Artificial Intelligence: Other Search Models

CPSC 433: Artificial Intelligence
Fall 2022

Jonathan Hudson, Ph.D
Assistant Professor (Teaching)
Department of Computer Science
University of Calgary

Thursday, October 13, 2022

UNIVERSITY OF CALGARY

Other Search Models and Processes

Problems with the models/processes so far:

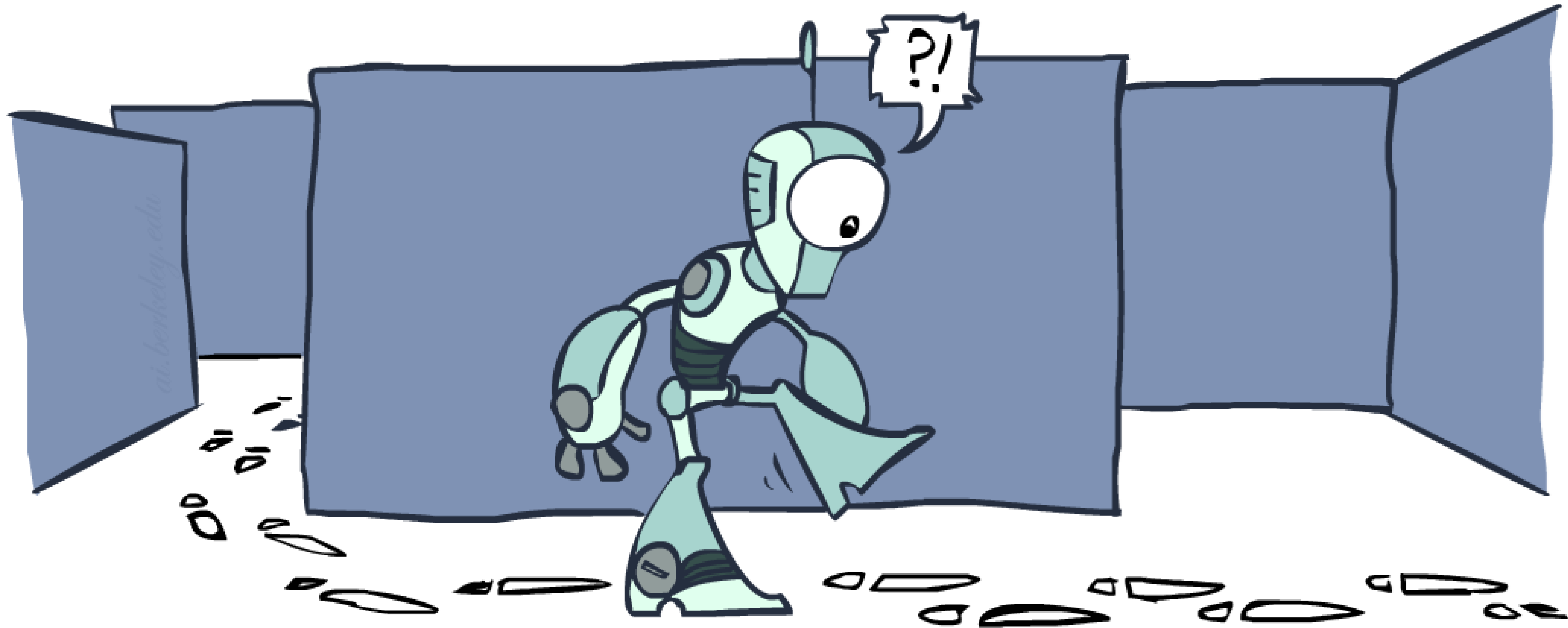
- What if our problem solution requires alternatives of problem divisions and we want this represented in the model?
- What about elements of `Prob` that appear repeatedly in a tree? Can we get rid of duplication and resulting redundancy?

UNIVERSITY OF CALGARY

Graph-Based Search

UNIVERSITY OF CALGARY

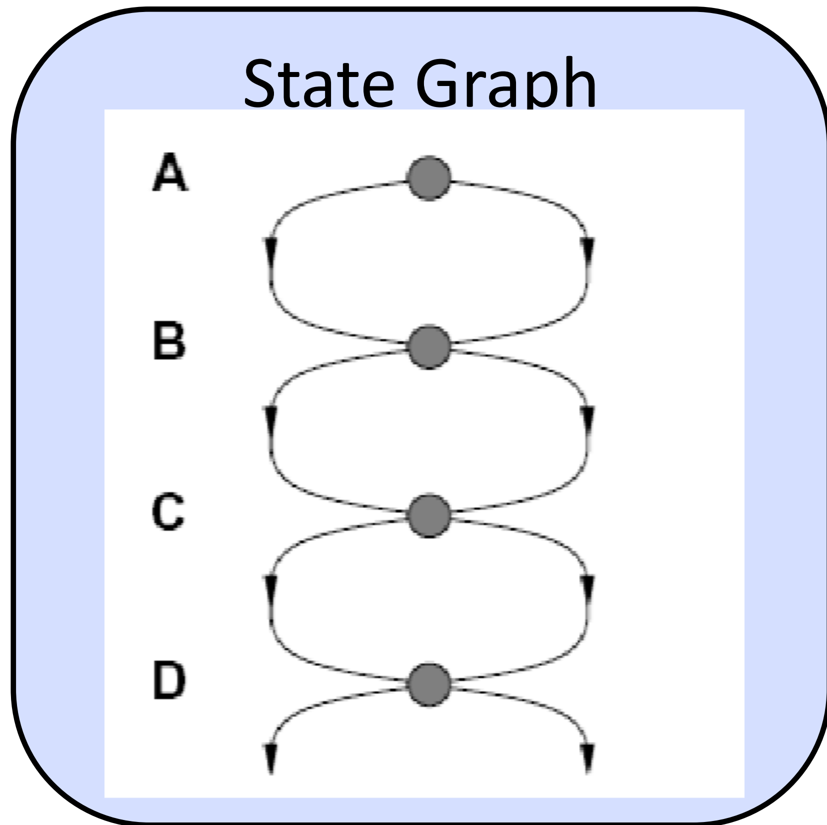
Graph Search

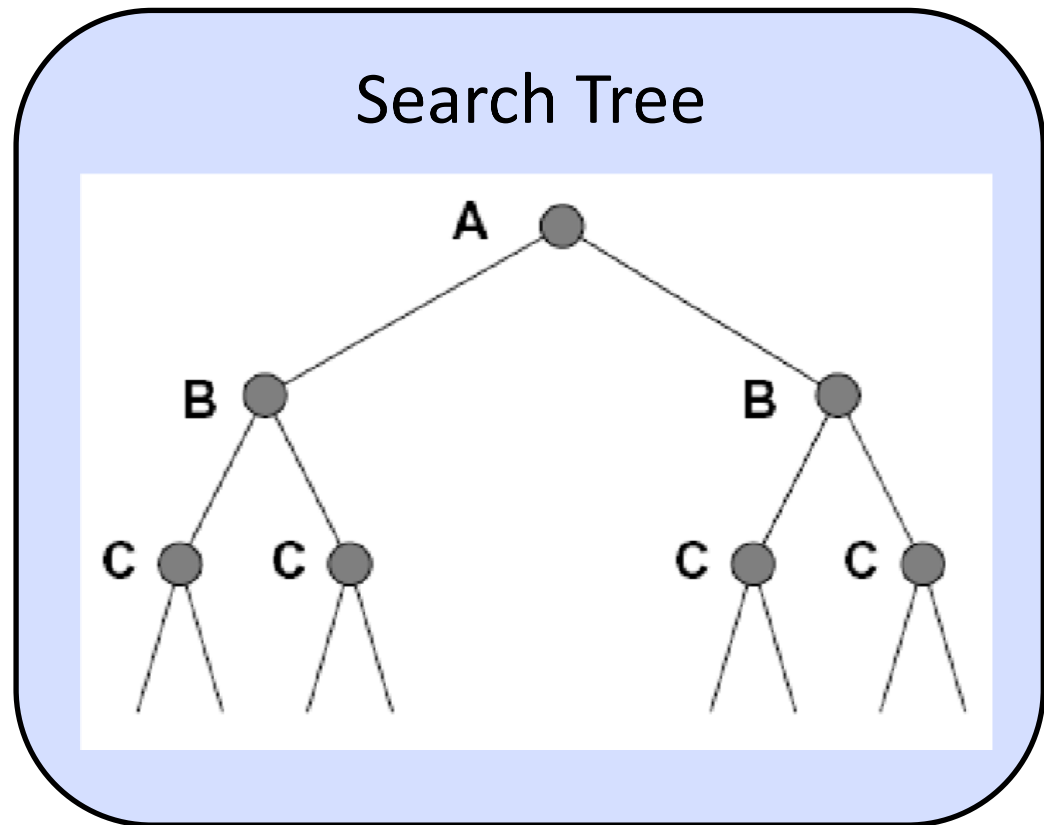


UNIVERSITY OF CALGARY

Tree Search: Extra Work!

- Failure to detect repeated states can cause exponentially more work.

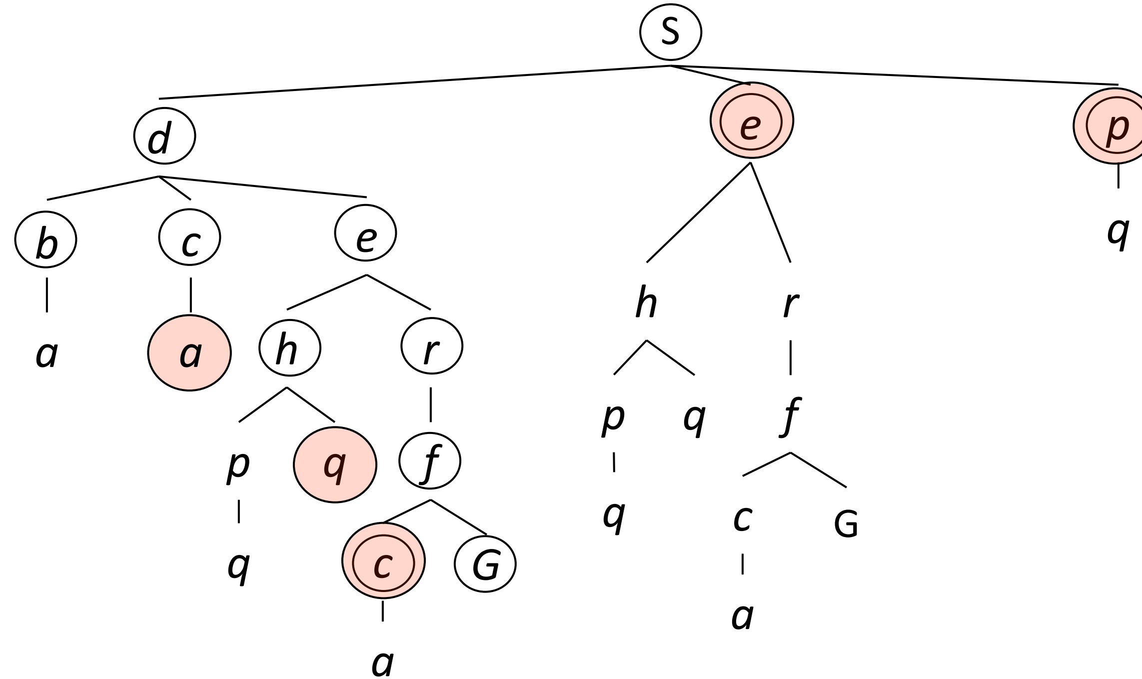




UNIVERSITY OF CALGARY

Graph Search

- In BFS, for example, we shouldn't bother expanding the pink circled nodes (why?)



UNIVERSITY OF CALGARY

Graph Search

- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states (“closed set”)
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set
- Important: store the closed set as a set, not a list
- Can graph search wreck completeness? Why/why not?
- How about optimality?

