NTNU

Adversarial search and games

Ole C. Eidheim

October 10, 2024

Department of Computer Science

Example games

	Deterministic	Chance (external)
Perfect information		
Imperfect information		

Example games

	Deterministic	Chance (external)
Perfect information	Chess, Go	Backgammon, Monopoly
Imperfect information	Battleship, Rock paper scissors	Poker, Scrabble

Outline

Turn-taking, perfect information games

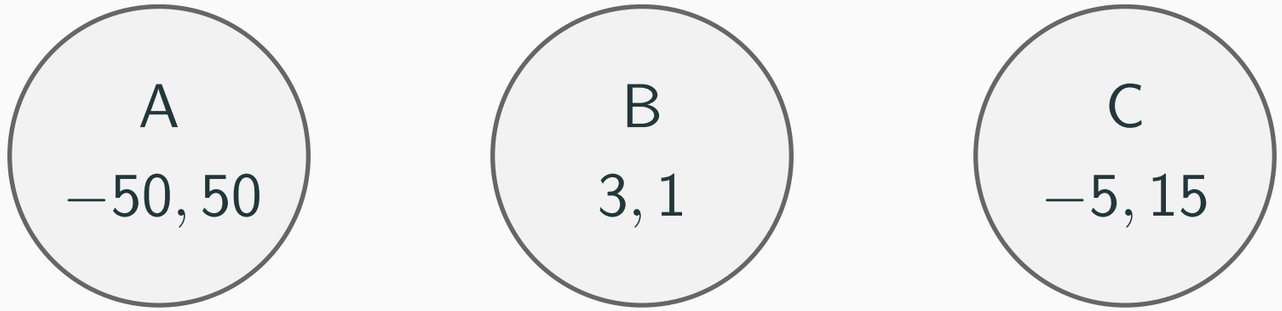
Definitions

- *initialState*: starting state, including which player's turn it is
- *toMove*(s): which player's turn it is at state s
- *actions*(s): legal actions at state s
- *result*(s, a): next state after taking action a at state s
- *isTerminal*(s): is state s an end state?
- *utility*(s, p): the value of the terminal state s for player p

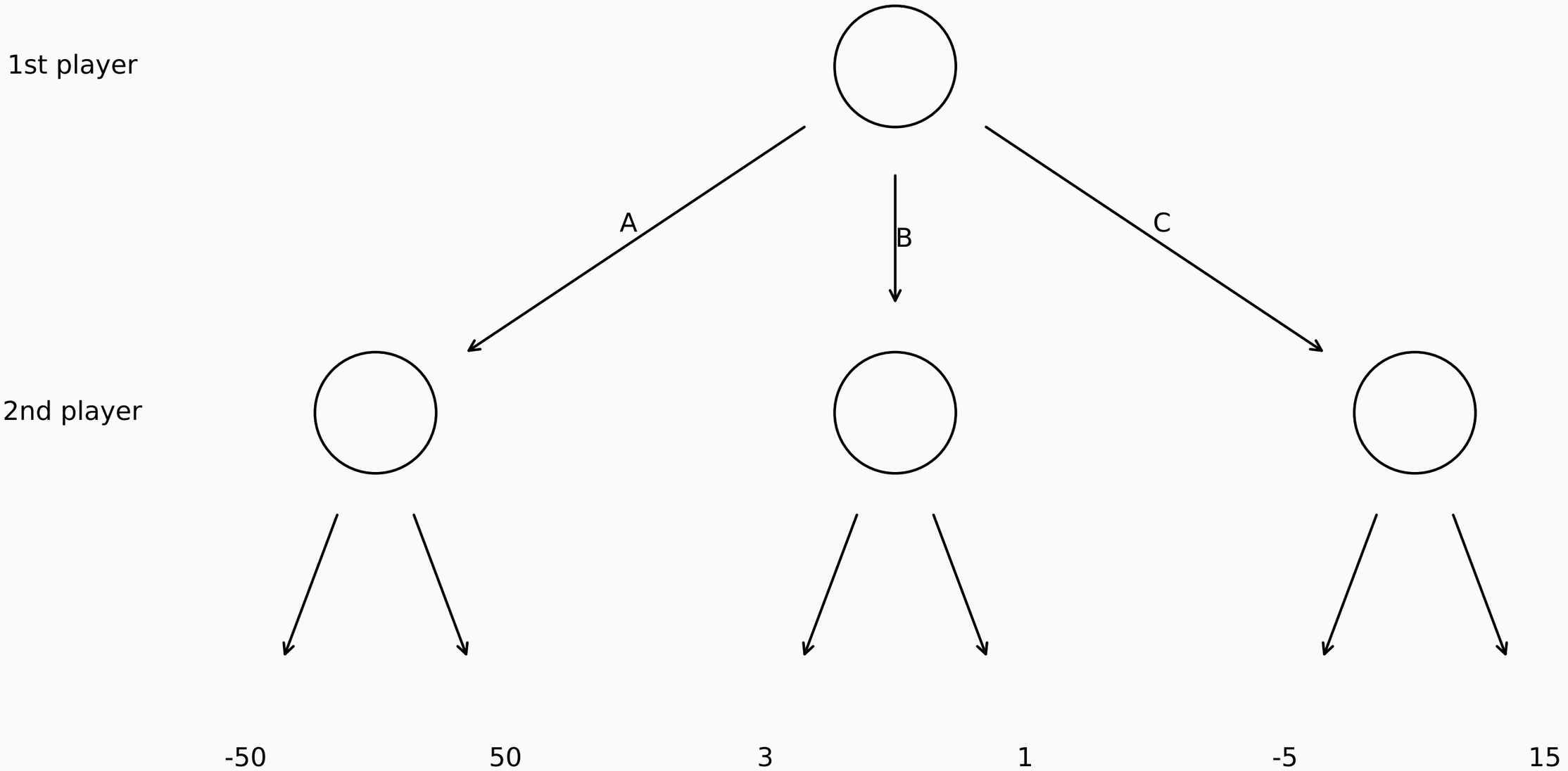
Objective: find player p's actions from *initialState* to a terminal state s that maximize *utility*(s, p)

Example: the bucket game

- The first player chooses bucket A, B or C
- The second player selects a number from the chosen bucket
- The goal of the first player is to maximize the selected number



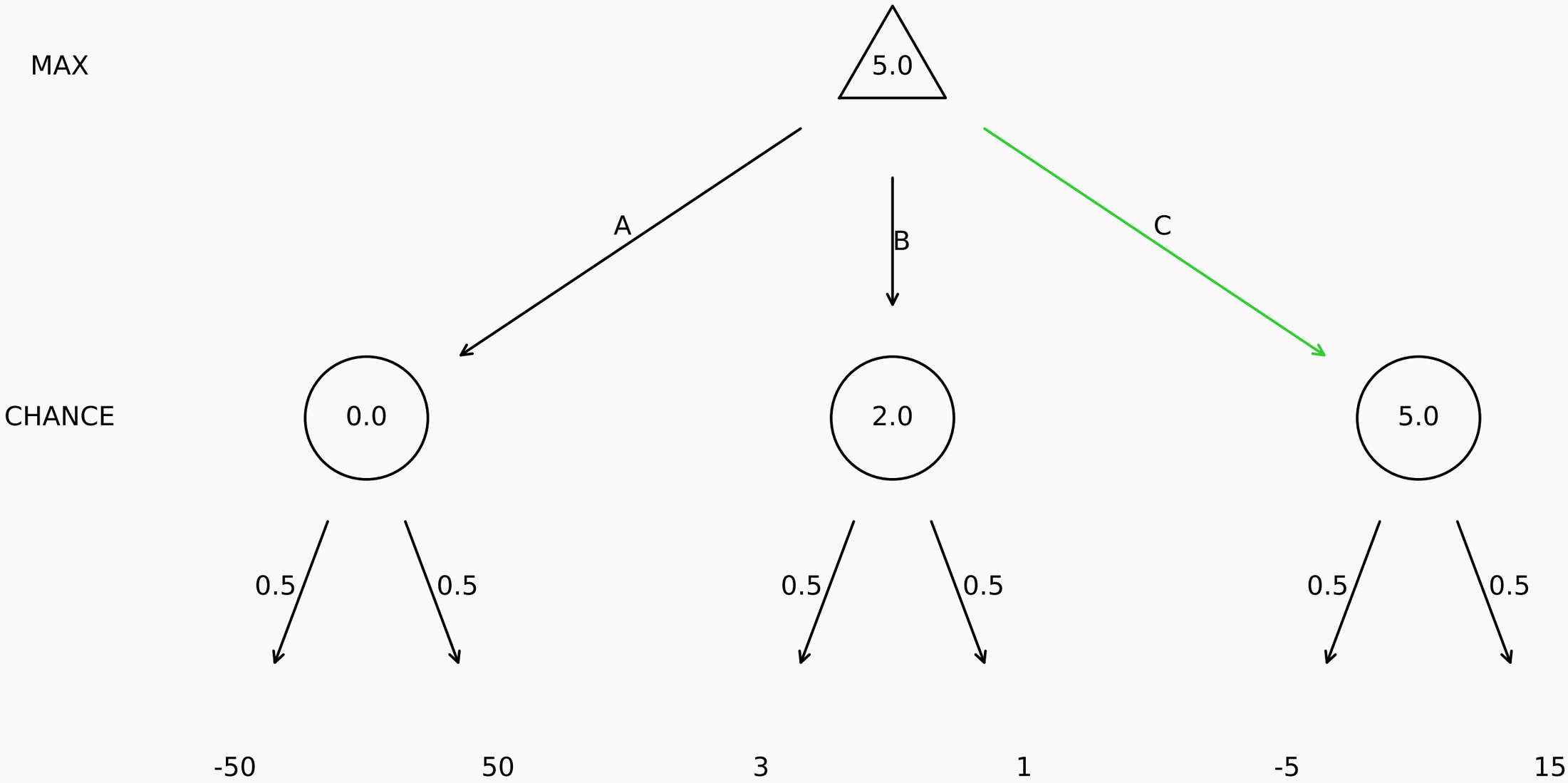
Example: the bucket game, game tree



The expectimax search algorithm

$$expectimax(s) = \begin{cases} utility(s, MAX), & \text{if } isTerminal(s) \\ max_{a \in actions(s)} expectimax(result(s, a)), & \text{if } toMove(s) \text{ is } MAX \\ \sum_r P(r) expectimax(result(s, r)), & \text{if } toMove(s) \text{ is } CHANCE \end{cases}$$

