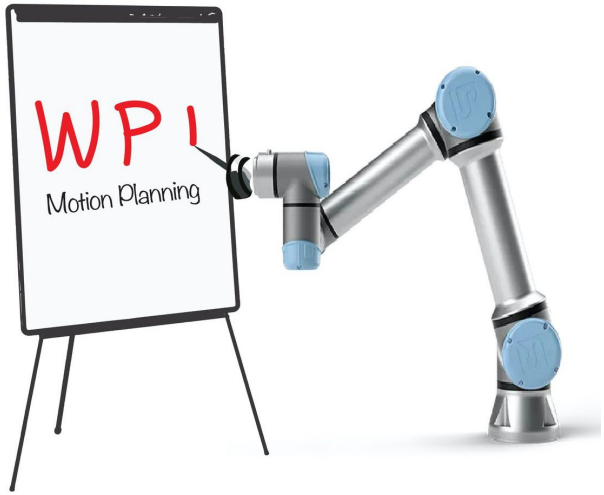
RBE550
Motion Planning
Planning Under Uncertainty



Constantinos Chamzas
www.cchamzas.com
www.elpislab.org

Acknowledgements

The slides are a compilation of work based on notes and slides from Constantinos Chamzas, Lydia Kavraki, Morteza Lahijanian, Ryan Luna

Overview

- Planning Under uncertainty
- Markov Decision Processes
- Value Iteration
- Stochastic motion roadmaps
- Partially Observable Decision Processes

Motivation

- Sand Flea Jumping Robot
  - 11-lb robot
  - Can jump 30 feet into the air
  - Onboard stabilization system keeps it oriented during flight

Source of Uncertainty

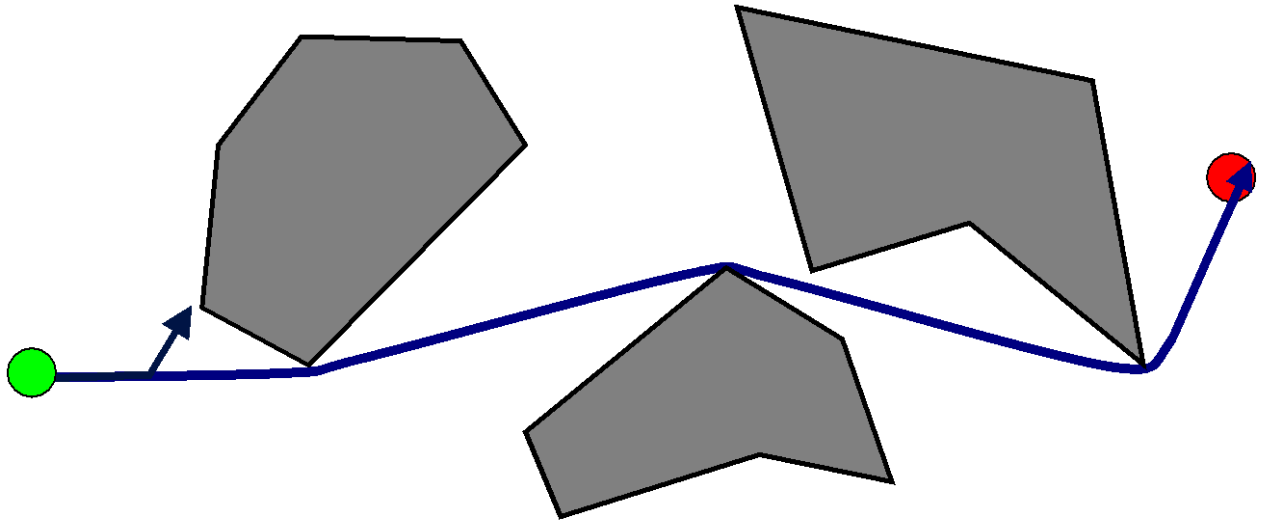
- **What are the sources of uncertainty**

- **Sensing (State)**
  - Where is the robot?
  - Where are the obstacles?
  - What is the map of the environment?

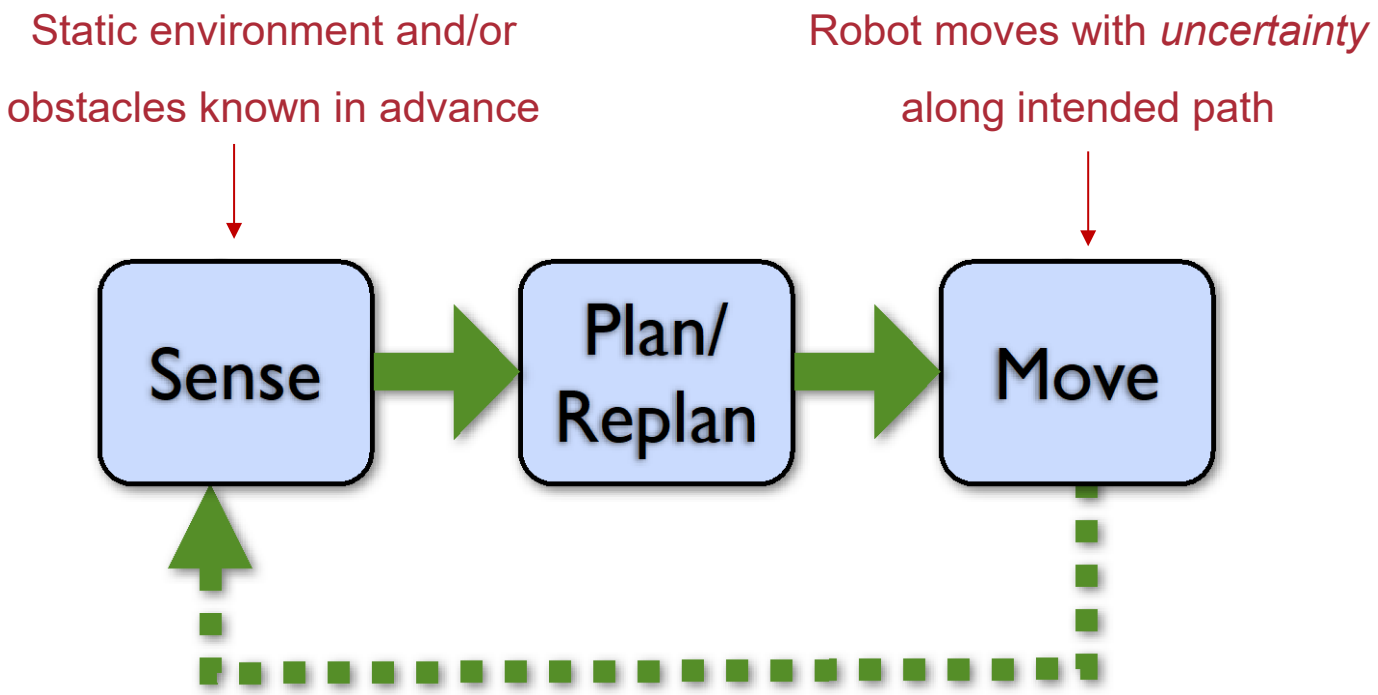
- **Motion (Actions)**
  - How will objects in the environment move?
  - Given a command action, where will the robot end up?

Uncertainty in Robot Motion

- Effect of motion uncertainty



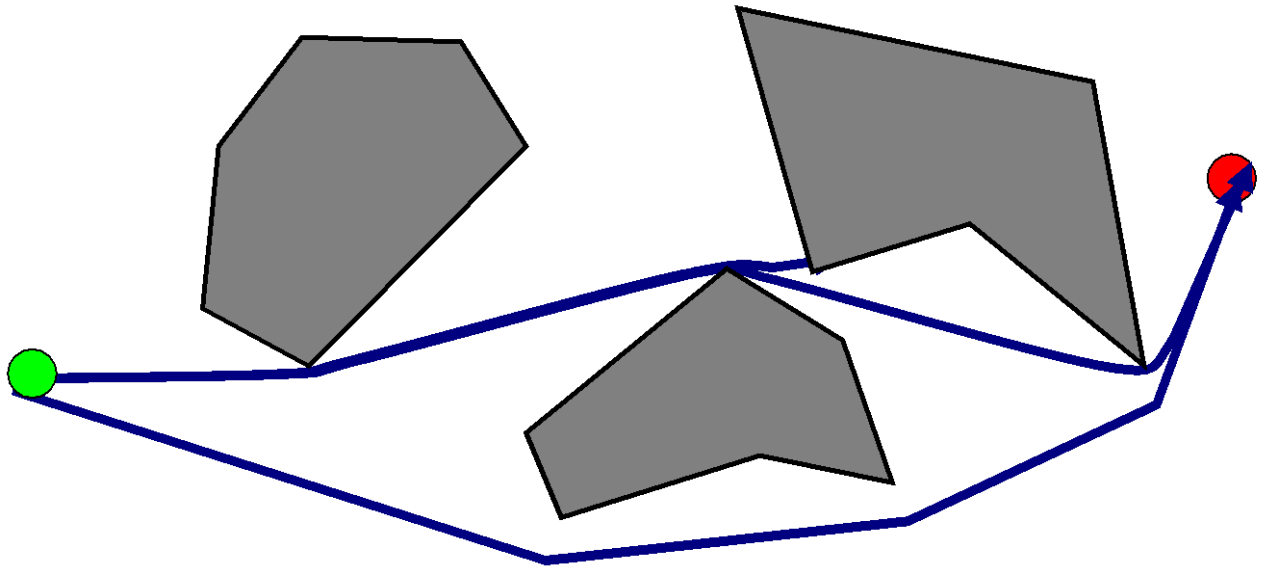
Uncertainty in Robot Motion



- Since we have perfect sensing, is it sufficient to react to uncertain/probabilistic events only after they occur (replanning)?

Uncertainty in Robot Motion

•Effect of motion uncertainty



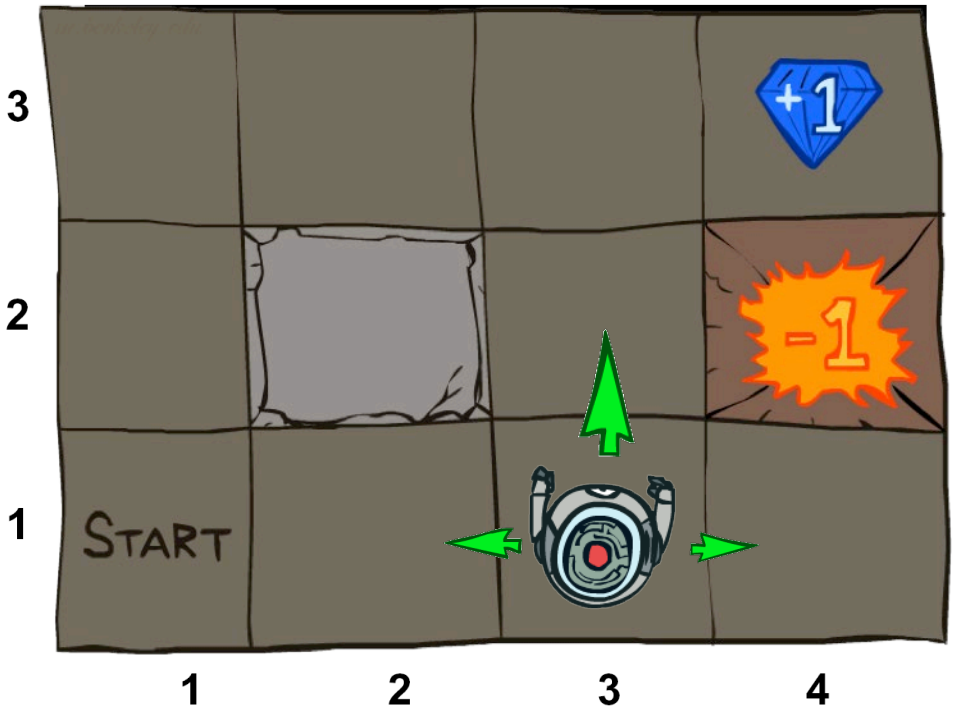
- If there is uncertainty in motion, is this a good initial plan?
- Online feedback must be at high rate to prevent collision
- Can we create a plan that a priori considers the impact of uncertainty?

Overview

- Planning Under uncertainty
- **Markov Decision Processes**
  - Value Iteration
- Stochastic motion roadmaps
- Partially Observable Decision Processes

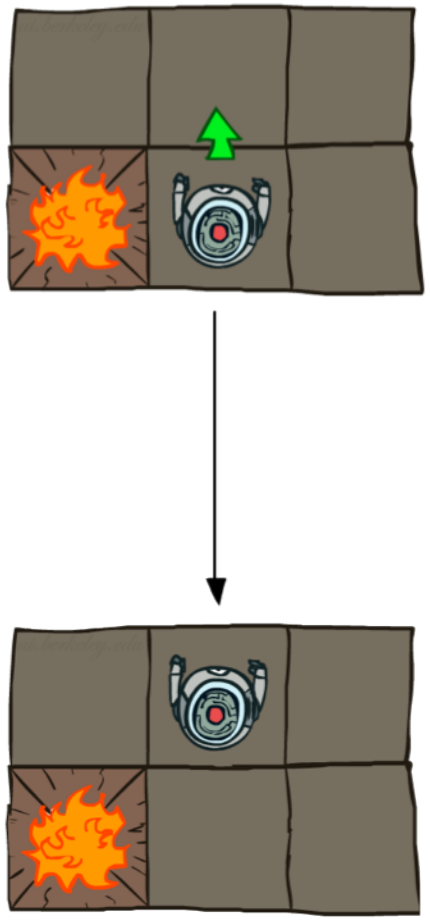
Example: Grid World

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: 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
- The agent receives rewards each time step
  - Small "living" reward each step (can be negative)
  - Big rewards come at the end (good or bad)
- Goal: maximize sum of rewards

