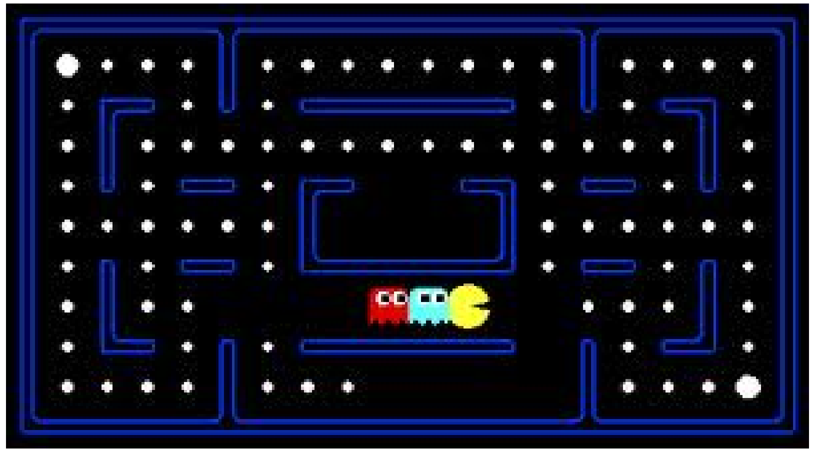
Games: modeling

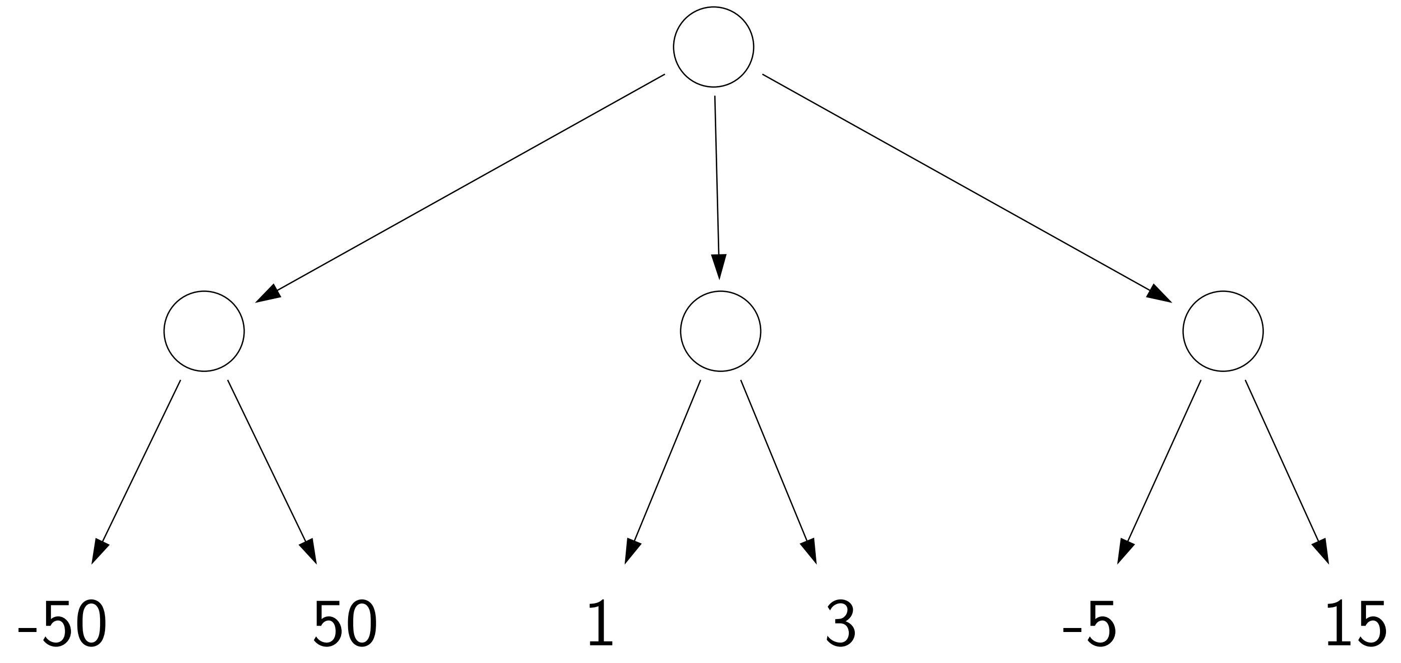


Game tree

Key idea: game tree

Each node is a decision point for a player.

Each root-to-leaf path is a possible outcome of the game.



- Just as in search problems, we will use a tree to describe the possibilities of the game. This tree is known as a **game tree**.
- Note: We could also think of a game graph to capture the fact that there are multiple ways to arrive at the same game state. However, all our algorithms will operate on the tree rather than the graph since games generally have enormous state spaces, and we will have to resort to algorithms similar to backtracking search for search problems.