The expectimax search algorithm, bucket game tree

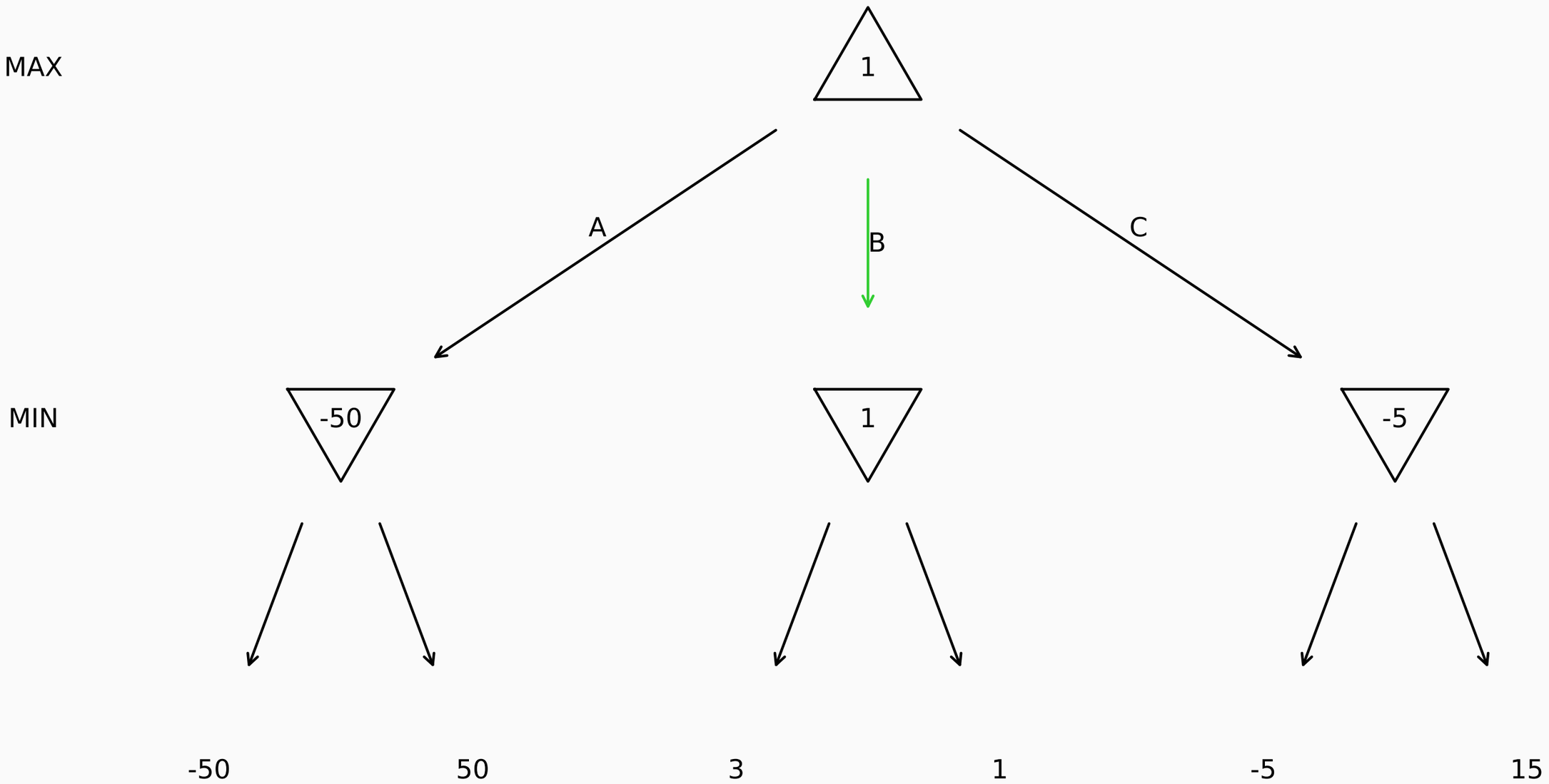


Outline

The minimax search algorithm

$$minimax(s) = \begin{cases} utility(s, MAX), & \text{if } isTerminal(s) \\ max_{a \in actions(s)} minimax(result(s, a)), & \text{if } toMove(s) \text{ is } MAX \\ min_{a \in actions(s)} minimax(result(s, a)), & \text{if } toMove(s) \text{ is } MIN \end{cases}$$

The minimax search algorithm, bucket game tree



Alpha-beta pruning

$$alphaBeta(s, \alpha, \beta) = \begin{cases} utility(s, MAX), & \text{if } isTerminal(s) \\ maxValue(s, \alpha, \beta), & \text{if } toMove(s) \text{ is } MAX \\ minValue(s, \alpha, \beta), & \text{if } toMove(s) \text{ is } MIN \end{cases}$$

α: the current best value of a max-node

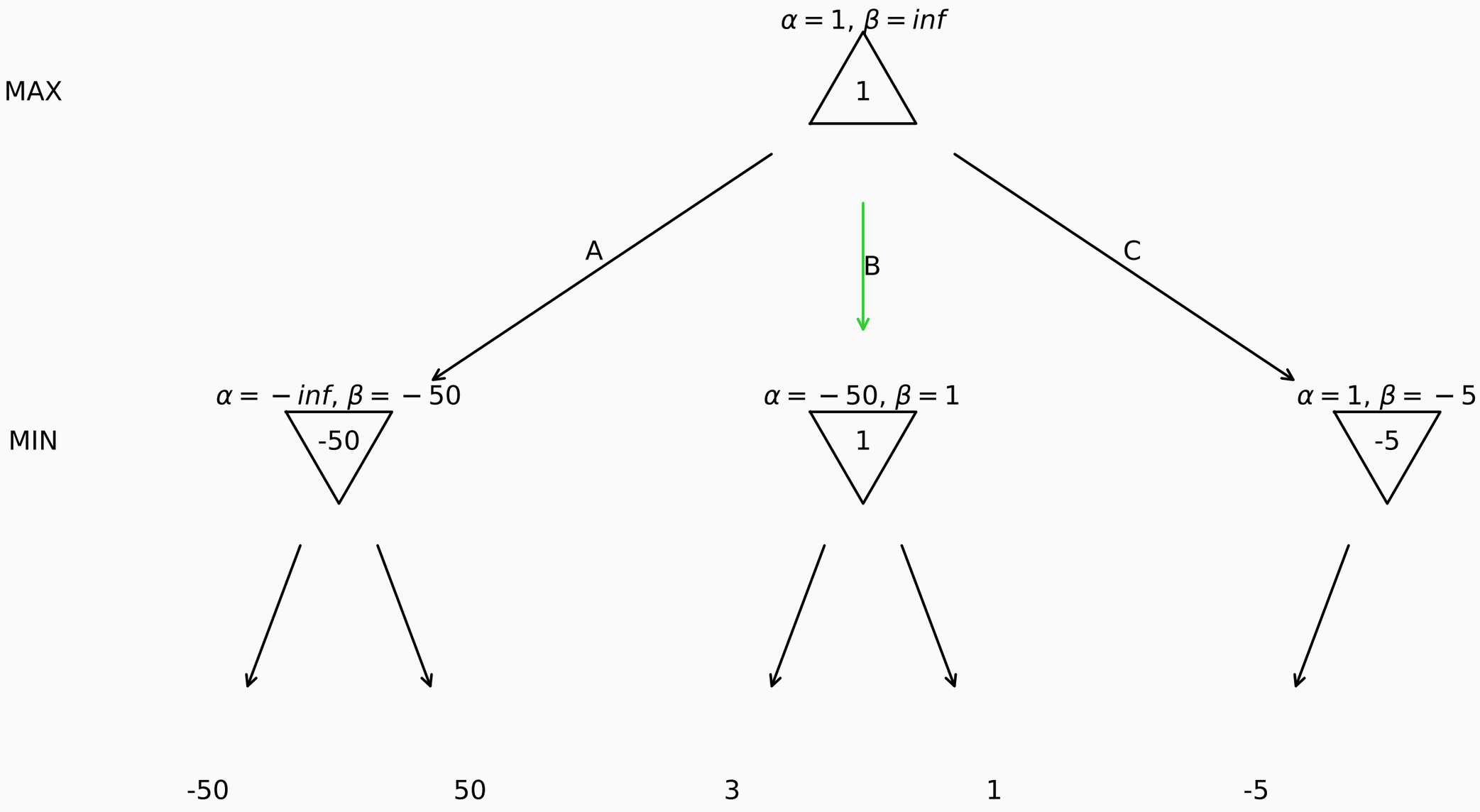
β: the current best value of a min-node

function MAXVALUE(s, α, β)
  $v \leftarrow -\infty$
  for each a **in** $actions(s)$ **do**
    $v \leftarrow max(v, alphaBeta(result(s, a), \alpha, \beta))$
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow max(\alpha, v)$
  end for
  return v
end function

function MINVALUE(s, α, β)
  $v \leftarrow \infty$
  for each a **in** $actions(s)$ **do**
    $v \leftarrow min(v, alphaBeta(result(s, a), \alpha, \beta))$
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow min(\beta, v)$
  end for
  return v
end function

