UMass Amherst

COMPSCI 383

# Artificial Intelligence

February 5, 2020

Lecture 4

# Adversarial Search

# Announcements & Administrivia

- Homework 1 due 11:59pm on Thursday, 2/13.
- Homework 0 is grades will be released soon; please submit a regrade request if you feel you have been wronged.
- Check the Moodle page for updated times/locations of UCA/TA office hours.
- Today I have to leave immediately after class.

# Our Time Today

- Understanding how machines can “play” games
- Game playing as search in adversarial environments
- Key concepts: game trees, Min and Max players, utilities of game states
- Methods for searching in games: minimax algorithm, alpha-beta pruning, approximate evaluation functions
- Extending to games with chance elements

# Some Context

- In previous lectures, we've talked about searching state spaces for a goal state, by considering available state transitions.
- In many cases, the state space comprises the possible configurations of some system or object.
- We use search to select the best transitions for our agent to reach a goal state from some initial state.
- The search procedure produces a solution path made up of states and legal transitions.

# Searching in Games

- For games, the **state** is the total configuration of a game in progress. The successor function defines the legal moves given the game state.
- For the types of games we're going to talk about, there is more than one agent selecting state transitions.
- There may be multiple, equivalent goal states. Furthermore, our goal states are not the same as our opponent's!
- We will identify and order goal states using a **utility function**, which gives us the "value" of any given state toward winning the game.

GREETINGS PROFESSOR FALKEN.
SHALL WE PLAY A GAME?

# Why are games interesting to AI?

- Simple to represent and reason about
- Must consider the moves of an adversary
- Russell & Norvig say:
  “Games, like the real world, therefore require the ability to make *some* decision even when calculating the *optimal* decision is infeasible.”

# We're Bad at Games

- Chinook was declared the Man-Machine World Champion in checkers (1994)
- Deep Blue defeated the reigning world chess champion Garry Kasparov (1997)
- Google's DeepMind AlphaGo defeated the world's number one Go player Ke Jie (2017)

# Types of Games

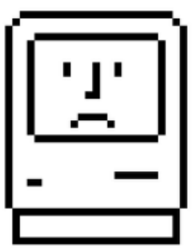
| | deterministic | stochastic |
|---|---|---|
| perfect info | chess, checkers, tic-tac-toe, go, othello | backgammon, monopoly |
| partial info | battleship | poker, scrabble, bridge |

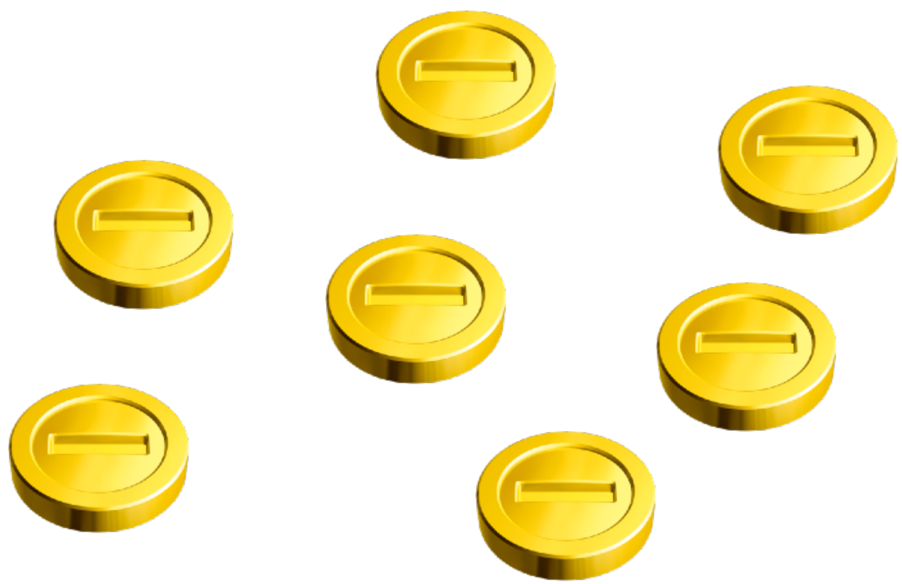
# Basic Games

- 2-players, alternating moves
- Zero-sum: my gain is your loss!
  (Cooperative, non-zero sum games apply game theory)
- Perfect information: both players know the complete game state
  (Hidden information games like Poker require belief states)
- Deterministic: no elements of chance (for the moment)

# Example: Nim

- On your turn, pick up 1 or 2 coins
- Whoever picks up the last coin loses





# Click time!

You are Player 2 in a game of 5-Nim. Before your opponent starts the game, you do some thinking about strategy. Which of the following statements is true?

(A) You can only win if Player 1 does not act rationally.

(B) You cannot win if Player 1 selects one coin.

(C) You can only win if Player 1 selects two coins.

(D) Your chances of winning are the same no matter how many coins Player 1 selects.

# AI Game Terminology

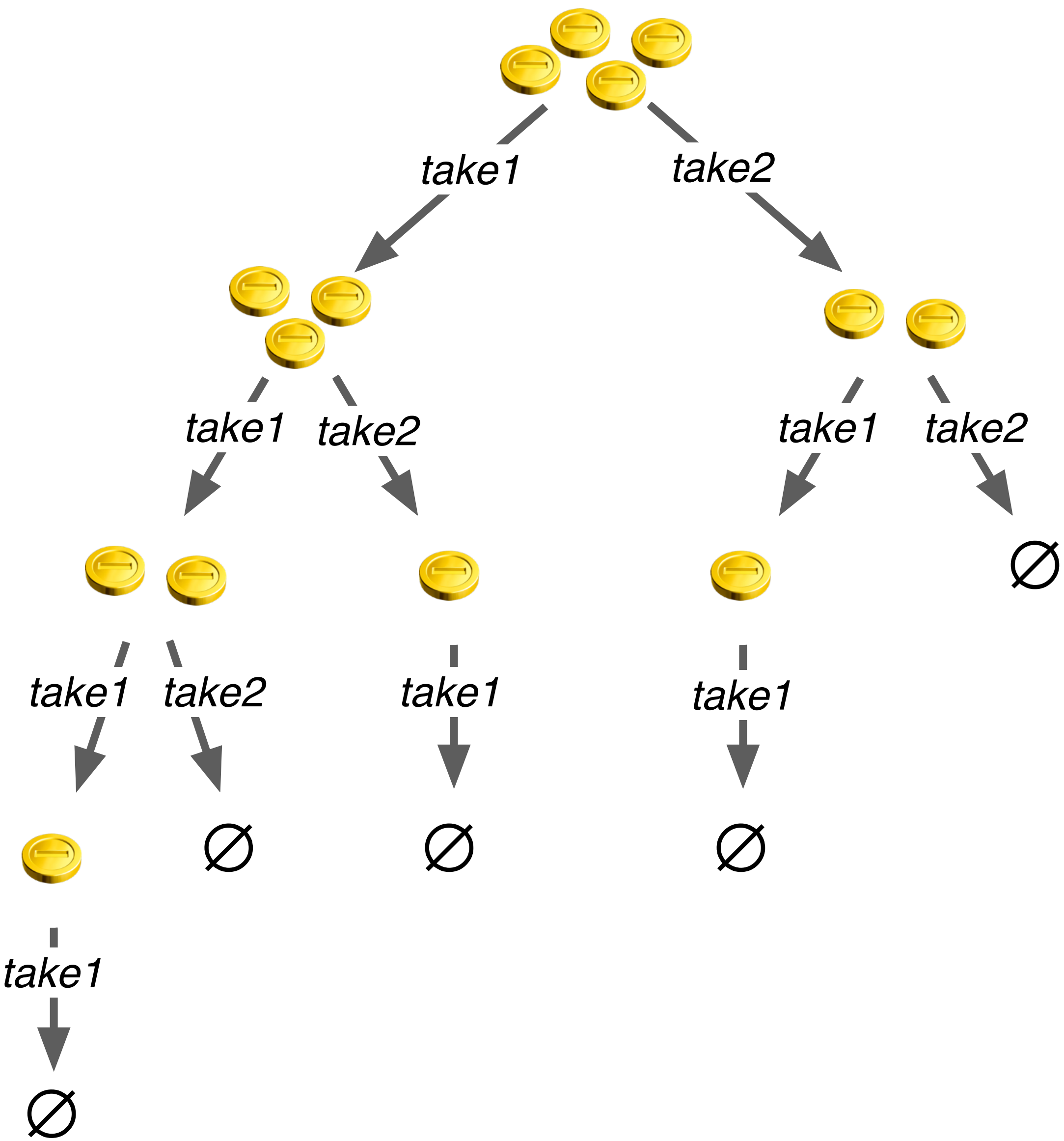
**Adversarial Search** the process of finding optimal state transitions in the presence of other agents who are working against us

**Game Tree** data structure defined by the initial game state and the legal moves for each player

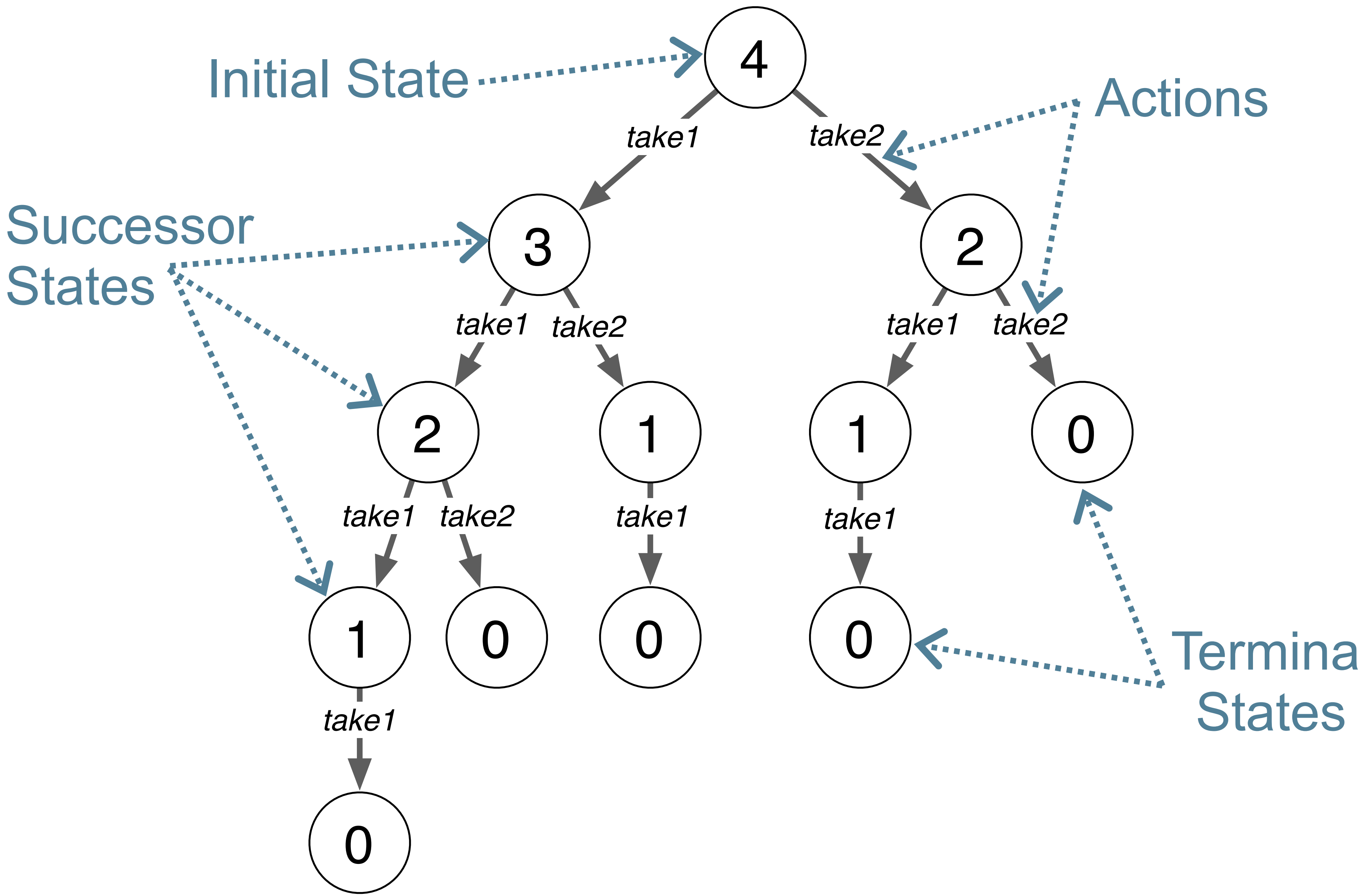
**Ply** a level of the search tree defined by a move by a single player

