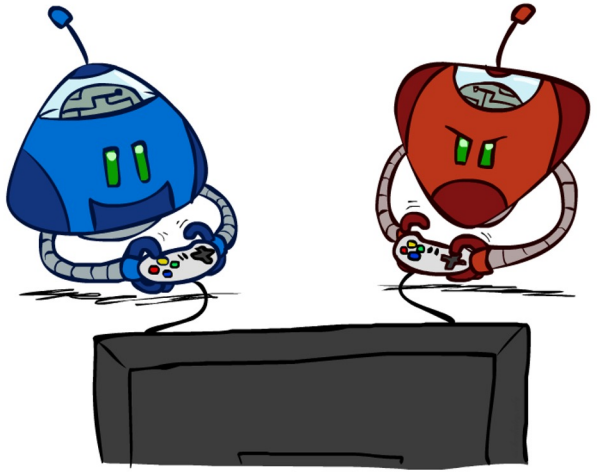
Adversarial Search

CSE 473: Introduction to Artificial Intelligence



James Weichert (jpw@cs)
cs.uw.edu/473
acknowledgements

Welcome!
Is cereal a soup?
sli.do **#cse473**

Administrivia

ANNOUNCEMENTS

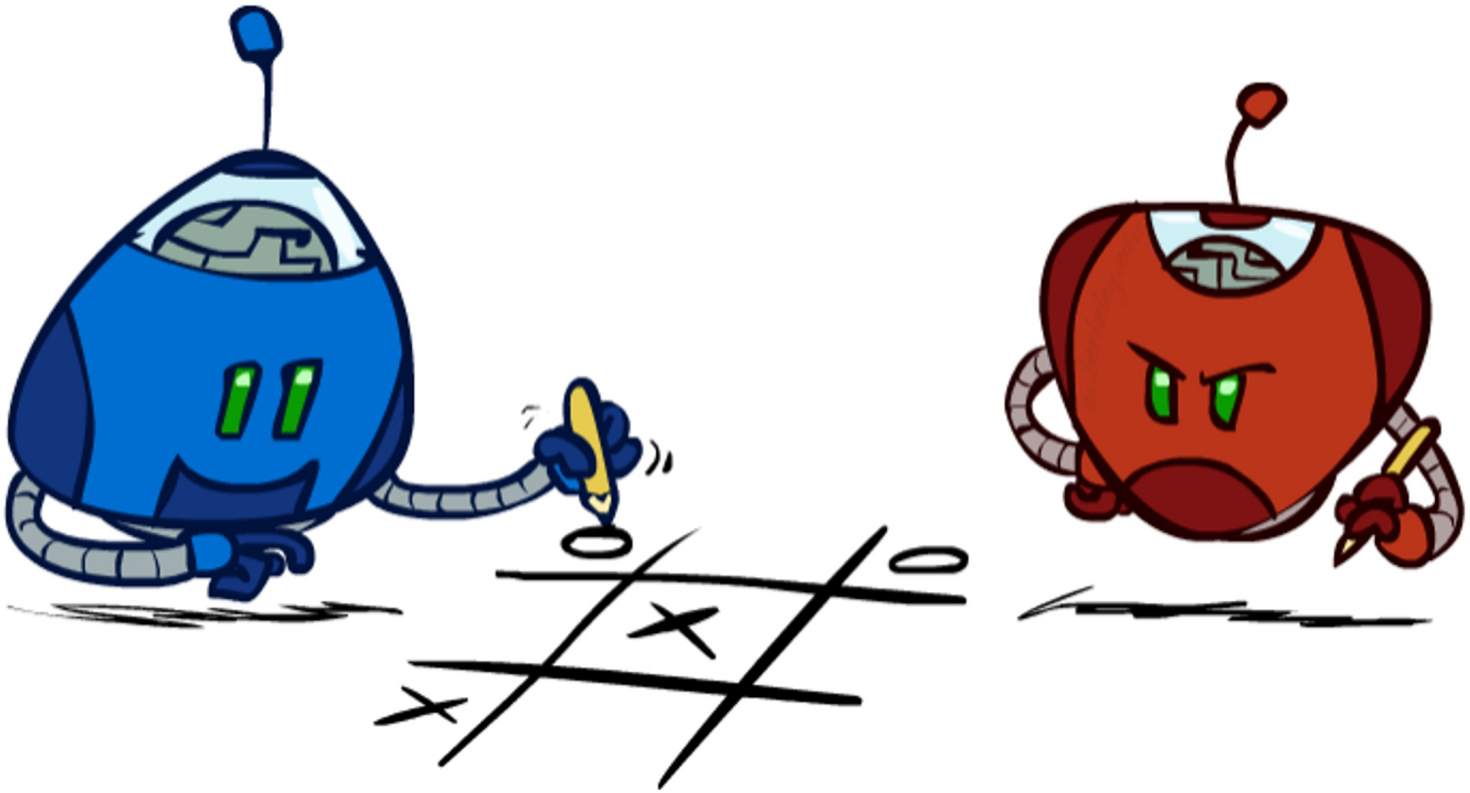
- For each project, make sure to fill out the **AI usage reflection** (cs.uw.edu/473/projects)
- **Test 1** next Friday
- No class this Friday (holiday)

ASSIGNMENTS

- **Homework 1 due** Friday (7.3)
- **Project 1 due** in next Thursday (7.9)

Games

FOUNDATIONS

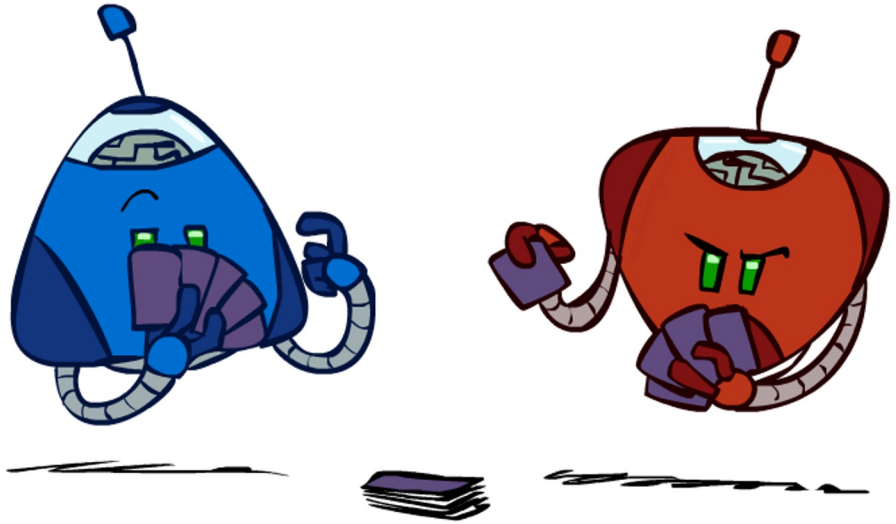


Types of Games

FOUNDATIONS

Games model environments with more than one agent

- Characteristics
  - deterministic vs stochastic?
  - fully observable vs. partially observable?
  - number of players?
  - turn-taking vs. simultaneous?
  - zero sum?
- *How to plan around opponent's moves?*

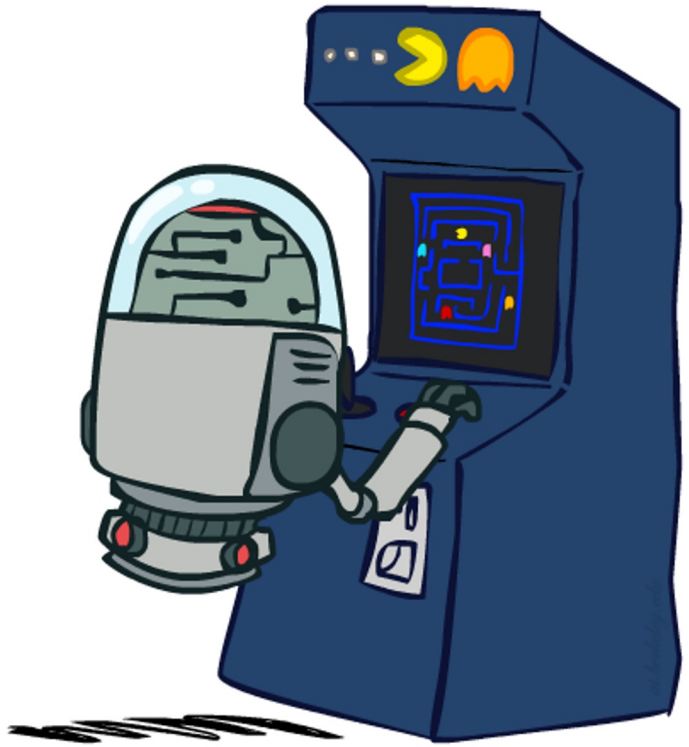


"Standard" Games

DEFINITION

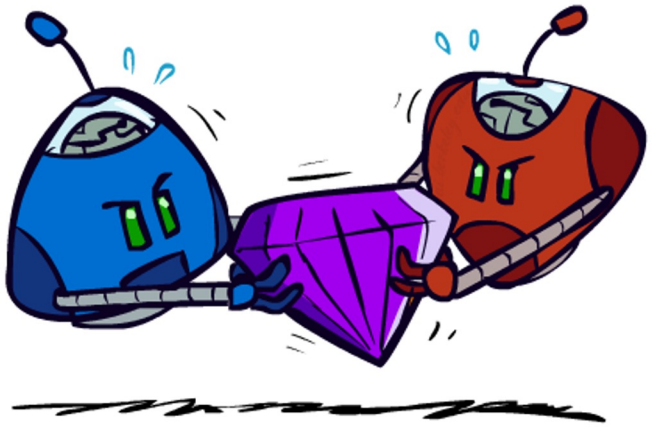
Deterministic, observable, two-player, turn-taking, and zero sum games