Initially, $alphaBeta(s, -\infty, \infty)$ is called

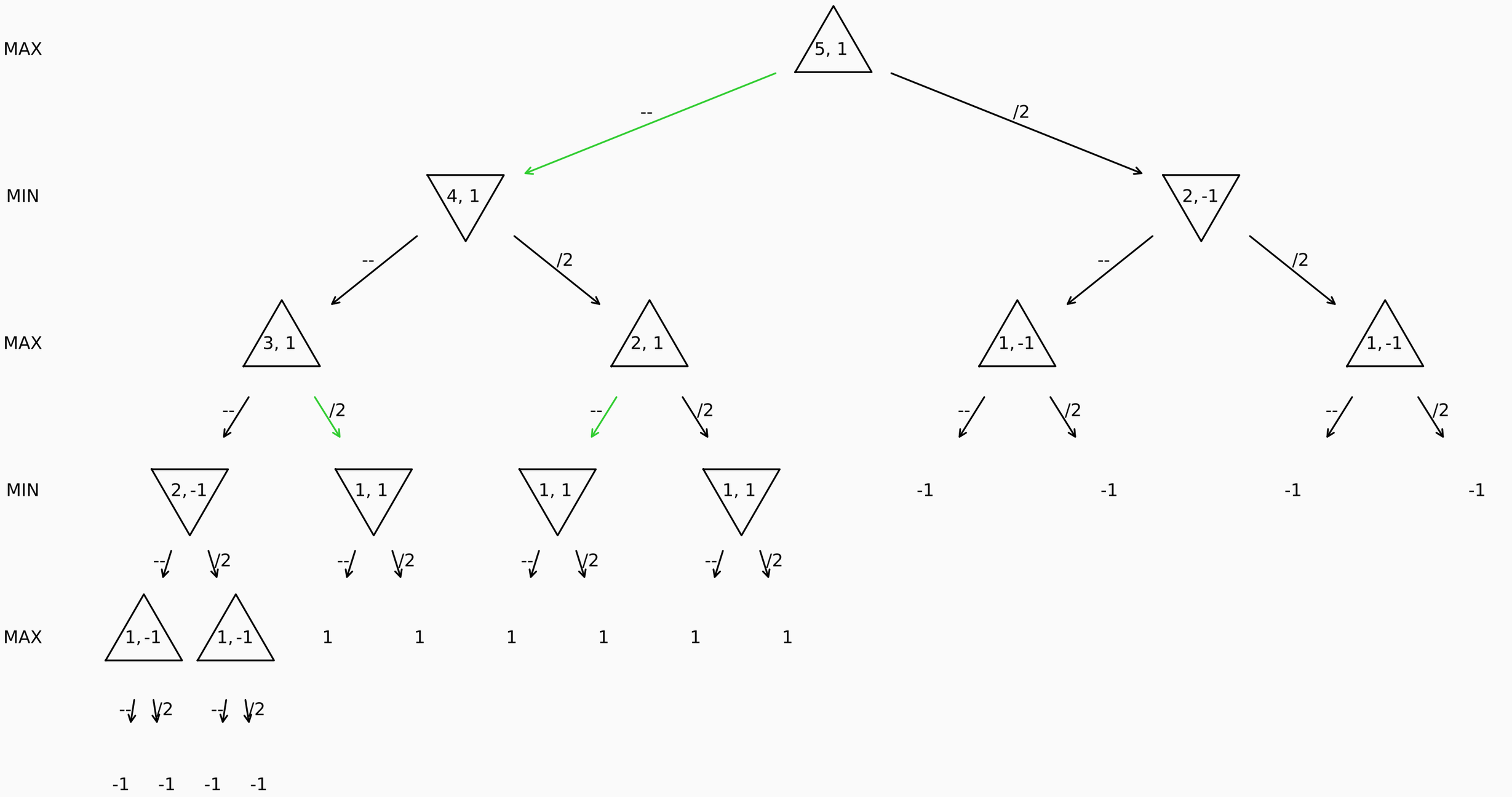
Alpha-beta pruning, bucket game tree



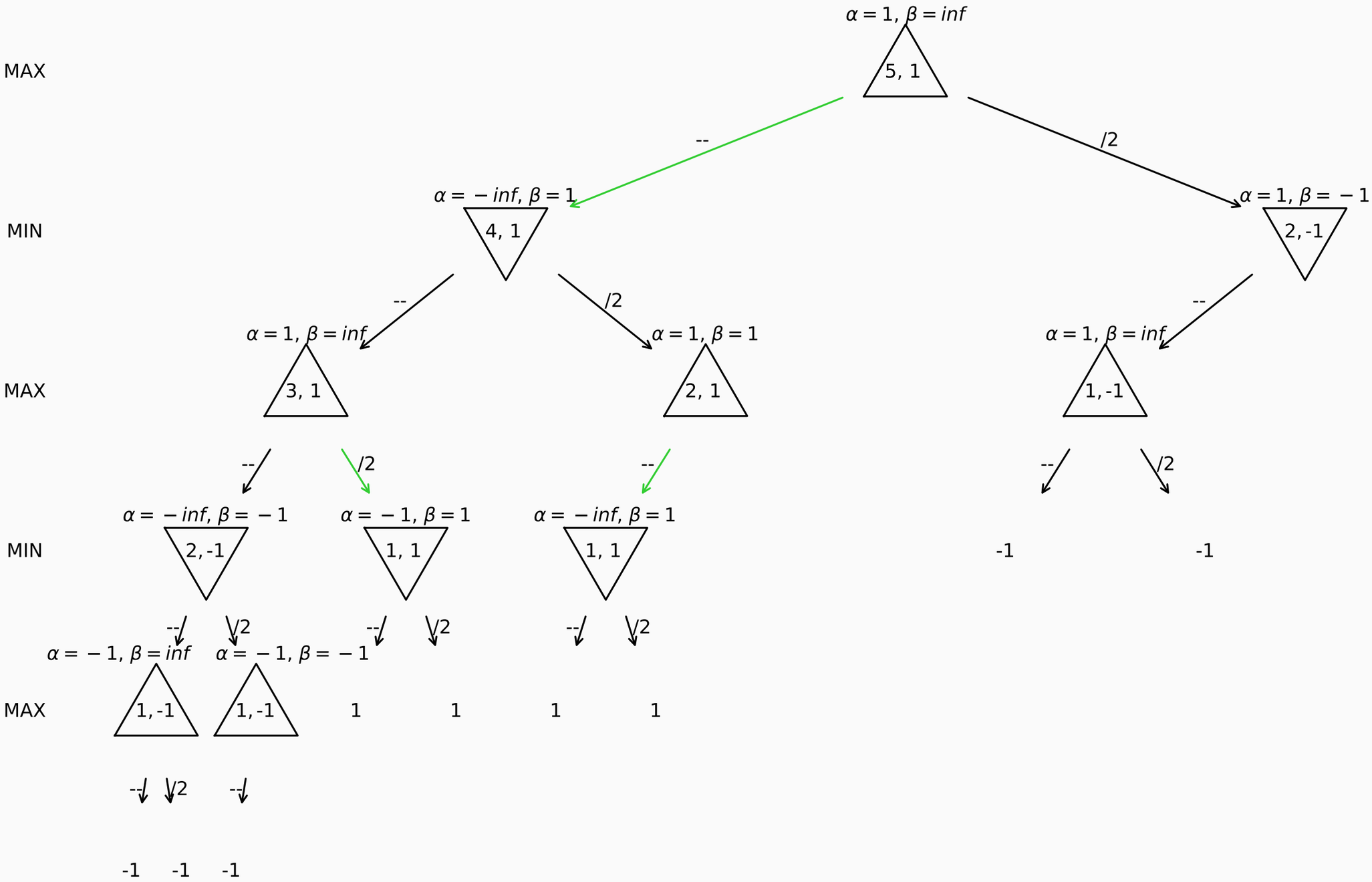
Example: the halving game (a variant of the Nim game)

- Start with a number N
- Players take turns either decrementing N or replacing it with $\frac{N}{2}$ rounded down
- The player that is left with 0 wins

The halving game tree, $N = 5$, without alpha-beta pruning



The halving game tree, $N = 5$, with alpha-beta pruning



Outline

Heuristic minimax (depth limited search)

$$hMinimax(s,d) = \begin{cases} utility(s, MAX), & \text{if } isTerminal(s) \\ eval(s, MAX), & \text{if } isCutoff(s,d) \\ max_{a \in actions(s)} hMinimax(result(s,a), d+1), & \text{if } toMove(s) \text{ is } MAX \\ min_{a \in actions(s)} hMinimax(result(s,a), d+1), & \text{if } toMove(s) \text{ is } MIN \end{cases}$$

- $eval(s, MAX)$ is an estimate of the true value $minimax(s)$
  - Chess example:
    - $eval(s) = w_1 material(s) + w_2 mobility(s) + w_3 kingSafety(s) + w_4 centerControl(s)$
    - $material(s) = 16(Q - Q') + 8(R - R') + 5(B - B' + N - N') + P - P'$
    - $mobility(s) = numLegalMoves - numLegalMoves'$
    - ...

Initially, $hMinimax(s, 0)$ is called

Search versus lookup

Should we perform search on the first chess move in a new game?

Outline

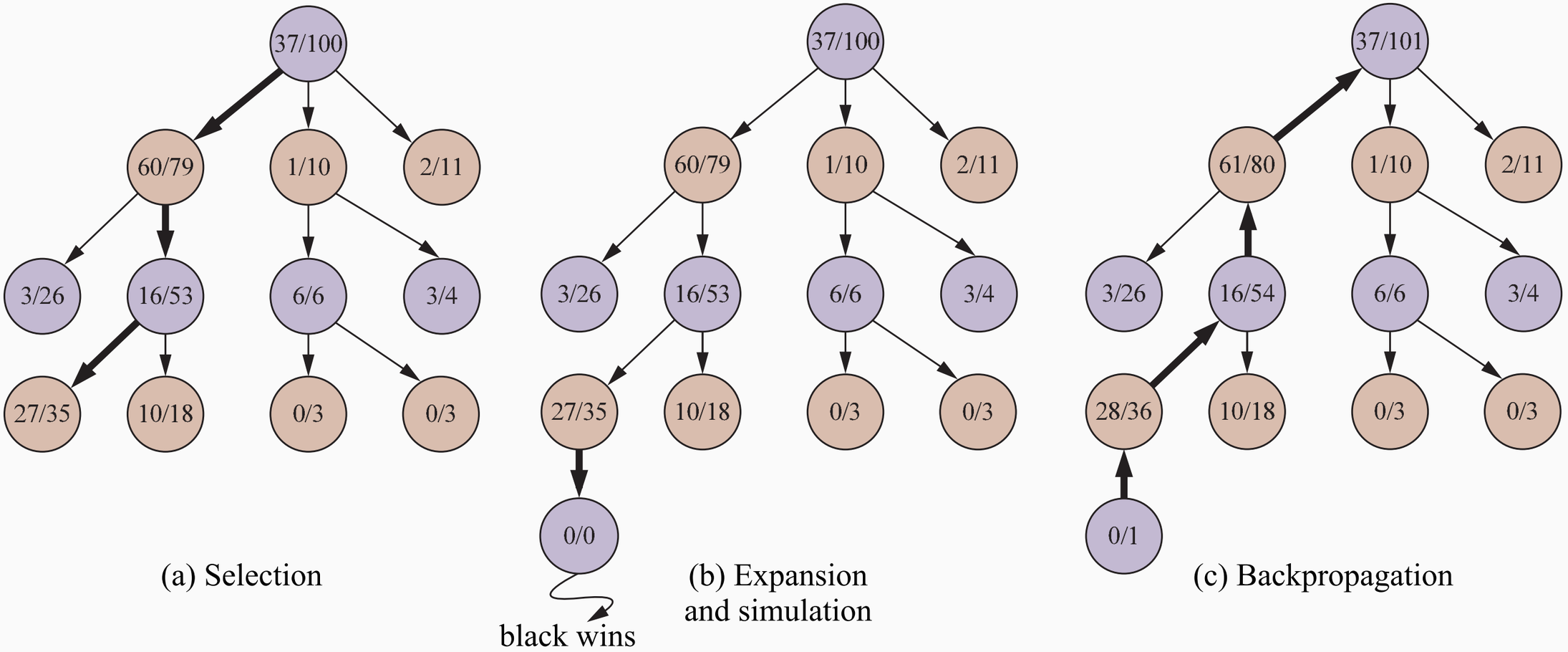
Monte Carlo methods

Monte Carlo methods, or Monte Carlo experiments, are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle.