Two-player zero-sum games

Players $= \{\text{agent}, \text{opp}\}$

Definition: two-player zero-sum game

s_{start}: starting state

Actions(s): possible actions from state s

Succ(s, a): resulting state if choose action a in state s

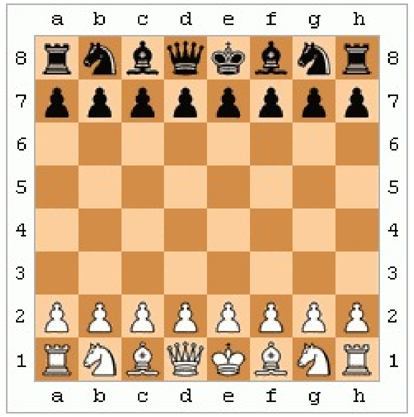
IsEnd(s): whether s is an end state (game over)

Utility(s): agent's utility for end state s

Player$(s) \in$ Players: player who controls state s

- In this lecture, we will specialize to **two-player zero-sum** games, such as chess. To be more precise, we will consider games in which people take turns (unlike rock-paper-scissors) and where the state of the game is fully-observed (unlike poker, where you don't know the other players' hands). By default, we will use the term **game** to refer to this restricted form.
- We will assume the two players are named agent (this is your program) and opp (the opponent). Zero-sum means that the utility of the agent is negative the utility of the opponent (equivalently, the sum of the two utilities is zero).
- Following our approach to search problems and MDPs, we start by formalizing a game. Since games are a type of state-based model, much of the skeleton is the same: we have a start state, actions from each state, a deterministic successor state for each state-action pair, and a test on whether a state is at the end.
- The main difference is that each state has a designated Player(s), which specifies whose turn it is. A player p only gets to choose the action for the states s such that Player$(s) = p$.
- Another difference is that instead of having edge costs in search problems or rewards in MDPs, we will instead have a utility function Utility(s) defined only at the end states. We could have used edge costs and rewards for games (in fact, that's strictly more general), but having all the utility at the end states emphasizes the all-or-nothing aspect of most games. You don't get utility for capturing pieces in chess; you only get utility if you win the game. This ultra-delayed utility makes games hard.

Example: chess



Players $= \{\text{white}, \text{black}\}$

State s: (position of all pieces, whose turn it is)

Actions(s): legal chess moves that Player(s) can make

IsEnd(s): whether s is checkmate or draw

Utility(s): $+\infty$ if white wins, 0 if draw, $-\infty$ if black wins

- Chess is a canonical example of a two-player zero-sum game. In chess, the state must represent the position of all pieces, and importantly, whose turn it is (white or black).
- Here, we are assuming that white is the agent and black is the opponent. White moves first and is trying to maximize the utility, whereas black is trying to minimize the utility.
- In most games that we'll consider, the utility is degenerate in that it will be $+\infty$, $-\infty$, or 0.

Characteristics of games

- All the utility is at the end state

- Different players in control at different states

- There are two important characteristics of games which make them hard.
- The first is that the utility is only at the end state. In typical search problems and MDPs that we might encounter, there are costs and rewards associated with each edge. These intermediate quantities make the problem easier to solve. In games, even if there are cues that indicate how well one is doing (number of pieces, score), technically all that matters is what happens at the end. In chess, it doesn't matter how many pieces you capture, your goal is just to checkmate the opponent's king.
- The second is the recognition that there are other people in the world! In search problems, you (the agent) controlled all actions. In MDPs, we already hinted at the loss of control where nature controlled the chance nodes, but we assumed we knew what distribution nature was using to transition. Now, we have another player that controls certain states, who is probably out to get us.

The halving game

Problem: halving game

Start with a number N.

Players take turns either decrementing N or replacing it with $\lfloor \frac{N}{2} \rfloor$.

The player that is left with 0 wins.

[semi-live solution: `HalvingGame`]

Policies

Deterministic policies: $\pi_p(s) \in \text{Actions}(s)$

action that player p takes in state s

Stochastic policies $\pi_p(s, a) \in [0, 1]$:

probability of player p taking action a in state s

[semi-live solution: `humanPolicy`]

- Following our presentation of MDPs, we revisit the notion of a **policy**. Instead of having a single policy π, we have a policy π_p for each player $p \in \text{Players}$. We require that π_p only be defined when it's p's turn; that is, for states s such that $\text{Player}(s) = p$.
- It will be convenient to allow policies to be stochastic. In this case, we will use $\pi_p(s, a)$ to denote the probability of player p choosing action a in state s.
- We can think of an MDP as a game between the agent and nature. The states of the game are all MDP states s and all chance nodes (s, a). It's the agent's turn on the MDP states s, and the agent acts according to π_{agent}. It's nature's turn on the chance nodes. Here, the actions are successor states s', and nature chooses s' with probability given by the transition probabilities of the MDP: $\pi_{\text{nature}}((s, a), s') = T(s, a, s')$.