**Minimax Value** the value of a node in the game tree for a given player, assuming that both players play optimally to the end of the game

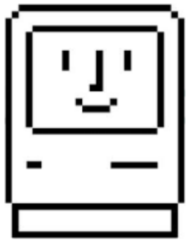
# 4-Nim State Space



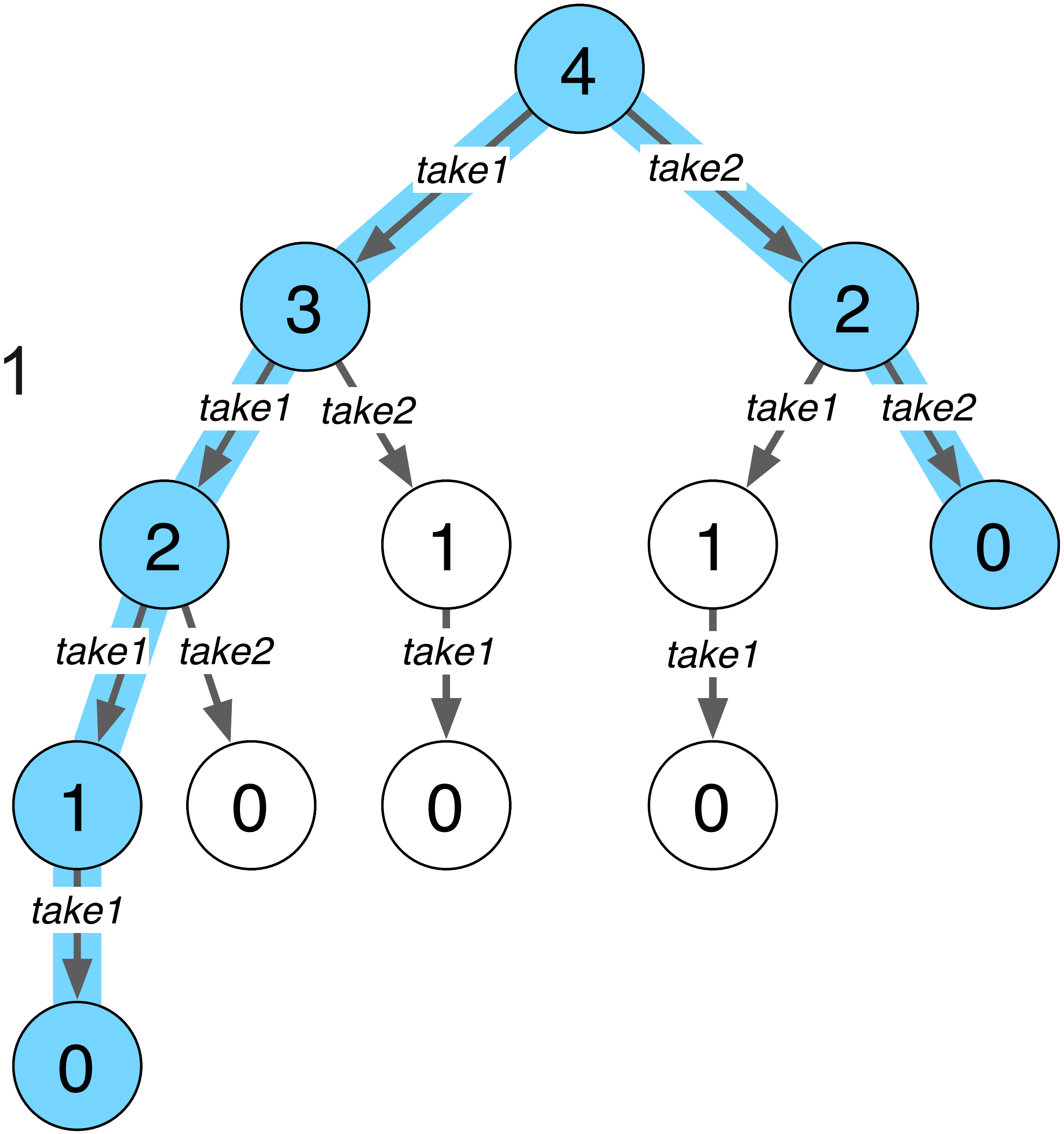
# 4-Nim State Space



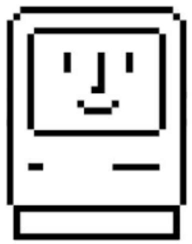
# 4-Nim State Space



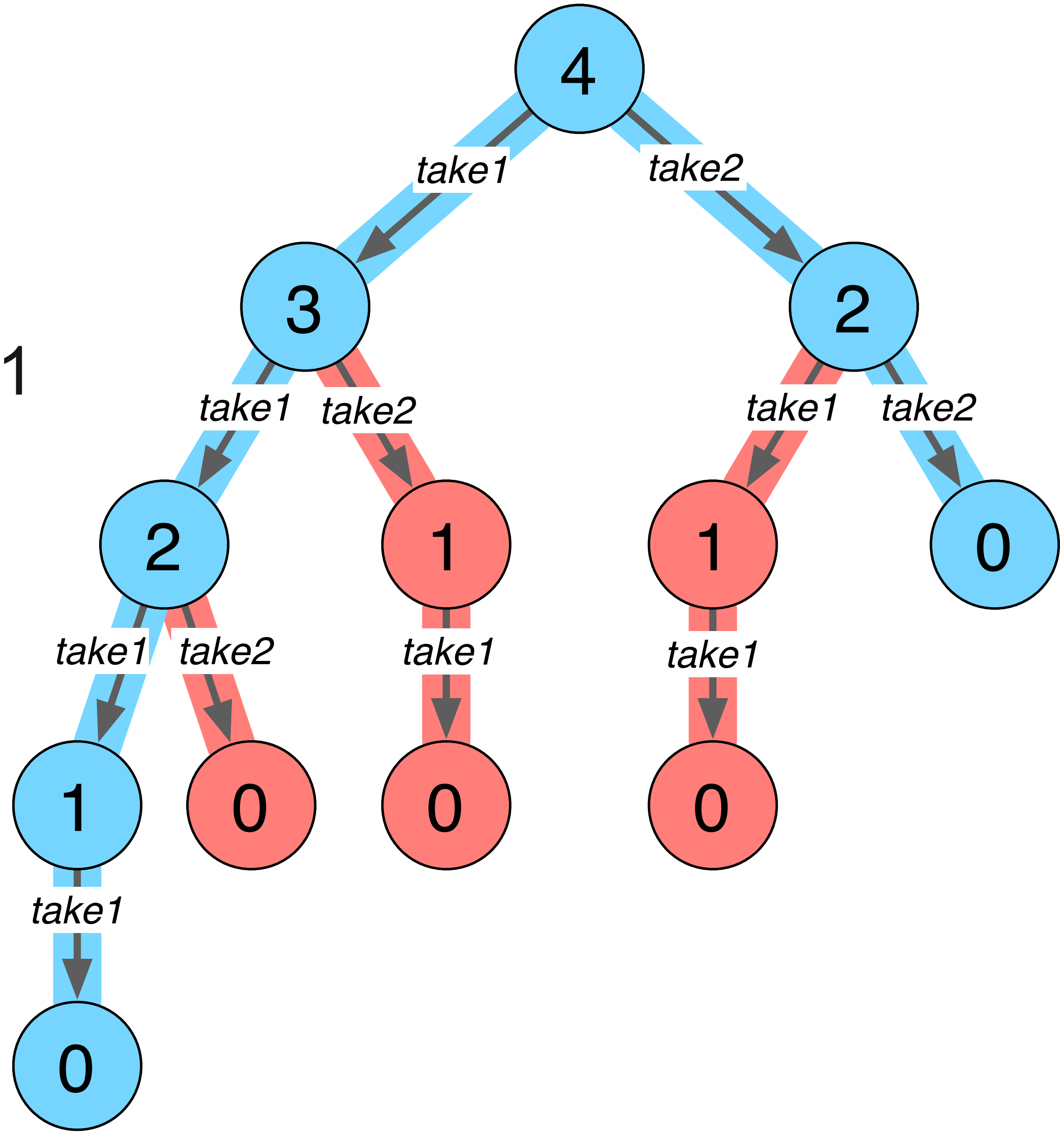
Player 1



# 4-Nim State Space



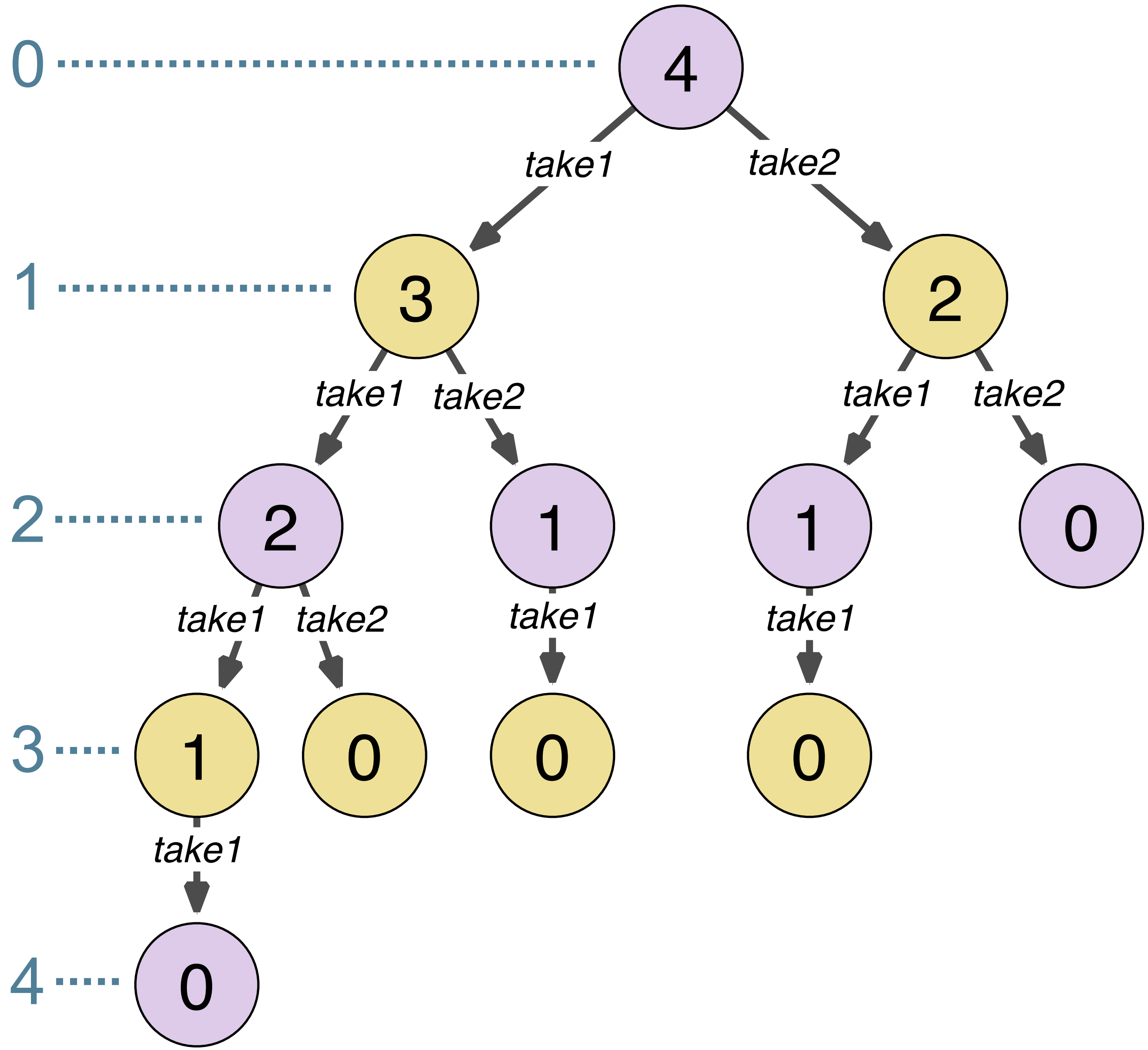
Player 1




Player 2

# Game Ply (4-Nim)

The **ply** (depth) indicates whose turn it is to move