UNIVERSITY OF CALGARY

Graph-based search

- "Improvement" of tree-based search
- Achieves that every element of `Prob` occurs in only one node
- Graph described as set of nodes with set of arcs (directed connections)
- Transitions extend nodes that have no arcs going out
- Transitions as in trees, except that we check if a certain node is already there and if yes, we do not create it again, we just add an arc

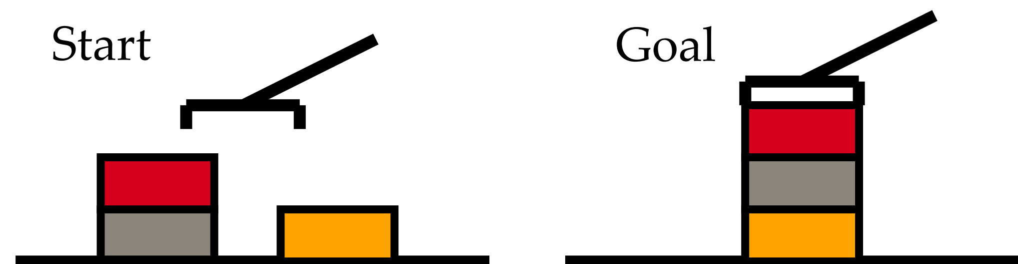
Blocks World

UNIVERSITY OF CALGARY

Example (Sketch): Planning in the Blocks World (I)

Problem: Given a set of blocks in certain relations to each other (situation), the same blocks in a different relation and a robot arm, determine a set of actions of the arm that transform the first situation into the second one

Idea: Do an or-tree-based search using the different possible actions of the robot arm as the alternatives



Example (Sketch): Planning in the Blocks World (II)

Observation: A lot of different action combinations lead to the same result (since for each action there is an action with exactly the opposite effect)

☞ a lot of problem descriptions occur in several nodes in the or-tree

☞ Switching from a tree to a graph avoids redundant work and takes a lot of pressure from the search control

Pros and Cons

- Less memory consumption
- No redundant work

- Some help (☞ overhead) necessary to quickly detect already represented elements of `Prob`
- Graphs are more difficult to debug

☞ Use only, if quite some duplication of nodes occurs in a tree

And-Or-Tree-based Search

UNIVERSITY OF CALGARY

And-or-tree-based search

- Combines and- and or-trees to represent all alternative divisions of problems in the current state
- Formal description very complex, especially end condition:
  For each collection of alternatives, one division (i.e. one alternative) has to be solved by compatible subsolutions (☞ recursive definition)
- And-or-transitions: Extend leaf by adding nodes representing alternative lists of nodes representing a division of the leaf's problem

Multiplayer Games

UNIVERSITY OF CALGARY

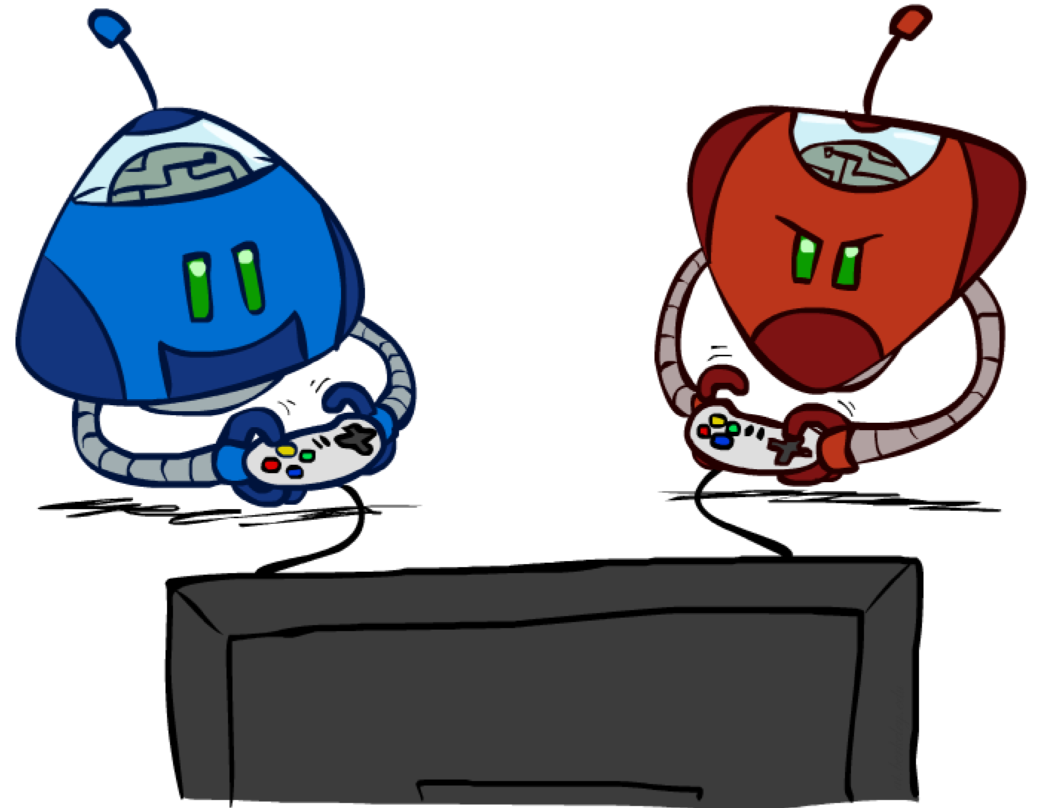
Playing multi-player games

Problem: How to determine the best next move in a game like chess or checkers given a limited amount of computing time.

Idea: Search among the possible alternative moves and their consequences

☞ use and-or-trees to represent search state

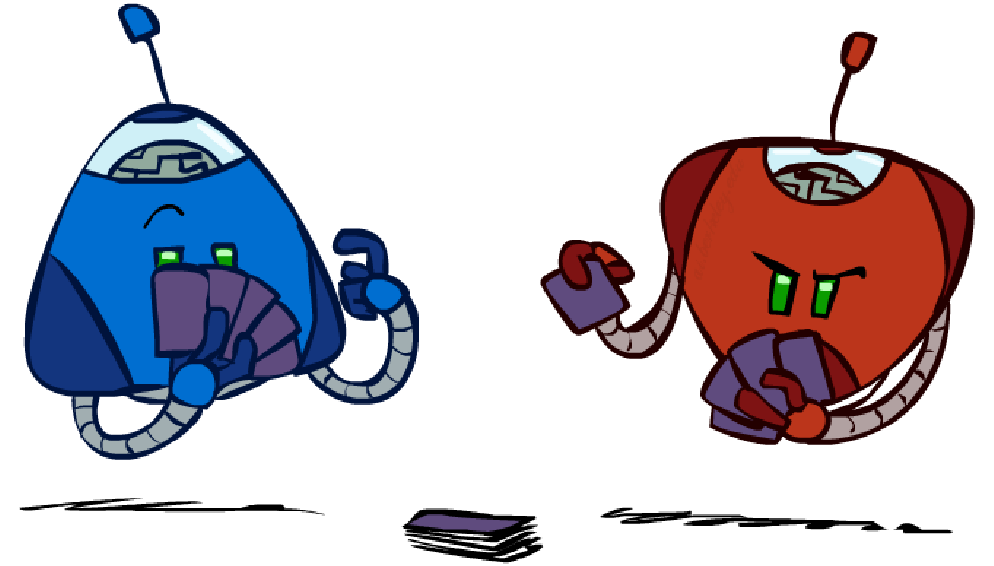
- and-part: select one of your possible moves
- or-part: considering all possible counter-moves of your opponent(s)
- problems: game situations



UNIVERSITY OF CALGARY

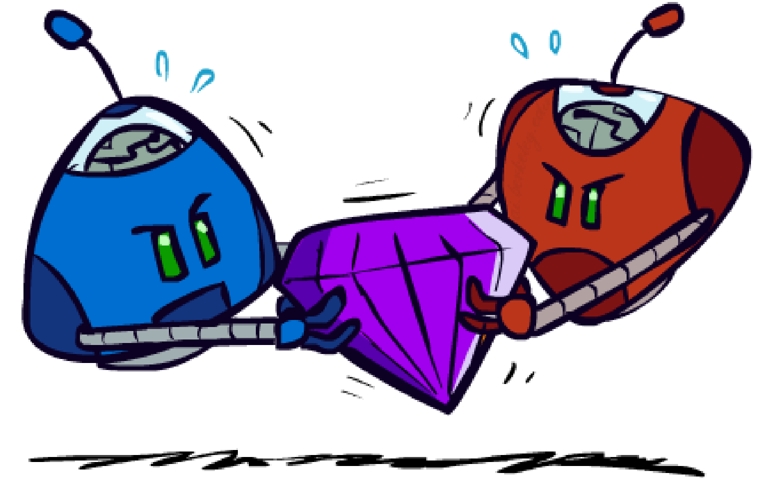
Types of Games

- Many different kinds of games!
- Axes:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Zero sum?
  - Perfect information (can you see the state)?



UNIVERSITY OF CALGARY

Types of Games



- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, and more are all possible
    - We don’t make AI to act in isolation, it should a) work around people and b) help people
    - That means that every AI agent needs to solve a game
- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition

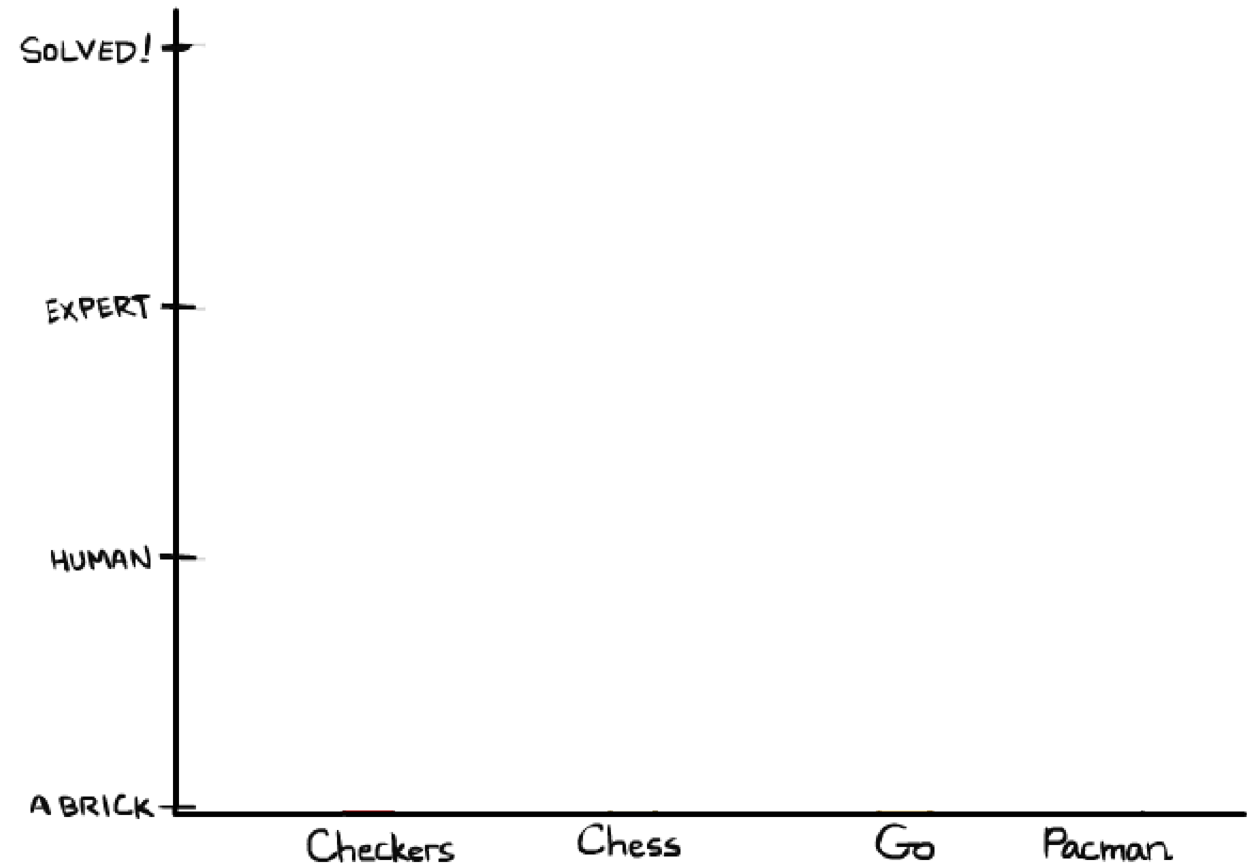
UNIVERSITY OF CALGARY

Zero-Sum Game Games ☺

- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!

- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

- **Go:** Human champions are now starting to be challenged by machines, though the best humans still beat the best machines. In go, b > 300! Classic programs use pattern knowledge bases, but big recent advances use Monte Carlo (randomized) expansion methods.

