# CS440/ECE448 Lecture 22: Expectiminimax

Mark Hasegawa-Johnson, 3/2024



A contemporary backgammon set. Public domain photo by Manuel Hegner, 2013, https://commons.wikimedia.org/w/index.php?curid=25006945

# Outline

- Expectiminimax: Minimax + randomness
- Relationship of expectiminimax to MDP

# Stochastic games

How can we incorporate dice throwing into the game tree?

# Minimax

State evolves deterministically (when a player acts, that action uniquely determines the following state).

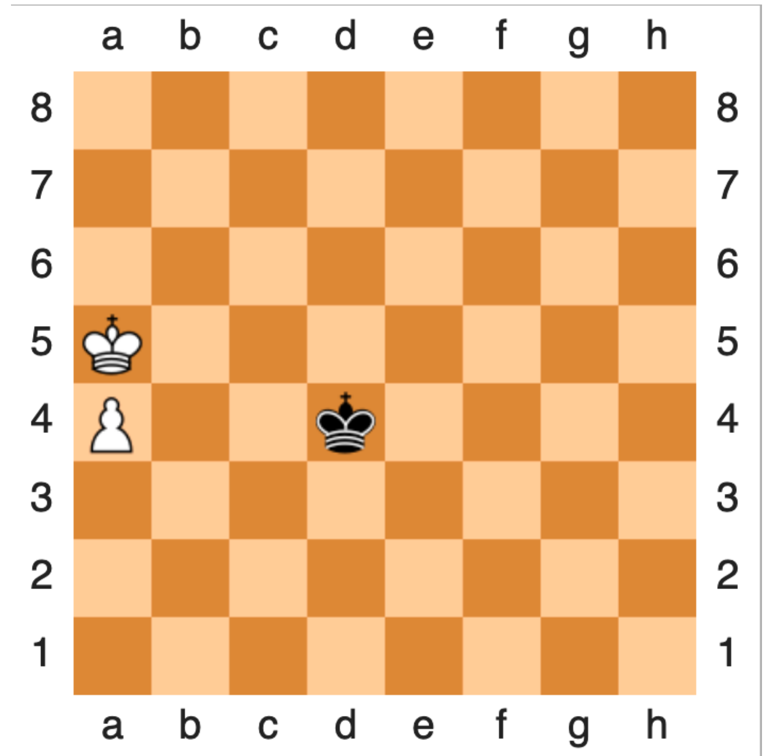
Current state is visible to both players.

Each player tries to maximize his or her own reward:

- **Maximize** (over all possible moves I can make) the
- **Minimum** (over all possible moves Min can make) of the resulting utility:

$$u(s) = \max_{s' \in C(s)} u(s')$$

$$u(s') = \min_{s'' \in C(s')} u(s'')$$



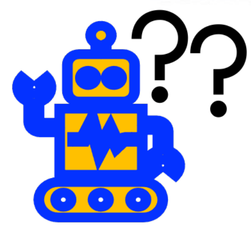
# Expectiminimax

State evolves **stochastically** (when a player acts, the game changes RANDOMLY, with a probability distribution $P(S_{t+1} = s'|S_t = s, a)$ that depends on the action, $a$).

The player tries to maximize his or her own reward:

- **Maximize** (over all possible moves I can make) the
- **Expected value** (over all possible successor states) of the resulting utility:

$$\sum_{s'} P(S_{t+1} = s'|S_t = s, a)u(s')$$



# Expectiminimax

State evolves **<u>stochastically</u>** (when a player acts, that action influences the state transition probability).

Current state is visible to both players.