[. . .] The method is useful for obtaining numerical solutions to problems too complicated to solve analytically.

Wikipedia

Monte Carlo tree search



- It is black's turn (root node is black)
- Black nodes show white's wins and white nodes show black's wins
  - Root node shows that white has won 37 out of 100 simulations (playouts)
- Black first chooses the move that has lead to 60 wins out of 79 playouts

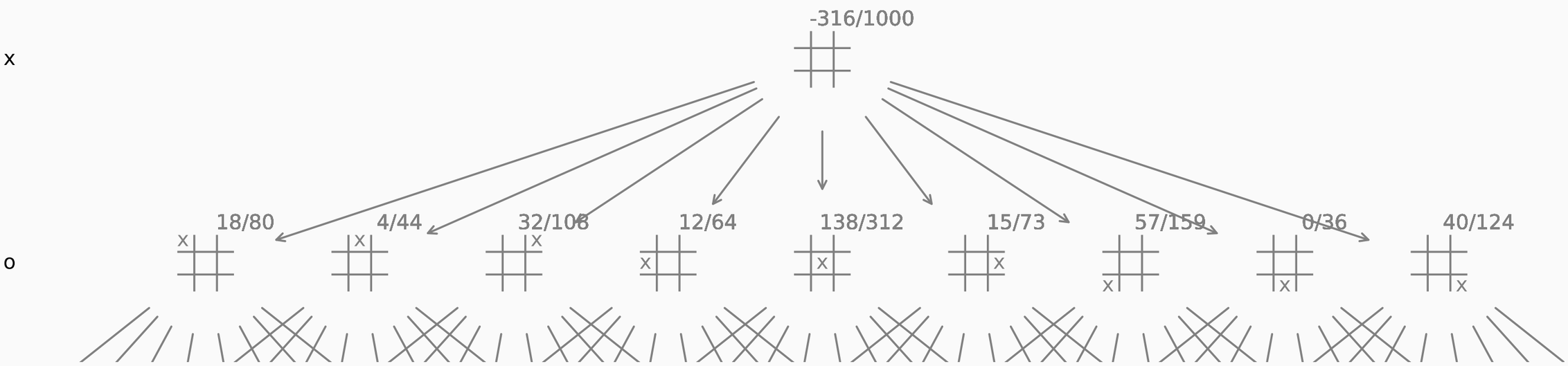
Monte Carl tree search algorithm

function MONTE-CARLO-TREE-SEARCH($state$) **returns** *an action*
  $tree \leftarrow$ NODE($state$)
  while IS-TIME-REMAINING() **do**
    $leaf \leftarrow$ SELECT($tree$)
    $child \leftarrow$ EXPAND($leaf$)
    $result \leftarrow$ SIMULATE($child$)
    BACK-PROPAGATE($result$, $child$)
  return the move in ACTIONS($state$) whose node has highest number of playouts

Monte Carl tree search algorithm

function MONTE-CARLO-TREE-SEARCH(*state*) **returns** *an action*
  tree ← NODE(*state*)
  while IS-TIME-REMAINING() **do**
    leaf ← SELECT(*tree*)
    child ← EXPAND(*leaf*)
    result ← SIMULATE(*child*)
    BACK-PROPAGATE(*result*, *child*)
  return the move in ACTIONS(*state*) whose node has highest number of playouts

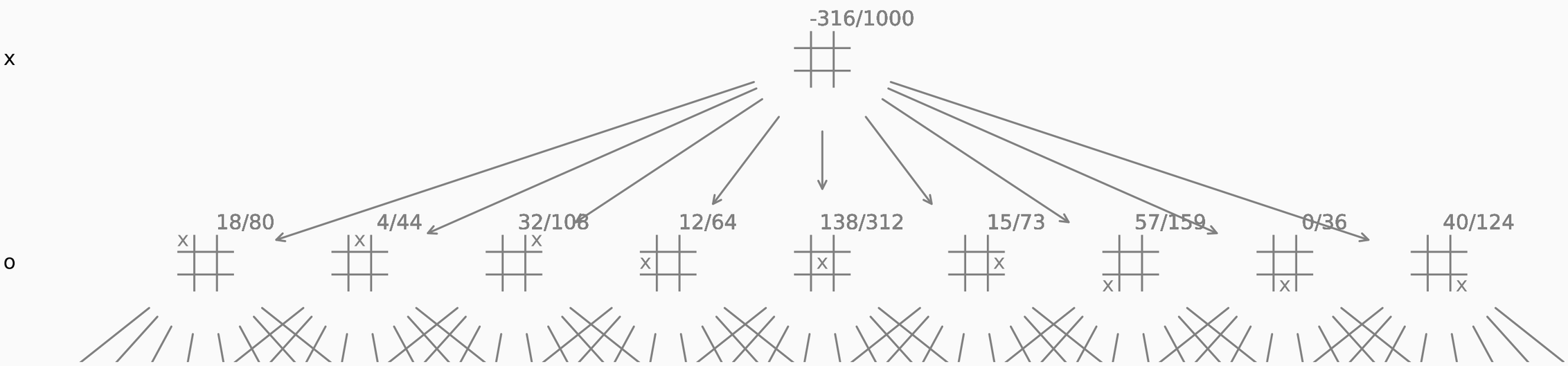
For example, which action would be returned here:



Monte Carl tree search algorithm

function MONTE-CARLO-TREE-SEARCH(*state*) **returns** *an action*
  tree ← NODE(*state*)
  while IS-TIME-REMAINING() **do**
    leaf ← SELECT(*tree*)
    child ← EXPAND(*leaf*)
    result ← SIMULATE(*child*)
    BACK-PROPAGATE(*result*, *child*)
  return the move in ACTIONS(*state*) whose node has highest number of playouts

For example, which action would be returned here:



Answer: place x in the middle square

Selection policy

- Exploration
  - Explore states that have had few playouts
- Exploitation
  - Exploit states that have done well in past playouts,

Selection policy

- Exploration
  - Explore states that have had few playouts
- Exploitation
  - Exploit states that have done well in past playouts,

$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\ln N(parent(n))}{N(n)}}$$

- The node n with highest $UCB1(n)$ is selected
- $U(n)$: total utility of all playouts that went through node n
- $N(n)$: the number of playouts through node n
- $parent(n)$: parent node of n
- C: constant that balances exploration and exploitation
  - High C favors exploration
  - Usually $C = \sqrt{2}$ is used

Playouts

A Tic-tac-toe agent is quite good after 1000 iterations

AlphaZero (2017): Monte Carlo tree search with neural networks

- Modified Monte Carlo tree search:
  - Value network: how likely the current state is a win
    - Used instead of playouts
    - Trained through self-play games (games against itself using modified Monte Carlo tree search)

AlphaZero (2017): Monte Carlo tree search with neural networks

- Modified Monte Carlo tree search:
  - Value network: how likely the current state is a win
    - Used instead of playouts
    - Trained through self-play games (games against itself using modified Monte Carlo tree search)
  - Policy network: action probabilities, where good actions should have high probabilities
    - Used to improve the selection policy
    - Trained through self-play games

