PURDUE CS47100

INTRODUCTION TO AI

ANNOUCEMENTS

- Reminder: Assignment 1 is due on this Friday!
- Cast your vote on Ed about Assignment 2 due date
  - Do you want the deadline of Assignment 2 to be extended by 2 days (from Oct 14 to Oct 16)?
  - Potentially decrease your workload during the fall break, but it also implies that we can at best release the solution to the written part of Assignment 2 on the midterm day (Oct 20)!
  - Closed by 5pm, Sept 23

RECAP: ADVERSARIAL SEARCH

- Minimax search:
  - The max player's minimax value is the maximum of its successors' values
  - The min player's minimax value is the minimum of its successors' values
- Alpha-Beta pruning:
  - If the current estimate for a min player's value is lower than the estimate for some of its ancestor max player's value, prune the remaining branches!
  - If the current estimate for a max player's value is higher than the estimate for some of its ancestor min player's value, prune the remaining branches!

Recap: Alpha-Beta Implementation

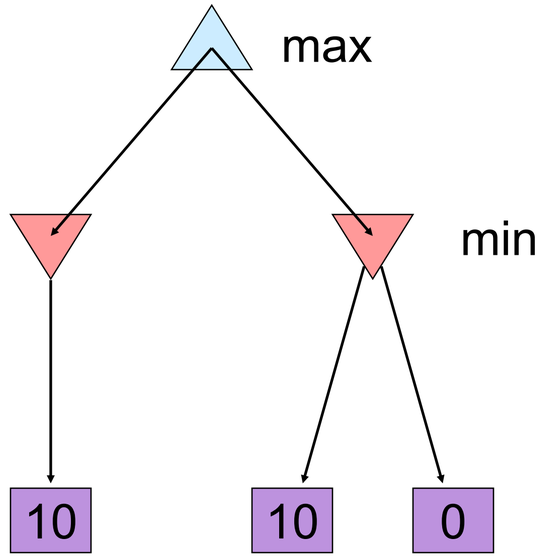
α: MAX's best option on path to root
β: MIN's best option on path to root

```
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
```

ALPHA-BETA PRUNING PROPERTIES

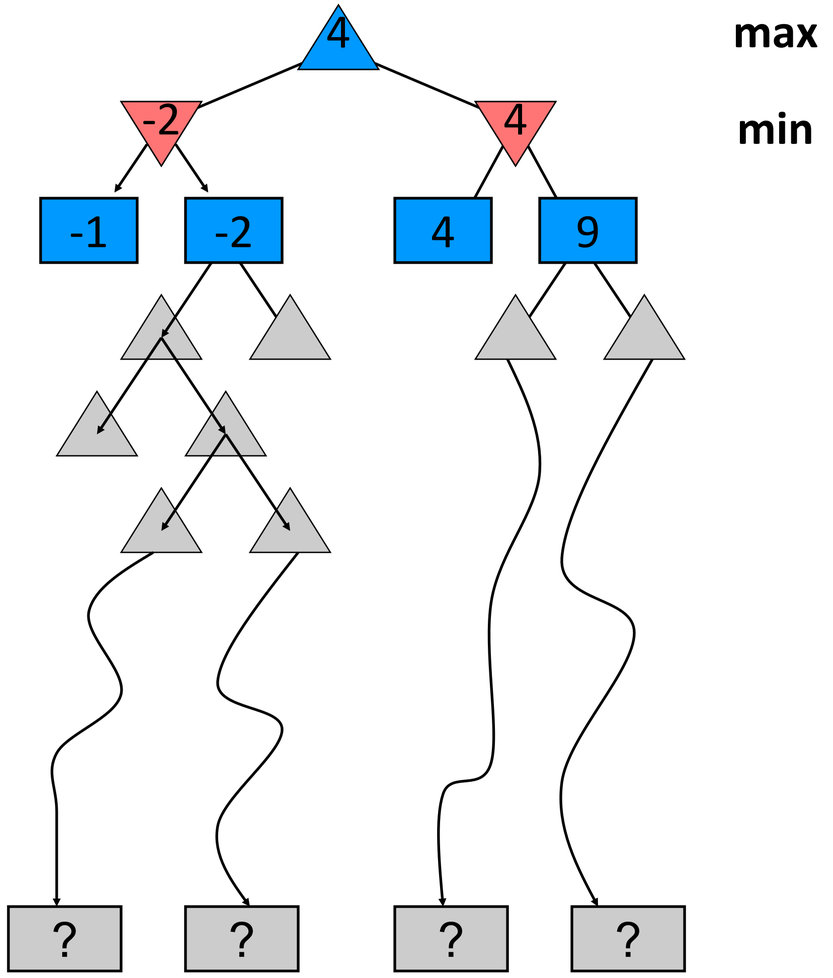
- This pruning has **no effect** on minimax value computed for the root
- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - So the most naïve version won't let you do action selection
- Good child ordering improves effectiveness of pruning
- 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)



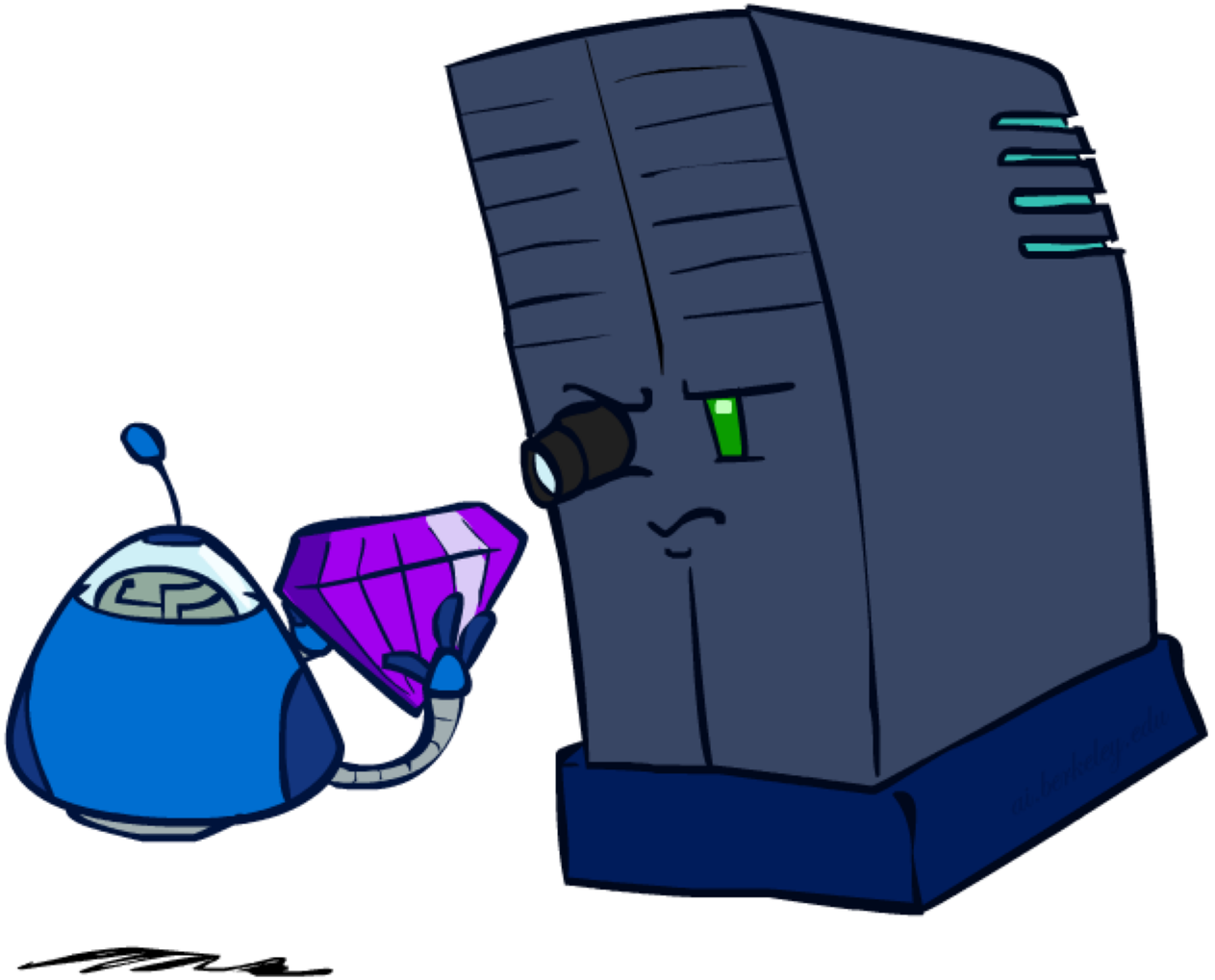
RESOURCE LIMITS

RESOURCE LIMITS

- **Problem**: In realistic games, cannot search to leaves
- 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
- **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
  - Guarantee of optimal play is gone; More plies makes a BIG difference



EVALUATION FUNCTIONS



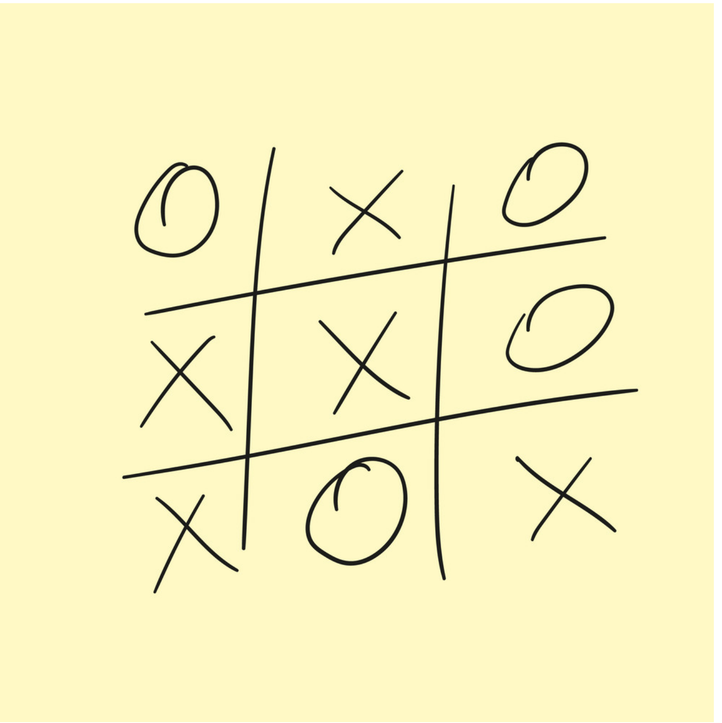
EVALUATION FUNCTIONS

- Desirable properties
  - Order terminal states in same way as true utility function
  - Strongly correlated with the actual minimax value of the states
  - Efficient to calculate
- Typically use **features** – simple characteristics of the game state that are correlated with the *probability of winning*
- The evaluation function combines feature values to produce a score:

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

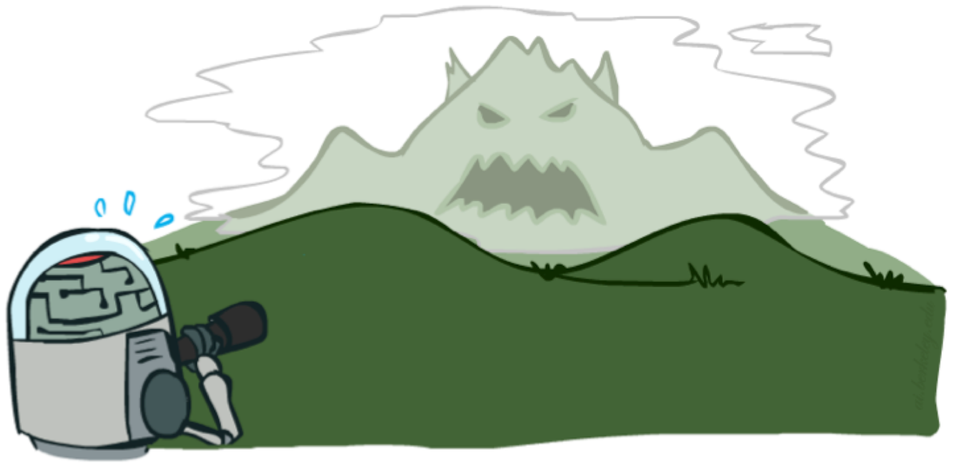
EXAMPLE FEATURES

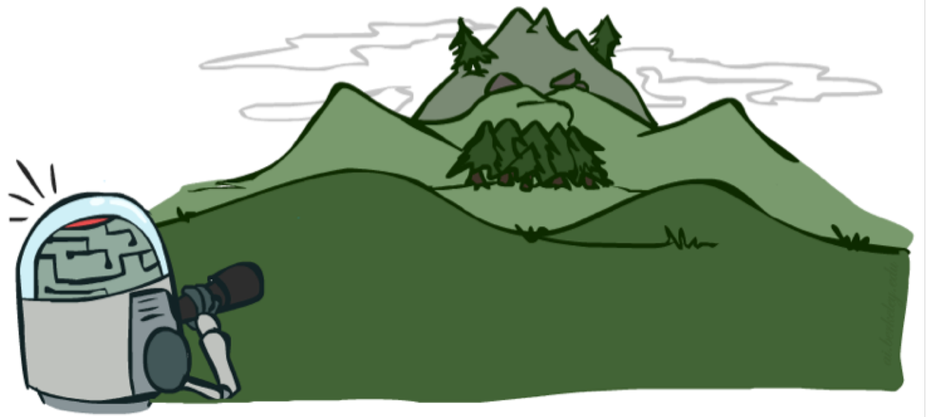
- What would be some useful features for tic-tac-toe?
  - The number of rows/columns you symbols spread?
  - Among all rows/columns/diagonals, how many are still reachable by you?
  - Appearance of winning patterns



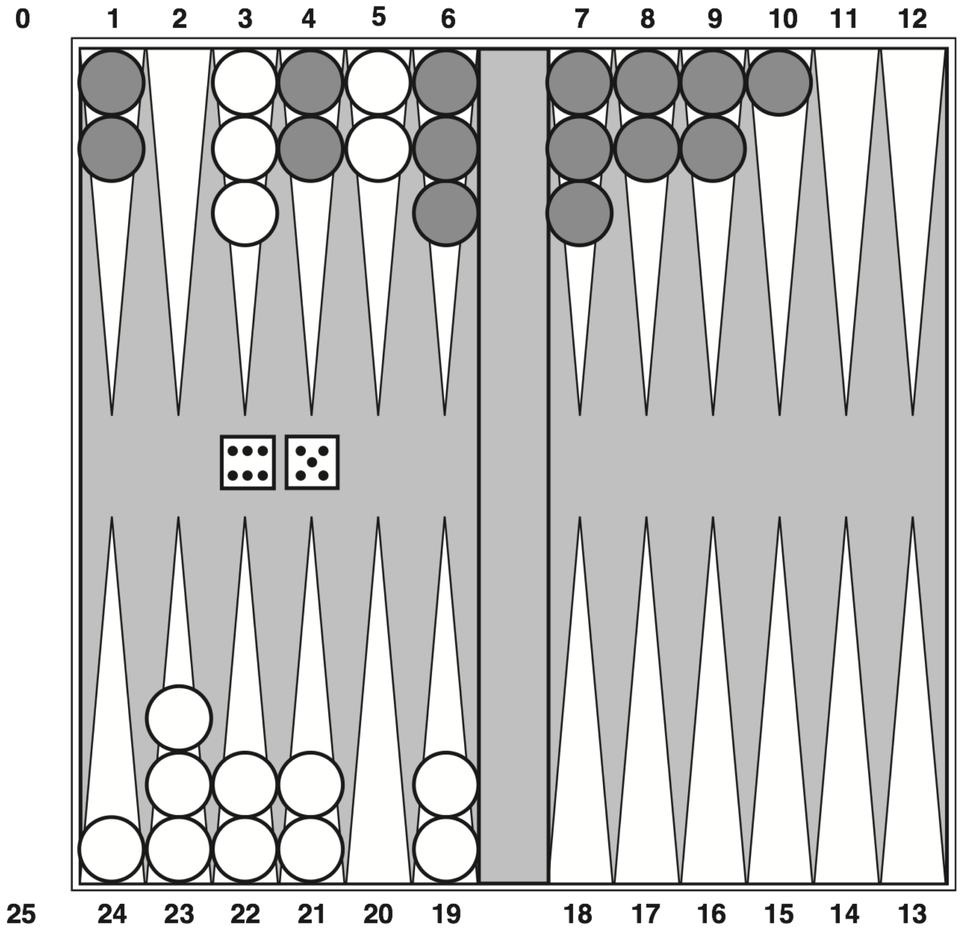
DEPTH MATTERS

- 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

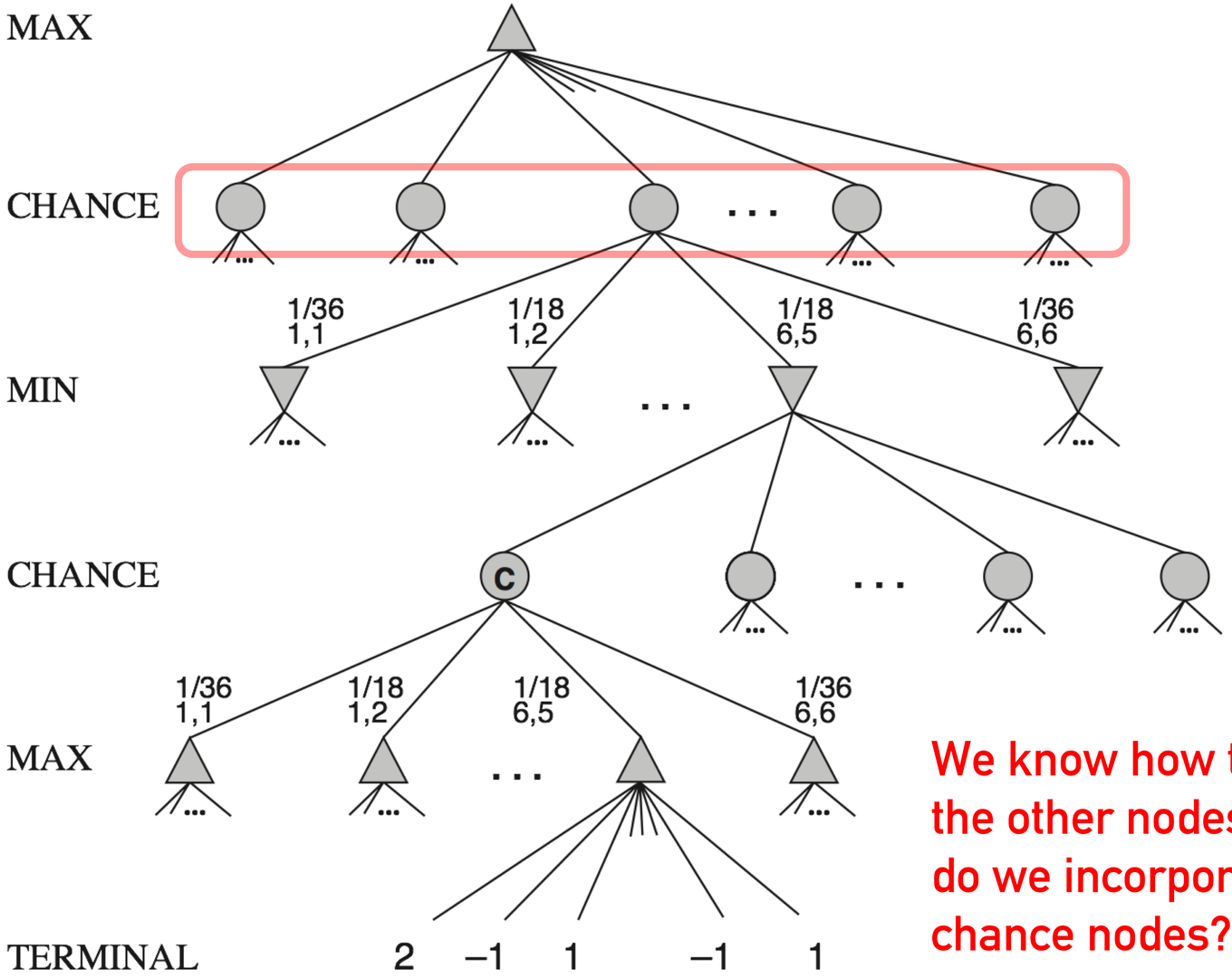




WHAT IF A GAME HAS A “CHANCE ELEMENT”?



GAME TREE WITH CHANCE ELEMENT

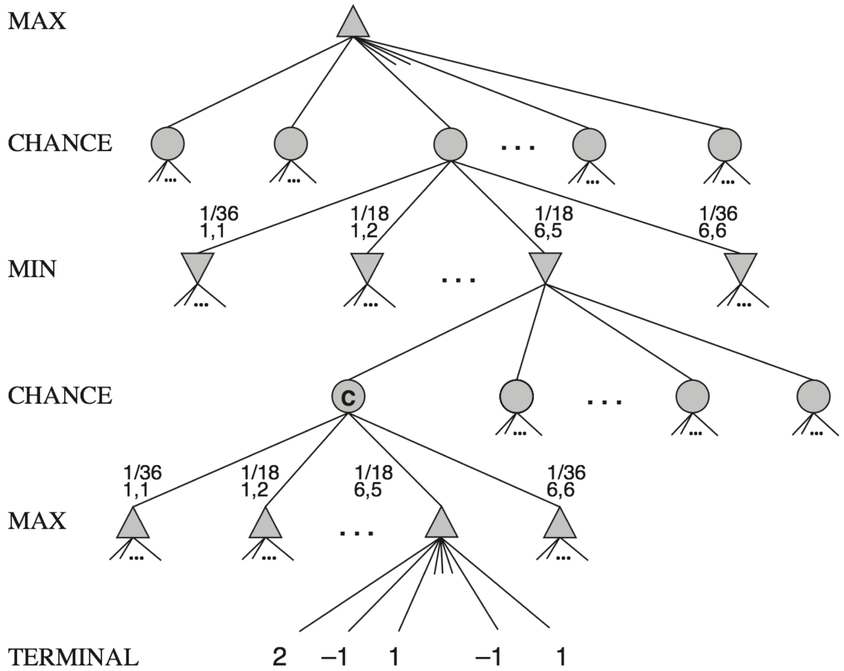


We know how to value the other nodes. How do we incorporate chance nodes?

EXPECTED MINIMAX VALUE

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

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



EXPECTED-MINIMAX-VALUE(n) =

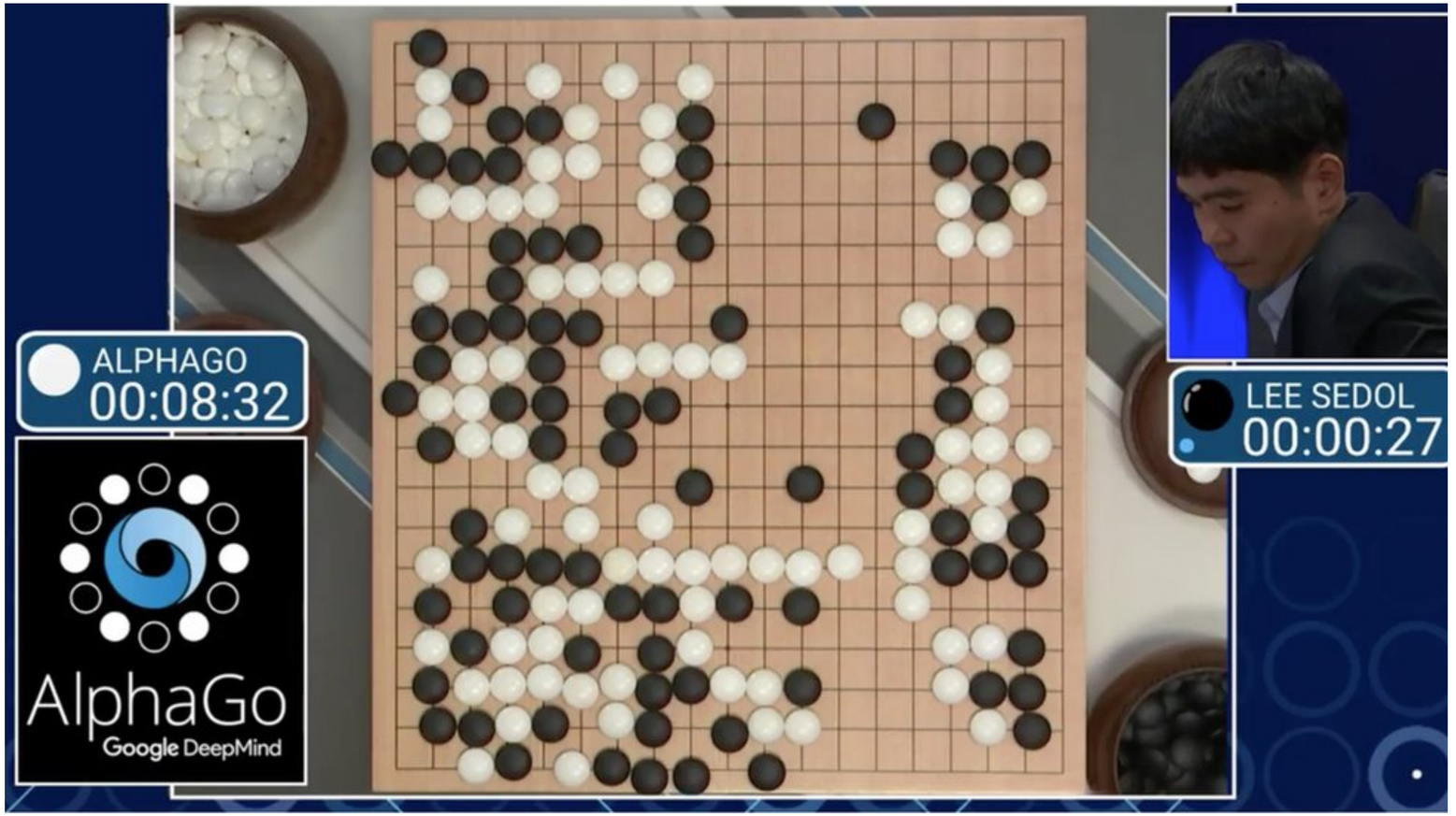
UTILITY(n) *if terminal node*

$\max_{s \in successors(n)}$ EXPECTED-MINIMAX-VALUE(s) *if MAX node*

$\min_{s \in successors(n)}$ EXPECTED-MINIMAX-VALUE(s) *if MIN node*

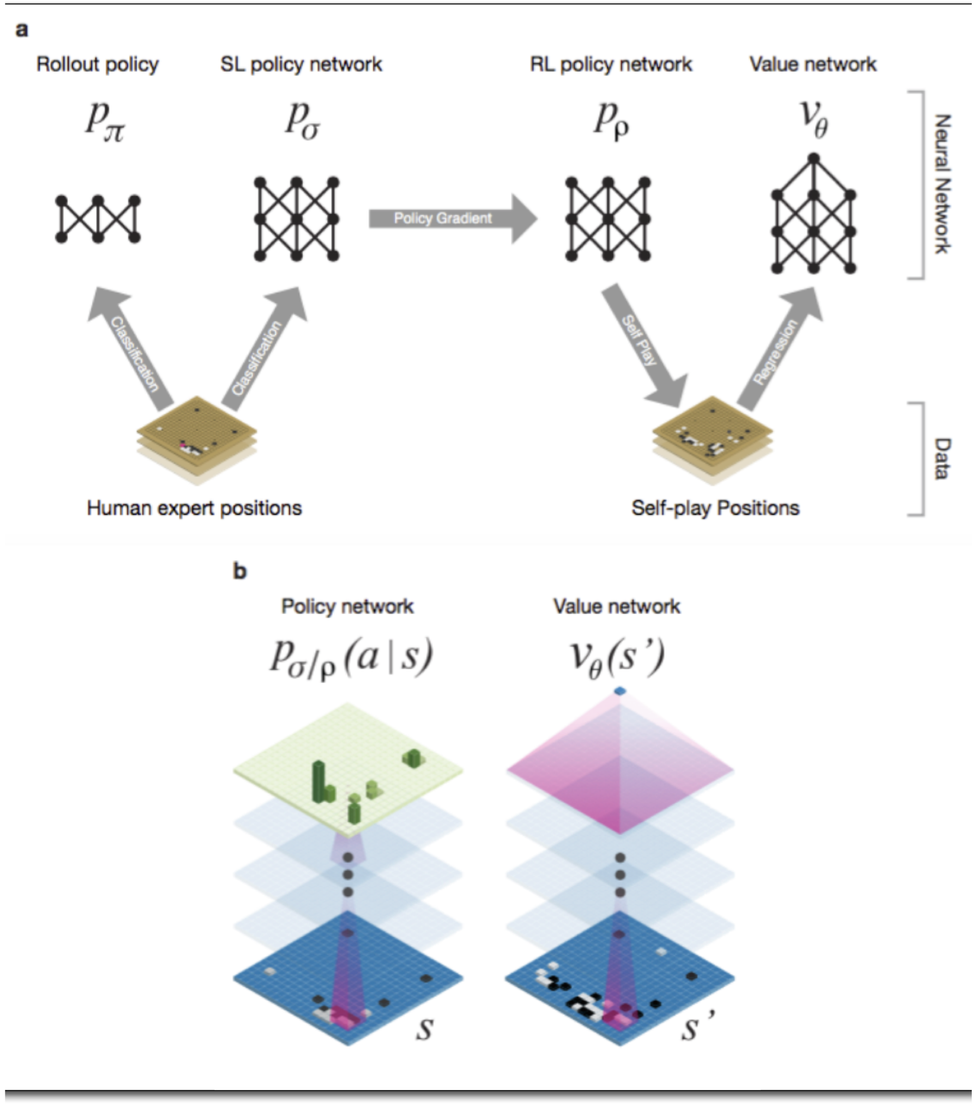
$\sum_{s \in successors(n)}$ P(s)×EXPECTED-MINIMAX-VALUE(s) *if CHANCE node*

APPLICATION: ALPHAGO



HOW DOES ALPHAGO WORK?

- **Challenge 1**: How to evaluate the goodness of a board (i.e., evaluation function of state *s*) **when players play optimally**?
  - Use 30 million positions from human expert games to predict $p_\sigma(a|s)$, i.e., the distributions of human moves at a state *s*, as a neural network
  - Use reinforcement learning to improve the policy to maximize the chance of winning via self-playing; resulting in a better policy $p_\rho(a|s)$
  - Learn value of each state $V_\theta(s)$ as another neural network by starting from 30 million positions and self-playing using policy $p_\rho(a|s)$ and record the end result (i.e., win/lose)



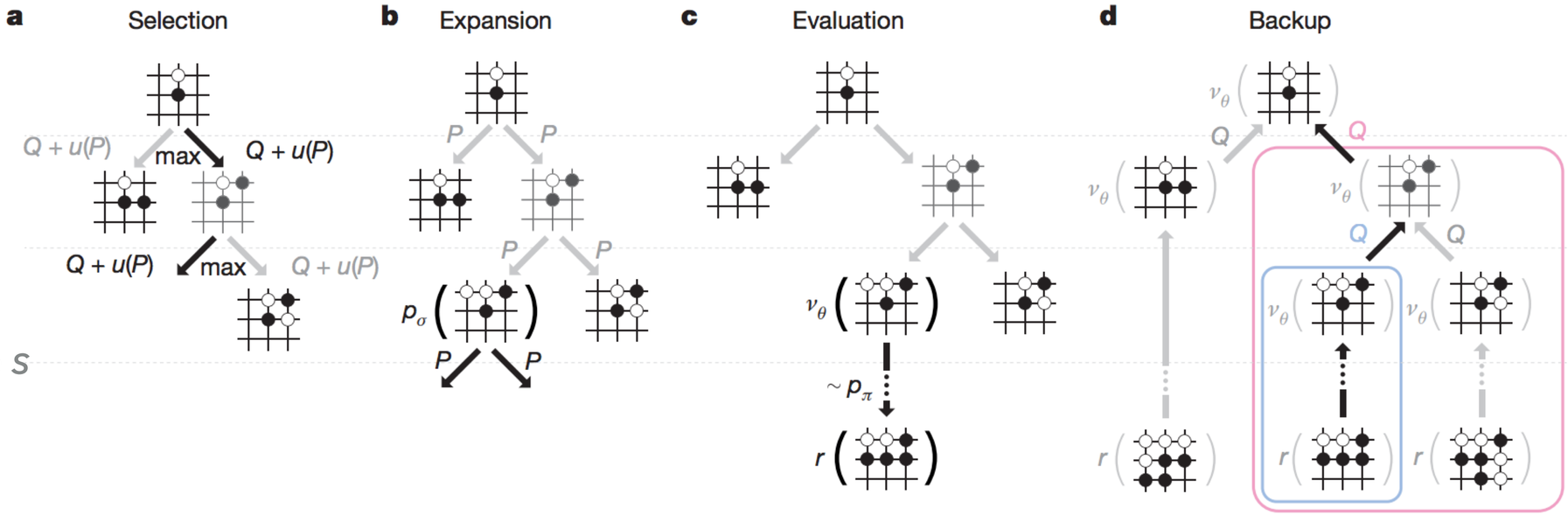
Mastering the game of Go with Deep Neural Networks & Tree Search, *Nature*, 2016

HOW DOES ALPHAGO WORK?