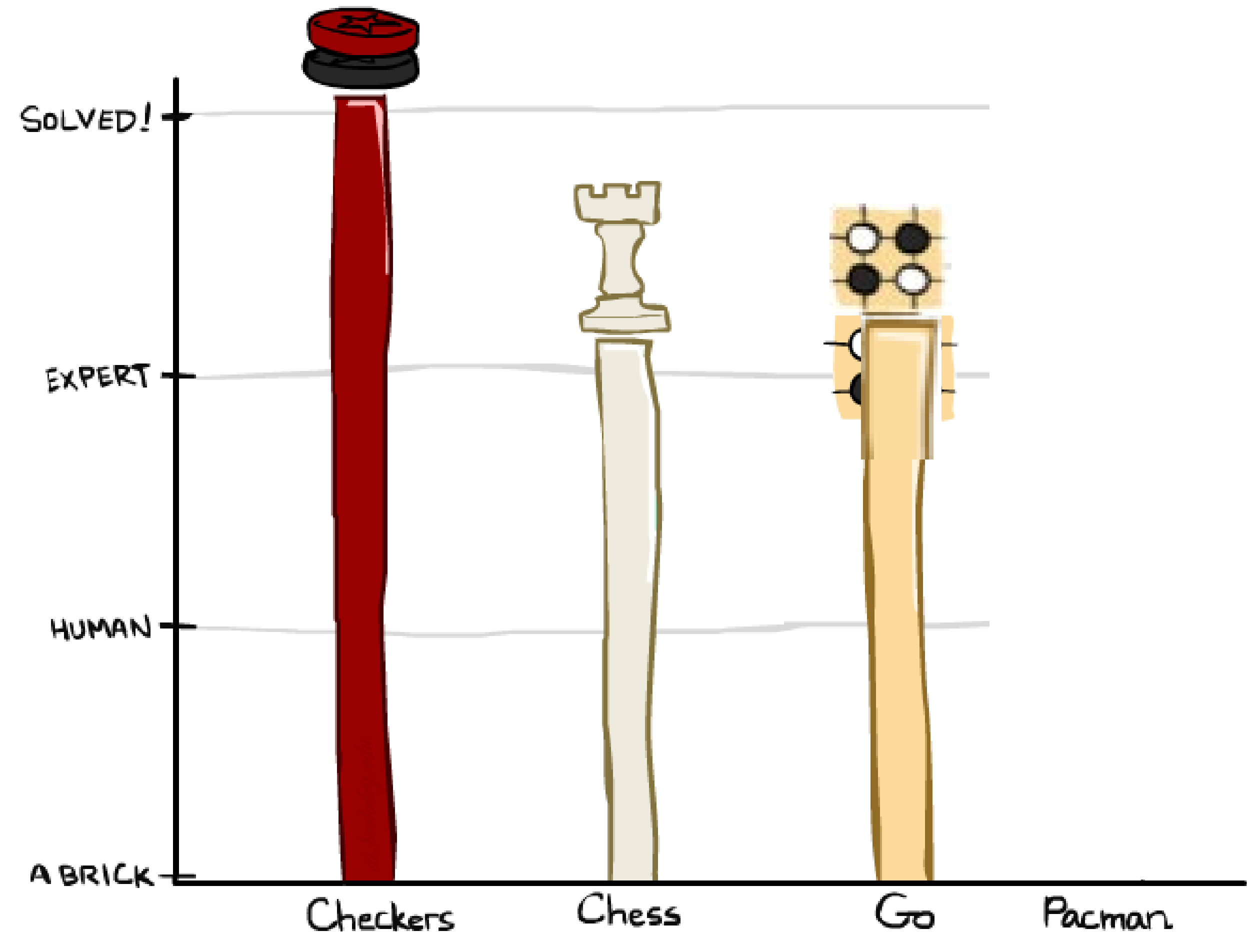


Zero-Sum Game Games ☺

- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!

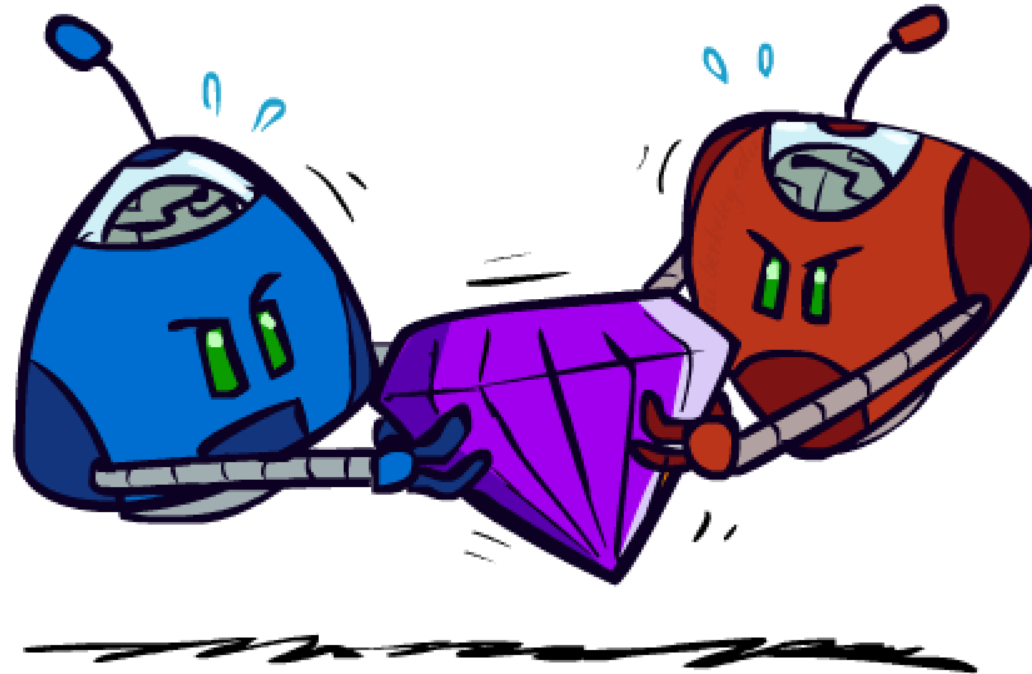
- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

- **Go :2016: Alpha GO defeats human champion. Uses Monte Carlo Tree Search, learned evaluation function.**

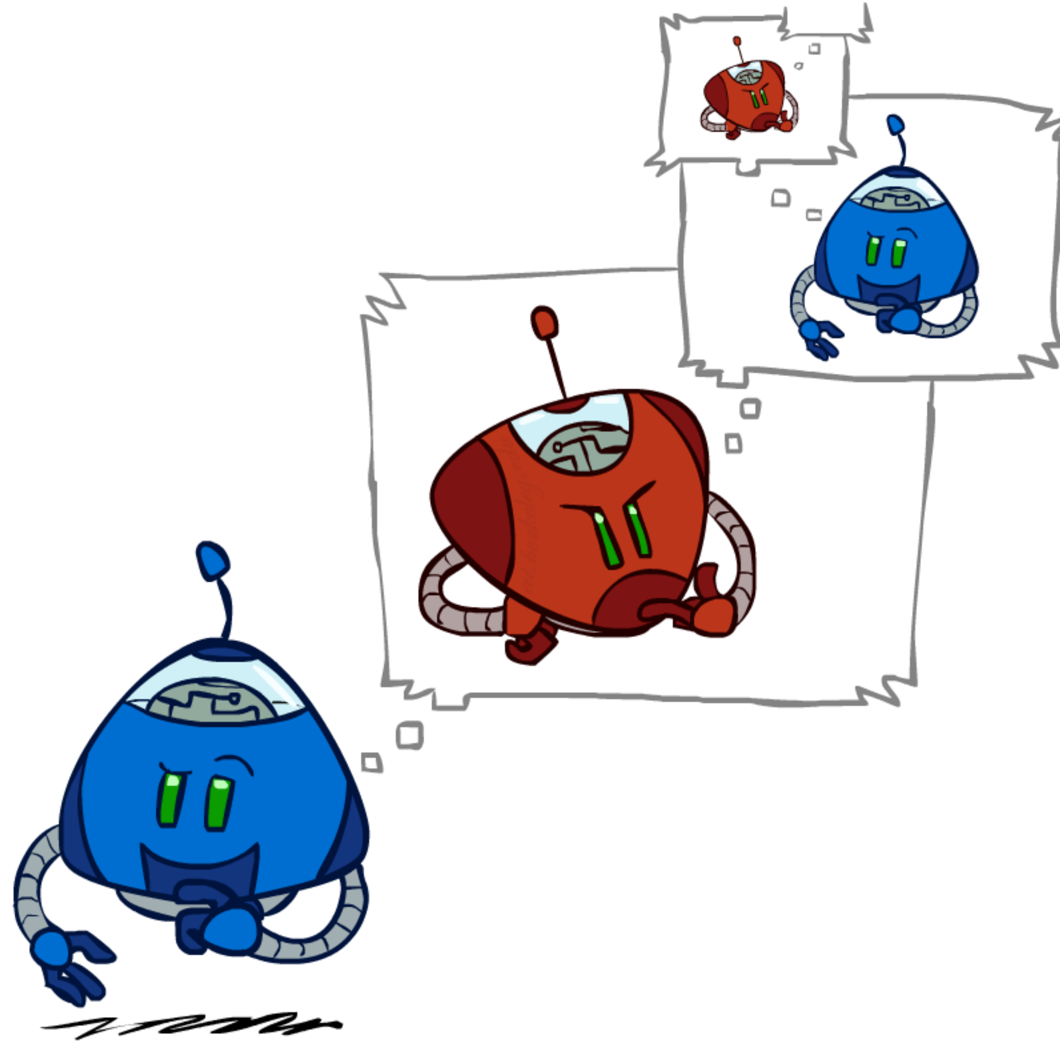


UNIVERSITY OF CALGARY

Adversarial Games

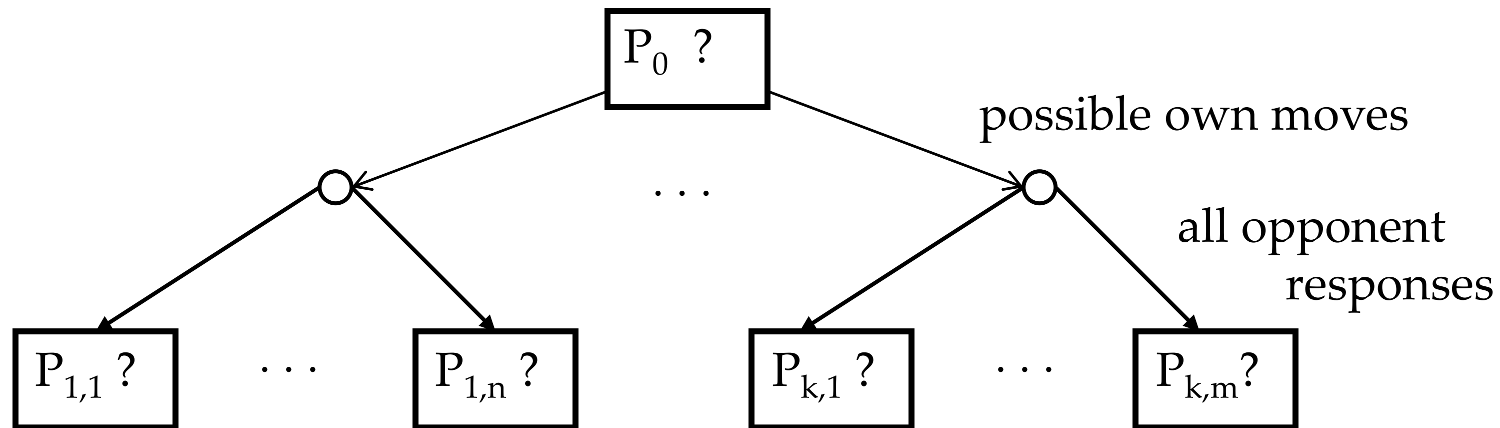


Adversarial Search



UNIVERSITY OF CALGARY

Playing multi-player games (II)



Problem: usually we can not search all branches until game is decided

☞ search control decides based on current situation where to go deeper

UNIVERSITY OF CALGARY

New: Cost -> Utility!

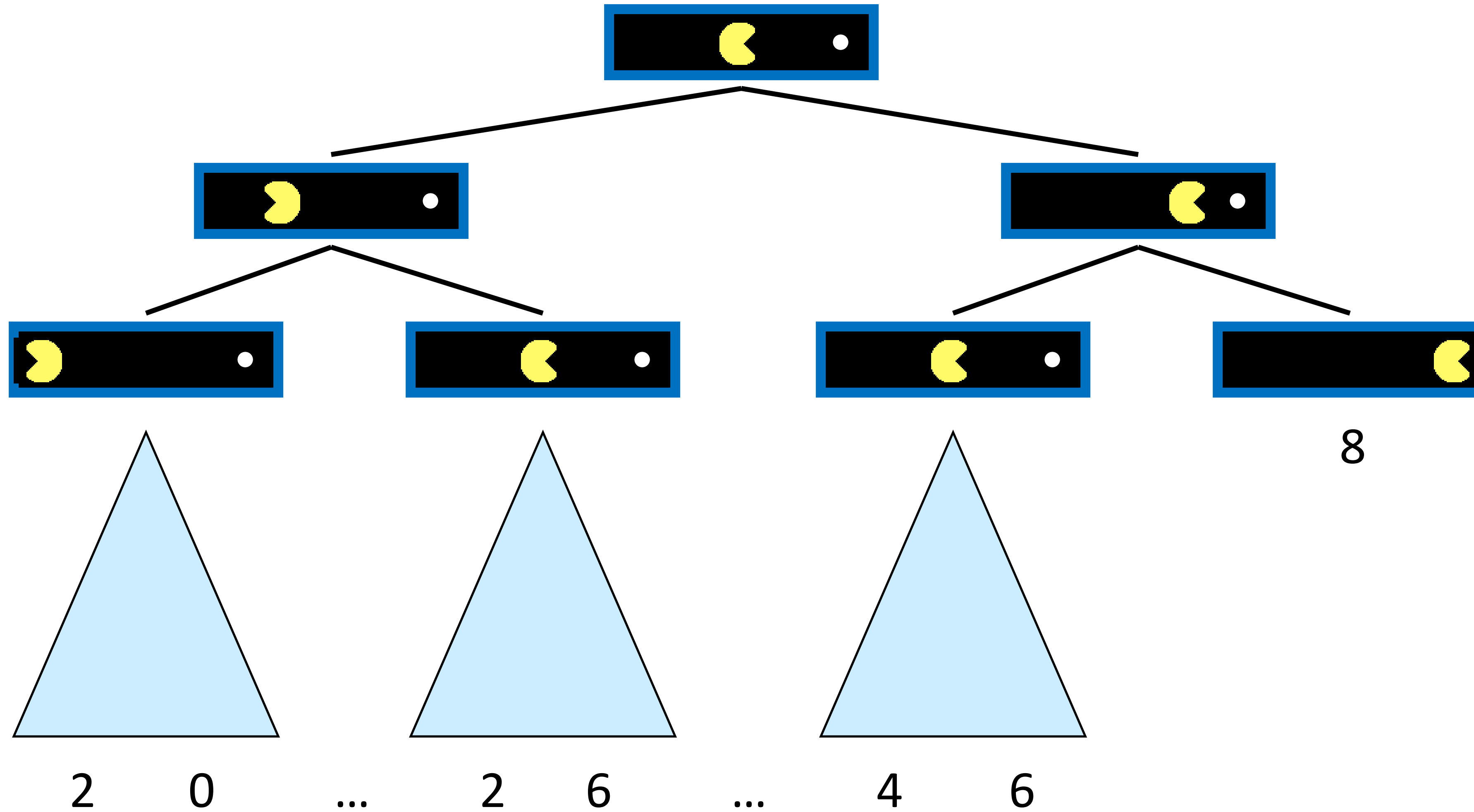
- no longer minimizing cost!
- agent now wants to maximize its score/utility!