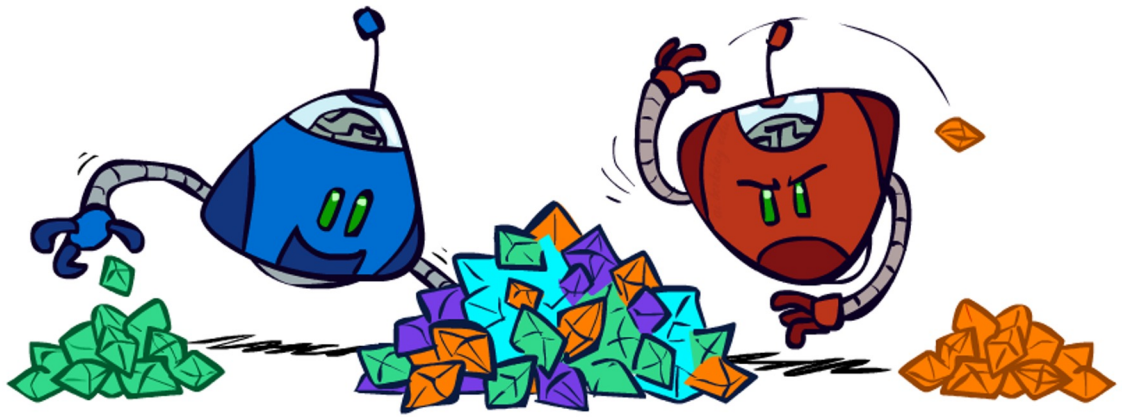
- Game Formulation
  - Initial state: s_0
  - Players: `player(s)` indicating turn
  - Actions: `actions(s)` for player on move
  - Transition model: `T(s, a)`
  - Terminal (goal) test: `terminal-test(s)`
  - Terminal values: `utility(s, p)`



Zero-Sum Games

FOUNDATIONS





Zero-Sum Games

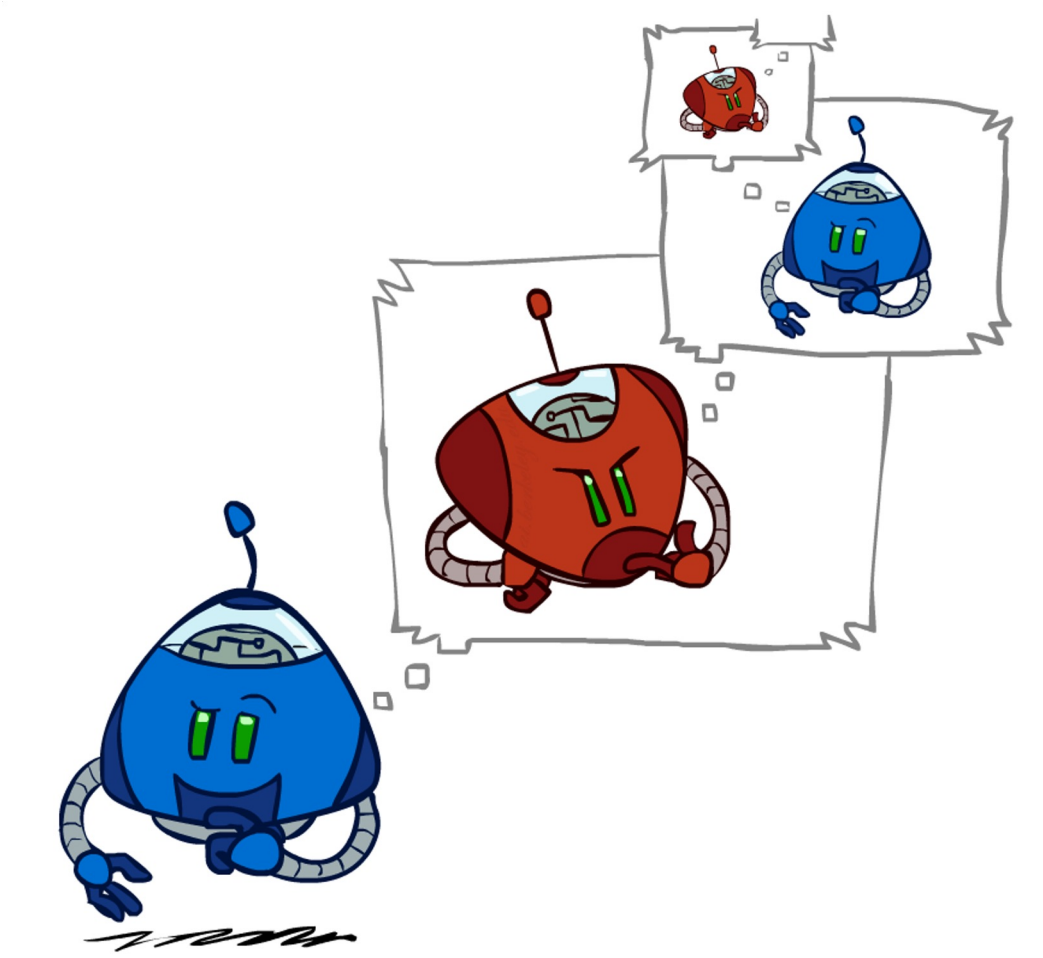
- Agents have *opposite* utilities
- One agent **maximizes**, the other **minimizes**

General Games

- Agents have *independent* utilities
- Cooperation, indifference, competition, etc. are all possible