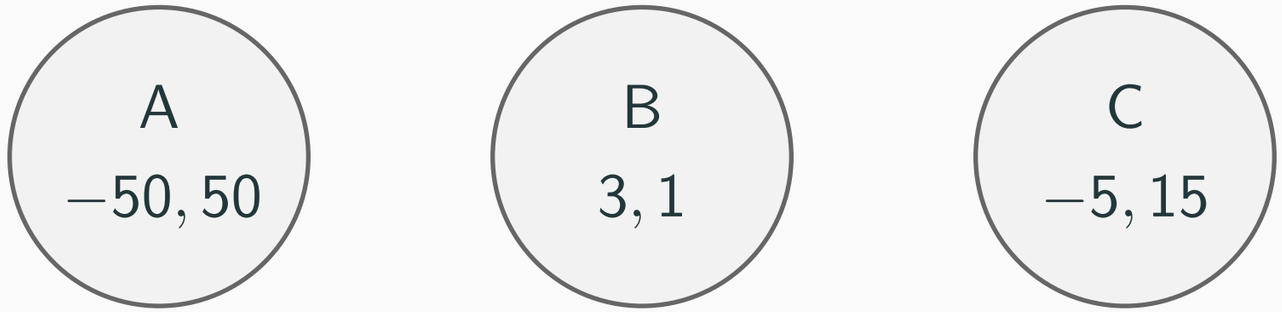
Outline

Stochastic games

Games involving external random elements, such as dice throws

Example: modified bucket game

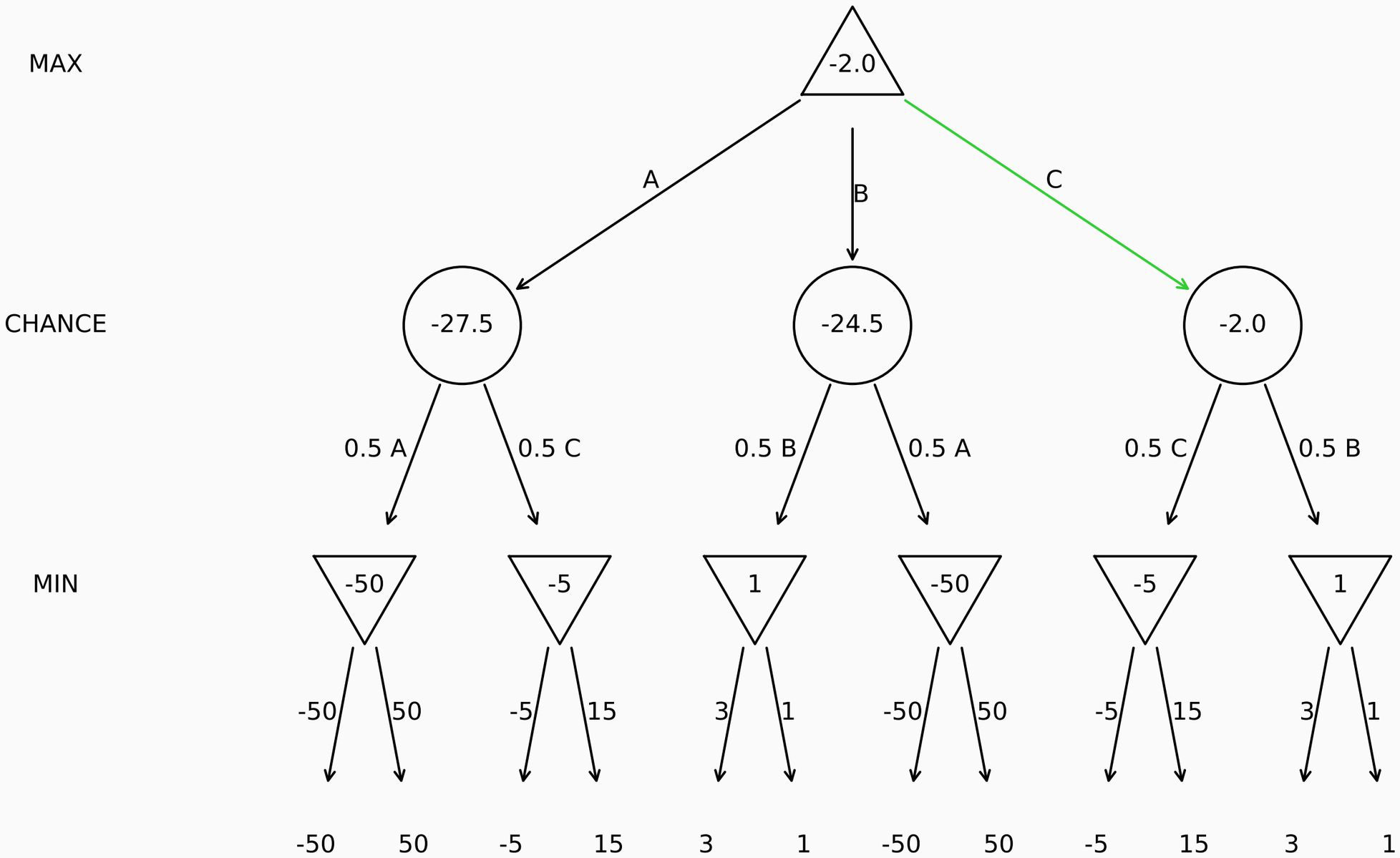
- The first player chooses bucket A, B or C
- **Flip a coin; if heads, move one bucket to the left (with wraparound)**
- The second player selects a number from the chosen bucket
- The goal of the first player is to maximize the selected number



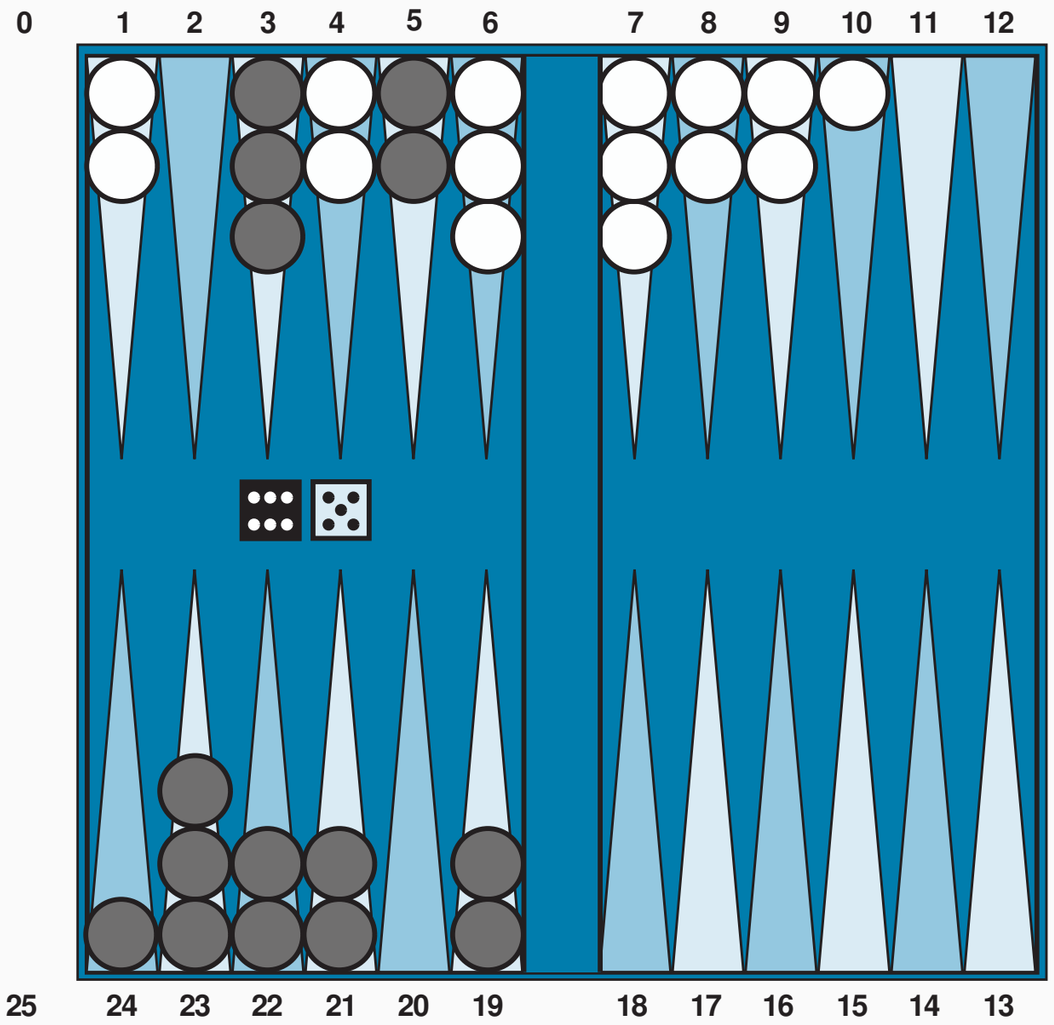
The expectiminimax search algorithm

$$expectiminimax(s) = \begin{cases} utility(s, MAX), & \text{if } isTerminal(s) \\ max_{a \in actions(s)} expectiminimax(result(s, a)), & \text{if } toMove(s) \text{ is } MAX \\ min_{a \in actions(s)} expectiminimax(result(s, a)), & \text{if } toMove(s) \text{ is } MIN \\ \sum_{a \in actions(s)} P(a) expectiminimax(result(s, a)), & \text{if } toMove(s) \text{ is } CHANCE \end{cases}$$

The expectiminimax search algorithm, modified bucket game tree



Example: backgammon



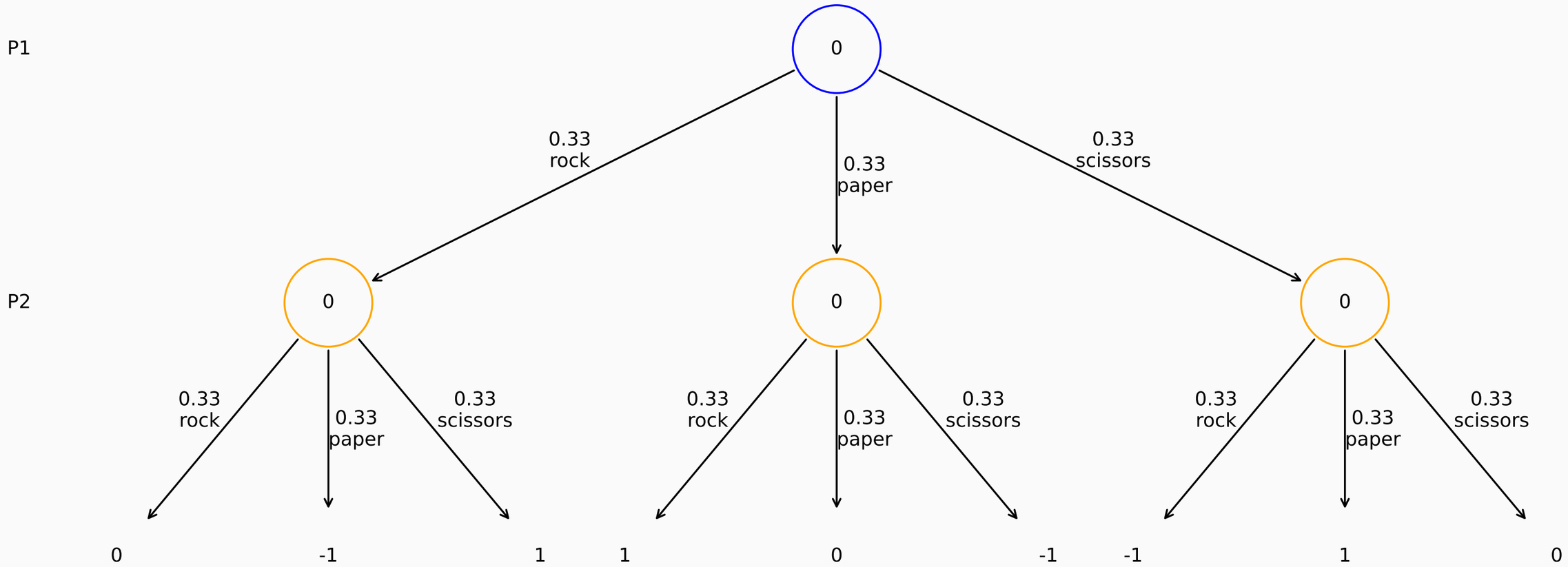
For more advanced stochastic games, Monte Carlo Tree Search can be used where each playout includes random dice rolls