UNIVERSITY OF CALGARY

Trees

UNIVERSITY OF CALGARY

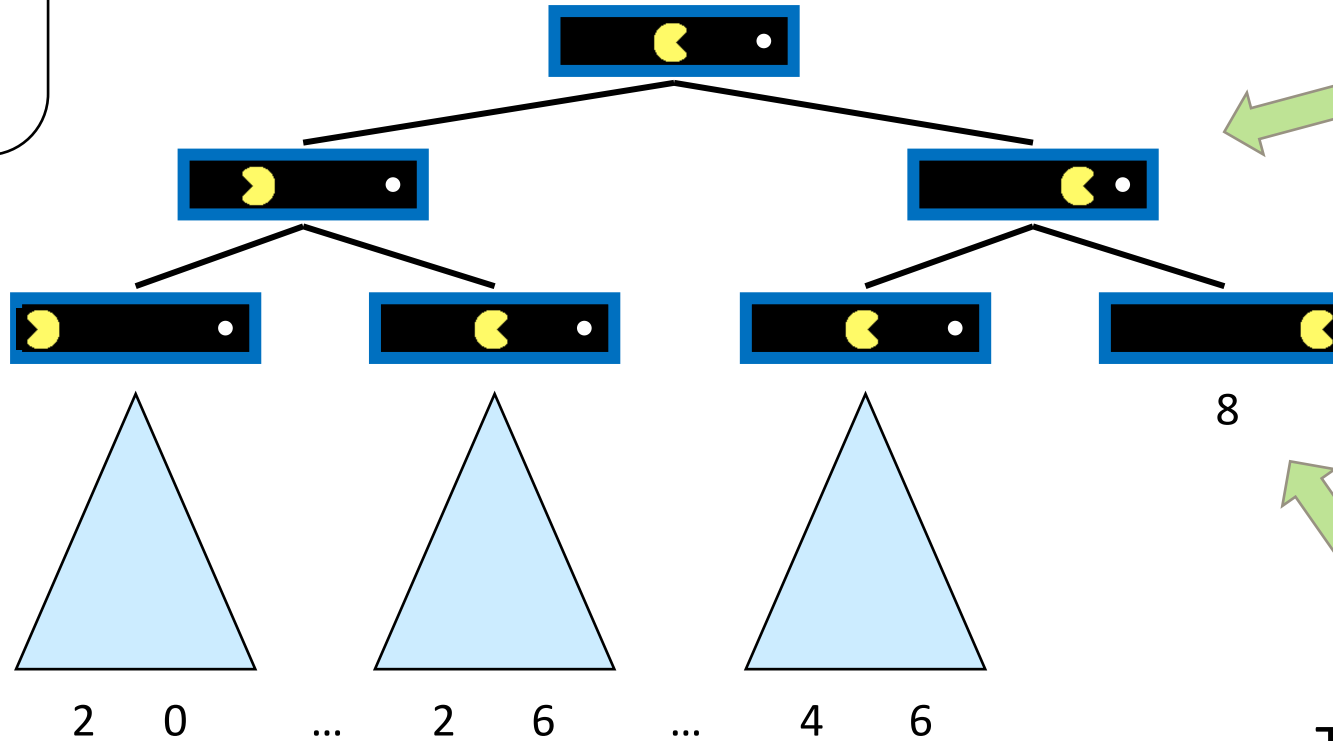
Single-Agent Trees



UNIVERSITY OF CALGARY

Value of a State

Value of a state:
The best achievable outcome (utility) from that state



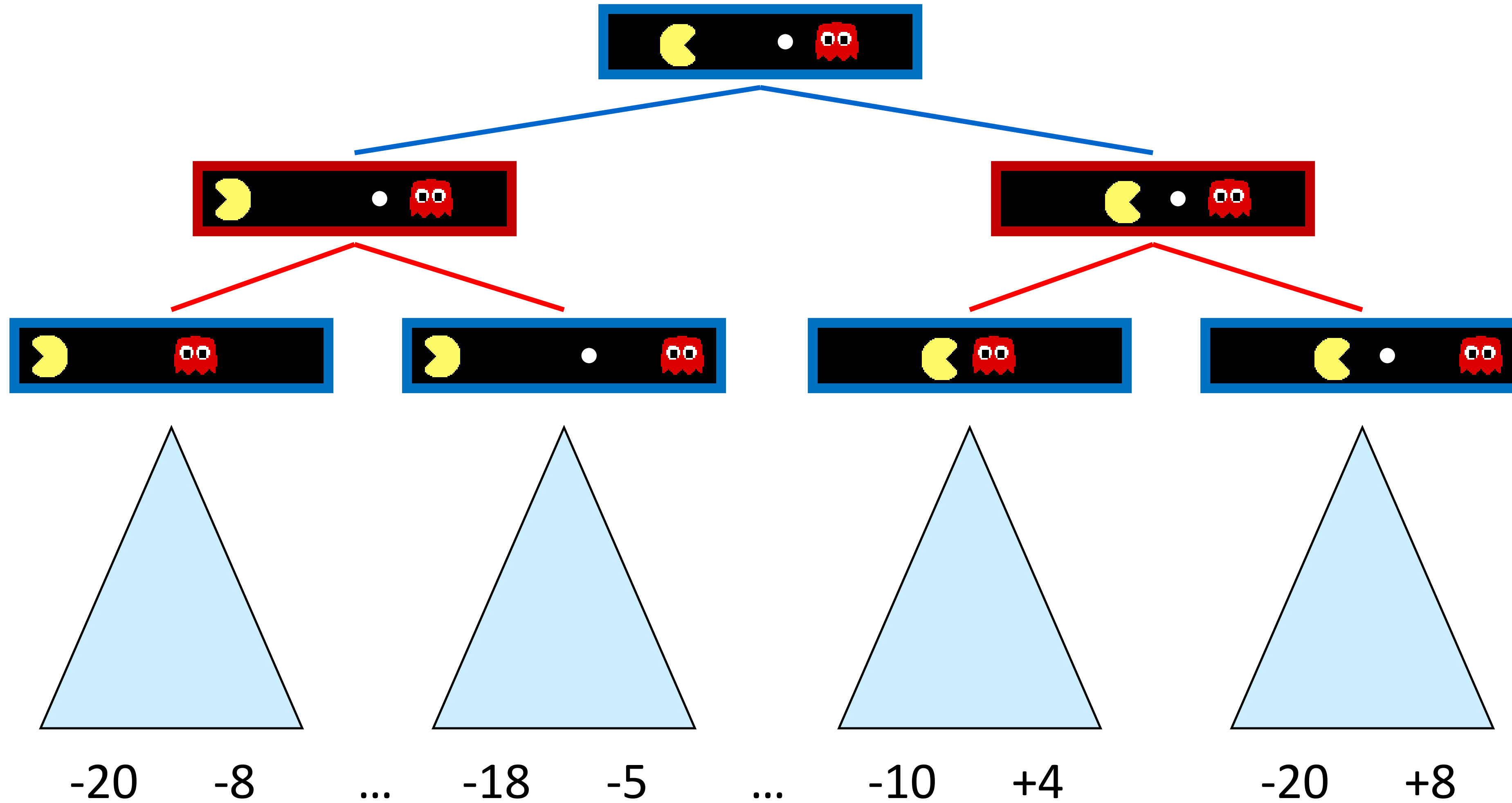
Non-Terminal States:

$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$

Terminal States:

$$V(s) = \text{known}$$

Adversarial Game Trees



UNIVERSITY OF CALGARY

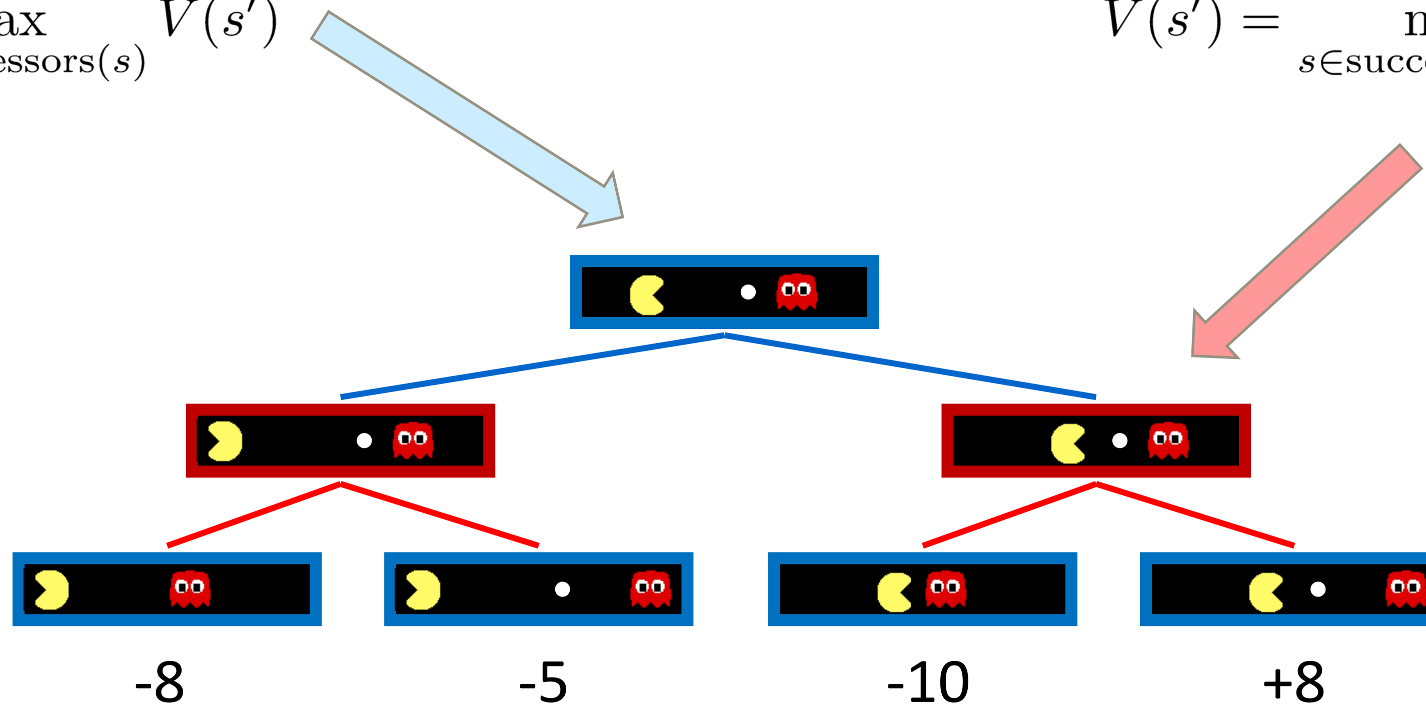
Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

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



Terminal States:

$$V(s) = \text{known}$$

UNIVERSITY OF CALGARY

Tic-Tac-Toe Game Tree

MAX (X)

MIN (O)

MAX (X)

MIN (O)