Each player tries to maximize his or her own reward:

- **Maximize** (over all possible moves I can make) the
- **Minimum** (over all possible moves Min can make) of the
- **Expected value** (over all possible successor states) of the resulting utility:

$$u(s) = \max_a \sum_{s'} P(S_{t+1} = s' | S_t = s, a) u(s')$$

$$u(s') = \min_{a'} \sum_{s''} P(S_{t+2} = s'' | S_{t+1} = s', a') u(s'')$$

# Expectiminimax: notation

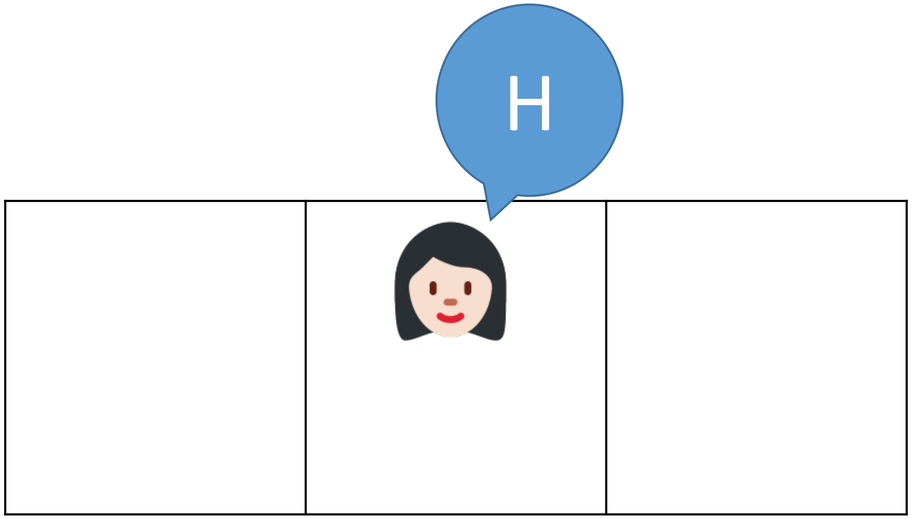
= MAX node. $u(s) = \max_{a \in A(s)} q(s,a)$

= MIN node. $u(s) = \min_{a \in A(s)} q(s,a)$

= Chance node. $q(s,a) = \sum_{s'} P(s'|s,a)u(s')$

# Expectiminimax example

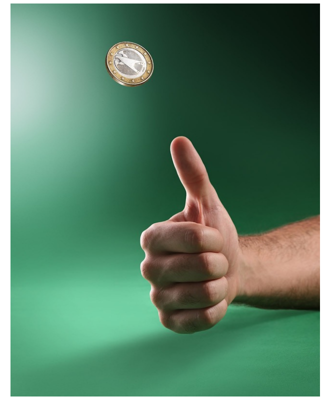
- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.



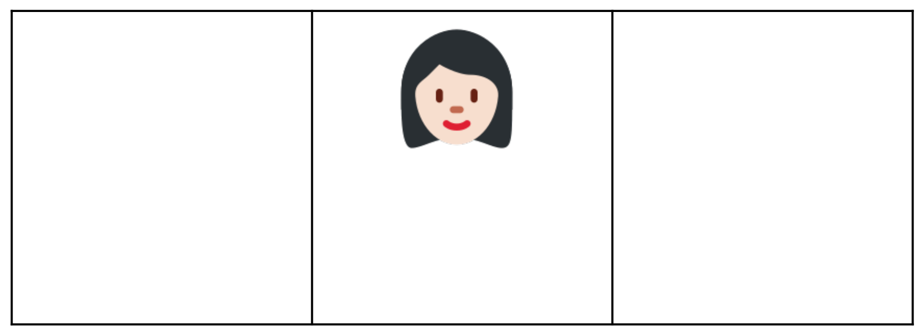

Emojis by Twitter, CC BY 4.0,
https://commons.wikimedia.org/w/index.php?curid=59974366

# Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.



By ICMA Photos - Coin Toss, CC BY-SA 2.0, https://commons.wikimedia.org/w/index.php?curid=71147286

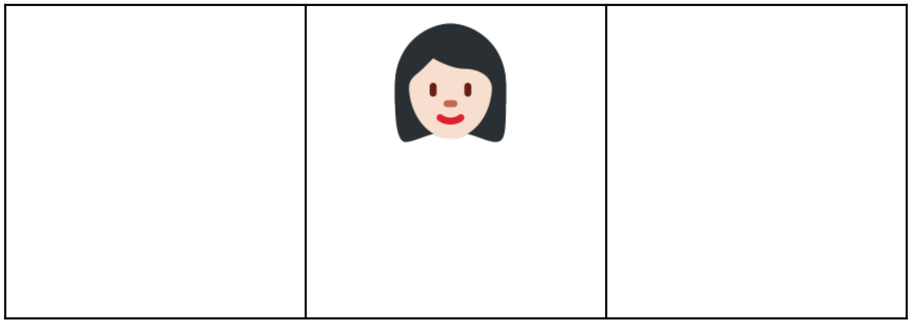


Emojis by Twitter, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=59974366

# Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.

