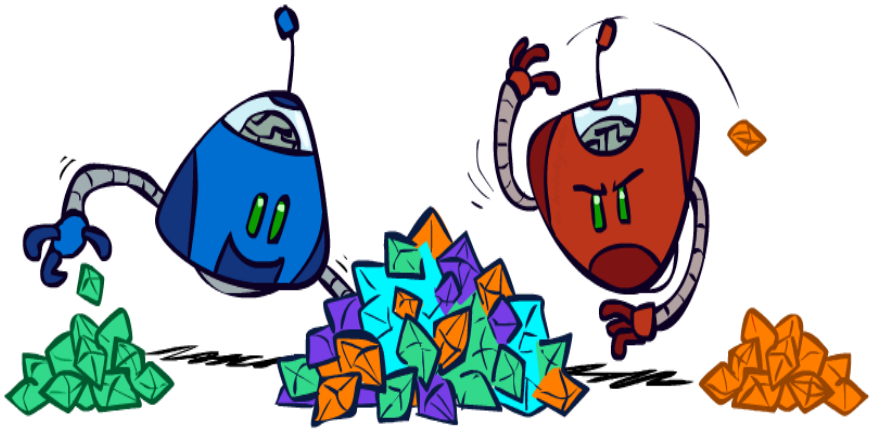
Deterministic Single-Player

- Deterministic, single player, perfect information:
  - Know the rules, action effects, winning states
  - E.g. Freecell, 8-Puzzle, Rubik's cube
- … it's just search!
- Slight reinterpretation:
  - Each node stores a value: the best outcome it can reach
  - This is the maximal outcome of its children (the max value)
  - Note that we don't have path sums as before (utilities at end)
- After search, can pick move that leads to best node