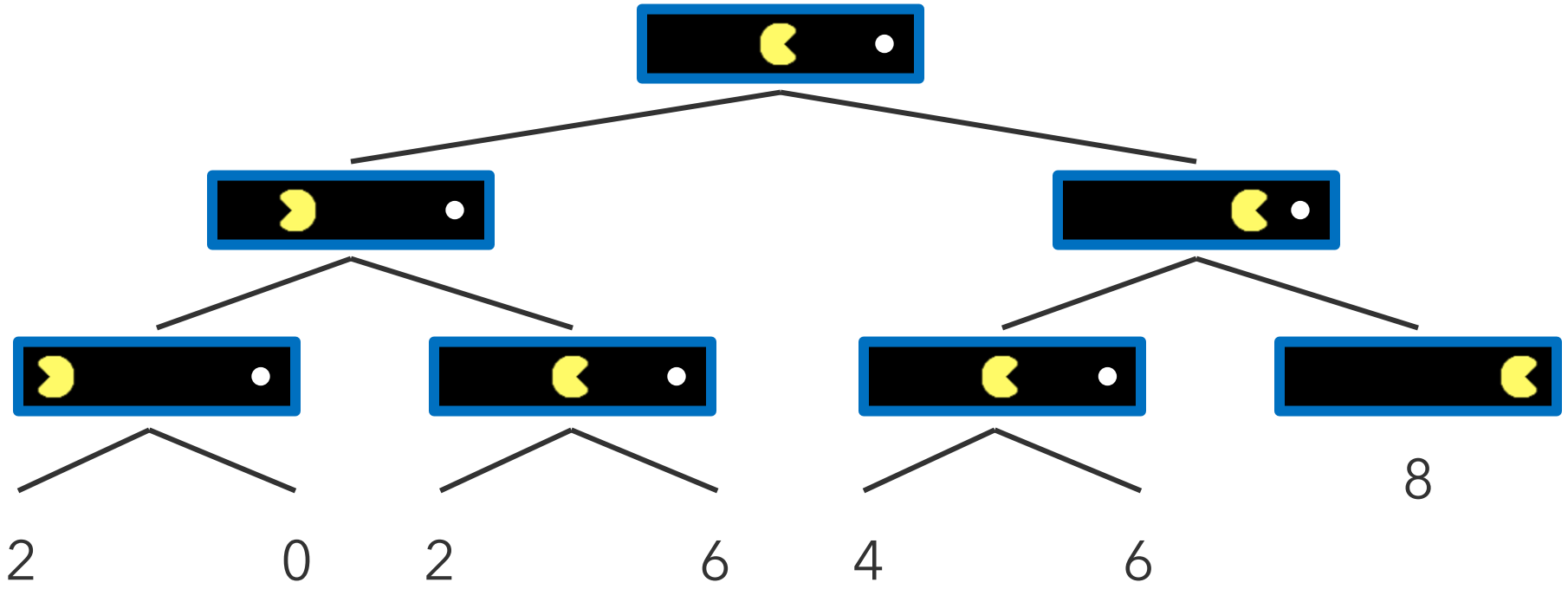
Adversarial Thinking

FOUNDATIONS



Single-Agent Trees

FOUNDATIONS

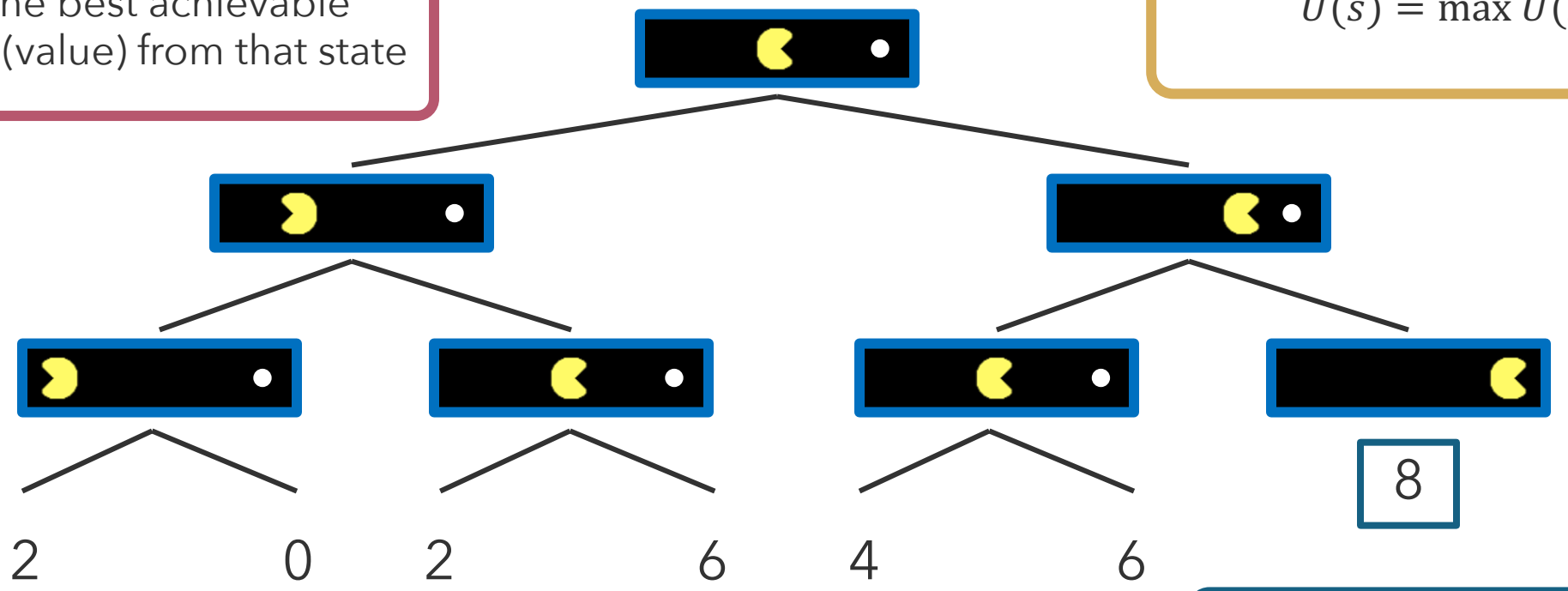


Utility

FOUNDATIONS

Utility is the best achievable outcome (value) from that state

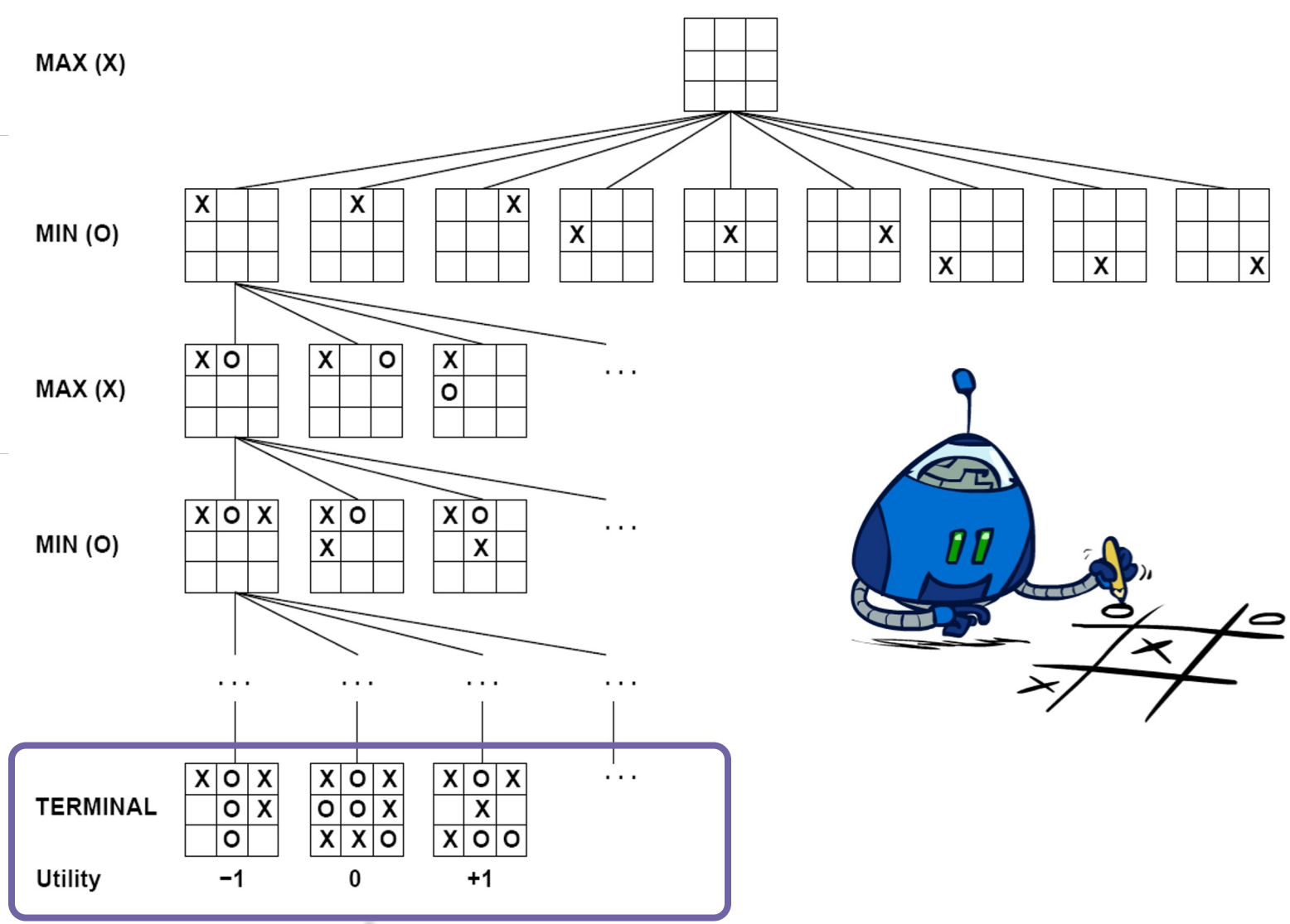
Non-Terminal States:
$U(s) = \max U(s')$



Terminal States:
$U(s)$ is known

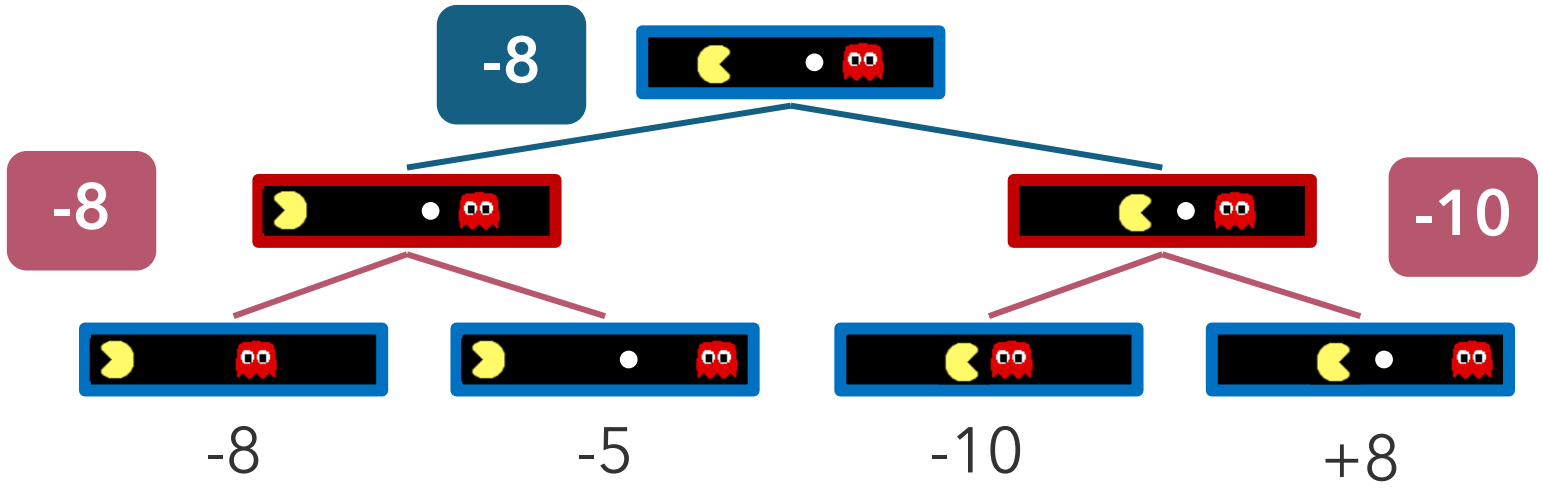
Tic-Tac-Toe

EXAMPLE



Minimax Values

FOUNDATIONS



MAX Nodes:

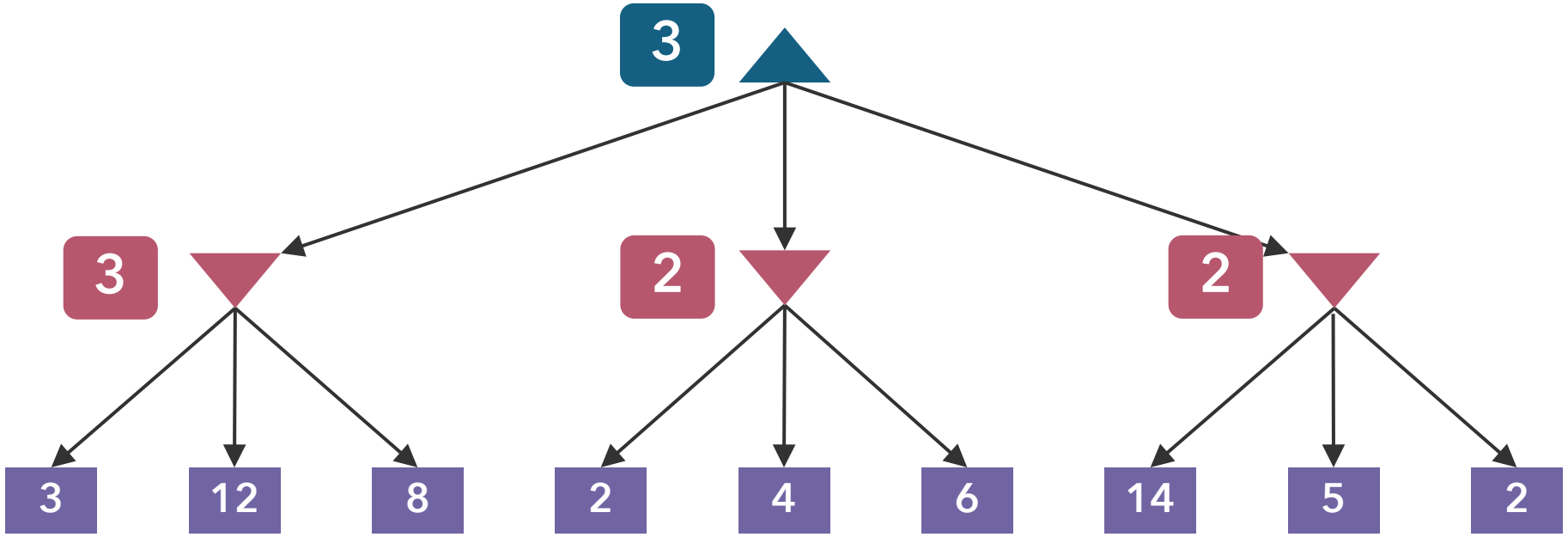
$$U(s) = \max_{s' \in \text{successors}(s)} U(s')$$

MIN Nodes:

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

Minimax Tree

EXAMPLE



Minimax

ALGORITHM

```
function MINIMAX(s) returns action
  return action a in actions(s) with highest minimax-values(T(s,a))
```

```
function MINIMAX-VALUES(s) returns value
  if terminal-test(s) then return utility(s)
  if player(s) == MAX then return max_{a in actions(s)} minimax-values(T(s,a))
  if player(s) == MIN then return min_{a in actions(s)} minimax-values(T(s,a))
```

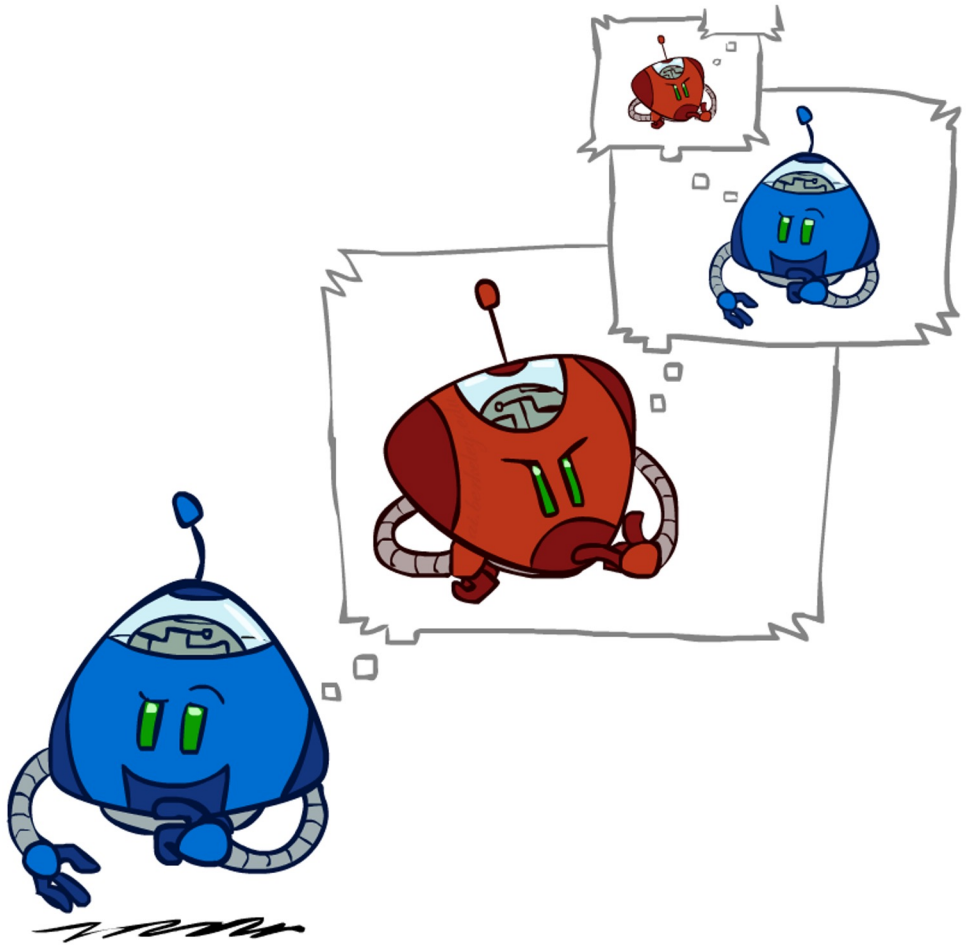
Questions?

live and on sli.do #cse473

Is It Efficient?