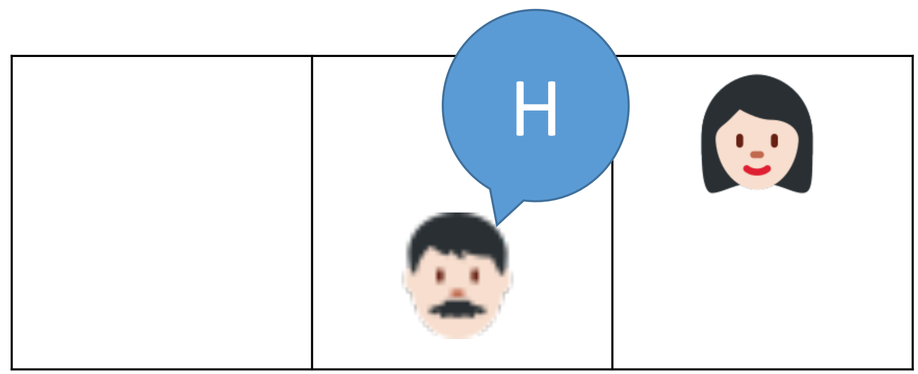


# Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.
- MAX: Max decides whether to count heads (action H) or tails (action T) as a forward movement.



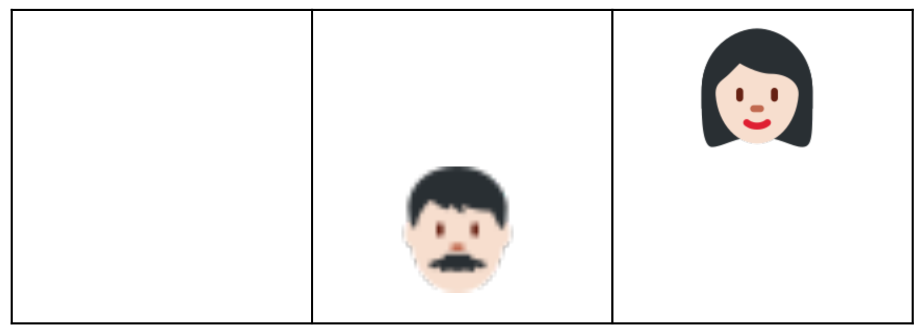
# Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.
- MAX: Max decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: he flips a coin and moves his game piece in the direction indicated.









# Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.
- MAX: Max decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: he flips a coin and moves his game piece in the direction indicated.

Reward: $2 to the winner, $0 for a draw.