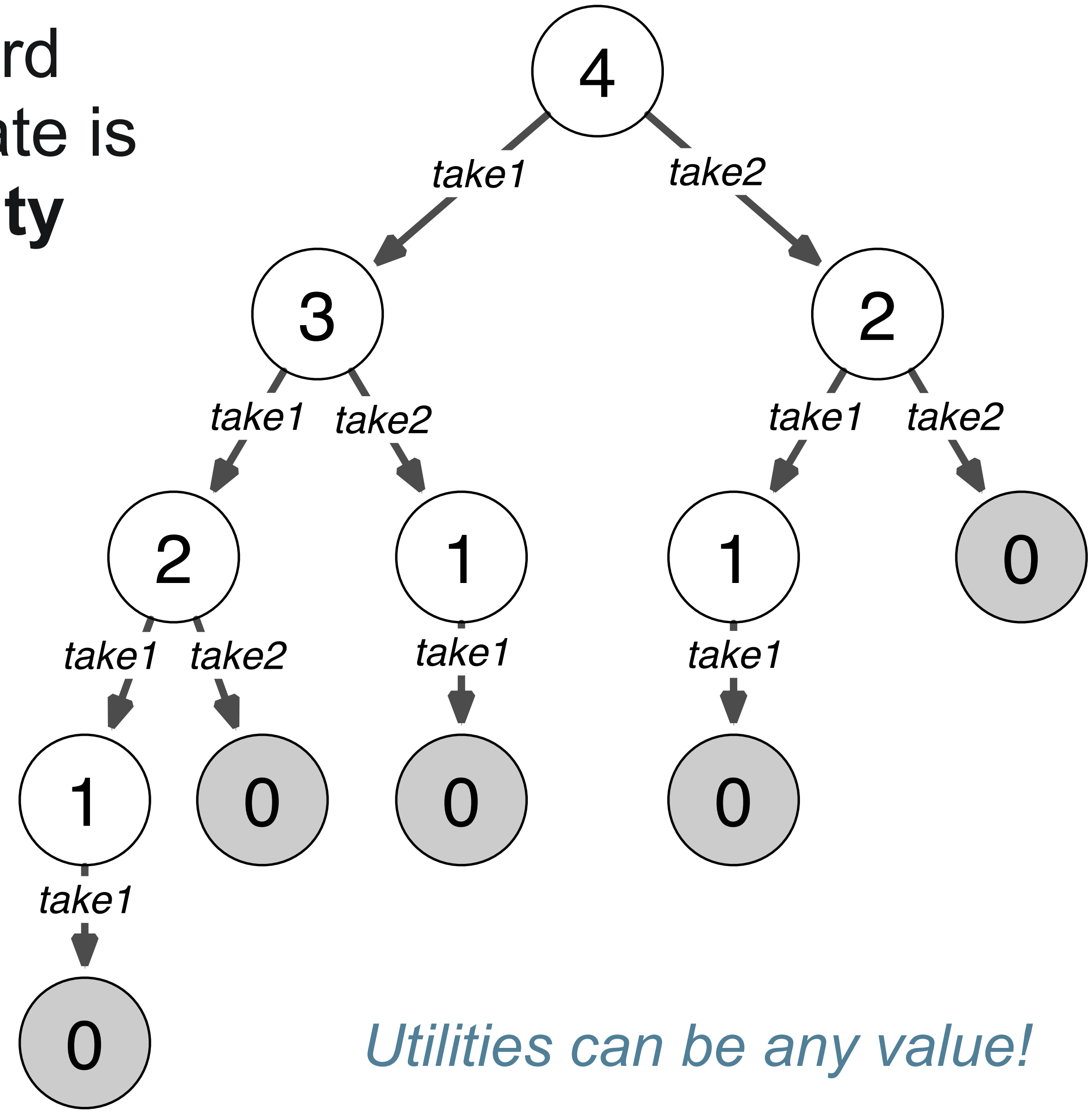


# Utilities (4-Nim)

The amount of reward for each terminal state is captured by the **utility function**.

$P_1$ Utility:
Win = 1
Loss = -1
Draw = 0

$P_2$ Utility:
Win = 1
Loss = -1
Draw = 0




$$\left.\begin{aligned} U(P_1) &= 1 \\ U(P_2) &= -1 \end{aligned}\right\}$$

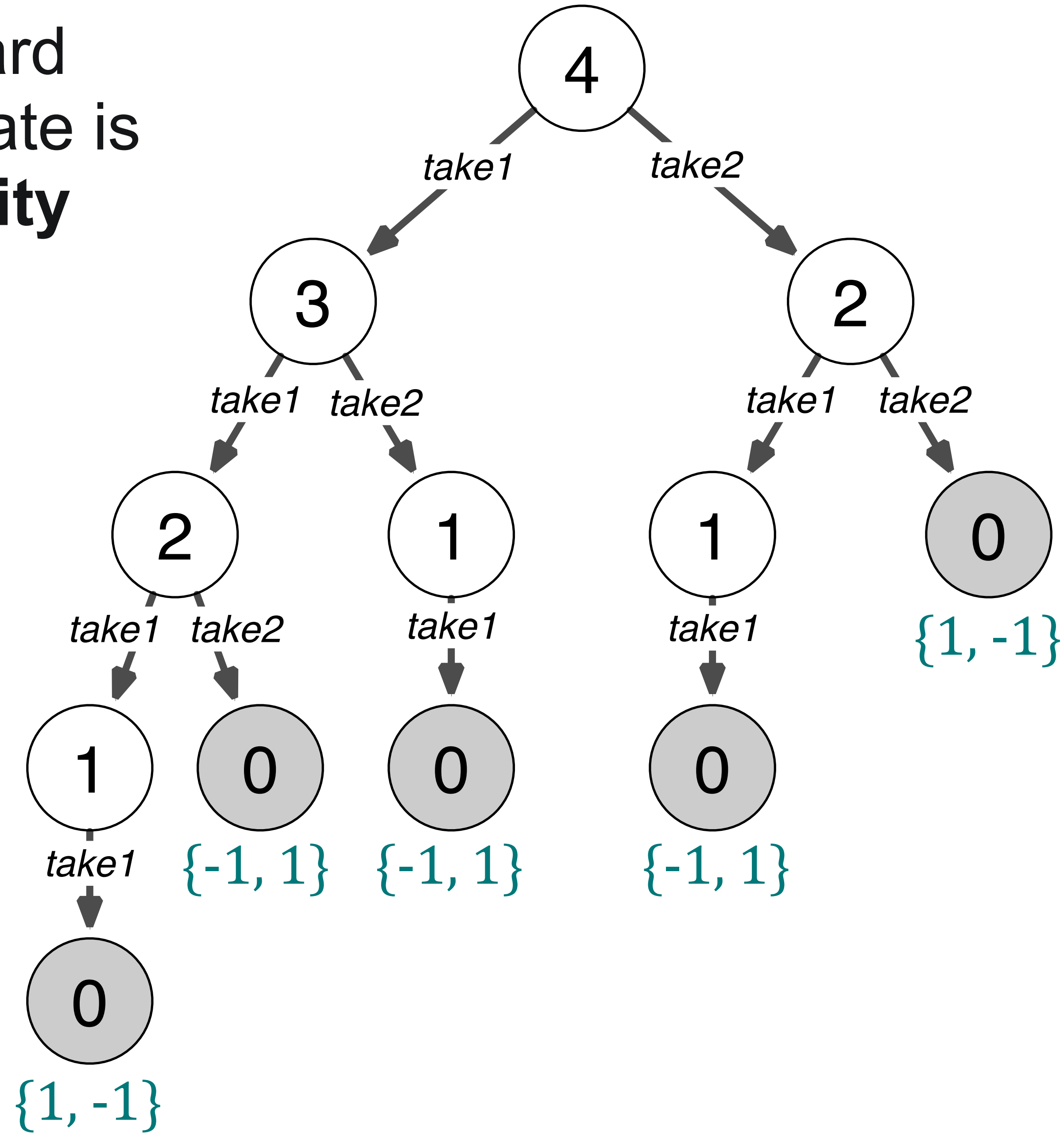
*Utilities can be any value!*

# Utilities (4-Nim)

The amount of reward for each terminal state is captured by the **utility function**.

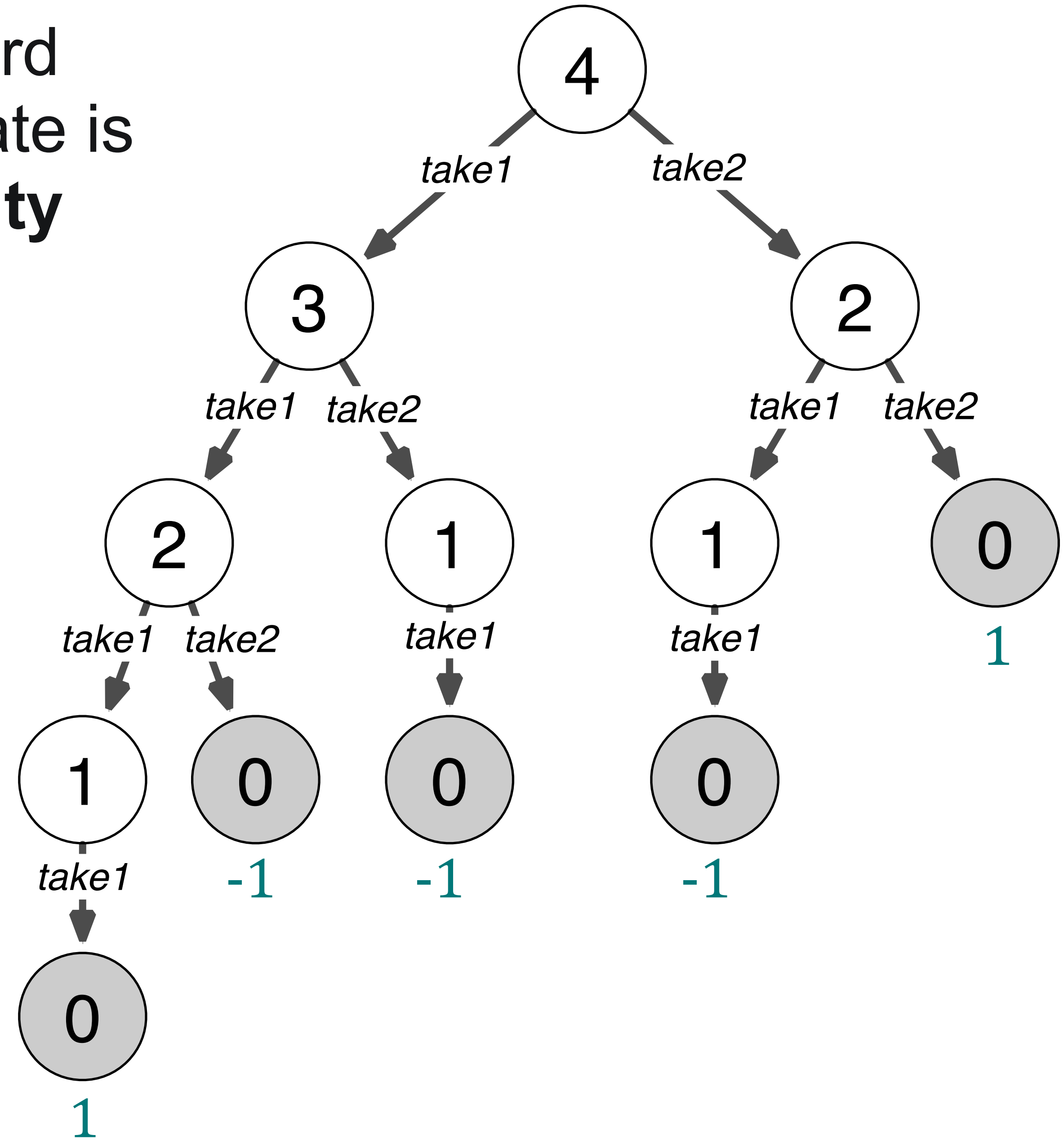
$P_1$ Utility:
Win = 1
Loss = -1
Draw = 0