Outline

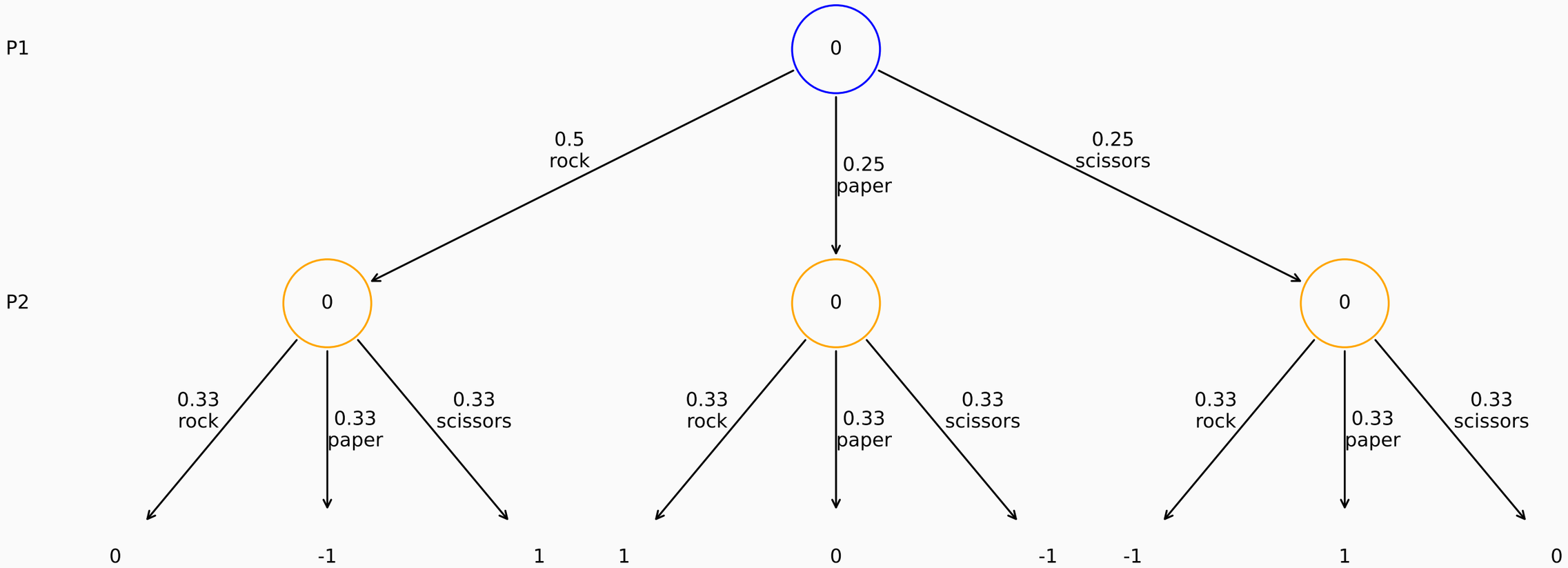
Partially observable games

- Game states not fully observable or players act simultaneously
- Minimax and similar algorithms cannot be used

Example: rock paper scissors, Nash equilibrium (next lecture)

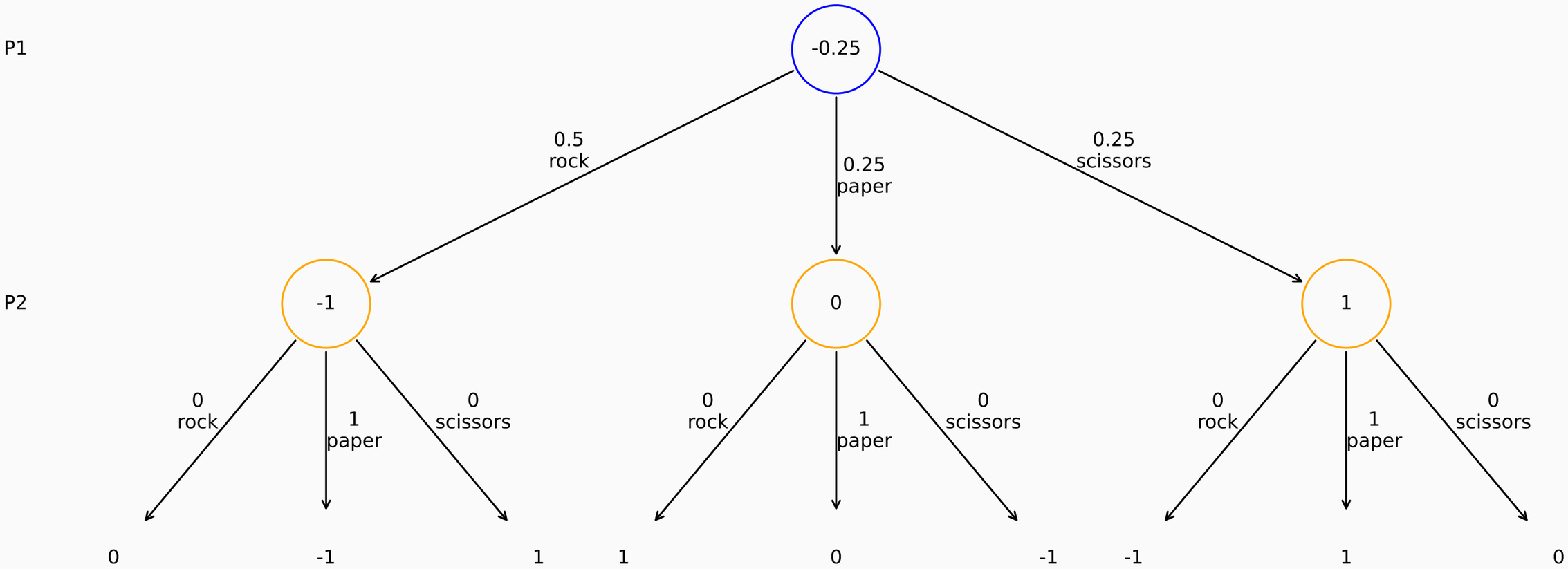


Example: rock paper scissors, P1 plays rock more often



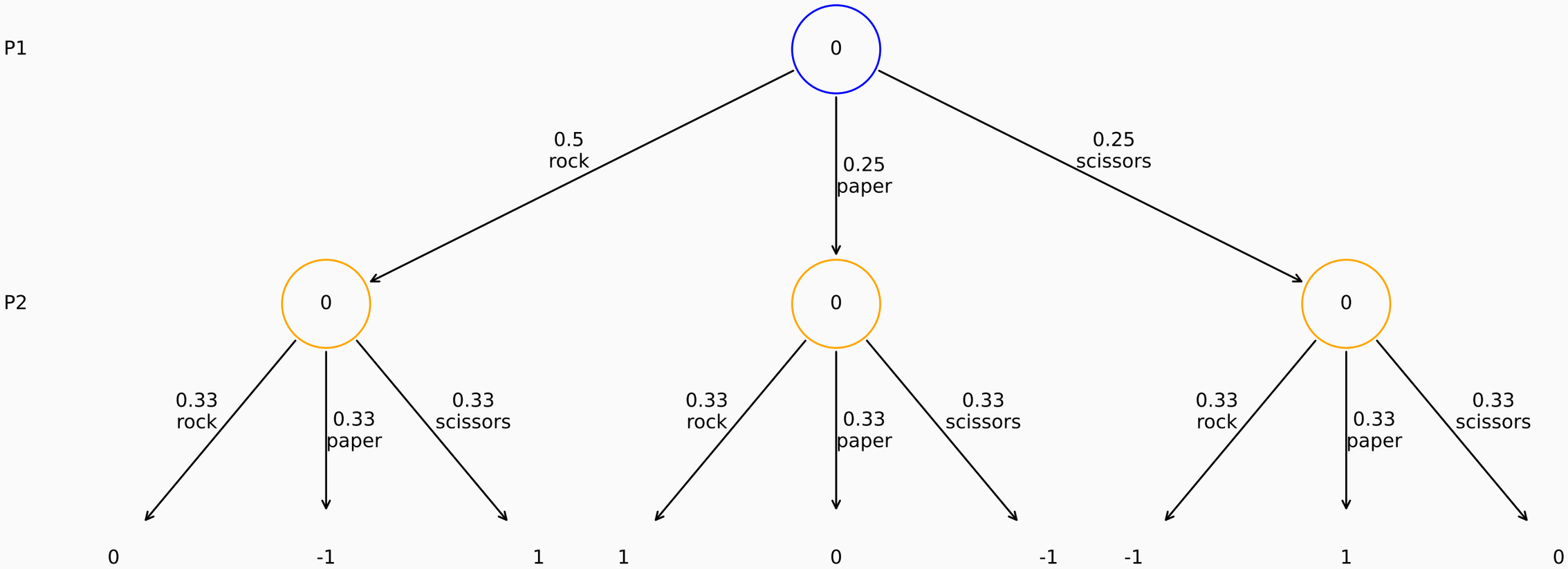
How can P2 exploit this?

Example: rock paper scissors, P1 plays rock more often, P2 exploits this



How can P2 exploit this? By playing paper more often

Regret minimization

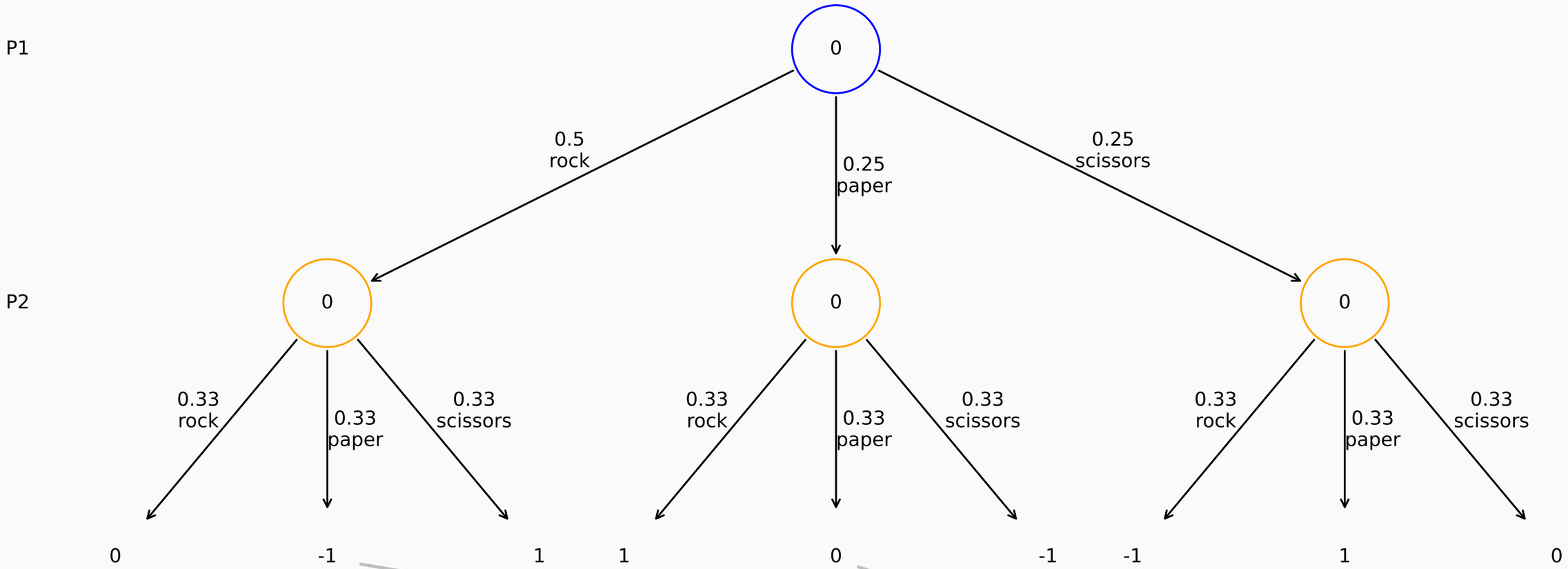


Regret

The utility an action would have received minus the utility actually received

- Regrets for each player's actions are summed over all games played
- Possible algorithm: choose the action with the highest summed regret