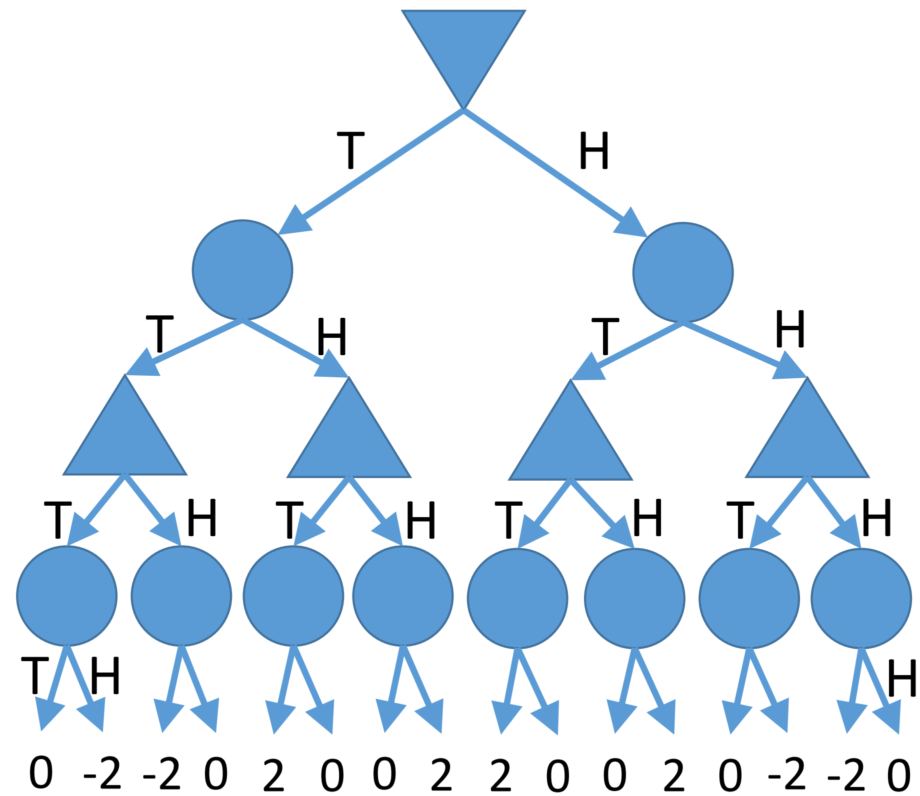




# Expectiminimax example

- MIN: Min decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: she flips a coin and moves her game piece in the direction indicated.
- MAX: Max decides whether to count heads (action H) or tails (action T) as a forward movement.
- Chance: he flips a coin and moves his game piece in the direction indicated.

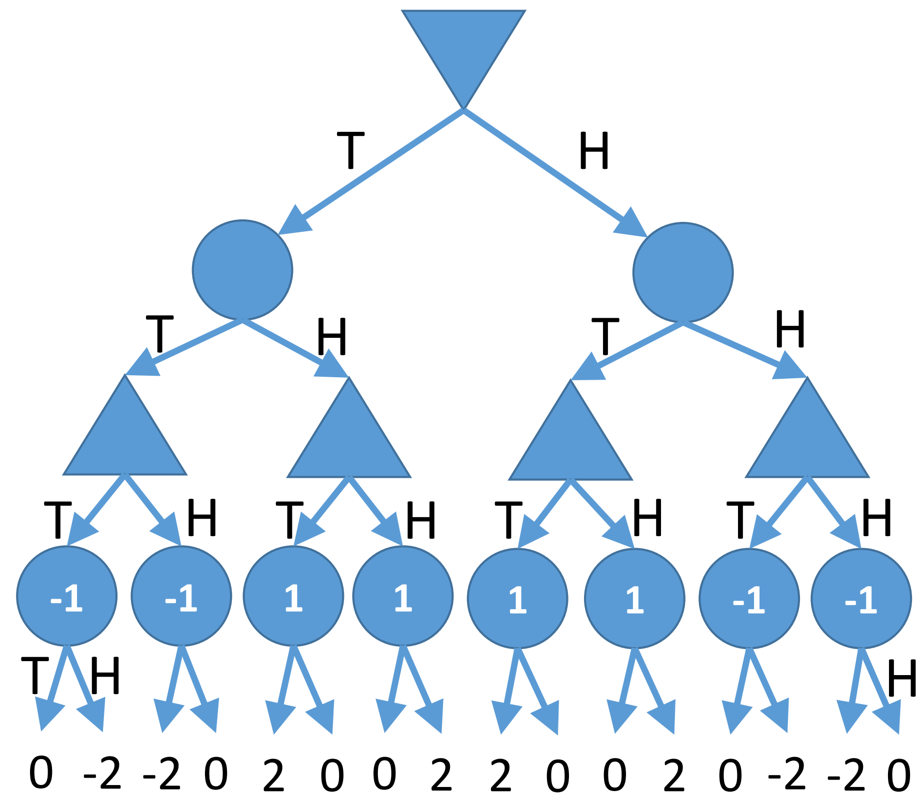
Reward: $2 to the winner, $0 for a draw.