$P_2$ Utility:
Win = 1
Loss = -1
Draw = 0



# Utilities (4-Nim)

The amount of reward for each terminal state is captured by the **utility function**.




| $P_1$ Utility: | $P_2$ Utility: |
| --- | --- |
| Win = 1 | Win = 1 |
| Loss = -1 | Loss = -1 |
| Draw = 0 | Draw = 0 |

If game is **zero sum**, then we only need to show one utility — the one for Player 1.

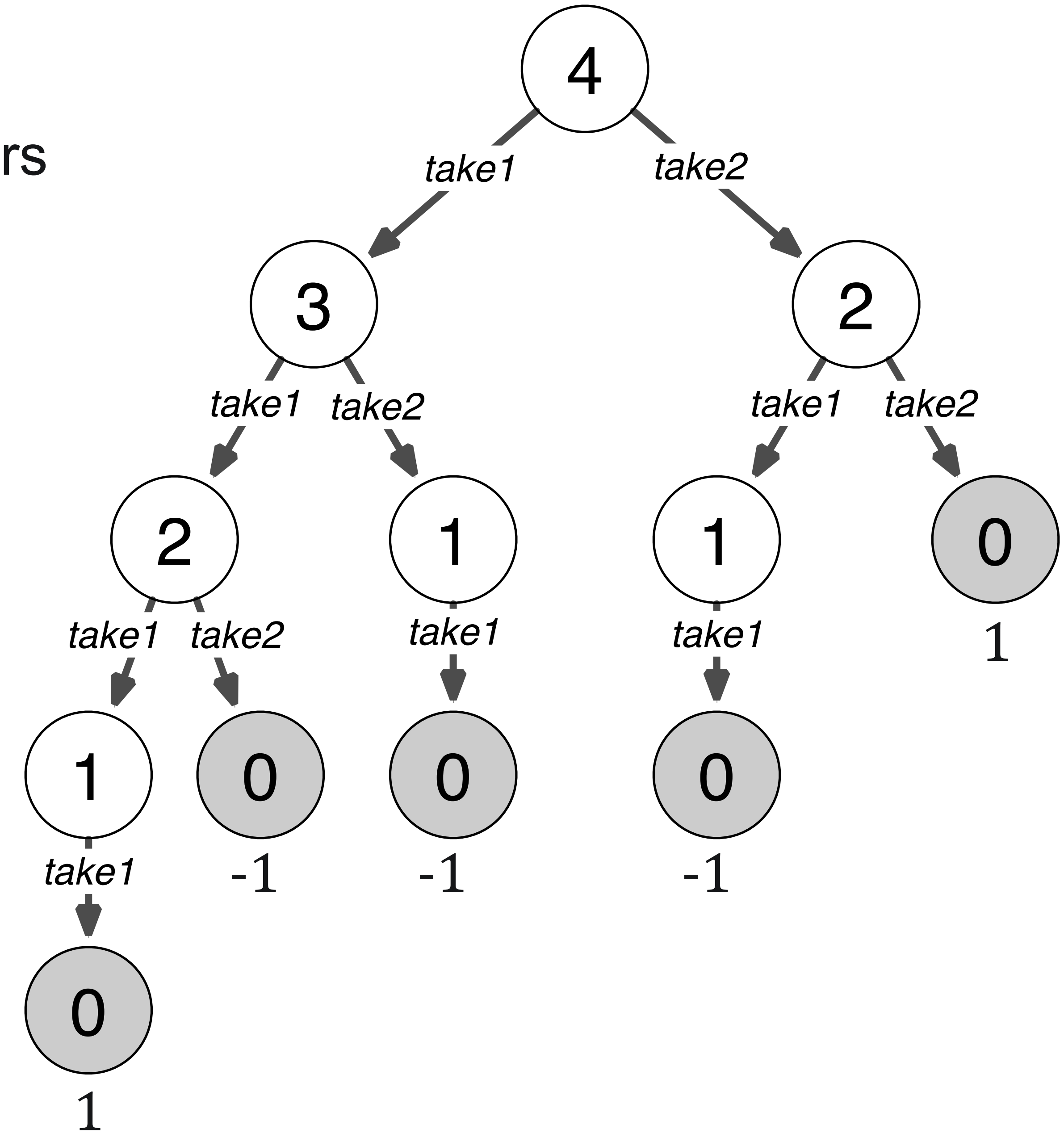
# Minimax (4-Nim)

If we assume both players play optimally:

$P_1$ will **maximize** $U(P_1)$

$P_2$ will **minimize** $U(P_1)$

We can compute the expected value of an interior node with the **Minimax** algorithm.



*How can we determine the utility of non-terminal states?*

*Minimax*

# Minimax

- The Minimax algorithm allows us to determine the utility value of any state in the game tree.
- Minimax assumes that your opponent will behave rationally (which might not be true).
- By selecting the action leading to a state with the highest value, we can maximize our expected payoff.

# Minimax Algorithm Pseudocode

**function** MINIMAX-DECISION(*state*) **returns** *an action*
  **inputs**: *state*, current state in game

  **return** the $a$ in ACTIONS(*state*) maximizing MIN-VALUE(RESULT($a$, *state*))

---

**function** MAX-VALUE(*state*) **returns** *a utility value*
  **if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  **for** $a$, $s$ in SUCCESSORS(*state*) **do** $v \leftarrow$ MAX($v$, MIN-VALUE($s$))
  **return** $v$

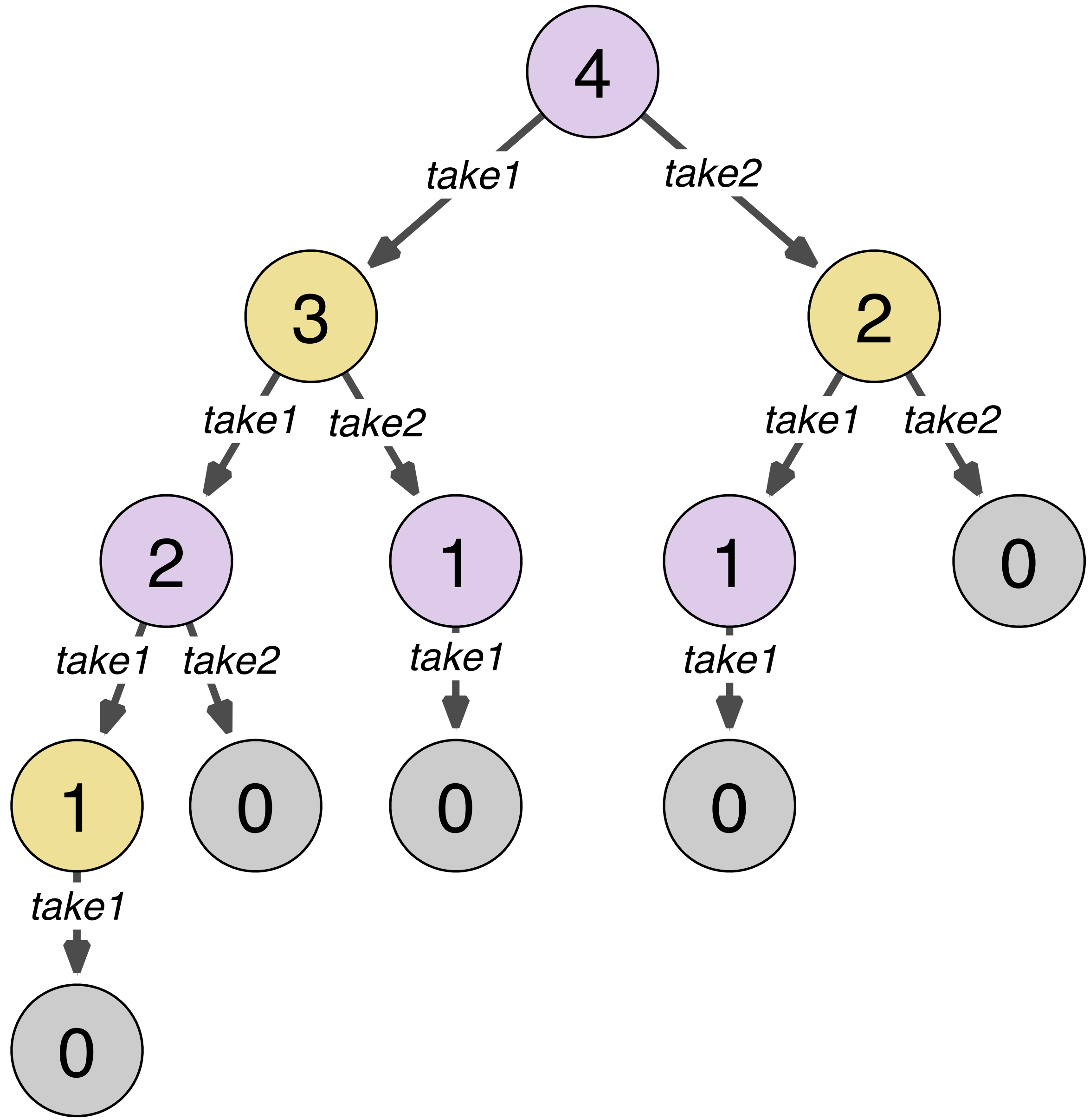
---

**function** MIN-VALUE(*state*) **returns** *a utility value*
  **if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  **for** $a$, $s$ in SUCCESSORS(*state*) **do** $v \leftarrow$ MIN($v$, MAX-VALUE($s$))
  **return** $v$

# Minimax (4-Nim)

Strategy:
select moves
with highest
minimax value.

That is, select
the best
achievable
payoff against
best play by
your opponent.