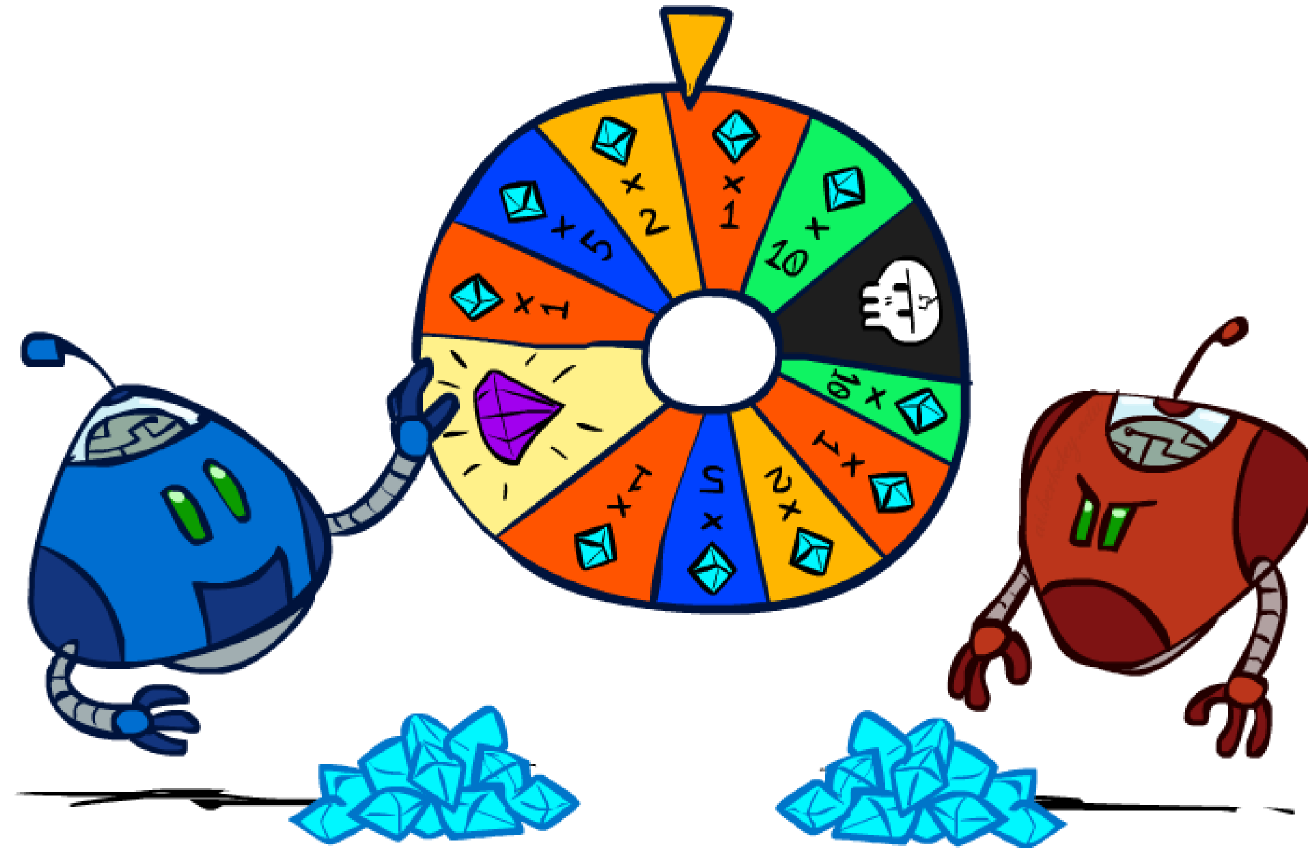
TERMINAL

Utility −1 0 +1

MiniMax

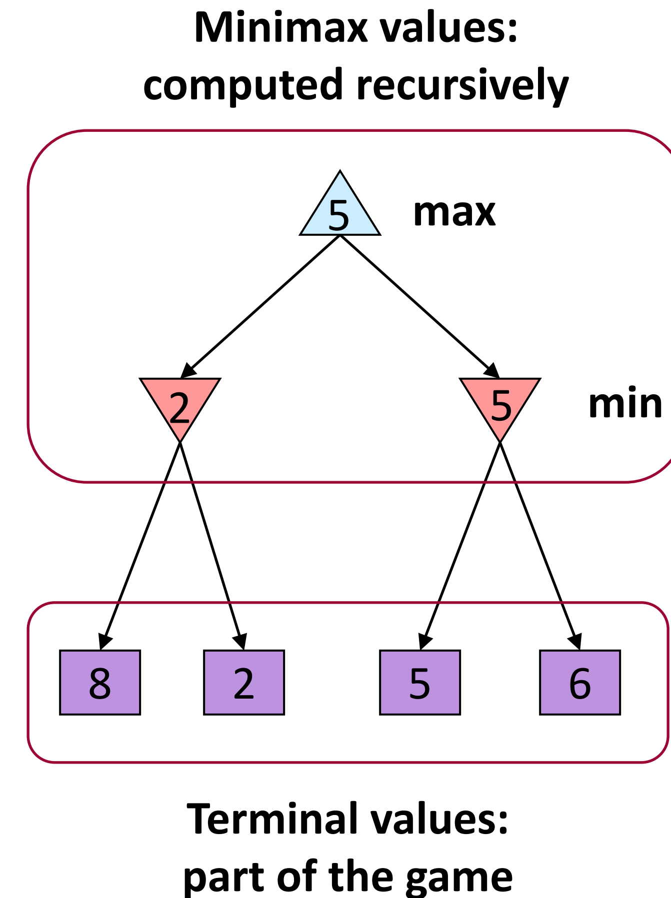
UNIVERSITY OF CALGARY

Min-Max



Adversarial Search (Minimax)

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result
- Minimax search:
  - A state-space search tree
  - Players alternate turns
  - Compute each node's minimax value: the best achievable utility against a rational (optimal) adversary



Minimax Implementation (Dispatch)

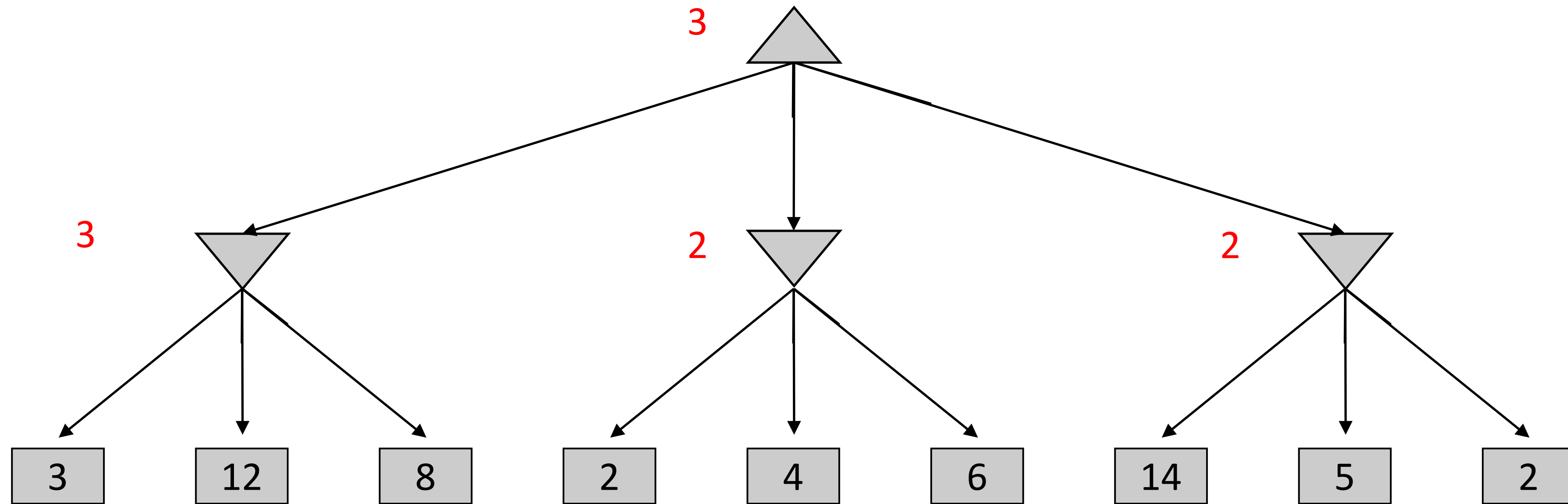
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

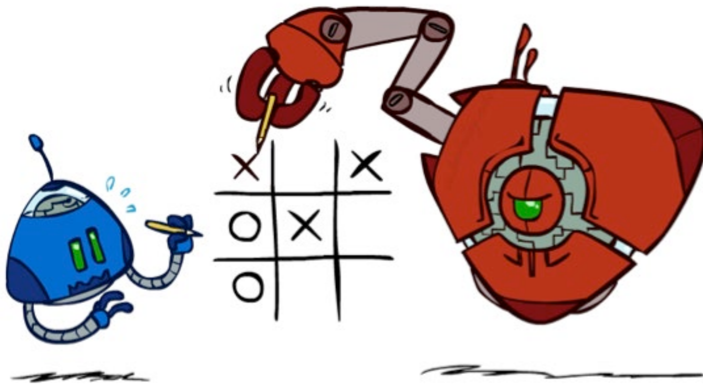
UNIVERSITY OF CALGARY

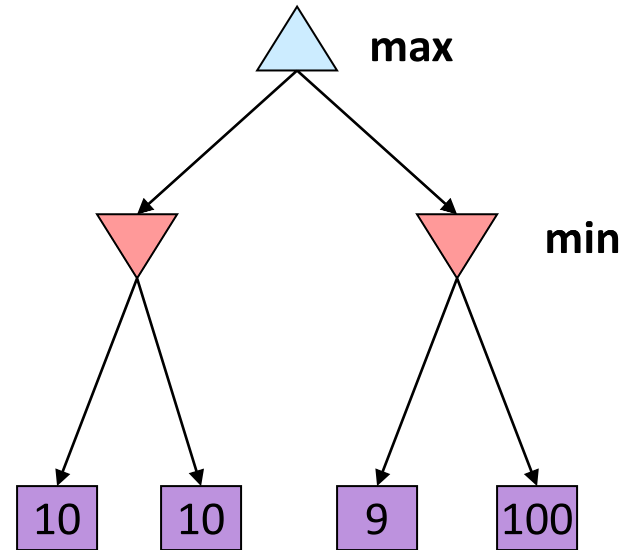
Minimax Example

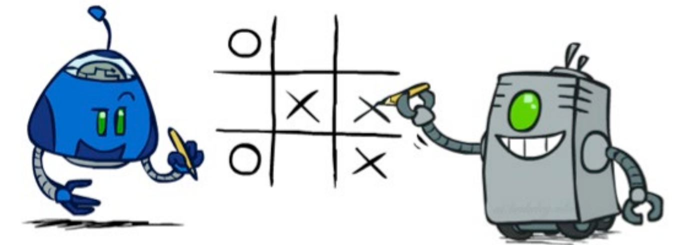


UNIVERSITY OF CALGARY

Minimax Properties





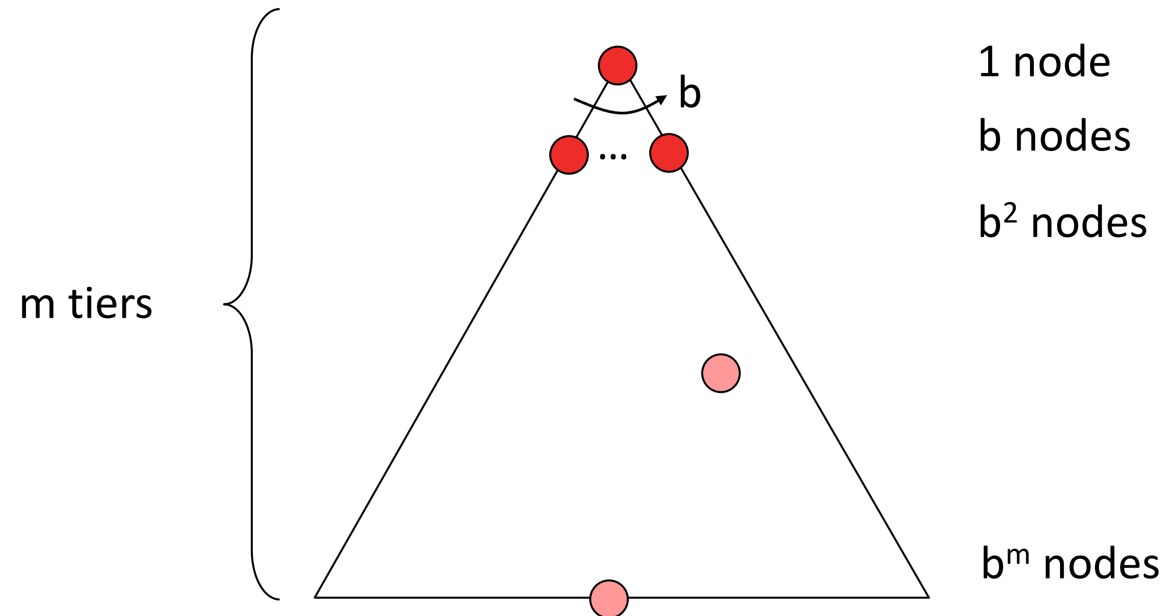


Optimal against a perfect player. Otherwise?

Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths



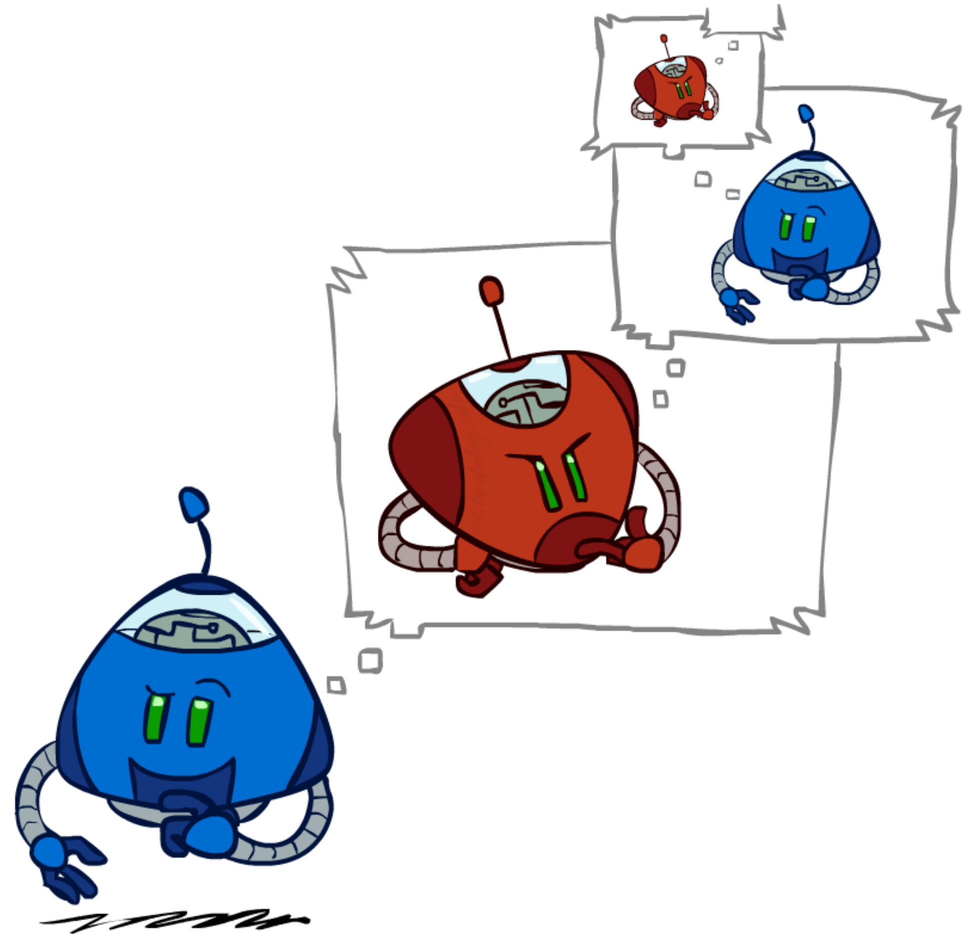
- Number of nodes in entire tree?
  - $1 + b + b^2 + ... b^m = O(b^m)$

UNIVERSITY OF CALGARY

Minimax Efficiency

- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: $O(bm)$

- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?



UNIVERSITY OF CALGARY

Considerations

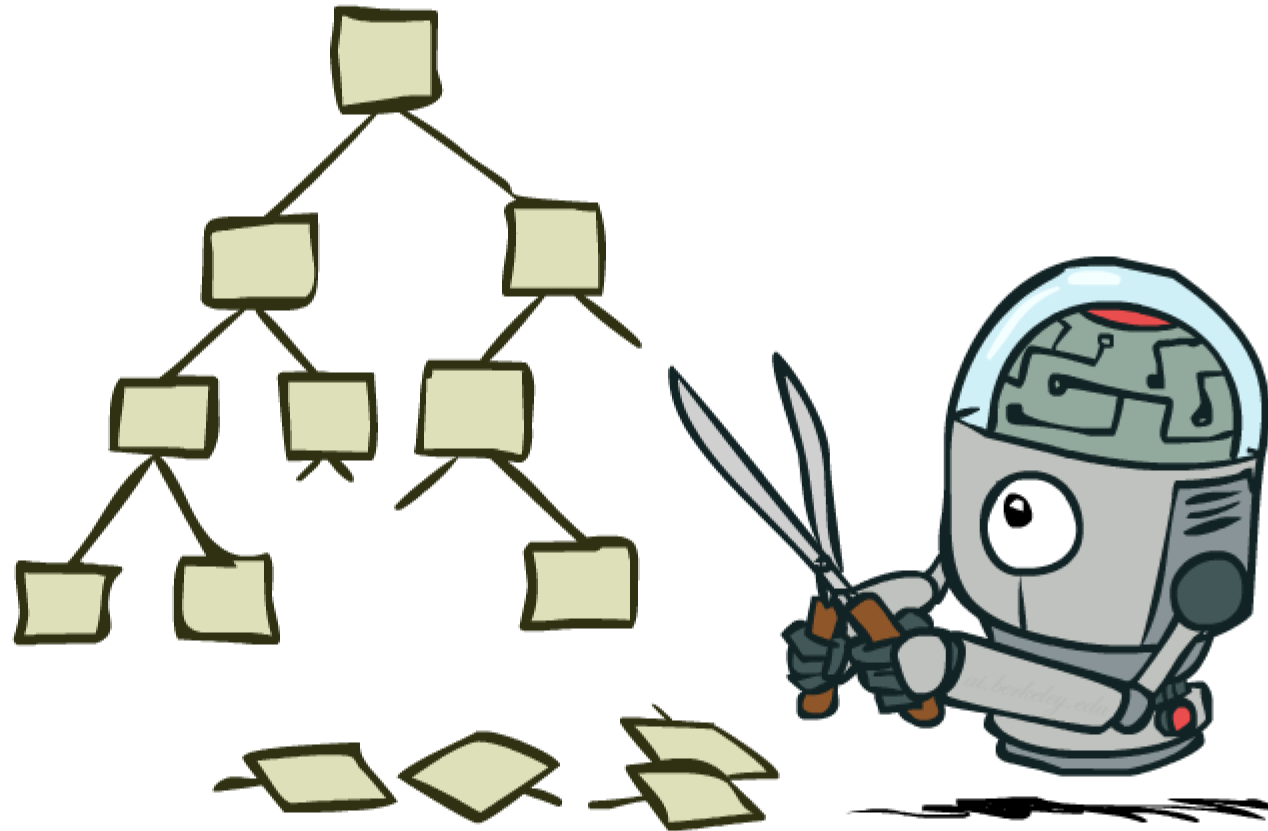
UNIVERSITY OF CALGARY

Resource Limits

Pruning

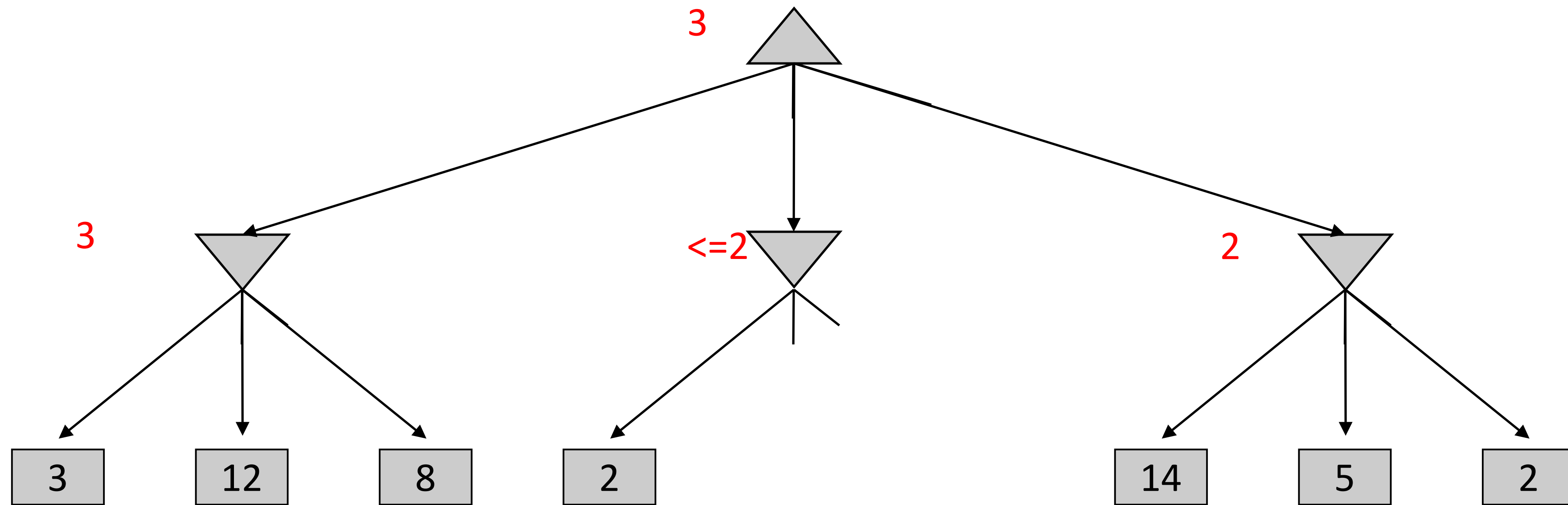
UNIVERSITY OF CALGARY

Game Tree Pruning



UNIVERSITY OF CALGARY

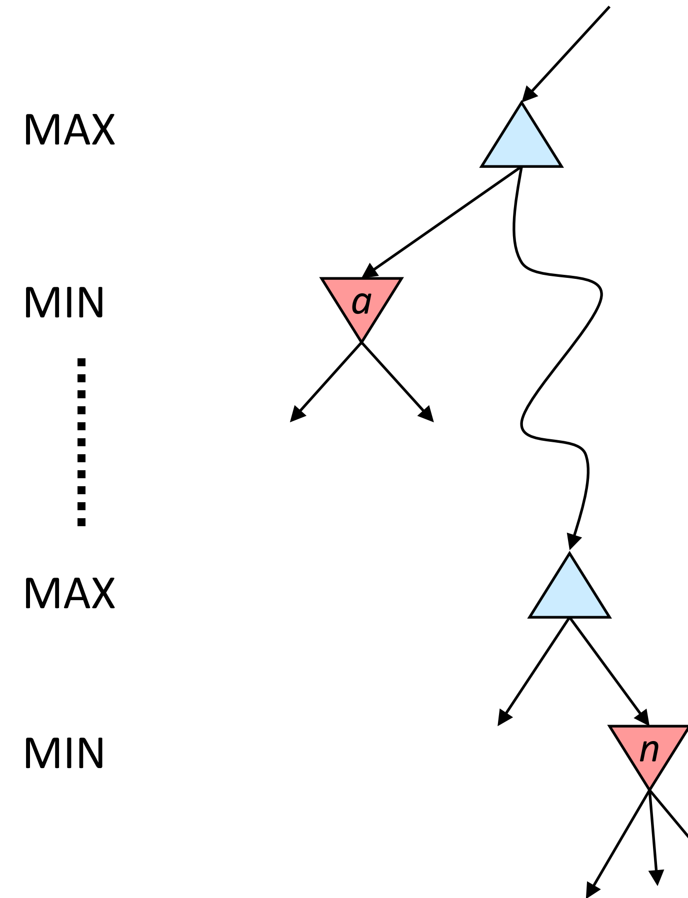
Minimax Example



UNIVERSITY OF CALGARY

Alpha-Beta Pruning

- General configuration (MIN version)
  - We're computing the MIN-VALUE at some node *n*
  - We're looping over *n*'s children
  - *n*'s estimate of the childrens' min is dropping
  - Who cares about *n*'s value? MAX
  - Let *a* be the best value that MAX can get at any choice point along the current path from the root
  - If *n* becomes worse than *a*, MAX will avoid it, so we can stop considering *n*'s other children (it's already bad enough that it won't be played)



