AI Robotics

Markov Decision Process

Overview

Robot Model

Sensor Measurements:

$Z_{0:t} = \{z_0, \ldots, z_t\}$

Robot State (or pose):

$X_{0:t} = \{X_0, \ldots, X_t\}$

sense

Environment

act

Robot Controls:

$u_{0:t} = \{u_0, u_1, \ldots, u_t\}$

Sense, Think, Act

Suppose you are given a task: *Rearrange the chairs in the room into a circle.* How would you proceed?

1. Look around the room and evaluate the situation. Where are the chairs? How many chairs are there?	Sense
2. Make a plan: 1. Go the first chair, pick it up, place it in the desired position 2. Repeat for all N chairs.	Plan
3. Execute the plan.	Act

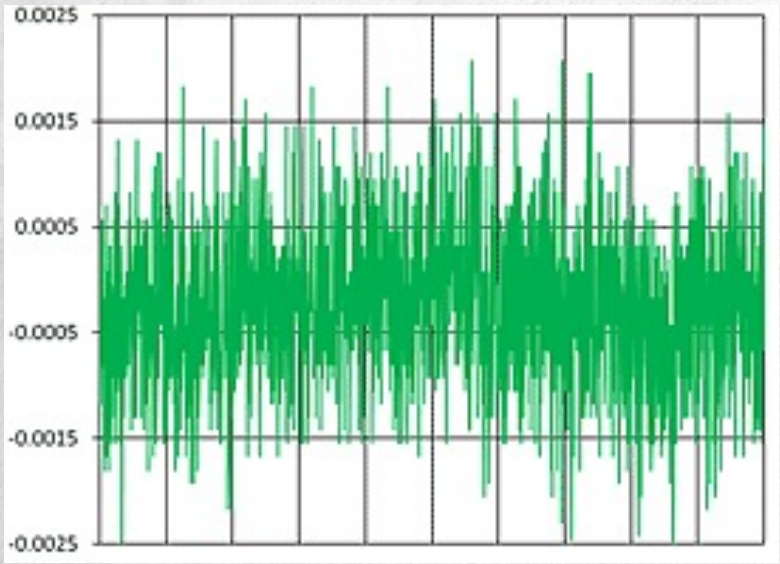
4. Learn from the experience. Does the plan work? Can we do better?

Learn

MDP Paradigm

Uncertainty

Sensor uncertainty: sensor measurements are noisy



Motion uncertainty: robot control is noisy due to vibration, external forces

Probabilistic Robot Model

$$P(x_t|u_1, z_1, u_2, z_2, \cdots, u_t, z_t) = P(x_t|z_{1:t}, u_{1:t})$$

Robot State

sense

Sensor Model

$$P(z_t|x_t)$$

act

Motion Model

$$P(x_t|x_{t-1}, u_t)$$

Action Policy

- Not a single path to the goal (e.g. A*)
- The policy specifies an action to take for any state that the robot could be in
- Can be implemented as a look-up table
- Applicable to problems of low-level control
- Assume discrete finite state and action spaces

Paradigm: MDPs

- The paradigm to handle uncertainty in robot motion is known as **Markov Decision Processes**, or **MDPs**
- MDPs assume we can observe the state of the robot and environment
- Allow for random effects of actions (**stochastic action**)
- Have a plan for whatever state the robot is in

Markov Decision Processes

The components of an MDP are:

- Set of **states** X (we will assume this is finite)
- An **initial state** $x_0 \in X$.
- A set of **actions** for each state $x \in X$, we call this $U(x)$.
- A **transition model** P, where $P(x' \mid x, u)$ is the probability of reaching state x' after performing control action u in state x.
- A **reward function** R specifying the reward for each state action combination, where $R(x, u)$ is the reward for state x.