# Expectiminimax example

Chance node:

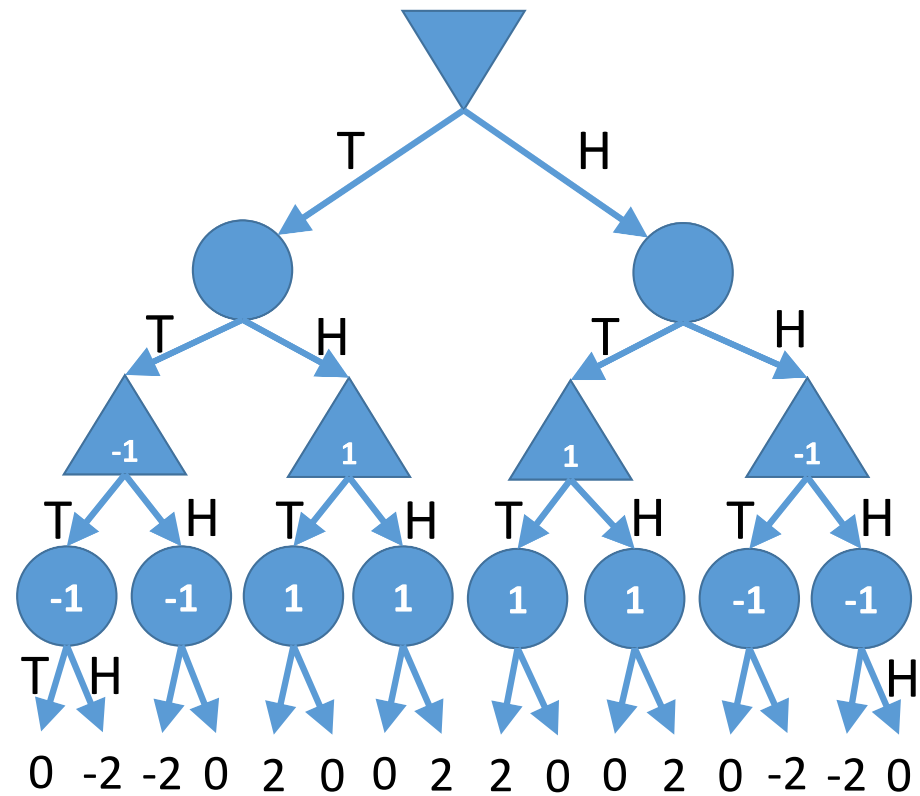
$$q(s,a) = \sum_{s'} P(s'|s,a)u(s')$$



# Expectiminimax example

Max node:

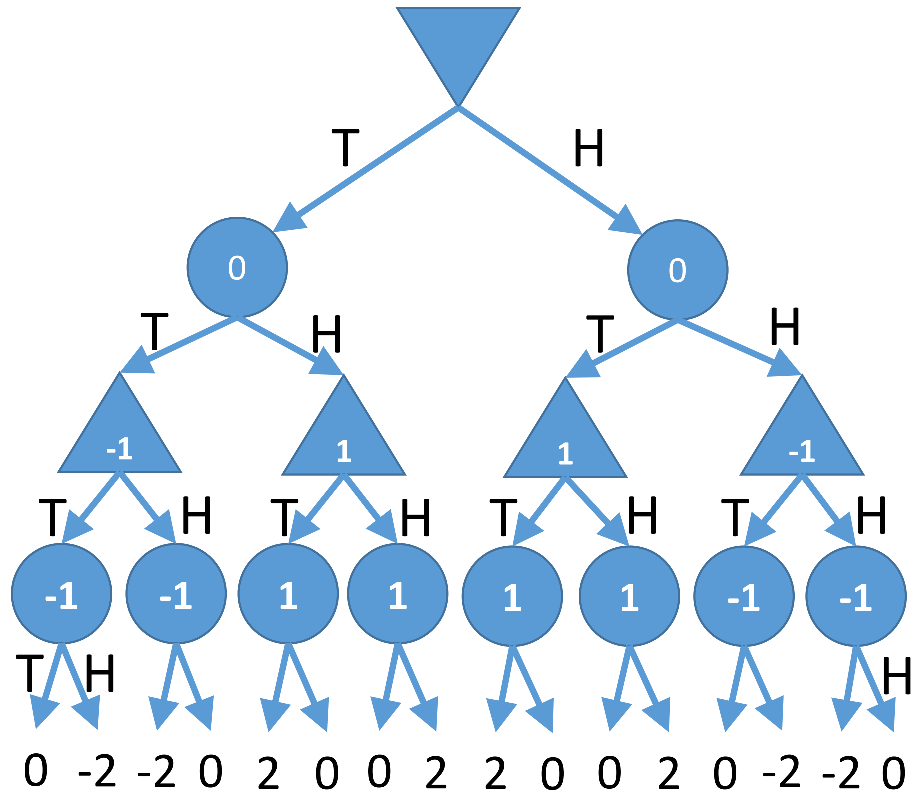
$$u(s) = \max_{a \in A(s)} q(s,a)$$



# Expectiminimax example

Chance node:

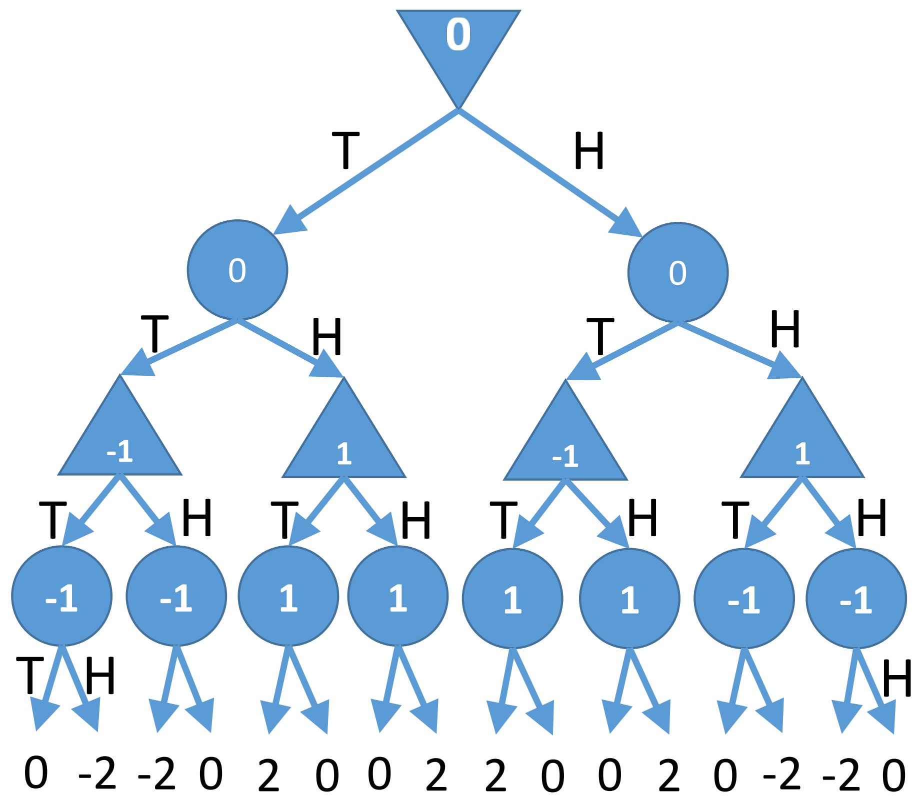
$$q(s,a) = \sum_{s'} P(s'|s,a)u(s')$$



# Expectiminimax example

Min node:

$$u(s) = \min_{a \in A(s)} q(s, a)$$



# Try the quiz!

https://us.prairielearn.com/pl/course_instance/147925/assessment/2404166

# Outline

- Expectiminimax: Minimax + randomness
- Relationship of expectiminimax to MDP

# Relationship of expectiminimax to MDP

Remember that the solution to a Markov decision process is the policy that maximizes the Bellman equation, which we could call the expectimax equation:

$$u(s) = \max_a \sum_{s'} P(S_{t+1} = s' | S_t = s, a) u(s')$$

Expectiminimax maximizes a generalized Bellman equation:

$$u(s) = \begin{cases} \max_a \sum_{s'} P(S_{t+1} = s' | S_t = s, a) u(s') & s \text{ is a max state} \\ \min_a \sum_{s'} P(S_{t+1} = s' | S_t = s, a) u(s') & s \text{ is a min state} \end{cases}$$

These two are fundamentally the same operation, and should require the same computations.