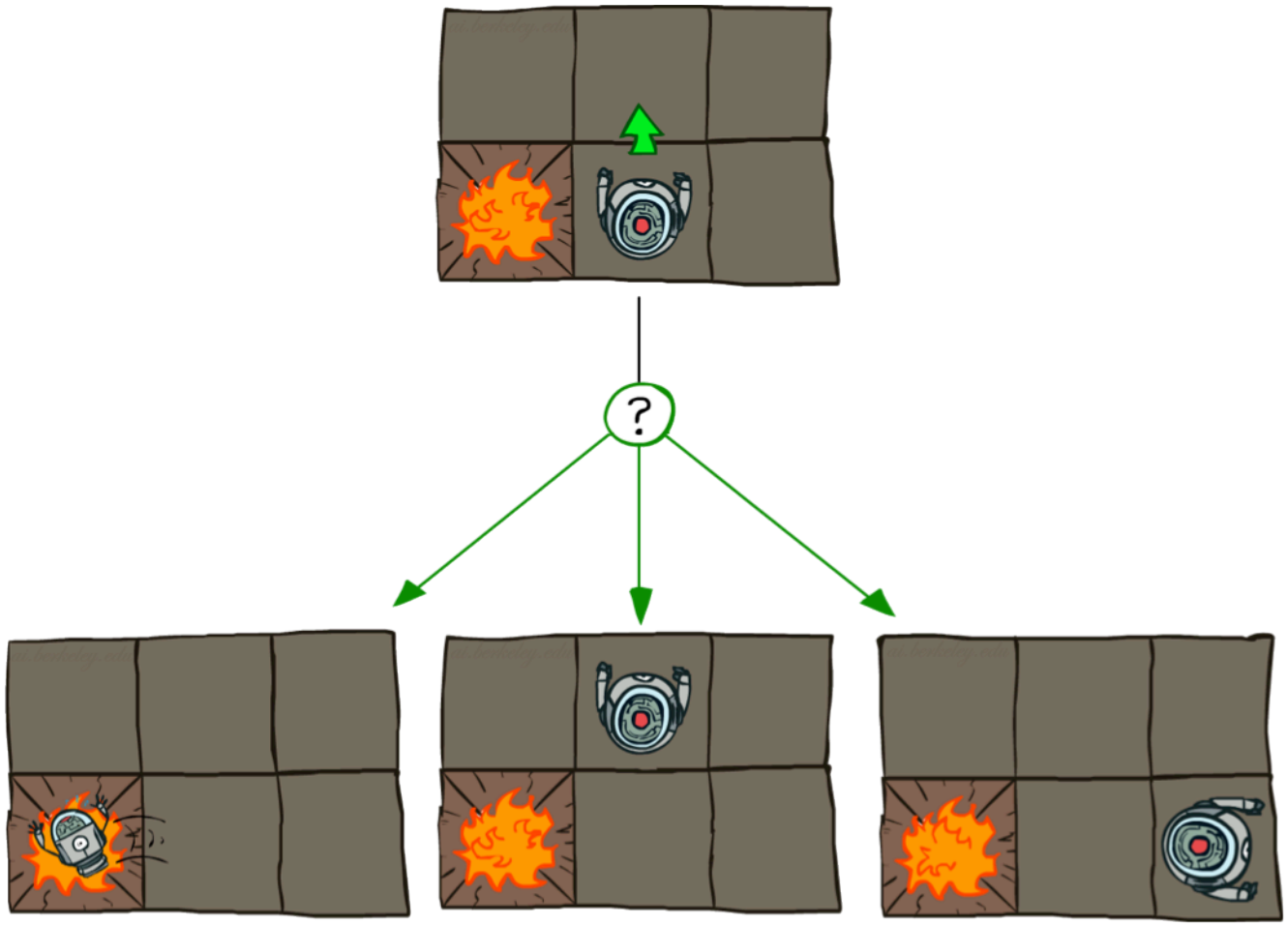


Grid World Actions

Deterministic Grid World



Stochastic Grid World



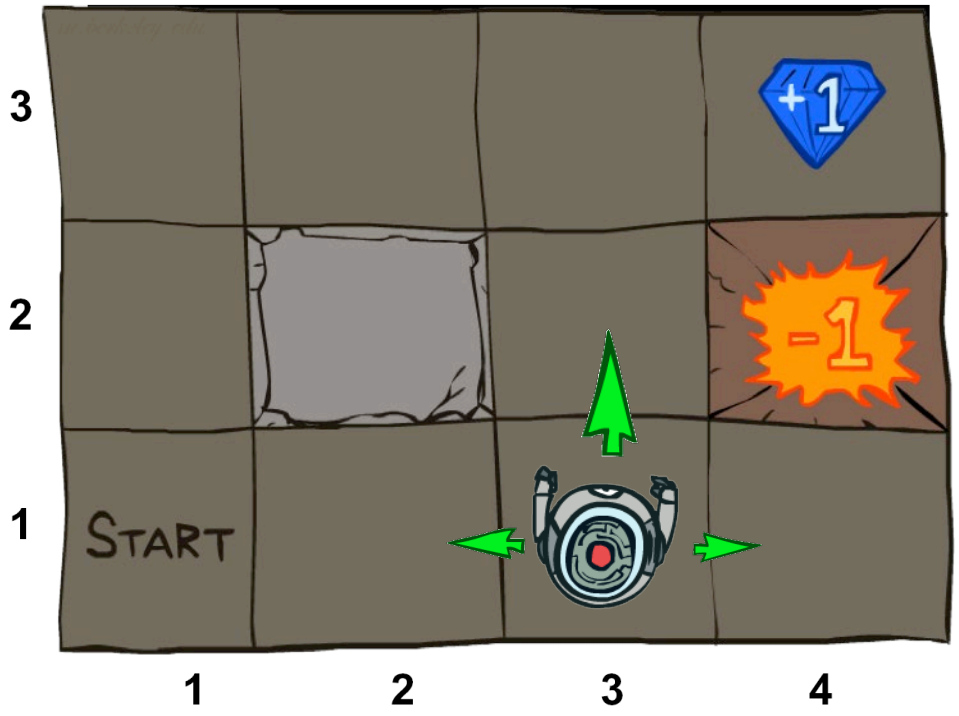
Markov Decision Processes

An MDP is defined by:

A set of states $s \in S$

A set of actions $a \in A$

- A transition function $T(s, a, s')$
  - Probability that a from s leads to s', i.e., $P(s'| s, a)$
  - Also called the model or the dynamics
- A reward function $R(s, a, s')$
  - Sometimes just R(s) or R(s')
- A start state
- Maybe a terminal state



What is Markov about MDPs?

- "Markov" generally means that given the present state, the future and the past are independent

Andrey Markov
(1856-1922)

- For Markov decision processes, "Markov" means action outcome depend only on the current state

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$

$$=$$

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

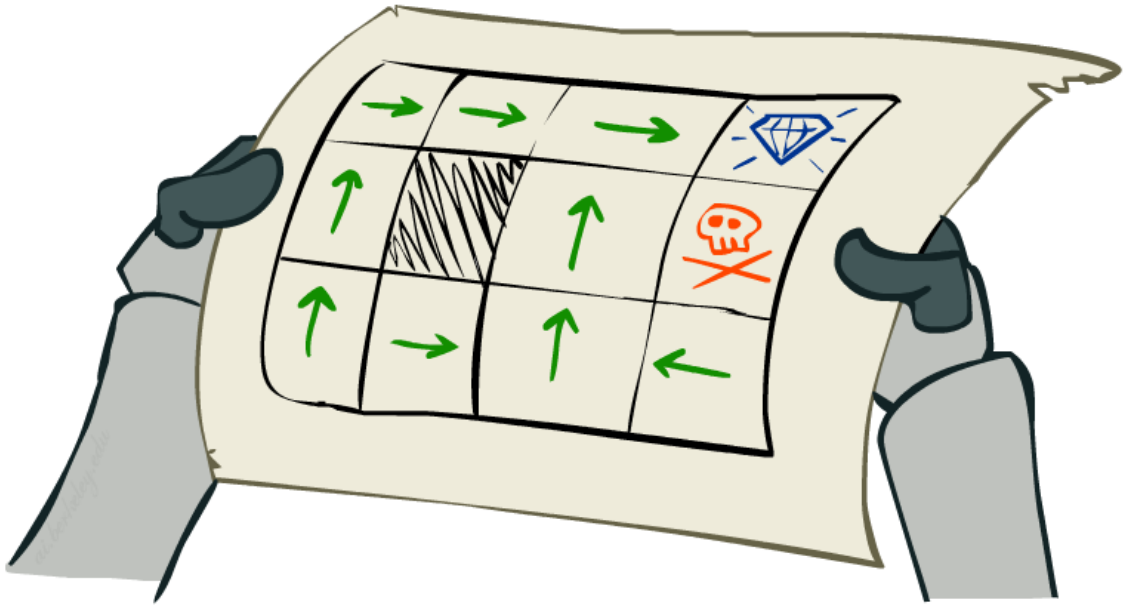
- This is just like search, where the successor function could only depend on the current state (not the history)

Policies

In deterministic single-agent search problem actions, from start to a goal

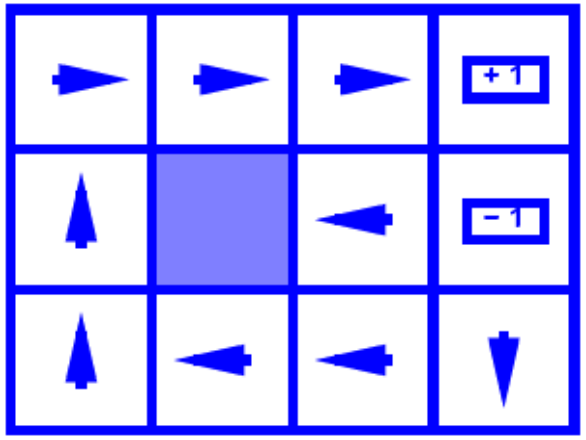
For MDPs, we want a policy π: S $\rightarrow$ A

- A policy π gives an action for each state
- An optimal policy π^* is one that maximizes expectec
- An explicit policy defines a reflex agent
- It computed the action for a single state only

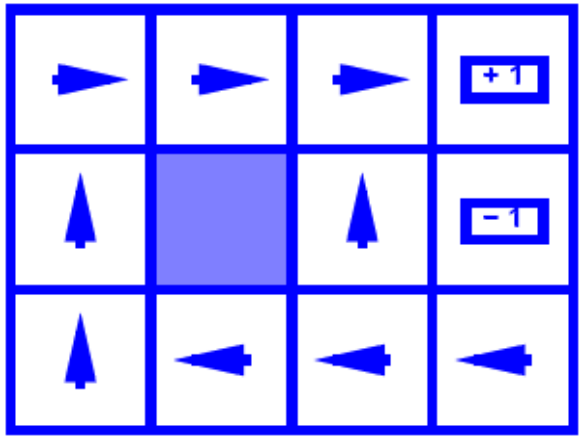


Optimal policy when R(s, a, s') = -0.03
for all non-terminals s

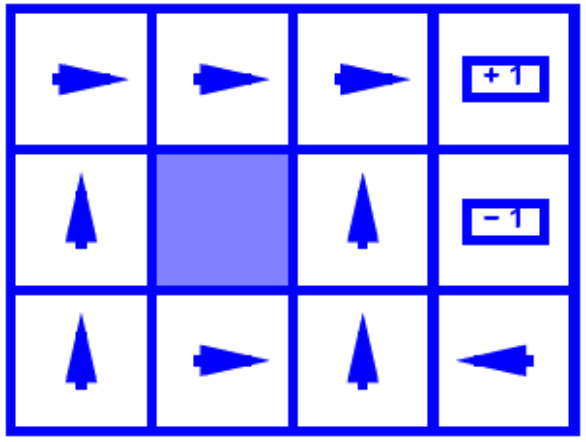
Optimal Policies



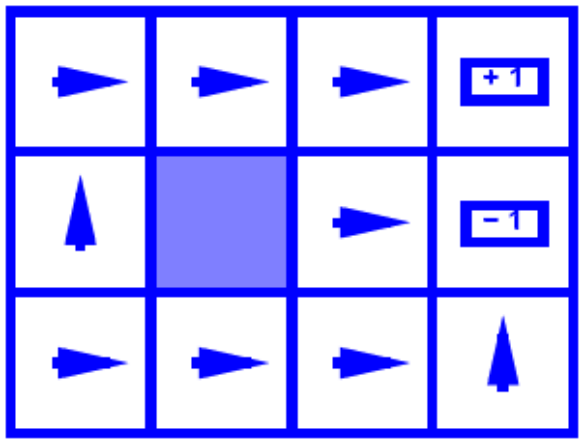
R(s) = -0.01



R(s) = -0.03

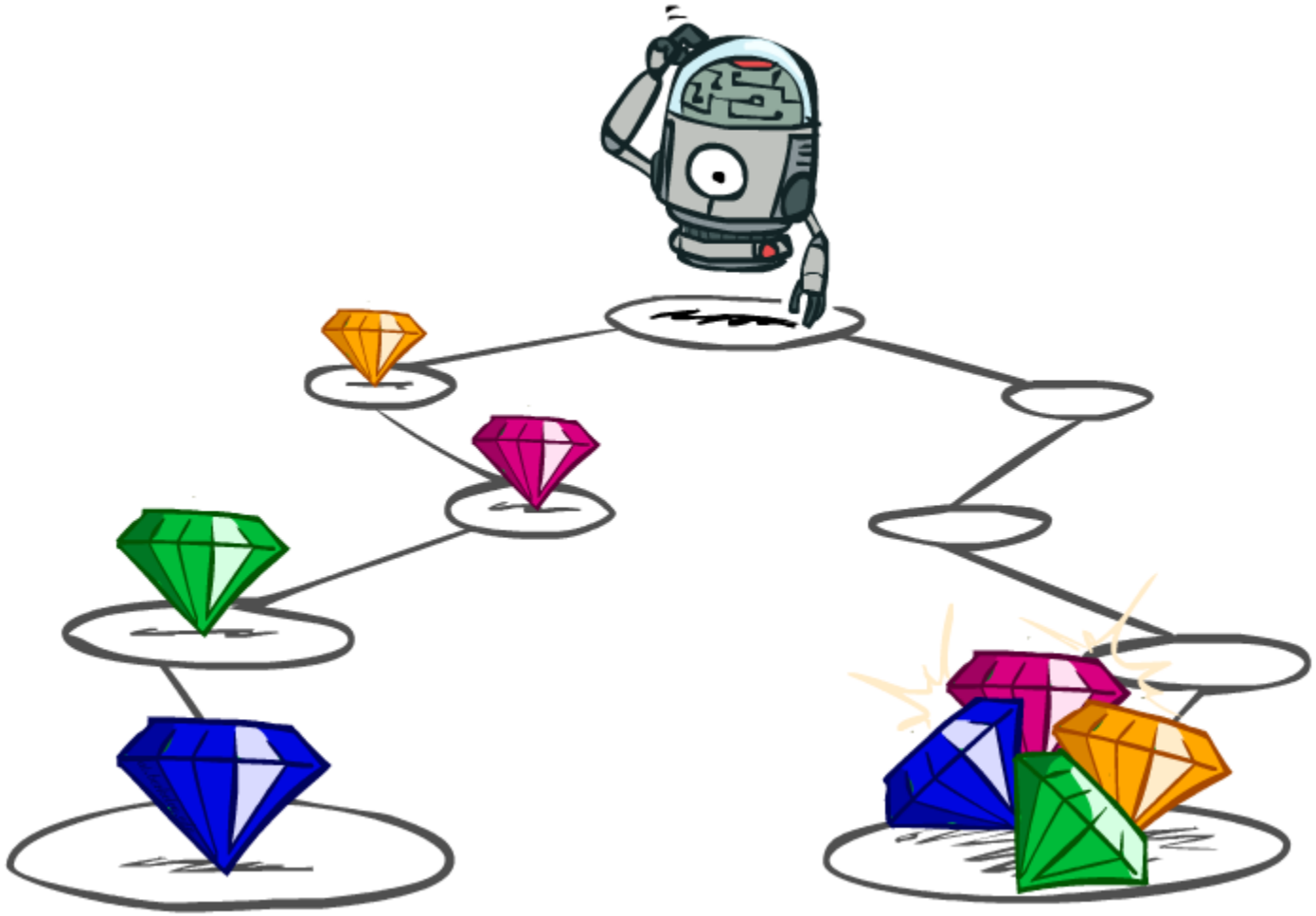


R(s) = -0.4



R(s) = -2.0

Utilities of Sequences

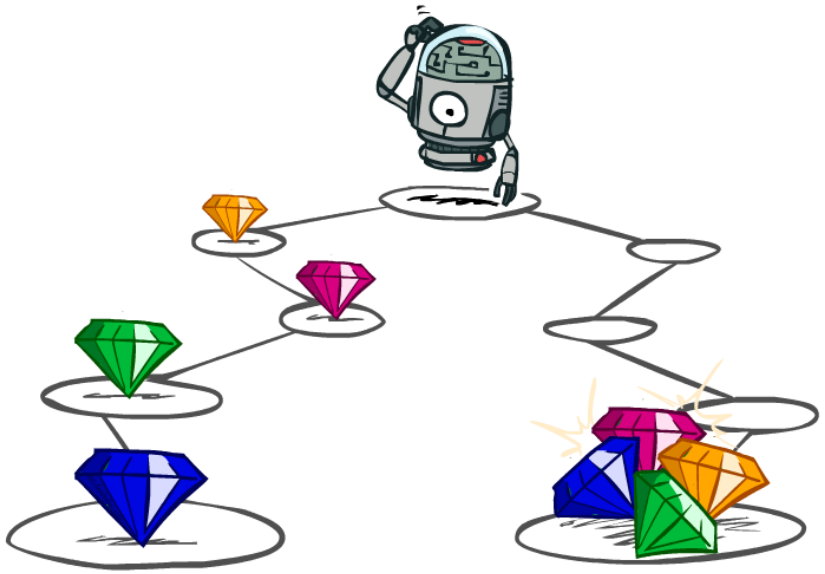


Utilities of Sequences

- What preferences should an agent have over reward sequences?

- More or less? [1, 2, 2] or [2, 3, 4]

- Now or later? [0, 0, 1] or [1, 0, 0]



Discounting

- It's reasonable to maximize the sum of rewards
- It's also reasonable to prefer rewards now to rewards later
- One solution: values of rewards decay exponentially

1

Worth Now

γ

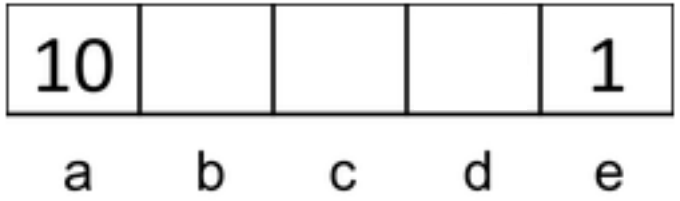
Worth Next Step

γ^2

Worth In Two Steps

Quiz: Discounting

- Given:

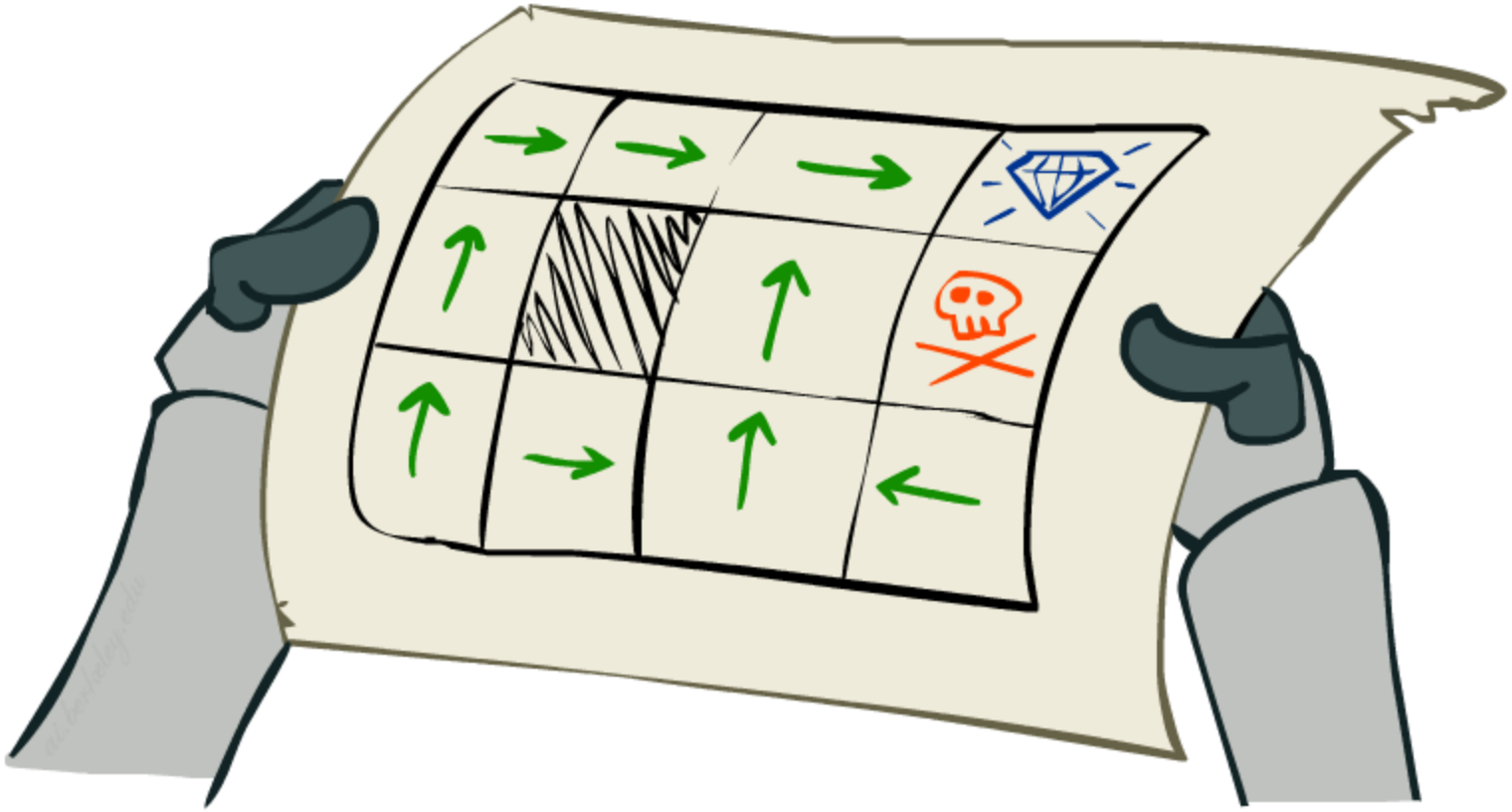


  - Actions: East, West, and Exit (only available in exit states a, e)
  - Transitions: deterministic

- Quiz 1: For $\gamma = 1$, what is the optimal policy?

- Quiz 2: For $\gamma = 0.1$, what is the optimal policy?

Solving MDPs



Optimal Quantities

- The value (utility) of a state s:

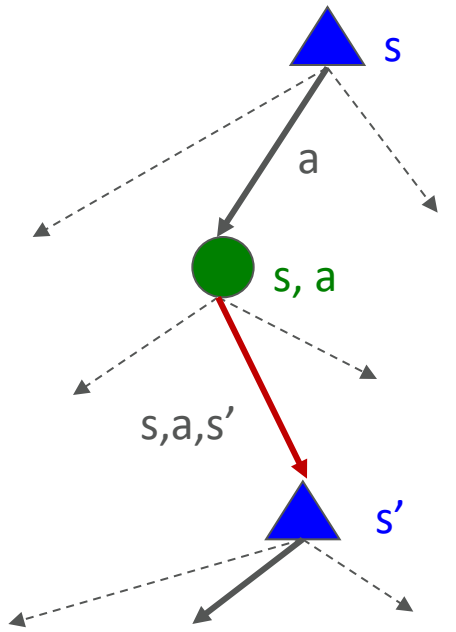
  $V^*(s)$ = expected utility starting in s and acting optimally

- The value (utility) of a q-state (s,a):

  $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:

  $\pi^*(s)$ = optimal action from state s



s is a *state*

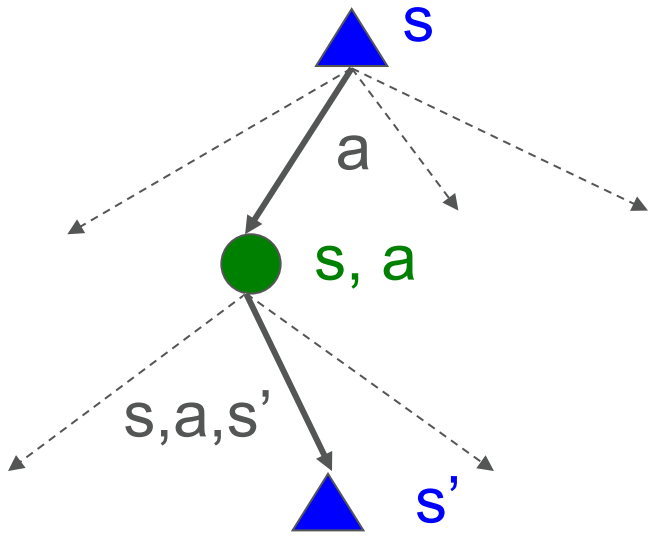
(s, a) is a *q-state*

(s,a,s') is a *transition*

[Demo – gridworld values (L8D4)]

Values of States

- Fundamental operation: compute the (expected) value of a state
  - Expected reward under optimal action
  - Average sum of (discounted) rewards
- Recursive definition of value:



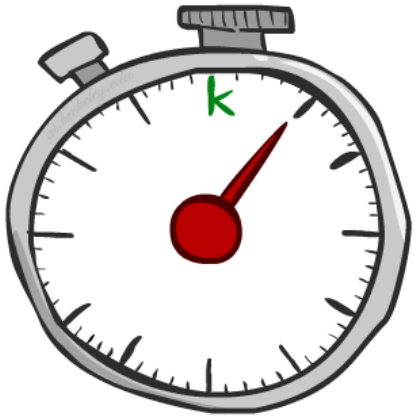
$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

Time-limited Values

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

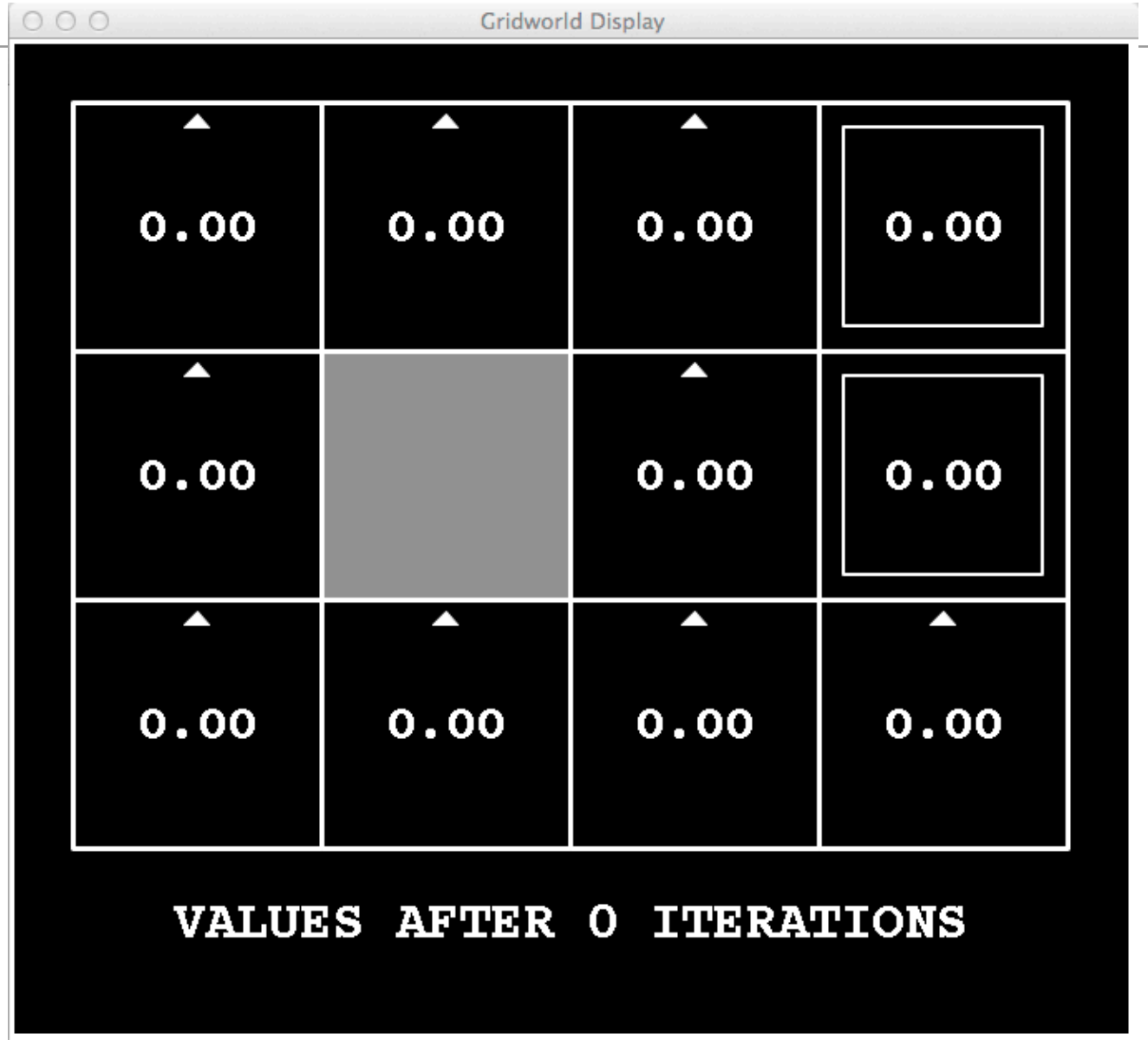


How can we compute this, without doing infinite recursion?

- Key idea: time-limited values
- Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps
- Then we can iteratively compute $V_0(s)$, $V_1(s)$, $V_2(s)$ etcetera

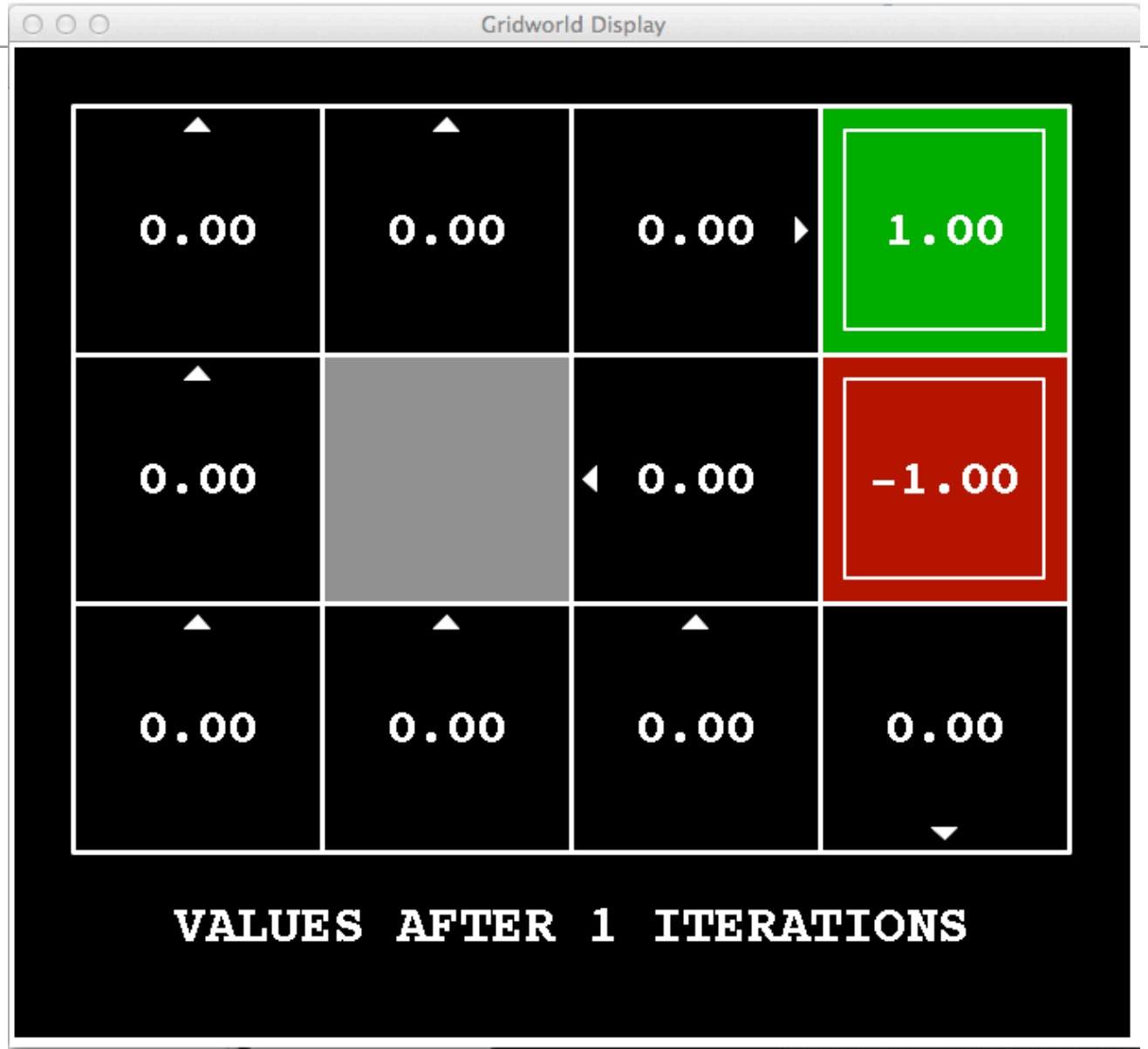
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

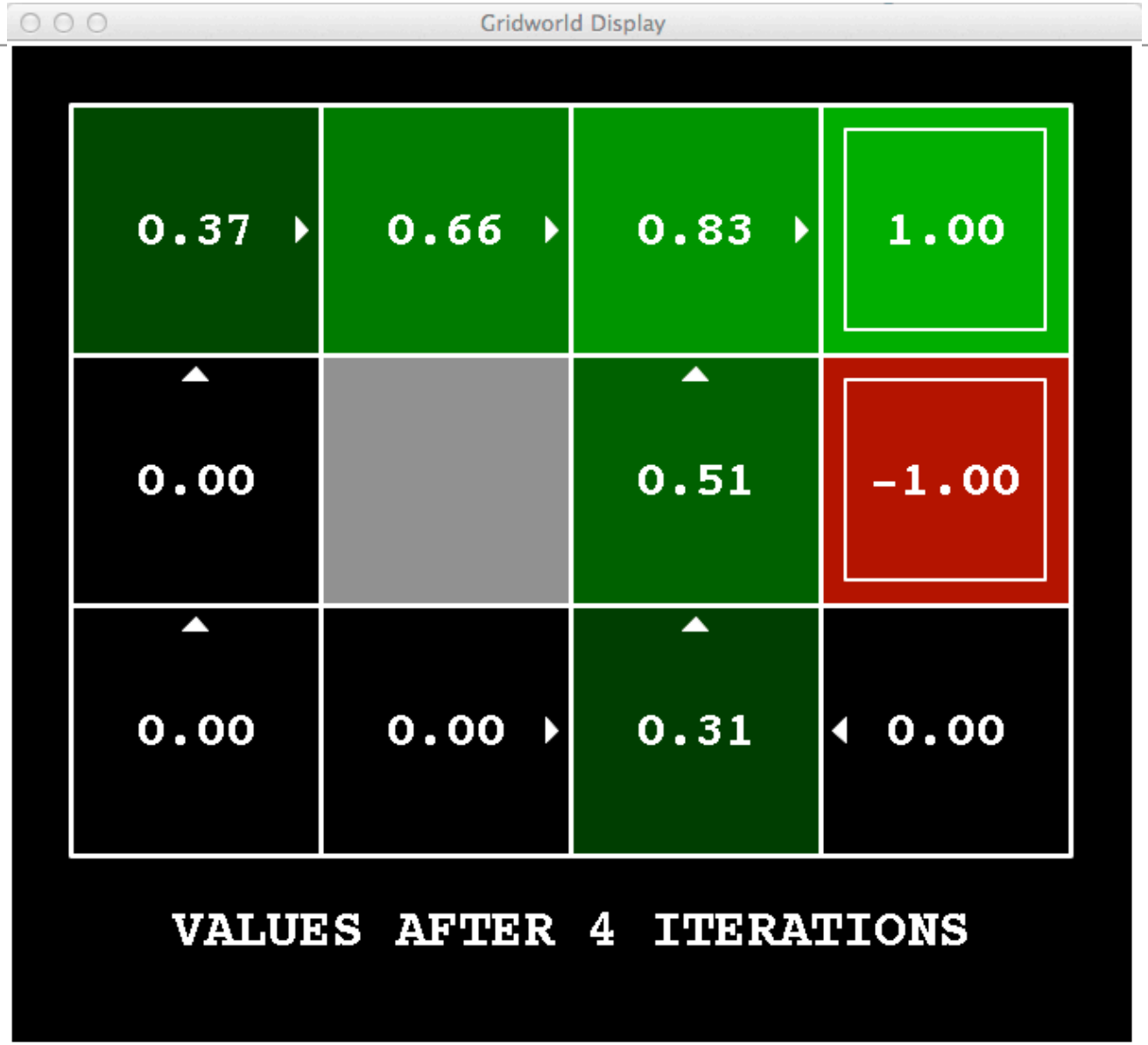
k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

k=3

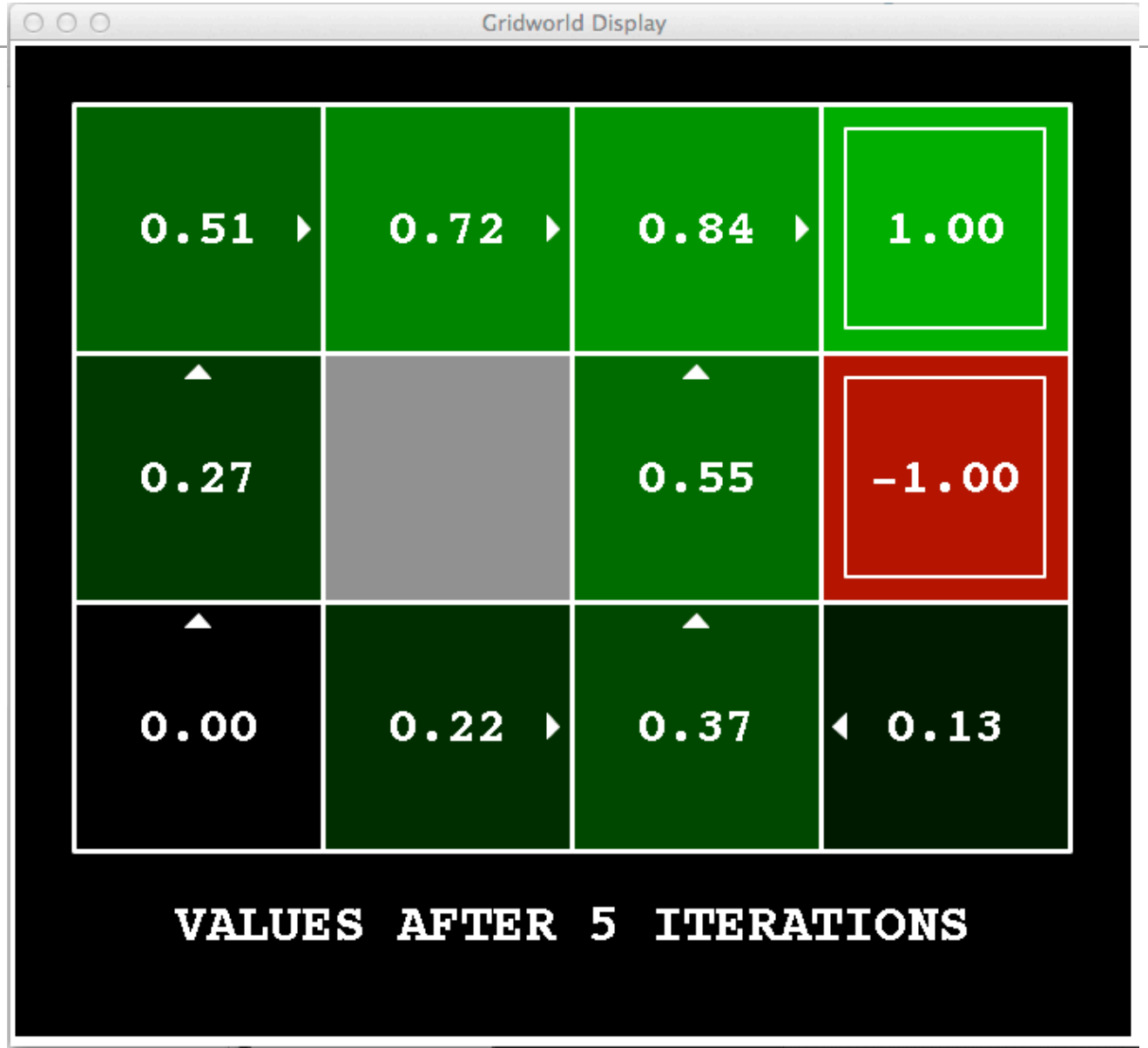
Noise = 0.2
Discount = 0.9
Living reward = 0

k=4



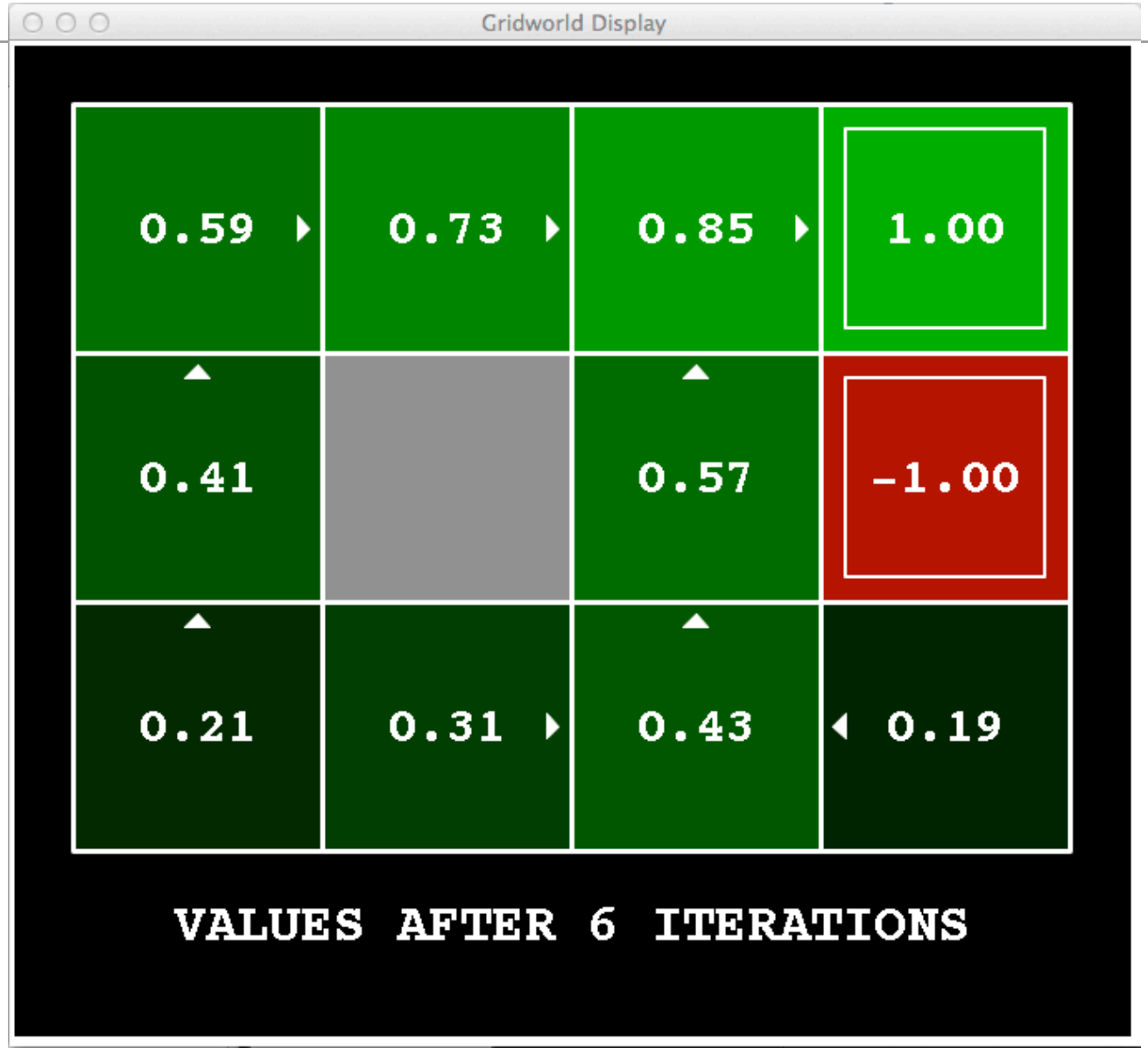
Noise = 0.2
Discount = 0.9
Living reward = 0

k=5



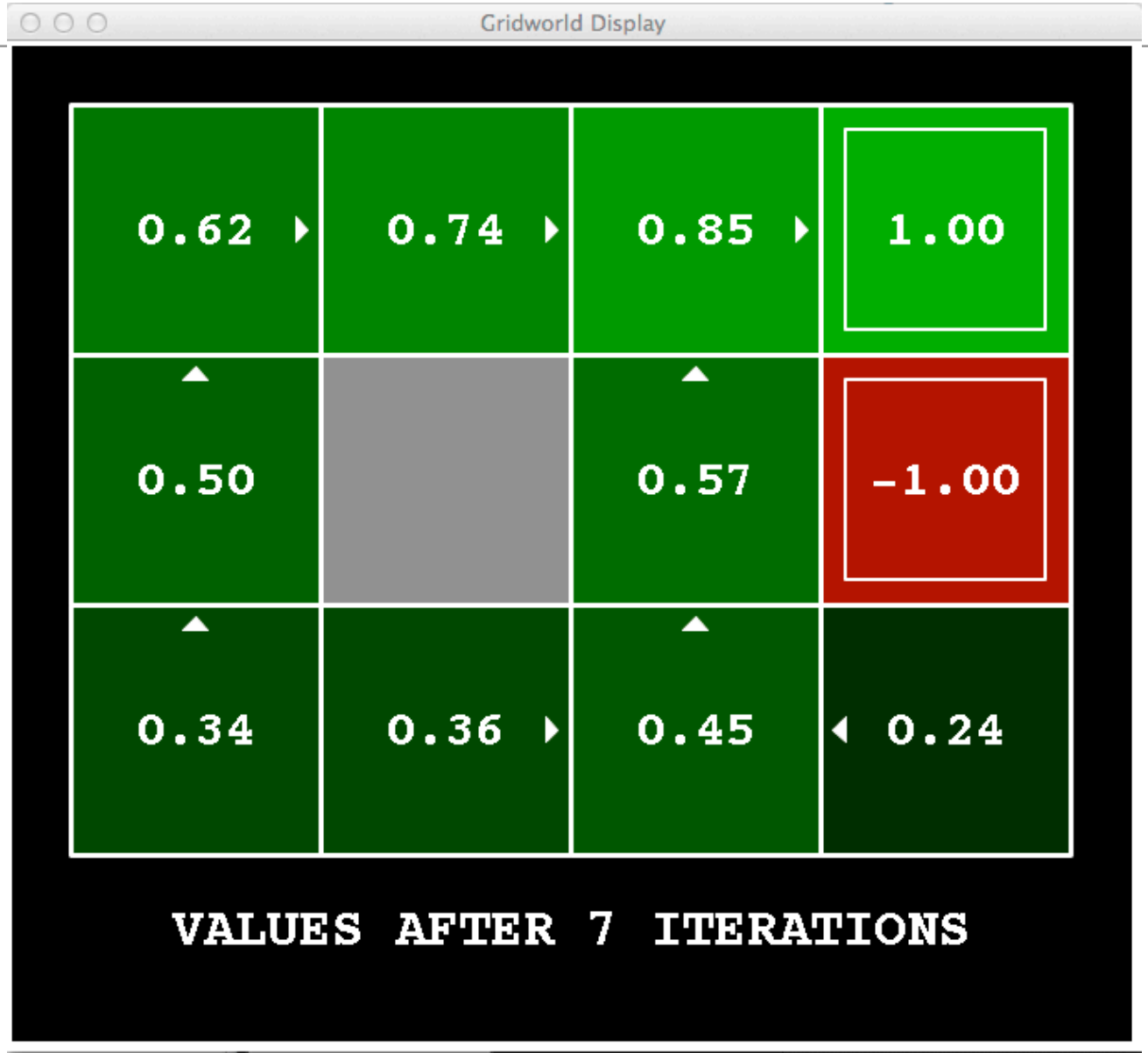
Noise = 0.2
Discount = 0.9
Living reward = 0

k=6



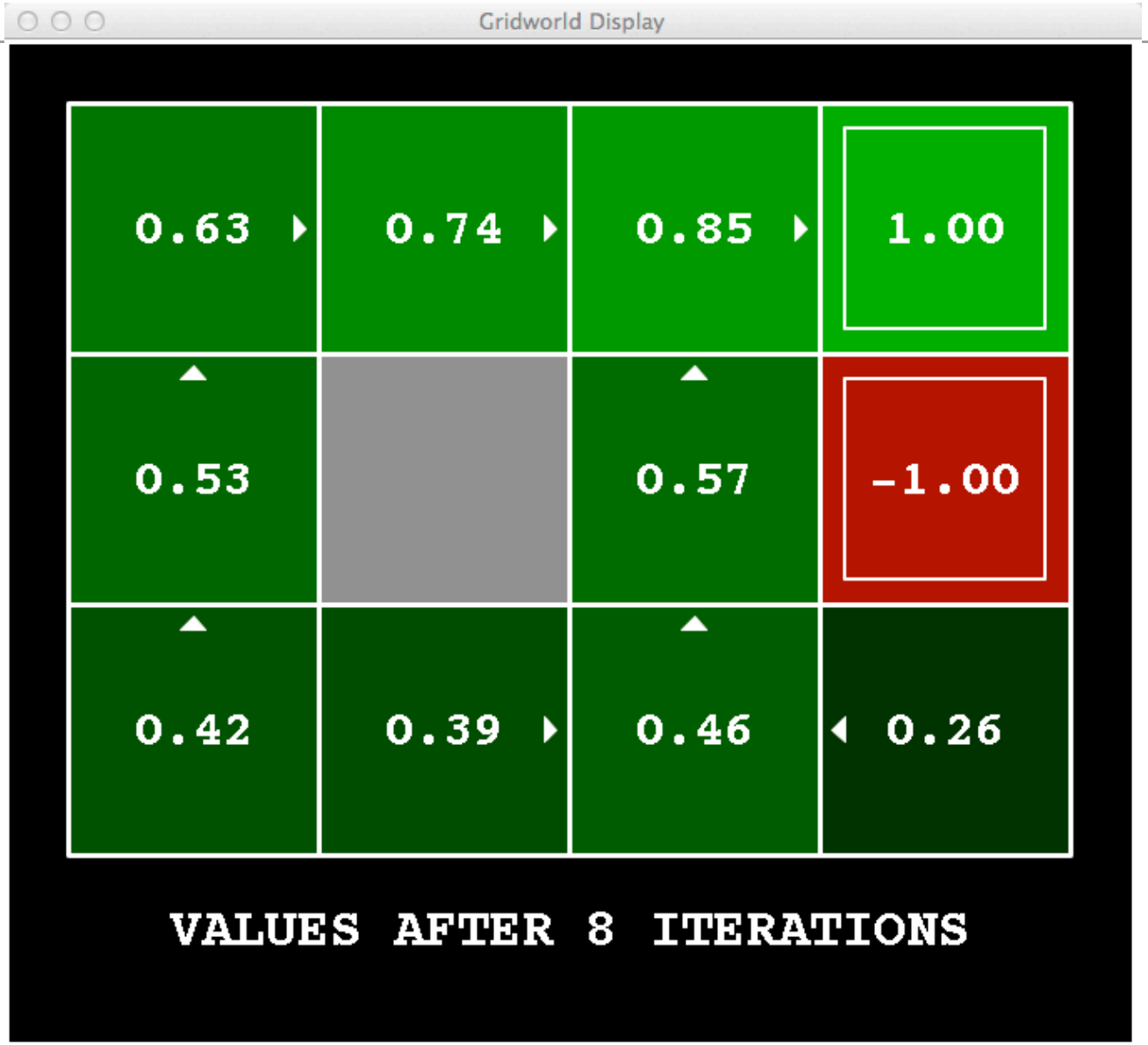
Noise = 0.2
Discount = 0.9
Living reward = 0

k=7



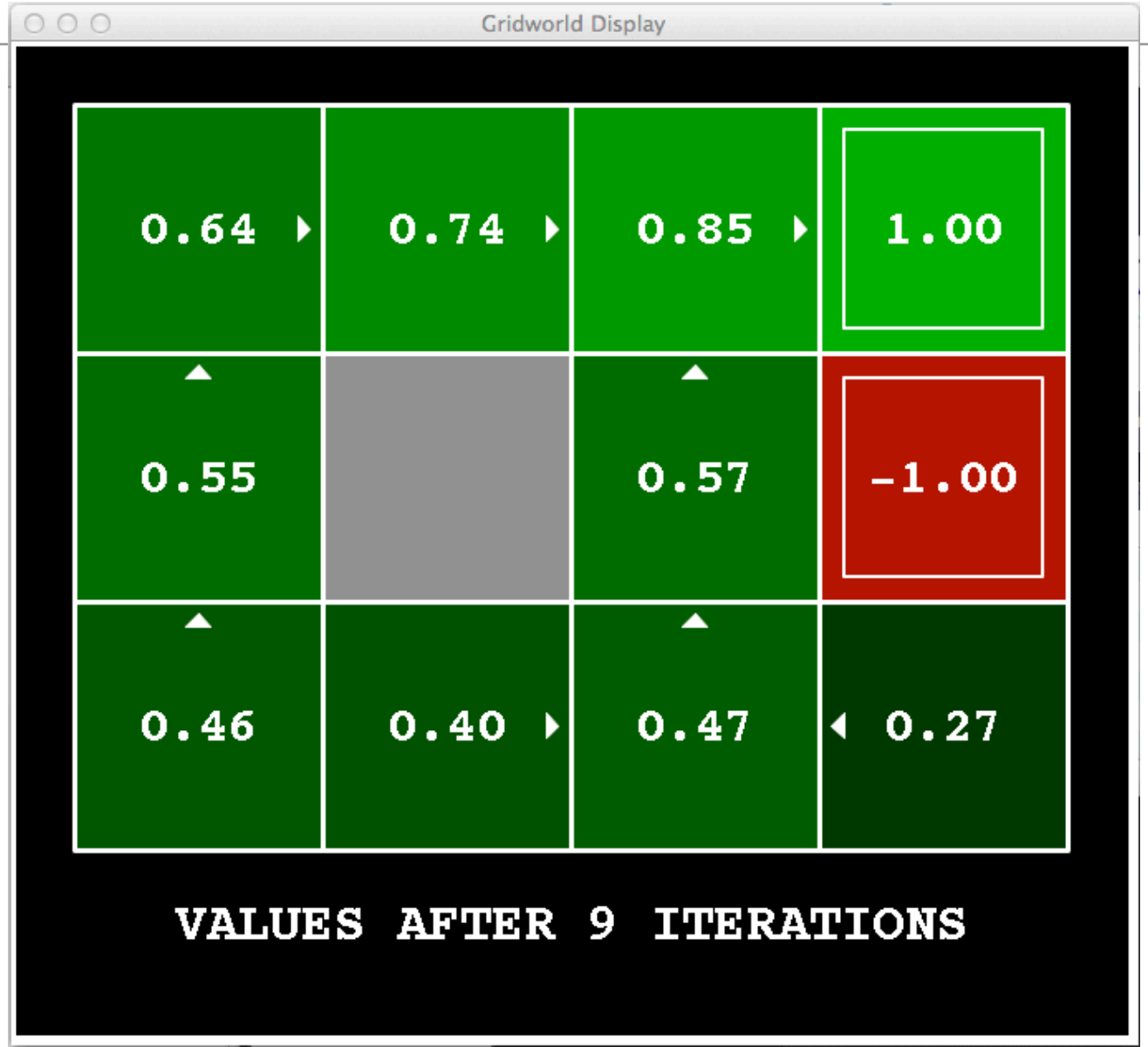
Noise = 0.2
Discount = 0.9
Living reward = 0

k=8



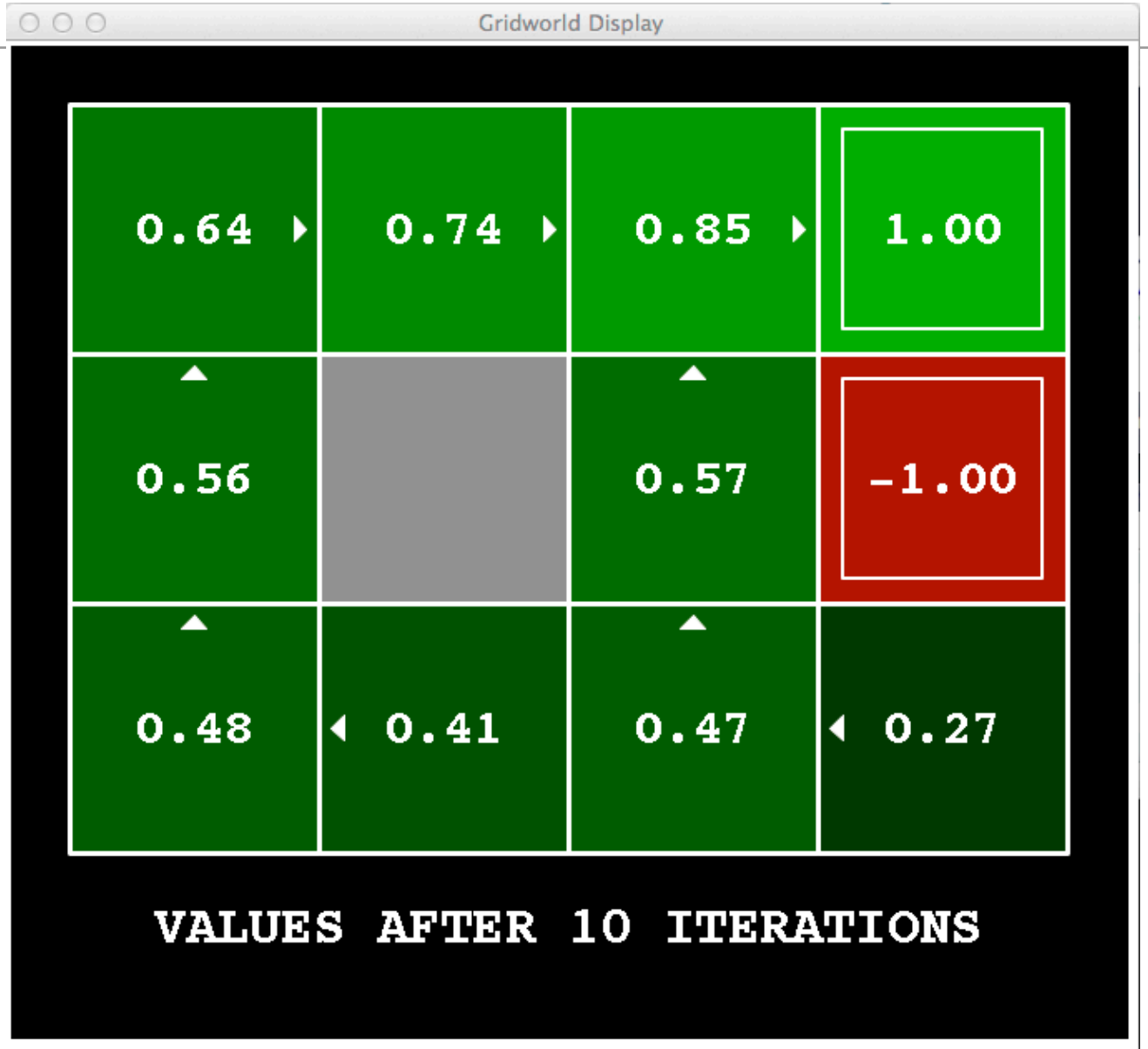
Noise = 0.2
Discount = 0.9
Living reward = 0

k=9



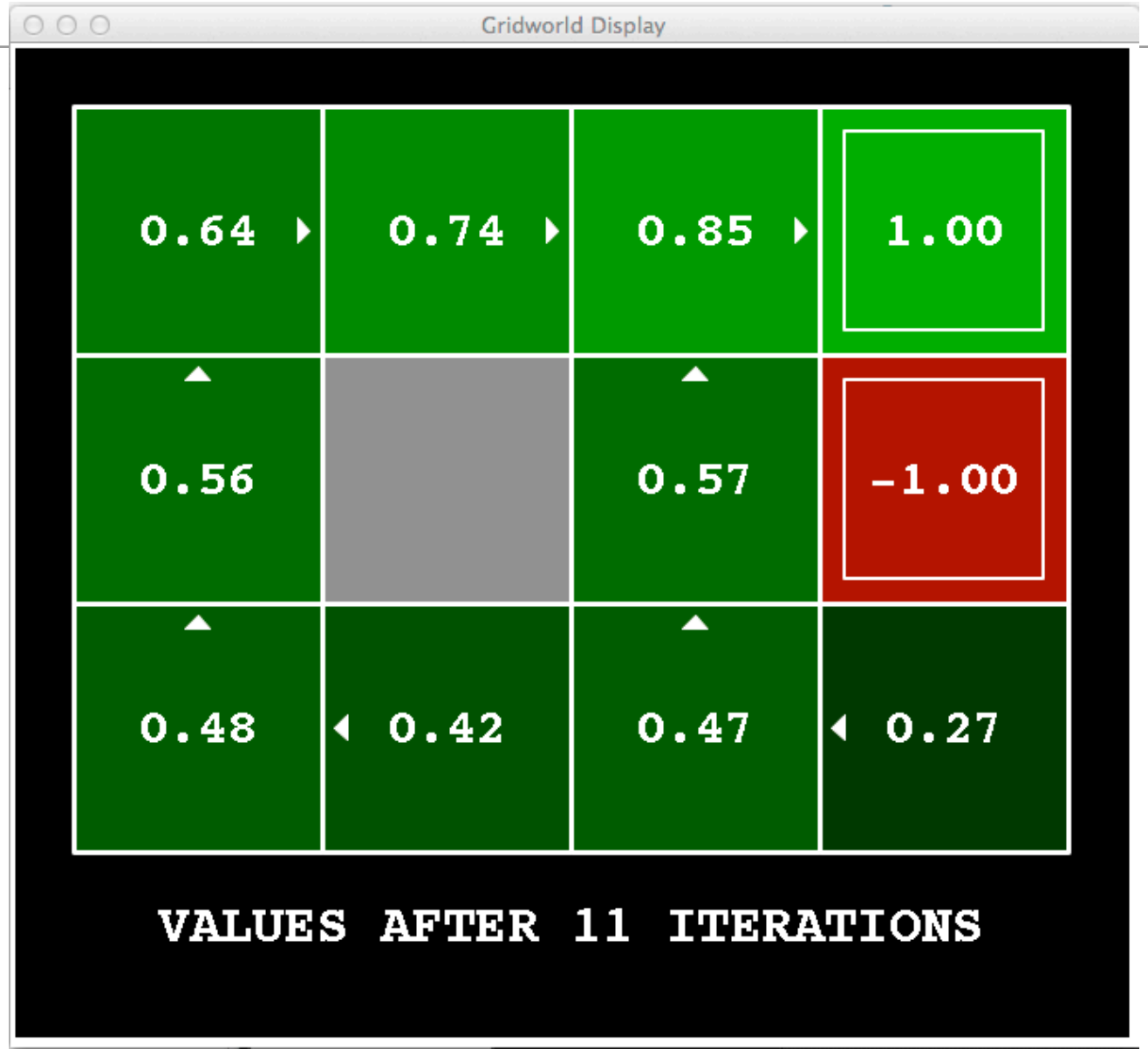
Noise = 0.2
Discount = 0.9
Living reward = 0

k=10



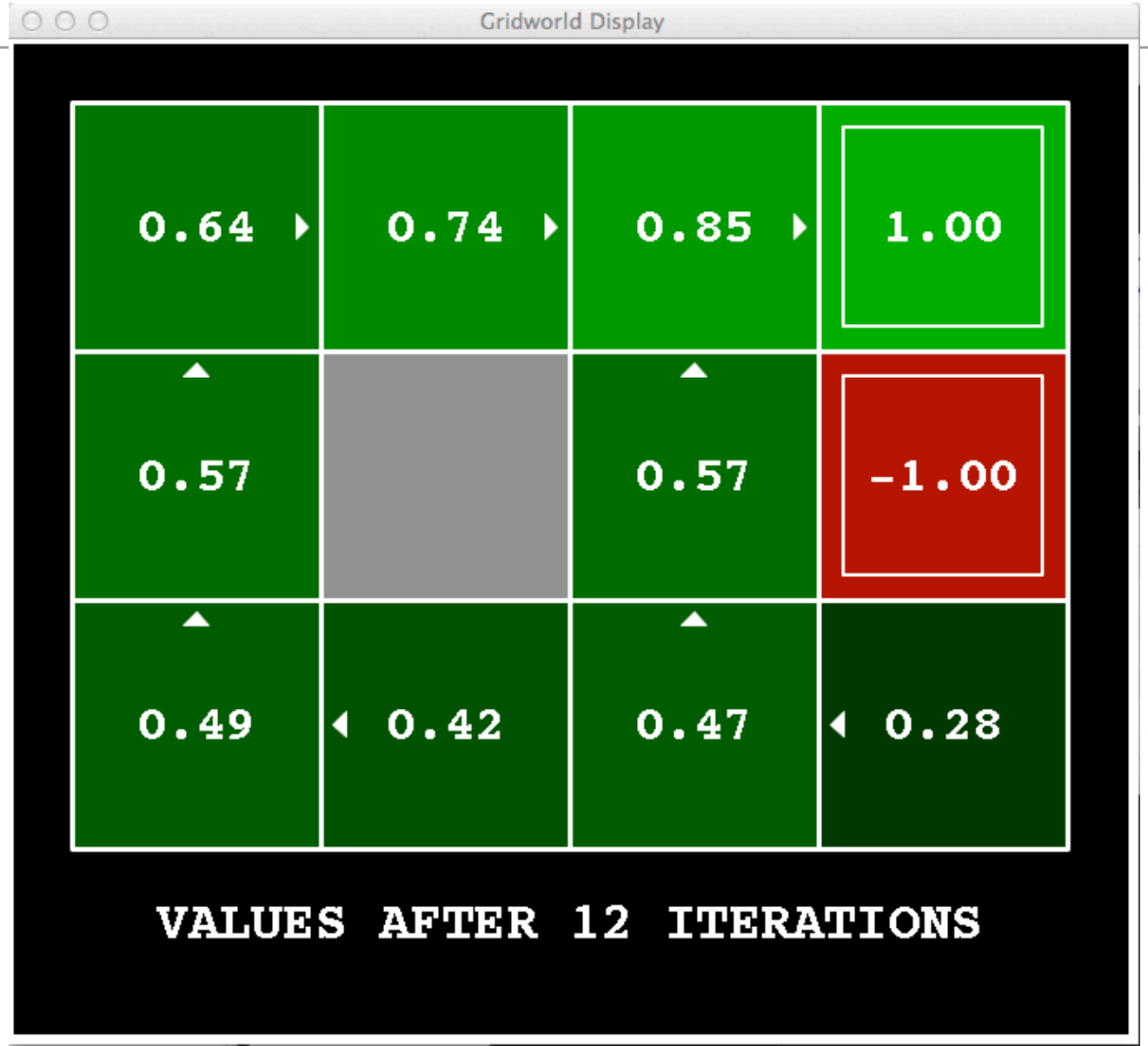
Noise = 0.2
Discount = 0.9
Living reward = 0

k=11



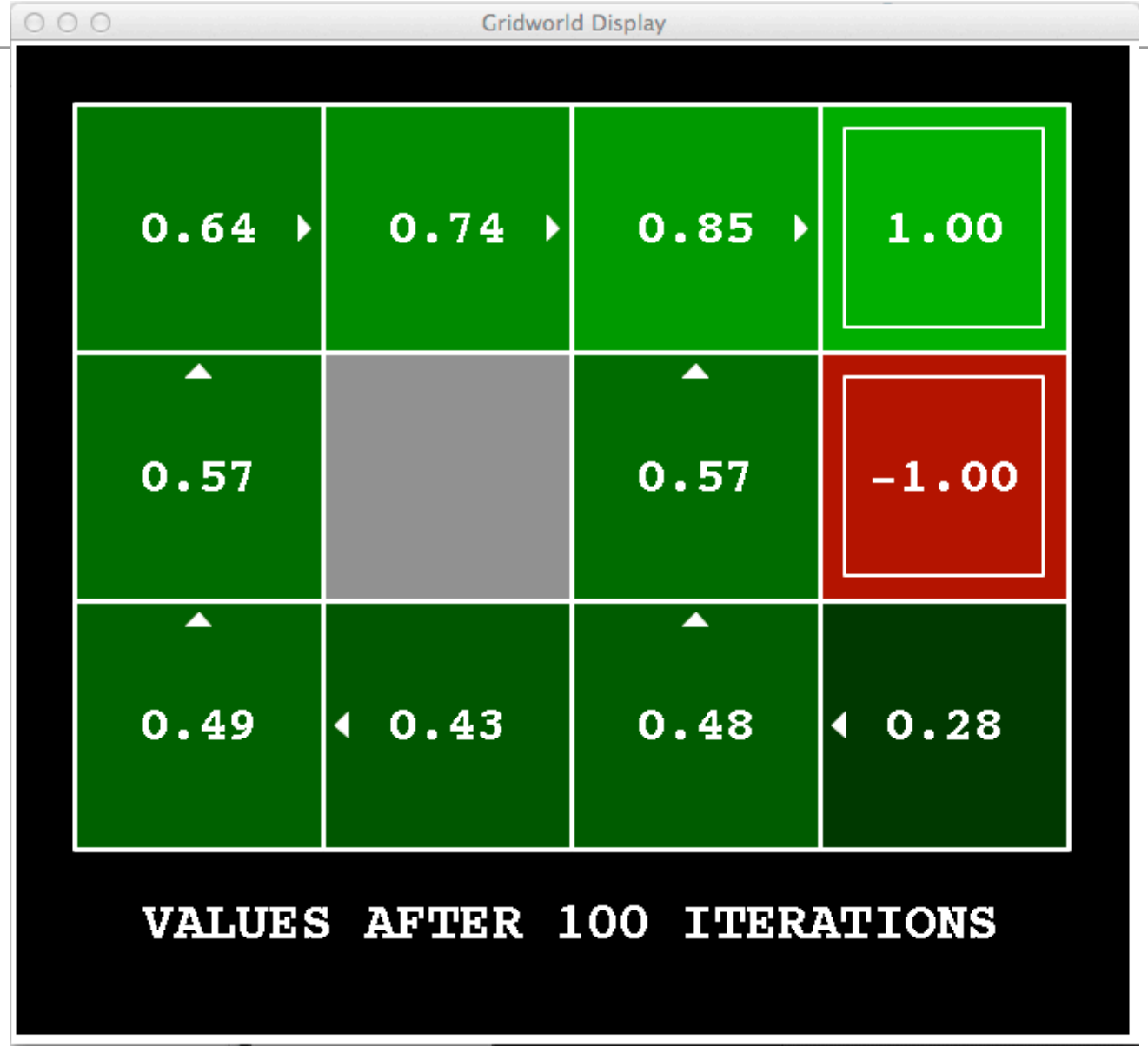
Noise = 0.2
Discount = 0.9
Living reward = 0

k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

k=100

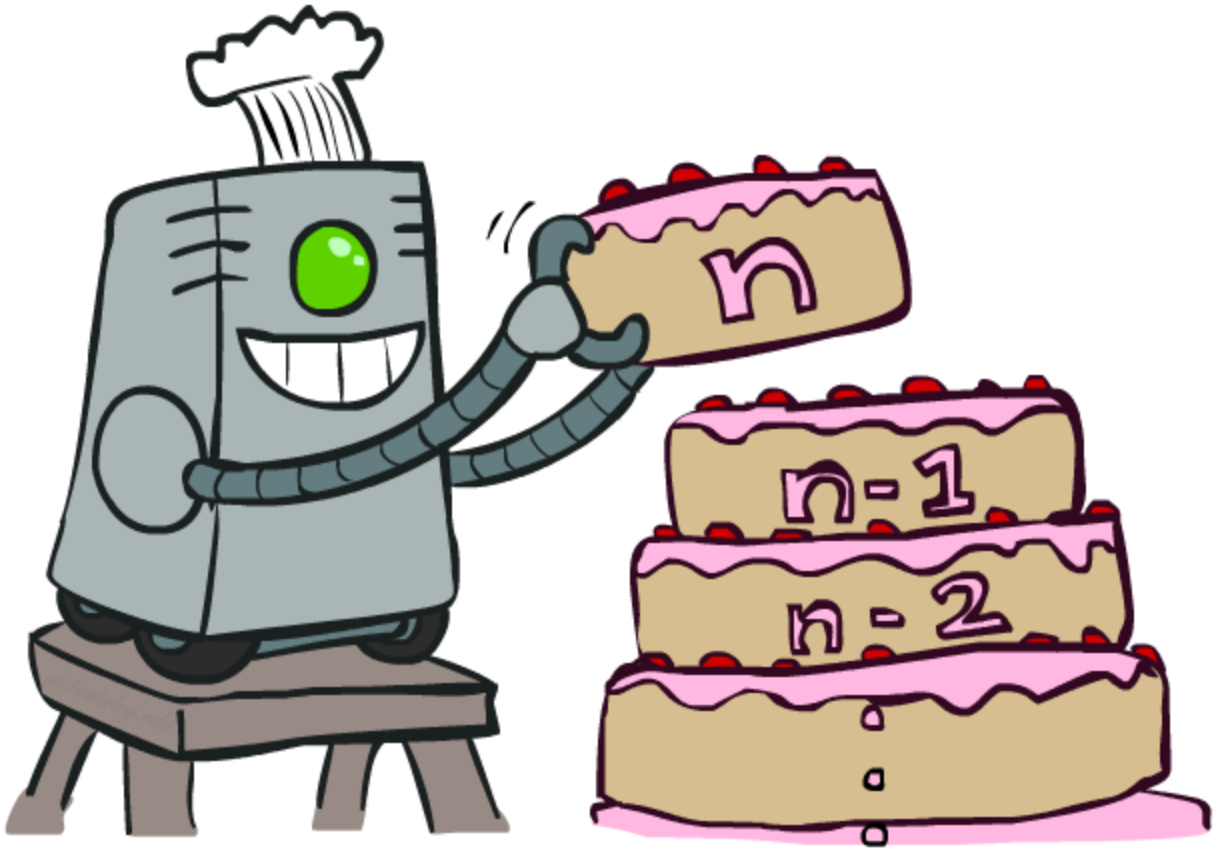


Noise = 0.2
Discount = 0.9
Living reward = 0

Overview

- Planning Under uncertainty
- **Markov Decision Processes**
  - **Value Iteration**
- Stochastic motion roadmaps
- Partially Observable Decision Processes

Value Iteration

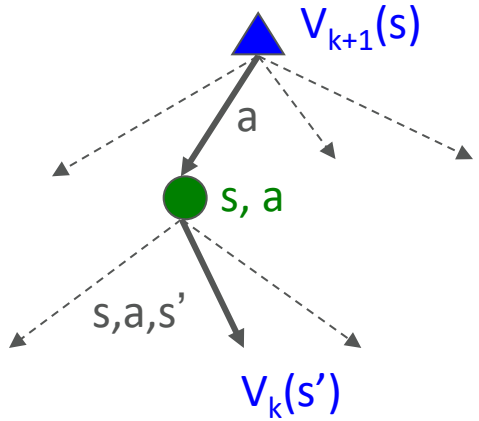


Value Iteration

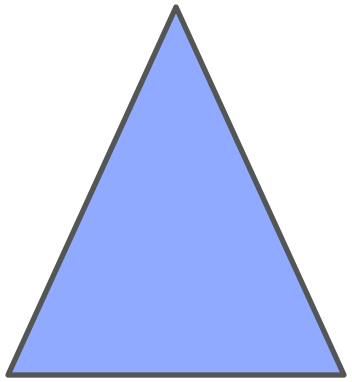
- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero

- Given vector of $V_k(s)$ values, do one step from each state:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$



- Repeat until convergence
- Complexity of each iteration: $O(S^2A)$
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

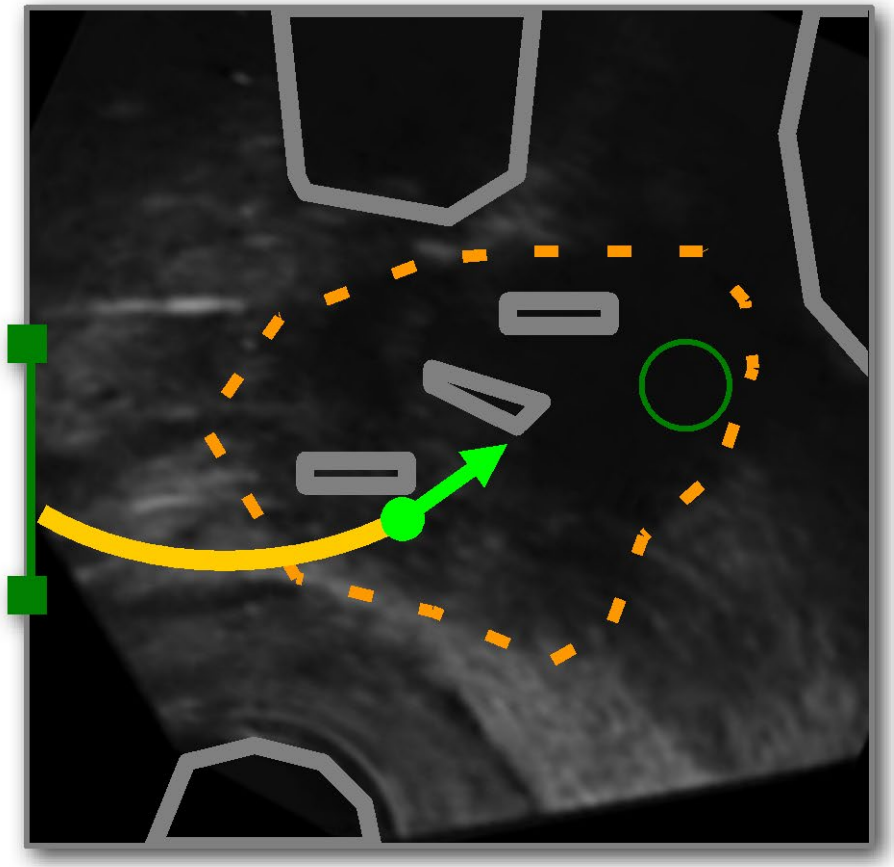


Overview

- Planning Under uncertainty
- Markov Decision Processes
  - Value Iteration
- **Stochastic motion roadmaps**
- Partially Observable Decision Processes

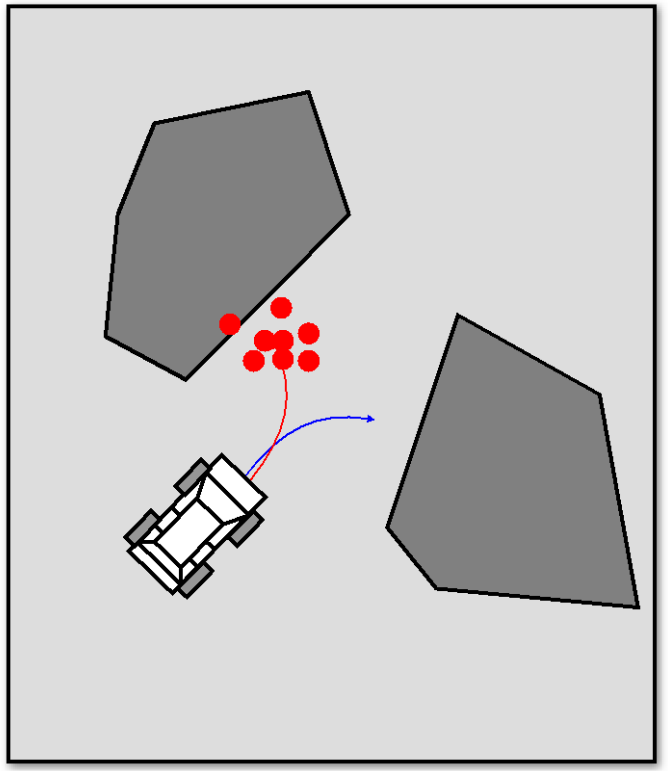
Motivation for Stochastic Motion Planning

- Image guided needle steering

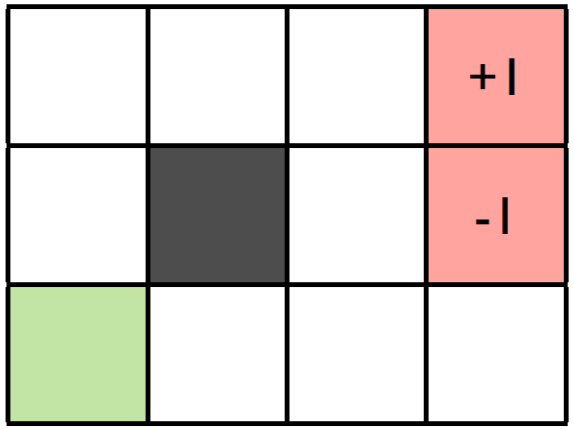


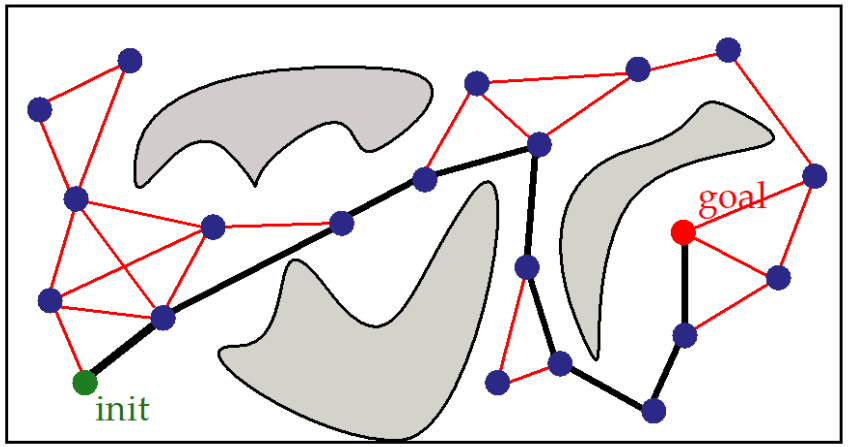
Stochastic Motion Roadmaps: Problem Statement

- Input:
  - Workspace
  - Discrete action set
  - Model of motion uncertainty
- Assumptions:
  - Motion uncertainty at the current state is independent of history (Markov Property)
  - Perfect sensing
- Query:
  - Input: Start, Goal states
  - Output: Actions to maximize the probability of success (minimize collision)



Stochastic Motion Planning (Combining MDPS with PRM)





- MDPs
  - Considers uncertainty
  - How to represent sampled configurations and states?
  - Maximize “reward”
- PRMs
  - No uncertainty in motion
  - Efficient representation of configurations
  - Shortest path

Creating an MDP using PRM

An MDP is defined by:

- A set of states $s \in S$
- A set of actions $a \in A$
- A transition function $T(s, a, s')$
  - Prob. a from s leads to s' $P(s'| s, a)$
- A reward function $R(s, a, s')$

- Stochastic Motion Roadmaps (SMR)
  - Sample configuration space as in PRMs
  - Choose Discrete Control Set
  - **Define connectivity by probability**
  - Estimate probabilities by sampling motion model
  - Minimize the Risk of Collision

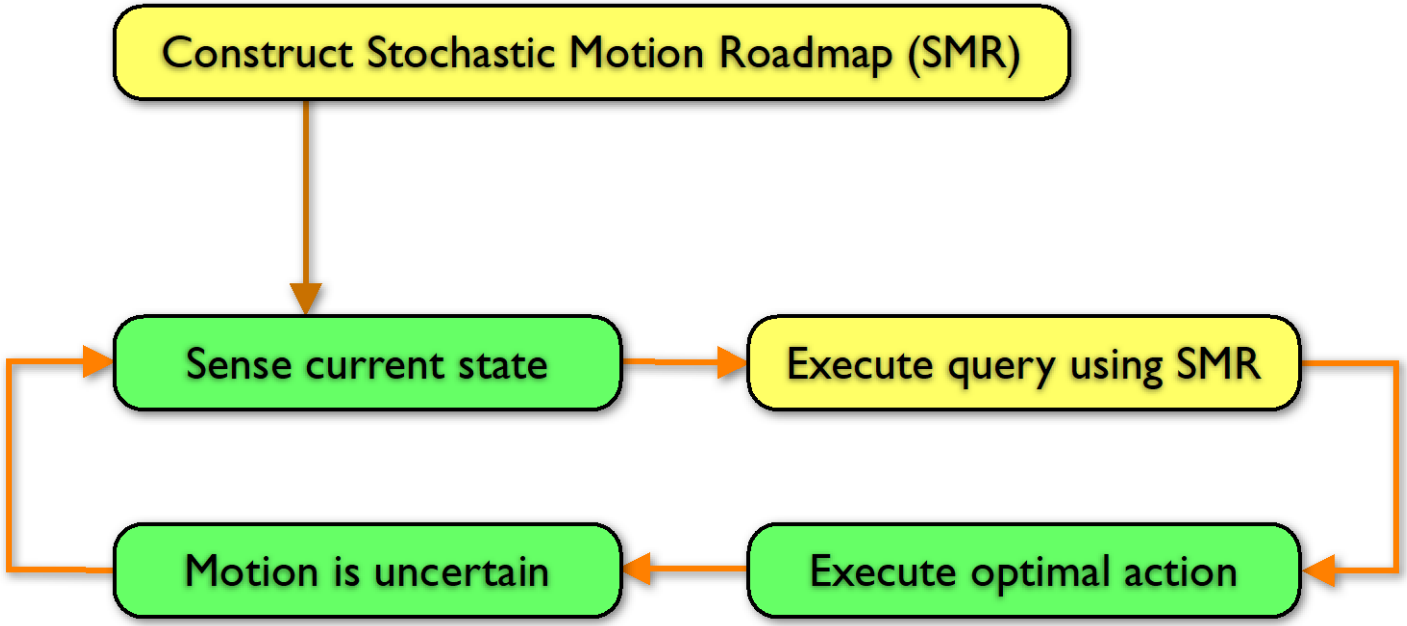
⇒ No problem-specific regular grids

⇒ Enables variety of motion uncertainty representations

SMR

Motion Planning with Feedback

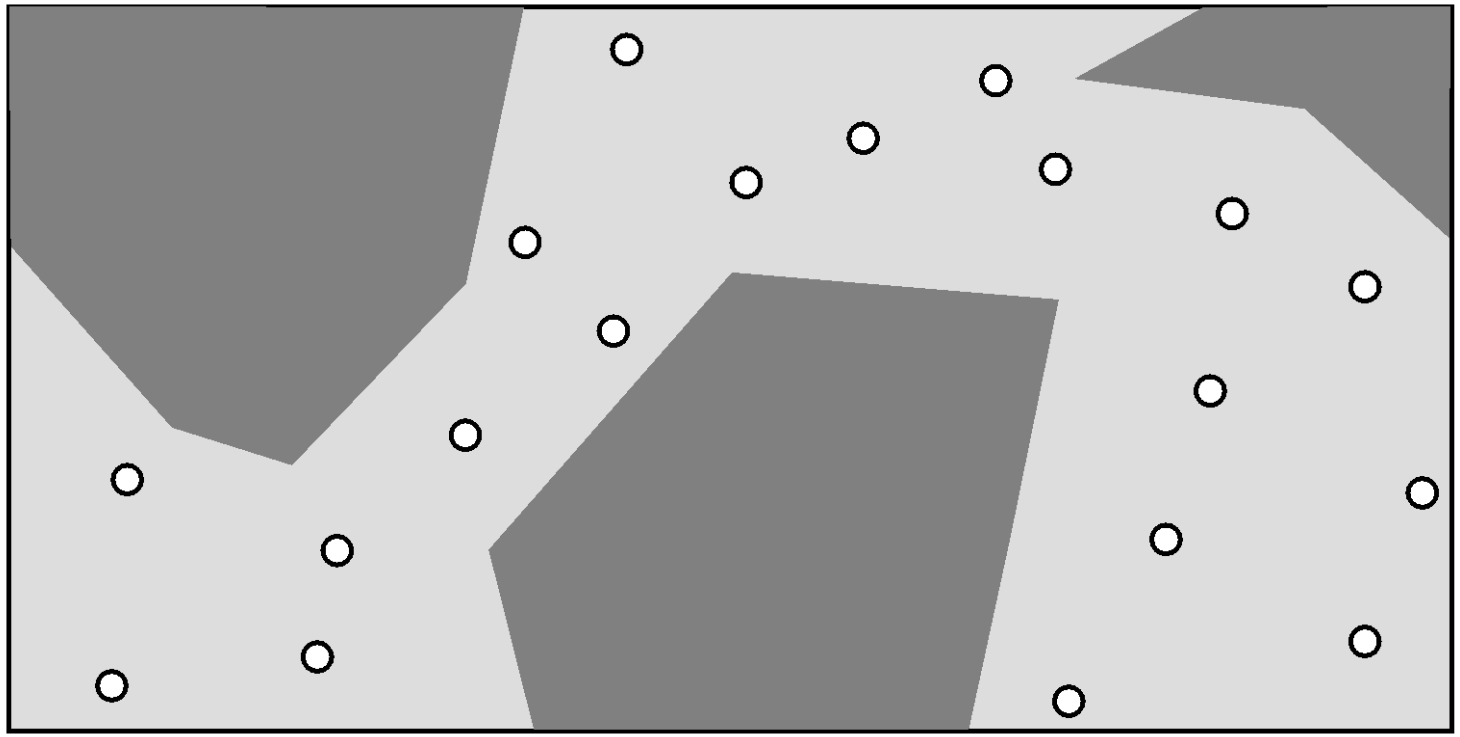
- Unlike path/trajectory planning, SMR produces a feedback controller (e.g., a policy)



- Optimal action considers effects of future uncertainty

set of states S

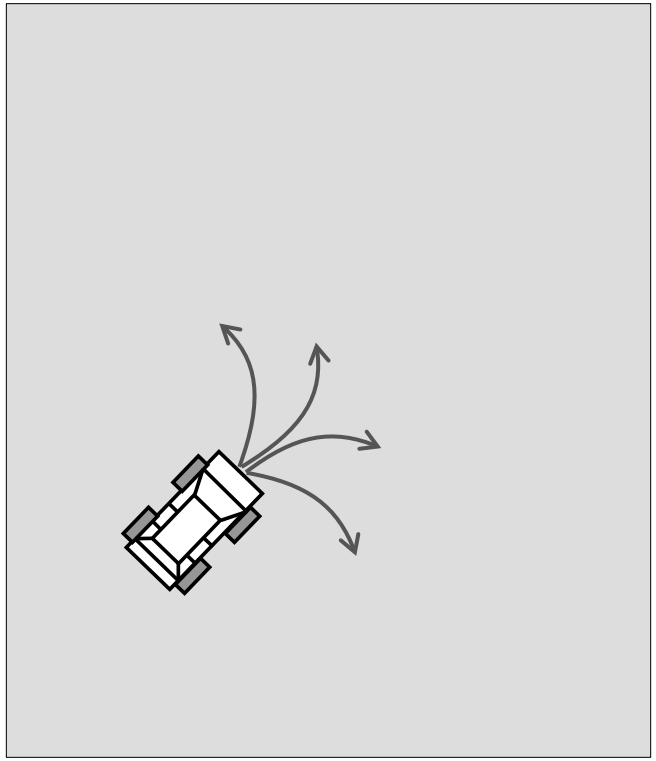
- Construction of SMR nodes:
  - Generate n collision-free states



- Problem-specific function: $\mathrm{isCollisionFree}(s)$

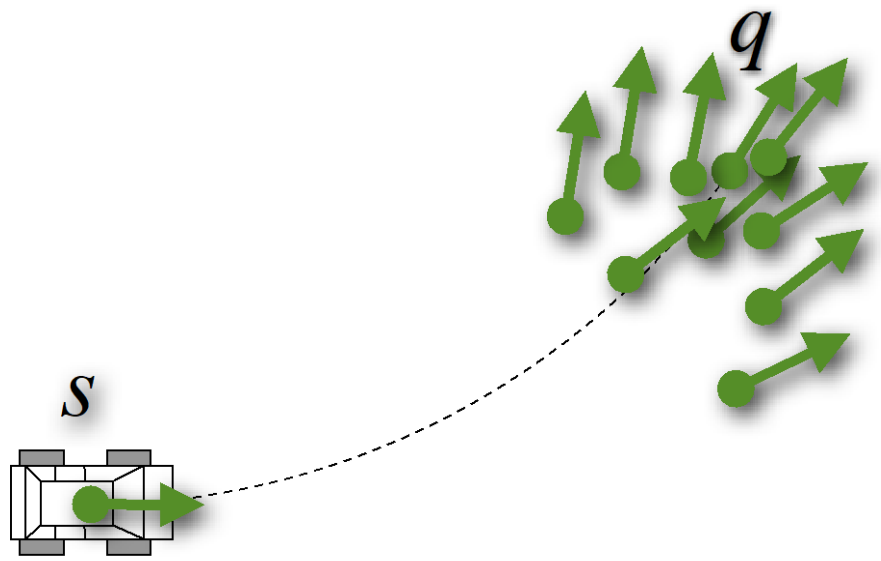
Set of Actions A

- Choose a discrete set of actions



Transition Functions and Probabilities P(s,a,s')

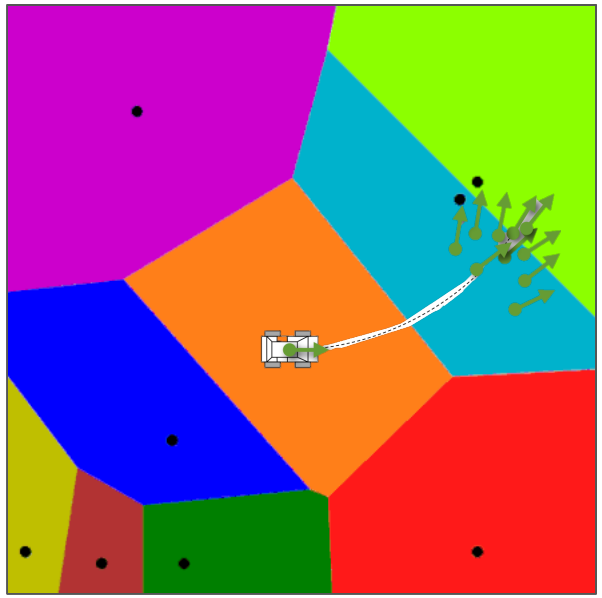
- Modeling Transition Uncertainty Using Sampling (transition probabilities are found by Monte Carlo simulation)



  - Robot-specific function: $\mathrm{q} = \mathrm{generateSampleTransition}(s, u)$
  - For each state s and action a:
    - Generate m motion samples

Transition Functions and Probabilities P(s,a,s')

- Voronoi tessellation (diagram) is a partition of a space into regions close to each of a given set of points.



- Given a set of n points $Q = \{q_1, \dots, q_n\}$, where $q_i \in C_{free}$, Voronoi tessellation defines a set of n regions $\{r_1, \dots, r_n\}$, where

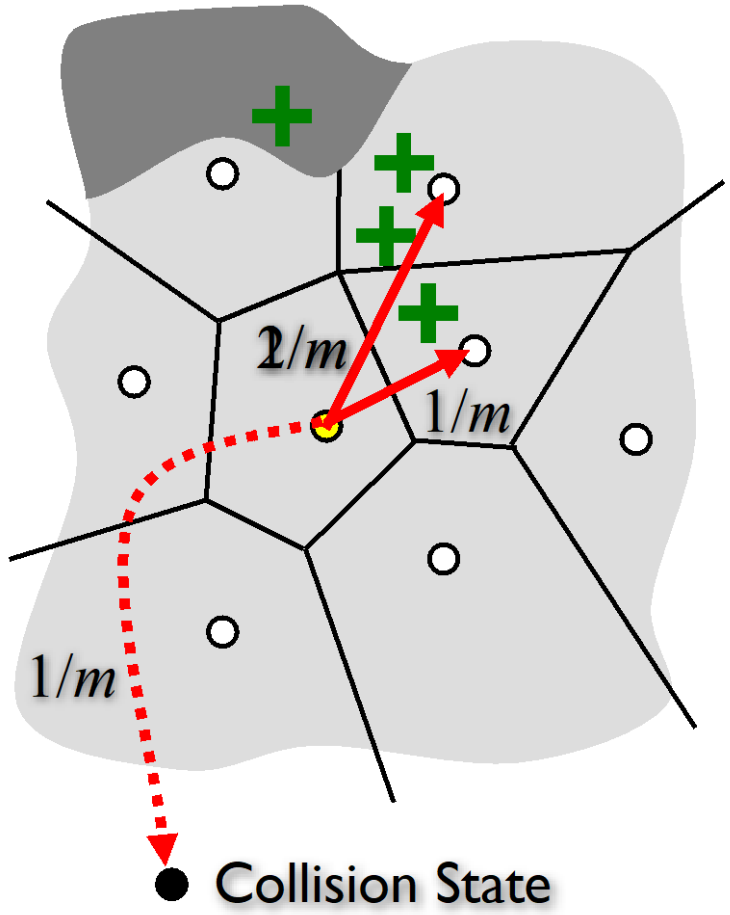
$$r_i = \{q \in C_{free} \mid d(q, q_i) \leq d(q, q_j) \;\; \forall q_j \neq q_i \wedge q_j \in Q\}$$

  - $d(q, q_i)$ is the distance from point q to point q_i

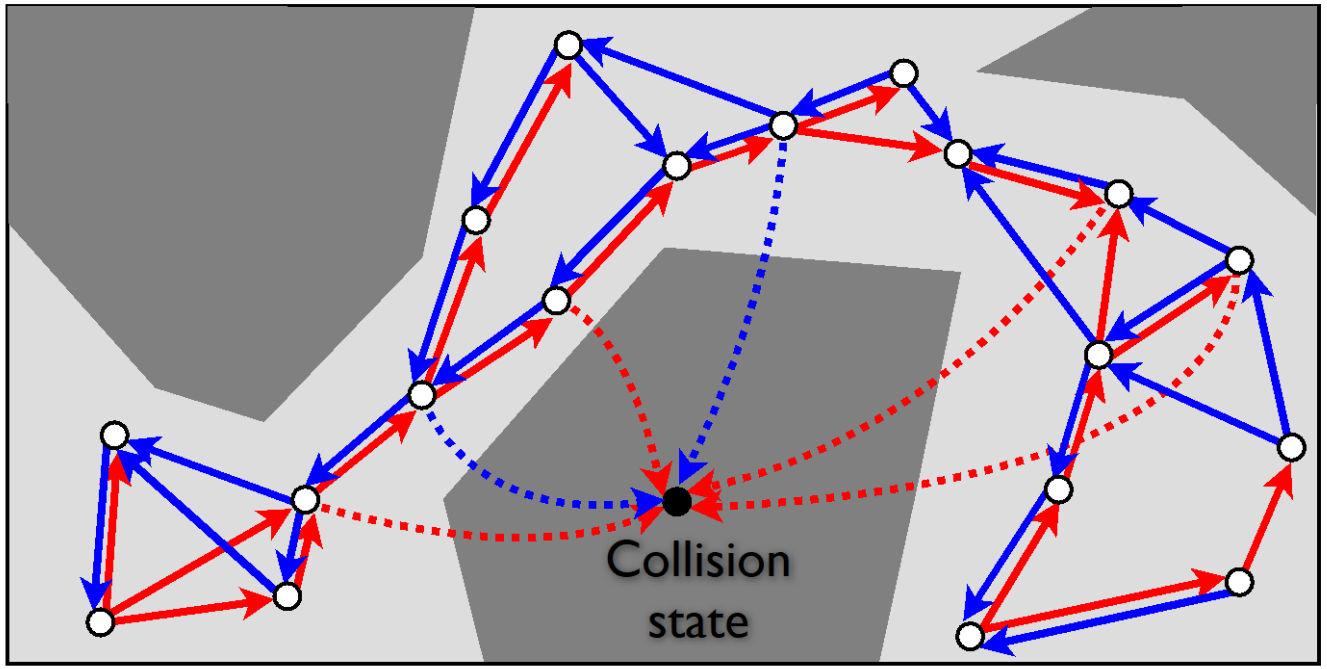
Rewards

- For each motion sample and possible transition
  - Check for collision

- Choose rewards so that the total expected reward becomes probability of success
  - Define:
    - $R(s_i) = 1$ if s_i is inside target
    - $R(s_i) = -1$ if s_i is the collision state
    - $R(s_i, a) = 0$ otherwise
  - Solve using value iteration



Completed Roadmap



- We can now solve using value iteration

Overview

- Planning Under uncertainty
- Markov Decision Processes
  - Value Iteration
- Stochastic motion roadmaps
- **Partially Observable Decision Processes**

Partially Observable MDPs

- State is only partially observable
- Errors in sensor measurements and in perception (or controller)
- Estimate the robot's state as probability distribution function over states, called ***beliefs***

Source of Uncertainty

- **What are the sources of uncertainty**

- **Sensing (State)**

  - Where is the robot?
  - Where are the obstacles?
  - What is the map of the environment?

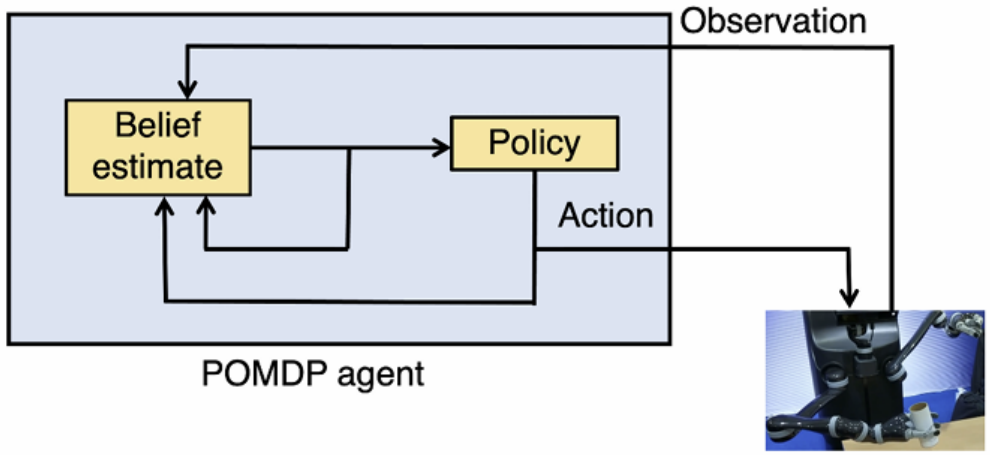
- **Motion (Actions)**

  - How will objects in the environment move?

  - Given a command action, where will the robot end up?

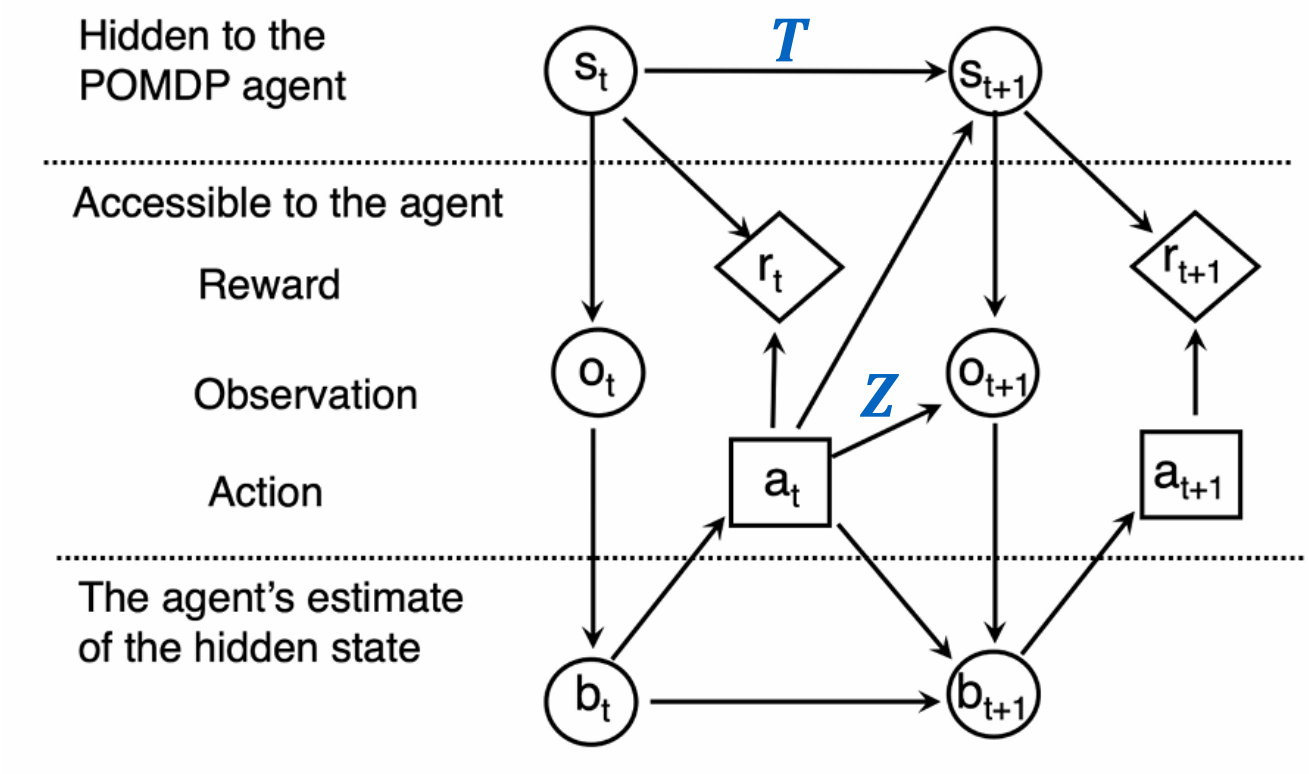
POMDP Definition

- 6-tuple model:
  - States: S
  - Actions: A
  - Transition function: $T(s, a, s')$ or $P(s'|s, a)$
    - Represents the non-deterministic effects of a[illegible]
  - Reward: $R(s, a, s')$
  - Observation space: O
  - Observation function: $Z(s', a, o)$ or $P(o|s', a)$
    - Represent errors and noise in measurement and perception

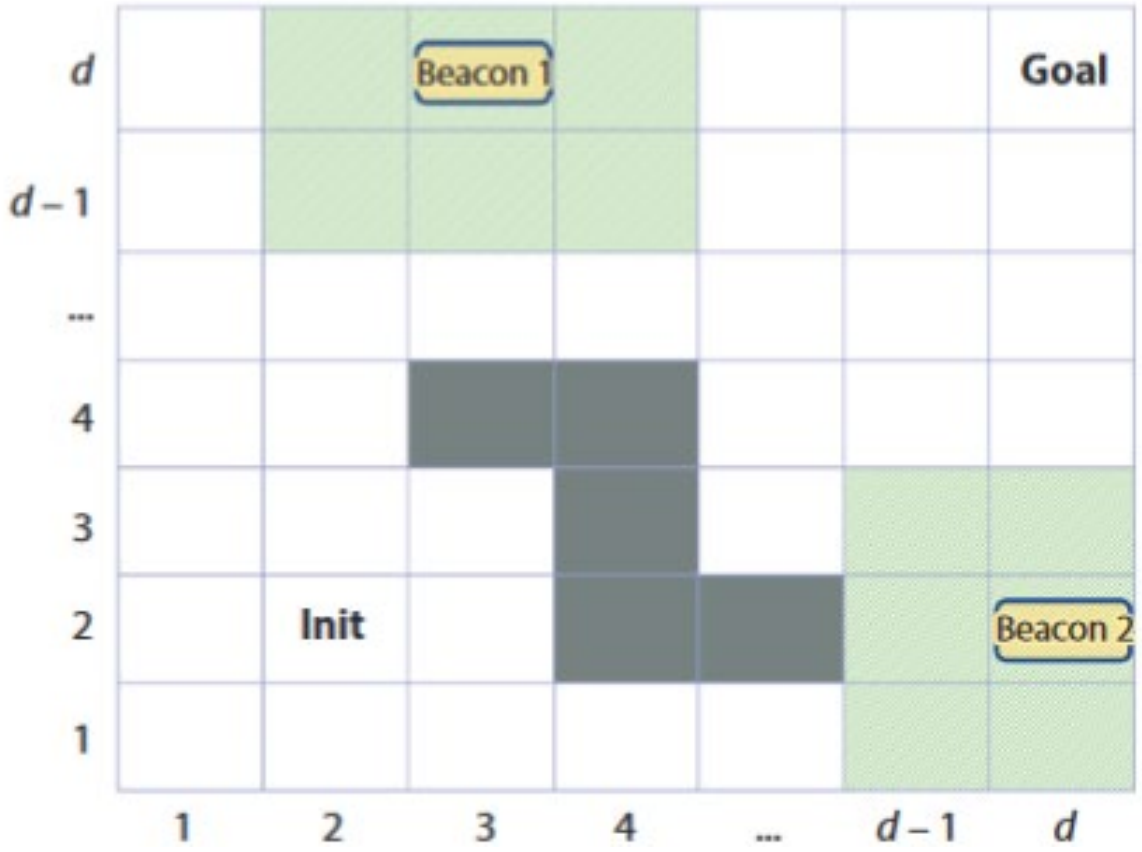


POMDP Steps

- An agent with model $< S, A, O, T, Z, R >$

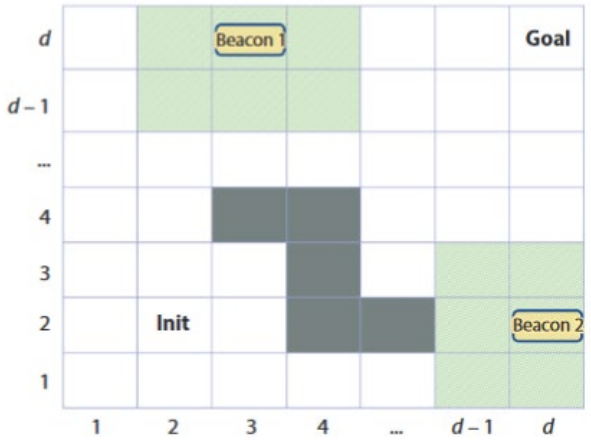


A POMDP example



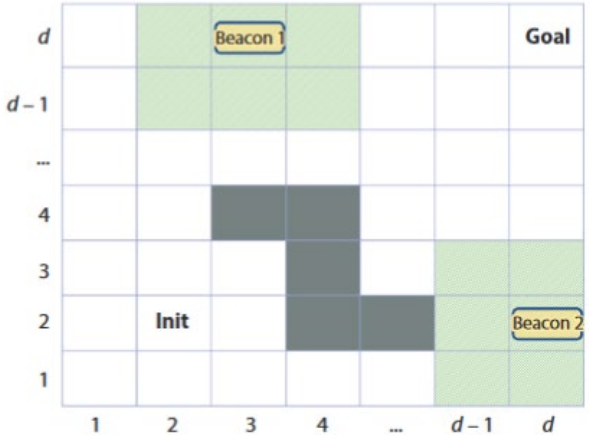
$$b = \begin{bmatrix} P(s = \text{cell-}(1,1)) \\ P(s = \text{cell-}(2,1)) \\ P(s = \text{cell-}(3,1)) \\ \\ \dots \\ \\ P(s = \text{cell-}(d,d)) \end{bmatrix}$$

A POMDP example



- $\mathcal{S}$ is the robot's location, which is the set {cell-(1, 1), cell-(1, 2), cell-(1, d), . . . , cell-(d, d)}.
- $\mathcal{A}$ is the action space, which in this example is the simplest case of moving one cell in one of four directions: {north, south, east, west}.
- $\mathcal{O}$ is the observation space, which in this example is a joint product of the distance to beacon 1 and the distance to beacon 2: {(NA from beacon 1, NA from beacon 2), (NA from beacon 1, in cell containing beacon 2), (NA from beacon 1, within 1 cell from beacon 2), . . . , (within k cells from beacon 1, within k cells from beacon 2), in goal cell}, where NA means not available. If the goal cell sensor is not perfect, then the observation space should be a joint product between the distance to both beacons as well as the goal-cell observation.

A POMDP example



- $Z(s', a, o)$ is a conditional probability function representing sensing errors—for example, $P(o = (\text{NA from beacon 1, within 1 cell from beacon 2}) \mid s' = \text{cell-}(d-1, 2), a = \text{north}) = 0.8$ and $P(o = (\text{NA from beacon 1, within 0 cells from beacon 2}) \mid s' = \text{cell-}(d-1, 2), a = \text{north}) = 0.2$. As with the transition function, the observation function can be represented as four probability matrices, where each matrix represents the action the robot has just performed. However, the size of each matrix is $|\mathcal{S}| \times |\mathcal{O}|$.

Updated Value Function

$$V^*(b) = \max_{a \in \mathcal{A}} \underbrace{\left(R(b,a) + \gamma \underbrace{\sum_{o \in \mathcal{O}} P(o|b,a) V^*(\tau(b,a,o))}_{J(b,a)} \right)}_{Q(b,a)},$$

Solving POMDP

- Value iteration over belief space
  - Treat the belief state $b(s)$ as the state in an equivalent MDP and perform value iteration
- Sampling-based approximate
  - Approximate the value function by focusing on a finite set of representative belief points
- Policy search method
  - Search directly in the space of policies
- Monte Carlo Tree search
  - Use random sampling to simulate possible future action-observation sequences

References

- Alterovitz, Ron, Thierry Siméon, and Ken Goldberg. "The stochastic motion roadmap: A sampling framework for planning with Markov motion uncertainty." (2008).

- Kurniawati, Hanna. "Partially observable markov decision processes and robotics." *Annual Review of Control, Robotics, and Autonomous Systems* 5.1 (2022): 253-277.

Sampling-Based Approximate POMDP Solvers

- Sample a set of representative beliefs

Algorithm 1 A typical program skeleton for sampling-based POMDP solvers

1: Initialize policy π and a set of sampled beliefs B
{Generally, B is initialised to contain only a single belief (e.g., the initial belief b_0)}
2: **repeat**
3: Sample a (set of) beliefs {Some methods sample histories (a history is a sequence of action–observation tuples) rather than beliefs. In POMDPs, beliefs provide sufficient statistics of the entire history [25], and therefore the two provide equivalent information}
4: Estimate the values of the sampled beliefs
{Generally, via a combination of heuristics and update / backup operation}
5: Update π {In most methods, this step is a byproduct of the previous step}
6: **until** Stopping criteria is satisfied

- Offline optimal solvers
  - Estimate current belief and execute the action
- Online solvers
  - Compute the good action to perform from current belief