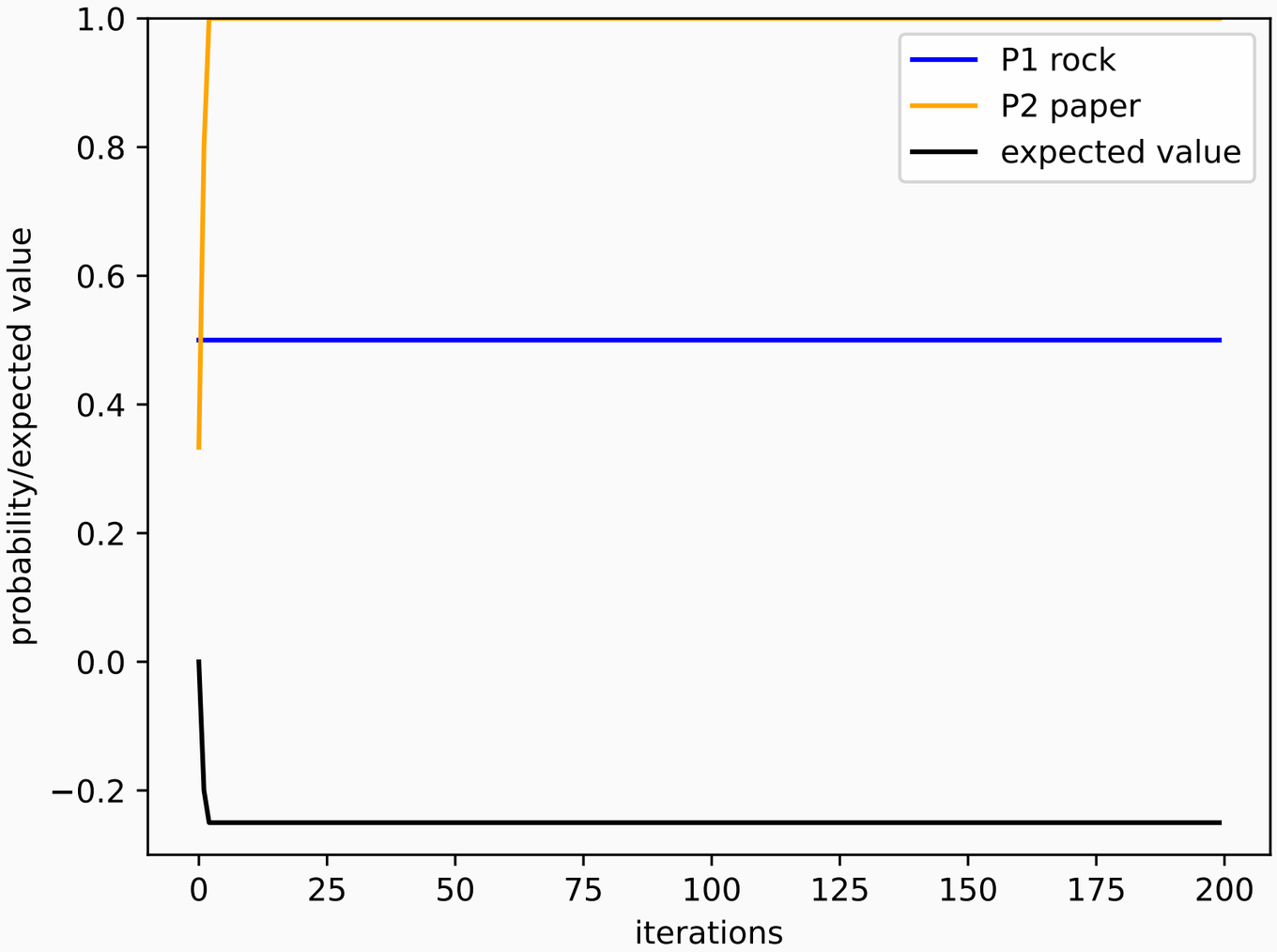
Regret minimization



For example, initially for P1, for actions $(\textit{rock}, \textit{paper}, \textit{scissors})$: $\textit{regrets} \leftarrow (0, 0, 0)$:

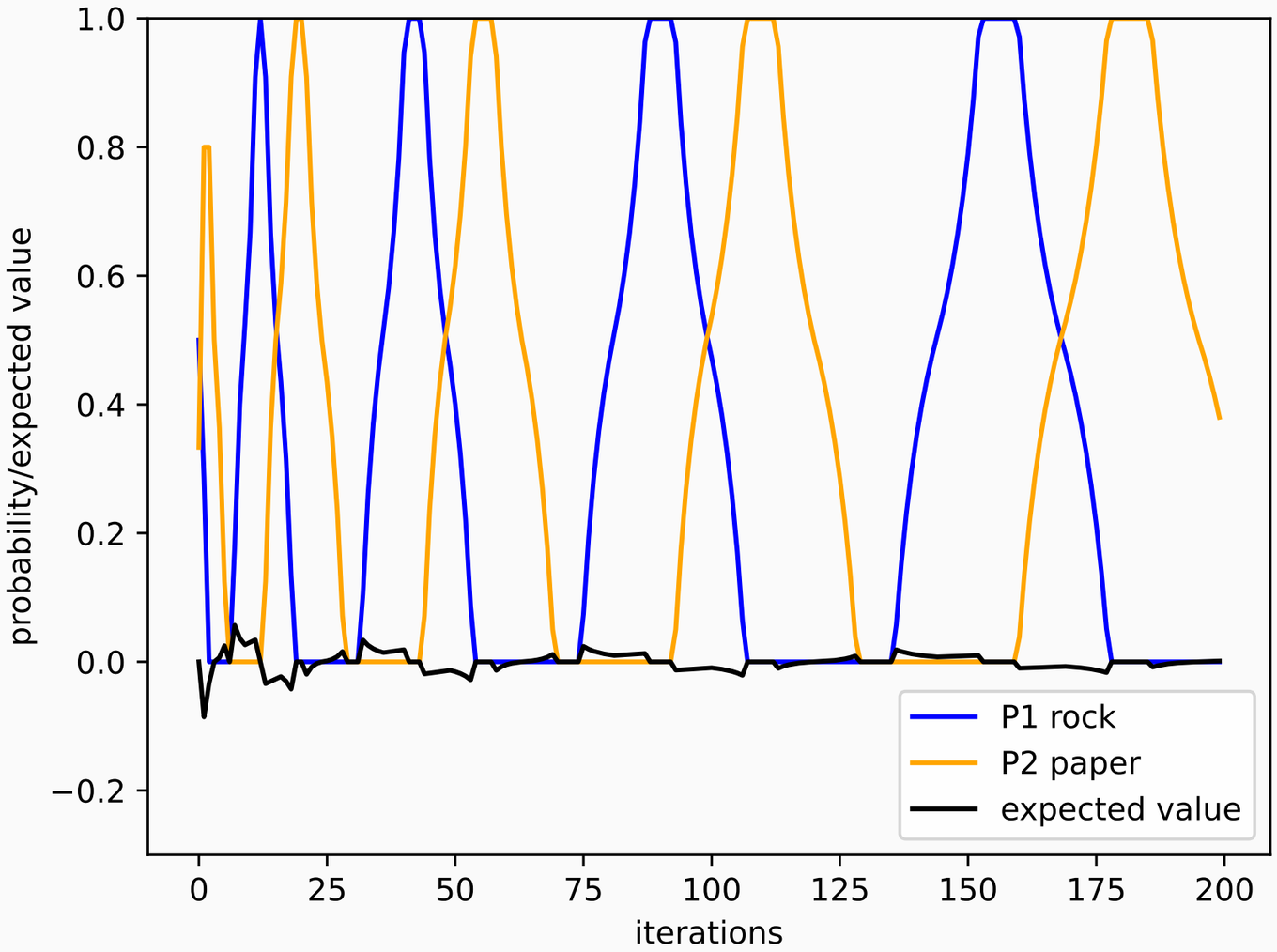
1. P1 chooses rock, P2 chooses paper: $\textit{utility} \leftarrow -1$, $\textit{regret} \leftarrow (-1, 0, 1) - \textit{utility}$
2. Update: $\textit{regrets} \leftarrow \textit{regrets} + \textit{regret}$
3. Result: $\textit{regrets} = (0, 1, 2)$

Example: rock paper scissors, regret minimization result for P2



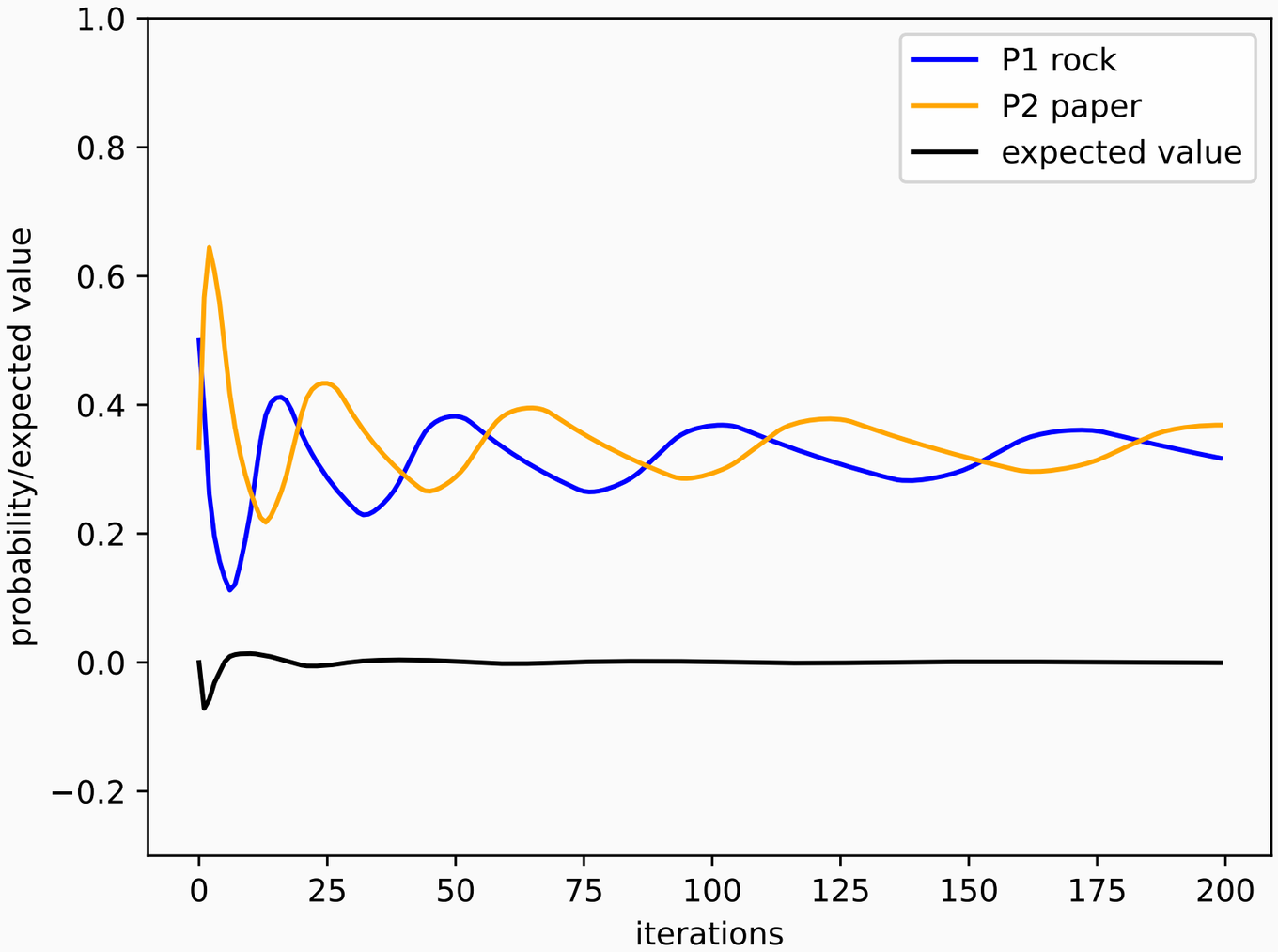
Regrets converted to probabilities, with the negative regrets first set to zero, for example, $(-1, 1, 2) \to (0, 1, 2) \to (0, \frac{1}{3}, \frac{2}{3})$