# Computational complexity of MDP

- Value iteration:

$$u_i(s) = r(s) + \gamma \max_a \sum_{s'} P(S_{t+1} = s' | S_t = s, a) u_{i-1}(s')$$

  - $\mathcal{O}\{nb\}$ computations per iteration, where:
    - $n =$ number of states,
    - $b =$ branching factor = (number of actions) $\times$ (number of successor states)
  - Number of iterations may be infinite; $i^{\text{th}}$ iteration explores paths of length up to $i$

- Policy iteration:

$$u_i(s) = r(s) + \gamma \sum_{s'} P(s' | s, \pi_i(s)) u_i(s')$$

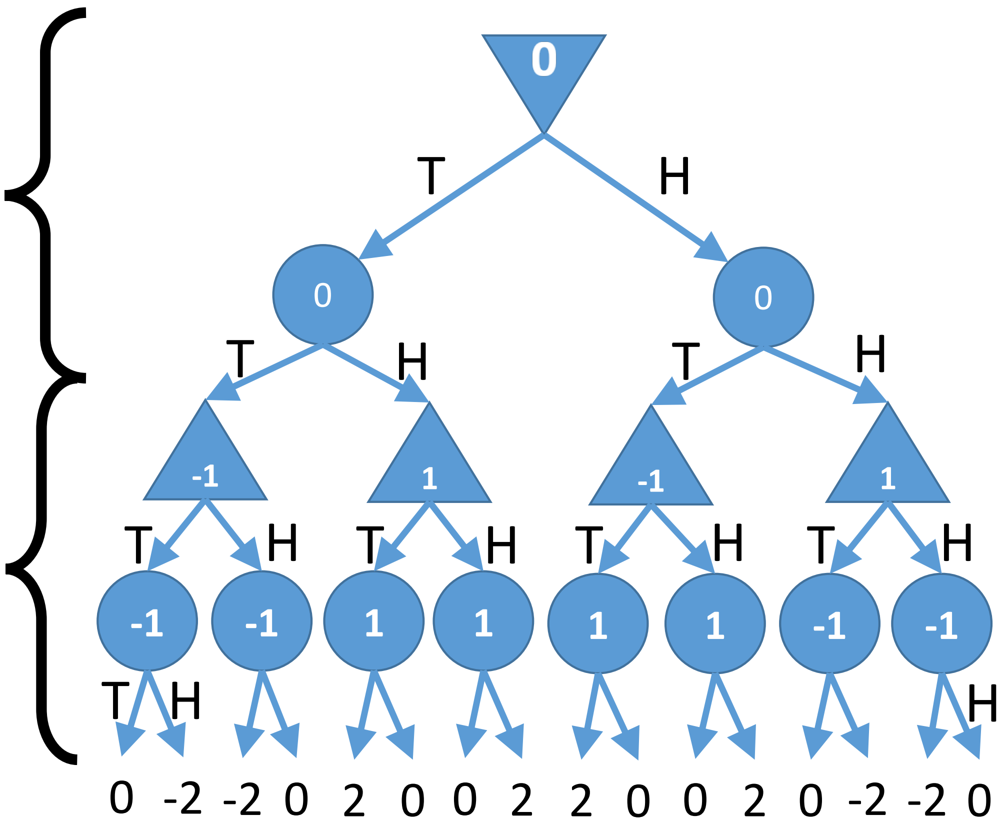
$$\pi_{i+1}(s) = \operatorname{argmax}_a \sum_{s'} P(S_{t+1} = s' | S_t = s, a) u_i(s')$$

  - $\mathcal{O}\{nb\}$ computations per iteration
  - Number of iterations guaranteed to be less than or equal to the number of possible policies, which is $\mathcal{O}\{b^n\}$

# Computational complexity of expectiminimax

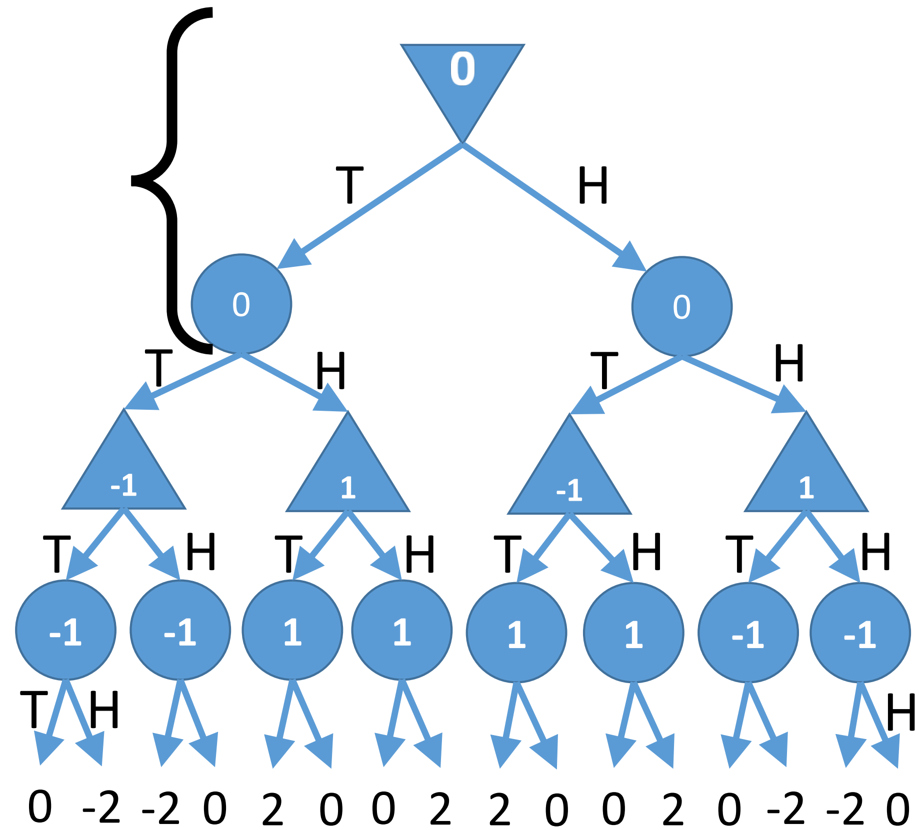
$$u_2(s) = r(s) + \gamma \min_a \sum_{s'} P(S_{t+1} = s' | S_t = s, a) u_1(s')$$

$$u_1(s) = r(s) + \gamma \max_a \sum_{s'} P(S_{t+1} = s' | S_t = s, a) u_0(s')$$



# Computational complexity of expectiminimax

- $u_d(s) = r(s) + \gamma \min_a \sum_{s'} P(S_{t+1} = s' | S_t = s, a) u_{d-1}(s')$
- Effectively, expectiminimax is finding the solution of a $d$-step value iteration: considering only paths that are up to $d$ moves long.
- Time complexity = $\mathcal{O}\{nbd\}$, where
  - $n =$ # states (assuming, as in an MDP, that loops are possible, so every state appears in every level of the tree)
  - $b =$ branching factor = (number of actions) $\times$ (number of successor states)
  - $d =$ depth of the tree



# Summary

Bellman equation = expectimax:

$$u(s) = \max_a \sum_{s'} P(S_{t+1} = s' | S_t = s, a) u(s')$$

Expectiminimax:

$$u(s) = \begin{cases} \max_a \sum_{s'} P(S_{t+1} = s' | S_t = s, a) u(s') & s \text{ is a max state} \\ \min_a \sum_{s'} P(S_{t+1} = s' | S_t = s, a) u(s') & s \text{ is a min state} \end{cases}$$