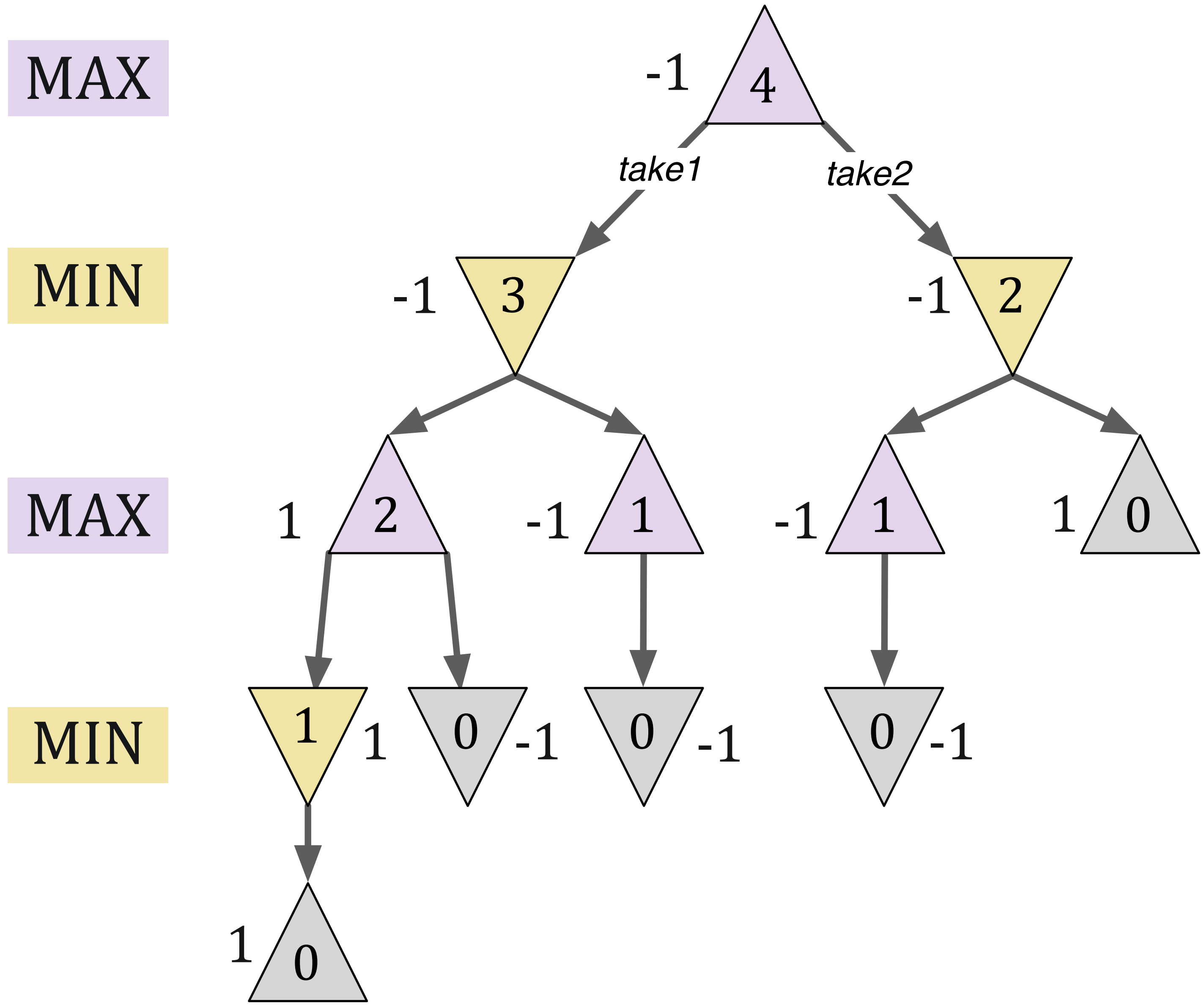


# Minimax (4-Nim)

Strategy: select moves with highest minimax value.

That is, select the best achievable payoff against best play by your opponent.

We use the algorithm to determine the utilities of all states.



# Click time!

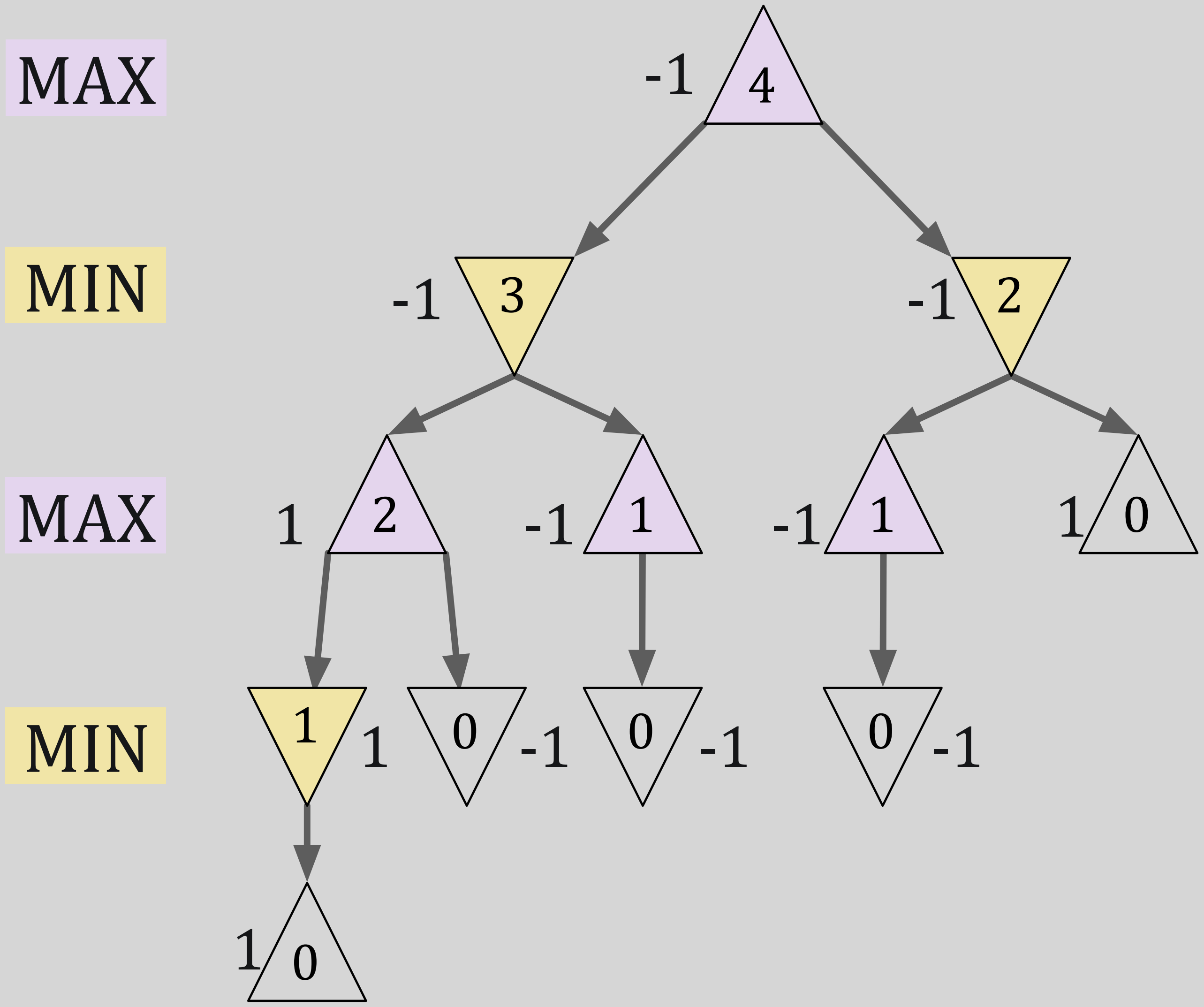
What does the game tree tell us about 4-Nim strategy?

(A) Player 1 has an advantage

(B) Player 2 has an advantage

(C) A priori, neither player has an advantage

(D) Most games will end in a tie



A STRANGE GAME.
THE ONLY WINNING MOVE IS
NOT TO PLAY.

HOW ABOUT A NICE GAME OF CHESS?

# Properties of Minimax

**Complete?** Yes, if tree is finite

**Optimal?** Yes, against optimal opponent

**Time** $O(b^m)$

**Space** $O(bm)$ (depth-first)

$b$ = branching factor, $m$ = max height of tree

…but for chess, b ≈ 35, m ≈100, *so an exact solution is completely infeasible!*

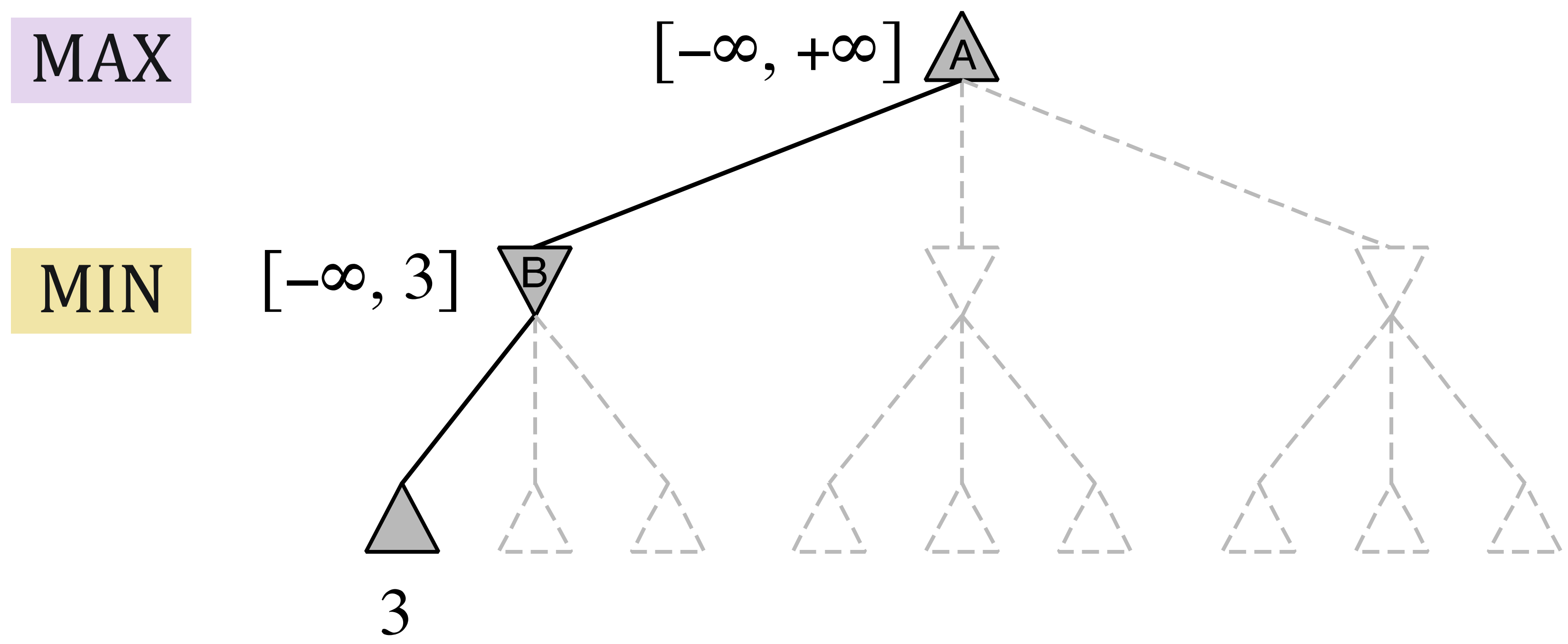
# Game Trees Can Be Big

- Checkers has a branching factor b ≈ 35, with m ≈ 50 yielding 500 billion billion game states.
- Chess has a branching factor of b ≈ 35, with moves m ≈ 100.
- Go has a branching factor b ≈ 250, with m ≈ 200
- For many of these games, Minimax traversing the entire game tree is impossible.