Example: rock paper scissors, regret minimization results for P1 and P2



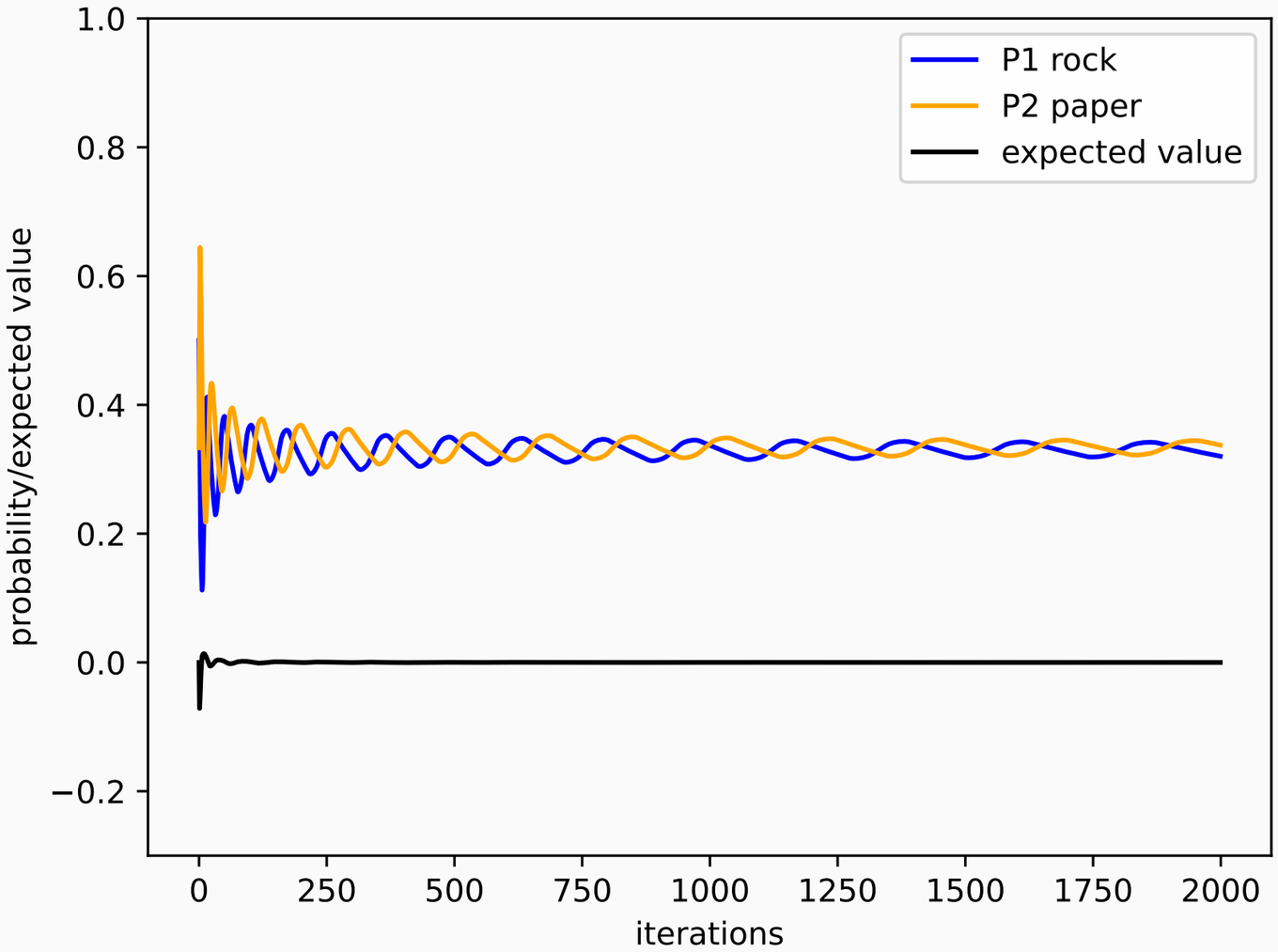
Regrets converted to probabilities, with the negative regrets first set to zero, for example, $(-1, 1, 2) \to (0, 1, 2) \to (0, \frac{1}{3}, \frac{2}{3})$

Example: rock paper scissors, regret minimization results for P1 and P2



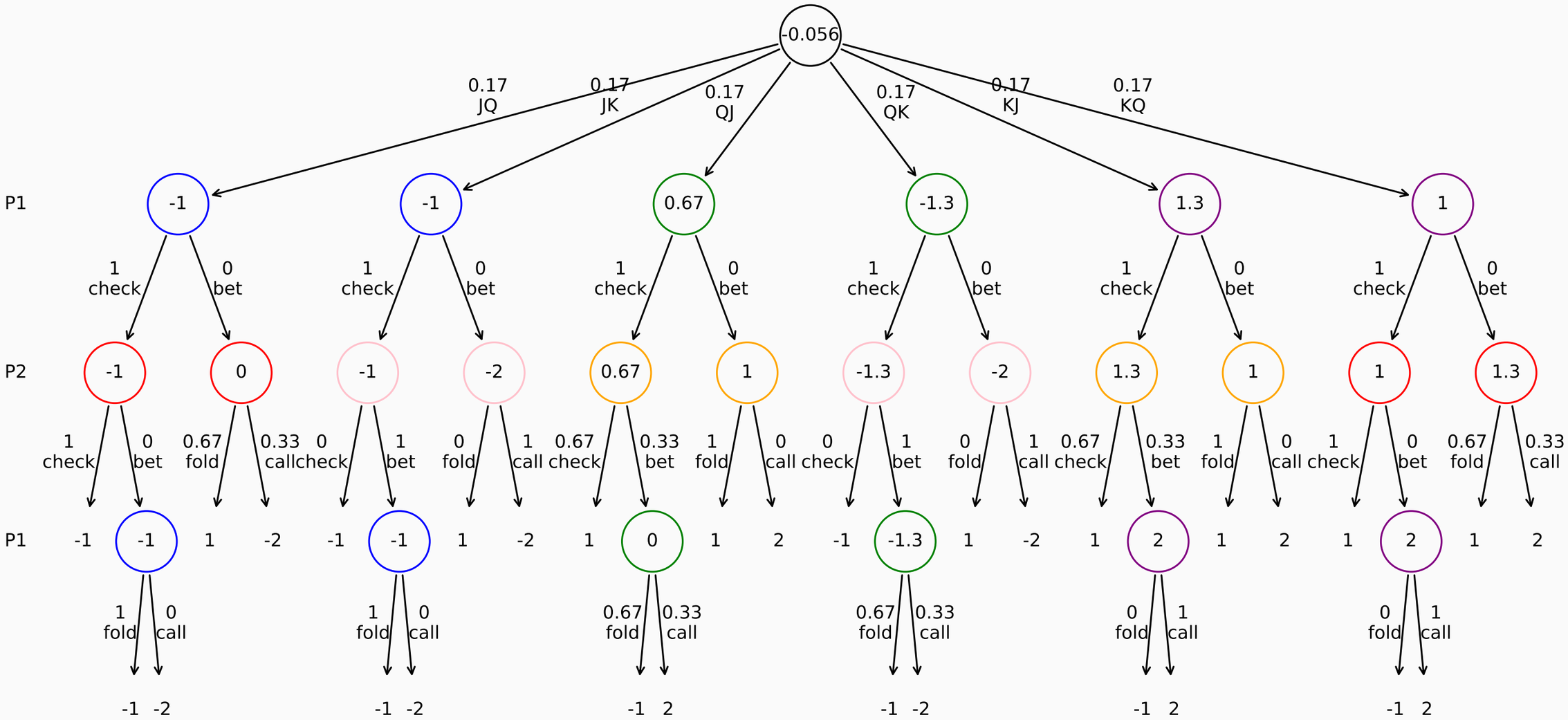
Convergence to Nash equilibrium can be achieved by using the cumulative average, $CA(x) = \frac{x_1 + \ldots + x_n}{n}$, for each iteration n and the probabilities $x_1, \ldots, x_n$ up until then

Example: rock paper scissors, regret minimization results for P1 and P2



Convergence to Nash equilibrium can be achieved by using the cumulative average, $CA(x) = \frac{x_1 + \ldots + x_n}{n}$, for each iteration n and the probabilities $x_1, \ldots, x_n$ up until then

Example: Kuhn poker, a Nash equilibrium



Pluribus (2019): poker bot

- Consistently beat professional poker players at 6-player no-limit Hold'em
- Key methods:
  - Regret minimization
  - Monte Carlo tree search
  - Self-play

Image by Todd Klassy - https://www.flickr.com/photos/latitudes/66492870/in/set-1442169/, CC BY 2.5

AlphaStar (2019): StarCaft II bot

- Similar to AlphaZero, but was initially made to imitate the moves of the best human players
- Grandmaster level (top 0.2 percent of human players)