ICAPS Summer School 2020: Probabilistic Planning (MDPs)

Scott Sanner

UNIVERSITY OF TORONTO

Model-based Probabilistic Planning

Mausam
Computer Science and Engineering
Indian Institute of Technology (IIT) Delhi

Planning under Uncertainty

- Definition:

  Computing sequences of actions to obtain occasional rewards in a known, stochastic environment

Applications

Elevator Control

- **Concurrent Actions**
  - Elevator: up/down/stay
  - 6 elevators: 3^6 actions
- **Dynamics**:
  - Random arrivals (e.g., Poisson)
- **Objective:**
  - Minimize total wait
  - (Requires being proactive about future arrivals)
- **Constraints:**
  - People might get annoyed if elevator reverses direction

Two-player Games

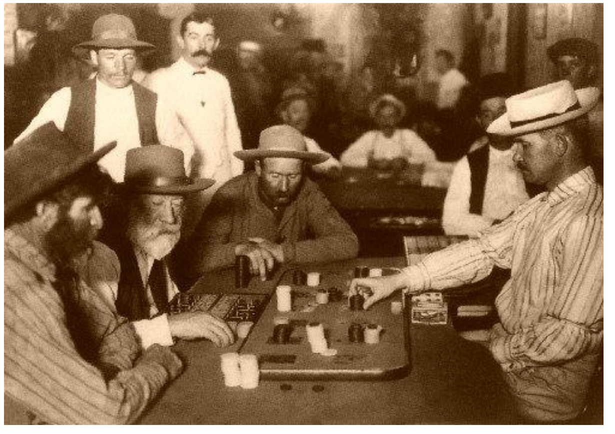
- **Othello / Reversi**
  - Solved by Logistello!
  - Monte Carlo RL (self-play)
    + Logistic regression + Search

- **Backgammon**
  - Solved by TD-Gammon!
  - Temporal Difference (self-play)
    + Artificial Neural Net + Search

- **Go**
  - Learning + Search
  - AlphaGo (MCTS + deep learning)
    recently the world champion

Multi-player Games: Poker

- **Multiagent (adversarial)**
  - Opponent may abruptly change strategy
  - Might prefer best outcome for *any* opponent strategy (e.g, a Nash equilibrium)
- **Multiple rounds (sequential)**
- **Partially observable!**
  - Earlier actions may reveal information
  - Or they may not (bluff)

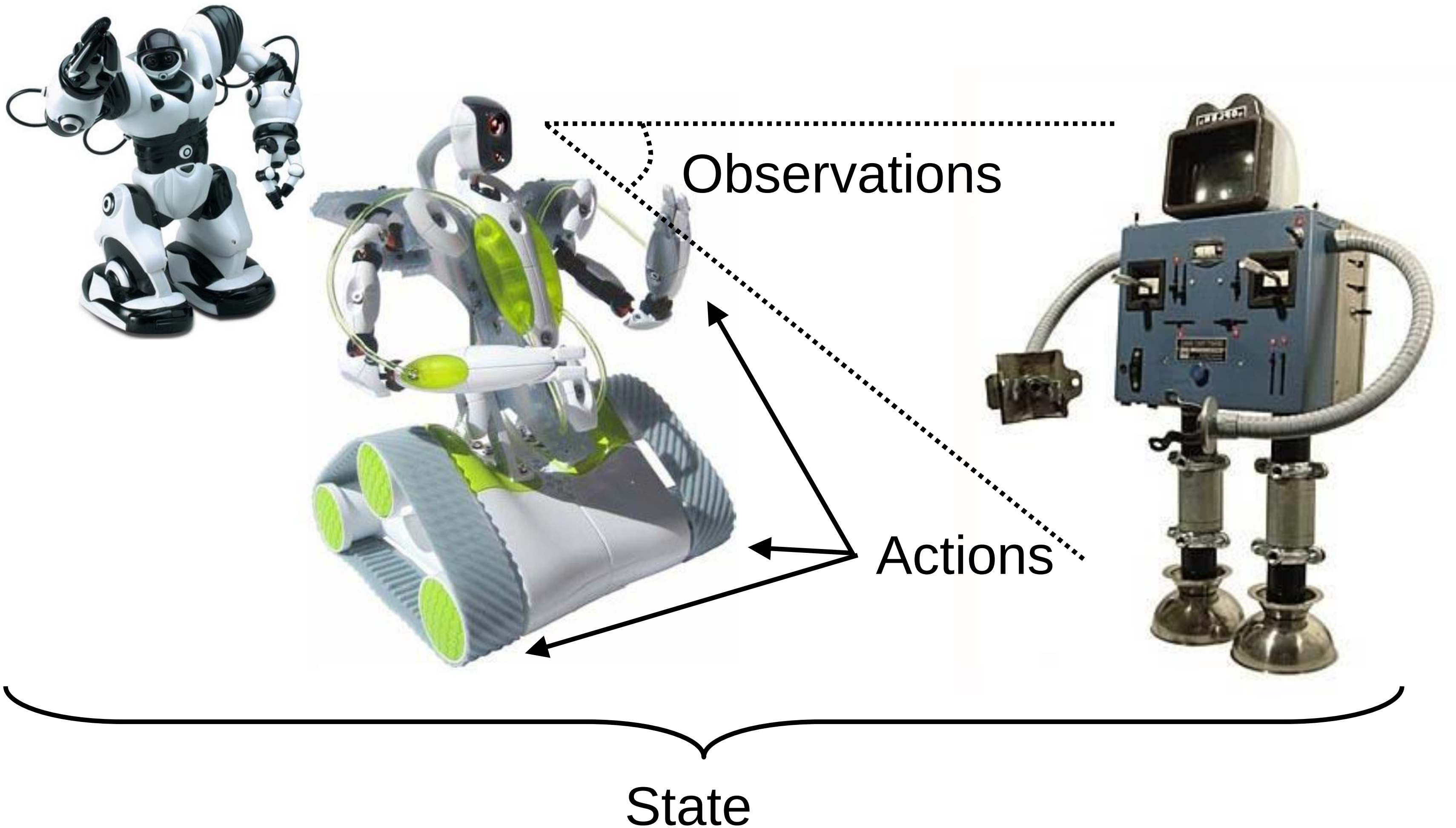


DARPA Grand Challenge

- **Autonomous mobile robotics**
  - Extremely complex task, requires expertise in vision, sensors, real-time operating systems
- **Partially observable**
  - e.g., only get noisy sensor readings
- **Model unknown**
  - e.g., steering response in different terrain


How to model these problems?

Observations, States, & Actions



Observations

- **Observation set O**
  - Perceptions, e.g.,
    - Distance from car to edge of road
    - My opponent’s bet in Poker

States

- **State set S**
  - At any point in time, system is in some state
    - Actual distance to edge of road
    - My opponent’s hand of cards in Poker

Agent Actions

- **Action set A**
  - Actions could be *concurrent*
  - If k actions, $\mathbf{A} = \mathbf{A_1} \times \dots \times \mathbf{A_k}$
    - Schedule all deliveries to be made at 10am

Agent Actions

- **Action set A**
  - All actions need not be under agent control
    - Other agents, e.g.,
      - Alternating turns: Poker, Othello
      - Concurrent turns: Highway Driving, Soccer
    - *Exogenous events* due to *Nature*, e.g.,
      - Random arrival of person waiting for elevator
      - Random failure of equipment
      - If uncontrolled, model as random variables

Observation Function

- How to relate states and observations?
  - ***Not observable*:**
    - **O =** $\emptyset$
    - e.g., heaven vs. hell
      - » only get feedback once you meet St. Pete
  - ***Fully observable*:**
    - **S** $\leftrightarrow$ **O** … the case we focus on!
    - e.g., many board games,
      - » Othello, Backgammon, Go
  - ***Partially observable*:**
    - all remaining cases
    - e.g., driving a car, Poker, the real world!

Recap

- So far
  - Actions
  - States
  - Observations

- How to map between
  - Previous states, actions, and future states?
  - States and observations?
  - States, actions and rewards?
  - Sequences of rewards and optimization criteria?

Transition Function

- How do actions take us between states?
  - $T(s,a,s')$ encodes $P(s'|s,a)$
  - Some properties
    - *Stationary*: T does not change over time
    - *Markovian*: Only depends on previous state / action
    - If T not Markovian or stationary
      - can sometimes achieve by augmenting state description
        - » e.g., elevator traffic differs throughout day…

          encode time in state to make T Markovian!

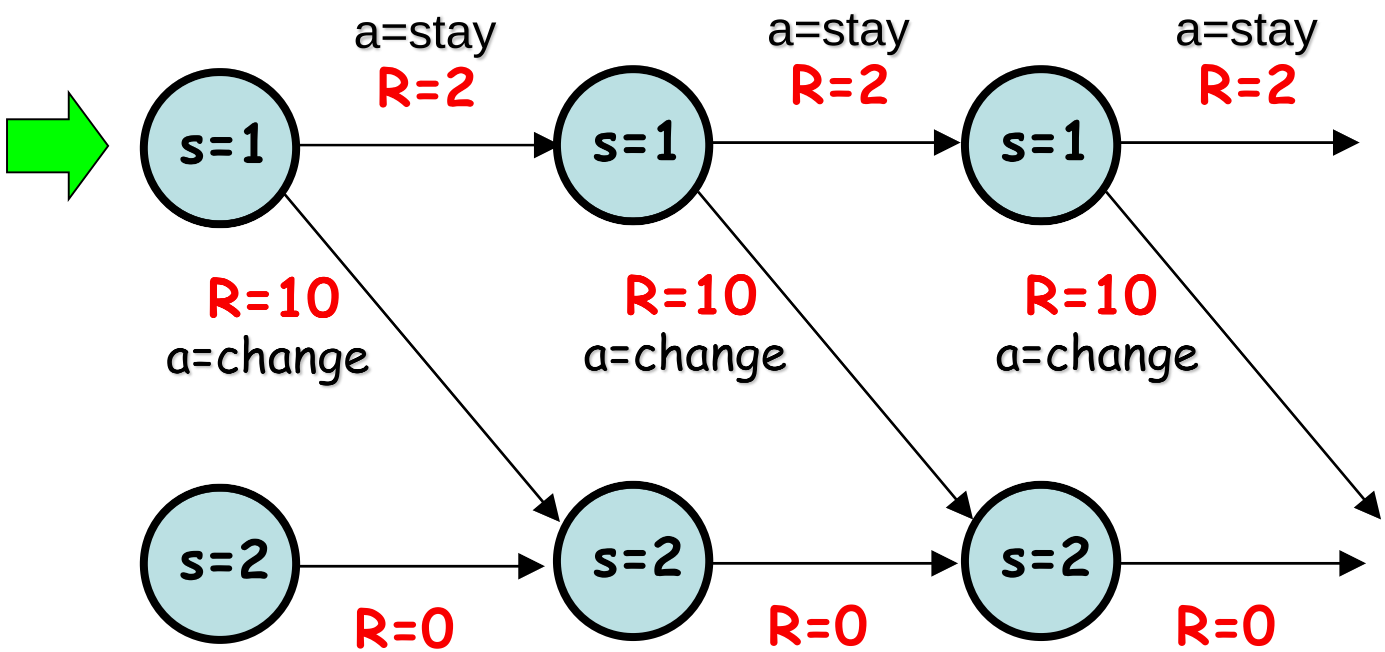
Goals and Rewards

- Goal-oriented rewards
  - Assign any reward value s.t. *R(success) > R(fail)*
  - Can have negative costs *C(a)* for action *a*

- What if multiple (or no) goals?
  - How to specify preferences?
  - *R(s,a)* assigns utilities to each state *s* and action *a*
    - Then *maximize expected reward (utility)*

But, how to trade off rewards over time?

Optimization: Best Action when s=1?



- Must define objective criterion to optimize!
  - How to trade off immediate vs. future reward?
  - E.g., use discount factor γ (try γ=.9 vs. γ=.1)

Trading Off Sequential Rewards

- Sequential-decision making objective
  - Horizon
    - *Finite*: Only care about h-steps into future
    - Infinite: Literally; will act same today as tomorrow
  - How to trade off reward over time?
    - *Expected average cumulative return*
    - *Expected discounted cumulative return*
      - Use discount factor γ
      - Reward t time steps in future discounted by γ^t

Recap

- Model so far
  - Actions A
  - States S
  - Observation O
  - Transition function T: P(s'|s,a)
  - Observation function Z: P(o'|s,a) – *POMDPs only*
  - Reward function: R(s,a)
  - Optimization criteria
- But are the above
  - Known or unknown?

Knowledge of Environment

- ***Model-known*:**
  - Know observation, transition, & reward functions
  - Called: *Planning (under uncertainty)*
    - Planning generally assumed to be goal-oriented
    - *Decision-theoretic* if maximizing expected utility

- ***Model-free*:**
  - ≥1 unknown: observation, transition, & reward functions
  - Called: *Reinforcement learning*
    - Have to interact with environment to obtain samples

- ***Model-based*: approximate model in model-free case**
  - Permits hybrid planning and learning

Saves expensive interaction!

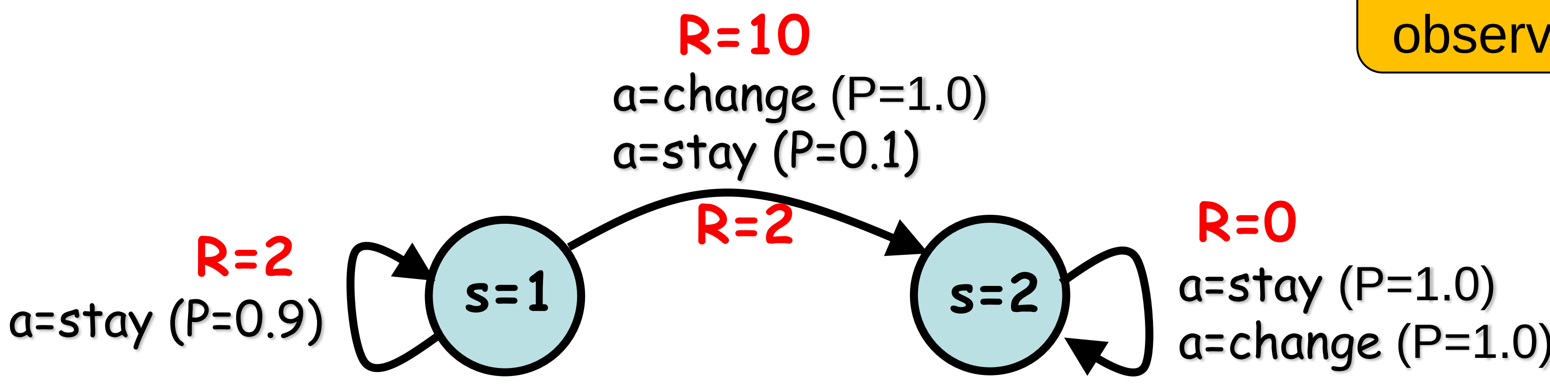
Finally a Formal Model

- **Define the previous model**
  - MDP: $\langle$ **S, A, T, R** $\rangle$
  - POMDP: $\langle$ **S, A, O, Z, T, R** $\rangle$
  - Whether known / unknown
- **Characterize the solutions**
  - And efficiently find them!

Model-based Solutions to MDPs

MDPs ⟨S,A,T,R⟩

Note: fully observable

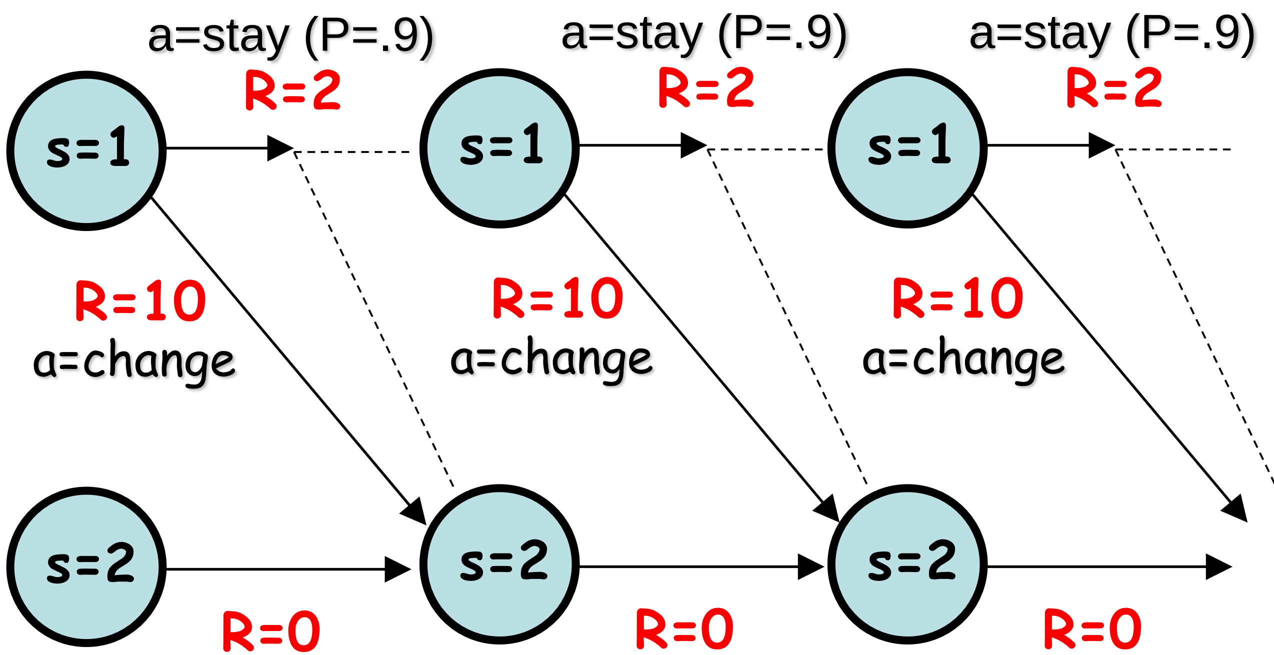


- **S** = {1,2}; **A** = {stay, change}
- **R**eward
  - R(s=1,a=stay) = 2
  - …
- **T**ransitions
  - T(s=1,a=stay,s'=1) = P(s'=1 | s=1, a=stay) = .9
  - …

How to act in an MDP?

Define policy
π: S → A

What's the best Policy?



- Must define reward criterion to optimize!
  - Discount factor γ important (γ=1.0 vs. γ=0.1)

Between Value and Policy Iteration

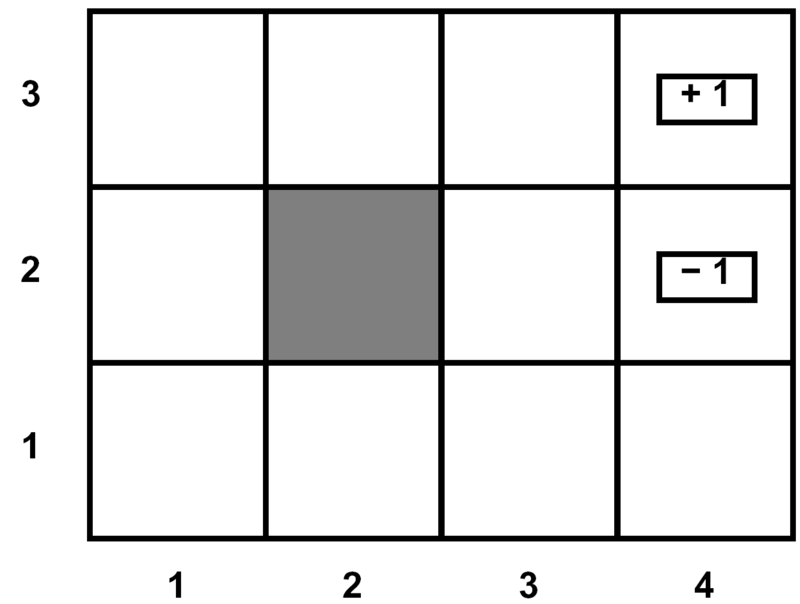
- *Value iteration*
  - Each iteration seen as doing 1-step of policy evaluation for current greedy policy
  - Bootstrap with value estimate of previous policy

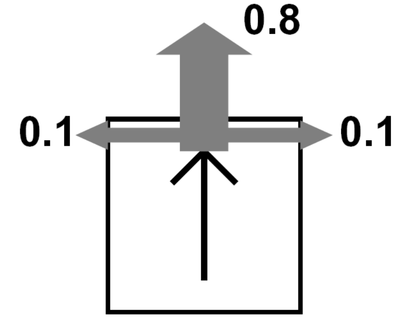
- *Policy iteration*
  - Each iteration is full evaluation of V_π for current policy π
  - Then do greedy policy update

- *Modified policy iteration*
  - Like policy iteration, but $V_{\pi i}$ need only be closer to V^* than $V_{\pi i-1}$
    - Fixed number of steps of successive approximation for $V_{\pi i}$ suffices when bootstrapped with $V_{\pi i-1}$
  - Typically faster than VI & PI in practice

Canonical Example: Grid World

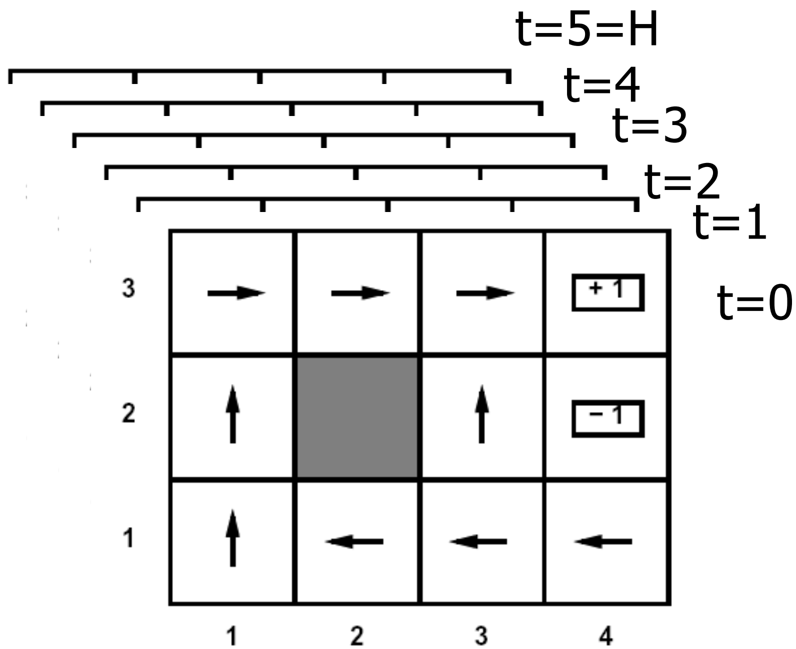
- The agent lives in a grid
- Walls block the agent's path
- The agent's actions do not always go as planned:
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- Big rewards come at the end





Solving MDPs

- In an MDP, we want an optimal policy π^*: S x 0:H → A
  - A policy π gives an action for each state for each time



  - An optimal policy maximizes expected sum of rewards
- Contrast: In deterministic, want an optimal plan, or sequence of actions, from start to a goal

Outline

- Optimal Control

  =

  given an MDP (S, A, T, R, γ, H)

  find the optimal policy π^*

- Exact Methods:
  - ***Value Iteration***
  - Policy Iteration

Value Iteration

- Algorithm:
  - Start with $V_0^*(s) = 0$ for all s.
  - For i=1, … , H

    Given V_i^*, calculate for all states $s \in S$:

$$V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + V_i^*(s')\right]$$

  - This is called a value update or Bellman update/back-up

- $V_i^*(s)$ = the expected sum of rewards accumulated when starting from state s and acting optimally for a horizon of i steps

Value Iteration

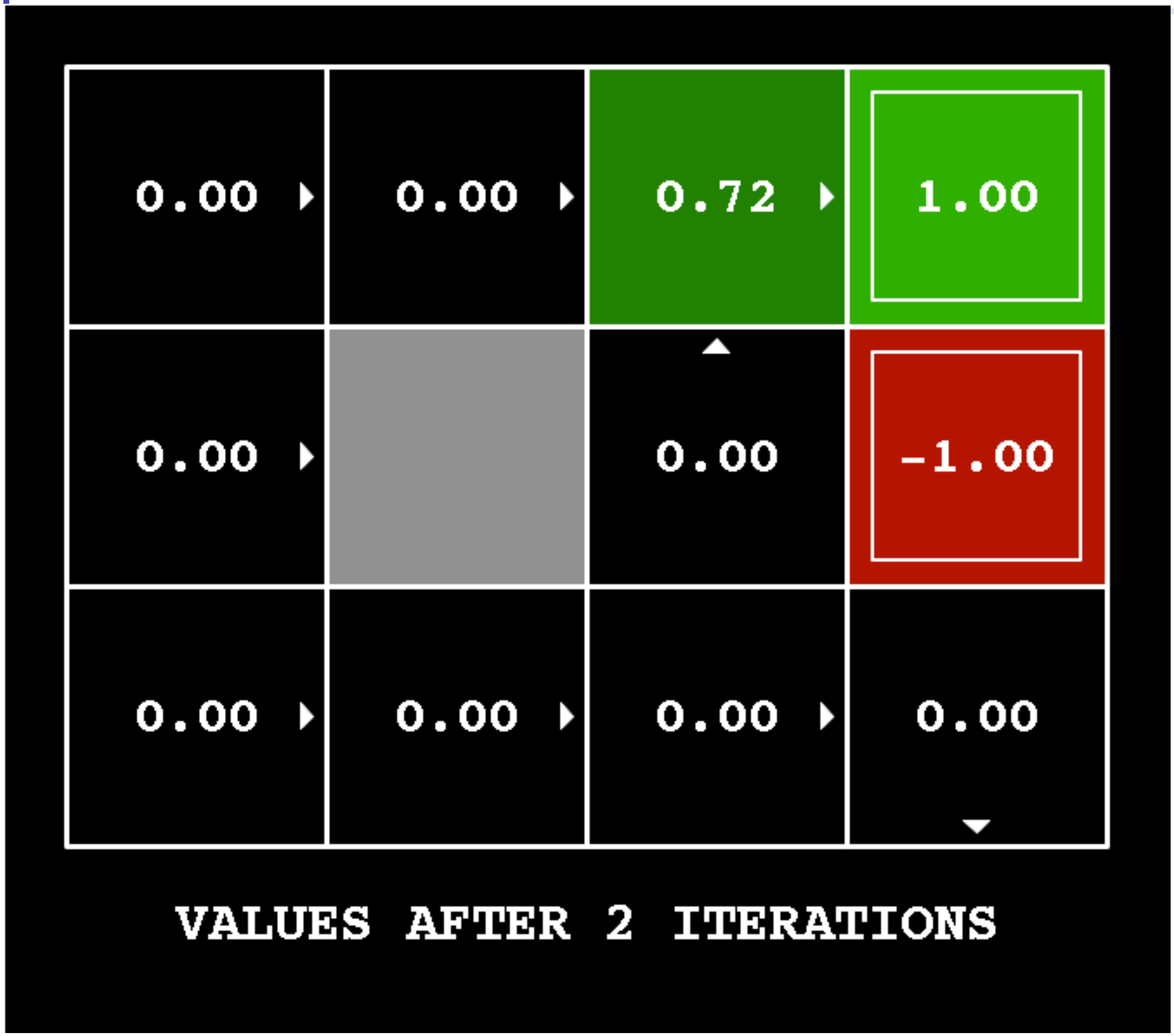
```
values = {each state : 0}
loop ITERATIONS times:
    previous = copy of values
    for all states:
        EVs = {each legal action : 0}
        for all legal actions:
            for each possible next_state:
                EVs[action] += prob * previous[next_state]
        values[state] = reward(state) + discount * max(EVs)
```

Value Iteration in Gridworld

noise = 0.2, γ =0.9, two terminal states with R = +1 and -1

Value Iteration in Gridworld

noise = 0.2, γ =0.9, two terminal states with R = +1 and -1



0.72=1*.8*.9

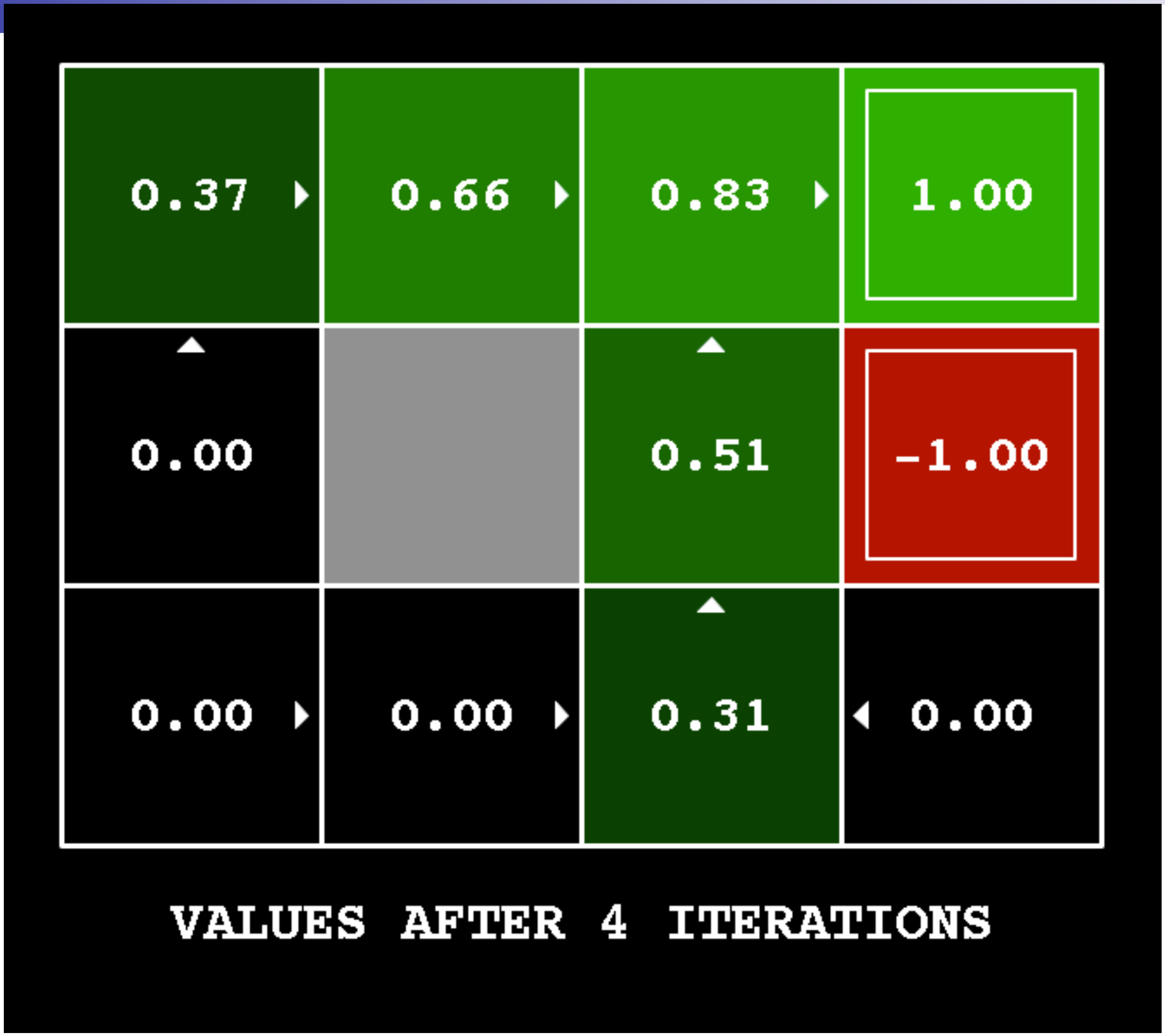
Value Iteration in Gridworld

noise = 0.2, γ =0.9, two terminal states with R = +1 and -1

0.78=0.72+
1*.9*.8*.9

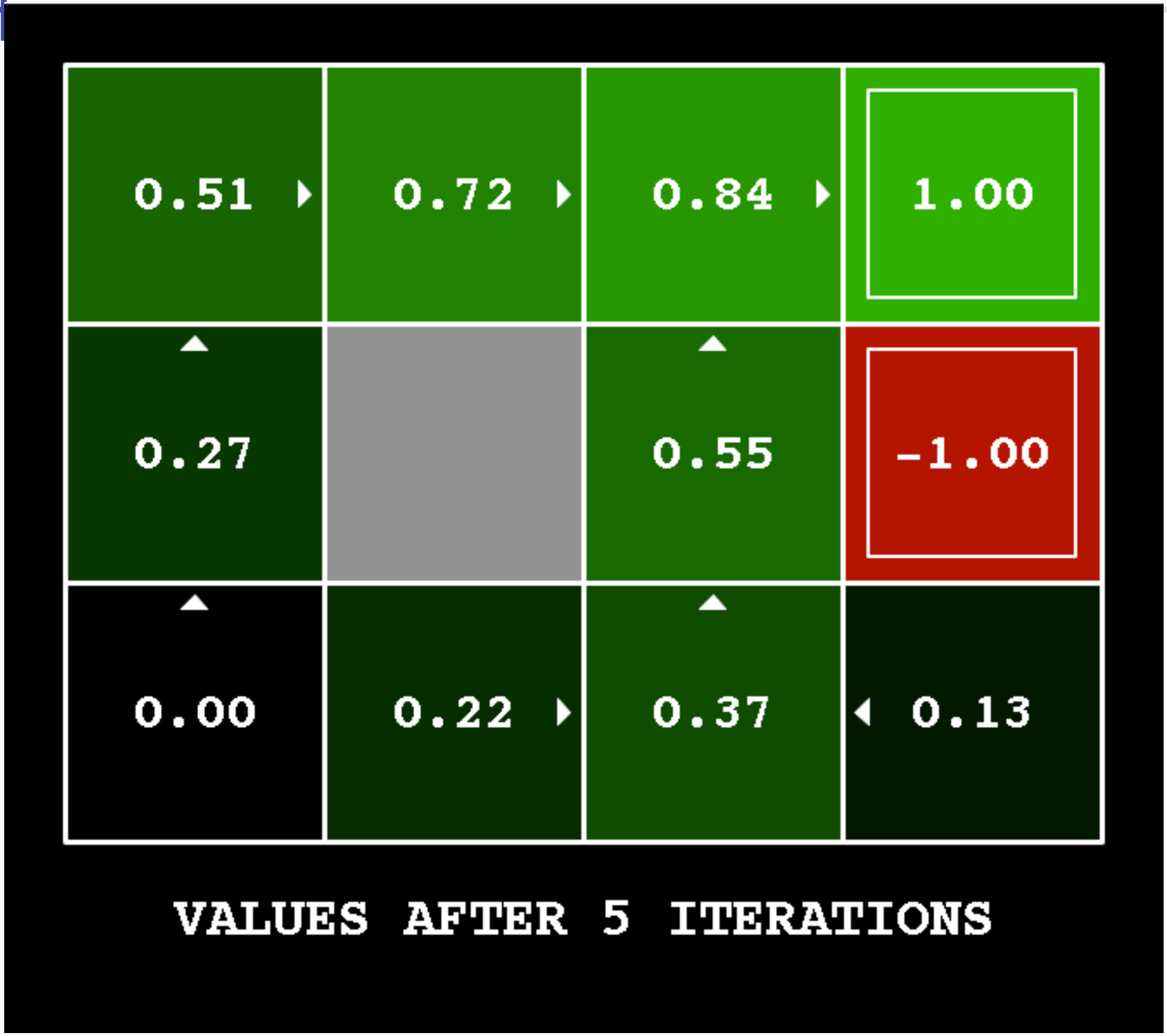
Value Iteration in Gridworld

noise = 0.2, γ =0.9, two terminal states with R = +1 and -1



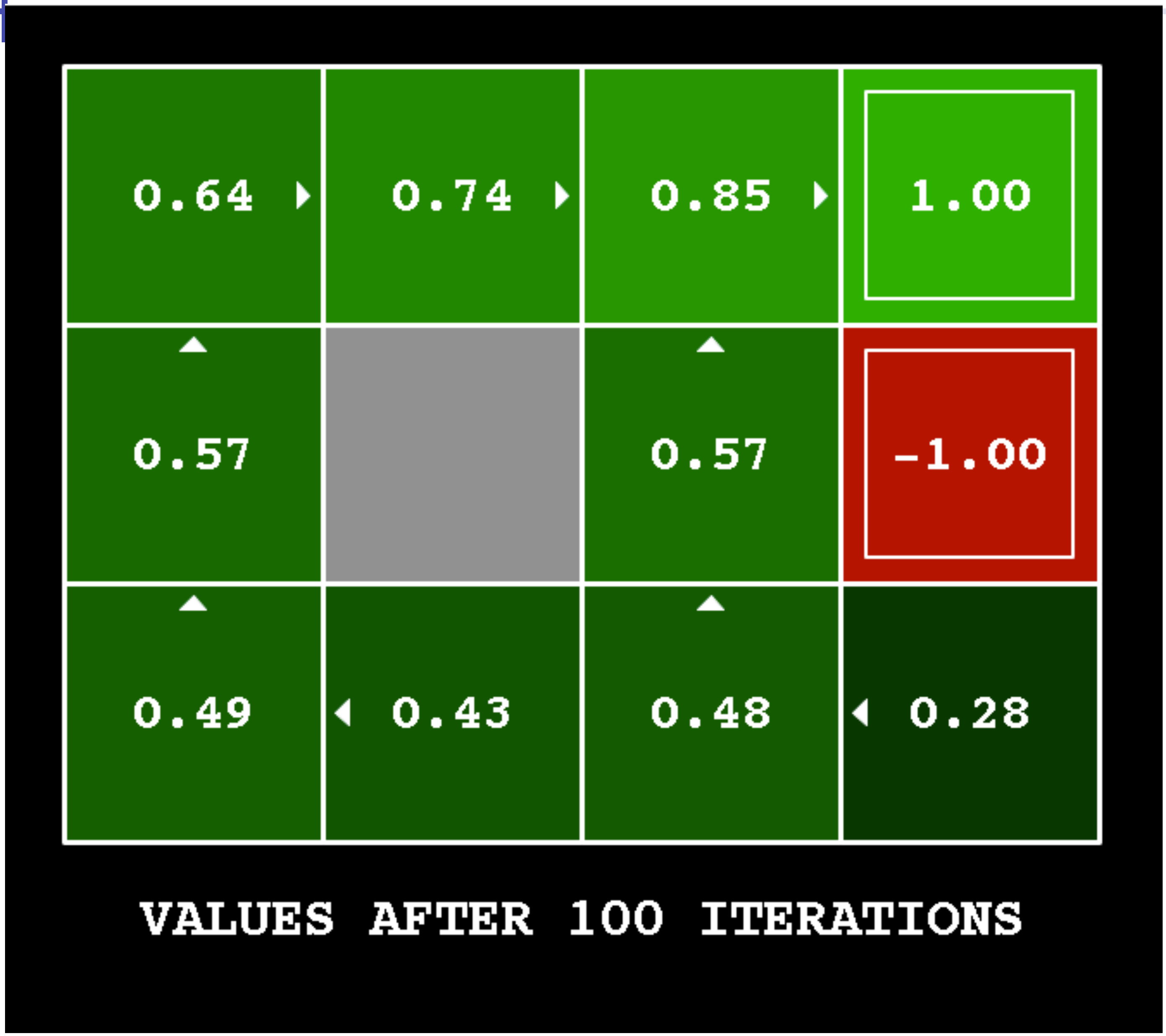
Value Iteration in Gridworld

noise = 0.2, γ =0.9, two terminal states with R = +1 and -1



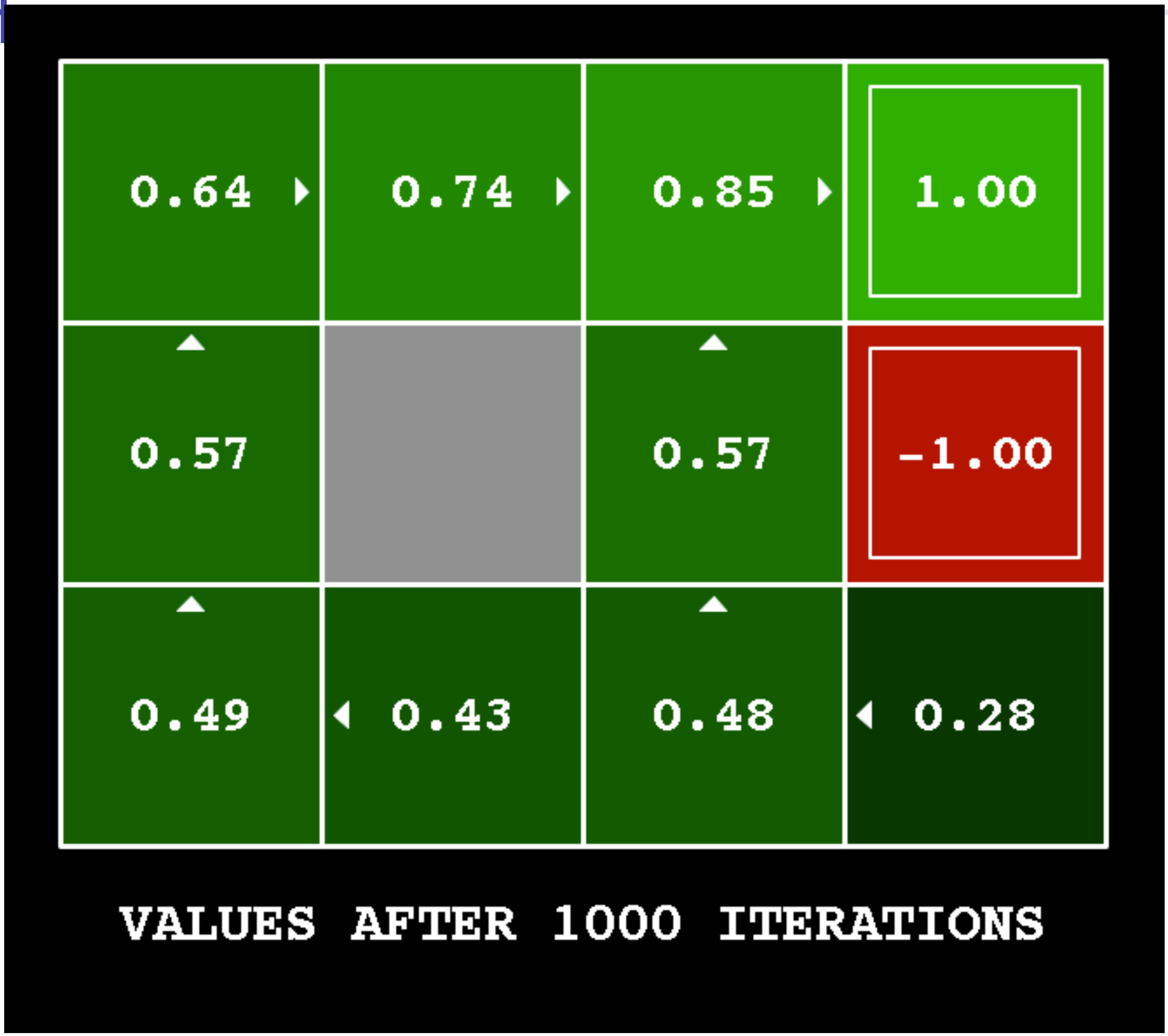
Value Iteration in Gridworld

noise = 0.2, γ =0.9, two terminal states with R = +1 and -1

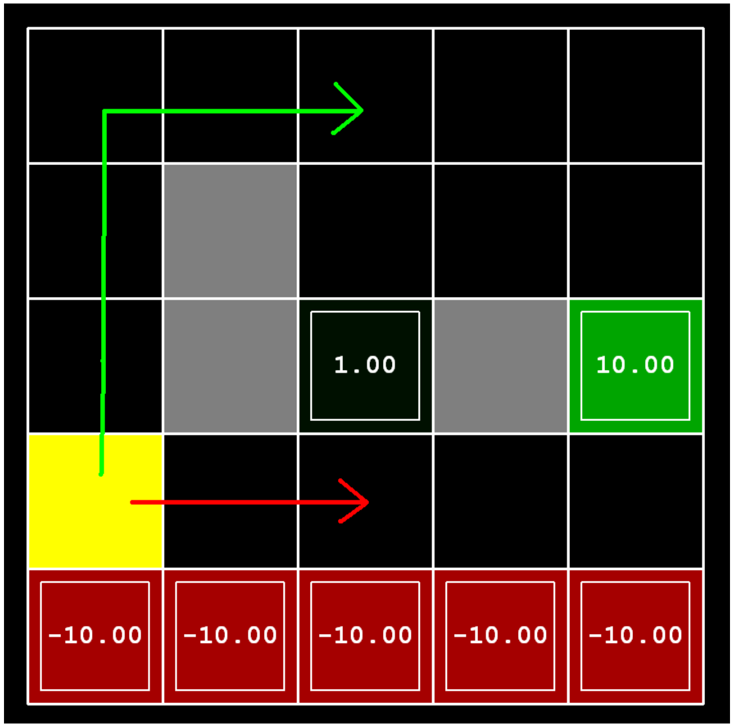


Value Iteration in Gridworld

noise = 0.2, γ =0.9, two terminal states with R = +1 and -1



Exercise 1: Effect of discount, noise



(a) Prefer the close exit (+1), risking the cliff (-10)

(b) Prefer the close exit (+1), but avoiding the cliff (-10)

(c) Prefer the distant exit (+10), risking the cliff (-10)

(d) Prefer the distant exit (+10), avoiding the cliff (-10)

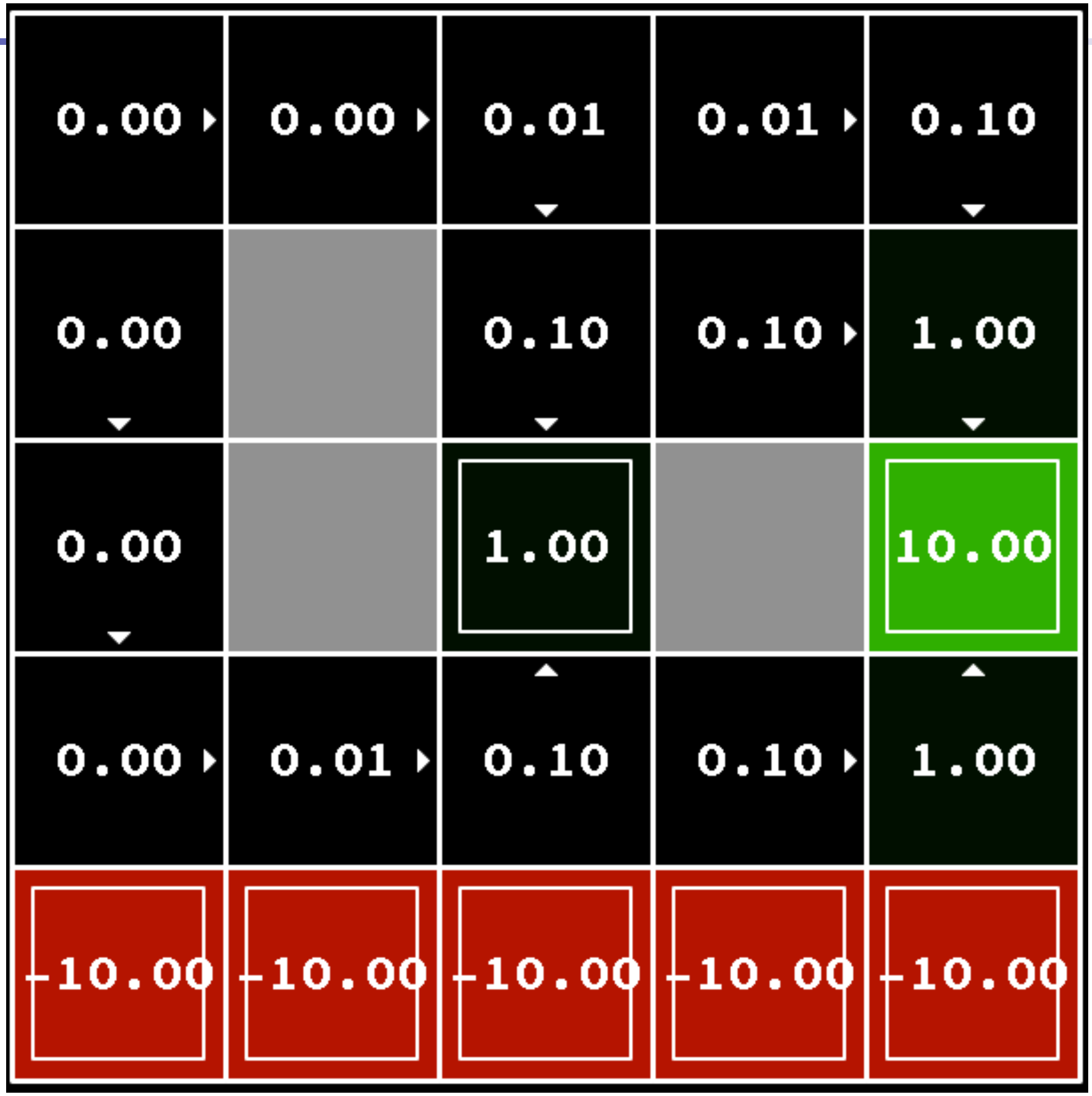
(1) γ = 0.1, noise = 0.5

(2) γ = 0.99, noise = 0

(3) γ = 0.99, noise = 0.5

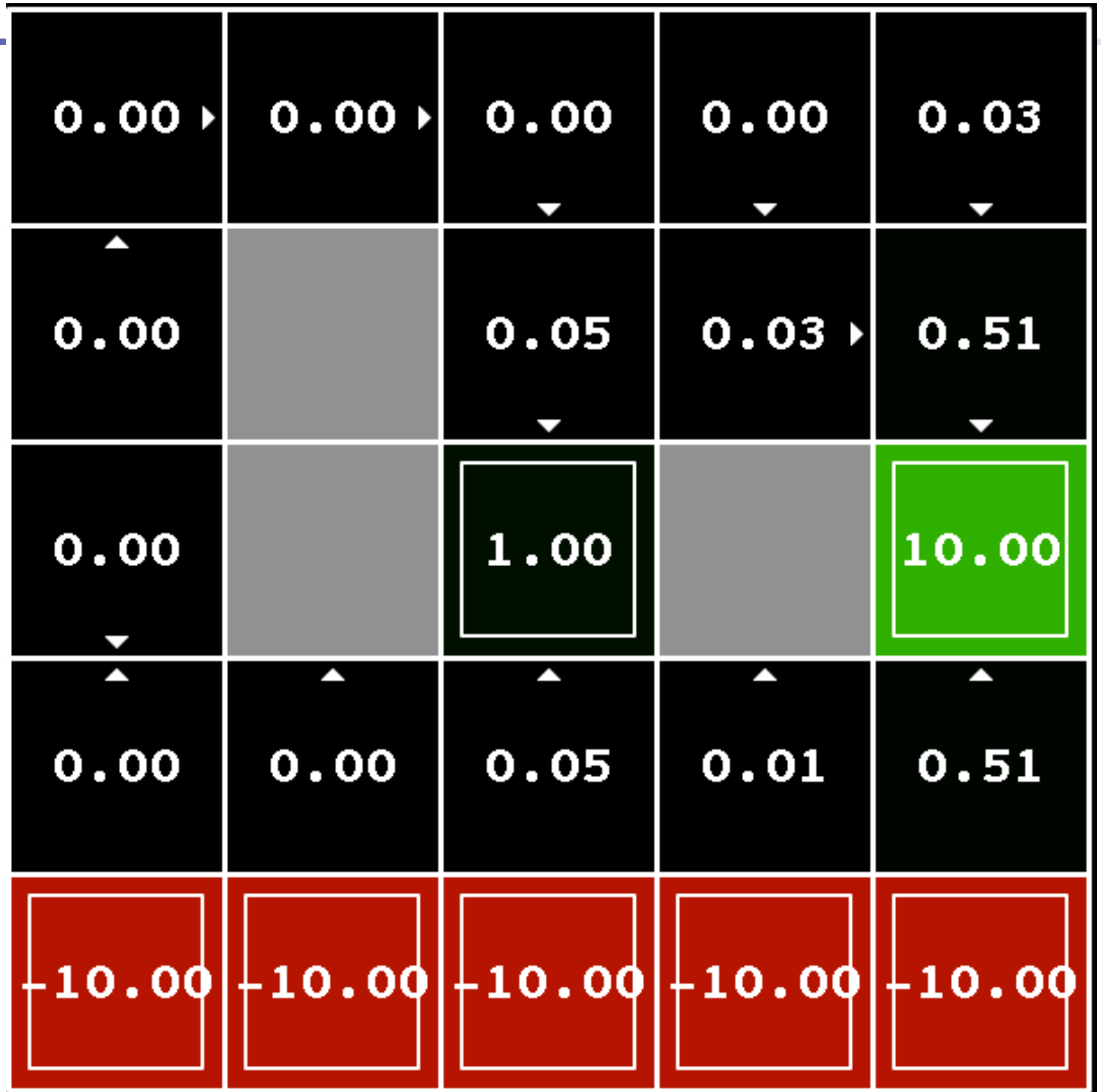
(4) γ = 0.1, noise = 0

Exercise 1 Solution



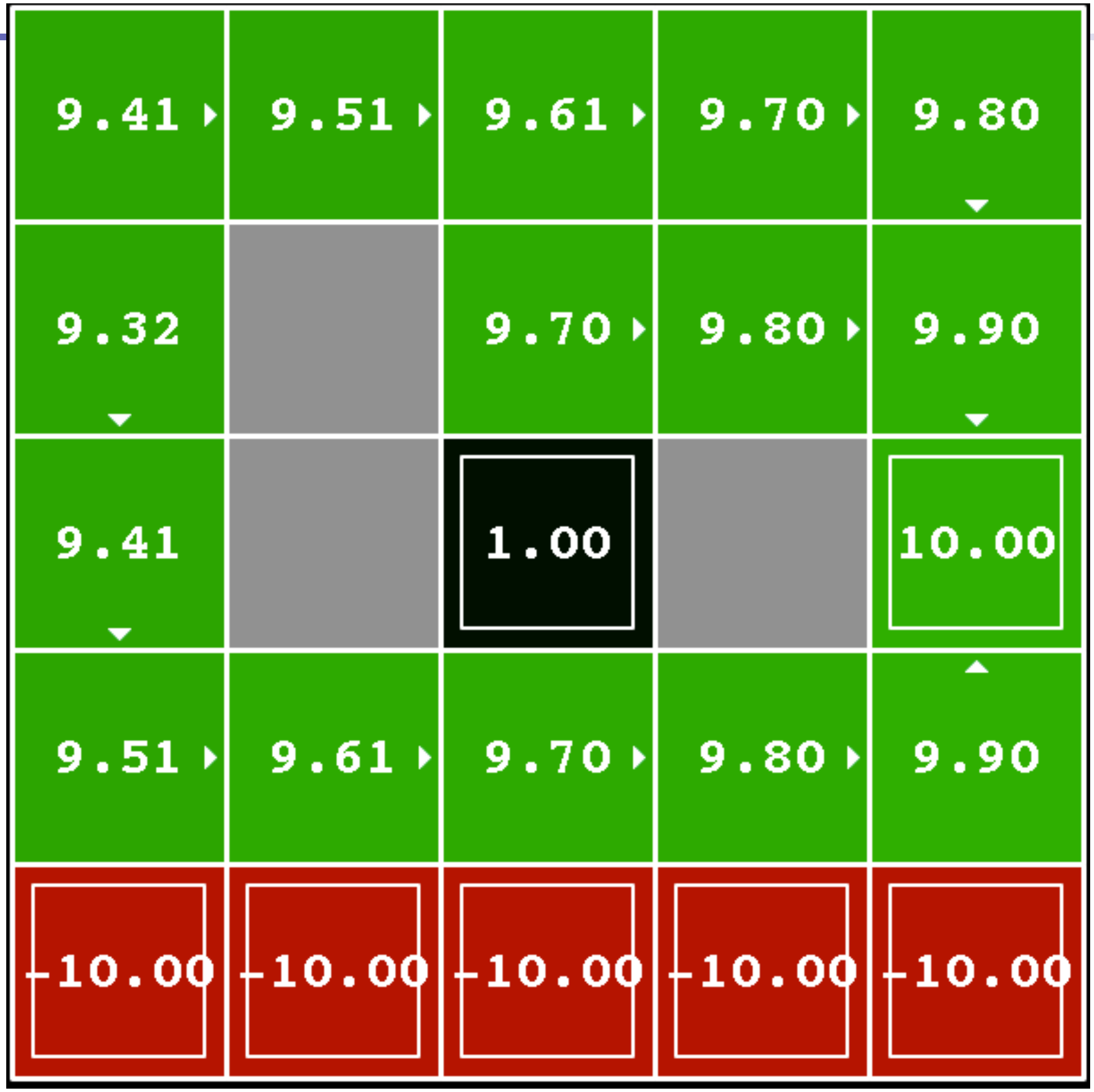
(a) Prefer close exit (+1), risking the cliff (-10) --- $\gamma = 0.1$, noise = 0

Exercise 1 Solution



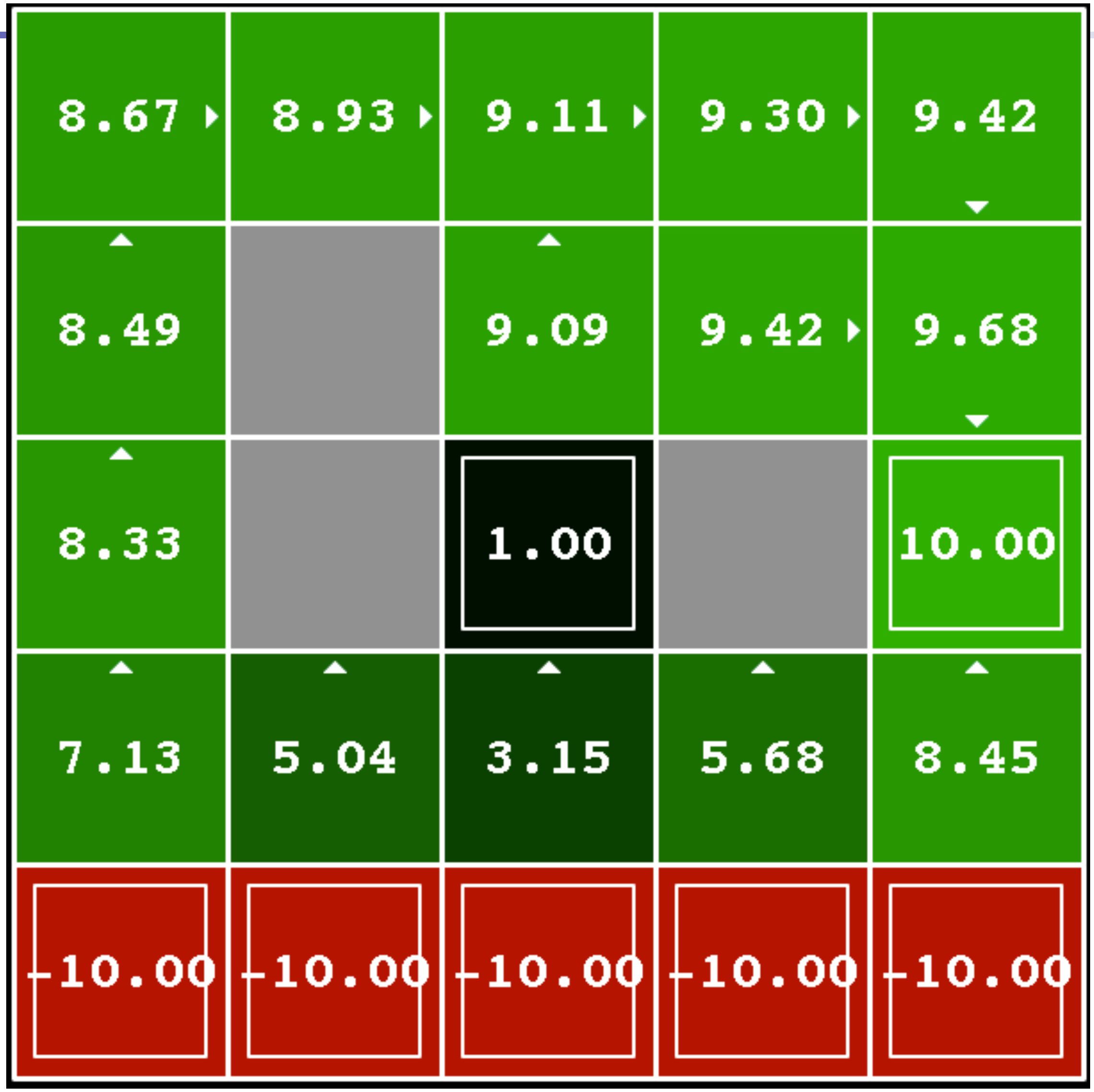
(b) Prefer close exit (+1), avoiding the cliff (-10) -- γ = 0.1, noise = 0.5

Exercise 1 Solution



(c) Prefer distant exit (+1), risking the cliff (-10) -- γ = 0.99, noise = 0

Exercise 1 Solution



(d) Prefer distant exit (+1), avoid the cliff (-10) -- γ = 0.99, noise = 0.5

Value Iteration Convergence

Theorem. Value iteration converges. At convergence, we have found the optimal value function V* for the discounted infinite horizon problem, which satisfies the Bellman equations

$$\forall S \in S: \quad V^*(s) = \max_A \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

- Now we know how to act for infinite horizon with discounted rewards!
  - Run value iteration till convergence.
  - This produces V*, which in turn tells us how to act, namely following:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

- Note: the infinite horizon optimal policy is stationary, i.e., the optimal action at a state s is the same action at all times. (Efficient to store!)

Convergence and Contractions

- Define the max-norm: $||U|| = \max_s |U(s)|$

- Theorem: For any two approximations U and V

$$||U_{i+1} - V_{i+1}|| \leq \gamma \, ||U_i - V_i||$$

  - I.e. any distinct approximations must get closer to each other, so, in particular, any approximation must get closer to the true U and value iteration converges to a unique, stable, optimal solution

- Theorem:

$$||V_{i+1} - V_i|| < \epsilon, \;\Rightarrow\; ||V_{i+1} - V^*|| < 2\epsilon\gamma/(1-\gamma)$$

  - I.e. once the change in our approximation is small, it must also be close to correct

Policy Evaluation

- Recall value iteration iterates:

$$V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i^*(s')]$$

- Policy evaluation:

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

  - At convergence:

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

Policy Iteration

- Alternative approach:
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges

- This is policy iteration
  - It's still optimal!
  - Can converge faster under some conditions

Policy Evaluation Revisited

- *Idea 1:* modify Bellman updates

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

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

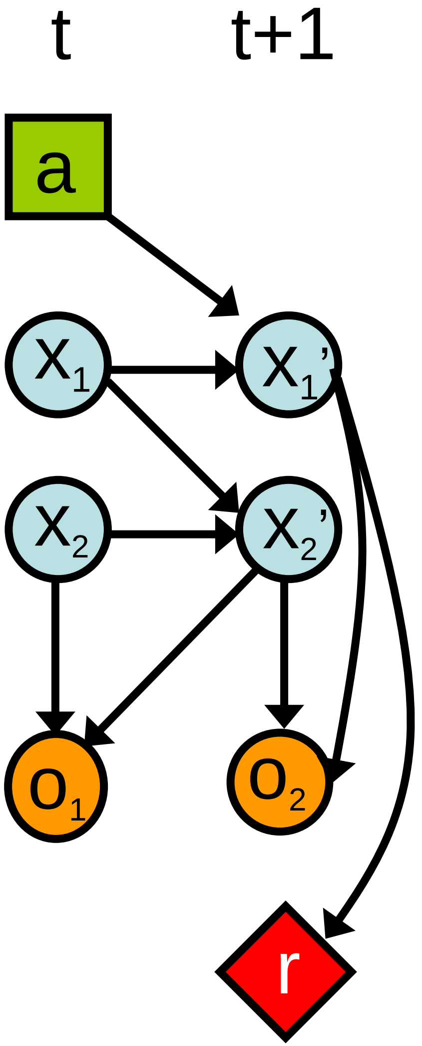
- Idea 2: it's just a linear system, solve with Matlab (or whatever), variables: $V^\pi(s)$, constants: T, R

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

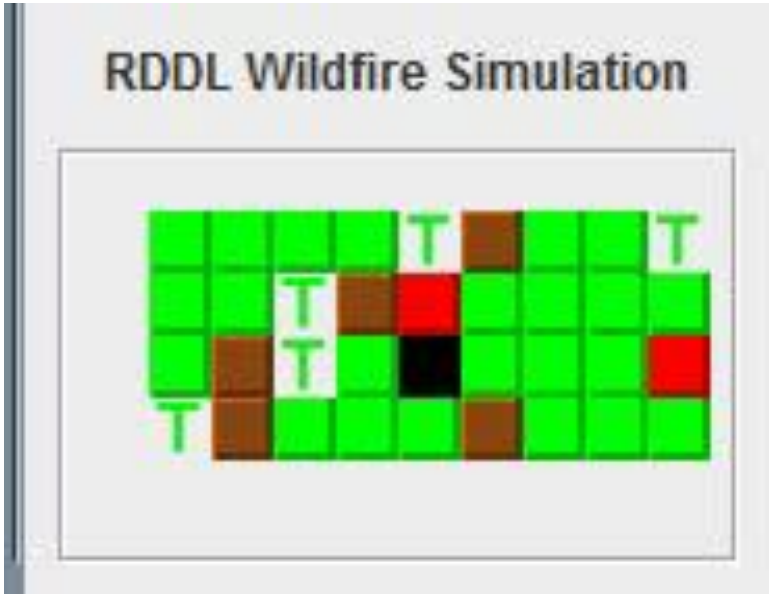
Advanced (PO)MDP Modeling with RDDL

What is RDDL?

- Relational Dynamic Influence Diagram Language
  - Relational [DBN + Influence Diagram]
  - Everything is a fluent!
    - states
    - observations
    - actions
  - Conditional distributions are *probabilistic programs*



Wildfire Domain



- Contributed by Zhenyu Yu (School of Economics and Management, Tongji University)
  - Karafyllidis, I., & Thanailakis, A. (1997). *A model for predicting forest fire spreading using gridular automata*. Ecological Modelling, 99(1), 87-97.

Wildfire in RDDL

```
cpfs {

      burning'(?x, ?y) =
            if ( put-out(?x, ?y) )
                  then false
            else if (~out-of-fuel(?x, ?y) ^ ~burning(?x, ?y))
                  then Bernoulli( 1.0 / (1.0 + exp[4.5 - (sum_{?x2: x_pos, ?y2: y_pos}
                                             (NEIGHBOR(?x, ?y, ?x2, ?y2) ^ burning(?x2, ?y2)))]) )
            else
                  burning(?x, ?y); // State persists

      out-of-fuel'(?x, ?y) = out-of-fuel(?x, ?y) | burning(?x,?y);

};

reward =
     [sum_{?x: x_pos, ?y: y_pos} [ COST_CUTOUT*cut-out(?x, ?y) ]]
   + [sum_{?x: x_pos, ?y: y_pos} [ COST_PUTOUT*put-out(?x, ?y) ]]
   + [sum_{?x: x_pos, ?y: y_pos} [ COST_NONTARGET_BURN*[ burning(?x, ?y) ^ ~TARGET(?x, ?y) ]]]
   + [sum_{?x: x_pos, ?y: y_pos}
          [ COST_TARGET_BURN*[ (burning(?x, ?y) | out-of-fuel(?x, ?y)) ^ TARGET(?x, ?y) ]]];
```

Sanner (2010)

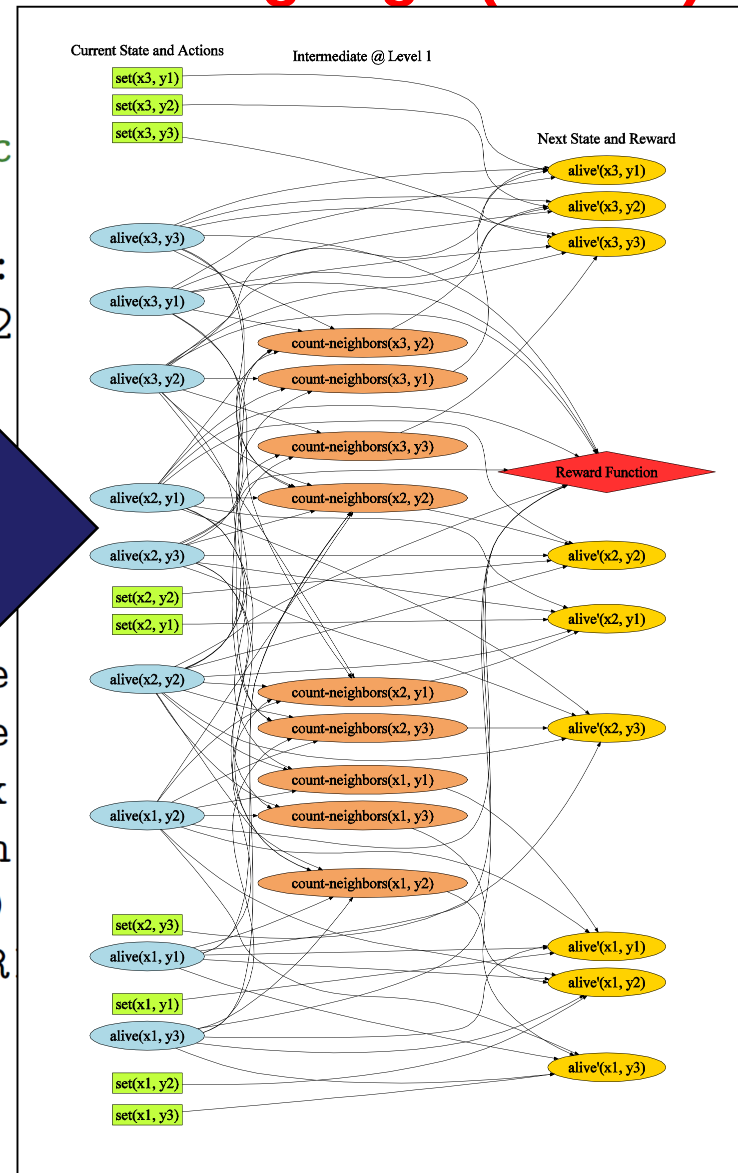
Facilitating Model Development by Writing Simulators: Relational Dynamic Influence Diagram Language (RDDL)

```
// Store alive-neighbor count for eac
count-neighbors(?x,?y) =
    KronDelta(sum_{?x2 : x_pos, ?y2 :
               [NEIGHBOR(?x,?y,?x2,?y2

// Determine whether
alive'(?x,?y) = if ([
                    t
           else
                        ^ (coun    ne
                        ^ (count-ne
                      | [~alive(?x
                         ^ (count-n
                      | set(?x,?y)
          hen Bernoulli(PROB_R
          lse Bernoulli(1.0 -
```

Automatic Translation

Write **probabilistic programs** for transitions



RDDLSim Software

Open source & online at

http://code.google.com/p/rddlsim/

RDDL Software Overview

- BNF grammar and parser
- Simulator
- Automatic compilation / translations
  - LISP-like format (easier to parse)
  - SPUDD & Symbolic Perseus (boolean subset)
  - Ground PPDDL (boolean subset)
- Client / Server
  - Java and C/C++ sample clients
  - Evaluation scripts for log files
- **Visualization**
  - DBN Visualization
  - Domain Visualization – *see* how your planner is doing

Initial Use of RDDL

- Have run two major competitions at ICAPS

- Translations to draw in different communities
  - UAI Factored MDP / POMDP community
  - ICAPS PPDDL community
  - 11 competitors in 2011, 6 competitors in 2014

- Competitions drive research progress!
  - Historically, ICAPS focused on deterministic replanning
  - With RDDL + new domains, **MCTS dominates** (namely PROST system by Thomas Keller *et al*)