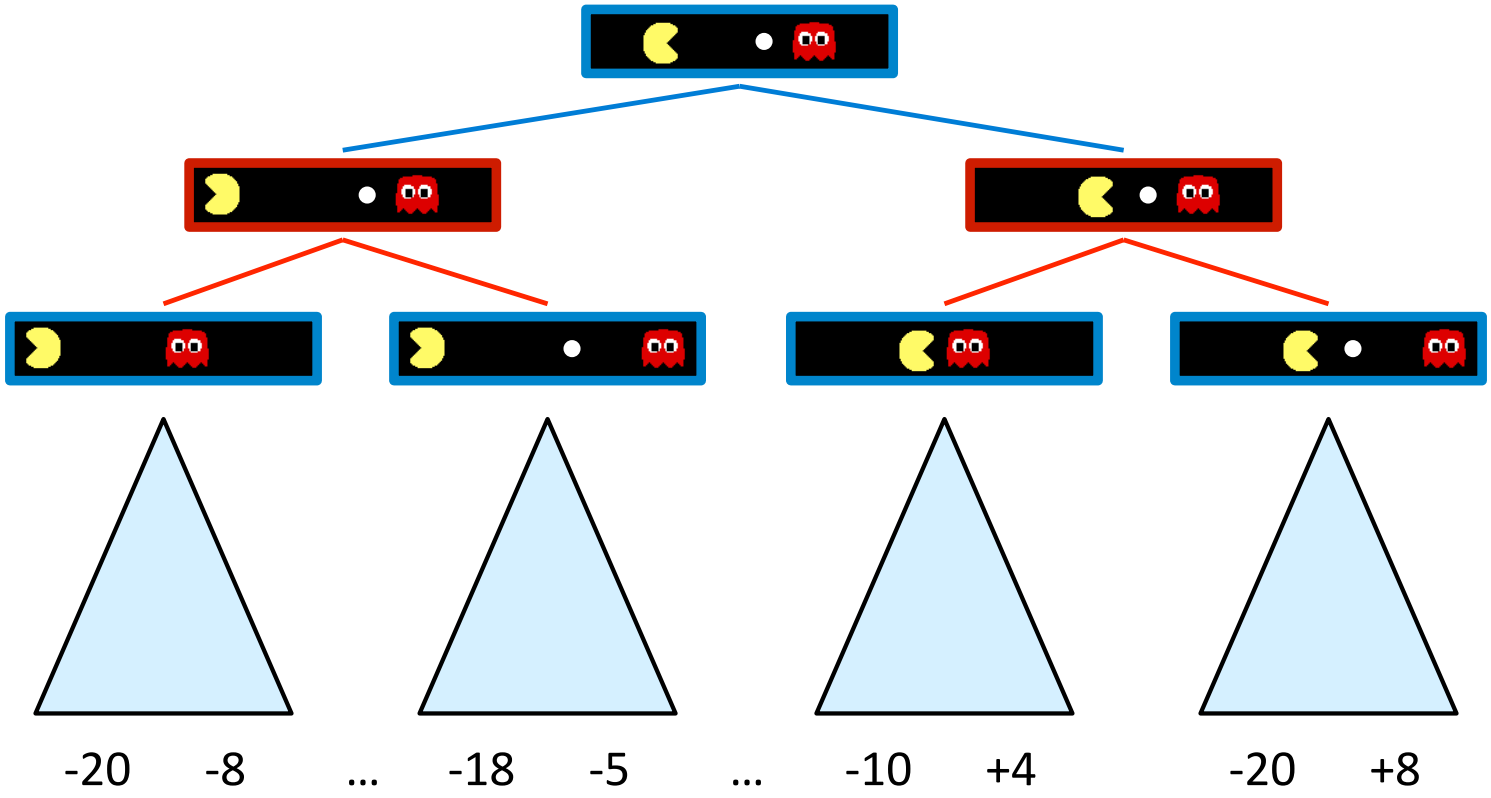
Multi-agent Games





- 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, & more are possible
  - More later on non-zero-sum games

Adversarial Game Trees



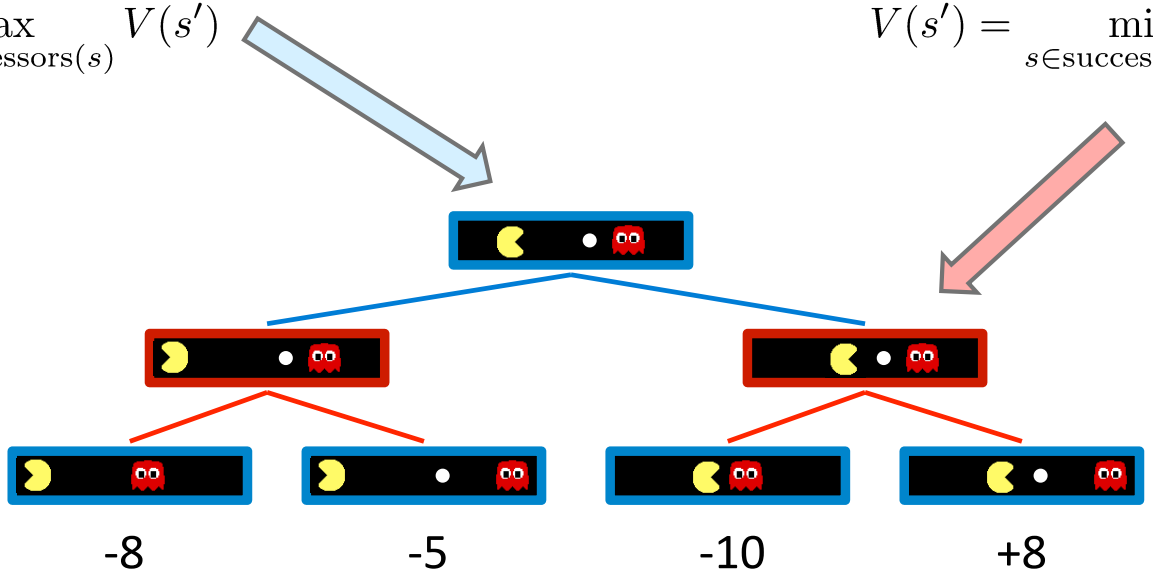
Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

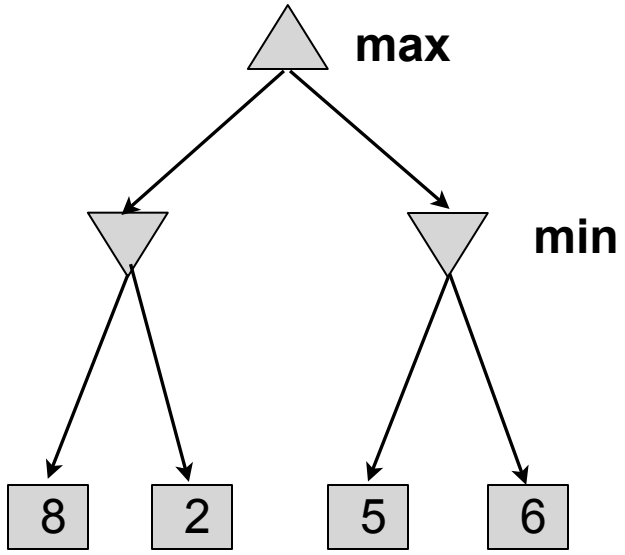


Terminal States:

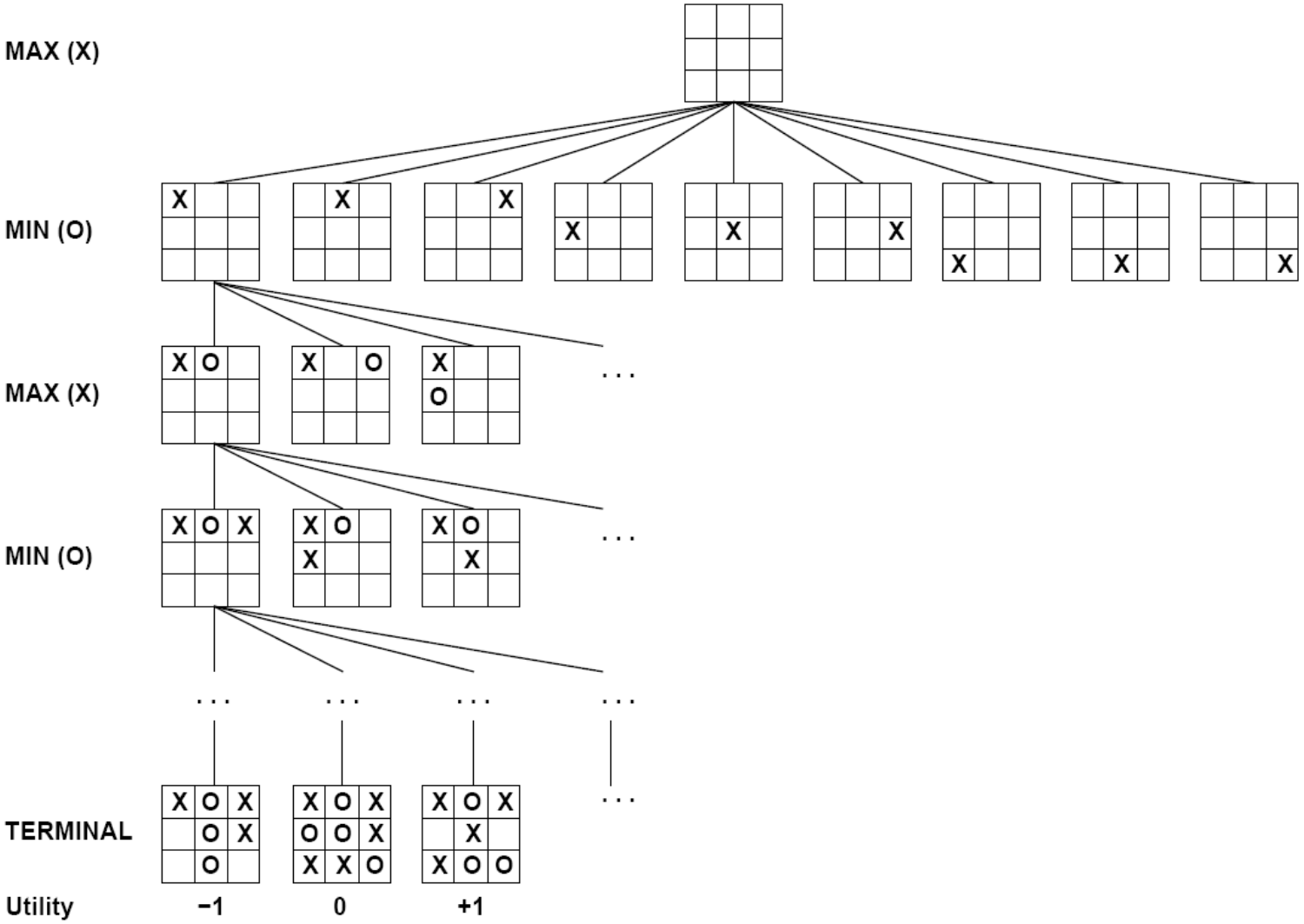
$$V(s) = \text{known}$$

Deterministic Two-Player

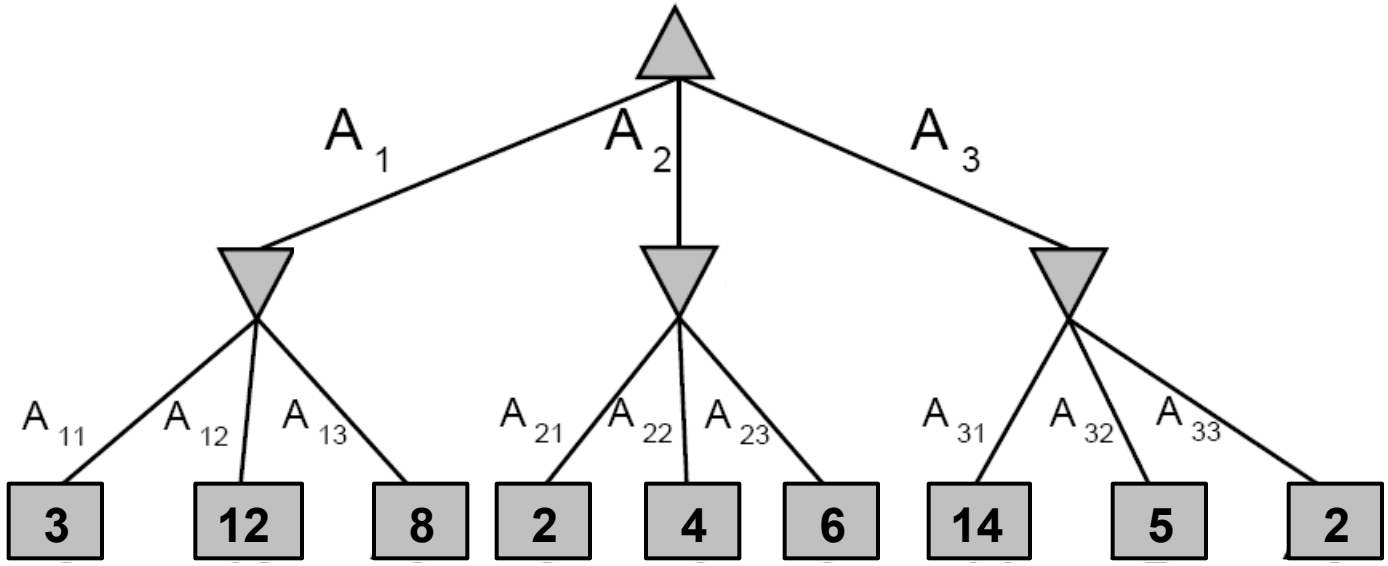
- E.g. tic-tac-toe, chess, checkers
- Zero-sum games
  - Agents have opposite utilities
  - One player maximizes result
  - The other minimizes result
- **Minimax search**
  - A state-space search tree
  - Players alternate
  - Choose move to position with highest minimax value = best achievable utility against best play