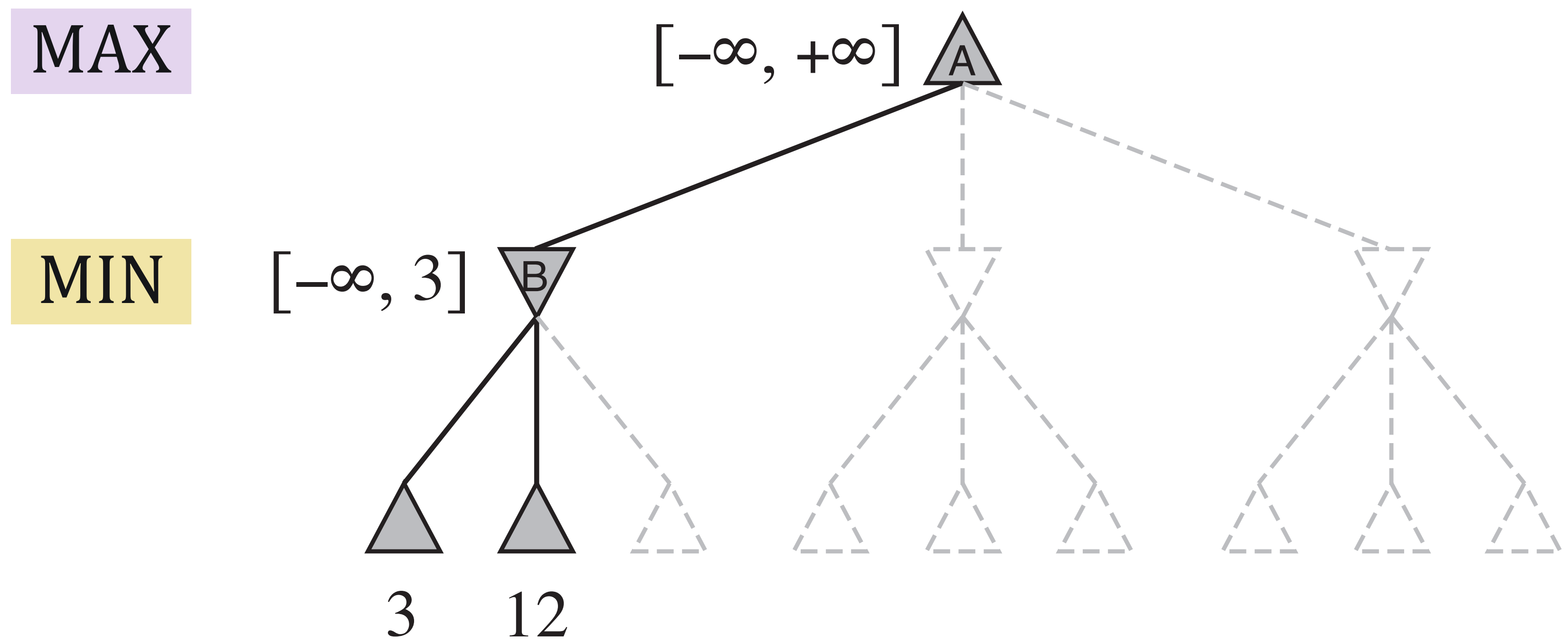
# Improving Minimax: α- β Pruning

- α is the best value (for MAX player) found so far on this path
- β is the best value (for MIN player) found so far on this path
- If we are searching a MAX node and we find even one child with value > β, then our parent MIN node will never choose this action so we should **prune**
- If we are searching a MIN node and we find even one child with value < α, then our parent MAX node will never choose this action so we should **prune**