- MAX version is symmetric

UNIVERSITY OF CALGARY

Alpha-Beta Implementation

α: MAX's best option on path to root
β: MIN's best option on path to root

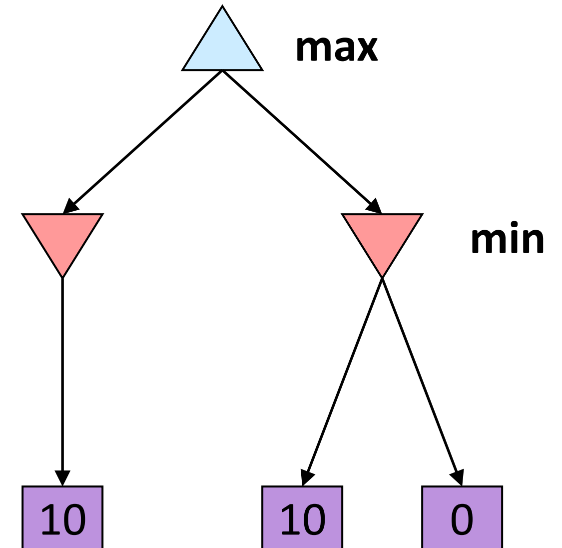
```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```


UNIVERSITY OF CALGARY

Alpha-Beta Pruning Properties

- This pruning has no effect on minimax value computed for the root!
- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - So the most naïve version won't let you do action selection
- Good child ordering improves effectiveness of pruning
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…
- This is a simple example of metareasoning (computing about what to compute)

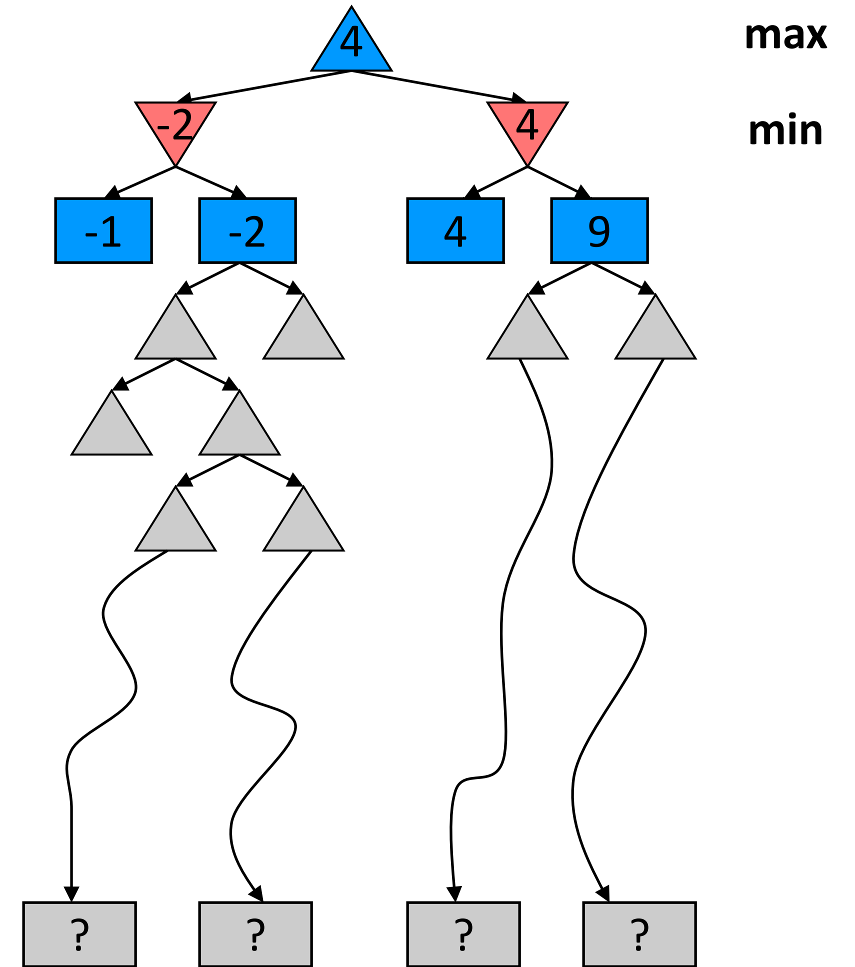


UNIVERSITY OF CALGARY

Depth Limit

Resource Limits

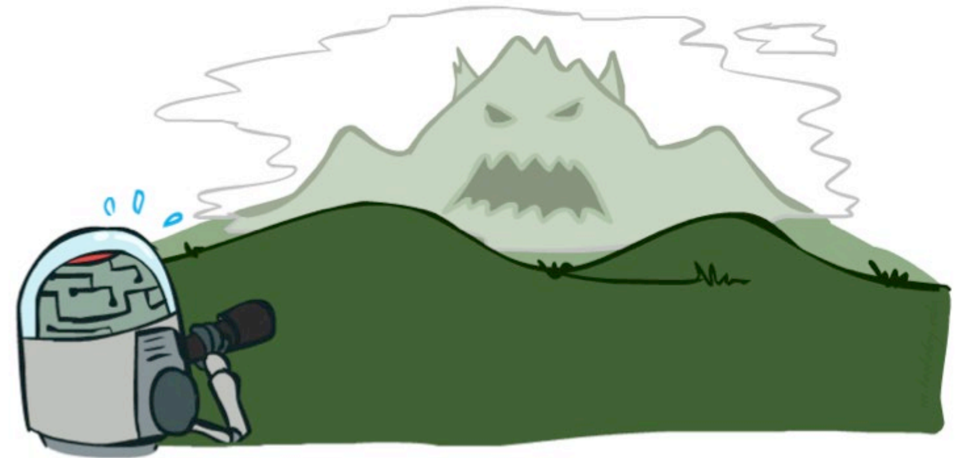
- Problem: In realistic games, cannot search to leaves!
- Solution: Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Replace terminal utilities with an evaluation function for non-terminal positions
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - α-β reaches about depth 8 – decent chess program
- Guarantee of optimal play is gone
- More plies makes a BIG difference
- Use iterative deepening for an anytime algorithm

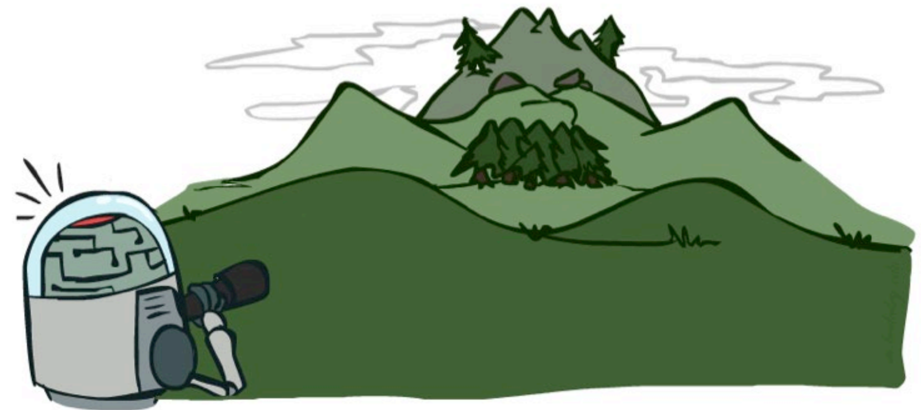


UNIVERSITY OF CALGARY

Depth Matters

- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- An important example of the tradeoff between complexity of features and complexity of computation





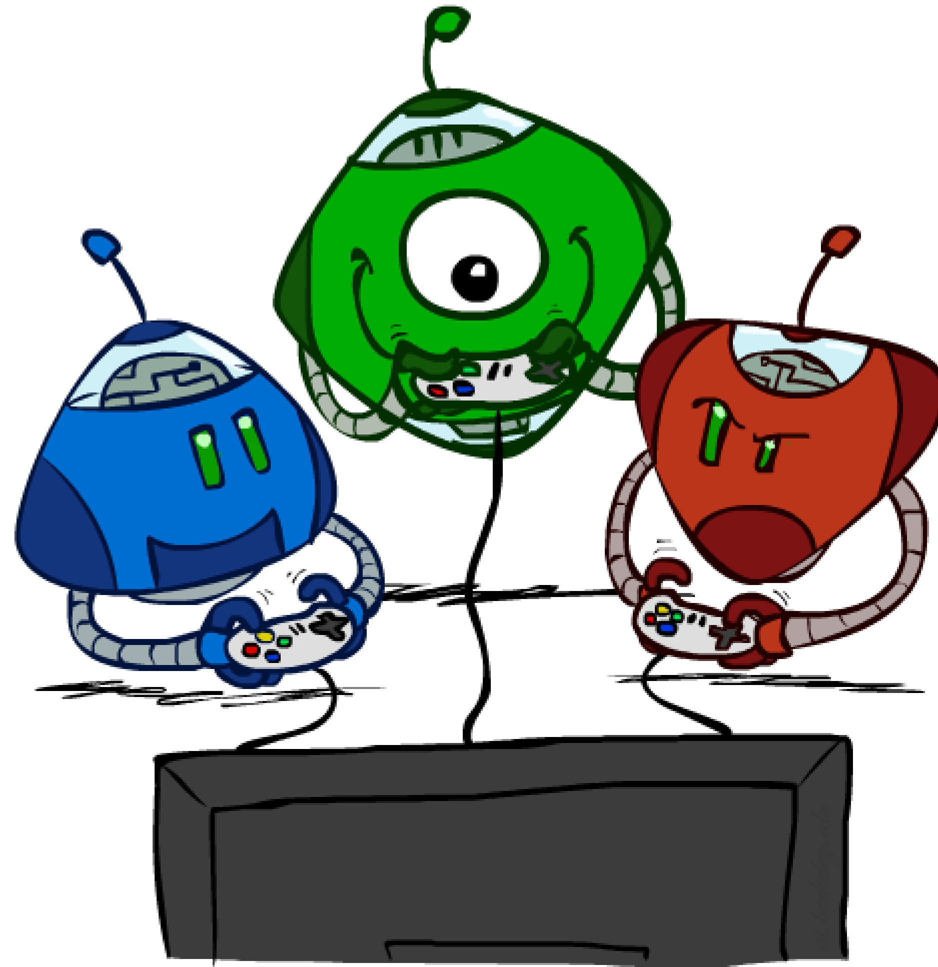
[Demo: depth limited (L6D4, L6D5)]

UNIVERSITY OF CALGARY

Other Game Types

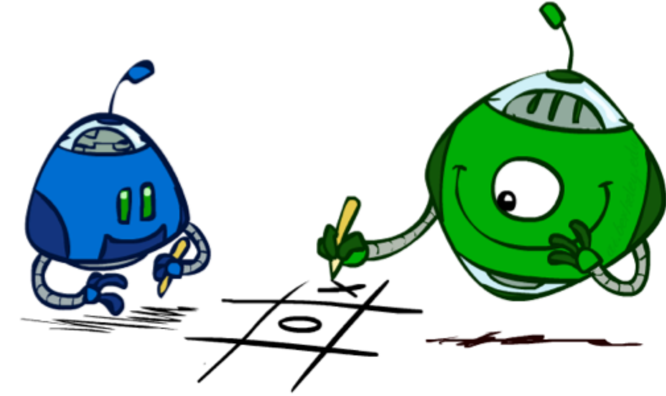
UNIVERSITY OF CALGARY

Other Game Types



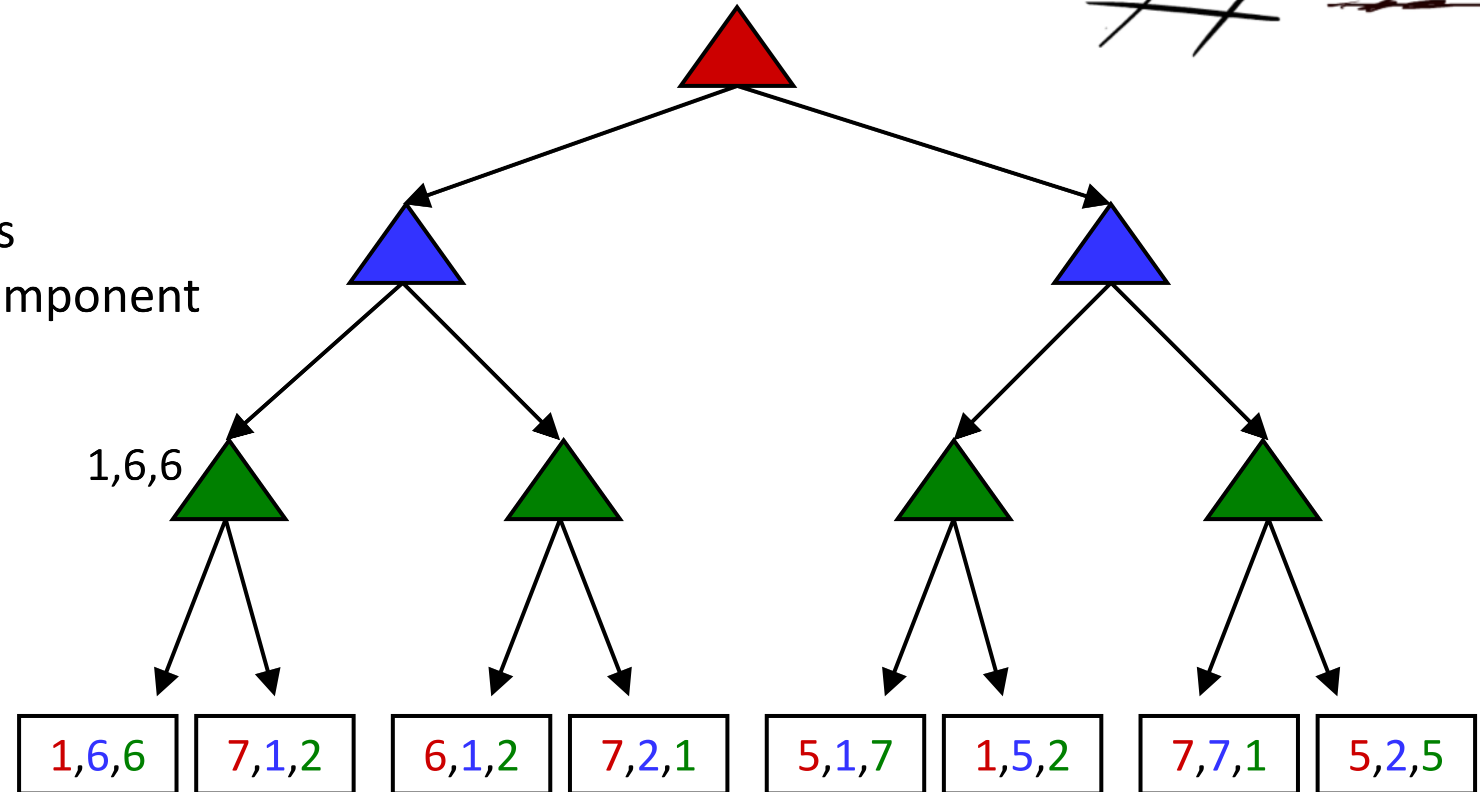
UNIVERSITY OF CALGARY

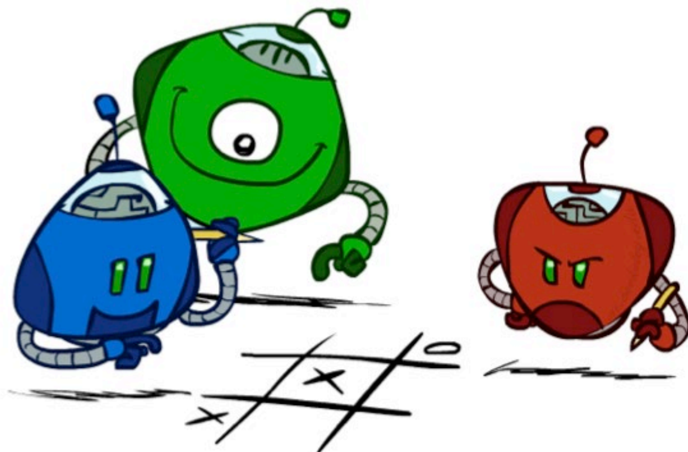
Multi-Agent Utilities



- What if the game is not zero-sum, or has multiple players?

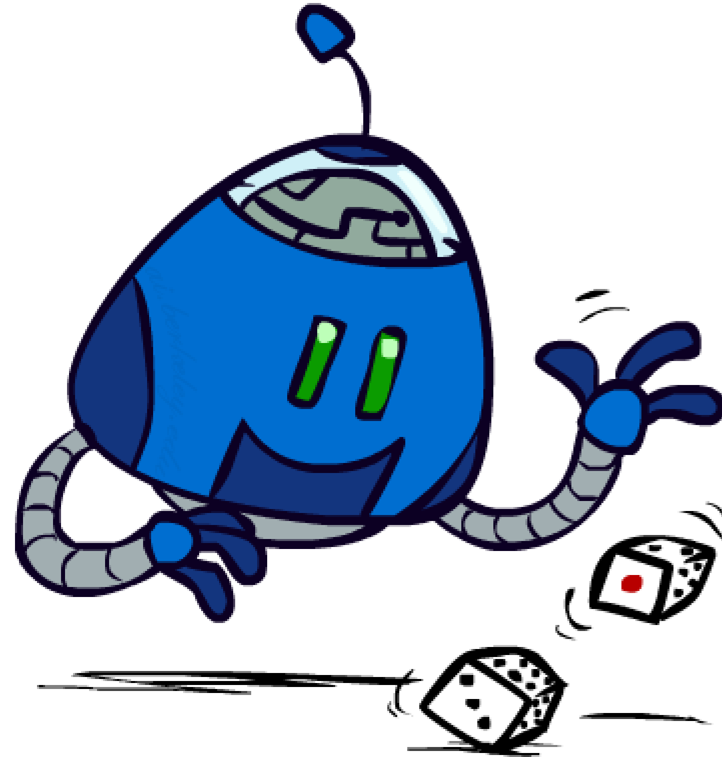
- Generalization of minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and competition dynamically…



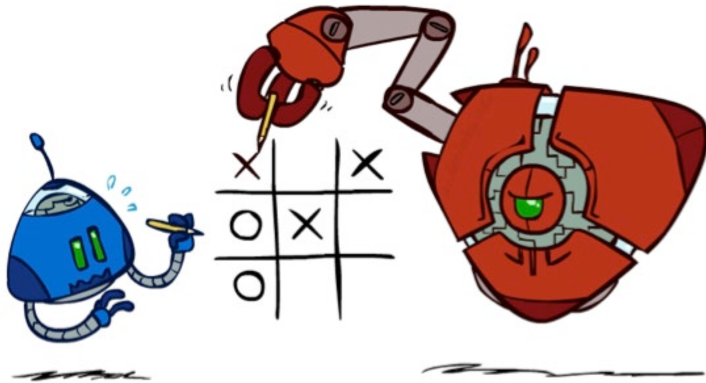


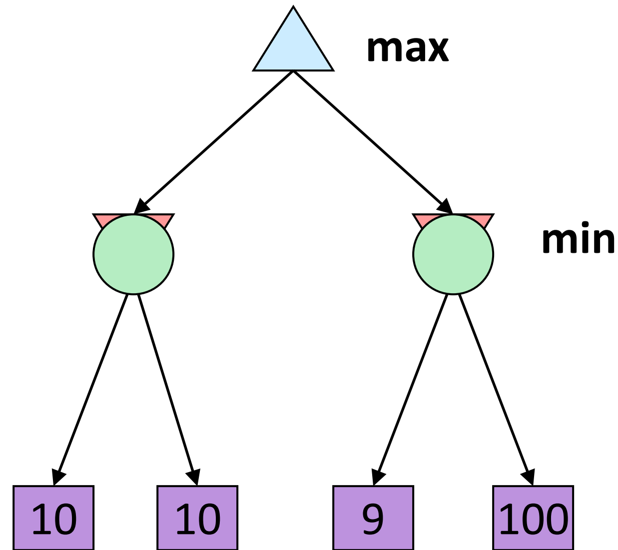
UNIVERSITY OF CALGARY

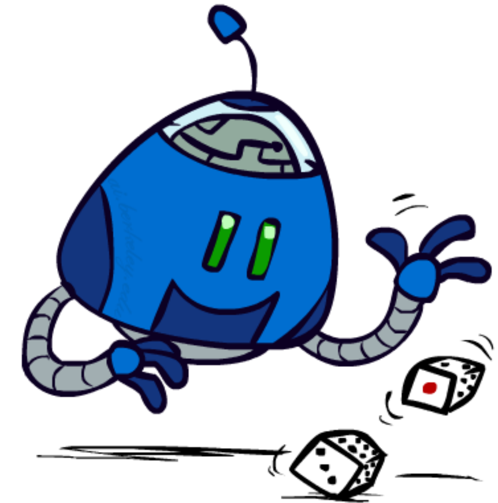
Uncertain Outcomes



Worst-Case vs. Average Case