Tic-tac-toe Game Tree



Minimax Example



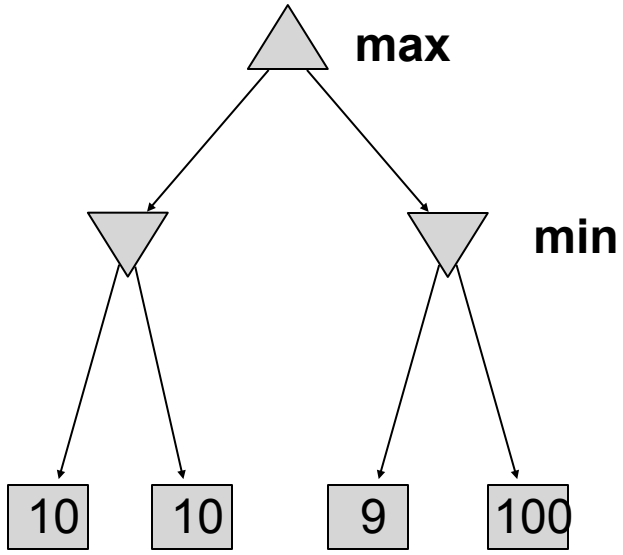
Minimax Search

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
    return v
```

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
    return v
```

Minimax Properties

- Optimal?
  - Yes, against perfect player. Otherwise?
- Time complexity?
  - $O(b^m)$
- Space complexity?
  - O(bm)



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