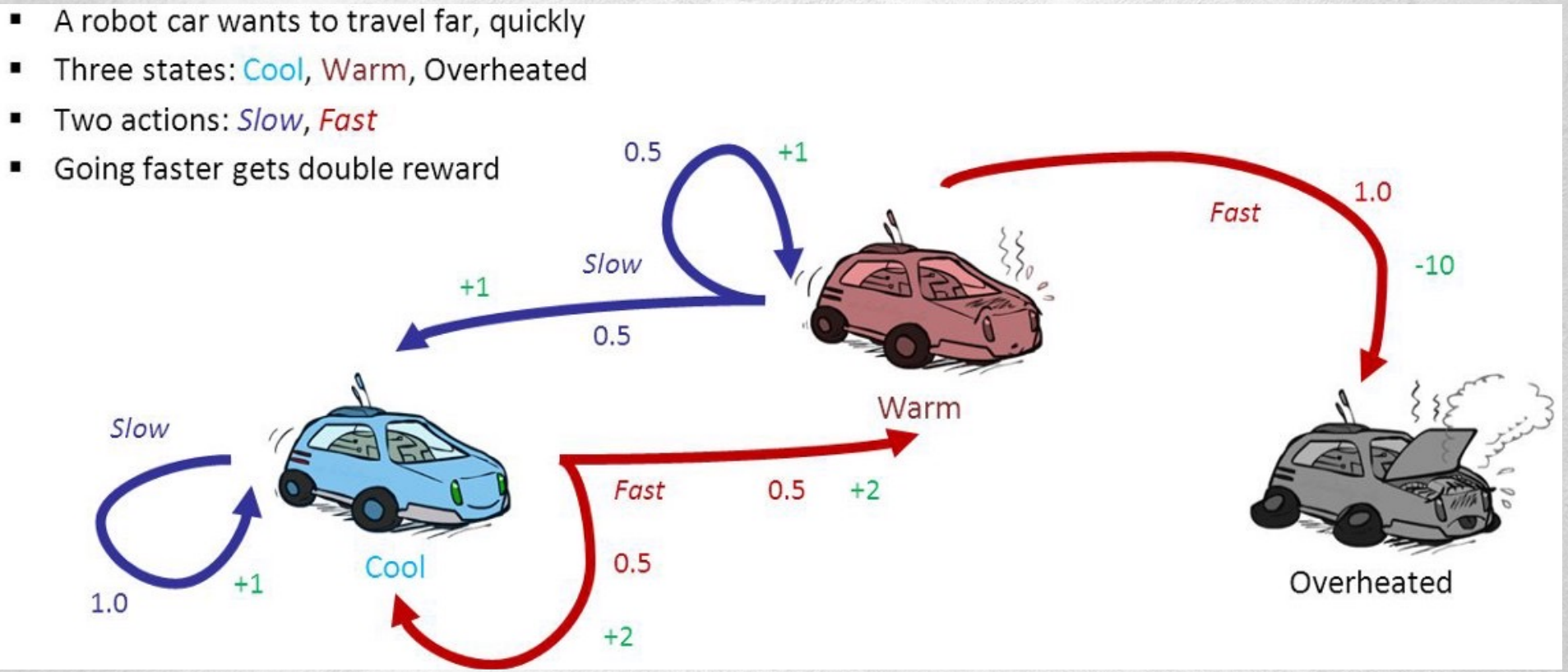
Markov Decision Processes

Properties

- Assume state is fully-observable (no need for a probabilistic sensor model)
- Assume Markov Property: the current state of the Robot depends only on its immediate previous state
- Some textbooks use ***s*** instead of ***x*** to denote the state
- Some textbooks use ***a*** instead of ***u*** to denote the action

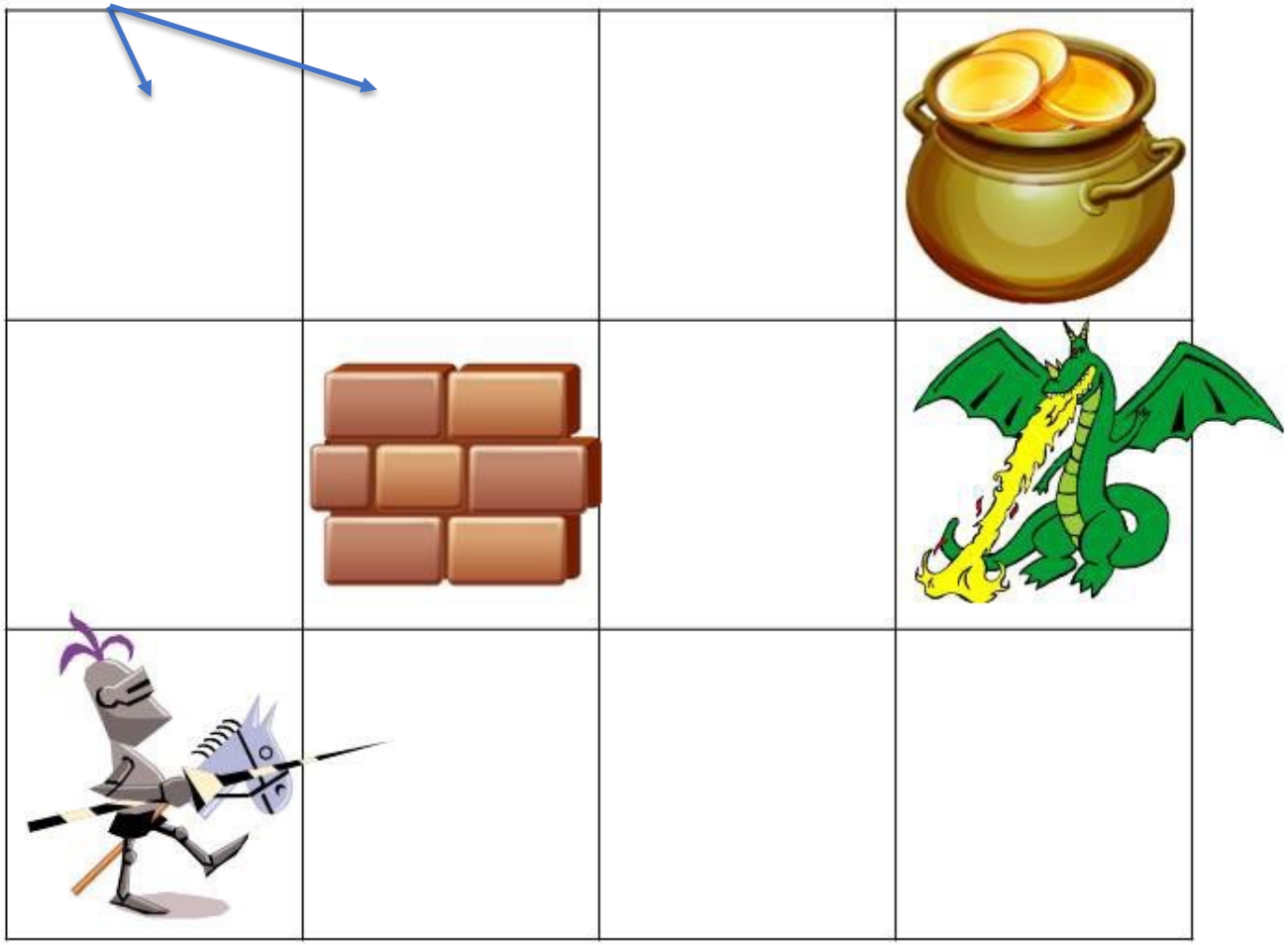
Example: Robot racing

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward

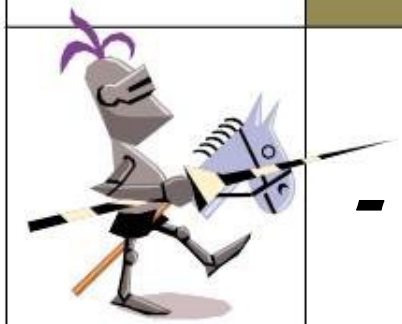


Example: Navigation on a Grid

States, x

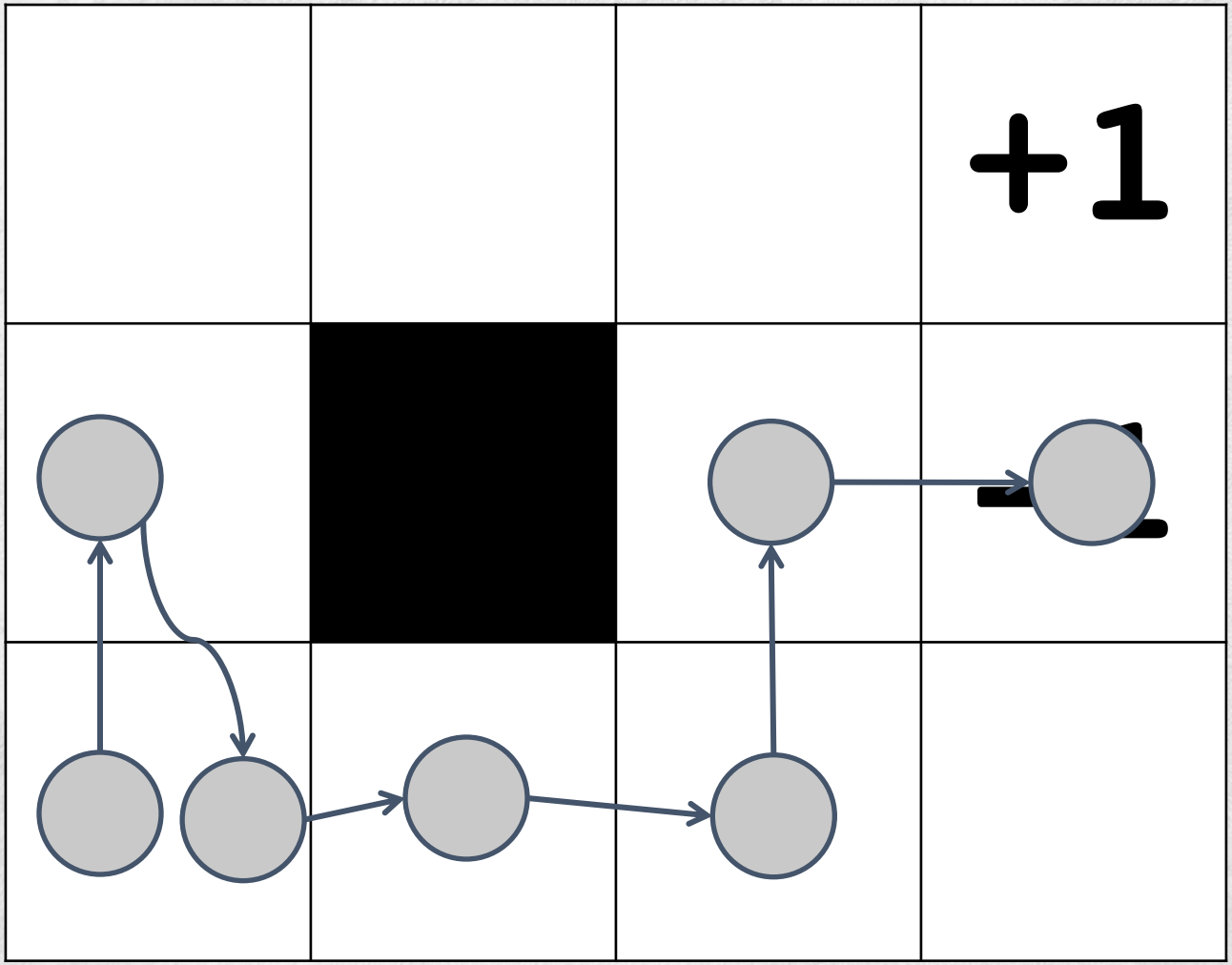


Example: Navigation on a Grid

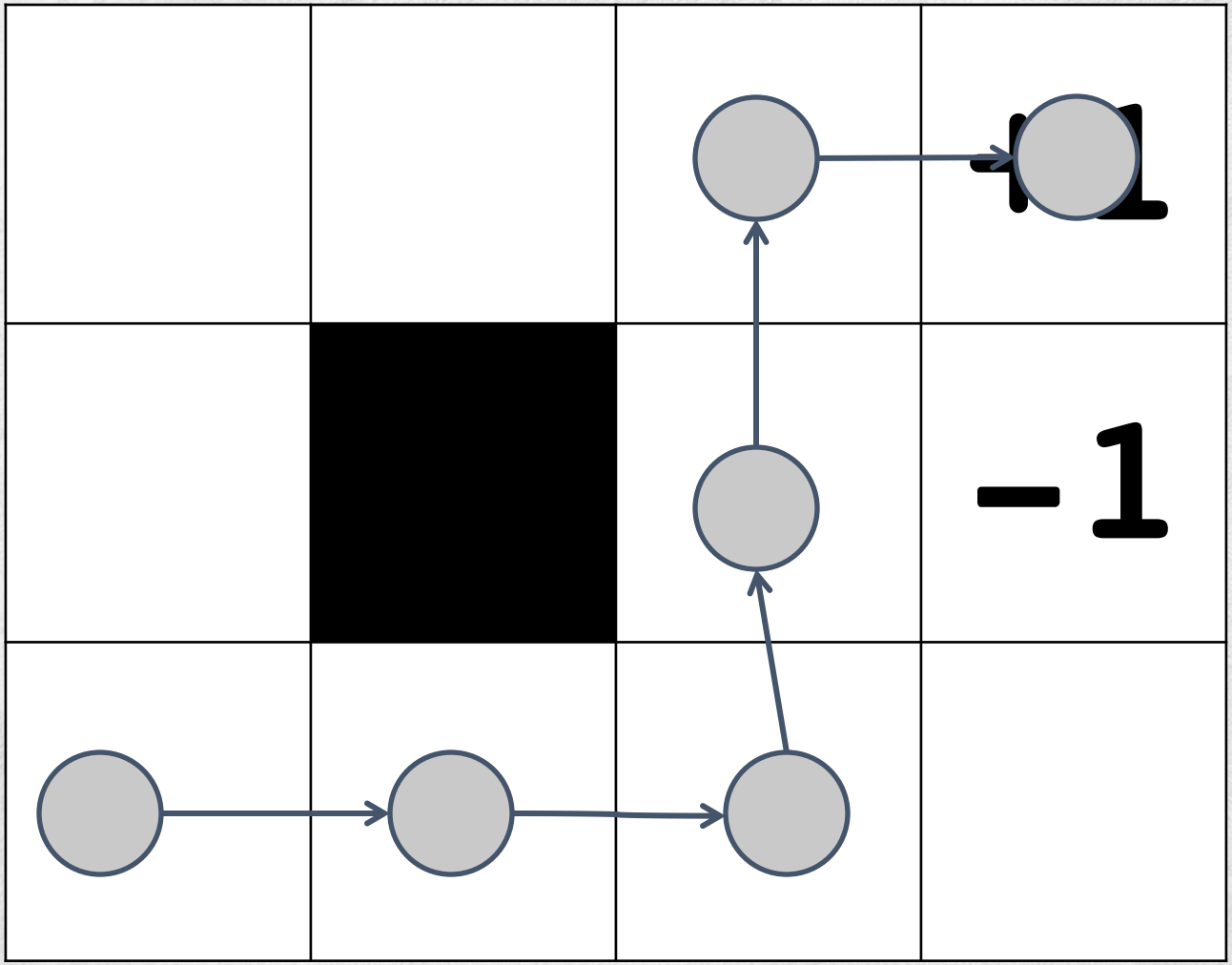
- 0.05	- 0.05	- 0.05	+1
- 0.05		- 0.05	−1
	- 0.05	- 0.05	- 0.05

Rewards, r

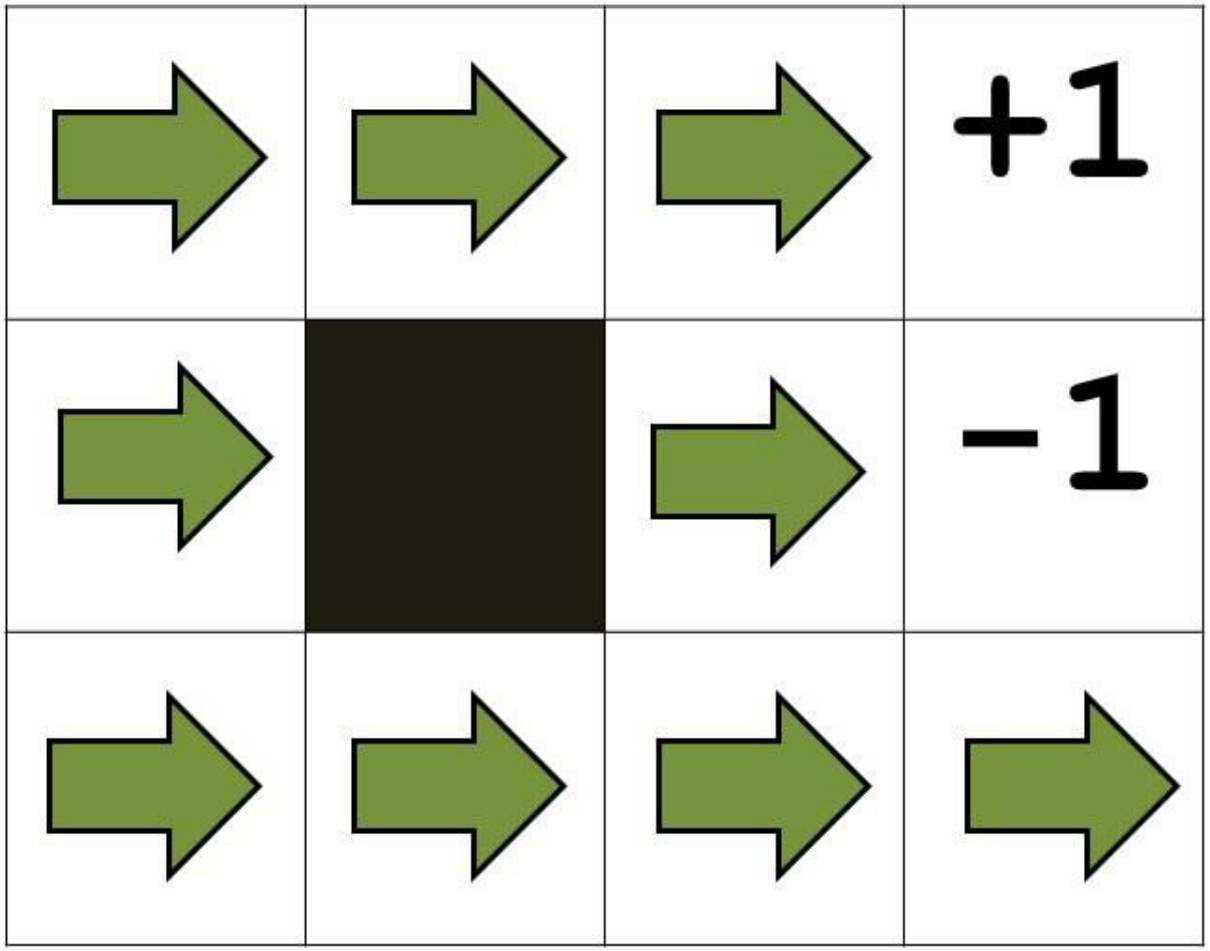
Example: Navigation on a Grid



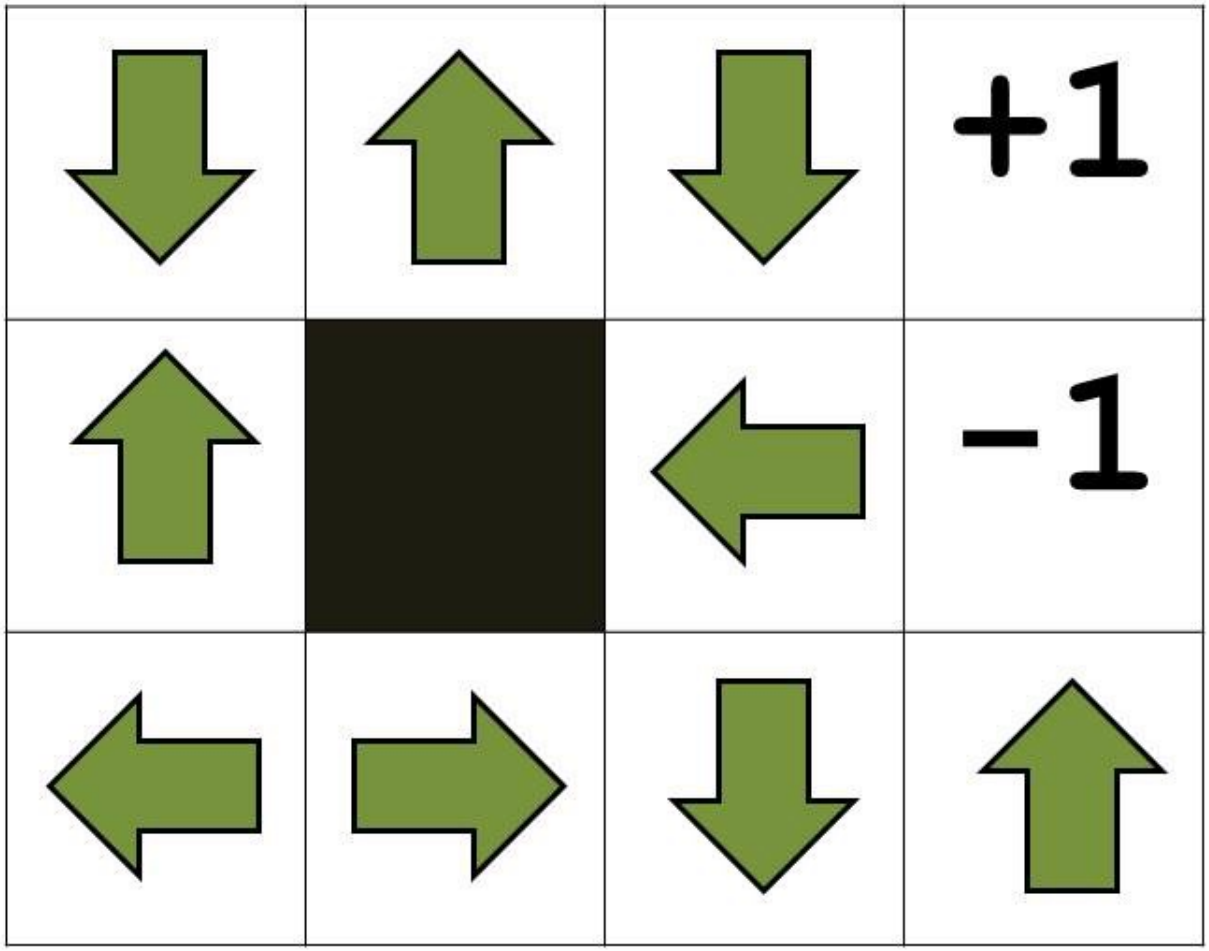
Example: Navigation on a Grid



Example: Policy π_1



Example: Policy π_2



Optimal Policies

- Let's start with the optimal policy for $T = 1$
- We will call this policy $\pi_1(x)$

$$\pi_1(x) = \arg\max_u R(x, u)$$

- This policy simply chooses the action that maximizes the immediate reward

Reward Horizon

One thing that will have a big impact on a policy is how far into the future is the robot trying to maximize its collected rewards.

1. Choose action which maximizes reward at next time step
   a. Big payoff might only come far into the future
2. Choose actions so that the sum of future rewards is maximized
   a. Need to maximize the **expected cumulative rewards**

Expected Cumulative Reward

$$R_T = \mathbb{E}\left[\sum_{\tau=1}^{T} \gamma^{\tau} r_{t+\tau}\right]$$

where the expectation E is taken over future reward values $r_{t+\tau}$

- γ is the **discount factor**, which is problem specific.
  - $\gamma = 1$: all rewards are weighted equally
  - $\gamma < 1$: future rewards are worth (exponentially) less
- similar to the value of money, which decreases over time

Value Functions

- Every policy has an associated **value function**, which measures the expected value (cumulative discounted future payoff) of executing this specific policy from a state x
- For $\pi_1(x)$

$$V_1(x) = \gamma \max_u R(x, u)$$

- Let's recursively define the value function for $T > 1$
- Policy for $T = 2$ should maximize the sum of the one-step value (what it can get on the last time step), and the immediate reward (what it can get now)

Value Functions

Value function for T = 2

$$V_2(x) = \gamma \max_u \left[R(x,u) + \sum_{x'} V_1(x') P(x' \mid x, u) \right]$$

Optimal policy for T = 2

$$\pi_2(x) = \arg\max_u \left[R(x,u) + \sum_{x'} V_1(x') P(x' \mid x, u) \right]$$

General Case

$$\pi_T(x) = \arg\max_u \left[R(x,u) + \sum_{x'} V_{T-1}(x') P(x' \mid x, u) \right]$$

$$V_T(x) = \gamma \max_u \left[R(x,u) + \sum_{x'} V_{T-1}(x') P(x' \mid x, u) \right]$$

Fixed Point

$$V_\infty(x) = \gamma \max_u \left[R(x,u) + \sum_{x'} V_\infty(x') P(x' \mid x, u) \right]$$

- This is known as the **Bellman Equation**
- Satisfying this equation is both **necessary** and **sufficient** for a policy derived from a value function to be optimal

Optimal Policy

Let $V^*(x)$ satisfy the Bellman Equation, then

$$\pi^*(s) = \arg\max_u \left[R(x,u) + \sum_{x'} V^*(x') P(x' \mid x, u) \right]$$

is an optimal policy

Method 1: Value Iteration

The system of Bellman's equations cannot be solved using linear algebra techniques, since they are non-linear due to the max operator. Instead, an iterative approach can be used:

- for $i = 1$ to N do
  - $\hat{V}(x_i) = R_{min}$
- endfor
- repeat until convergence
  - for $i = 1$ to N do

$$\hat{V}(x_i) = \gamma \max_u \left[R(x_i, u) + \sum_{x'} \hat{V}(x') P(x' \mid x_i, u) \right]$$

  - endfor

Method 2: Policy Iteration

1. **Policy Evaluation** - given the current policy π_i, determine $V_i = V^{\pi}$, the utility of each state if π_i is executed.

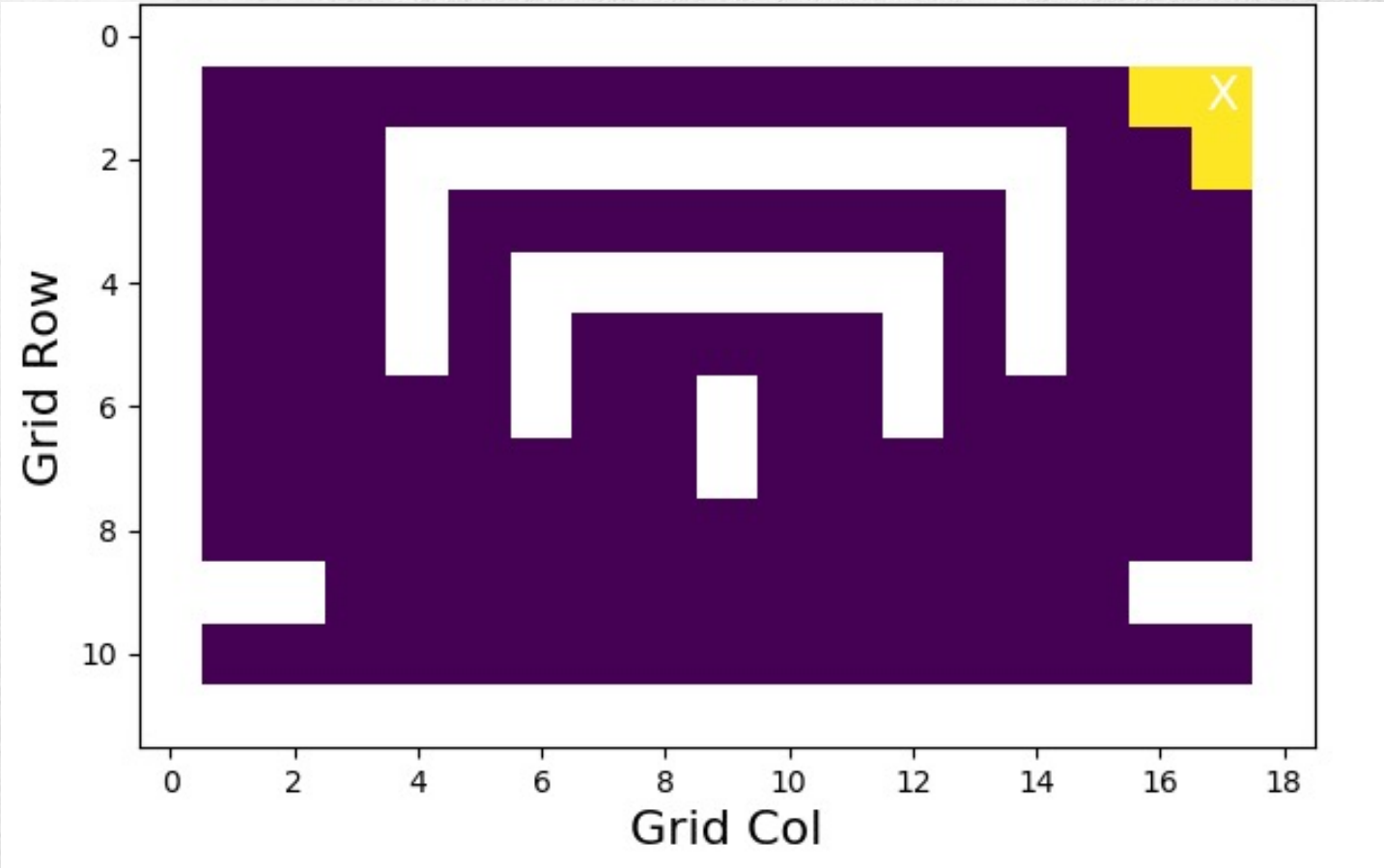
$$V(x) \leftarrow R(x, \pi_i(x)) + \gamma \sum_{x'} V(x')\, P(x' \mid x, \pi_i(x))$$

2. **Policy improvement** - calculate a new policy π_{i+1}, using one step look ahead based on V_i.

$$\pi_{i+1}(x) = arg\max_{u} \sum_{x'} V(x')\, P(x' \mid x, u)$$

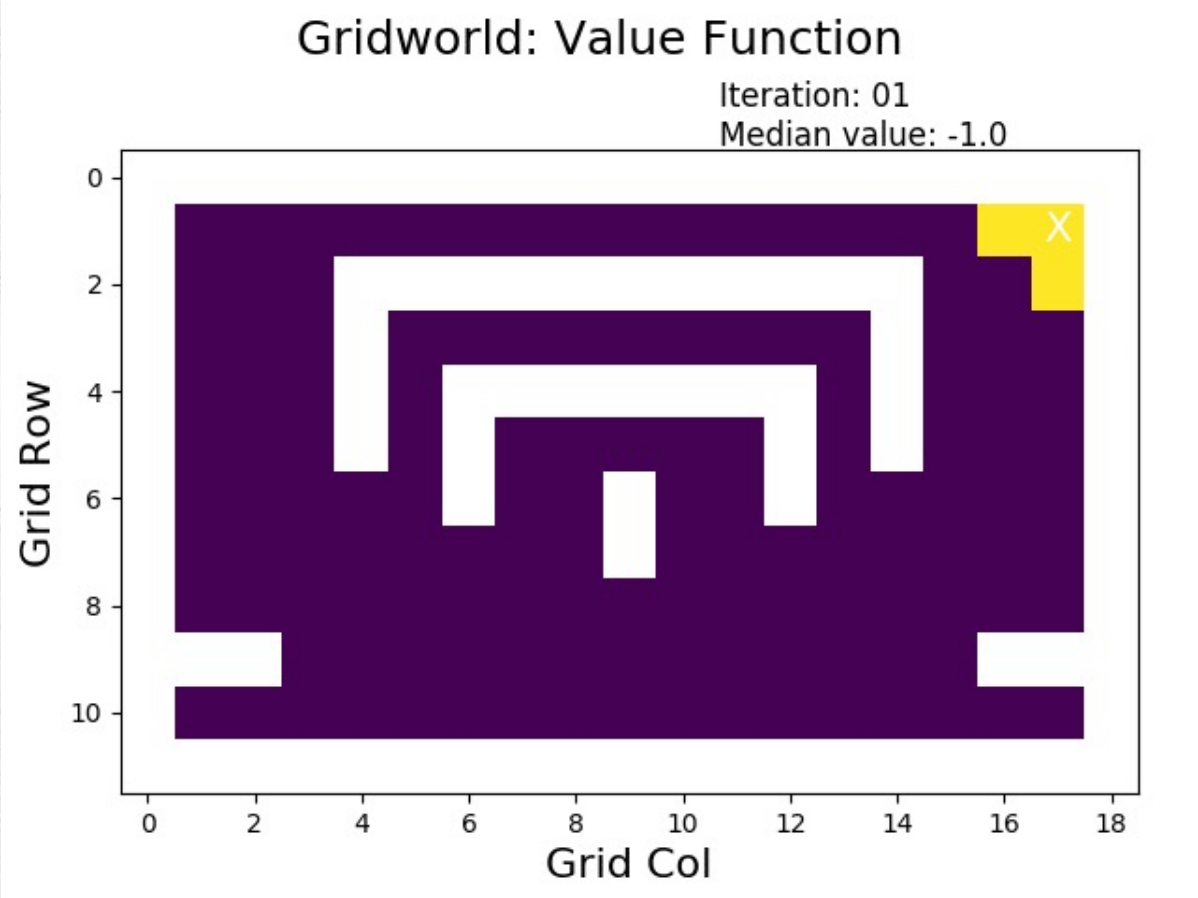
Example

Grid World Navigation



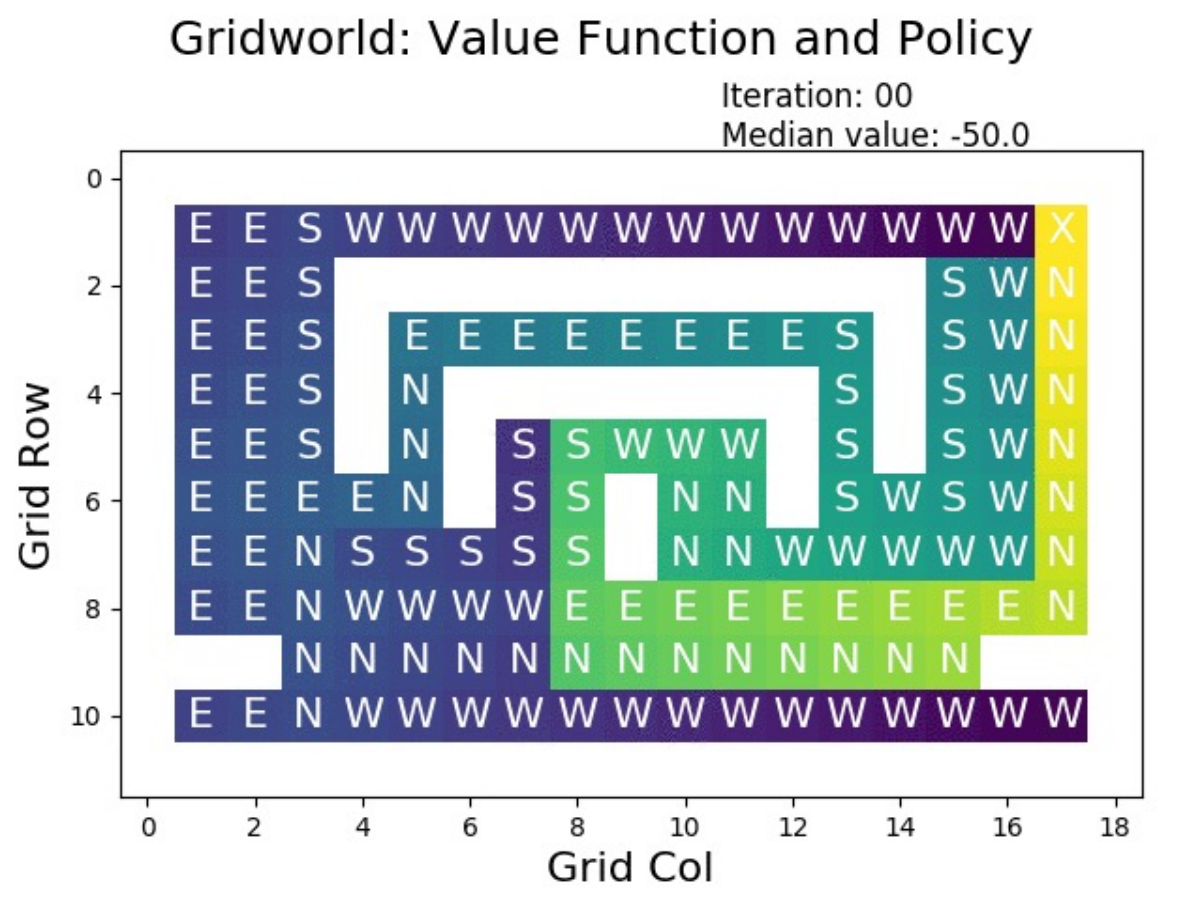
Value Iteration

1. Initialize value function
2. Update value function according to Bellman equation
3. Repeat until convergence to optimal value function
4. Calculate optimal policy from optimal value function



Policy Iteration

1. Initialize policy
2. Calculate value function according to current policy
3. Improve the policy using greedy one-step look-ahead
4. Repeat until convergence



References

1. https://towardsdatascience.com/understanding-the-markov-decision-process-mdp-8f838510f150
2. https://www.datascienceblog.net/post/reinforcement-learning/mdps_dynamic_programming/
3. https://www.youtube.com/watch?v=4KGC_3GWuPY