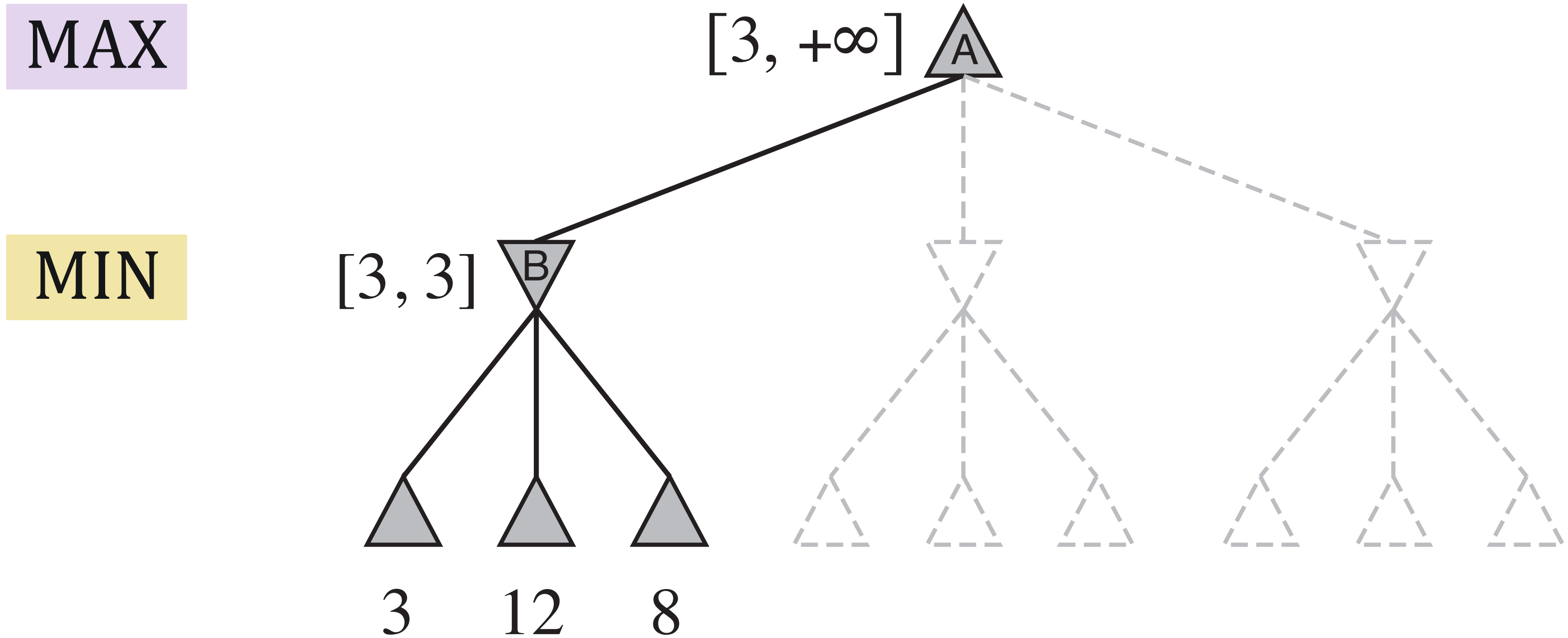
# α-β Pruning



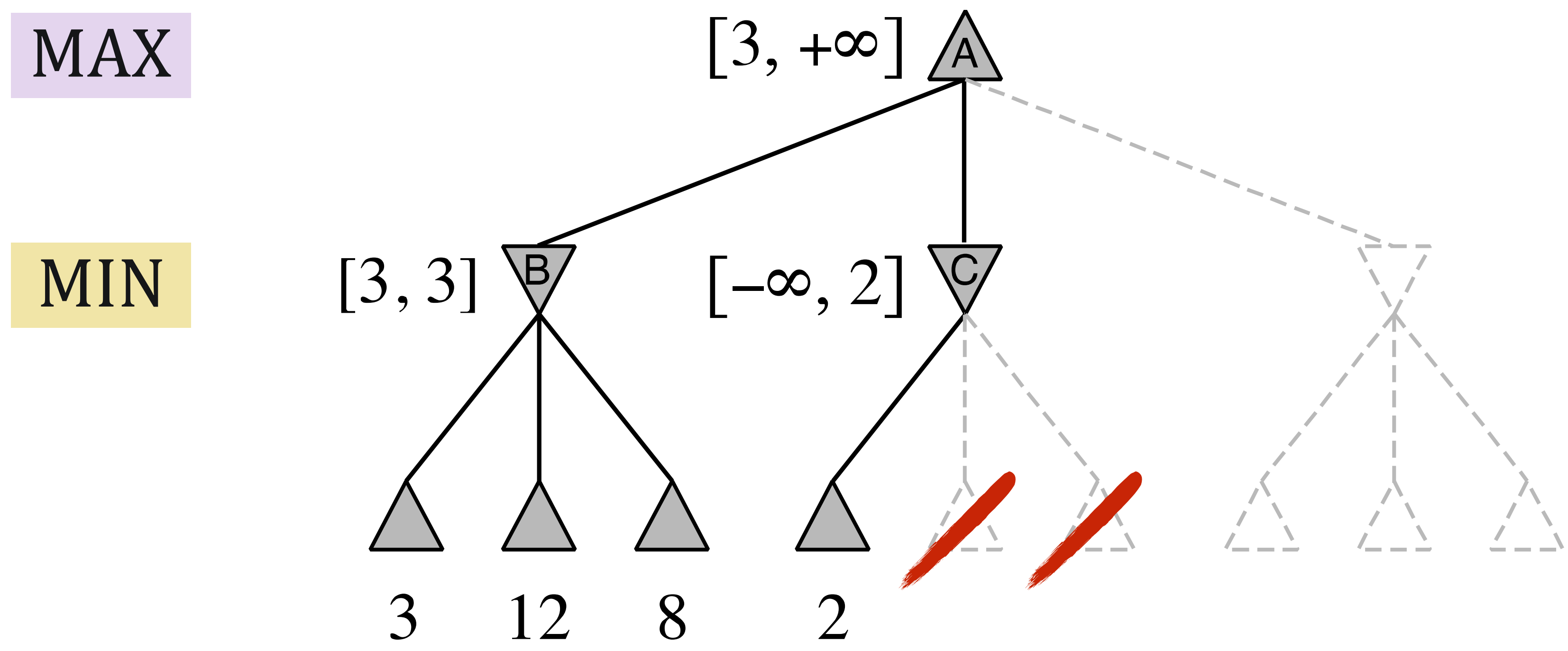
# α-β Pruning



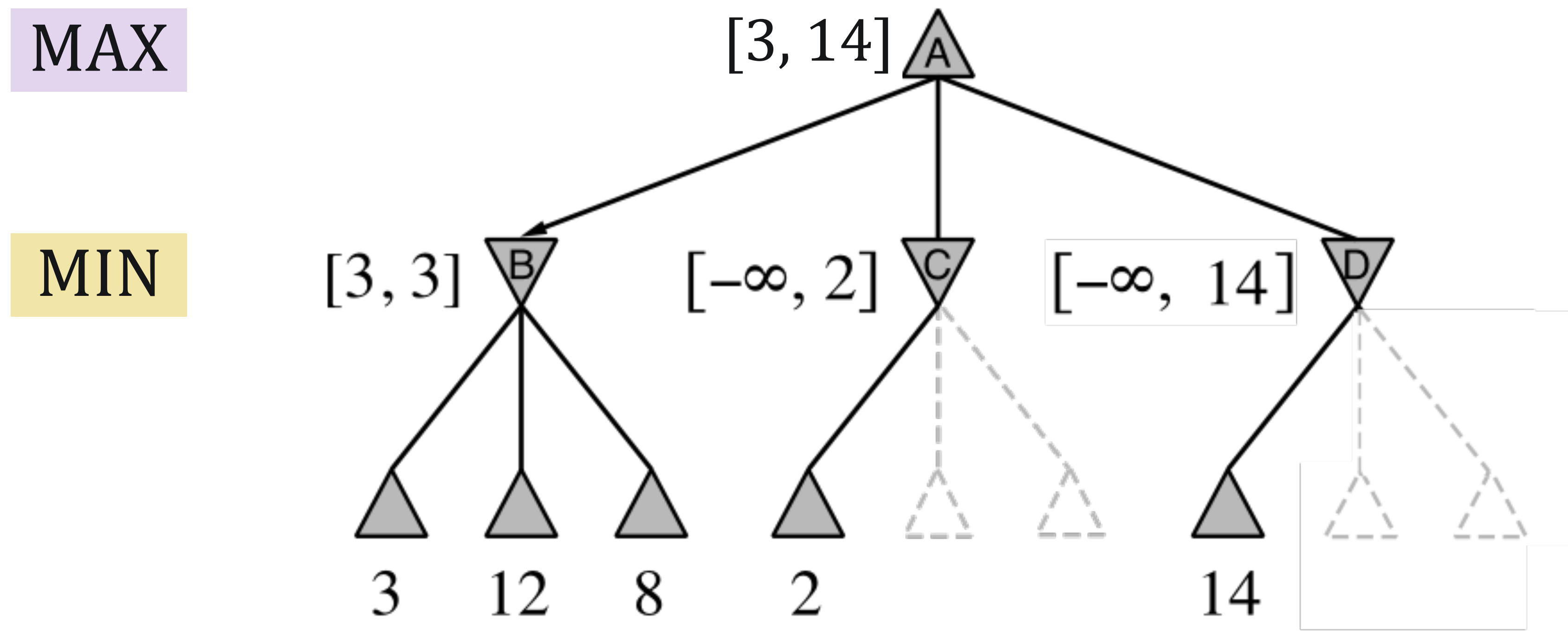
# α-β Pruning



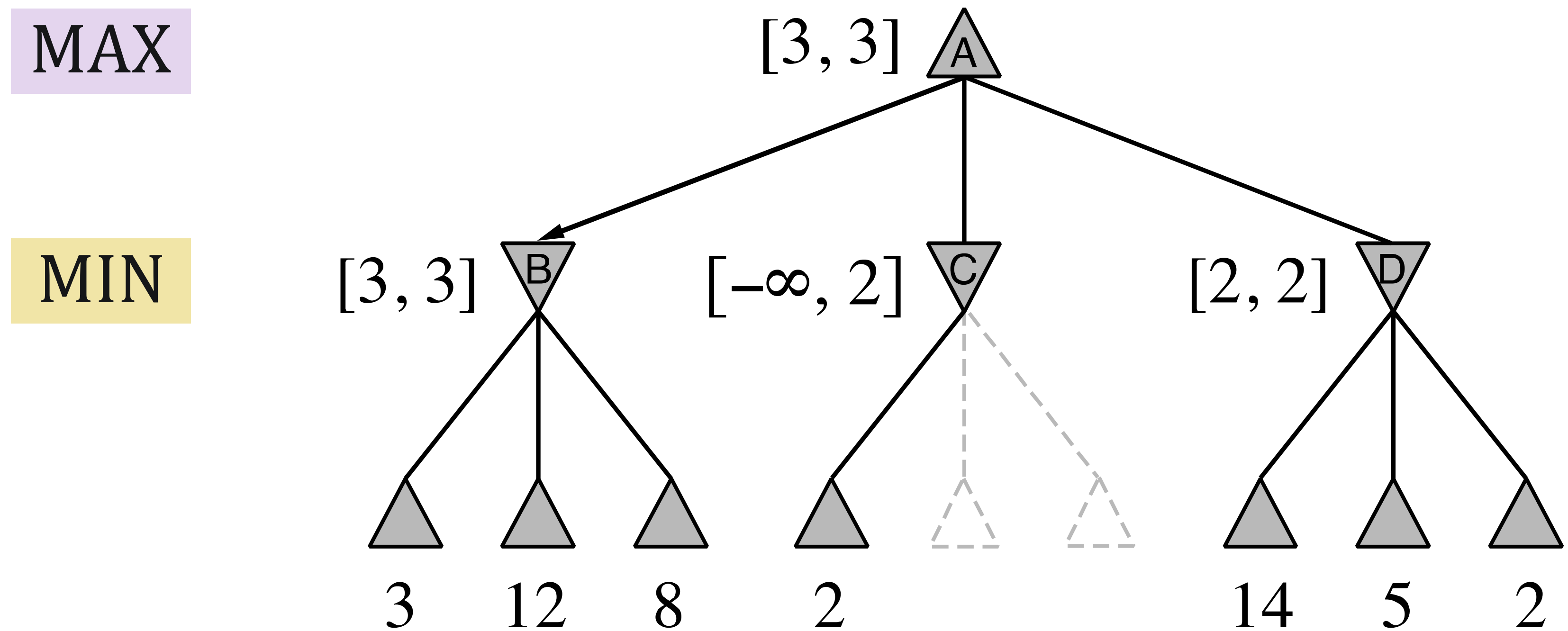
# α-β Pruning



# α-β Pruning



# α-β Pruning



# What just happened?

- α is the value of the best (highest-value) choice found so far at any choice point along the path for MAX
- If v is worse than α, MAX will avoid it, so that branch can be pruned.
- β is the value of the best (lowest-value) choice found so far at any choice point along the path for MIN
- If v is worse than β , MIN will avoid it, so that branch can be pruned.

# α-β Pruning

- Pruning produces results that are exactly equivalent to complete (unpruned) search.
- Entire subtrees can be pruned.
- Node ordering can improve effectiveness. Perfect ordering gives complexity $O(b^{m/2})$, since branching factor goes from b to $\mathrm{sqrt}(b)$. Thus α-β can search twice as far in equal time.
- You can avoid recomputing the value of previously seen states by storing them in a **transposition table**.

# Minimax with Lookahead

- Even with pruning, full exploration of the game tree is often intractable (tic-tac-toe has 9! = 362,880 states, chess has more than $10^{40}$ states!)

- Alternatively, you can stop the search before you reach terminal states using a cutoff test

- At some **lookahead limit** (depth limit), evaluate nodes using an **evaluation function** instead of the normal utility function

- Use the evaluation function values as if they were the true state utility values

# Evaluation Functions

- Must be efficient to calculate
- Must order terminal states in the same way as the true utility function
- Typically calculate **features** – simple characteristics of the game that are strongly correlated with the probability of winning
- The evaluation function combines feature to produce a score. For example:

$$Eval(x) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

*(linear equations assume feature independence, but in practice it often doesn't matter)*

# Example Chess Features

- Total number of pieces
- Relative number of bishops, knights, rooks and/or pawns
- Distance of furthest pawn from start
- Relative freedom (number of possible moves
- Other binary indicators (Has queen? Castled? In check?)

# Games with Chance Elements

- For games with chance elements, we construct a game tree with a separate ply to represent the stochastic moves
- To determine the values in such a game tree, we use a variant of the Minimax algorithm called **Expectiminimax**
- Rather than minimizing or maximizing, we determine the *expected value* of chance nodes

# Expected Value

The sum of the probability of each possible outcome multiplied by its value:

$$E(X) = \sum_i p_i x_i$$

$$E(roll) = \left(\frac{1}{6} * 1\right) + \left(\frac{1}{6} * 2\right) + \left(\frac{1}{6} * 3\right) + \left(\frac{1}{6} * 4\right) + \left(\frac{1}{6} * 5\right) + \left(\frac{1}{6} * 6\right)$$

$$= \frac{1}{6} + \frac{1}{3} + \frac{1}{2} + \frac{2}{3} + \frac{5}{6} + 1 = 3.5$$

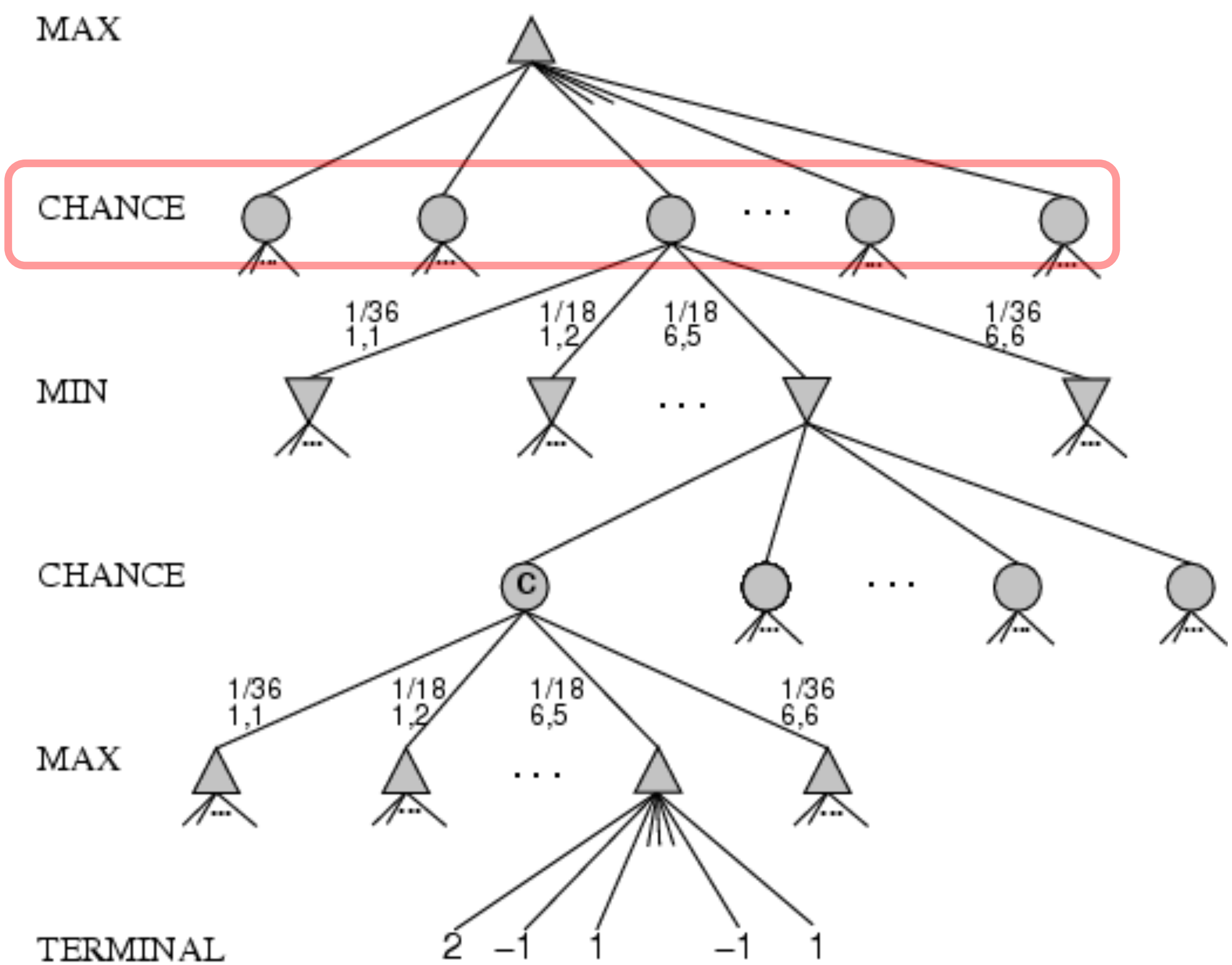
# Expected Value

- The sum of the probability of each possible outcome multiplied by its value:

$$E(X) = \sum_i p_i x_i$$

- There are pathological cases where this statistic could do something strange
  - Extreme values ("outliers")
  - Functions that are a non-linear transformation of the probability of winning

# Game Tree with Chance Elements




*Three different cases to evaluate, rather than just two.*

# Expectiminimax

EXPECTIMINIMAX(n) =

If terminal node, UTILITY(n)

If MAX node, $\max_{s \in \text{successors}(n)}$ MINIMAX(s)

If MIN node, $\min_{s \in \text{successors}(n)}$ MINIMAX(s)

