Lecture 29: Machine Learning

Marvin Zhang
08/10/2015

(Some images borrowed from CS 188.)

Announcements

Announcements

- Project 3 composition revisions due Wednesday night.

Announcements

- Project 3 composition revisions due Wednesday night.
- Project 4 due tonight (Monday).

Announcements

- Project 3 composition revisions due Wednesday night.

- Project 4 due tonight (Monday).
  - Office hours from 4-7pm today in the Woz.

Announcements

- Project 3 composition revisions due Wednesday night.

- Project 4 due tonight (Monday).
  - Office hours from 4-7pm today in the Woz.

- Homework 11 due tonight - just a survey!

Announcements

- Project 3 composition revisions due Wednesday night.
- Project 4 due tonight (Monday).
  - Office hours from 4-7pm today in the Woz.
- Homework 11 due tonight - just a survey!
- Project 4 contest due tomorrow (Tuesday) night.

Announcements

- Project 3 composition revisions due Wednesday night.

- Project 4 due tonight (Monday).
  - Office hours from 4-7pm today in the Woz.

- Homework 11 due tonight - just a survey!

- Project 4 contest due tomorrow (Tuesday) night.
  - Top 3 entries in each category get extra credit! Only one entry so far.

Announcements

- Project 3 composition revisions due Wednesday night.
- Project 4 due tonight (Monday).
  - Office hours from 4-7pm today in the Woz.
- Homework 11 due tonight - just a survey!
- Project 4 contest due tomorrow (Tuesday) night.
  - Top 3 entries in each category get extra credit! Only one entry so far.
- Final on Thursday, 3-6pm in 2050 VLSB.

Announcements

- Project 3 composition revisions due Wednesday night.

- Project 4 due tonight (Monday).
  - Office hours from 4-7pm today in the Woz.

- Homework 11 due tonight - just a survey!

- Project 4 contest due tomorrow (Tuesday) night.
  - Top 3 entries in each category get extra credit! Only one entry so far.

- Final on Thursday, 3-6pm in 2050 VLSB.
  - Review session tomorrow, 5-8pm in the Woz.

Announcements

- Project 3 composition revisions due Wednesday night.
- Project 4 due tonight (Monday).
  - Office hours from 4-7pm today in the Woz.
- Homework 11 due tonight - just a survey!
- Project 4 contest due tomorrow (Tuesday) night.
  - Top 3 entries in each category get extra credit! Only one entry so far.
- Final on Thursday, 3-6pm in 2050 VLSB.
  - Review session tomorrow, 5-8pm in the Woz.
- Tuesday, Wednesday, and Thursday sections canceled.

Announcements

- Project 3 composition revisions due Wednesday night.
- Project 4 due tonight (Monday).
  - Office hours from 4-7pm today in the Woz.
- Homework 11 due tonight - just a survey!
- Project 4 contest due tomorrow (Tuesday) night.
  - Top 3 entries in each category get extra credit! Only one entry so far.
- Final on Thursday, 3-6pm in 2050 VLSB.
  - Review session tomorrow, 5-8pm in the Woz.
- Tuesday, Wednesday, and Thursday sections canceled.
  - Instead, TAs will lead topic-themed discussions Tuesday and Wednesday.

Announcements

- Project 3 composition revisions due Wednesday night.
- Project 4 due tonight (Monday).
  - Office hours from 4-7pm today in the Woz.
- Homework 11 due tonight - just a survey!
- Project 4 contest due tomorrow (Tuesday) night.
  - Top 3 entries in each category get extra credit! Only one entry so far.
- Final on Thursday, 3-6pm in 2050 VLSB.
  - Review session tomorrow, 5-8pm in the Woz.
- Tuesday, Wednesday, and Thursday sections canceled.
  - Instead, TAs will lead topic-themed discussions Tuesday and Wednesday.
  - Details will be posted on Piazza.

Announcements

- Project 3 composition revisions due Wednesday night.

- Project 4 due tonight (Monday).
  - Office hours from 4-7pm today in the Woz.

- Homework 11 due tonight - just a survey!

- Project 4 contest due tomorrow (Tuesday) night.
  - Top 3 entries in each category get extra credit! Only one entry so far.

- Final on Thursday, 3-6pm in 2050 VLSB.
  - Review session tomorrow, 5-8pm in the Woz.

- Tuesday, Wednesday, and Thursday sections canceled.
  - Instead, TAs will lead topic-themed discussions Tuesday and Wednesday.
  - Details will be posted on Piazza.
  - Chris and Cale's 9:30-11am labs on Tuesday are NOT canceled.

What is Machine Learning?

What is Machine Learning?

- Natural Language Processing

What is Machine Learning?

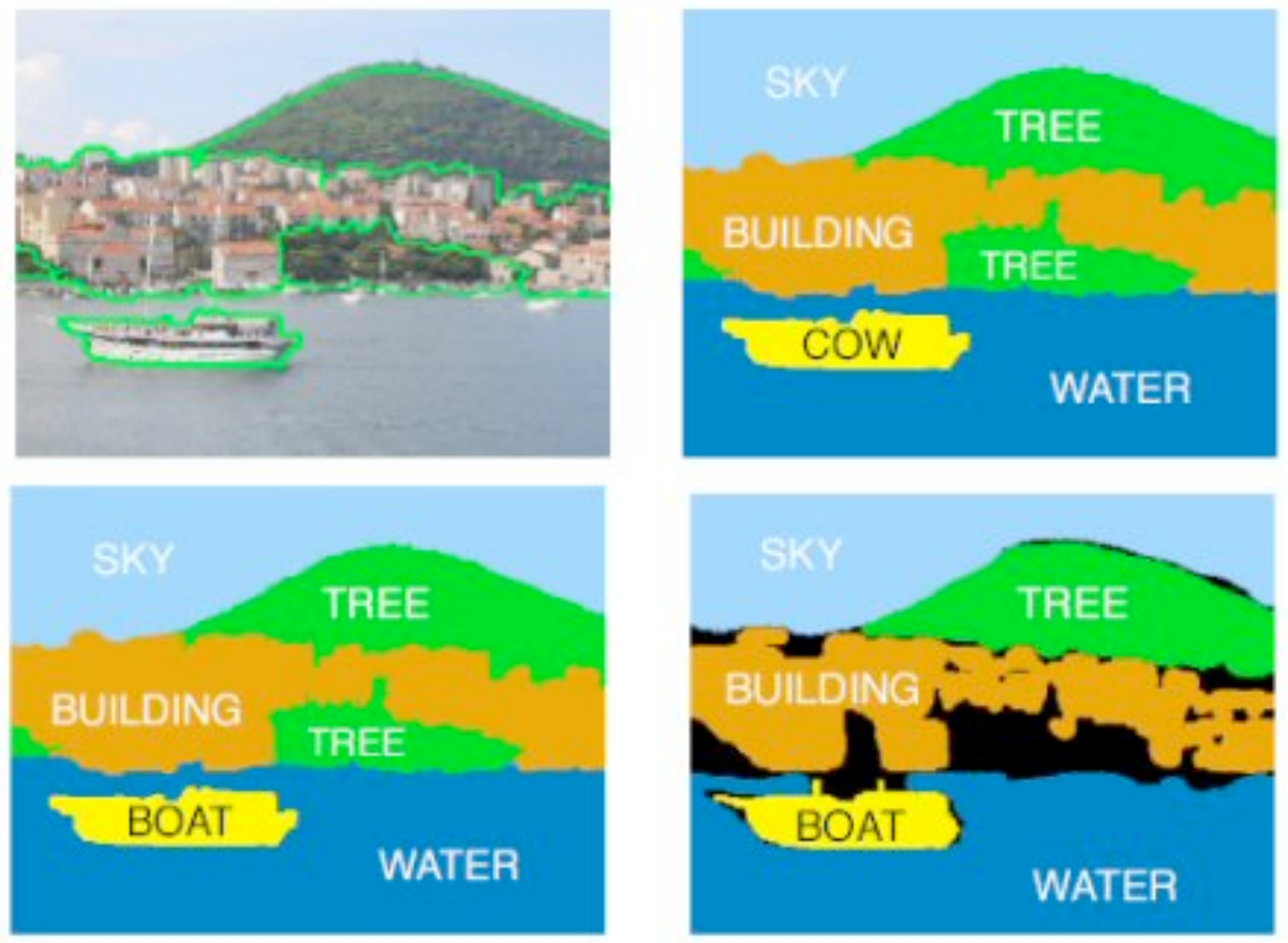
- Natural Language Processing

What is Machine Learning?

- Natural Language Processing
- Computer Vision

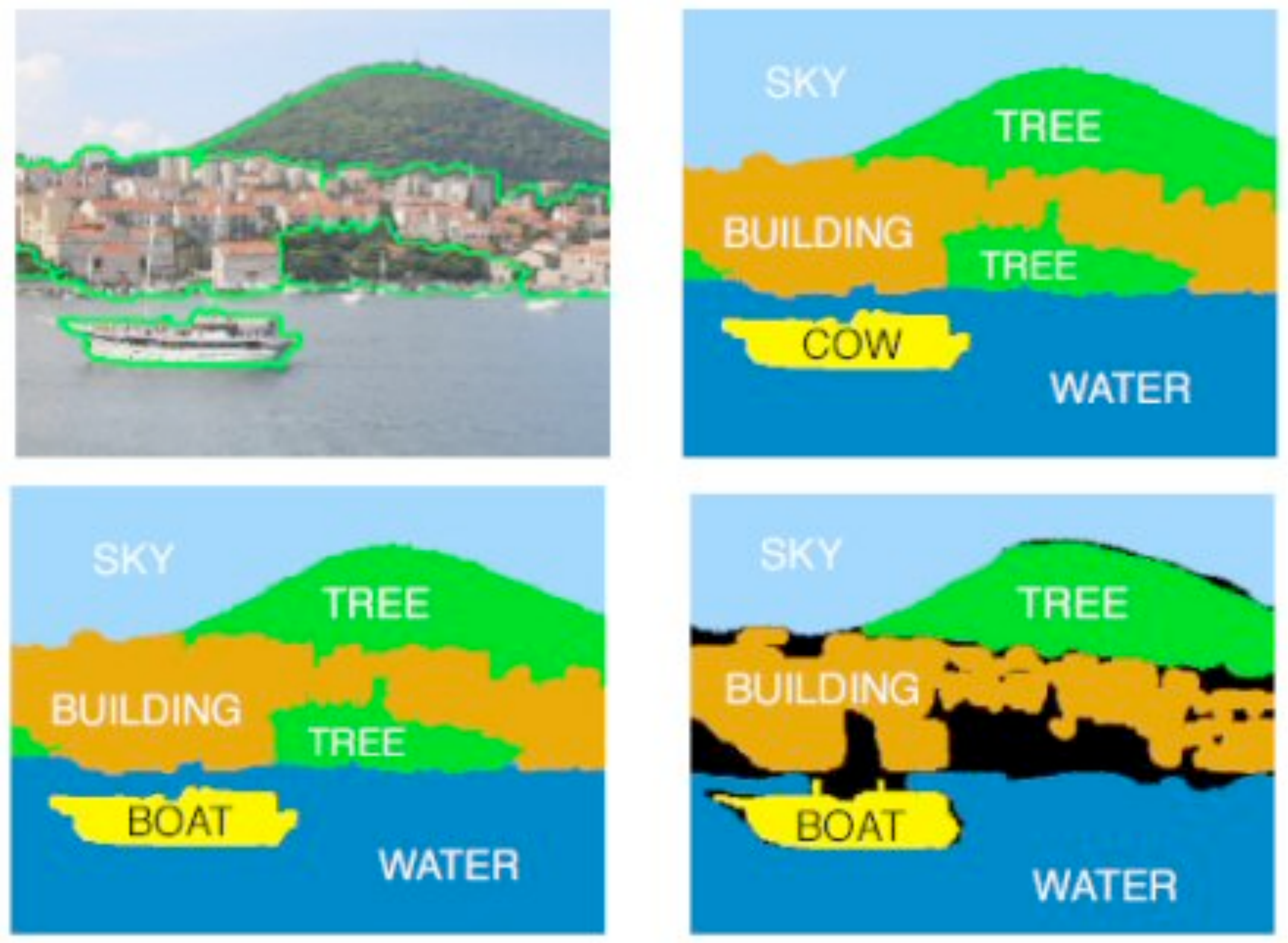
What is Machine Learning?

- Natural Language Processing
- Computer Vision



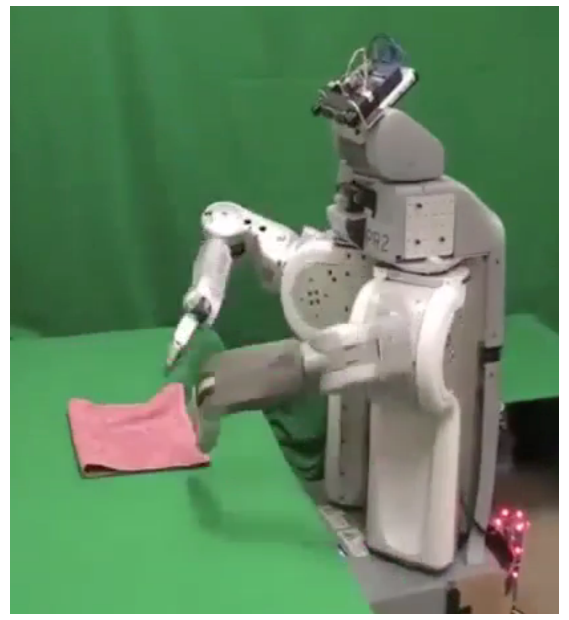
What is Machine Learning?

- Natural Language Processing
- Computer Vision
- Robotics



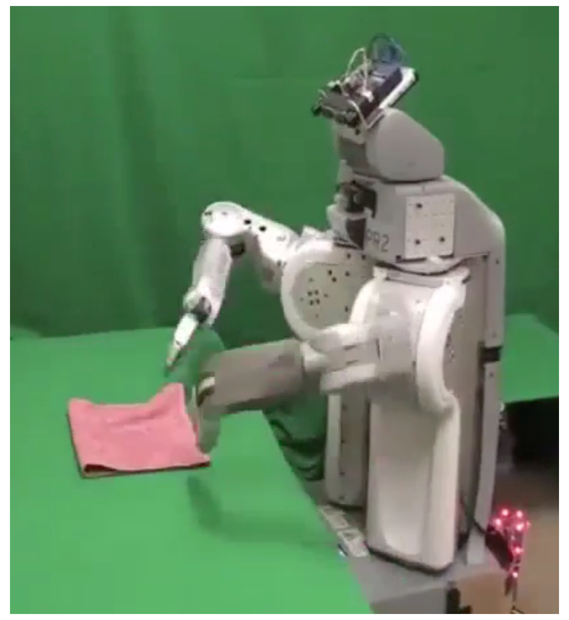
What is Machine Learning?

- Natural Language Processing
- Computer Vision
- Robotics



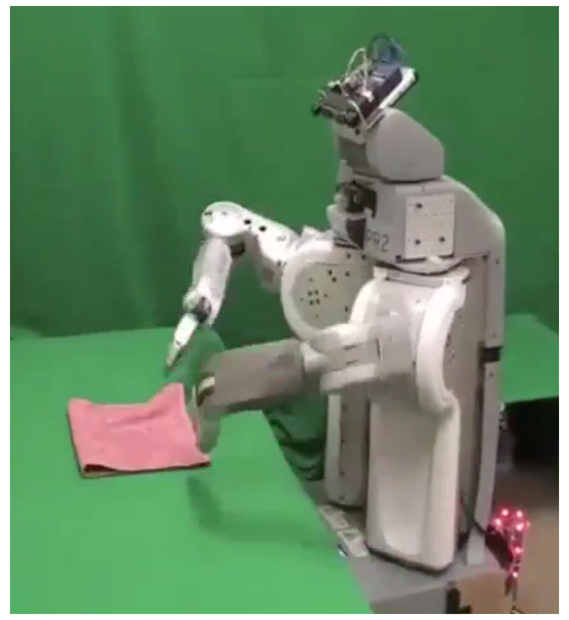
What is Machine Learning?

- Natural Language Processing
- Computer Vision
- Robotics



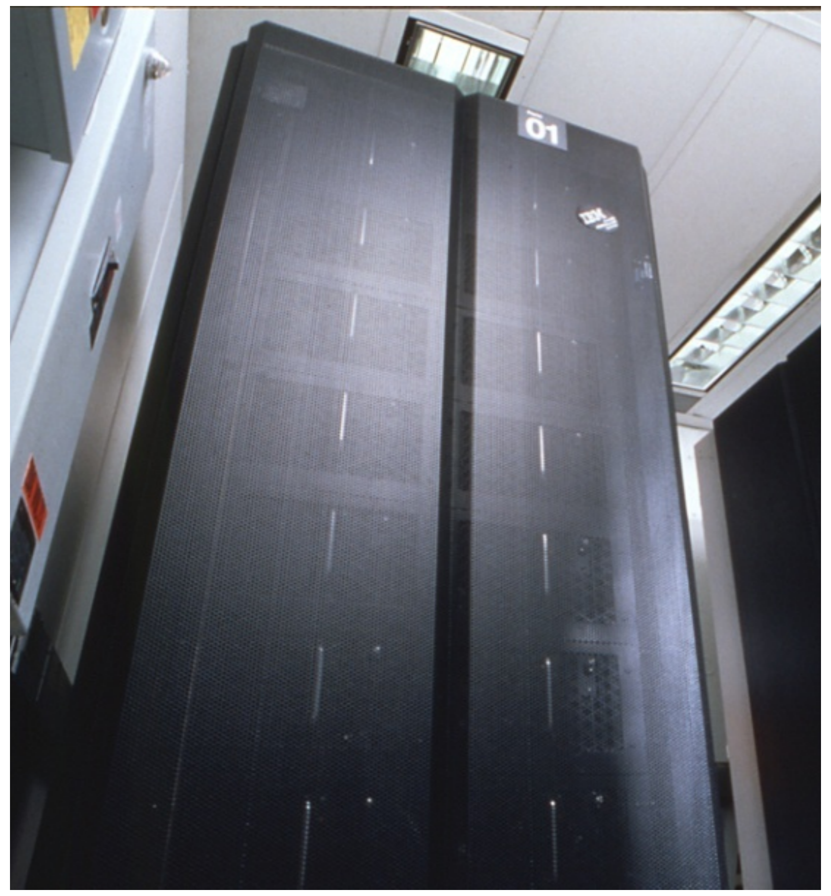

What is Machine Learning?

- Natural Language Processing
- Computer Vision
- Robotics
- Game Playing



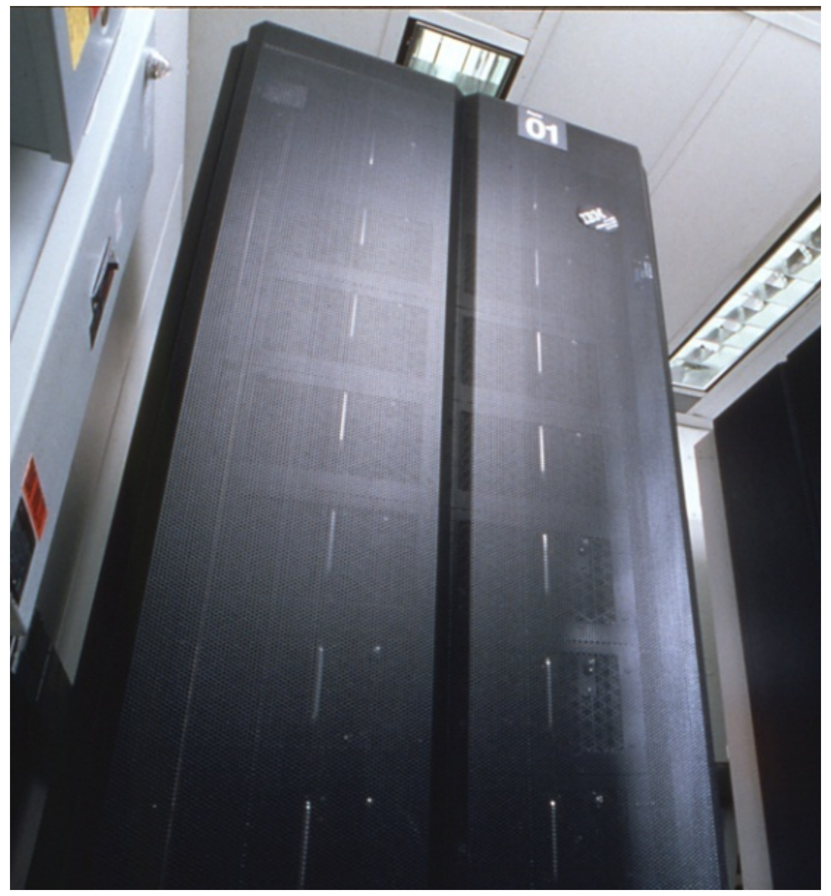
What is Machine Learning?

- Natural Language Processing
- Computer Vision
- Robotics
- Game Playing



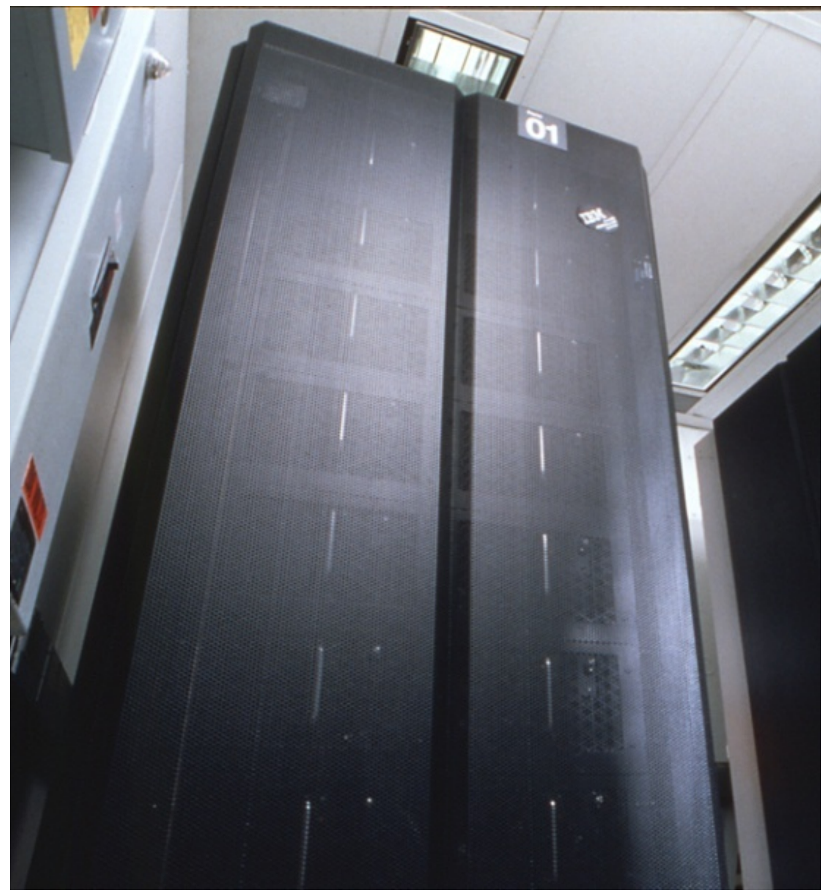
What is Machine Learning?

- Natural Language Processing
- Computer Vision
- Robotics
- Game Playing
- and much more!



What is Machine Learning?

- Natural Language Processing
- Computer Vision
- Robotics
- Game Playing
- and much more!



- What do these all have in common?

Machine Learning

Machine Learning

What is machine learning?

Machine Learning

What is machine learning?

- A subfield of computer science.

Machine Learning

What is machine learning?

- A subfield of computer science.
- The study of algorithms that *analyze data to make decisions*.

Machine Learning

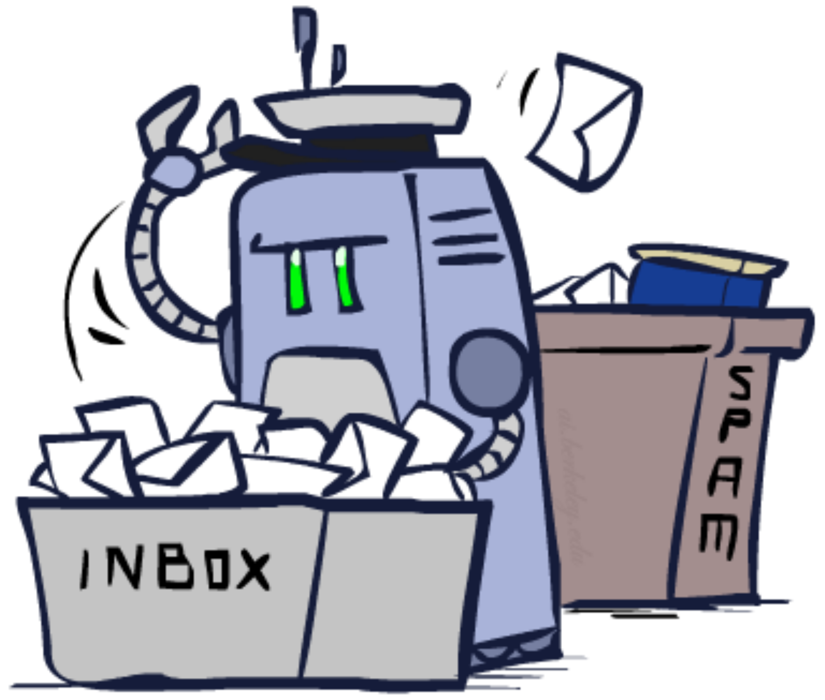
What is machine learning?

- A subfield of computer science.
- The study of algorithms that *analyze data to make decisions*.
- Examples of decisions:

Machine Learning

What is machine learning?

- A subfield of computer science.
- The study of algorithms that *analyze data to make decisions*.
- Examples of decisions:
    - Is this email ham or spam?



Machine Learning

What is machine learning?

- A subfield of computer science.
- The study of algorithms that *analyze data to make decisions*.
- Examples of decisions:
  - Is this email ham or spam?
  - How do I translate this sentence?

"Il est impossible aux journalistes de rentrer dans les régions tibétaines"

Bruno Philip, correspondant du "Monde" en Chine, estime que les journalistes de l'AFP qui ont été expulsés de la province tibétaine du Qinghai "n'étaient pas dans l'illégalité".

Les faits Le dalaï-lama dénonce l'"enfer" imposé au Tibet depuis sa fuite, en 1959

Vidéo Anniversaire de la rébellion

"It is impossible for journalists to enter Tibetan areas"

Philip Bruno, correspondent for "World" in China, said that journalists of the AFP who have been deported from the Tibetan province of Qinghai "were not illegal."

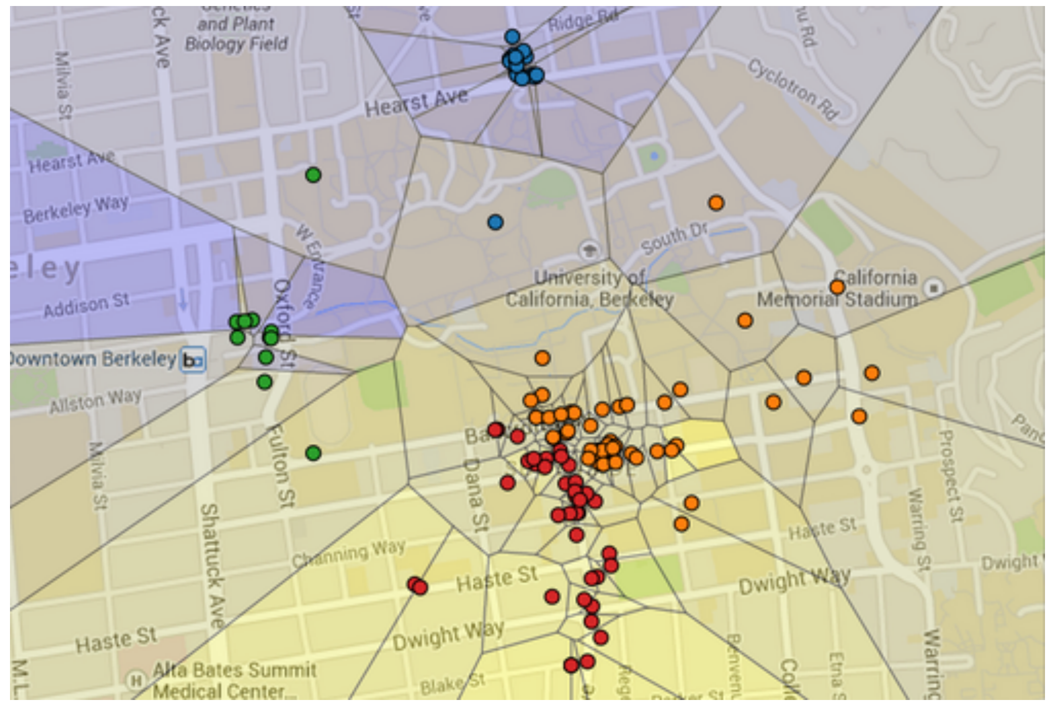
Facts The Dalai Lama denounces the "hell" imposed since he fled Tibet in 1959

Video Anniversary of the Tibetan rebellion: China on guard

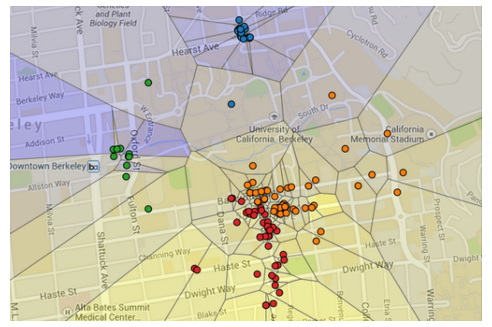
Machine Learning

What is machine learning?

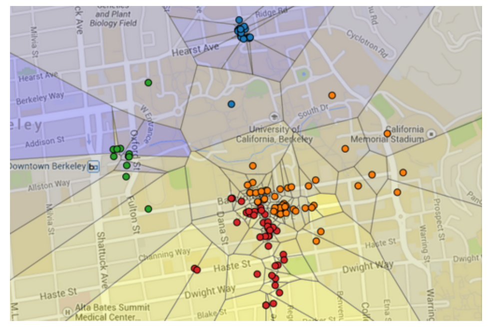
- A subfield of computer science.
- The study of algorithms that *analyze data to make decisions*.
- Examples of decisions:
  - Is this email ham or spam?
  - How do I translate this sentence?
  - Will this user like this restaurant?



Machine Learning Example: Maps

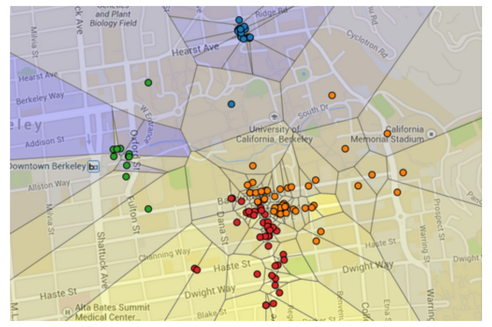


Machine Learning Example: Maps

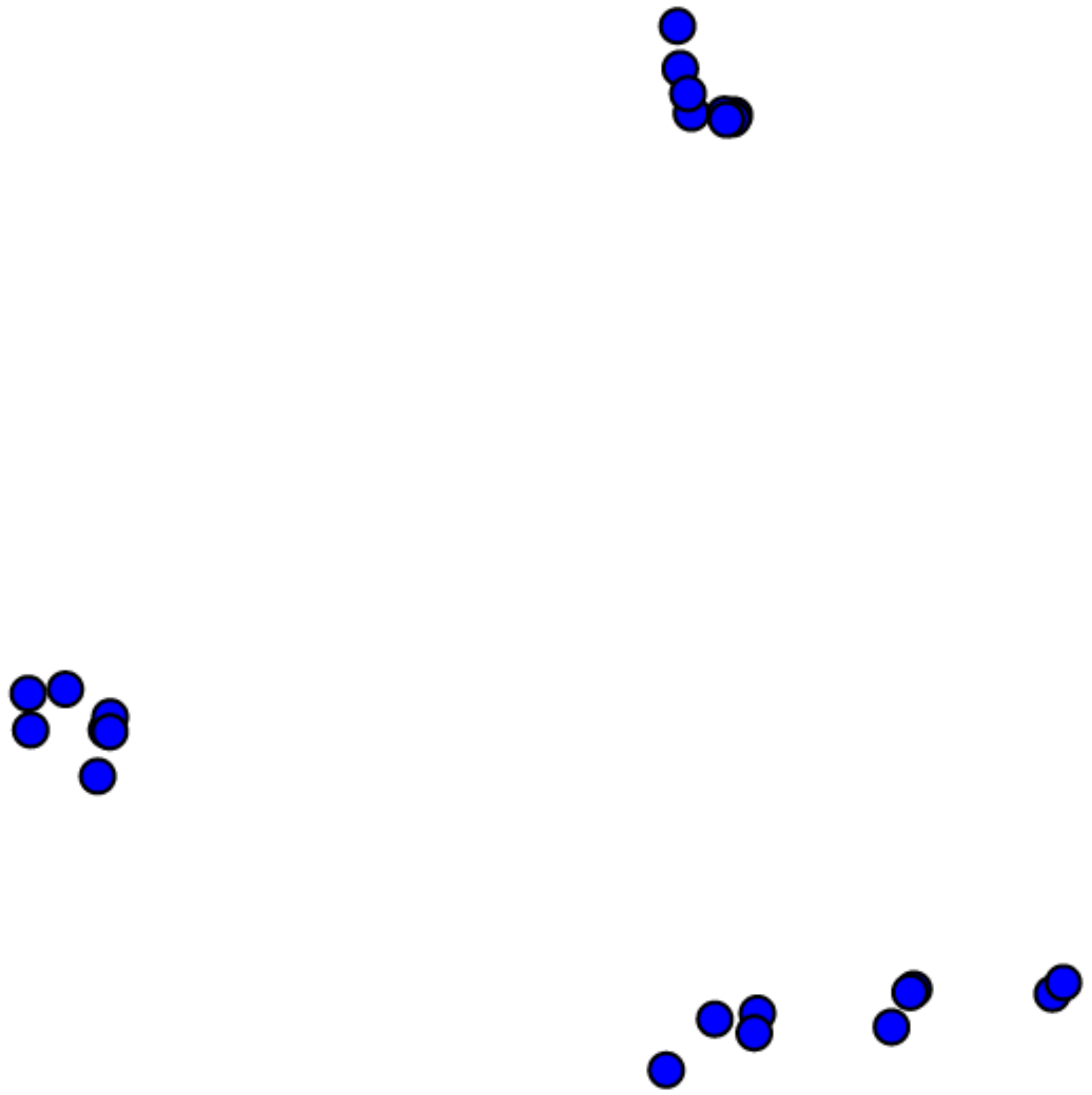


K-means Clustering

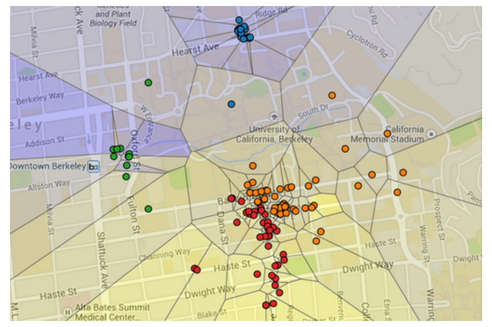
Machine Learning Example: Maps



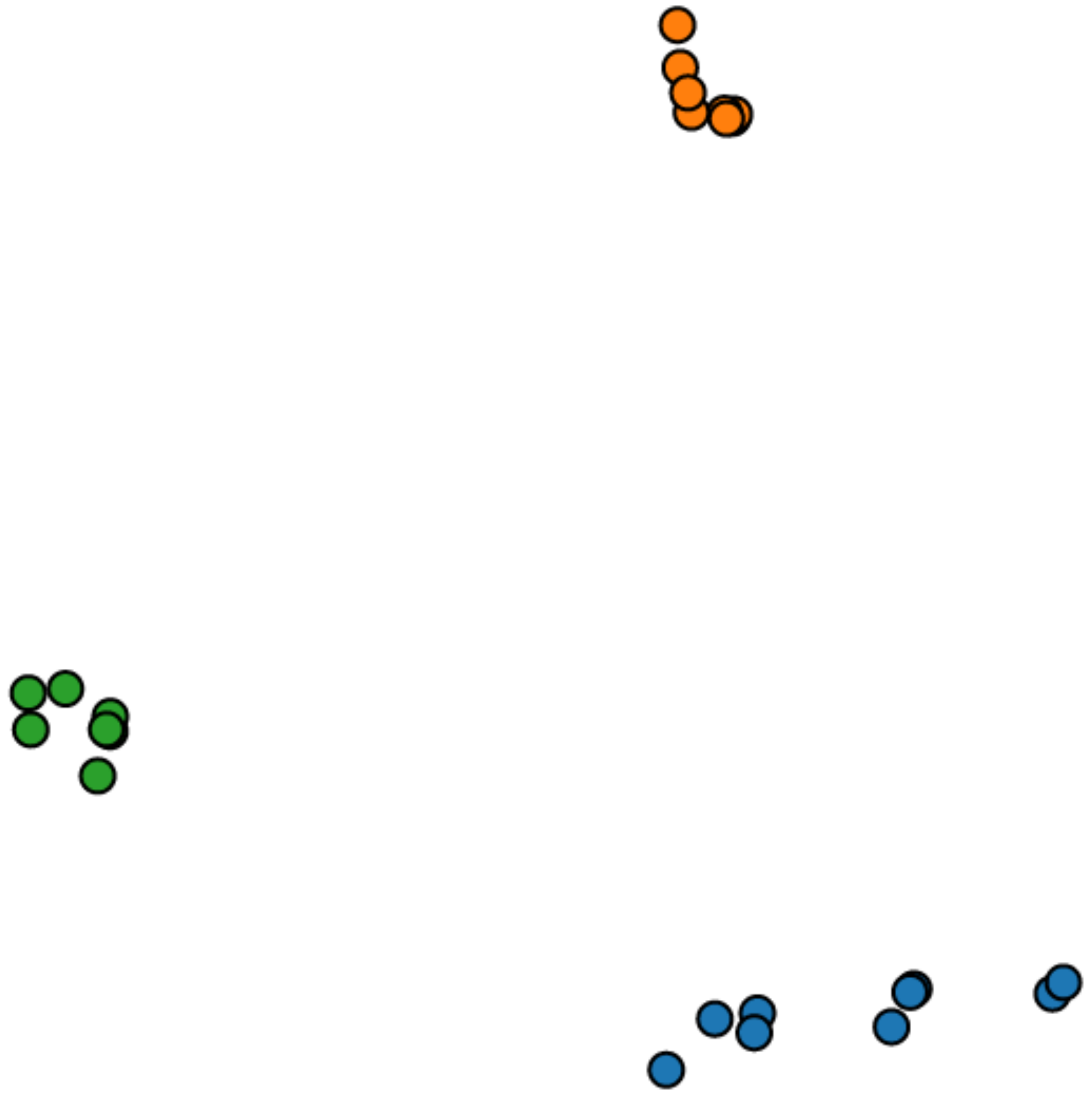
K-means Clustering



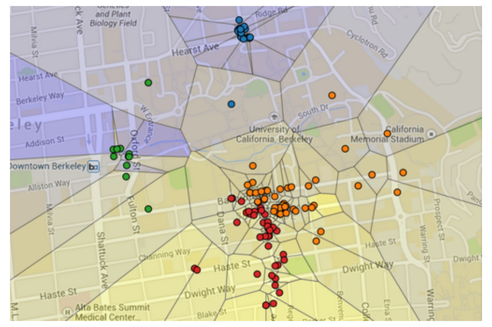
Machine Learning Example: Maps



K-means Clustering

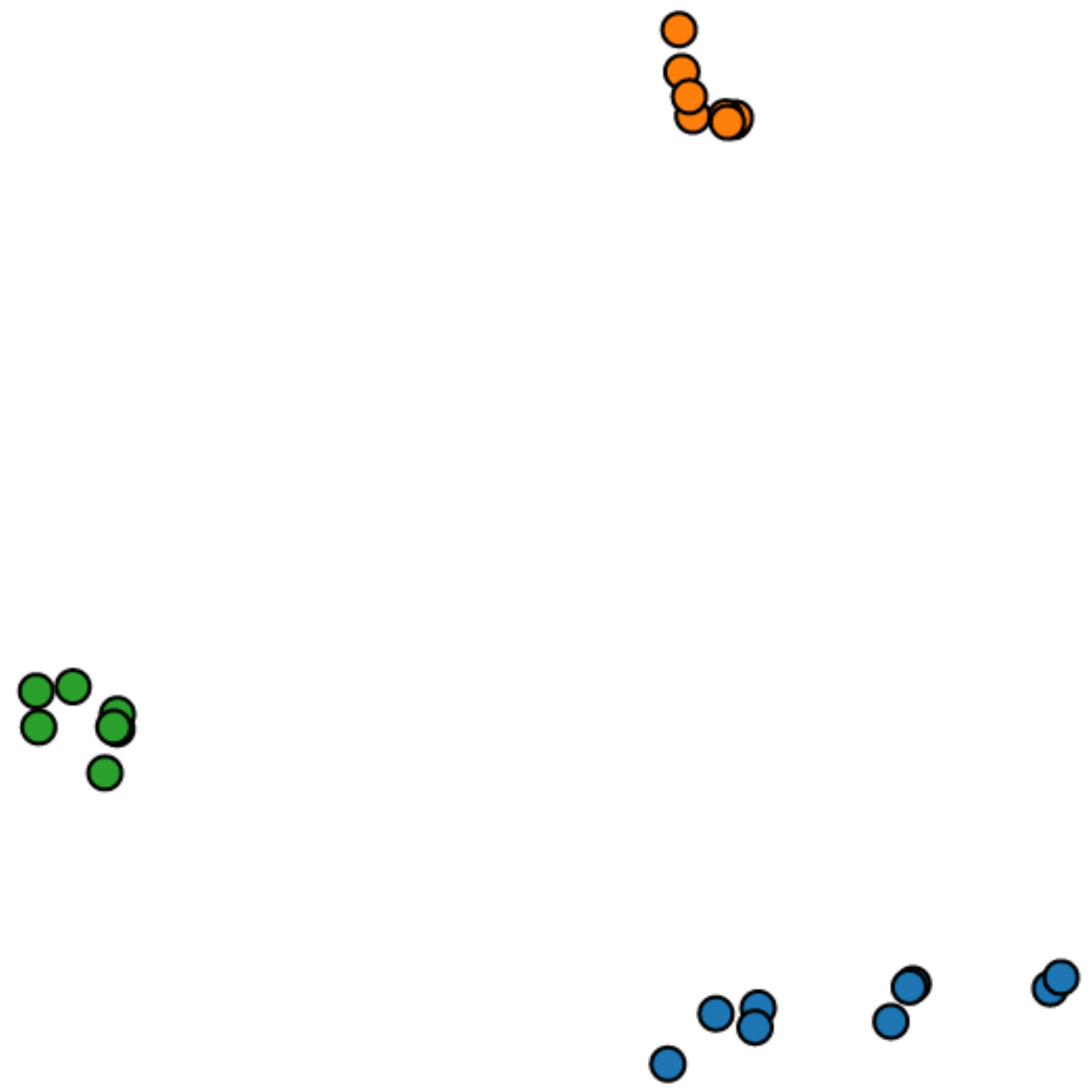


Machine Learning Example: Maps

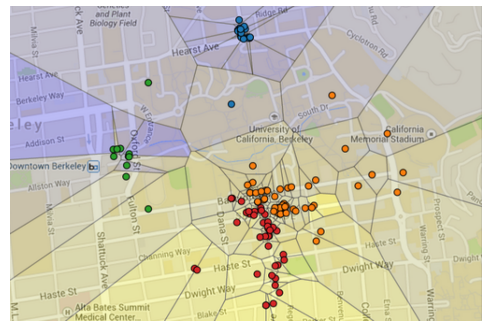


K-means Clustering

- The data: restaurant locations

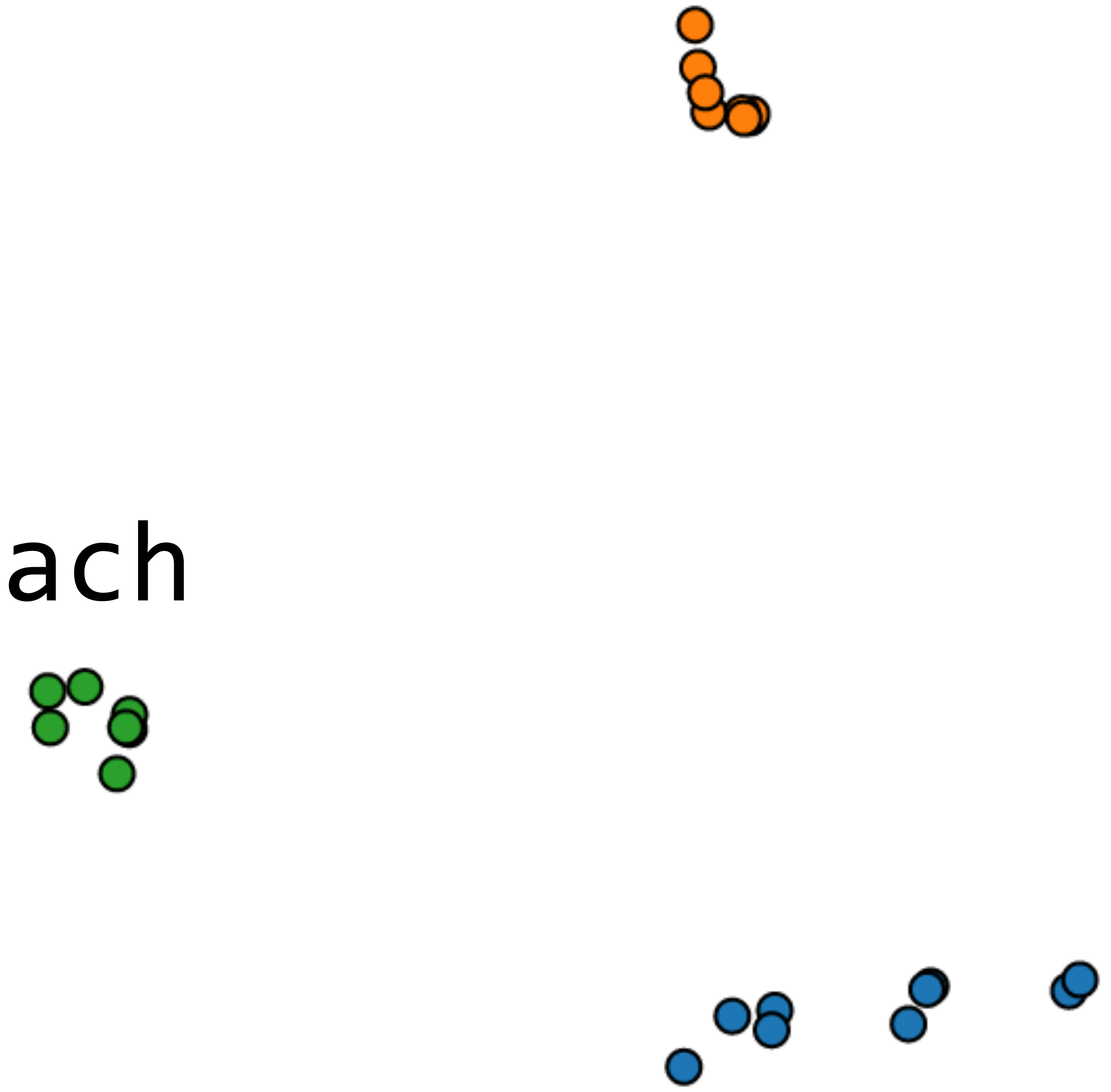


Machine Learning Example: Maps

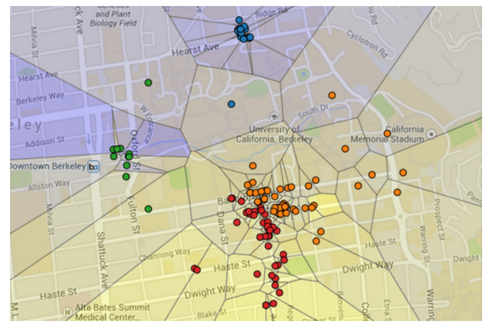


K-means Clustering

- The data: restaurant locations
- The decision: which cluster does each belong to?



Machine Learning Example: Maps

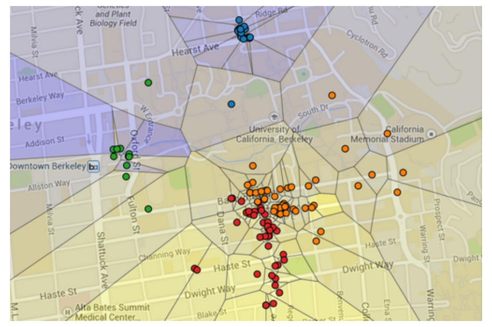


K-means Clustering

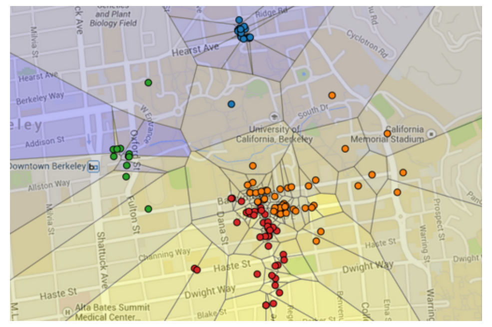
- The data: restaurant locations
- The decision: which cluster does each belong to?

Called *unsupervised learning*, because no one tells it what the correct decision is.

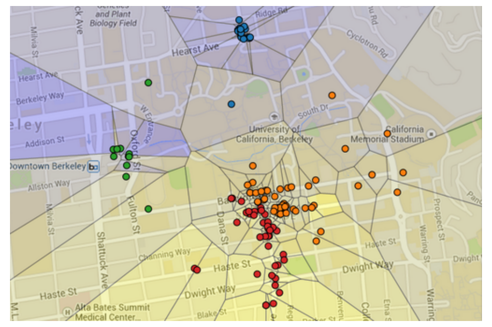
Machine Learning Example: Maps



Machine Learning Example: Maps

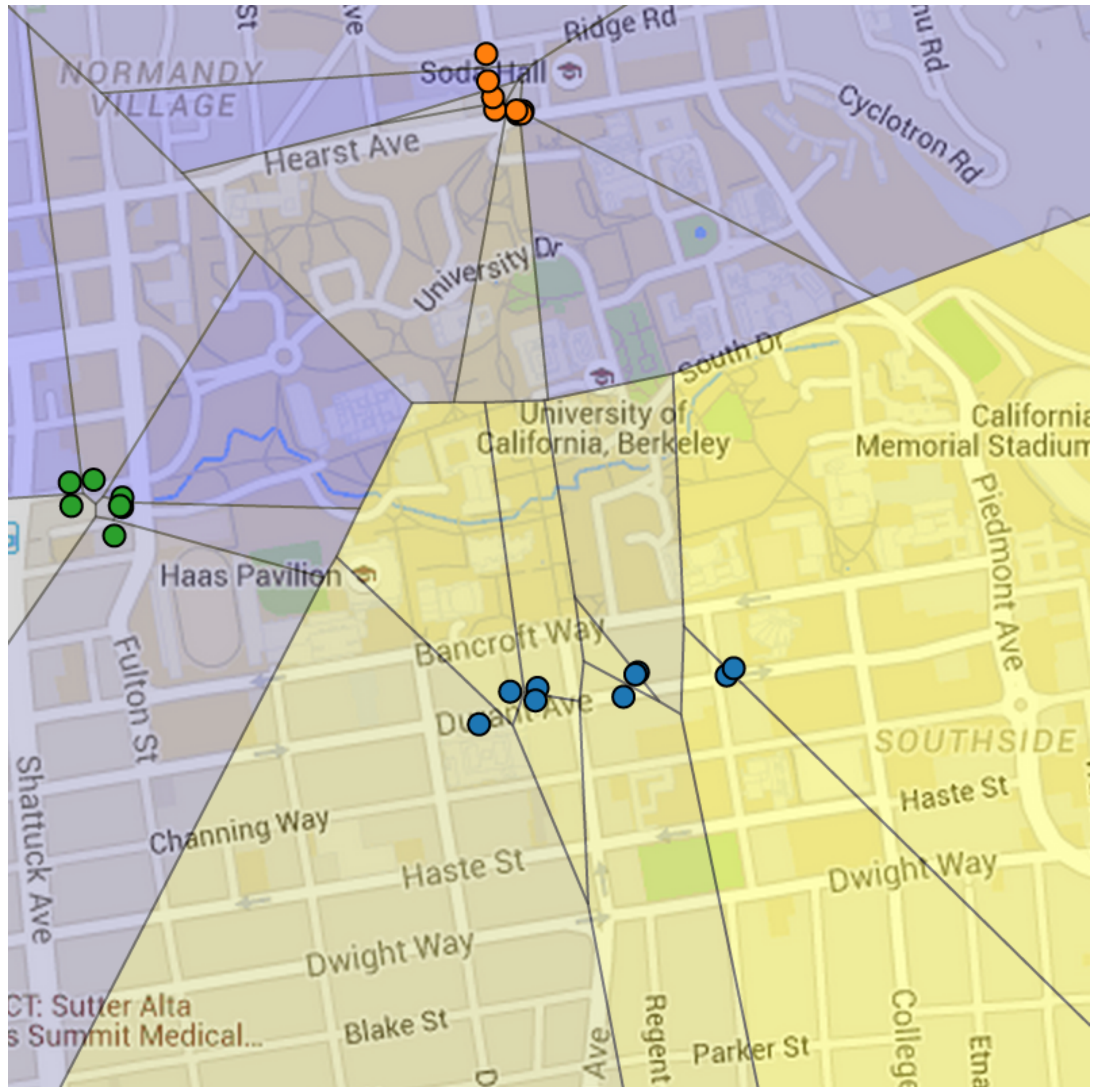


Linear Regression

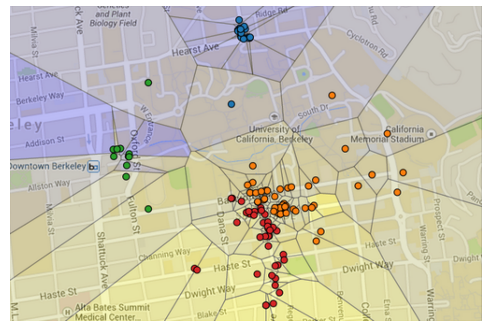


Machine Learning Example: Maps

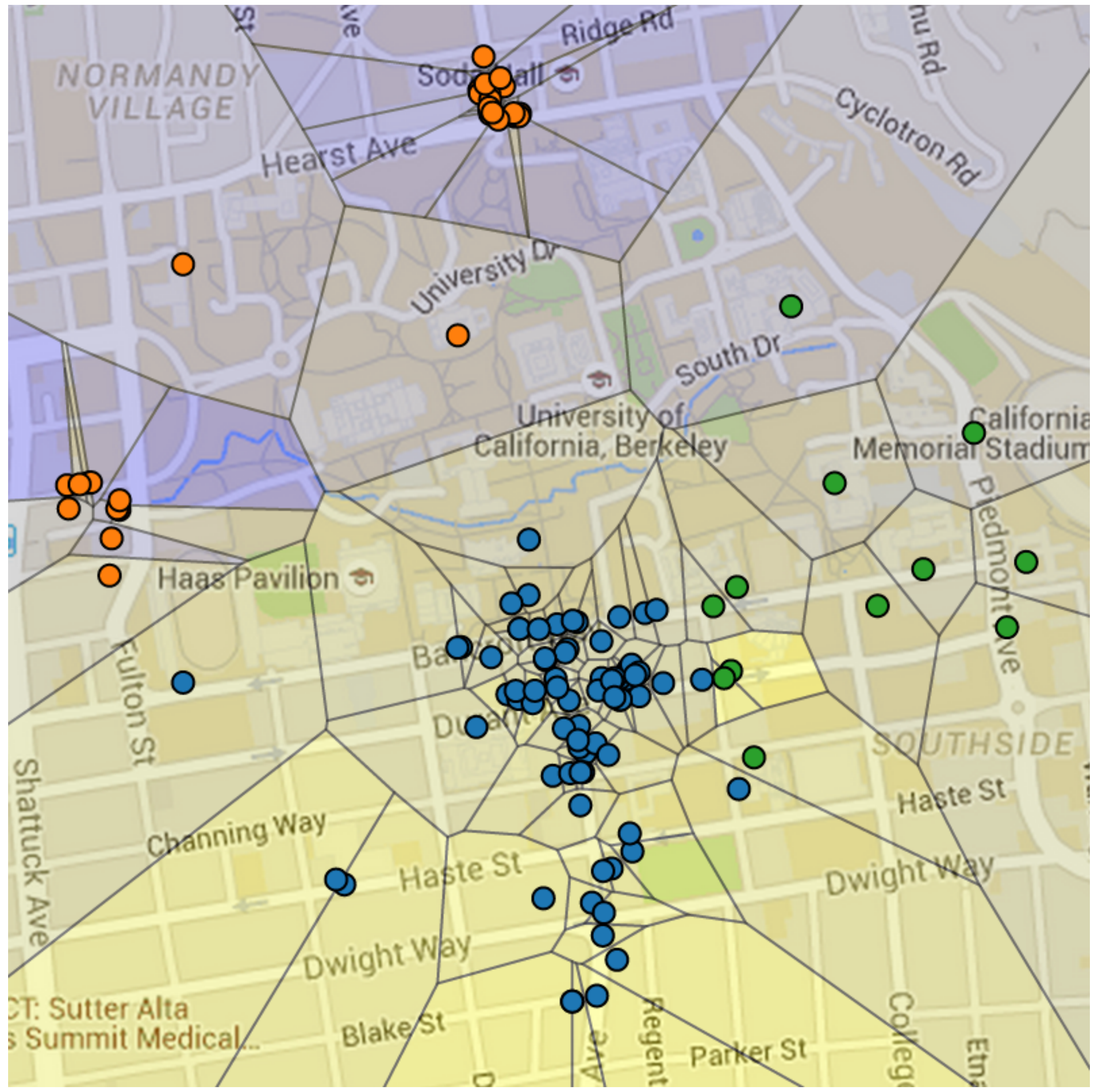
Linear Regression



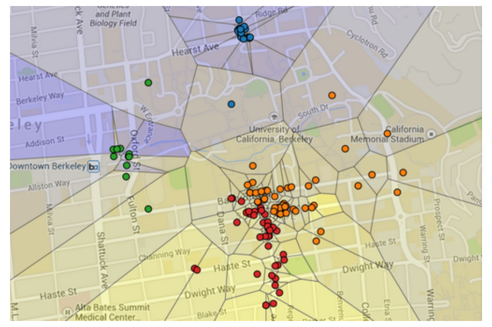
Machine Learning Example: Maps



Linear Regression

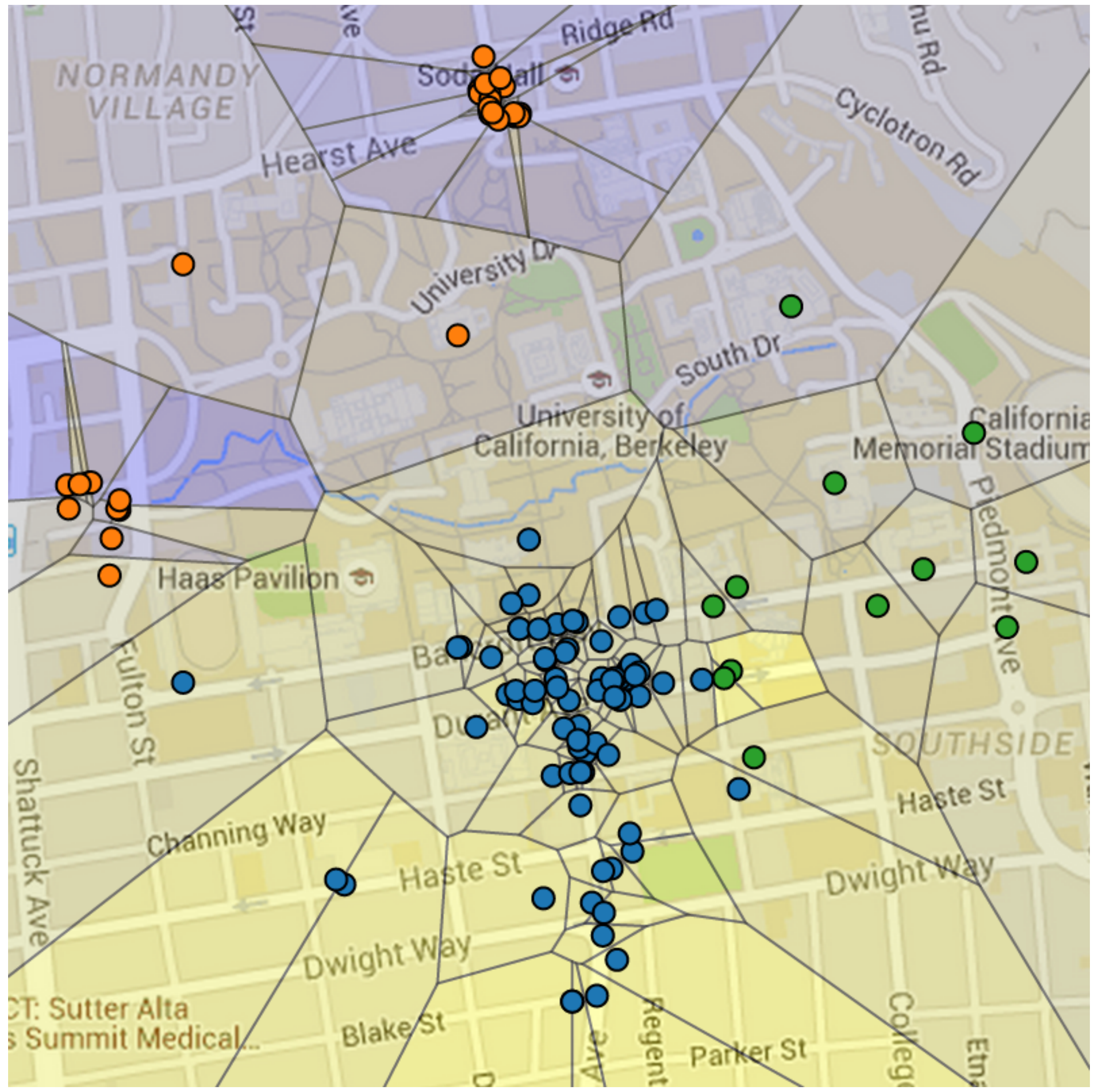


Machine Learning Example: Maps

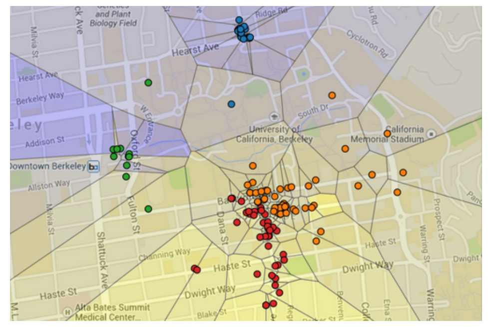


Linear Regression

- The data: user ratings

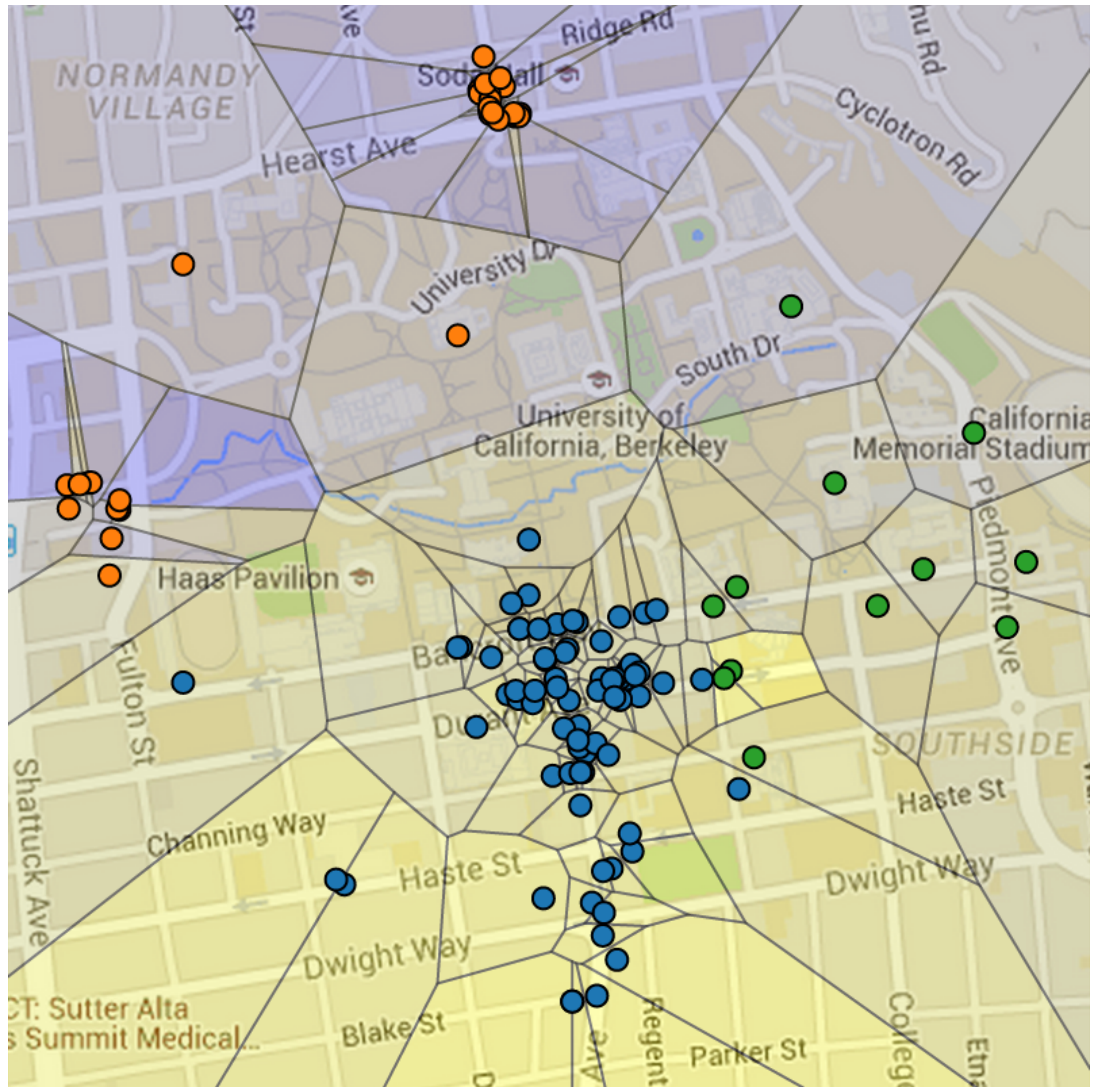


Machine Learning Example: Maps

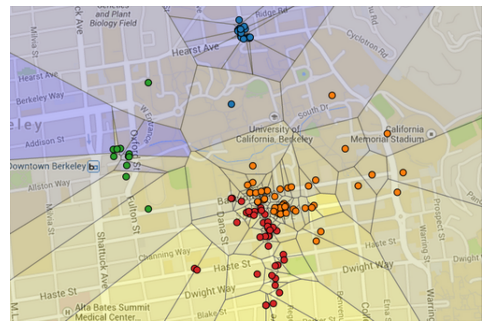


Linear Regression

- The data: user ratings
- The decision: what rating would the user give a new restaurant?



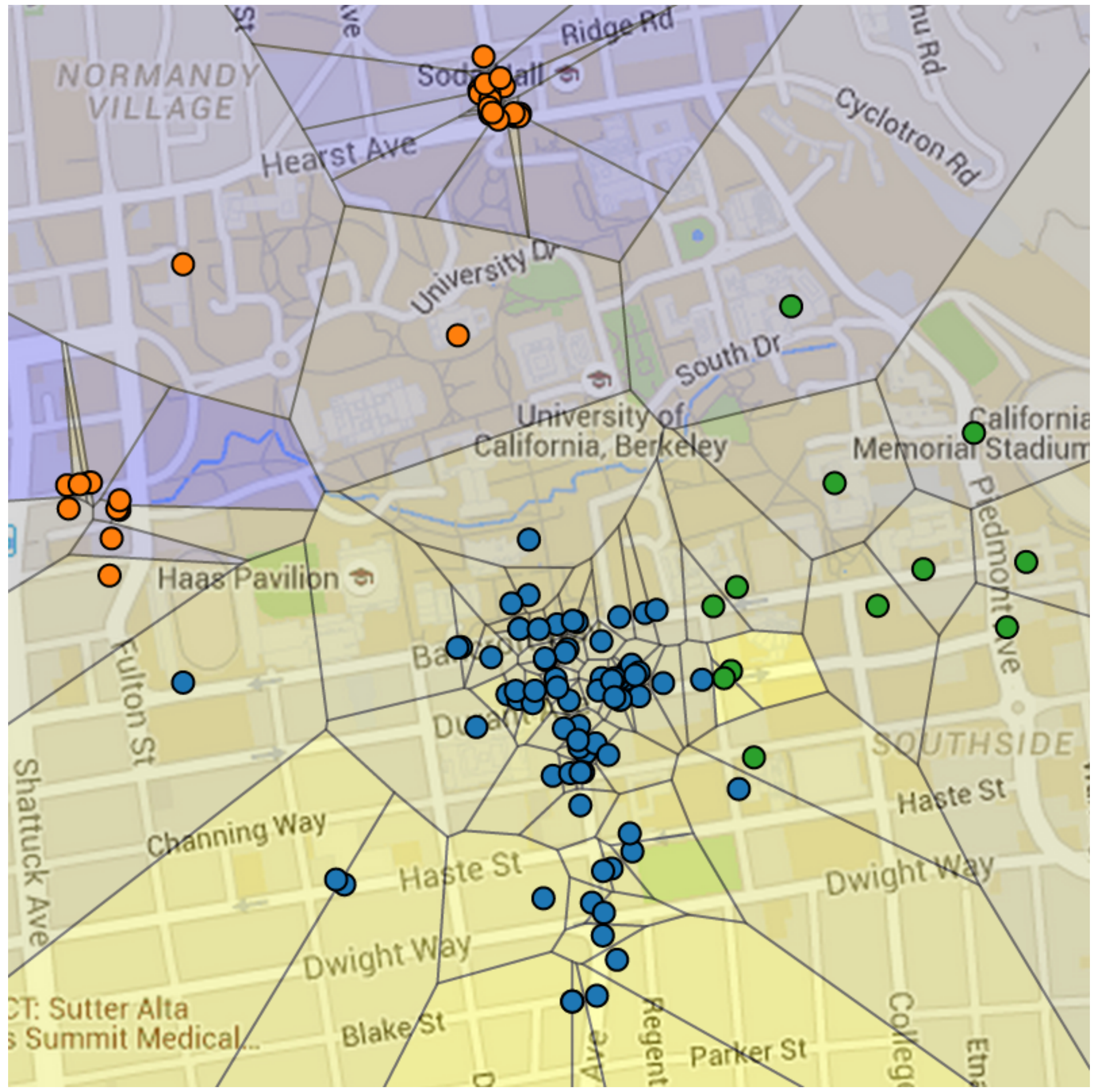
Machine Learning Example: Maps



Linear Regression

- The data: user ratings
- The decision: what rating would the user give a new restaurant?

Called *supervised learning*, because some correct decisions are given.



Outline

Outline

- So far, we’ve looked at two specific machine learning algorithms from two different domains.

Outline

- So far, we’ve looked at two specific machine learning algorithms from two different domains.

- Today, we will focus on a subclass of problems in machine learning, known as *reinforcement learning* problems, and algorithms for these problems.

Reinforcement Learning

Reinforcement Learning

What is reinforcement learning?

Reinforcement Learning

What is reinforcement learning?

- Concerned with *learning behavior through experience*.

Reinforcement Learning

What is reinforcement learning?

- Concerned with *learning behavior through experience*.
- Two main components: the *agent* and the *environment*.

Reinforcement Learning

What is reinforcement learning?

- Concerned with *learning behavior through experience*.
- Two main components: the *agent* and the *environment*.
- The agent lives in and interacts with the environment, and through this experience learns a good pattern of behavior.

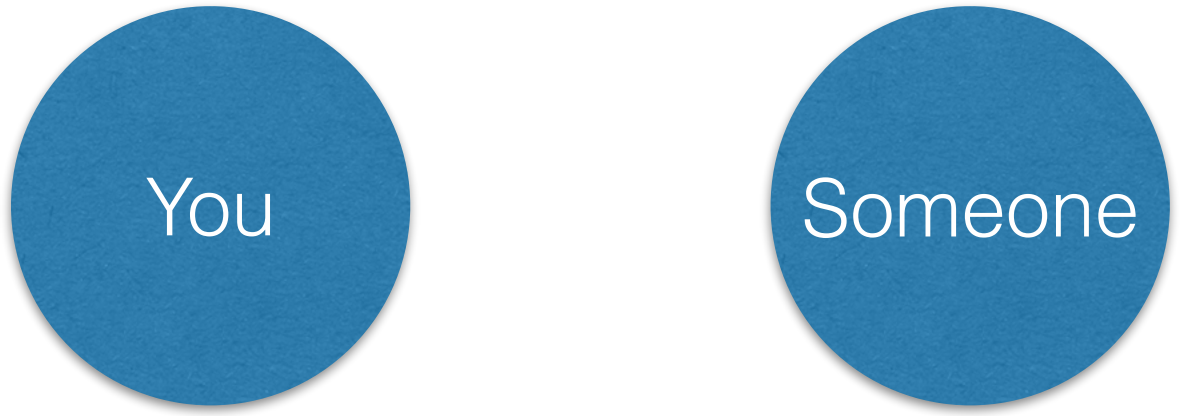
An Analogy

An Analogy

Suppose you go on a date with someone.

An Analogy

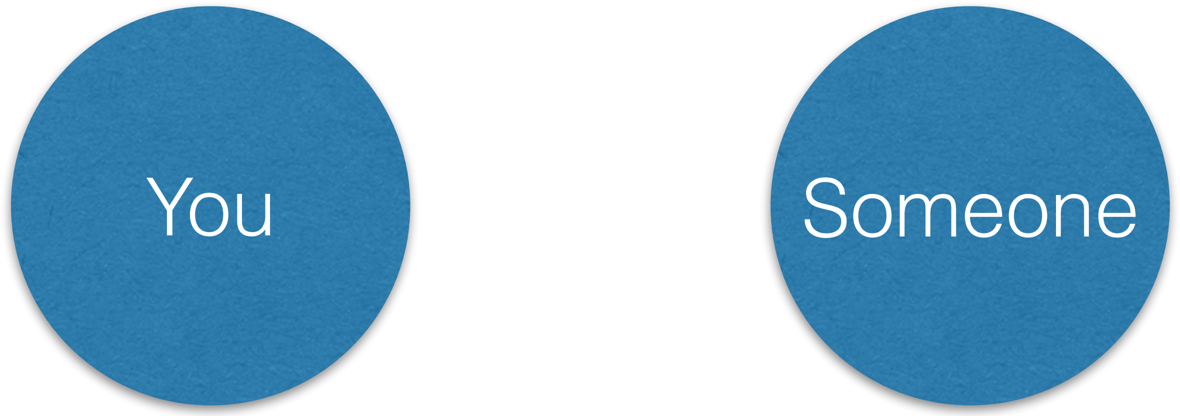
Suppose you go on a date with someone.



An Analogy

Suppose you go on a date with someone.

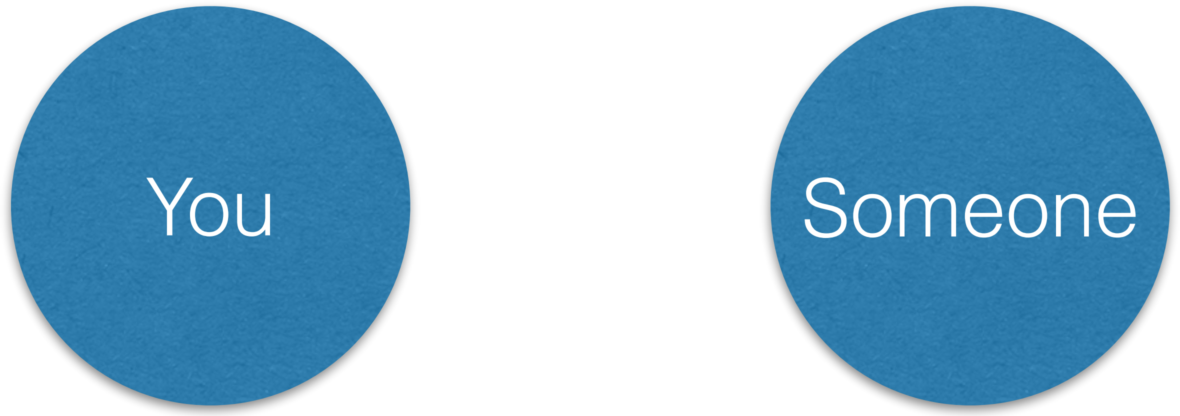
- In reinforcement learning terms, you are the *agent*.



An Analogy

Suppose you go on a date with someone.

- In reinforcement learning terms, you are the *agent*.
- Everything else (the other person, the setting, etc.) is the *environment*.



An Analogy

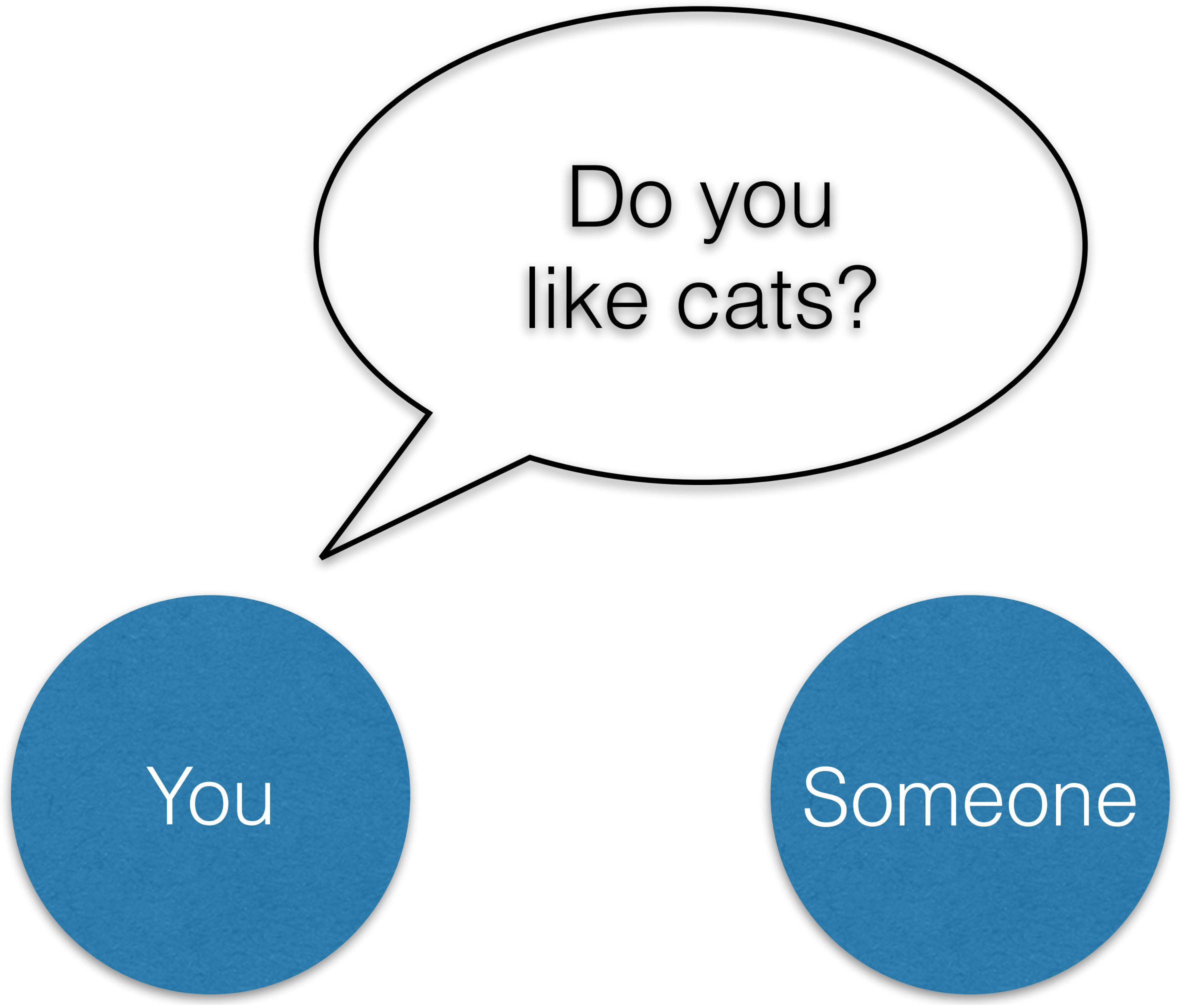
Suppose you go on a date with someone.

- In reinforcement learning terms, you are the *agent*.
- Everything else (the other person, the setting, etc.) is the *environment*.
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds.

An Analogy

Suppose you go on a date with someone.

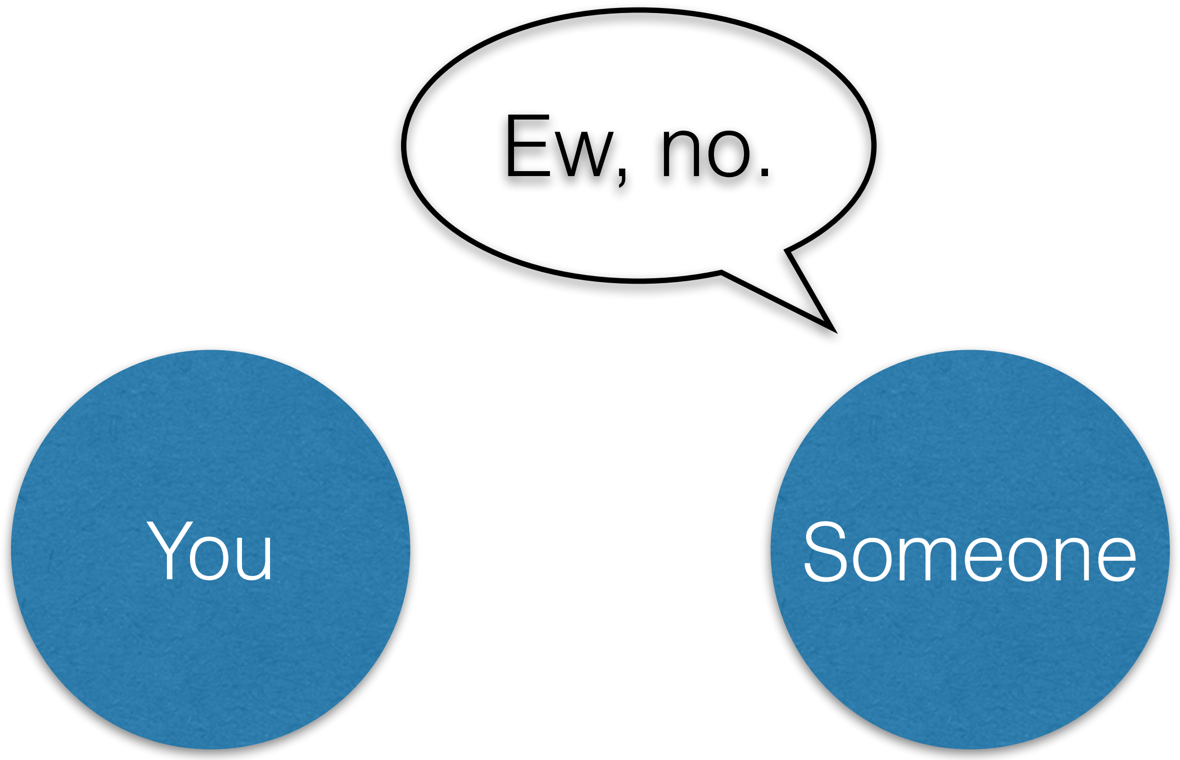
- In reinforcement learning terms, you are the *agent*.
- Everything else (the other person, the setting, etc.) is the *environment*.
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds.



An Analogy

Suppose you go on a date with someone.

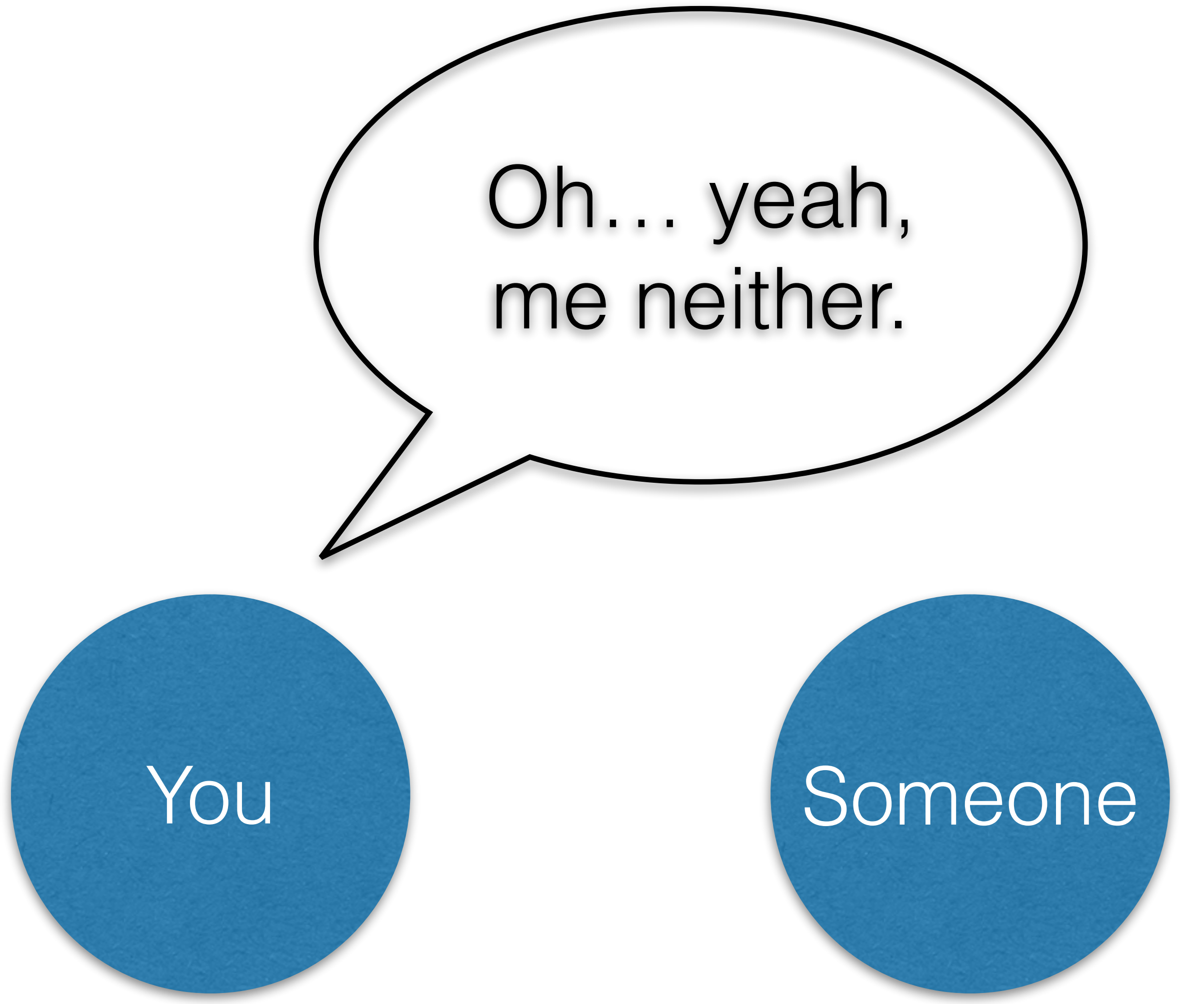
- In reinforcement learning terms, you are the *agent*.
- Everything else (the other person, the setting, etc.) is the *environment*.
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds.



An Analogy

Suppose you go on a date with someone.

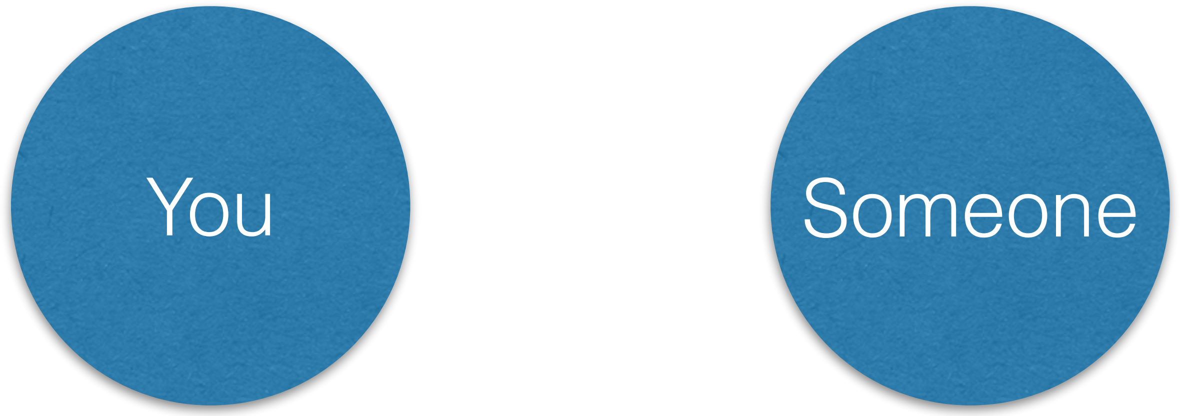
- In reinforcement learning terms, you are the *agent*.
- Everything else (the other person, the setting, etc.) is the *environment*.
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds.



An Analogy

Suppose you go on a date with someone.

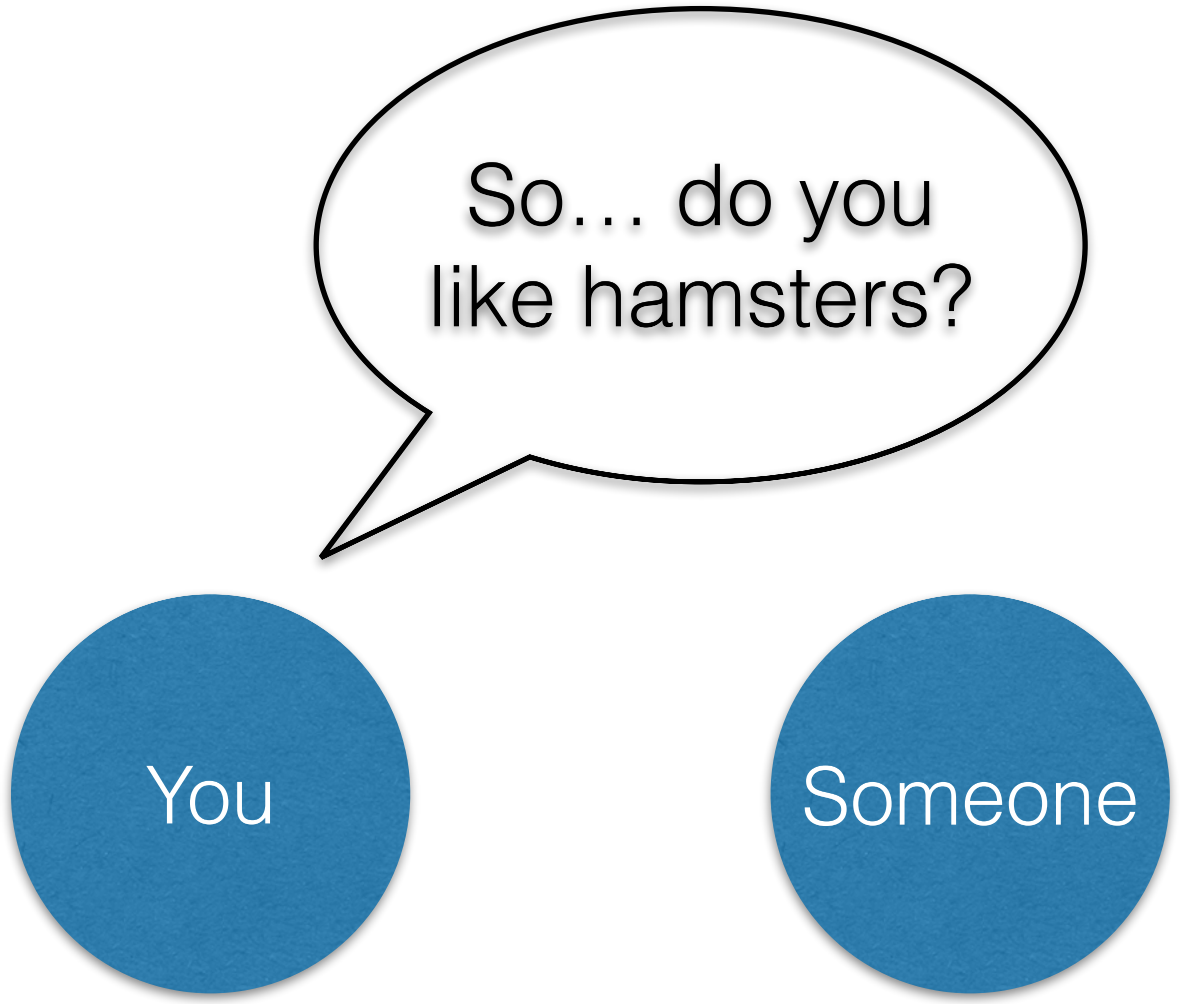
- In reinforcement learning terms, you are the *agent*.
- Everything else (the other person, the setting, etc.) is the *environment*.
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds.
- As the date goes on, you slowly figure out how you should behave based on what you’ve tried so far, and how it went.



An Analogy

Suppose you go on a date with someone.

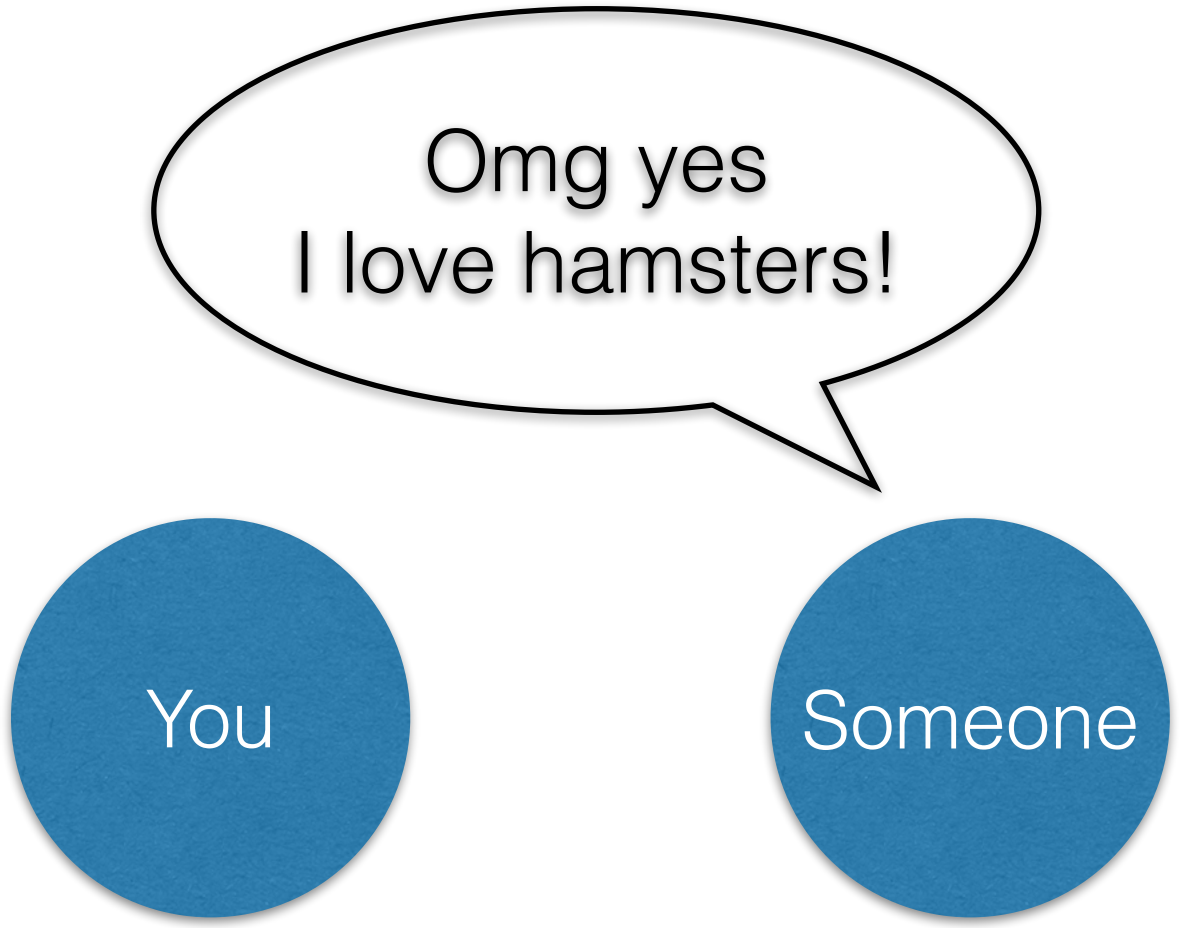
- In reinforcement learning terms, you are the *agent*.
- Everything else (the other person, the setting, etc.) is the *environment*.
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds.
- As the date goes on, you slowly figure out how you should behave based on what you've tried so far, and how it went.



An Analogy

Suppose you go on a date with someone.

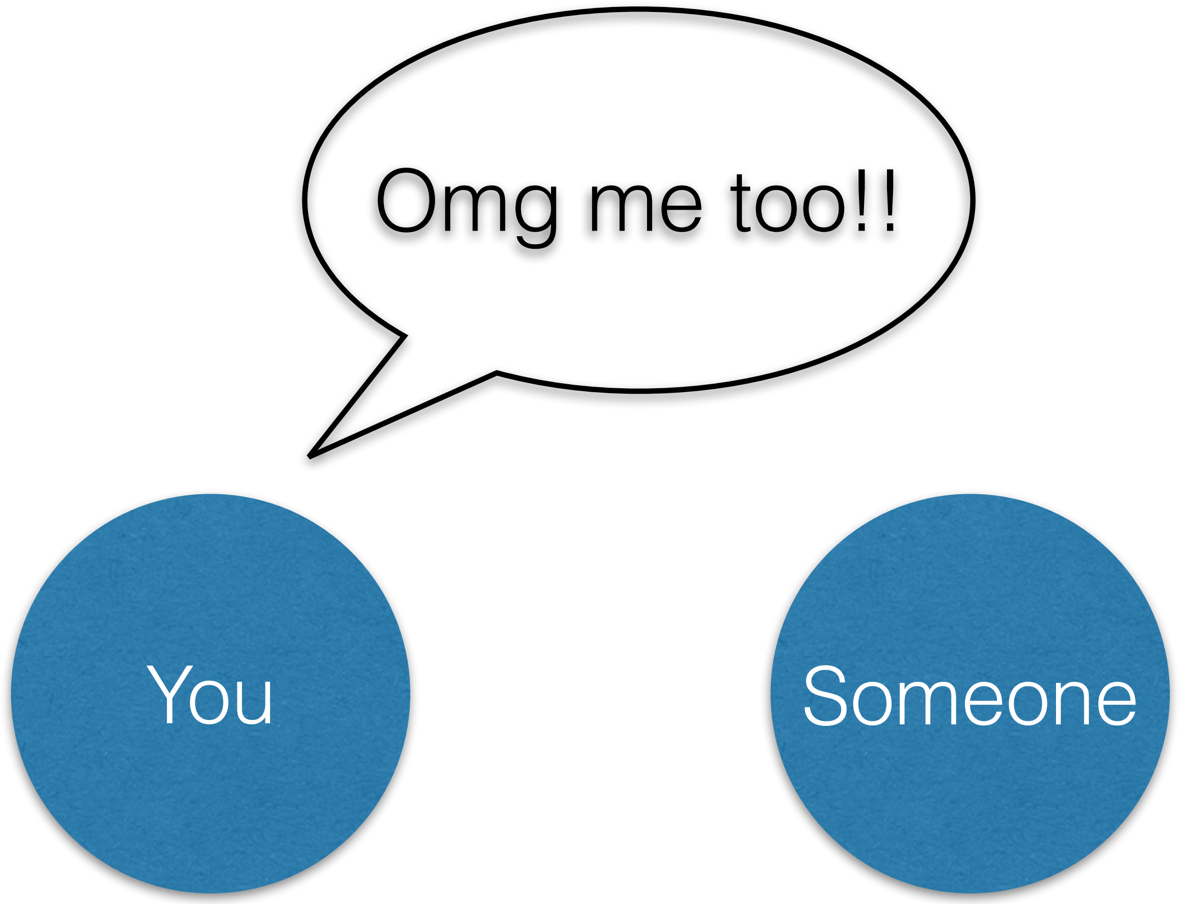
- In reinforcement learning terms, you are the *agent*.
- Everything else (the other person, the setting, etc.) is the *environment*.
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds.
- As the date goes on, you slowly figure out how you should behave based on what you've tried so far, and how it went.



An Analogy

Suppose you go on a date with someone.

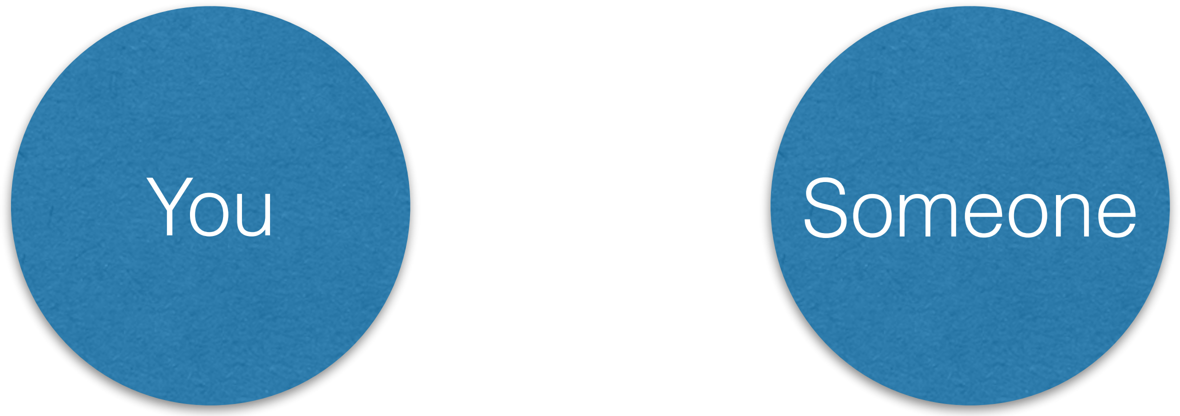
- In reinforcement learning terms, you are the *agent*.
- Everything else (the other person, the setting, etc.) is the *environment*.
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds.
- As the date goes on, you slowly figure out how you should behave based on what you've tried so far, and how it went.



An Analogy

Suppose you go on a date with someone.

- In reinforcement learning terms, you are the *agent*.
- Everything else (the other person, the setting, etc.) is the *environment*.
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds.
- As the date goes on, you slowly figure out how you should behave based on what you've tried so far, and how it went.
- If you're a good agent, you may even learn how to behave really, really well!



An Analogy

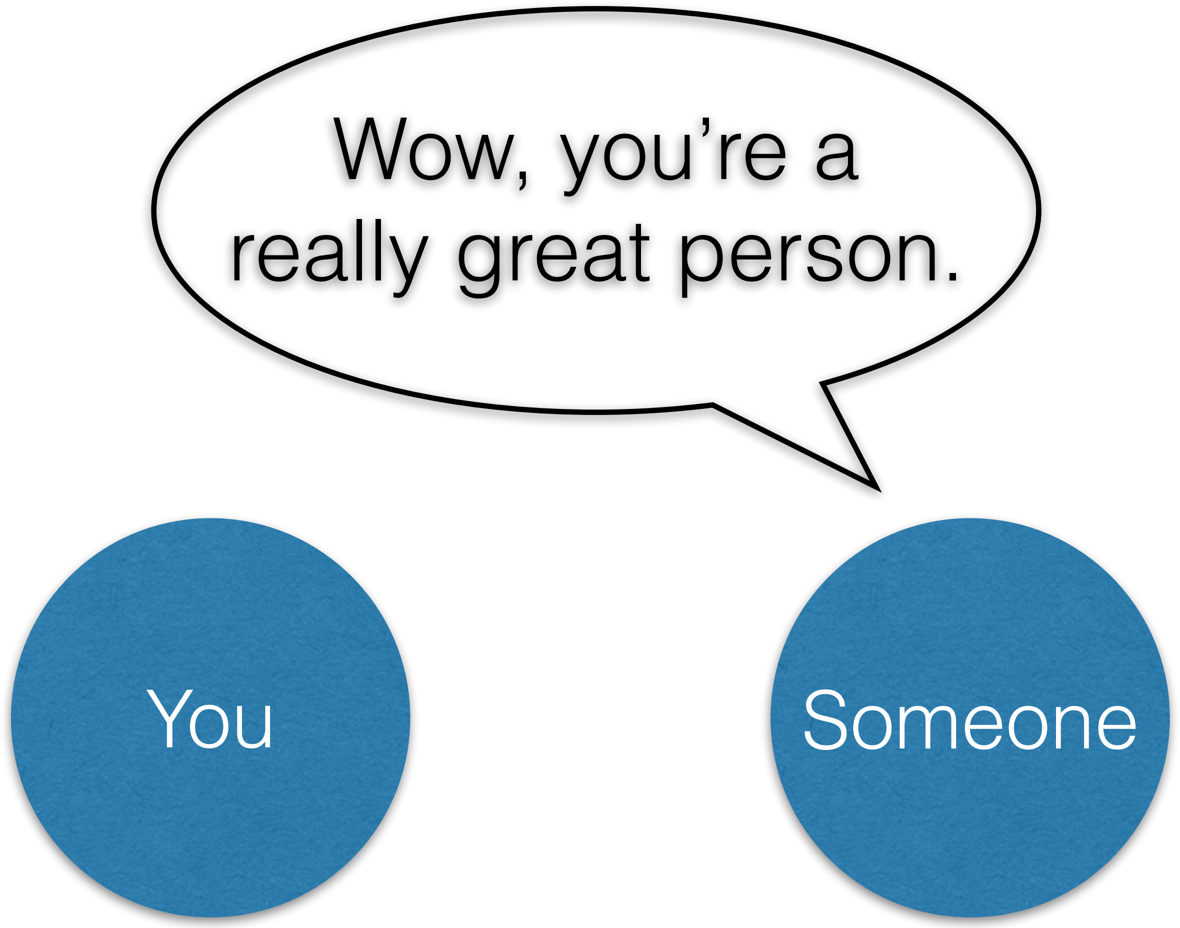
Suppose you go on a date with someone.

- In reinforcement learning terms, you are the *agent*.
- Everything else (the other person, the setting, etc.) is the *environment*.
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds.
- As the date goes on, you slowly figure out how you should behave based on what you’ve tried so far, and how it went.
- If you’re a good agent, you may even learn how to behave really, really well!

An Analogy

Suppose you go on a date with someone.

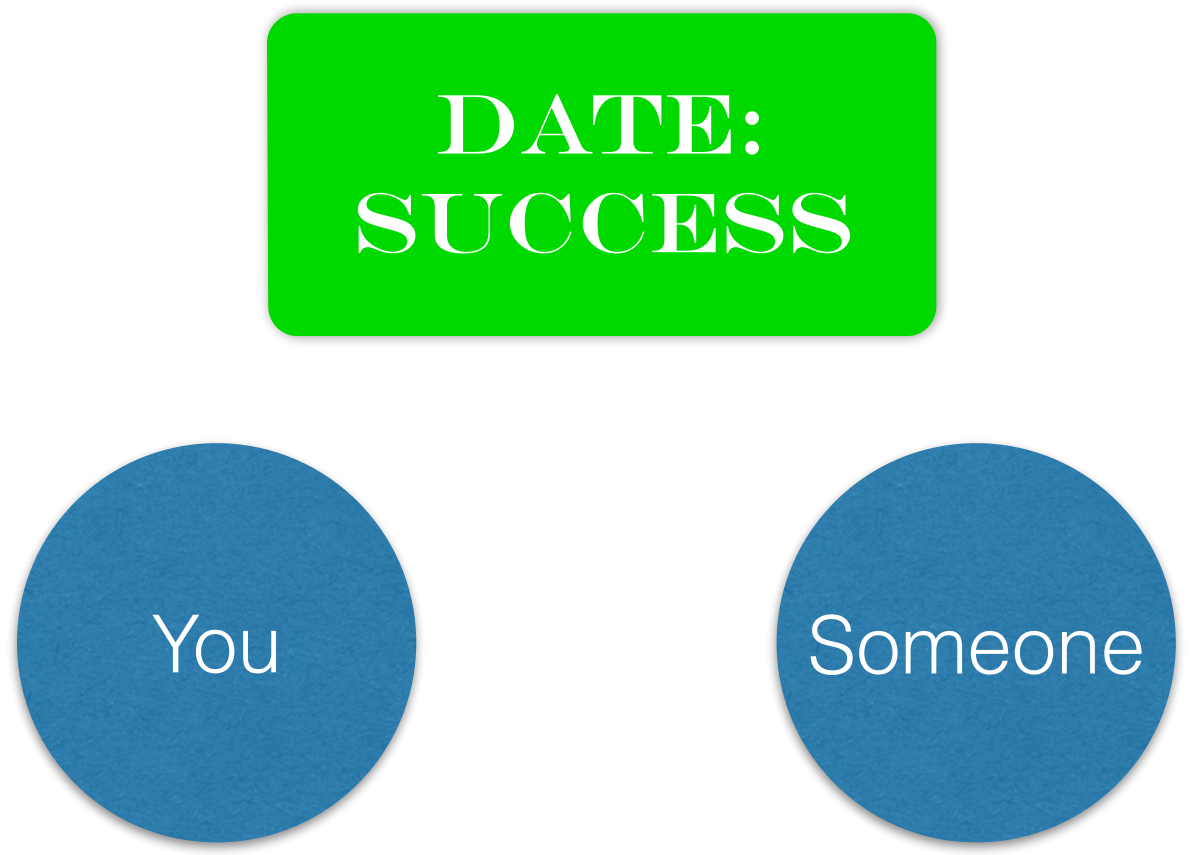
- In reinforcement learning terms, you are the *agent*.
- Everything else (the other person, the setting, etc.) is the *environment*.
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds.
- As the date goes on, you slowly figure out how you should behave based on what you've tried so far, and how it went.
- If you're a good agent, you may even learn how to behave really, really well!



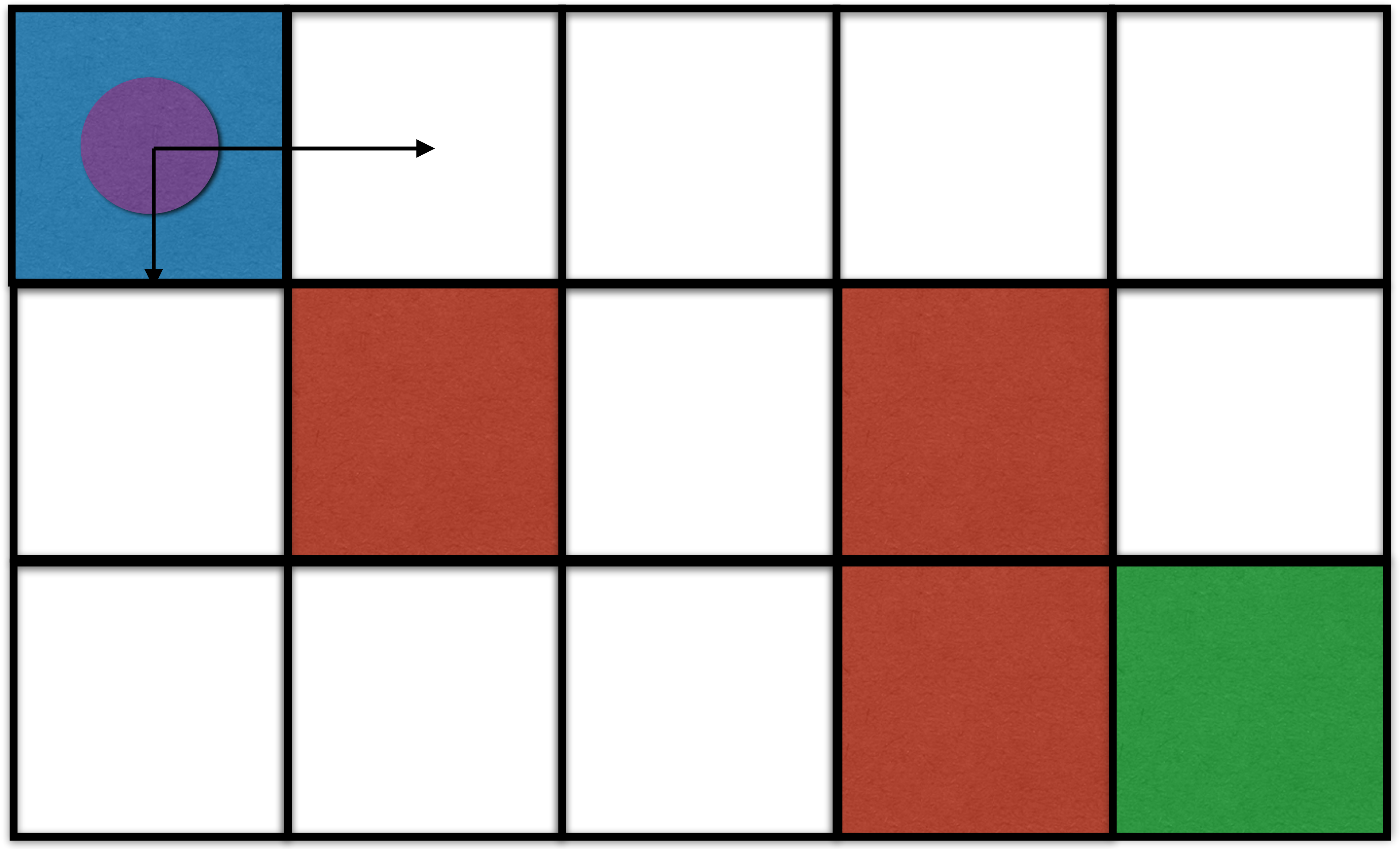
An Analogy

Suppose you go on a date with someone.

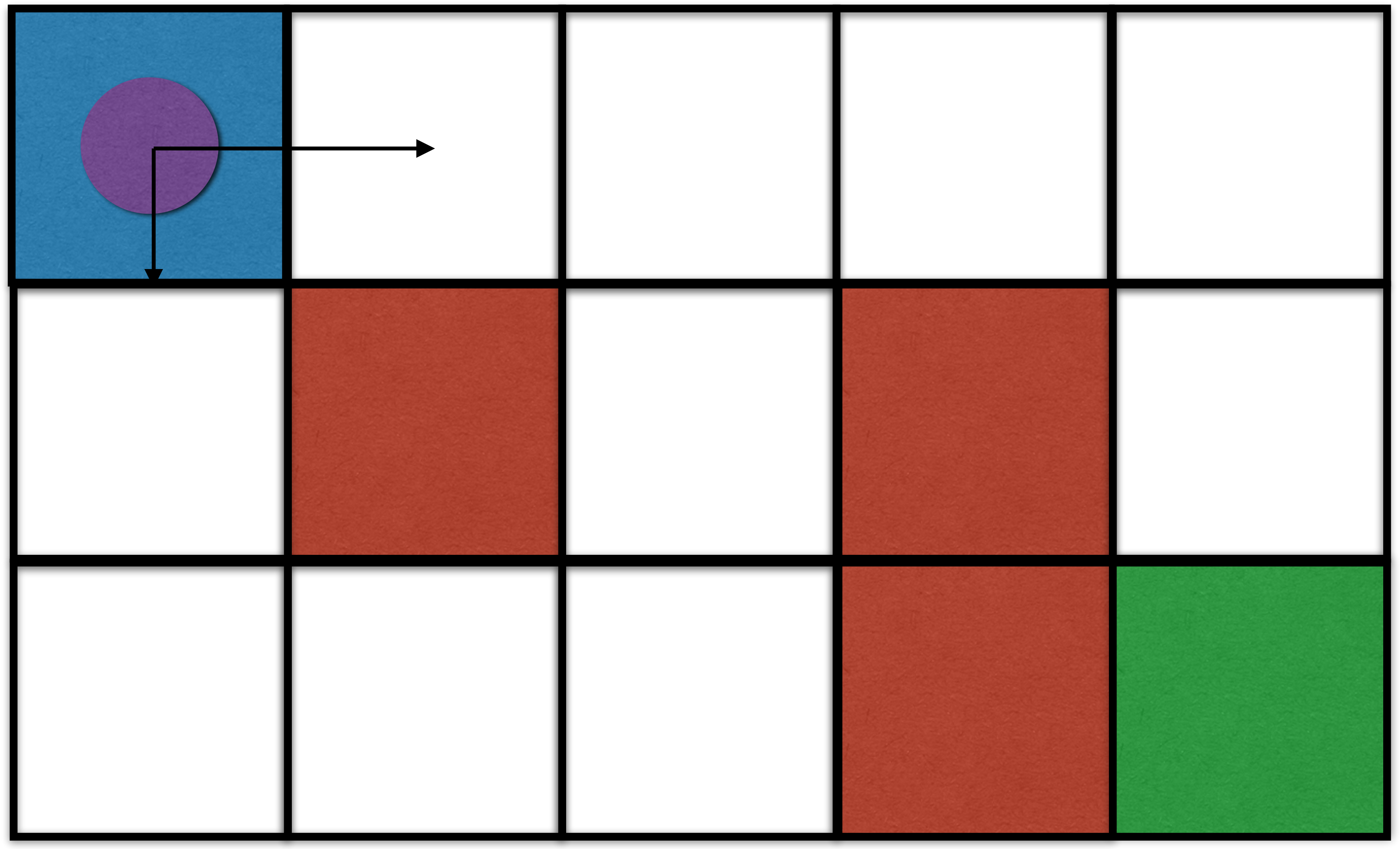
- In reinforcement learning terms, you are the *agent*.
- Everything else (the other person, the setting, etc.) is the *environment*.
- At the beginning of the date, you might not know how to act, so you try different things to see how the other person responds.
- As the date goes on, you slowly figure out how you should behave based on what you've tried so far, and how it went.
- If you're a good agent, you may even learn how to behave really, really well!



RL Example: Gridworld



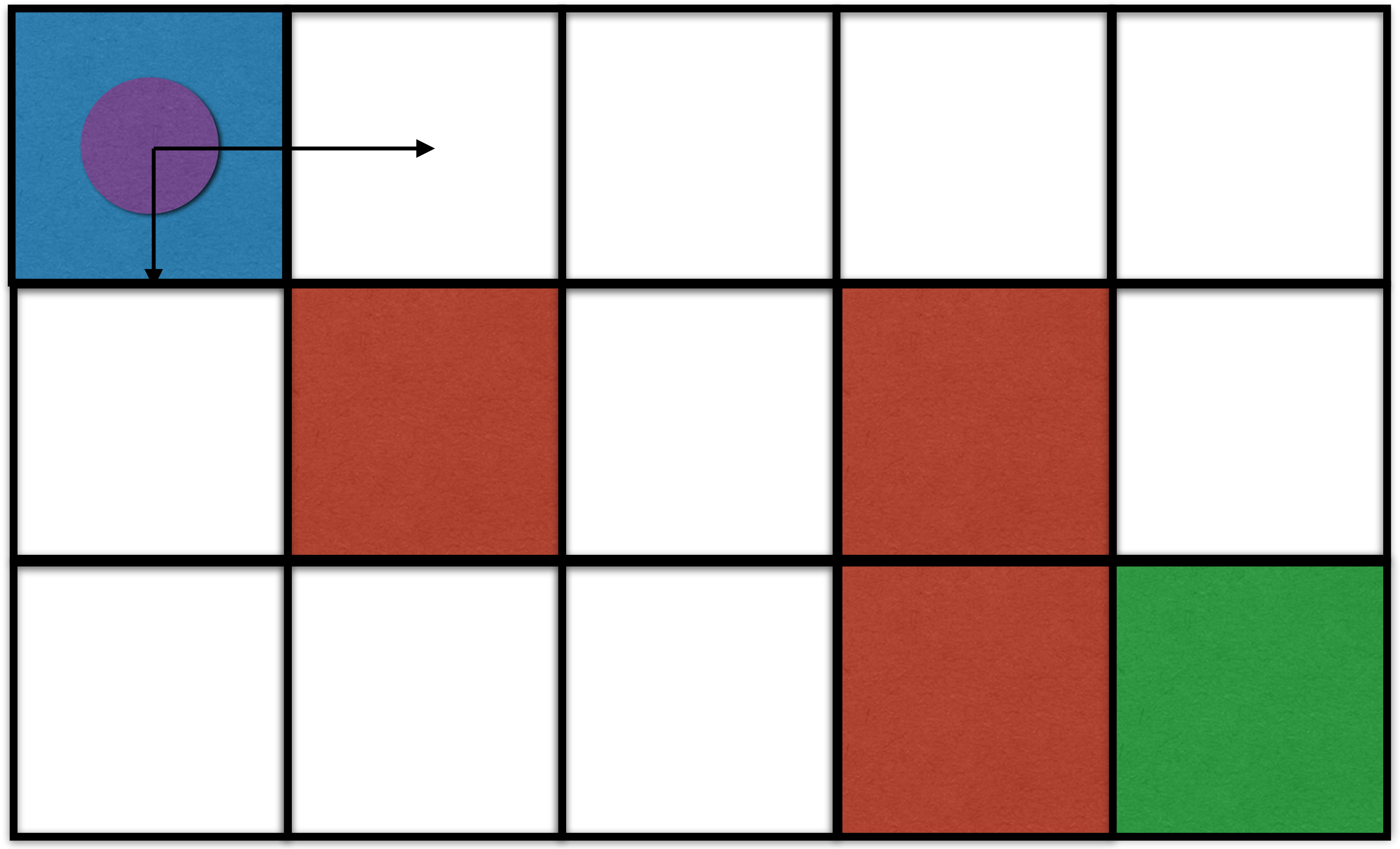
RL Example: Gridworld



What is the environment?

What is the agent?

RL Example: Gridworld

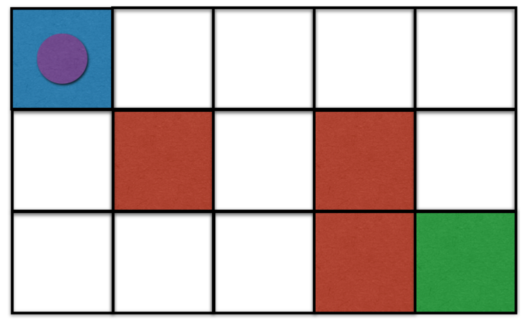


What is the environment?

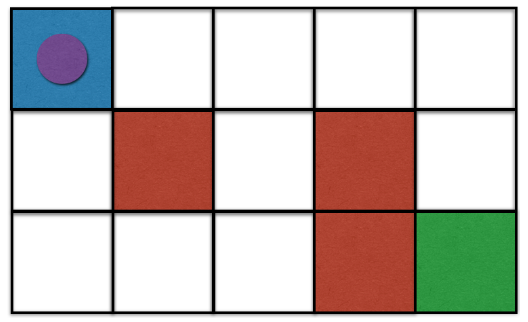
What is the agent?

The Problem: How do we get to the goal (green) from the start (blue) *as quickly as possible* while avoiding the obstacles (red)?

The RL Setting

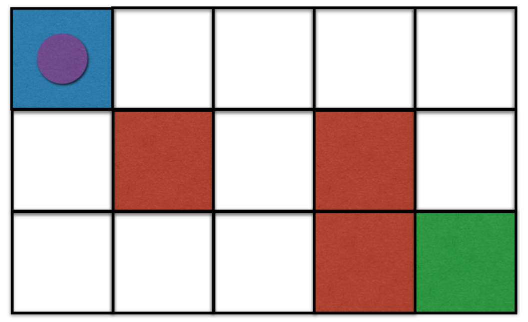


The RL Setting



The environment:

The RL Setting

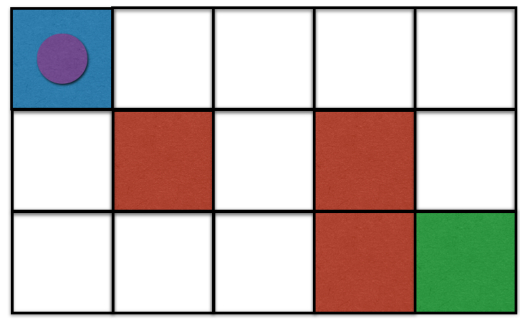


The environment:

- States (s):

  Configuration of the agent and environment.

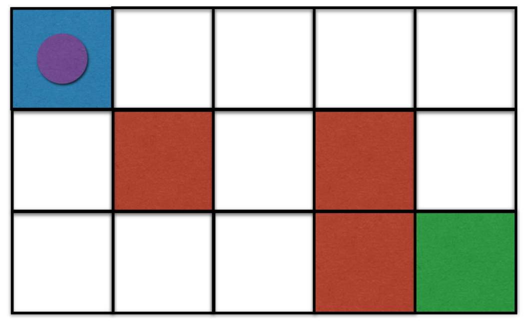
The RL Setting



The environment:

- States (s):
  Configuration of the agent and environment.
- Actions (a):
  What can the agent do in a state?

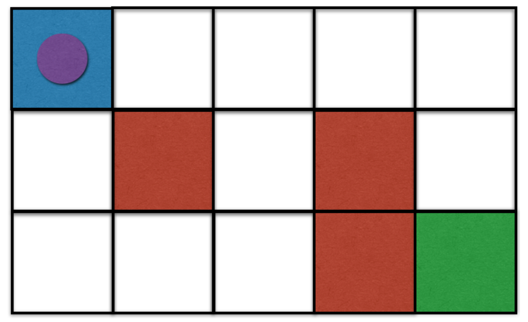
The RL Setting



The environment:

- States (s):
  Configuration of the agent and environment.
- Actions (a):
  What can the agent do in a state?
- Reward function (R):
  What reward does the agent get for each state?

The RL Setting

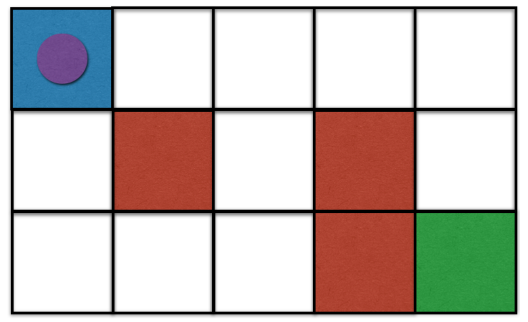


The environment:

- States (s):
  Configuration of the agent and environment.
- Actions (a):
  What can the agent do in a state?
- Reward function (R):
  What reward does the agent get for each state?

The agent:

The RL Setting



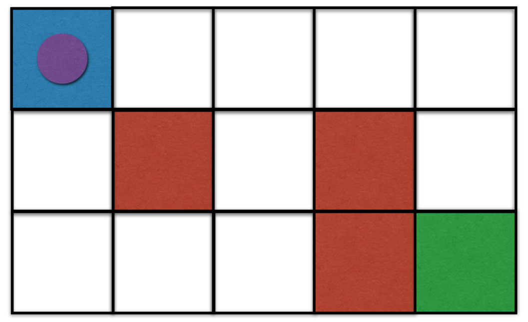
The environment:

- States (s):
  Configuration of the agent and environment.
- Actions (a):
  What can the agent do in a state?
- Reward function (R):
  What reward does the agent get for each state?

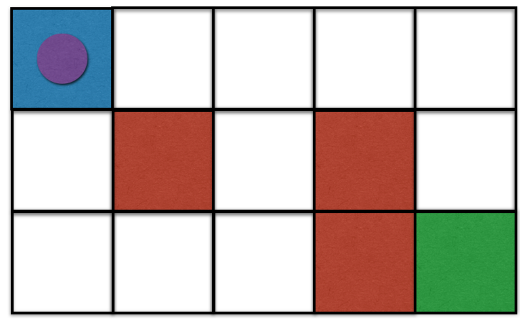
The agent:

- Policy (π):
  Given a state, what action will the agent take?

Gridworld Revisited

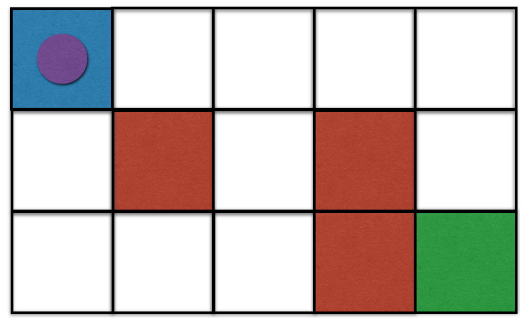


Gridworld Revisited



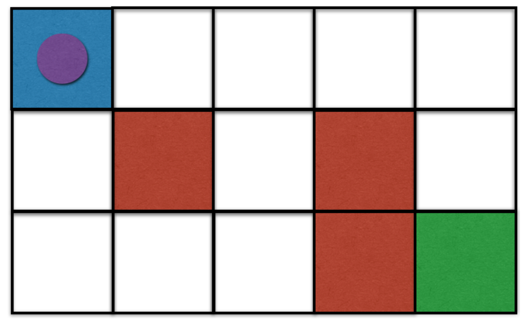
- The environment of Gridworld, in more detail:

Gridworld Revisited



- The environment of Gridworld, in more detail:
  - States (s):

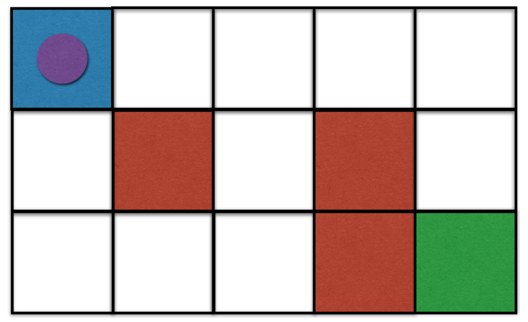
Gridworld Revisited



- The environment of Gridworld, in more detail:
  - States (s):

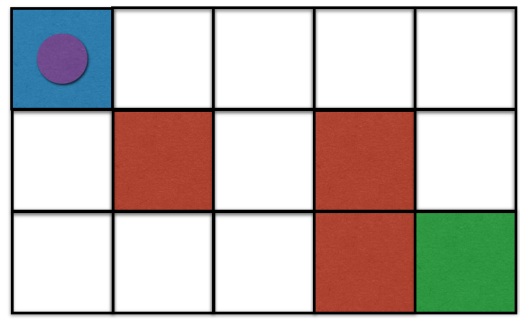
    What square is the agent in?

Gridworld Revisited



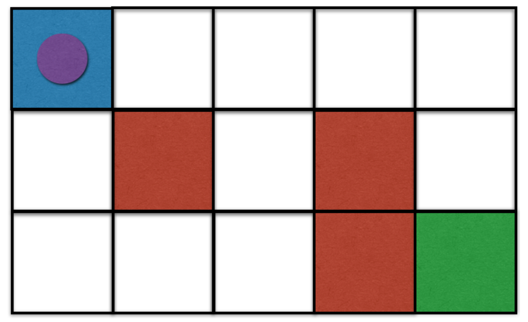
- The environment of Gridworld, in more detail:
  - States (s):
    What square is the agent in?
  - Actions (a):

Gridworld Revisited



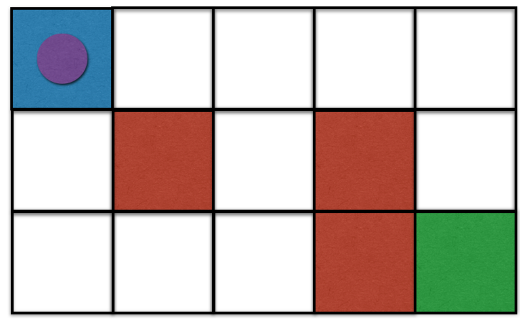
- The environment of Gridworld, in more detail:
  - States (s):
    What square is the agent in?
  - Actions (a):
    Go to an adjacent square, or stay put.

Gridworld Revisited



- The environment of Gridworld, in more detail:
  - States (s):
    What square is the agent in?
  - Actions (a):
    Go to an adjacent square, or stay put.
  - Reward function: ???

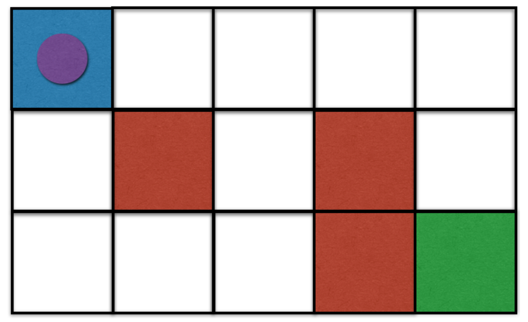
Gridworld Revisited



- The environment of Gridworld, in more detail:
  - States (s):
    What square is the agent in?
  - Actions (a):
    Go to an adjacent square, or stay put.
  - Reward function: ???

The Problem:
How do we get to the goal from the start *as quickly as possible* while avoiding the obstacles?

Gridworld Revisited



- The environment of Gridworld, in more detail:
  - States (s):
    What square is the agent in?
  - Actions (a):
    Go to an adjacent square, or stay put.
  - Reward function: ???

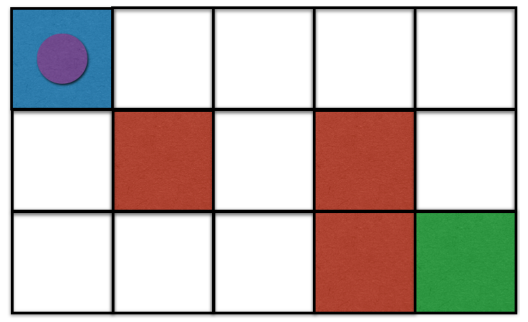
The Problem:

How do we get to the goal from the start *as quickly as possible* while avoiding the obstacles?

In RL terminology:

What is the optimal policy π^* that *maximizes* my expected reward over time?

Gridworld Revisited

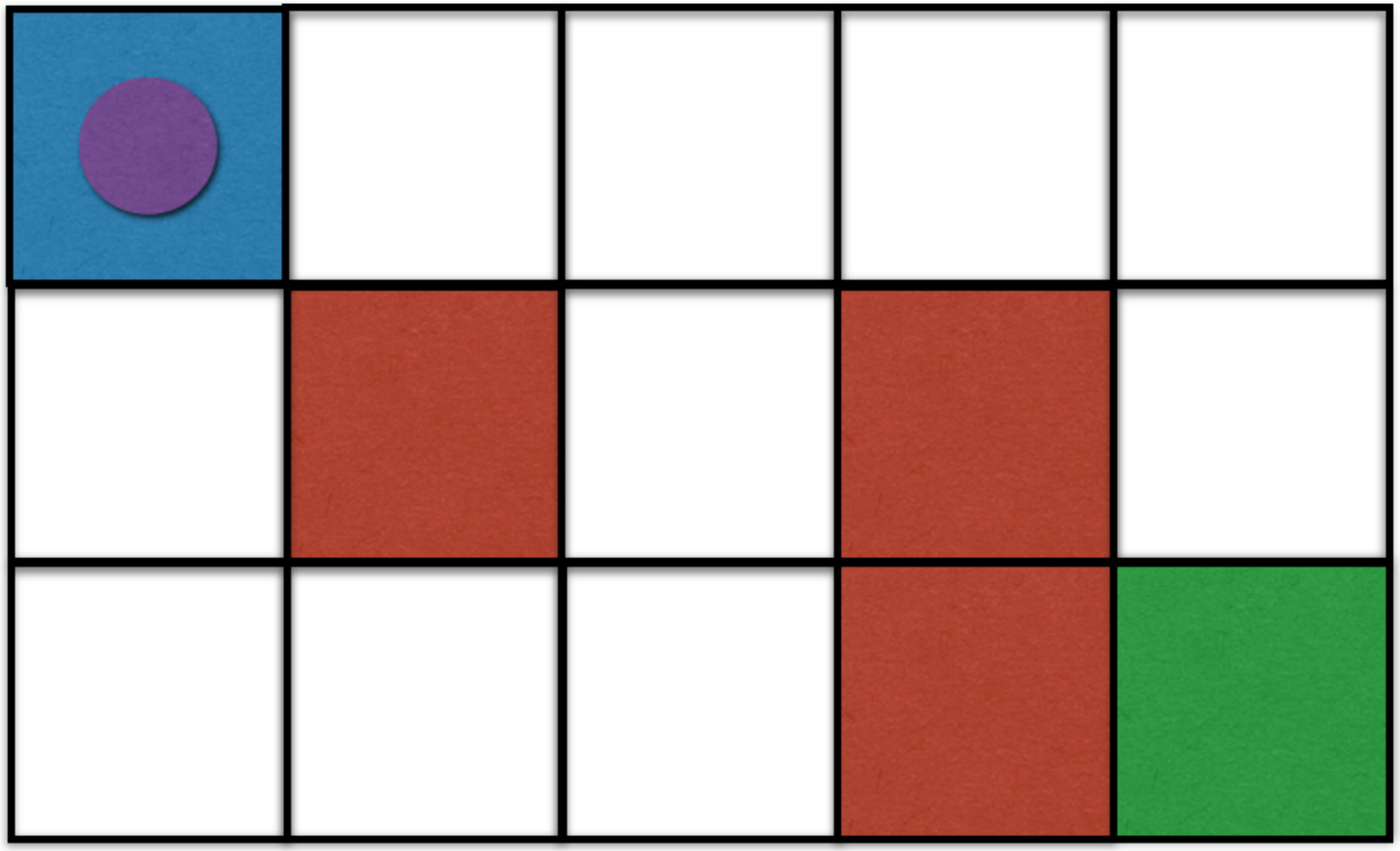


- The environment of Gridworld, in more detail:
  - States (s):
    What square is the agent in?
  - Actions (a):
    Go to an adjacent square, or stay put.
  - Reward function: ???

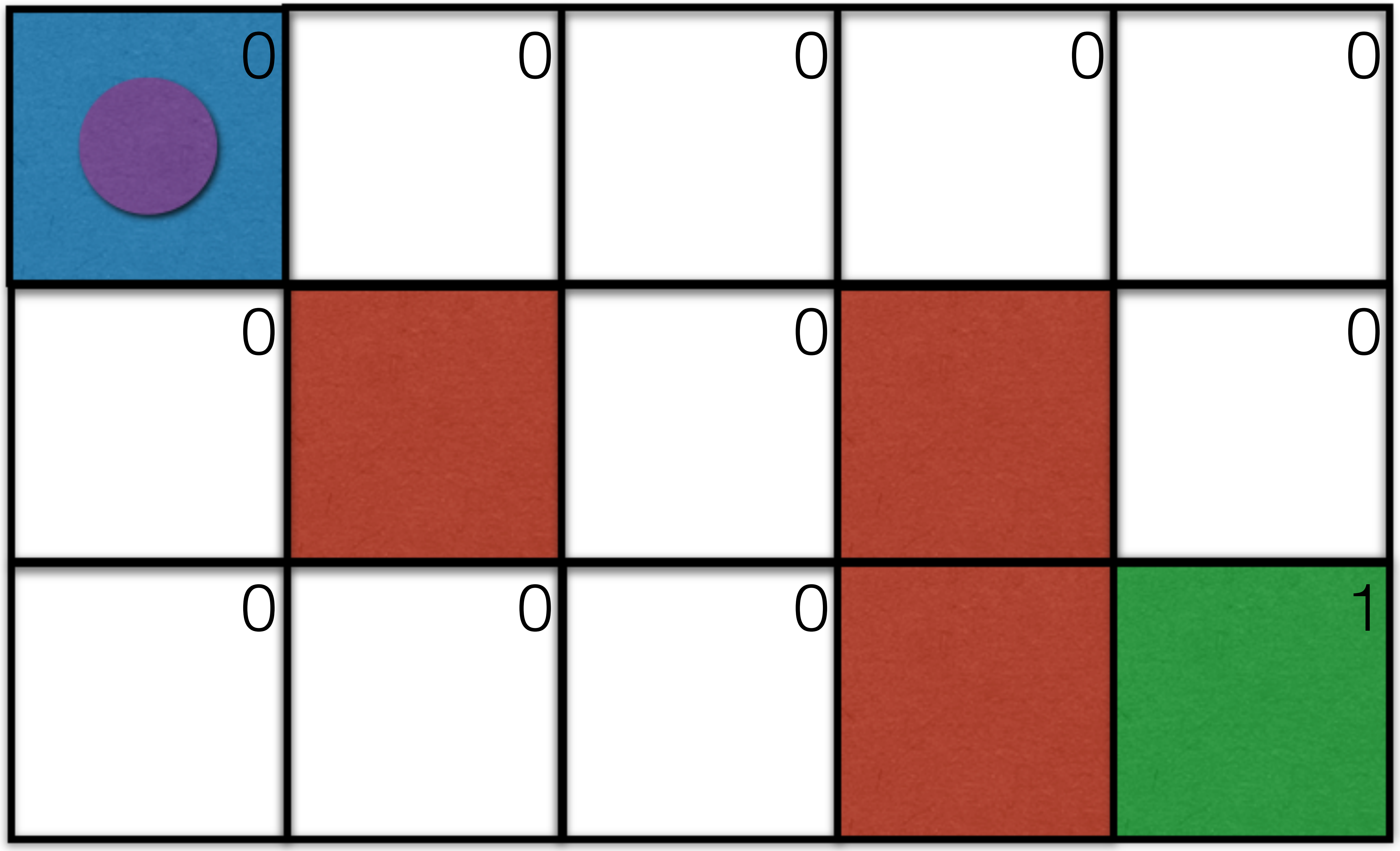
The Problem:
How do we get to the goal from the start *as quickly as possible* while avoiding the obstacles?

In RL terminology:
What is the optimal policy $\pi*$ that *maximizes* my expected reward over time?

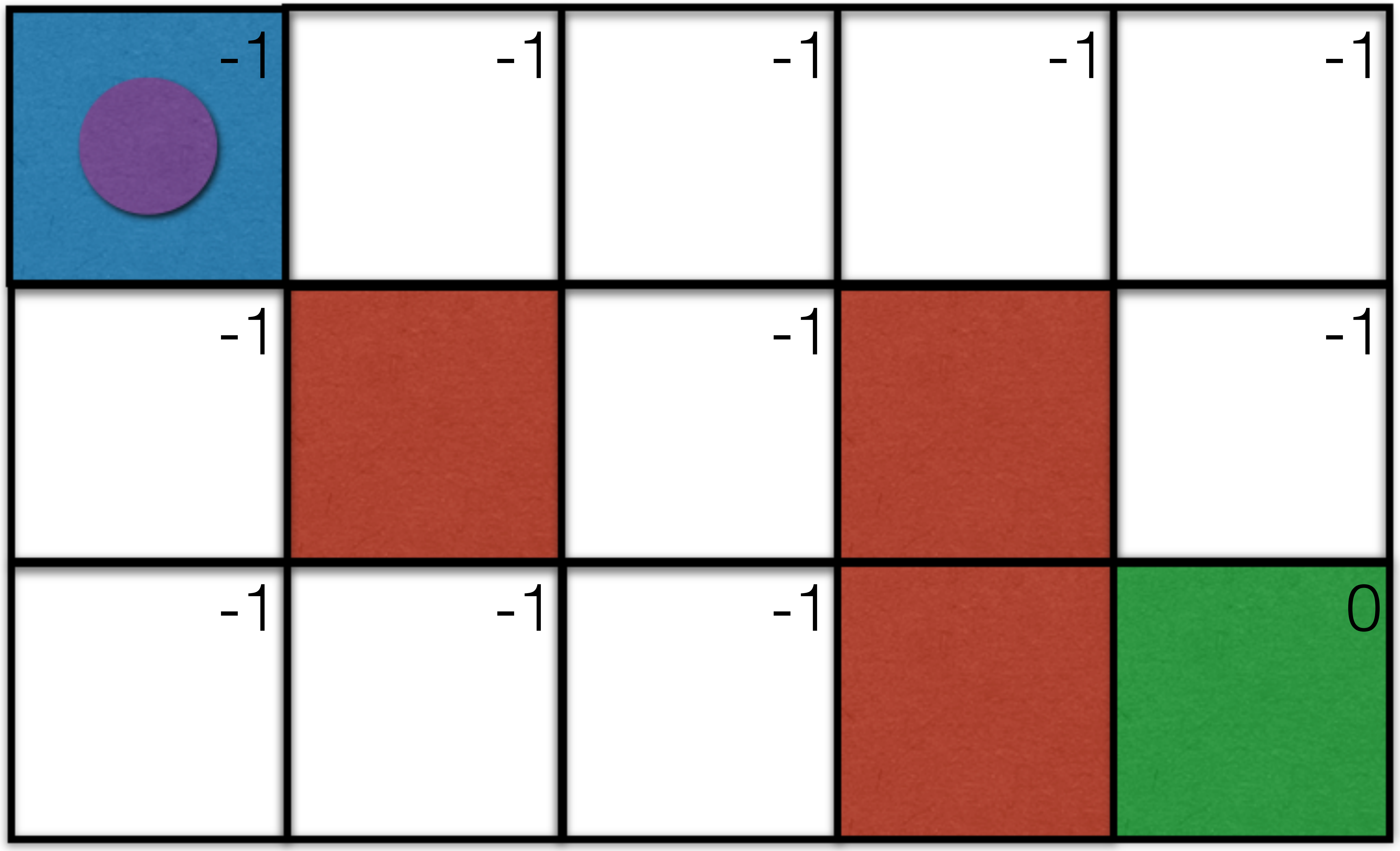
Gridworld Reward Function



Gridworld Reward Function



Gridworld Reward Function



Value Function

Value Function

- Reward function: R(s) = reward of being in state s

Value Function

- Reward function: R(s) = reward of being in state s
- Value function: V(s) = value of being in state s

Value Function

- Reward function: R(s) = reward of being in state s
- Value function: V(s) = value of being in state s
- The value of s is the *long-term expected reward* starting from s.

Value Function

- Reward function: R(s) = reward of being in state s
- Value function: V(s) = value of being in state s
- The value of s is the *long-term expected reward* starting from s.

- How do we determine where to go after s?

Value Function

- Reward function: R(s) = reward of being in state s
- Value function: V(s) = value of being in state s
- The value of s is the *long-term expected reward* starting from s.

- How do we determine where to go after s?
  - We use our policy π to determine which actions to take.

Value Function

- Reward function: R(s) = reward of being in state s
- Value function: V(s) = value of being in state s
- The value of s is the *long-term expected reward* starting from s.

- How do we determine where to go after s?
  - We use our policy π to determine which actions to take.
  - So, the value function *also depends on our policy*.

Value Function

- Reward function: R(s) = reward of being in state s
- Value function: V(s) = value of being in state s
- The value of s is the *long-term expected reward* starting from s.

- How do we determine where to go after s?
  - We use our policy π to determine which actions to take.
  - So, the value function *also depends on our policy*.

- How do we determine the value of a state?

Value Function

- Reward function: R(s) = reward of being in state s
- Value function: V(s) = value of being in state s
- The value of s is the *long-term expected reward* starting from s.

- How do we determine where to go after s?
  - We use our policy π to determine which actions to take.
  - So, the value function *also depends on our policy*.

- How do we determine the value of a state?
  - The value of a state is the *reward* of the state plus the *value* of the state we end up in next.

Value Function

- Reward function: R(s) = reward of being in state s
- Value function: V(s) = value of being in state s
- The value of s is the *long-term expected reward* starting from s.

- How do we determine where to go after s?
  - We use our policy π to determine which actions to take.
  - So, the value function *also depends on our policy*.

- How do we determine the value of a state?
  - The value of a state is the *reward* of the state plus the *value* of the state we end up in next.

$$V^{\pi}(s) = R(s) + \sum_{s'} P(s, \pi(s), s') V^{\pi}(s')$$

Value Function

$$V^{\pi}(s) = R(s) + \sum_{s'} P(s, \pi(s), s') V^{\pi}(s')$$

Value Function

$$V^\pi(s) = R(s) + \sum_{s'} P(s, \pi(s), s')V^\pi(s')$$

- How do we solve this equation? Use recursion!

Value Function

$$V^\pi(s) = R(s) + \sum_{s'} P(s, \pi(s), s') V^\pi(s')$$

- How do we solve this equation? Use recursion!
- What’s our base case?

Value Function

$$V^\pi(s) = R(s) + \sum_{s'} P(s, \pi(s), s') V^\pi(s')$$

- How do we solve this equation? Use recursion!
- What’s our base case?
  - If we’re at our *goal*, then there is no next state, so the value is just the reward.

Value Function

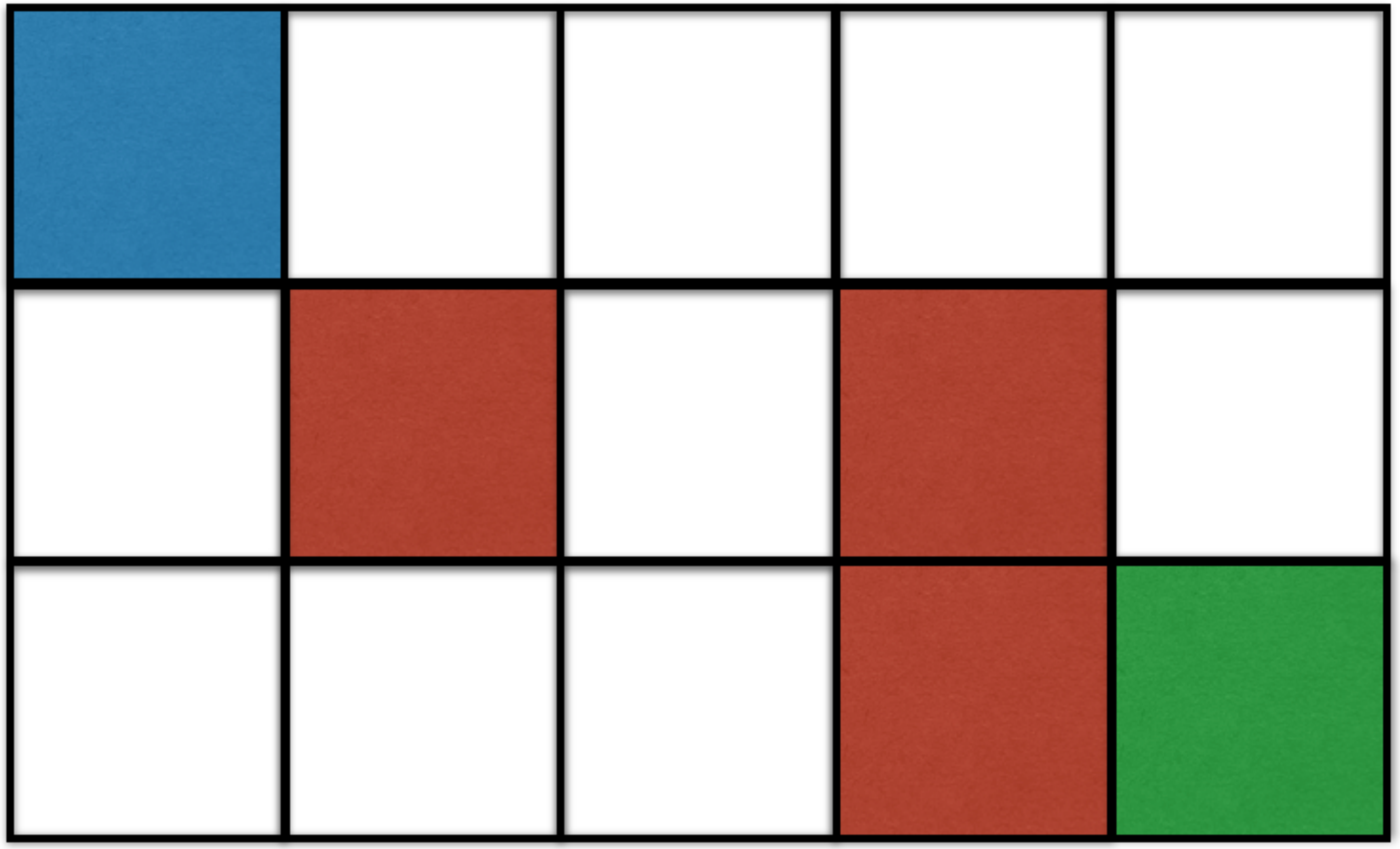
$$V^\pi(s) = R(s) + \sum_{s'} P(s, \pi(s), s')V^\pi(s')$$

- How do we solve this equation? Use recursion!
- What's our base case?
  - If we're at our *goal*, then there is no next state, so the value is just the reward.

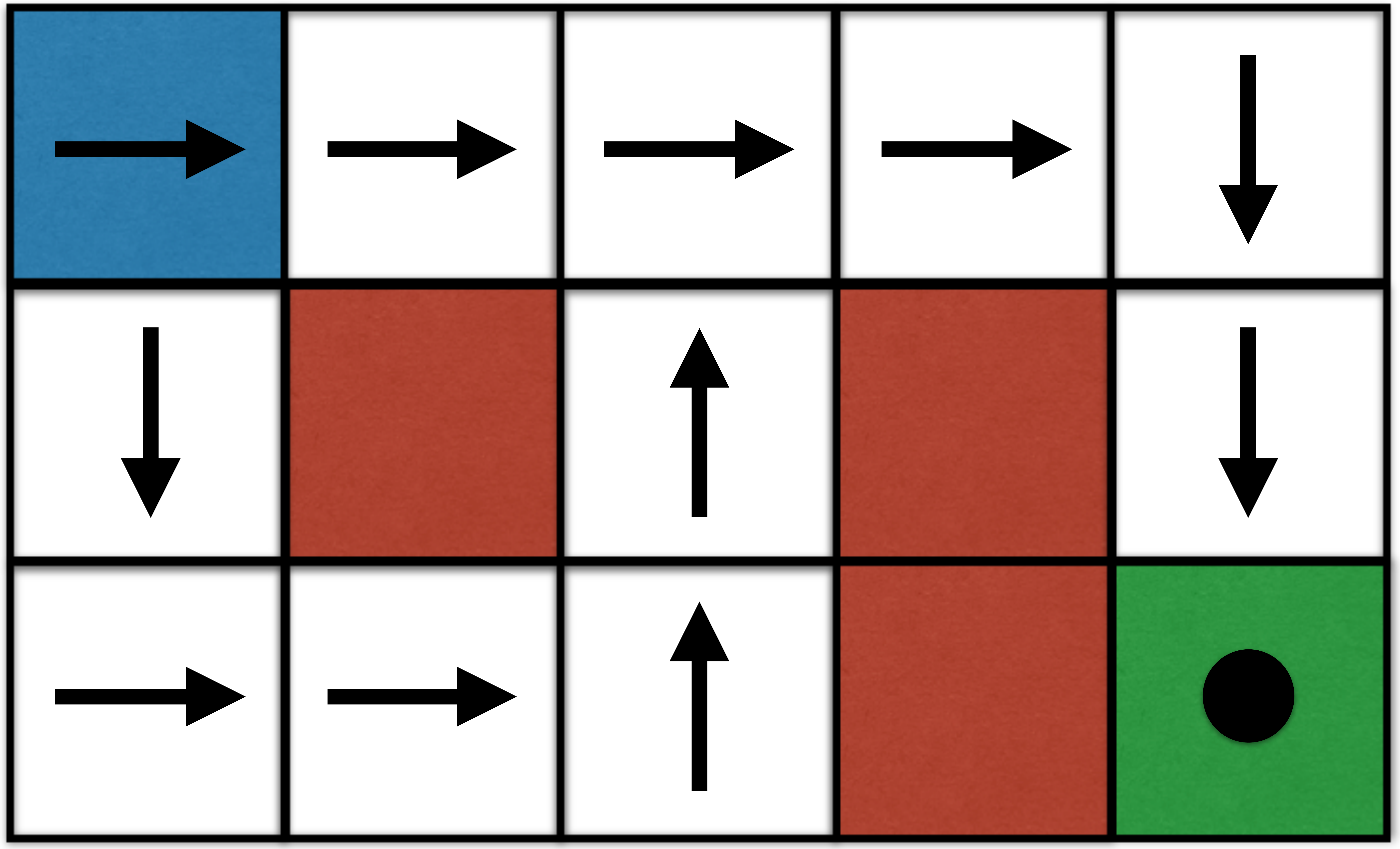
```
def V(s):
    reward = R(s)
    if is_goal(s):
        return reward
    return reward +
        sum([P(s, pi(s), n_s) * V(n_s) for n_s in states])
```

Gridworld Value Function Example

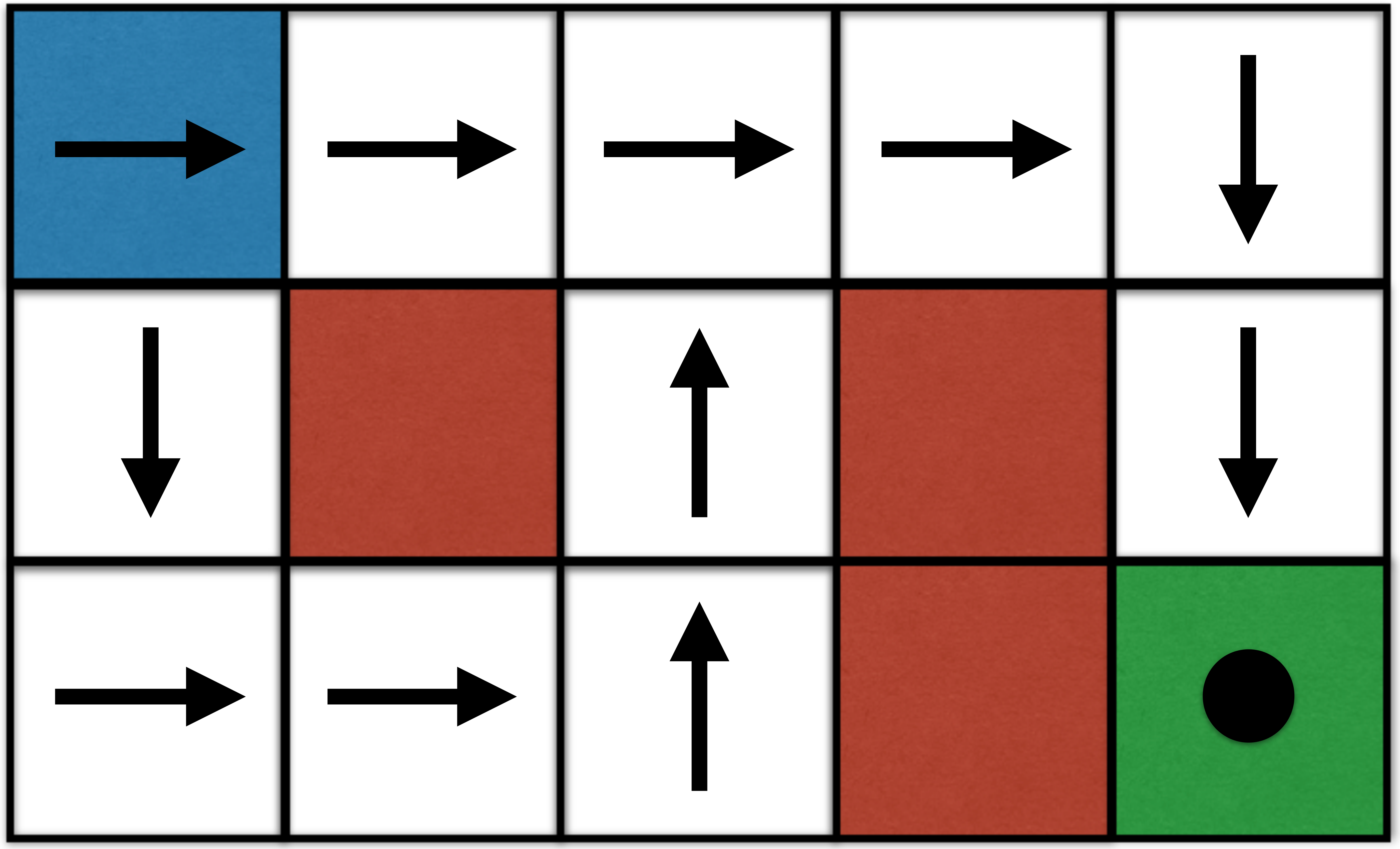
Gridworld Value Function Example



Gridworld Value Function Example

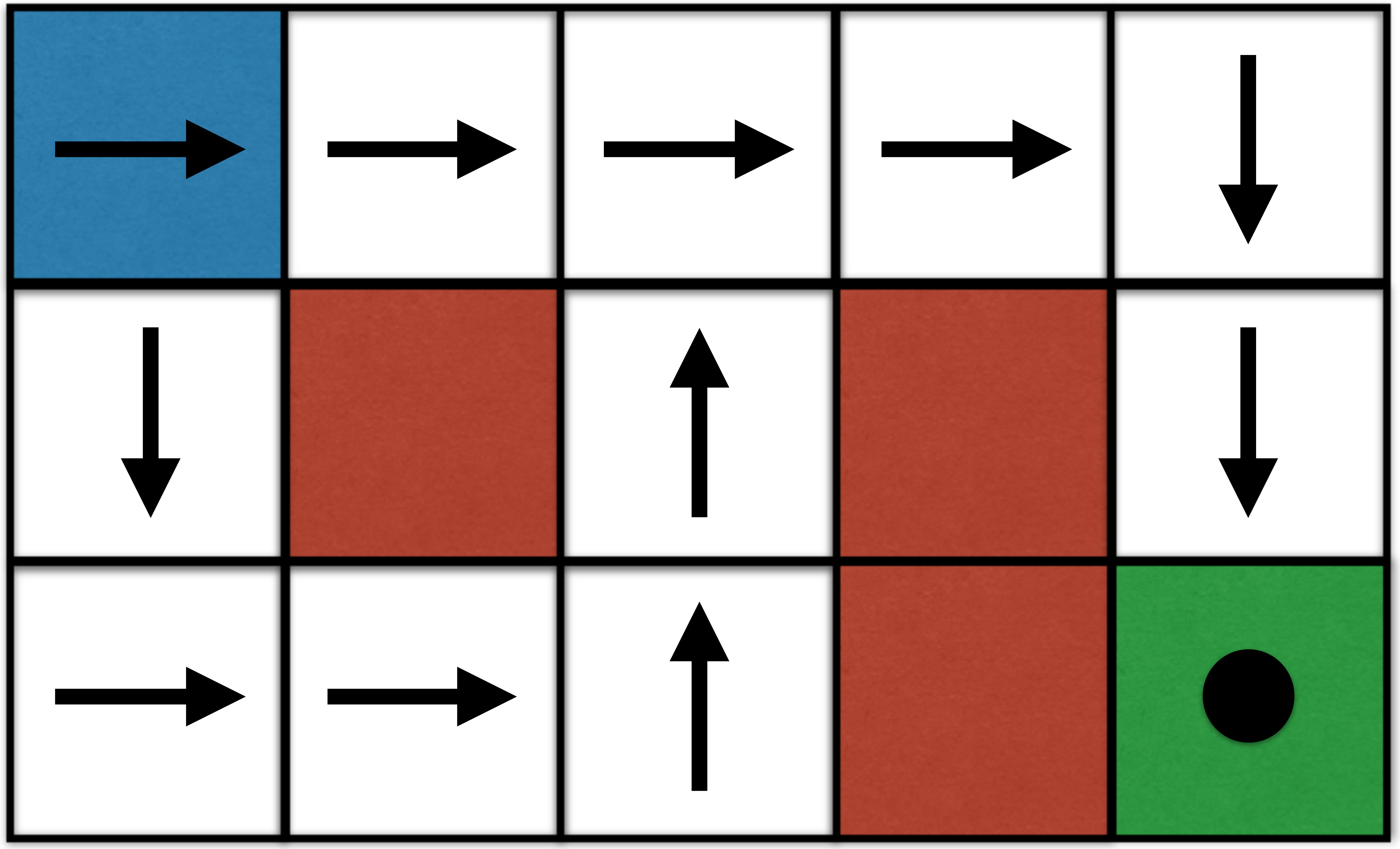


Gridworld Value Function Example



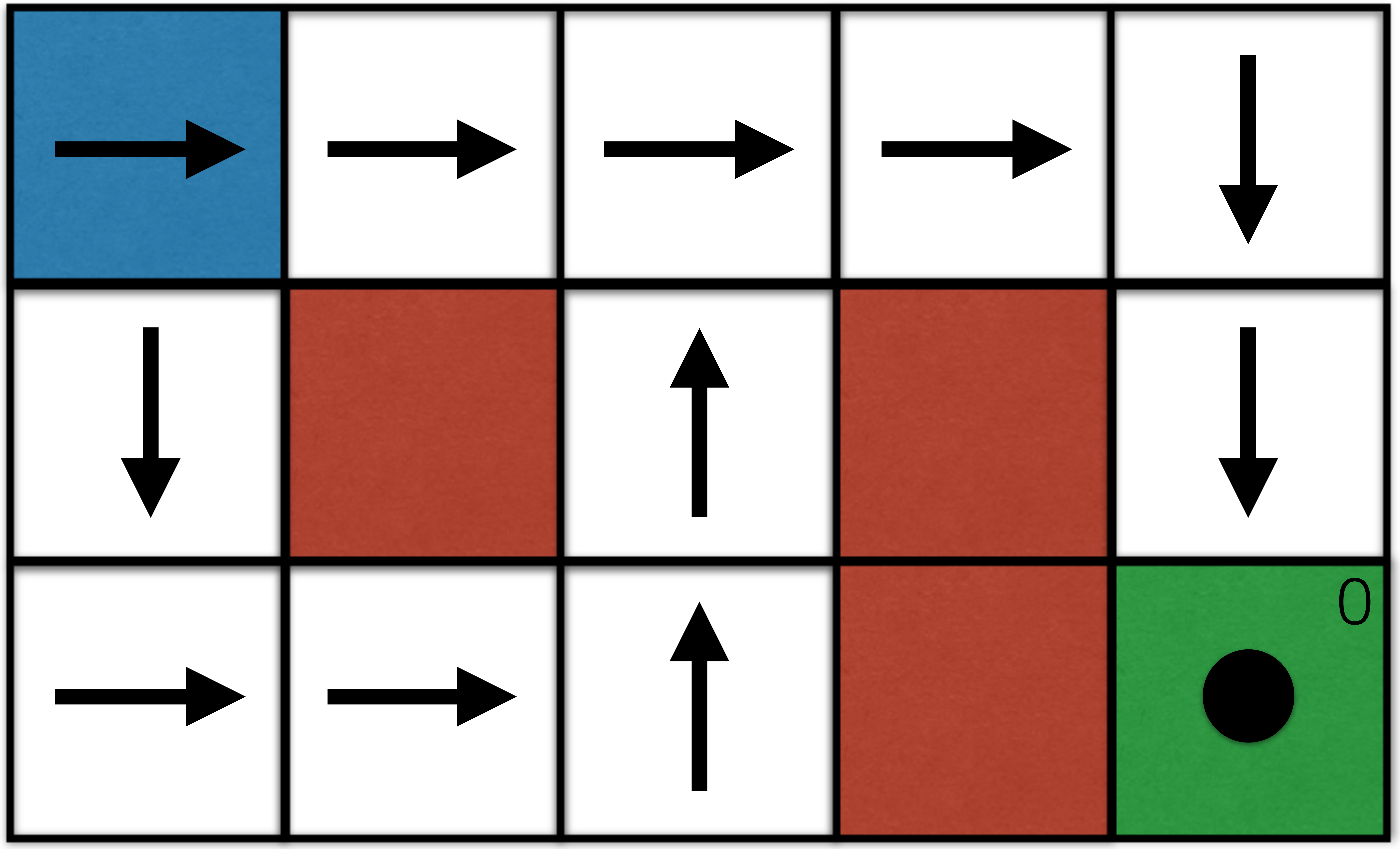
- Arrows denote the policy π.

Gridworld Value Function Example



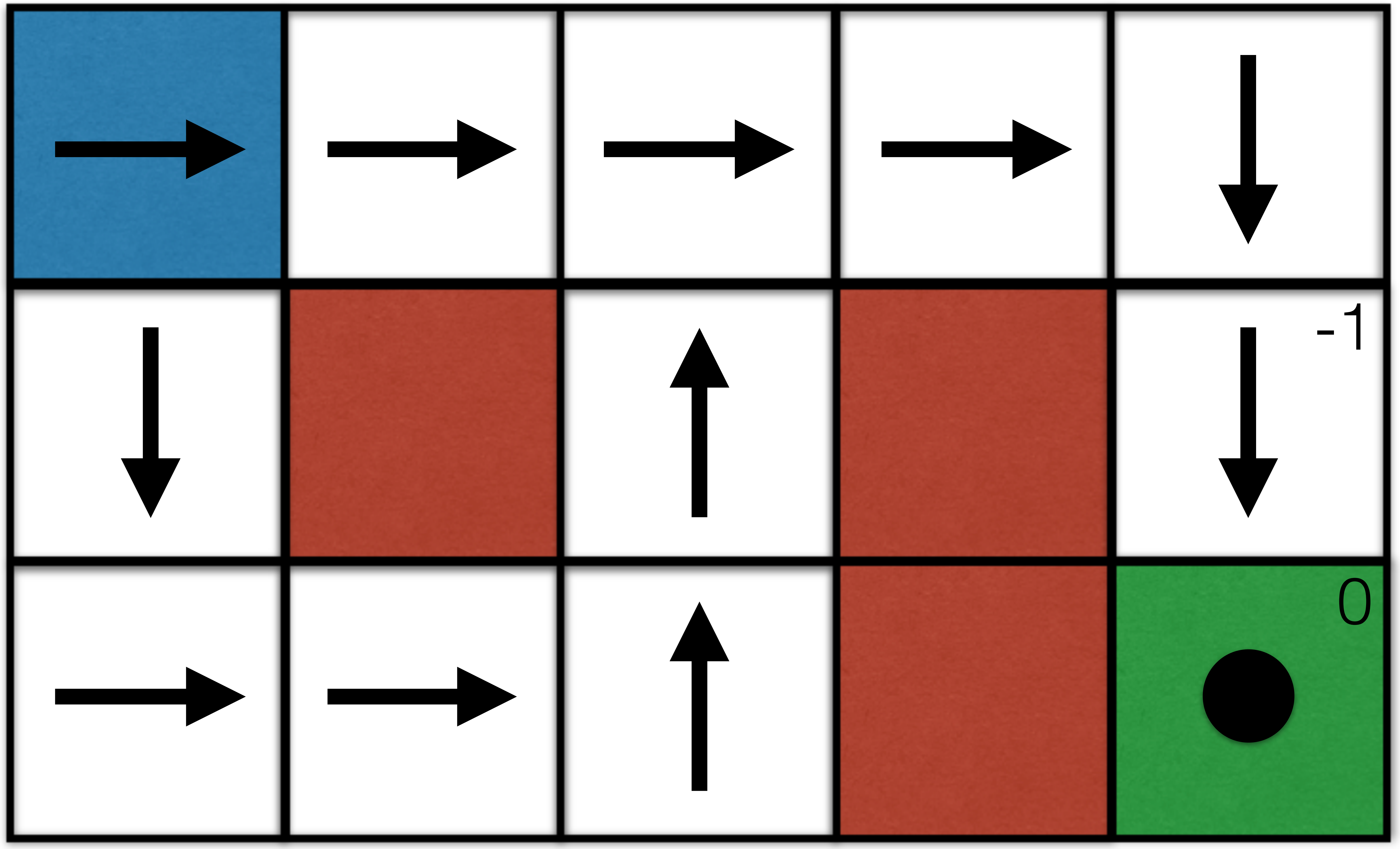
- Arrows denote the policy π.
- What are the values of the states, assuming no random movements?

Gridworld Value Function Example



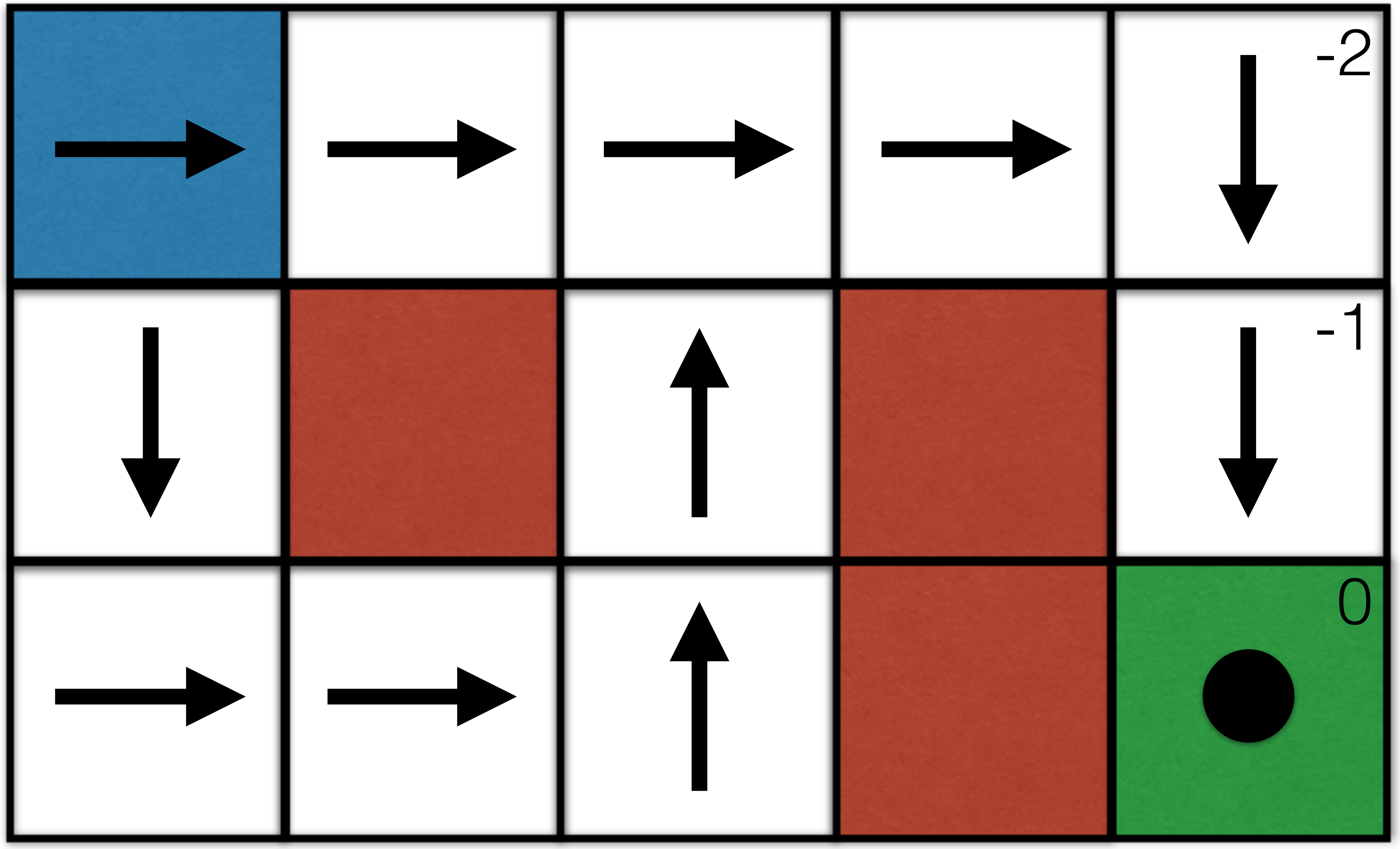
- Arrows denote the policy π.
- What are the values of the states, assuming no random movements?