Idea: Uncertain outcomes controlled by chance, not an adversary!

UNIVERSITY OF CALGARY

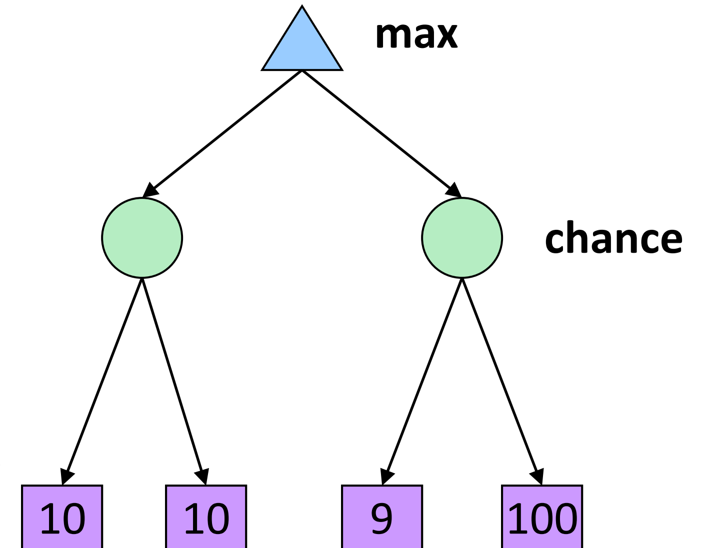
Why not minimax?

- Worst case reasoning is too conservative
- Need average case reasoning

UNIVERSITY OF CALGARY

Expectimax Search

- Why wouldn't we know what the result of an action will be?
  - Explicit randomness: rolling dice
  - Unpredictable opponents: the ghosts respond randomly
  - Unpredictable humans: humans are not perfect
  - Actions can fail: when moving a robot, wheels might slip
- Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes
- Expectimax search: compute the average score under optimal play
  - Max nodes as in minimax search
  - Chance nodes are like min nodes but the outcome is uncertain
  - Calculate their expected utilities
  - I.e. take weighted average (expectation) of children
- Later, we'll learn how to formalize the underlying uncertain-result problems as Markov Decision Processes



UNIVERSITY OF CALGARY

The Dangers of Optimism and Pessimism

Dangerous Optimism

Assuming chance when the world is adversarial

Dangerous Pessimism

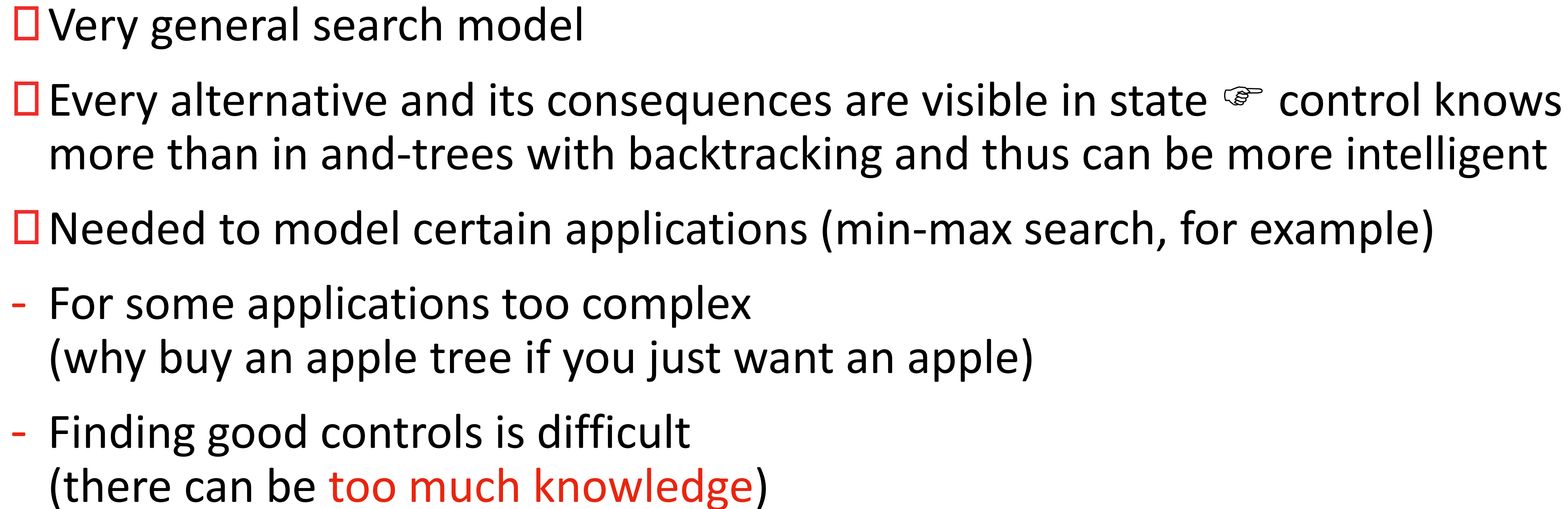
Assuming the worst case when it's not likely

UNIVERSITY OF CALGARY

Summary

UNIVERSITY OF CALGARY

Pros and Cons



- Very general search model
- Every alternative and its consequences are visible in state ☞ control knows more than in and-trees with backtracking and thus can be more intelligent
- Needed to model certain applications (min-max search, for example)

- For some applications too complex
  (why buy an apple tree if you just want an apple)
- Finding good controls is difficult
  (there can be too much knowledge)

Final Model Remarks

UNIVERSITY OF CALGARY

Some general remarks

- Some authors have suggested and-or-graph-based search with problems in nodes represented as sets of constraints as ultimate search model
- Since or-tree- and or-graph-based search processes often use an estimate on how good (with respect to distance to a solution) a leaf is, some people see them as special and-trees, resp. and-graphs (without backtracking) for optimization
  ☞ A*-algorithm (graph-based variant of branch-and-bound search)

Onward to …
Search Controls

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY