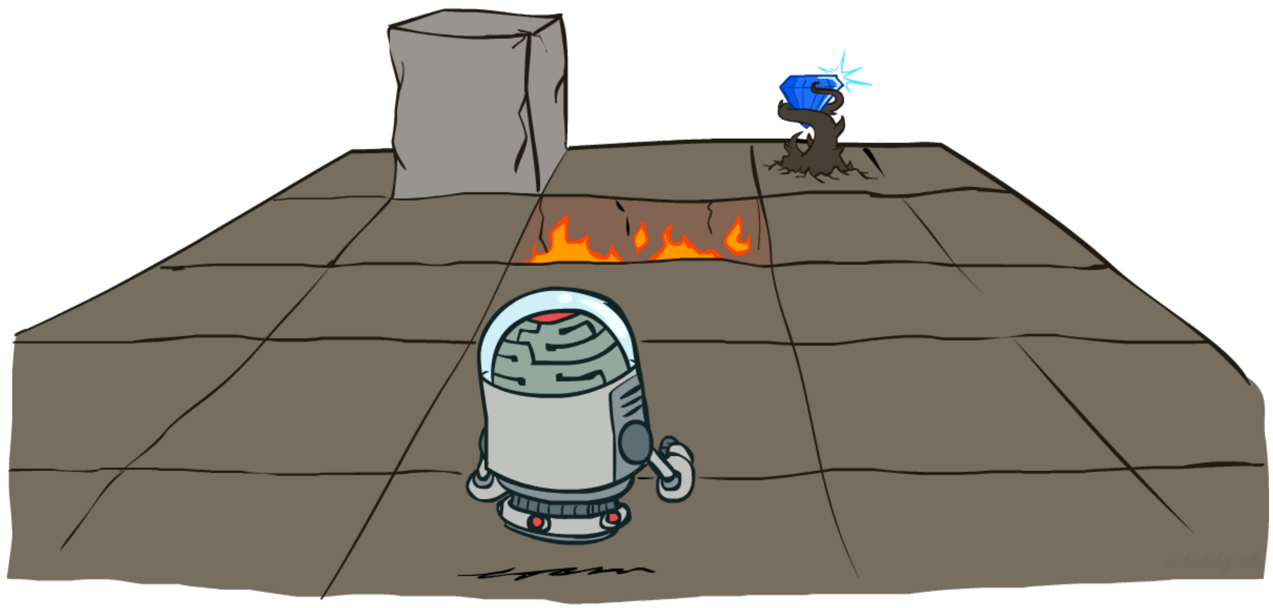
CS 573: Artificial Intelligence

Markov Decision Processes



Dan Weld

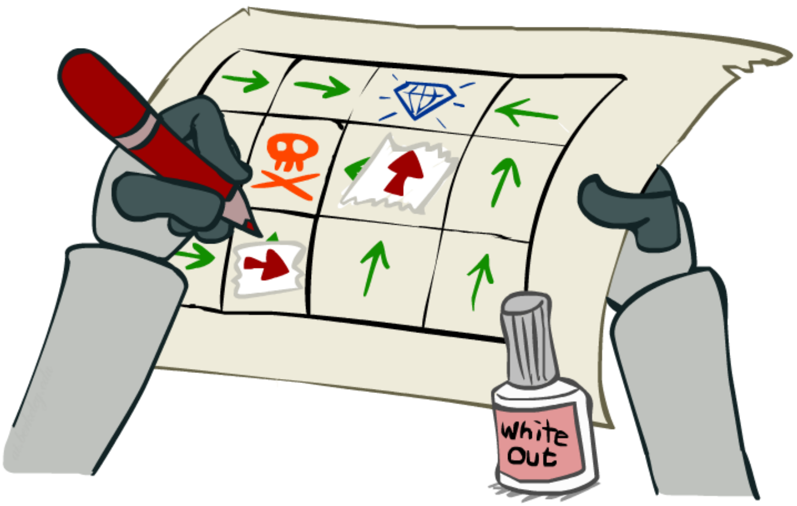
University of Washington

Many slides by Dan Klein & Pieter Abbeel / UC Berkeley. (http://ai.berkeley.edu) and some by Mausam & Andrey Kolobov

Logistics

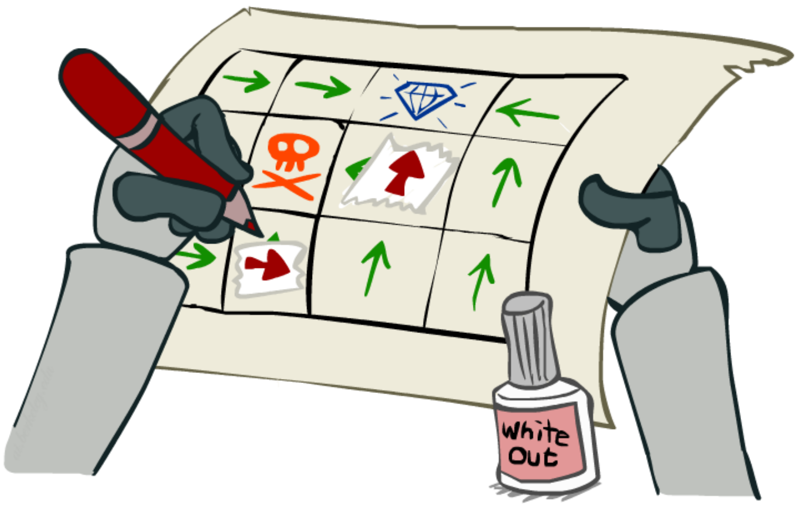
- No class next Tues 2/7
- PS3 – due next wed
- Reinforcement learning starting next Thurs

Solving MDPs



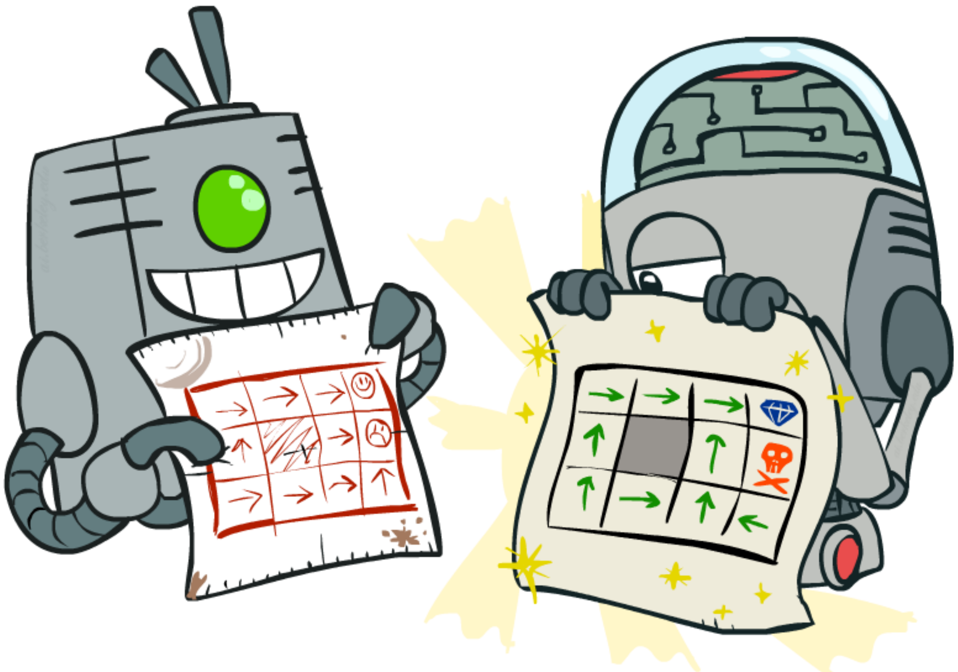
- Value Iteration
- Real-Time Dynamic programming
- Policy Iteration
- Heuristic Search Methods
- Reinforcement Learning

Solving MDPs



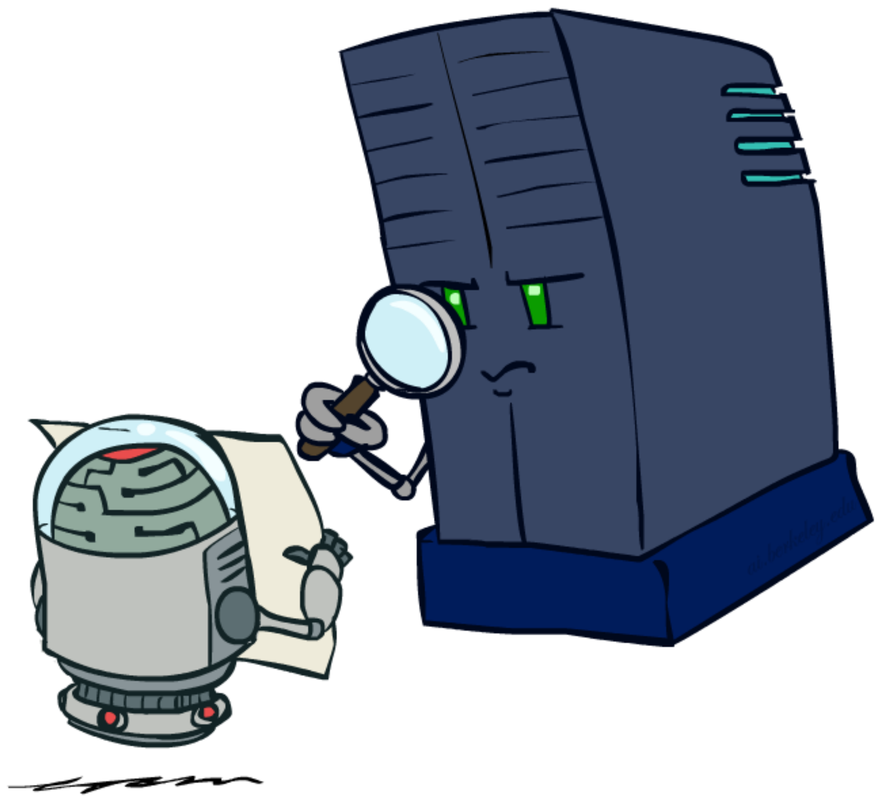
- Value Iteration (IHDR)
- Real-Time Dynamic programming (SSP)
- Policy Iteration (IHDR)
- Heuristic Search Methods (SSP)
- Reinforcement Learning (IHDR)

Policy Iteration



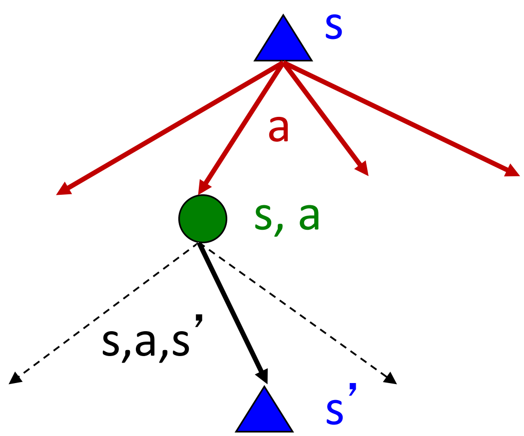
1. Policy Evaluation
2. Policy Improvement

Part 1 - Policy Evaluation

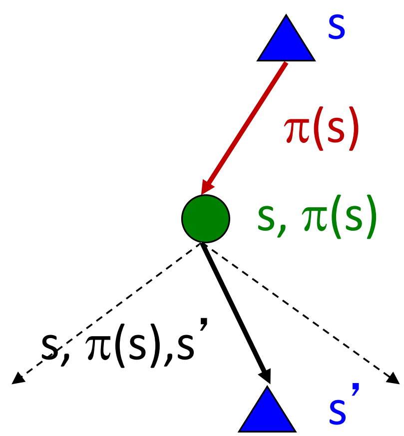


Fixed Policies

Do the optimal action



Do what π says to do



- Expectimax trees max over all actions to compute the optimal values
- If we fixed some policy $\pi(s)$, then the tree would be simpler – only one action per state
  - … though the tree's value would depend on which policy we fixed

Computing Utilities for a Fixed Policy

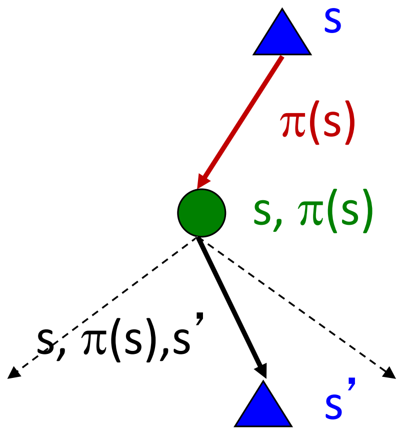
- A new basic operation: compute the utility of a state s under a fixed (generally non-optimal) policy

- Define the utility of a state s, under a fixed policy π:

  $V^\pi(s)$ = expected total discounted rewards starting in s and following π

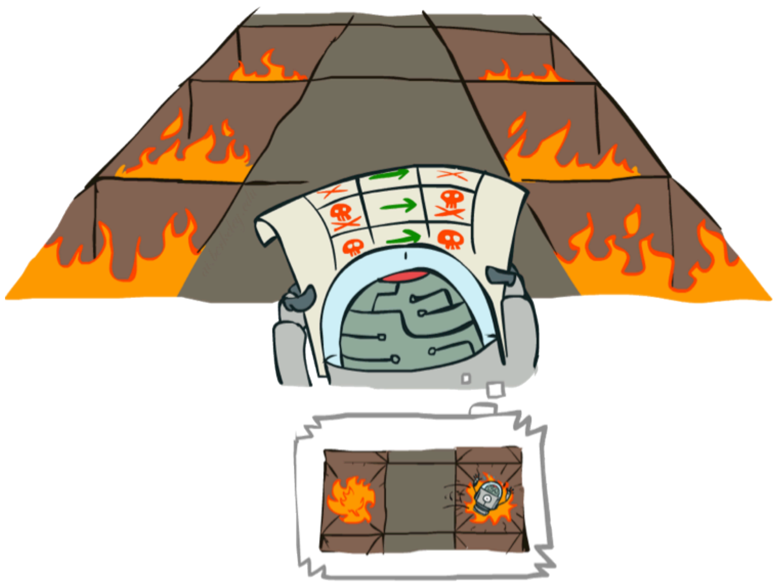
- Recursive relation (variation of Bellman equation):

$$V^\pi(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^\pi(s')]$$

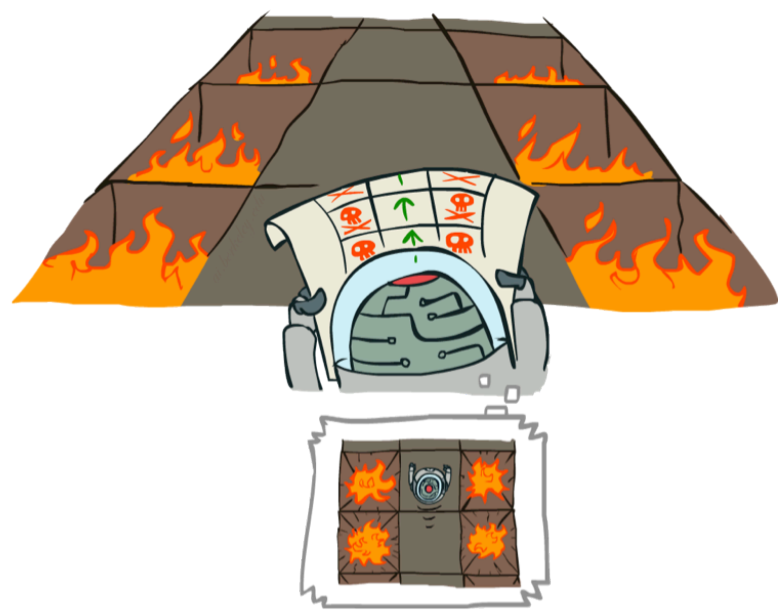


Example: Policy Evaluation

Always Go Right



Always Go Forward



Example: Policy Evaluation

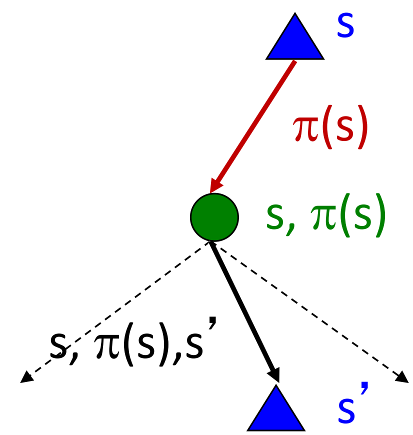
Always Go Right

Always Go Forward

Iterative Policy Evaluation Algorithm

- How do we calculate the V's for a fixed policy π?

- Idea 1: Turn recursive Bellman equations into updates (like value iteration)



$$V_0^\pi(s) = 0$$

$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$

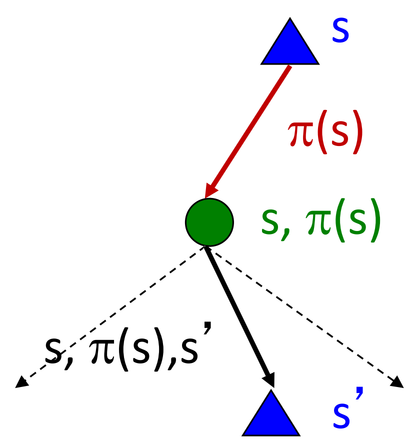
- Efficiency: $O(S^2)$ per iteration
  - Often converges in much smaller number of iterations compared to VI

Linear Policy Evaluation Algorithm

- Another way to calculate the V's for a fixed policy π?

- Idea 2: Without the maxes, the Bellman equations are just a linear system of equations

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$



- Solve with Matlab (or your favorite linear system solver)
  - S equations, S unknowns = $O(S^3)$ and EXACT!
  - In large spaces, still too expensive

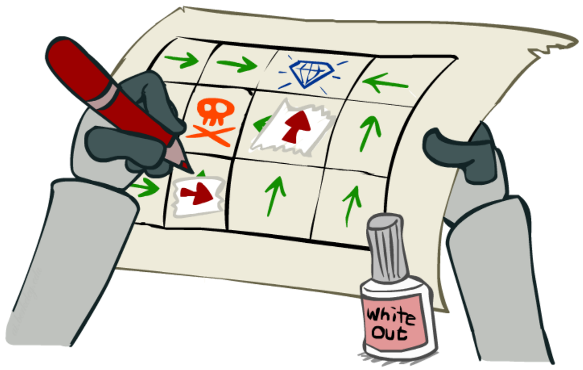
Policy Iteration

- Initialize π(s) to random actions
- Repeat
  - Step 1: Policy evaluation: calculate utilities of π at each s using a nested loop
  - Step 2: Policy improvement: update policy using one-step look-ahead

    For each s, what's the best action to execute, ***assuming agent then follows π?***

    Let π'(s) = this best action.

    π = π'
- Until policy doesn't change



Policy Iteration Details

- Let i =0
- Initialize $\pi_i(s)$ to random actions
- Repeat
  - Step 1: Policy evaluation:
    - Initialize k=0; Forall s, $V_0^\pi(s) = 0$
    - Repeat until V^π converges
      - For each state s, $V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s,\pi_i(s),s') \left[R(s,\pi_i(s),s') + \gamma V_k^{\pi_i}(s')\right]$
      - Let k += 1
  - Step 2: Policy improvement:
    - For each state, s, $\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi_i}(s')\right]$
    - If $\pi_i == \pi_{i+1}$ then it's optimal; return it.
    - Else let i += 1

Example

Initialize π_0 to "always go right"

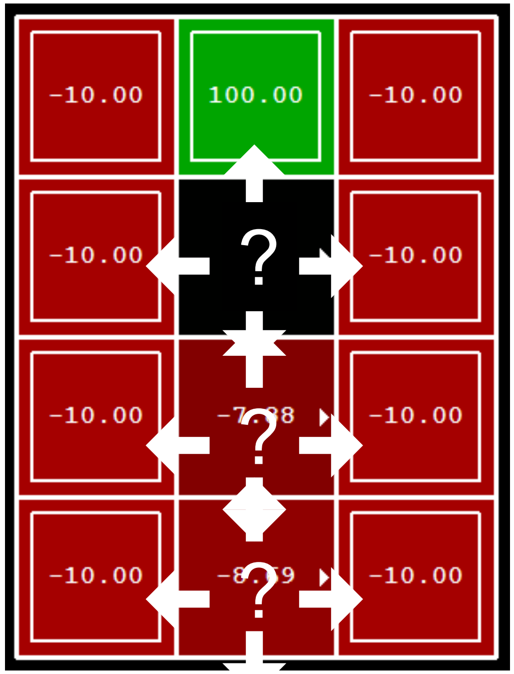
Perform policy evaluation

Perform policy improvement

Iterate through states

Has policy changed?

Yes! i += 1



Example

π_1 says "always go up"

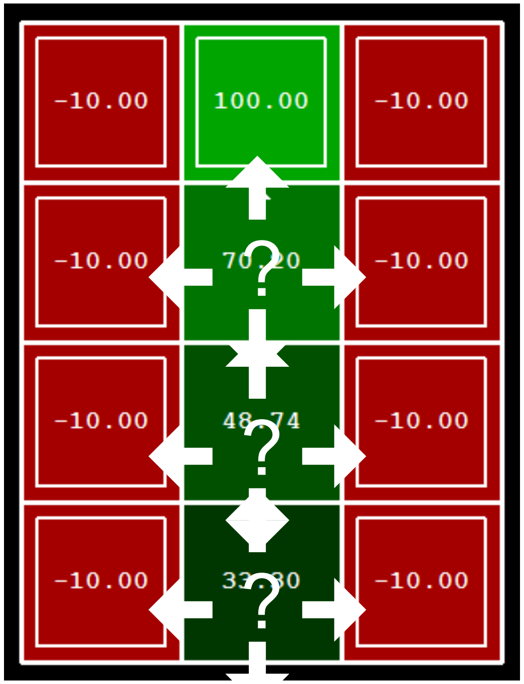
Perform policy evaluation

Perform policy improvement

Iterate through states

Has policy changed?

No! We have the optimal policy



Policy Iteration Properties

- Policy iteration finds the optimal policy, guaranteed (assuming exact evaluation)!
- Often converges (much) faster

Modified Policy Iteration [van Nunen 76]

- initialize π_0 as a random [proper] policy
- Repeat

  Approximate Policy Evaluation: Compute $V^{\pi_{n-1}}$

  by running only few iterations of iterative policy eval.

  Policy Improvement: Construct π_n greedy wrt $V^{\pi_{n-1}}$
- Until convergence
- return π_n

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don't track the policy, but taking the max over actions implicitly recomputes it
  - What is the space being searched?
- In policy iteration:
  - We do fewer iterations
  - Each one is slower (must update all V^{π} and then choose new best π)
  - What is the space being searched?
- Both are dynamic programs for planning in MDPs

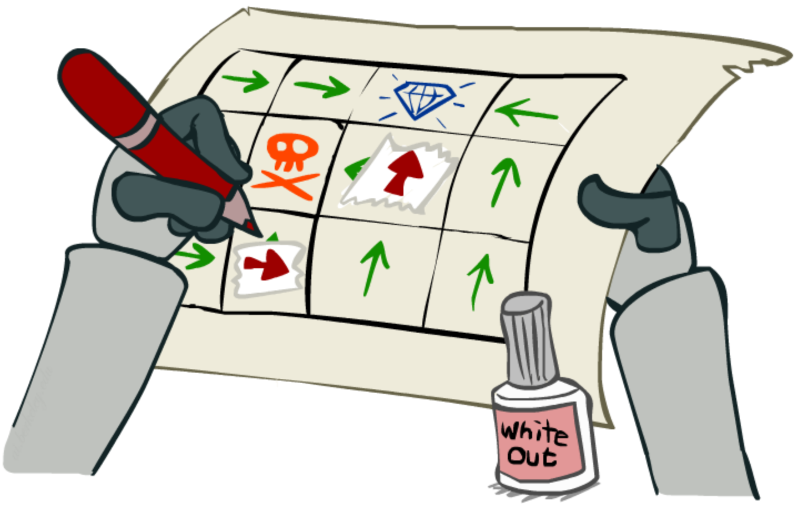
Comparison II

- Changing the search space.

- Policy Iteration
  - Search over policies
  - Compute the resulting value

- Value Iteration
  - Search over values
  - Compute the resulting policy

Solving MDPs



- Value Iteration
- Real-Time Dynamic programming
- Policy Iteration
- Heuristic Search Methods
- Reinforcement Learning