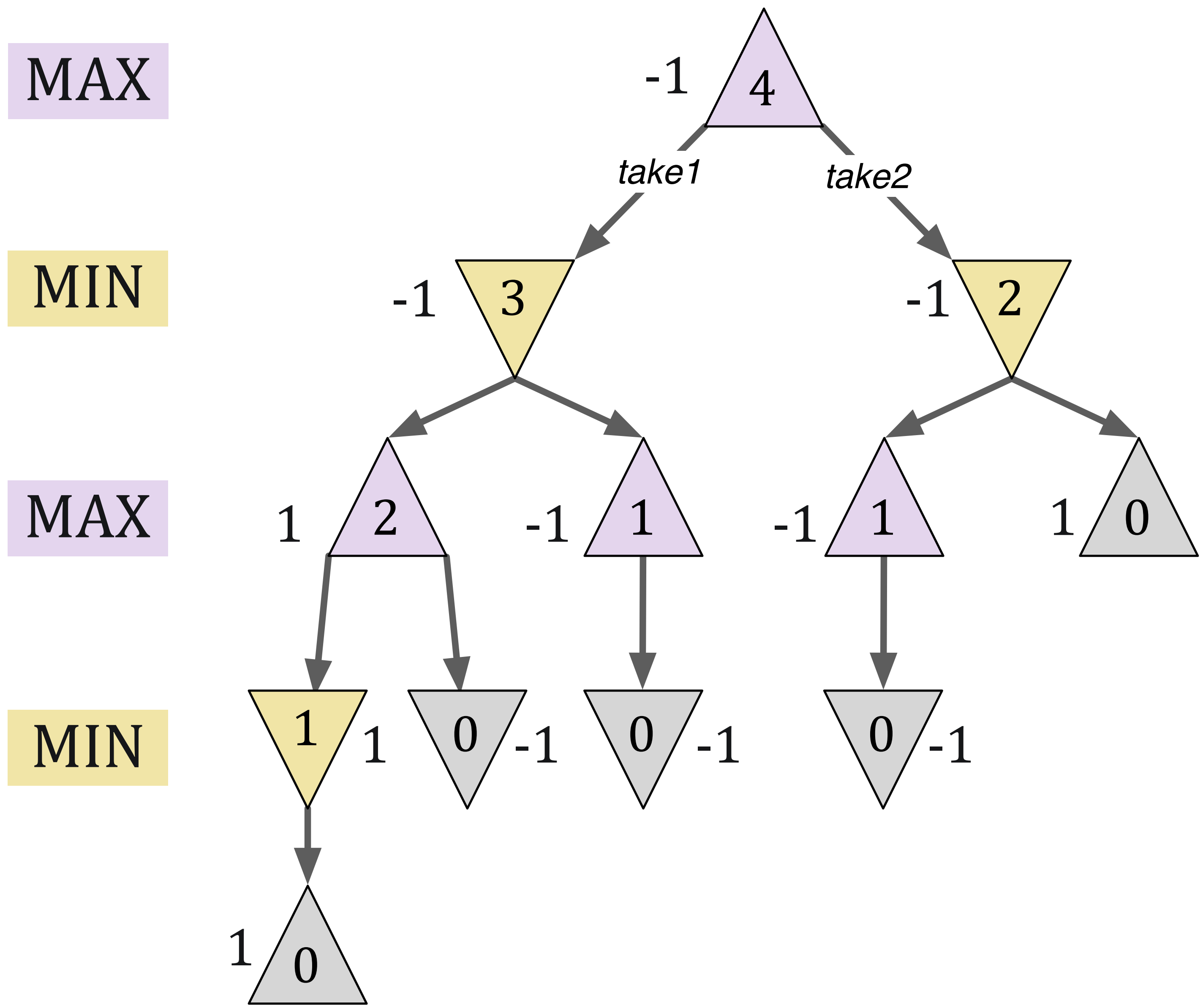
If CHANCE node, $\sum_{s \in \text{successors}(n)}$ P(s) x EXPECTIMINIMAX(s)

# Dropsy Nim

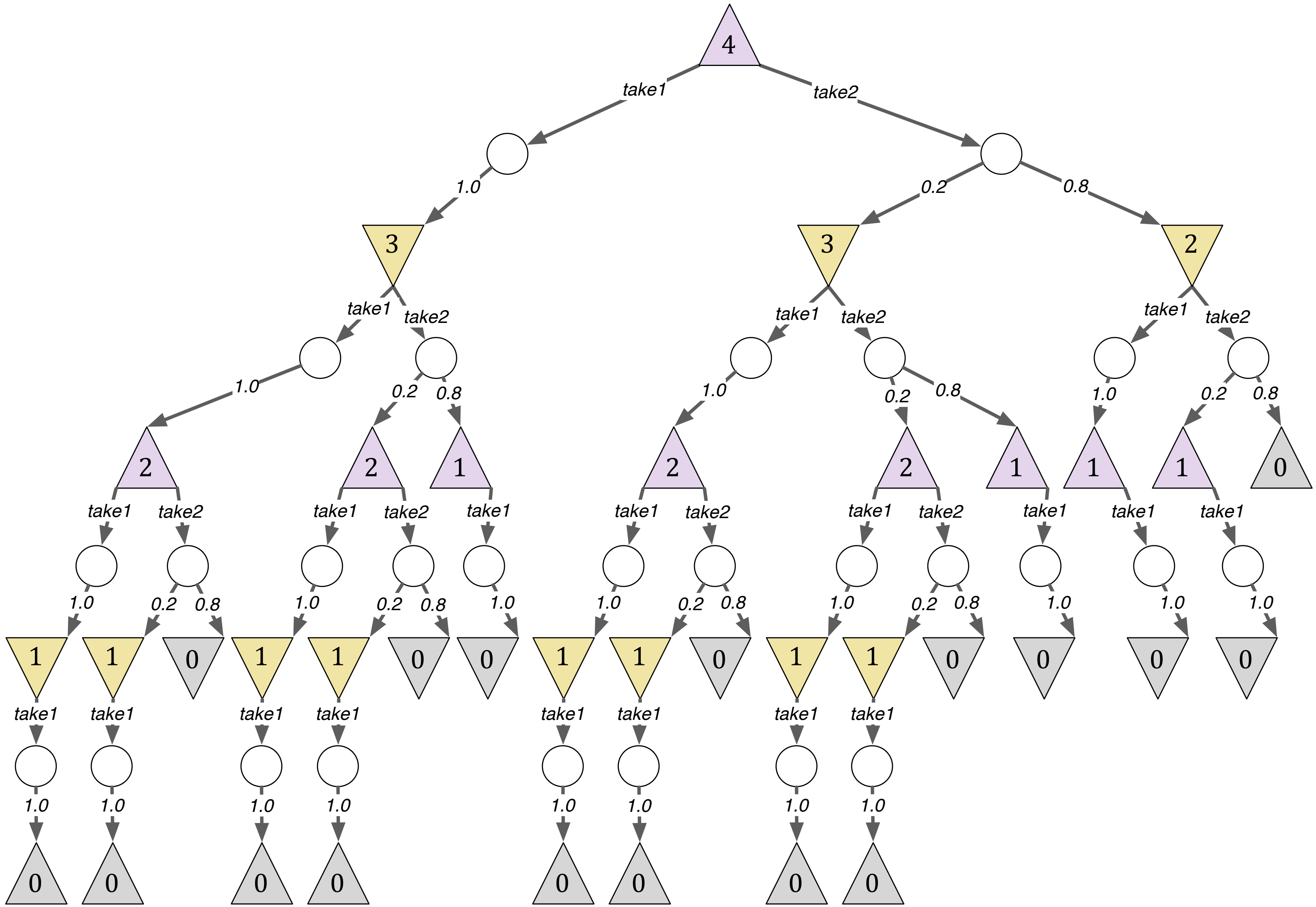
- We can add an element of stochasticity to Nim, where the results of certain actions are uncertain

- “Dropsy Nim” is the same as regular Nim, except when a player picks up two coins, they may only get one, according to some fixed probability (for example, 0.2)

# Minimax (4-Nim)



# Expectiminimax (Dropsy 4-Nim)



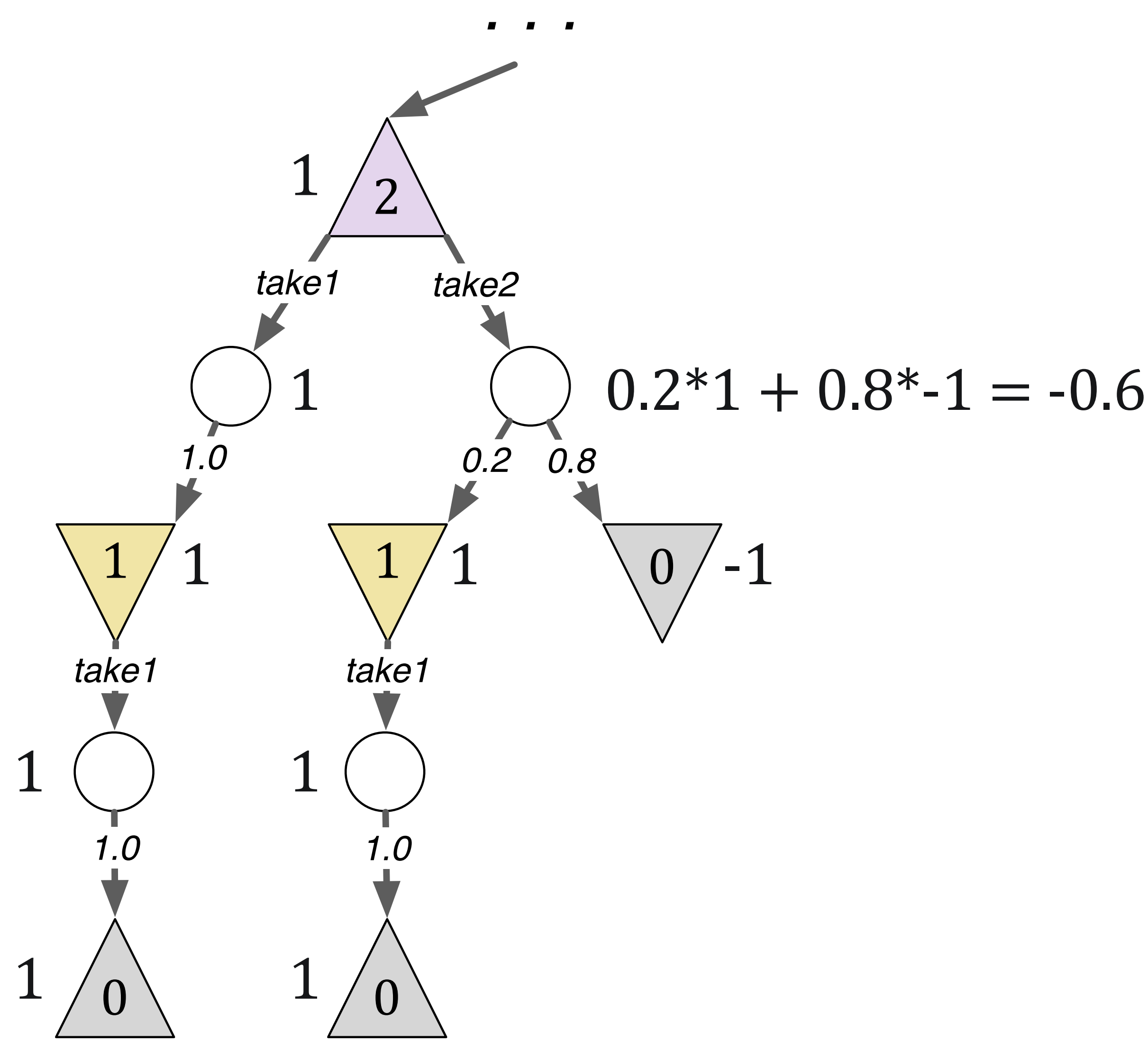
# Expectiminimax (Dropsy 4-Nim)

MAX

$$E(X) = \sum_i p_i x_i$$

MIN

$$E(X) = \sum_i p_i x_i$$



# What’s Next?

For Monday:

- Constraint Satisfaction Problems
- Read Russell & Norvig (AIMA)
  Sections 6.1-6.5

THIS SLIDE INTENTIONALLY LEFT BLANK