- **Challenge 2**: The branching factor of Go is huge; even a search with several steps is computationally expensive
  - Use Monte Carlo Tree Search (MCTS) to only focus on the "promising" part!

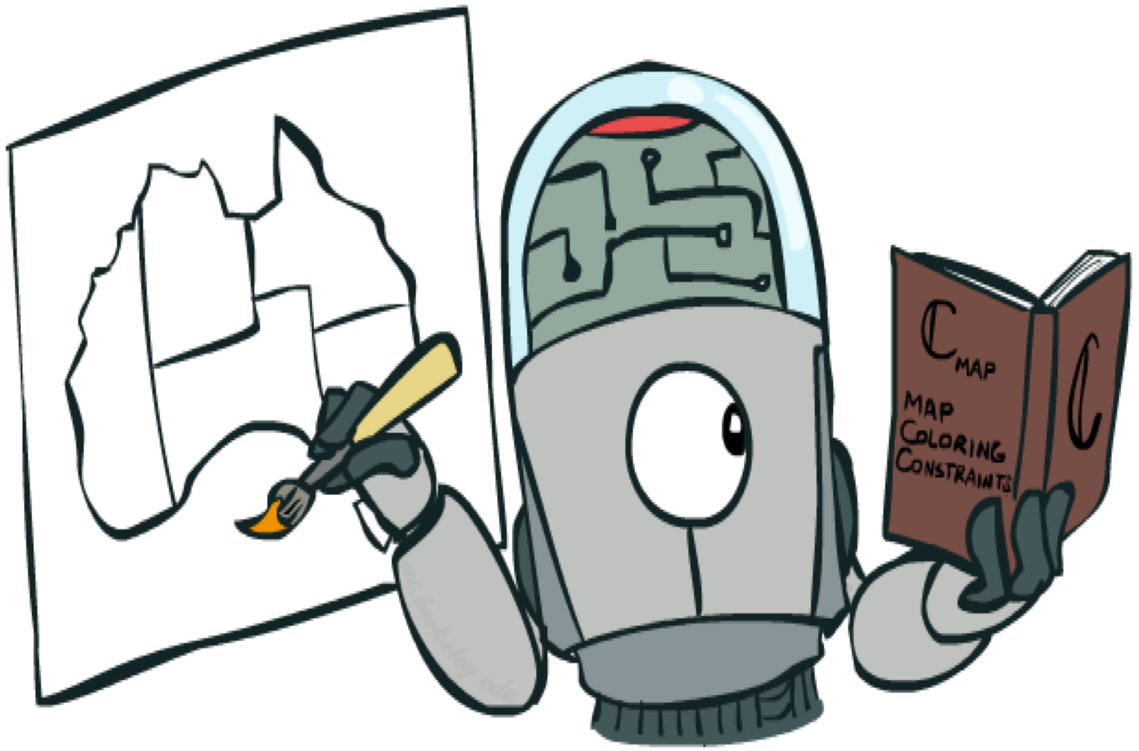
u(s,a) is initialized with $p_\sigma(a|s)$ and decrease with N(s,a);

Q(s,a) reflects the estimated value of taking action *a* at state *s*



Mastering the game of Go with Deep Neural Networks & Tree Search, *Nature*, 2016

CONSTRAINT SATISFACTION PROBLEMS

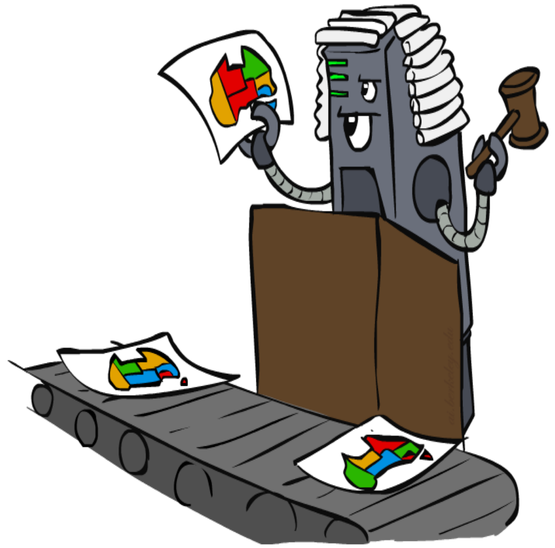


WHAT IS SEARCH FOR?

- **Search/Planning**: sequences of actions
  - The path to the goal is the important thing
  - Paths have various costs, depths
  - Heuristics give problem-specific guidance

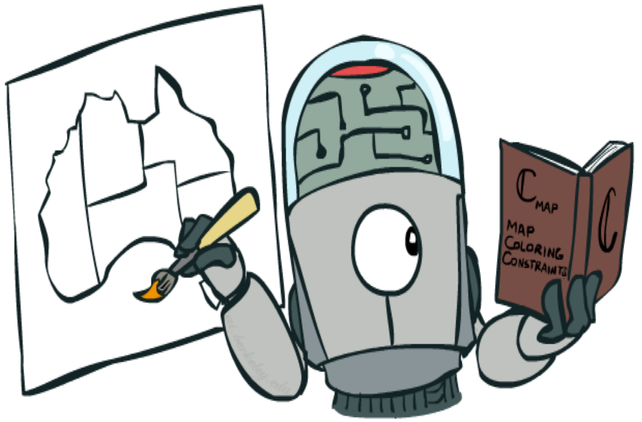
- **Identification**: assignments to variables
  - The goal itself is important, not the path
  - All paths at the same depth (for some formulations)