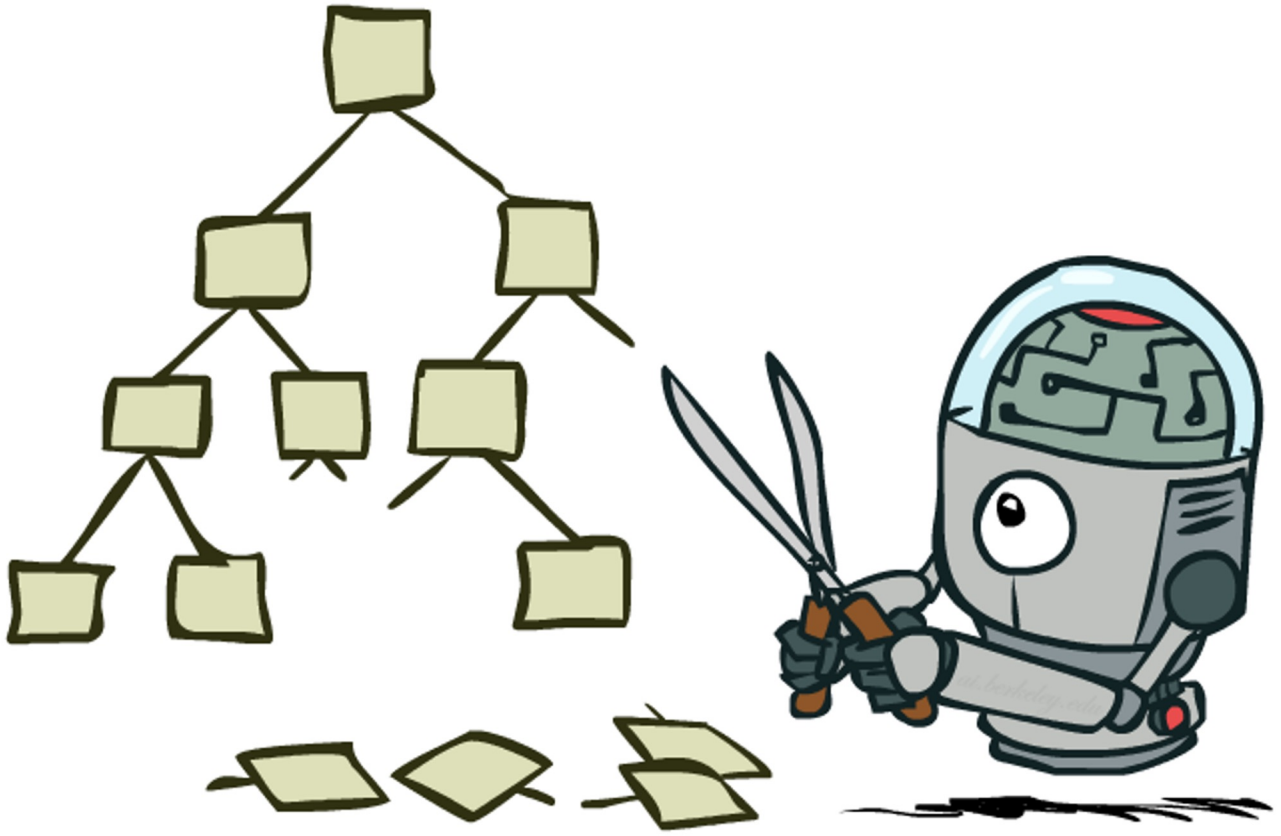
FOUNDATIONS

- How efficient is minimax?
  - Operates just like (exhaustive) DFS
  - Time Complexity: ***O(b^m)***
  - Space Complexity: ***O(bm)***
- Example: Chess
  - b ≈ 35, m ≈ 100
  - Completely infeasible
  - *Is this how humans play chess?*



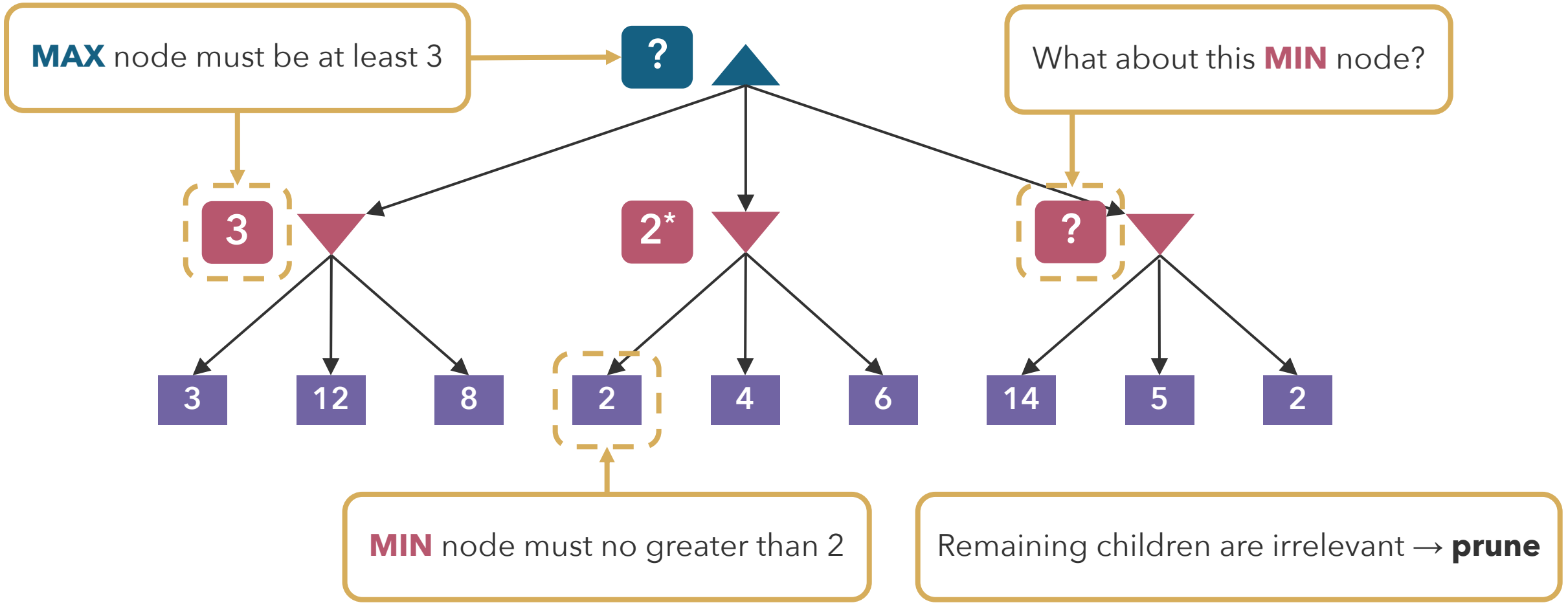
Avoiding Unnecessary Work

CONTEXT



Pruning

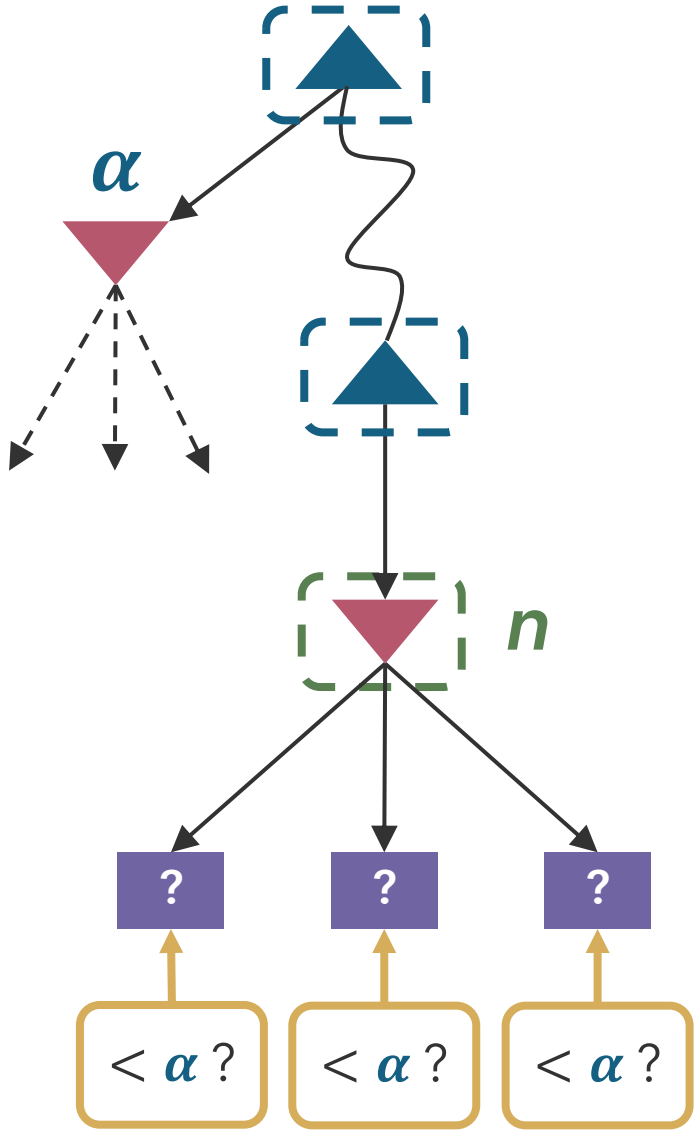
EXAMPLE



Alpha-Beta Pruning

FOUNDATIONS

- Idea
  - Avoid extra work by ***pruning*** branches when we know their utilities won't affect the decision
- General Configuration (**MIN** version)
  - To compute `min-value` at node ***n***, loop over ***n***'s children
  - ***n***'s value is relevant for **MAX** node above
  - Let α be best value **MAX** can get at any point along current path from root
  - If ***n*** becomes worse than α, **MAX** will avoid it, so we can stop considering ***n***'s children
- Version for **MAX** is symmetric (with β)



Alpha-Beta Implementation

ALGORITHM

```
function MINIMAX-DECISION(s) returns action
  return action a in actions(s) with highest max-value(T(s,a),-∞,+∞)
```

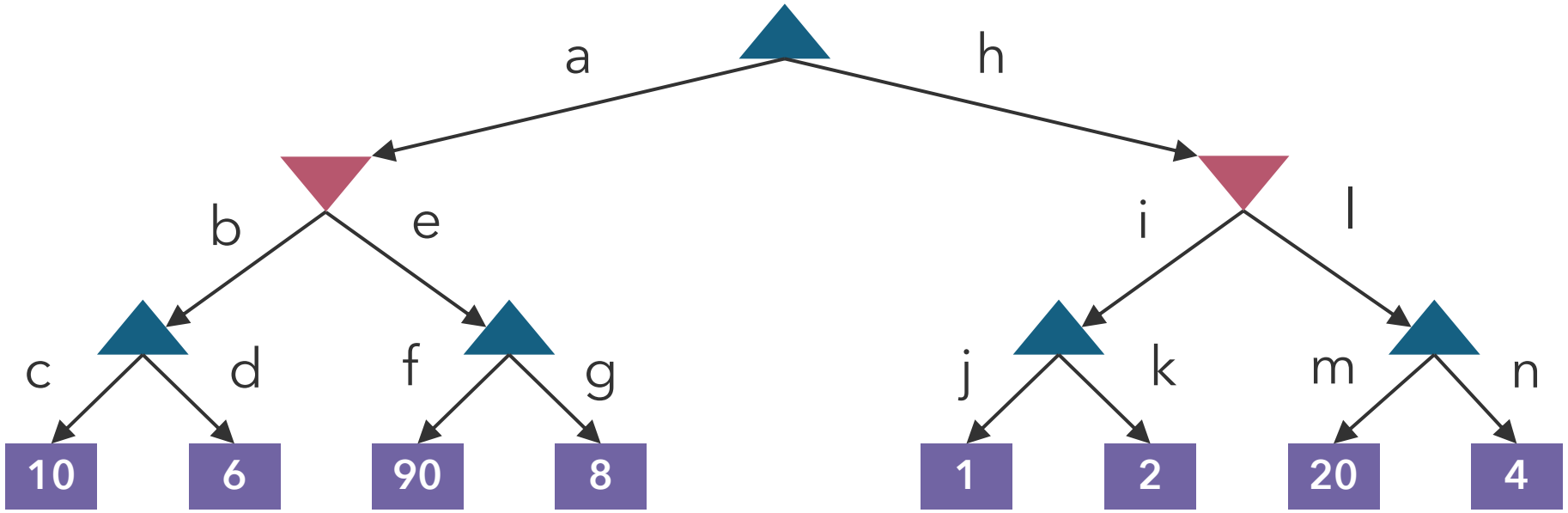
```
function MAX-VALUE(s, α, β) returns value
  v ← -∞
  for each s' in successors(s)
        v = max(v, min-value(s', α, β))
        if v >= β
            return v
         α = max(α,v)
  return v
```

```
function MIN-VALUE(s, α, β) returns value
  v ← +∞
  for each s' in successors(s)
        v = min(v, max-value(s', α, β))
        if v <= α
            return v
         β = max(β,v)
  return v
```

Pruning Example

EXAMPLE

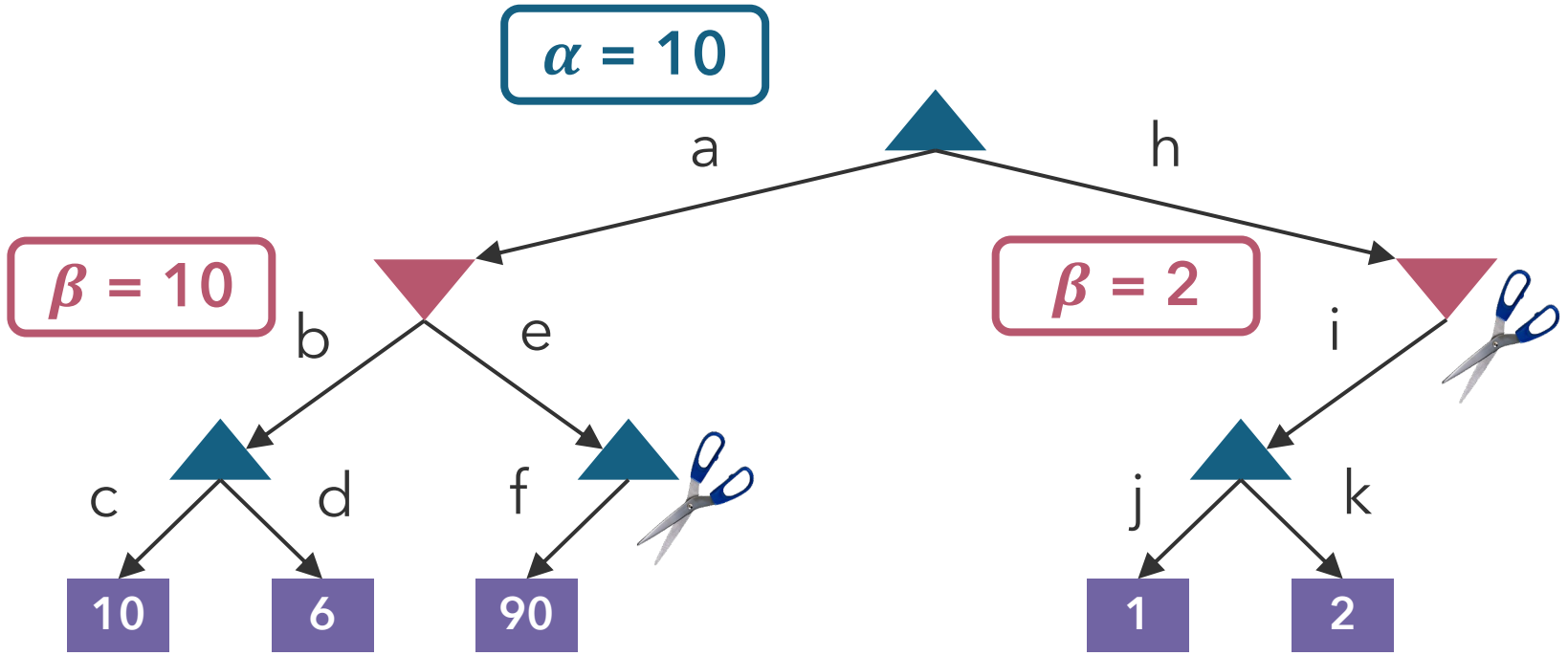
Which nodes get pruned? What are the α and β values at the end?



Pruning Example (Solution)

EXAMPLE

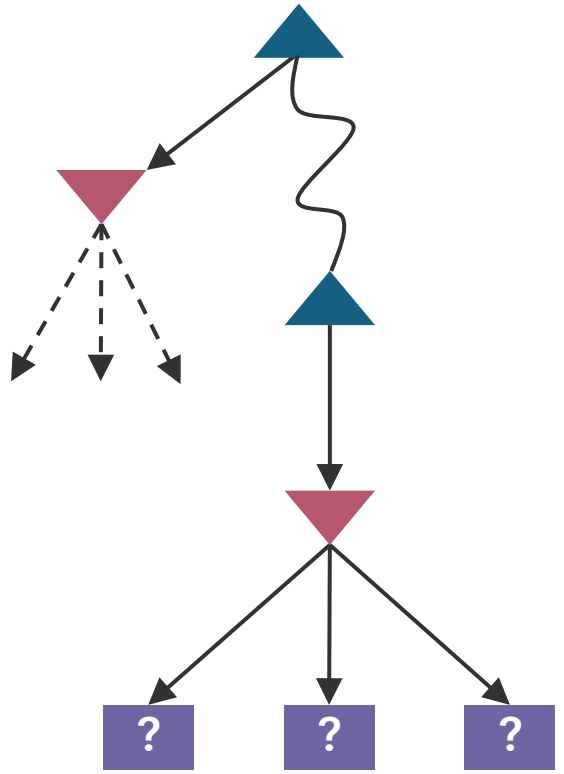
Which nodes get pruned? What are the α and β values?



Properties of Alpha-Beta Pruning

DEFINITION

- **Theorem**: Alpha-Beta pruning has *no effect* on computed minimax value
- Good child ordering improves efficiency
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles the solvable depth!
- Example of **metareasoning**
  - Reasoning about reasoning
- For chess
  - Only 35^{50} instead of 35^{100}