CONSTRAINT SATISFACTION PROBLEMS

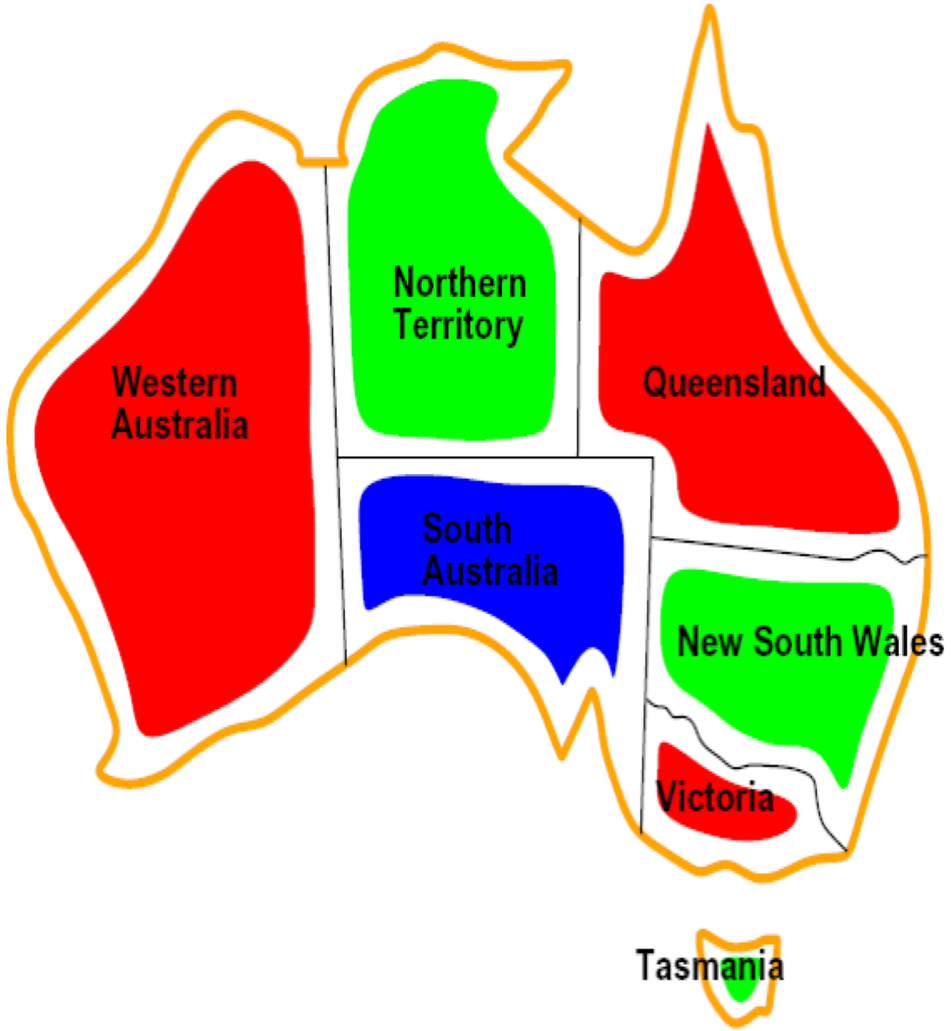


- Standard search problems:
  - State is a “black box”: arbitrary data structure
  - Goal test can be any function over states

- Constraint satisfaction problems (CSPs) – a special subset of search problems
  - State is defined by **variables X_i** with values from a **domain D**
  - Goal test is a set of constraints specifying allowable combinations of values for subsets of variables
- Allows useful general-purpose algorithms with more power than standard search algorithms



CSP EXAMPLES

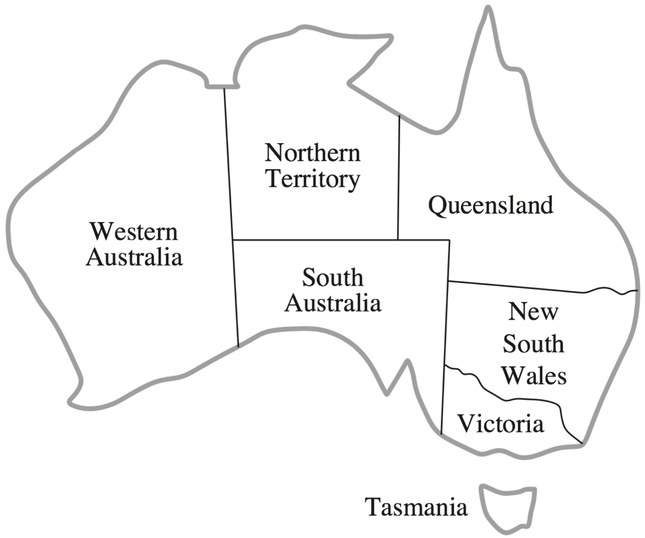


EXAMPLE: MAP COLORING

- Variables: WA, NT, Q, NSW, V, SA, T
- Domains: $D = \{red, green, blue\}$
- Constraints: adjacent regions must have different colors

  Implicit: $WA \neq NT$

  Explicit: $(WA, NT) \in \{(red, green), (red, blue), \ldots\}$



- Solutions are assignments satisfying all constraints, e.g.:

  {WA=red, NT=green, Q=red, NSW=green,
  V=red, SA=blue, T=green}