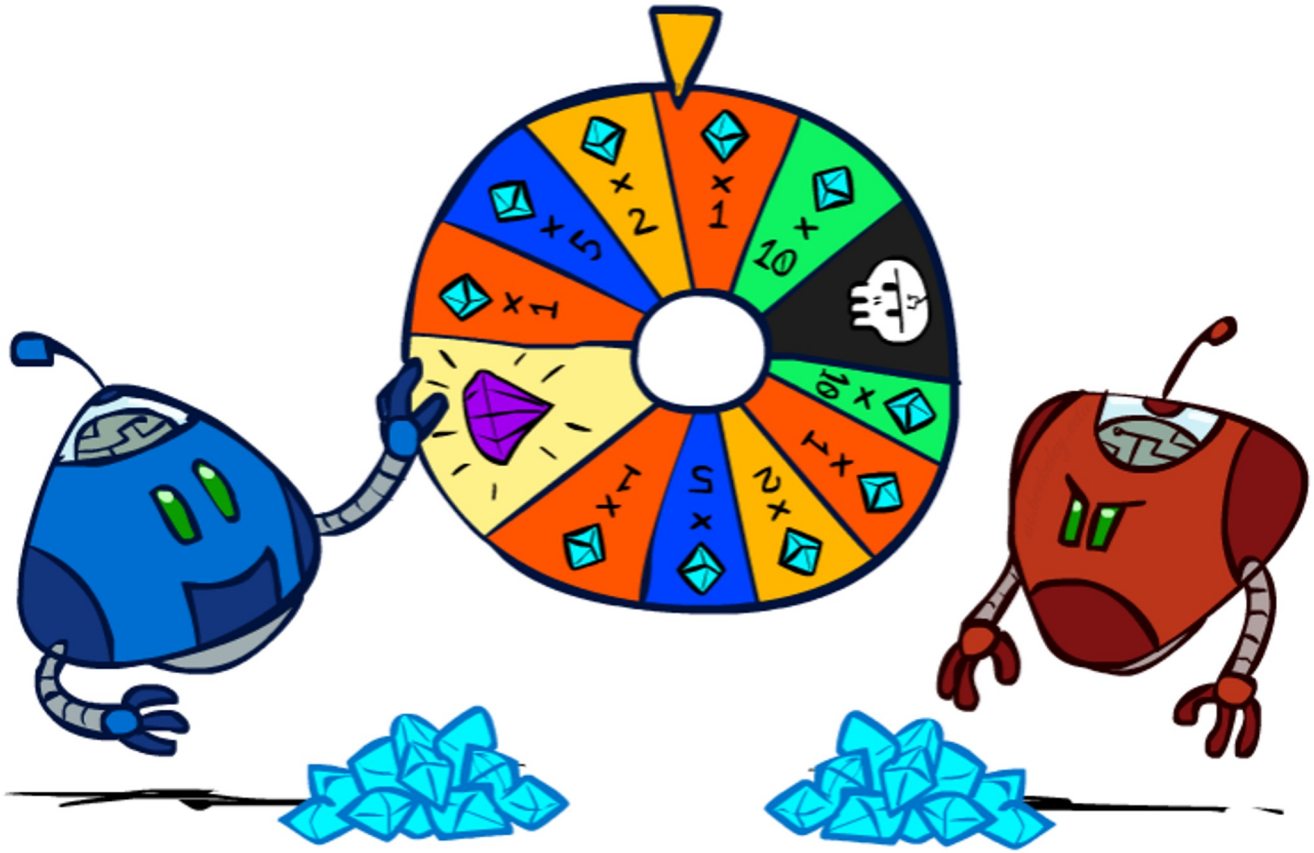


Questions?

live and on sli.do #cse473

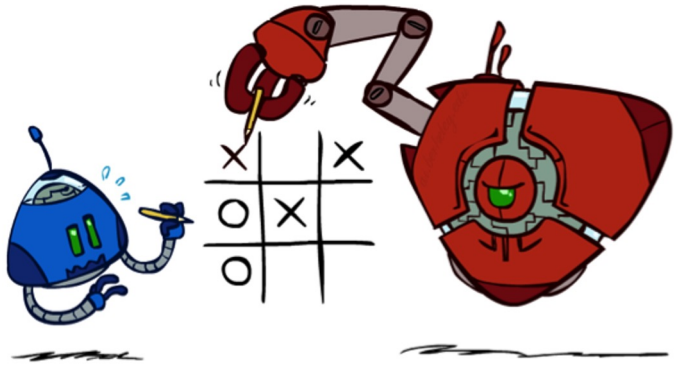
Games with Uncertainty

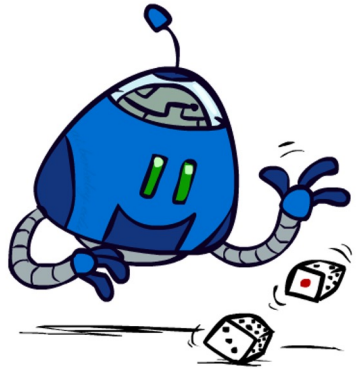
CONTEXT

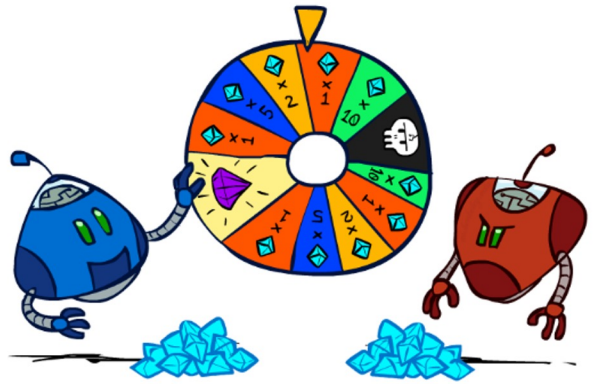


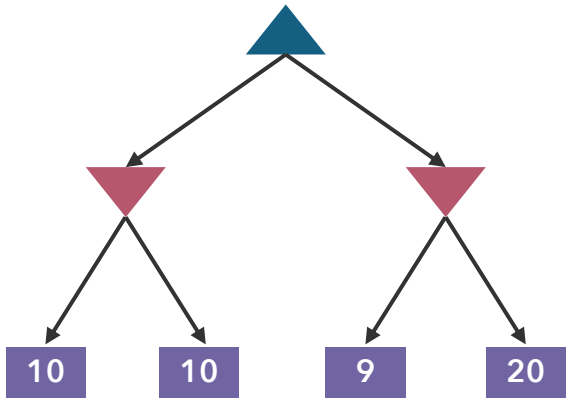
Chance Outcomes in Trees

EXAMPLE

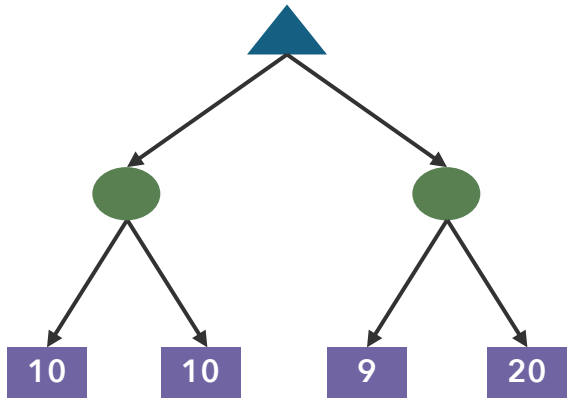




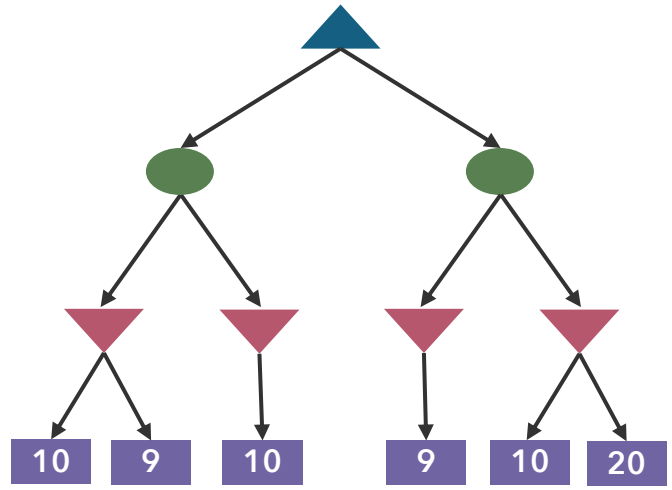




Minimax



Expectimax



Expectiminimax

Expectiminimax

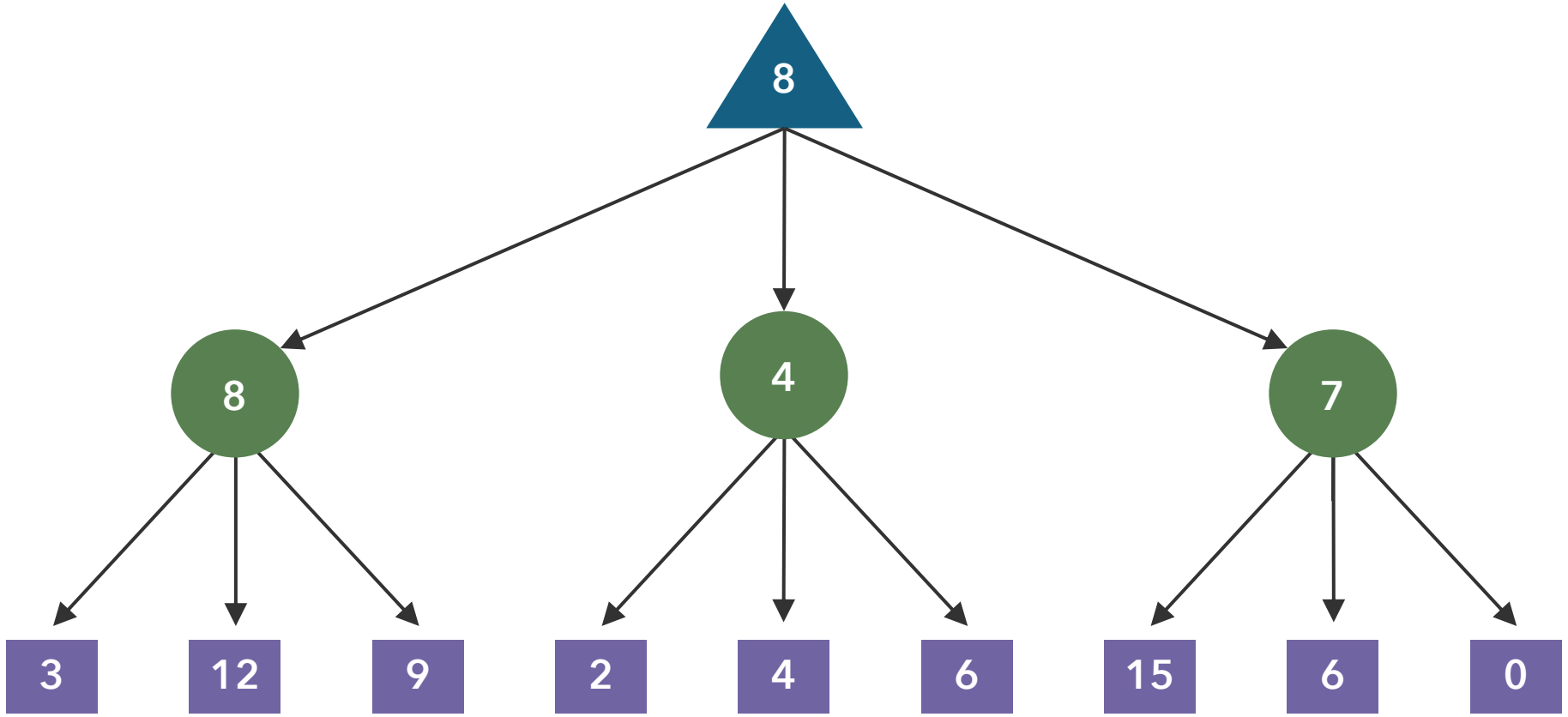
ALGORITHM

```
function DECISION(s) returns action
  return action a in actions(s) with highest value(T(s,a))
```

```
function VALUE(s) returns value
  if terminal-test(s) then return utility(s)
  if player(s) == MAX then return max a in actions(s) value(T(s,a))
  if player(s) == MIN then return min a in actions(s) value(T(s,a))
  if player(s) == CHANCE then return sum a in actions(s) P(a) * value(T(s,a))
```

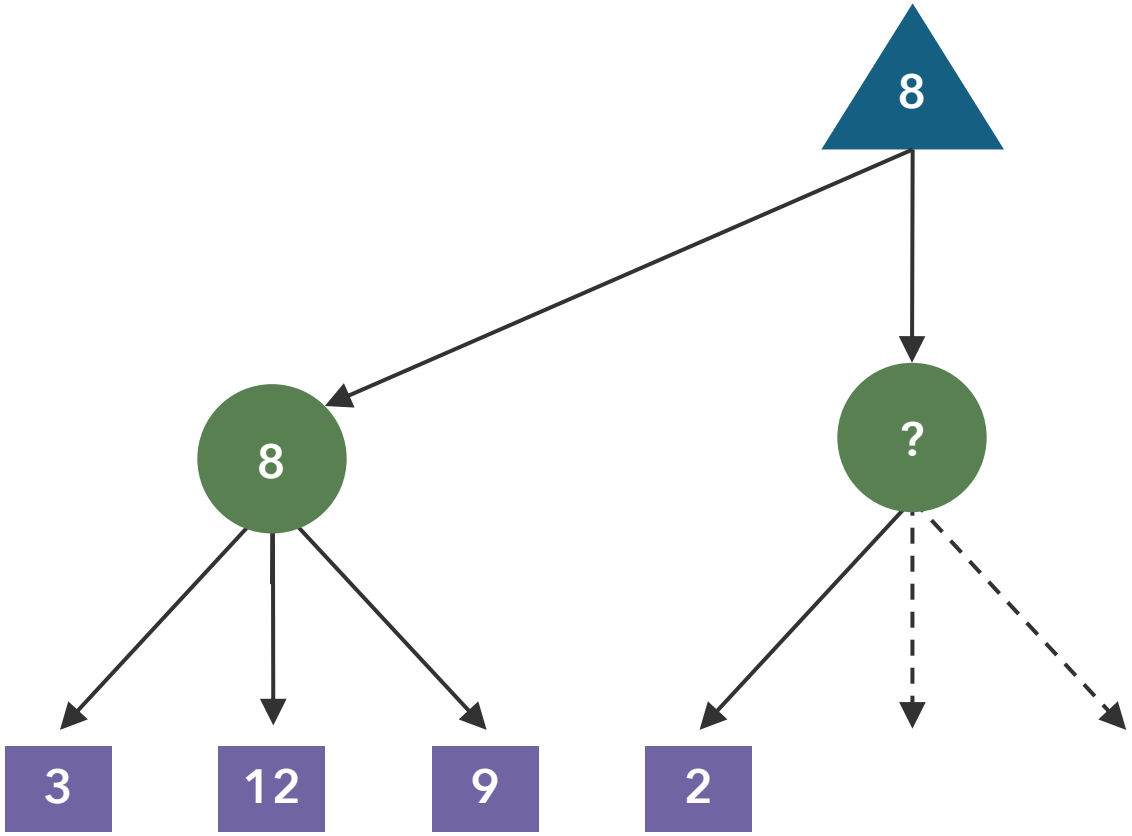
Expectimax

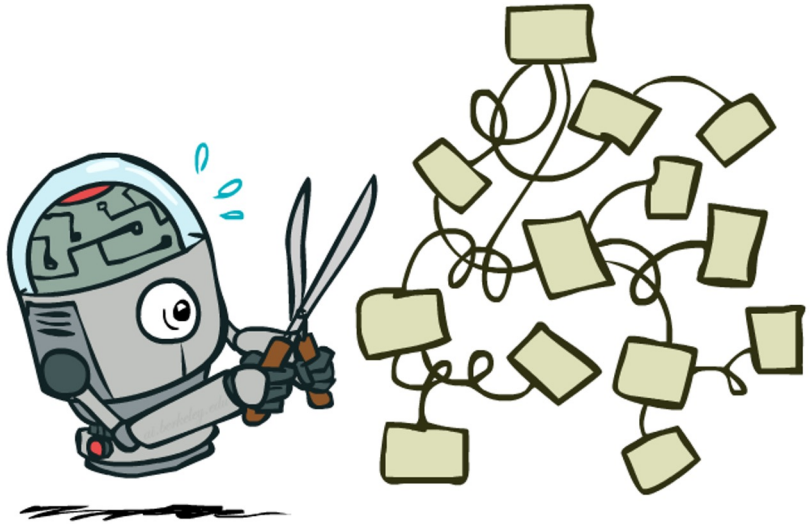
FOUNDATIONS



Expectimax Pruning?

EXAMPLE





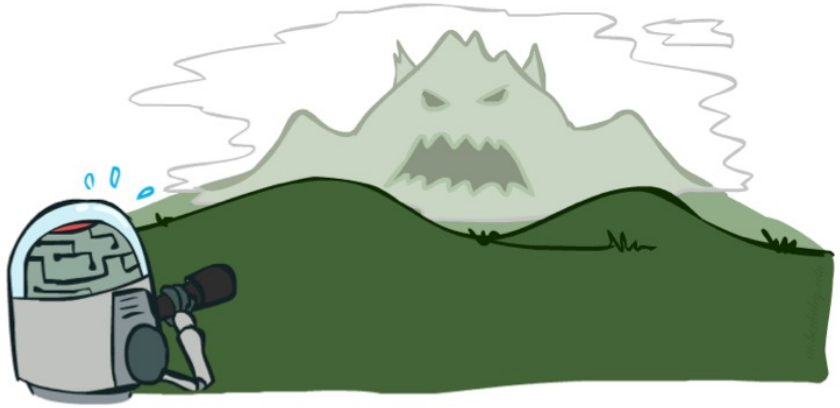
Resource Limits

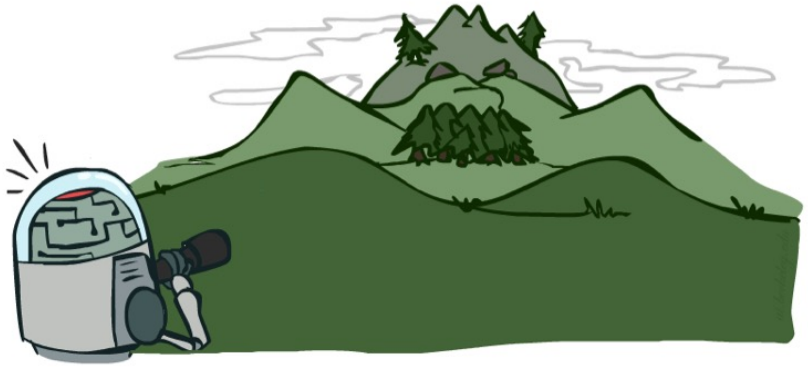
CONTEXT

Real-World Constraints

FOUNDATIONS

- **Problem**: In realistic games, searching to leaves is infeasible
- Solution: **Bounded Lookahead**
  - Search only to a preset **depth limit** (**horizon**)
  - Use an **evaluation function** for non-terminal utilities
  - Guarantee of optimality is gone
  - Marginal increase in depth makes a big difference!
- Tradeoff
  - Deeper search (usually) leads to better play, but is considerably more costly
  - Complexity of features vs. complexity of computation





Questions?

live and on sli.do #cse473

that's it for today!

SUMMARY

- Multi-Agent Search
- Adversarial search via minimax
- When transitions are deterministic and players play optimally, pruning is possible

UPCOMING

- Modeling Stochasticity
- Reinforcement Learning

REMINDERS

- Complete **Practice Problem 5**
- Do **Homework 1** (due 7.3)
- Do **Project 1** (due 7.9)