Gridworld Value Function Example



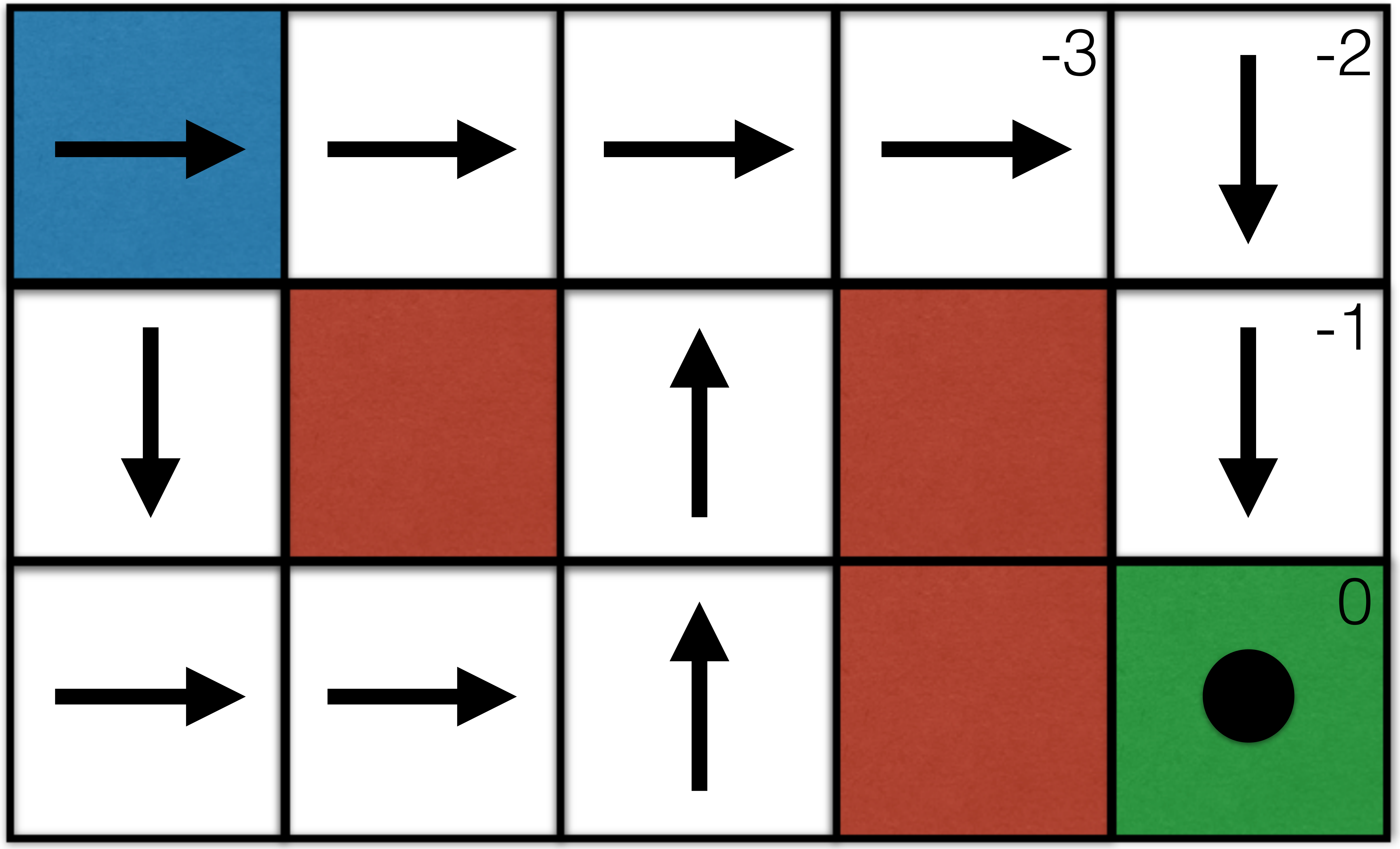
- Arrows denote the policy π.
- What are the values of the states, assuming no random movements?

Gridworld Value Function Example



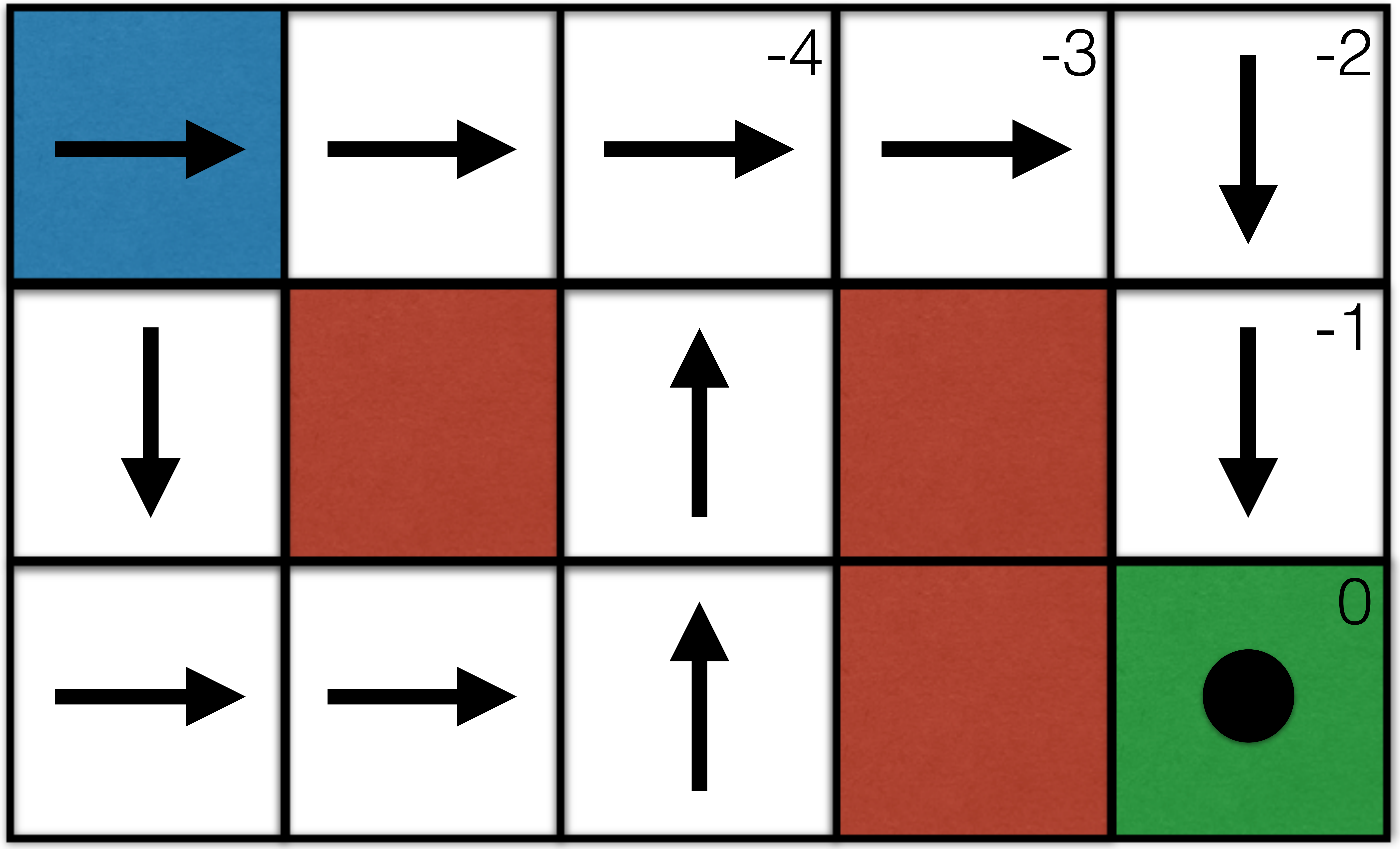
- Arrows denote the policy π.
- What are the values of the states, assuming no random movements?

Gridworld Value Function Example



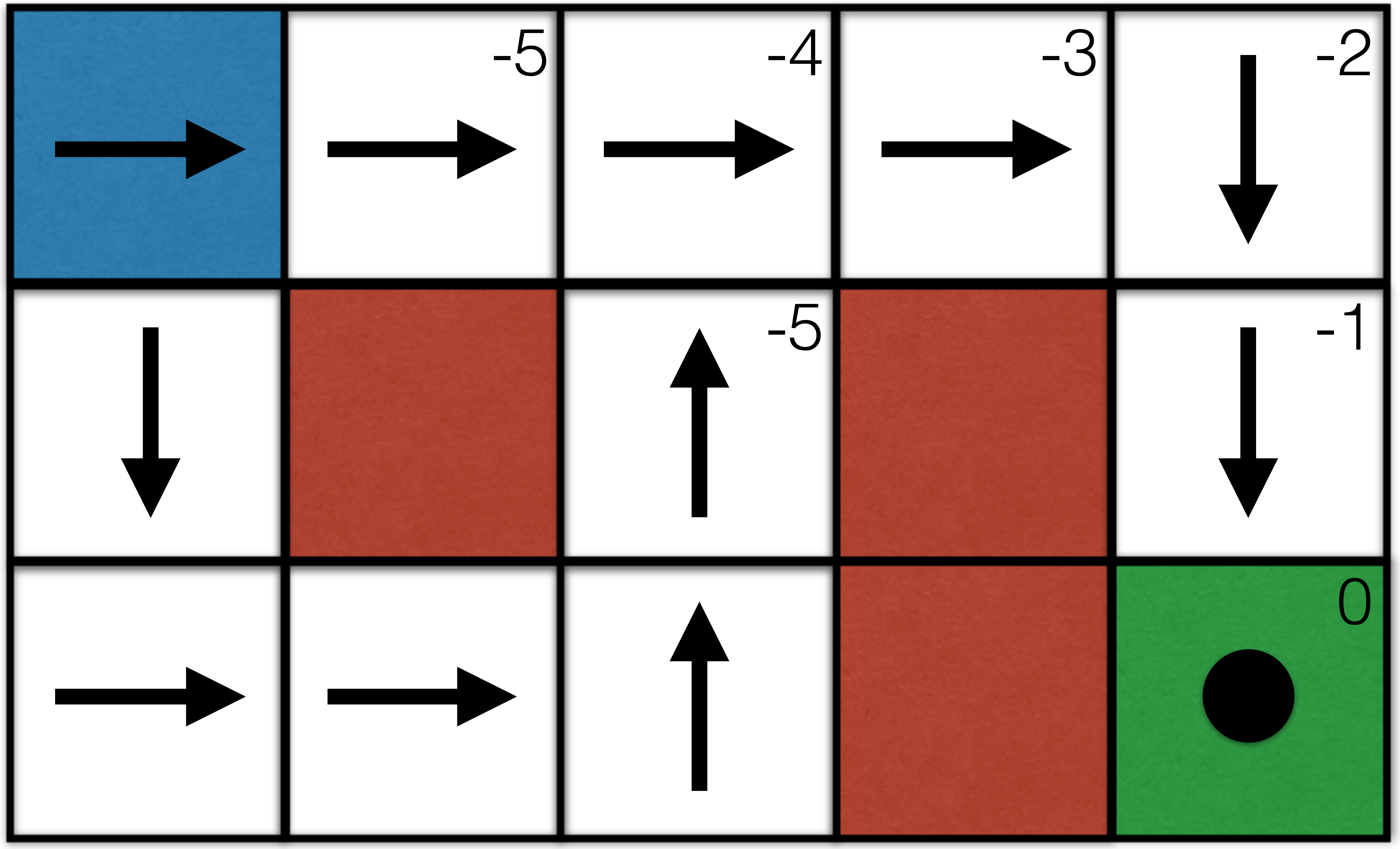
- Arrows denote the policy π.
- What are the values of the states, assuming no random movements?

Gridworld Value Function Example



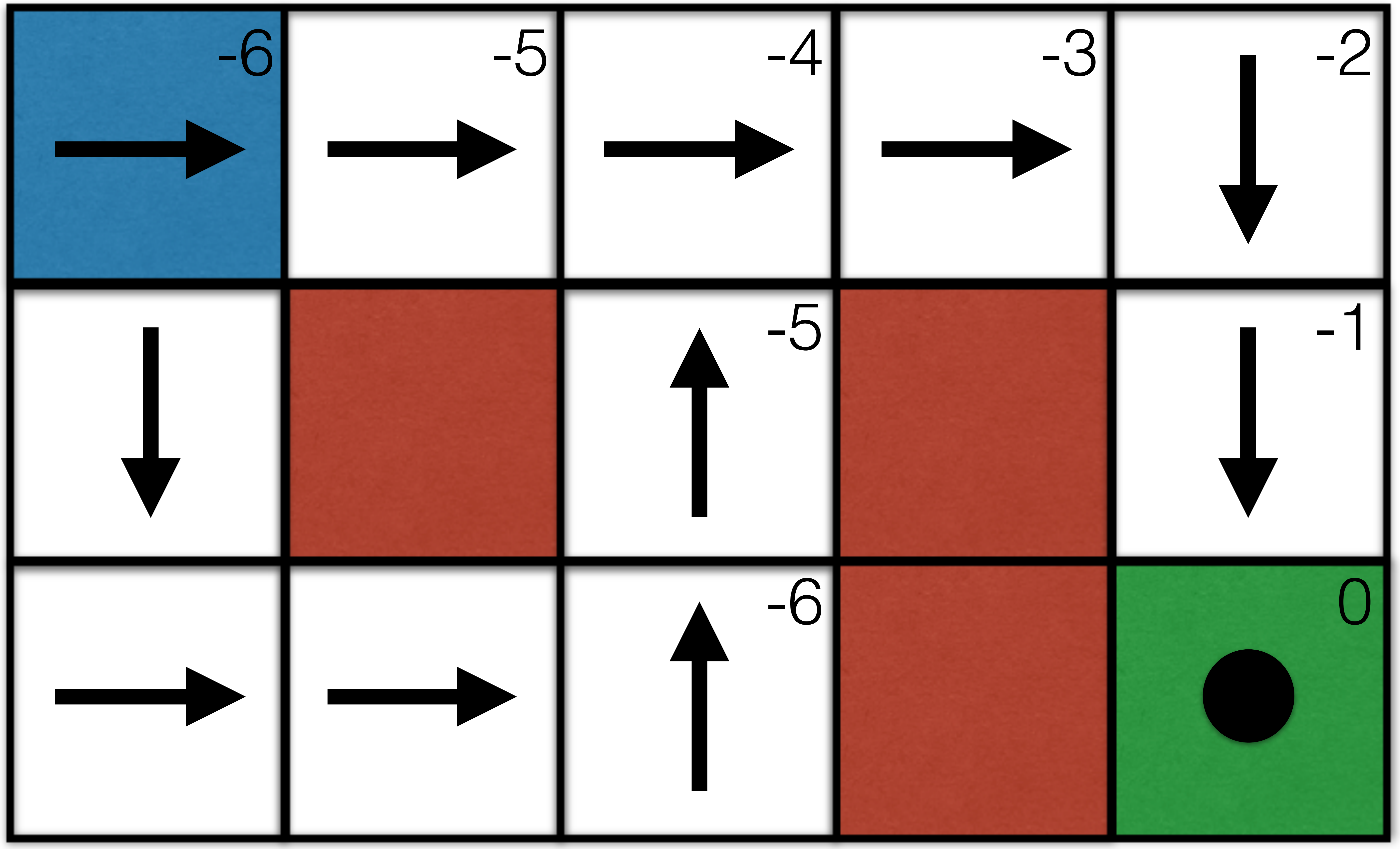
- Arrows denote the policy π.
- What are the values of the states, assuming no random movements?

Gridworld Value Function Example



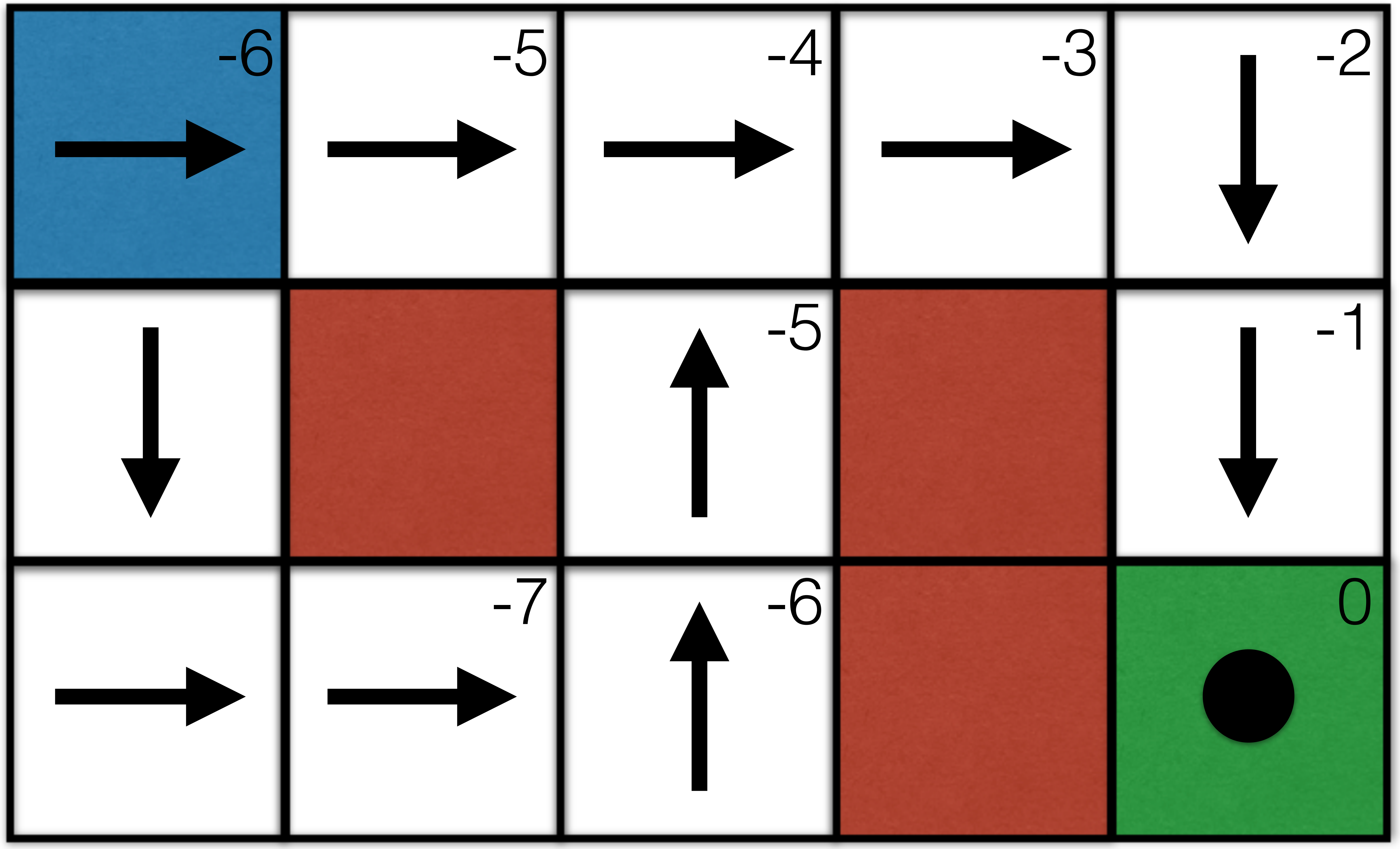
- Arrows denote the policy π.
- What are the values of the states, assuming no random movements?

Gridworld Value Function Example



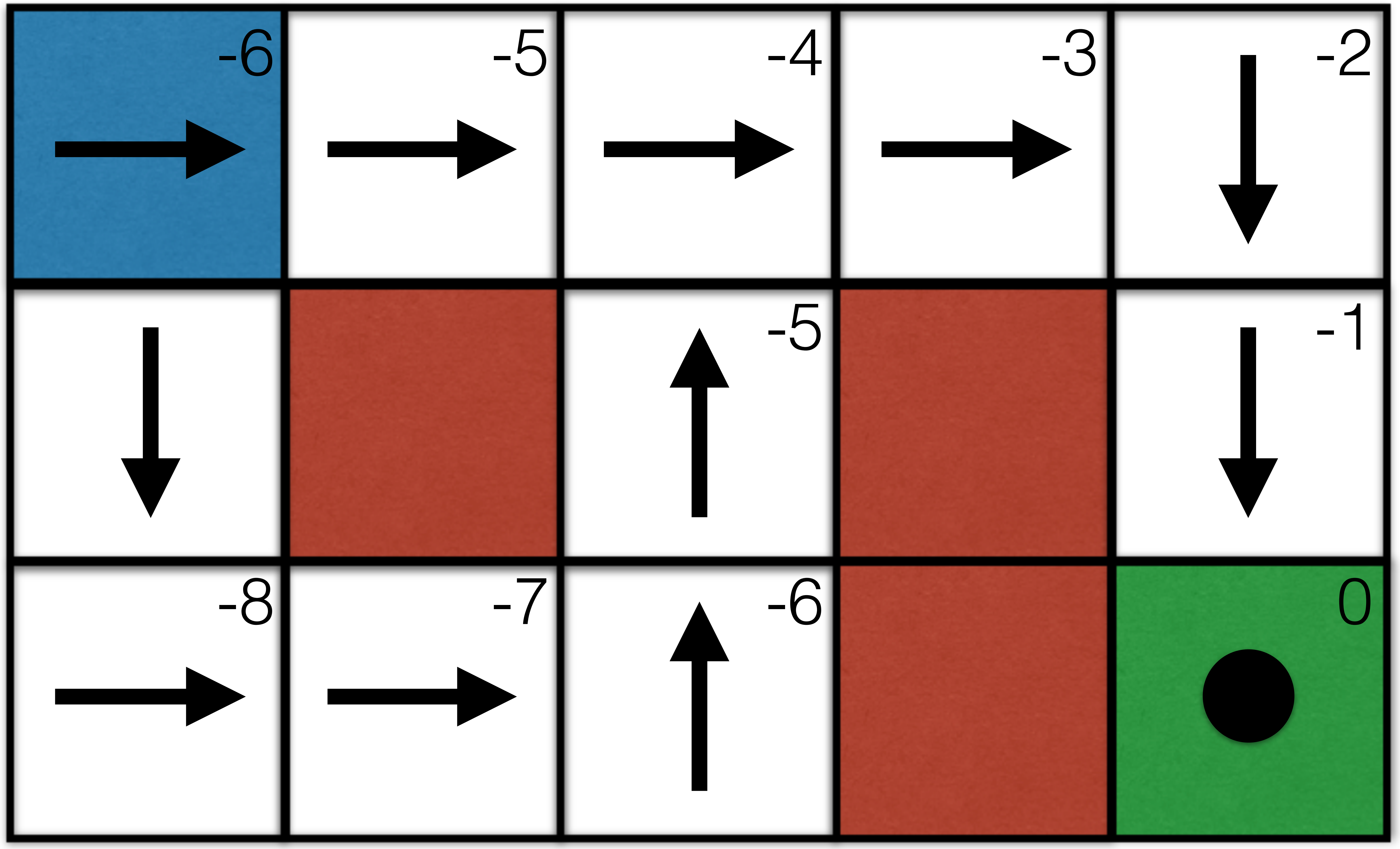
- Arrows denote the policy π.
- What are the values of the states, assuming no random movements?

Gridworld Value Function Example



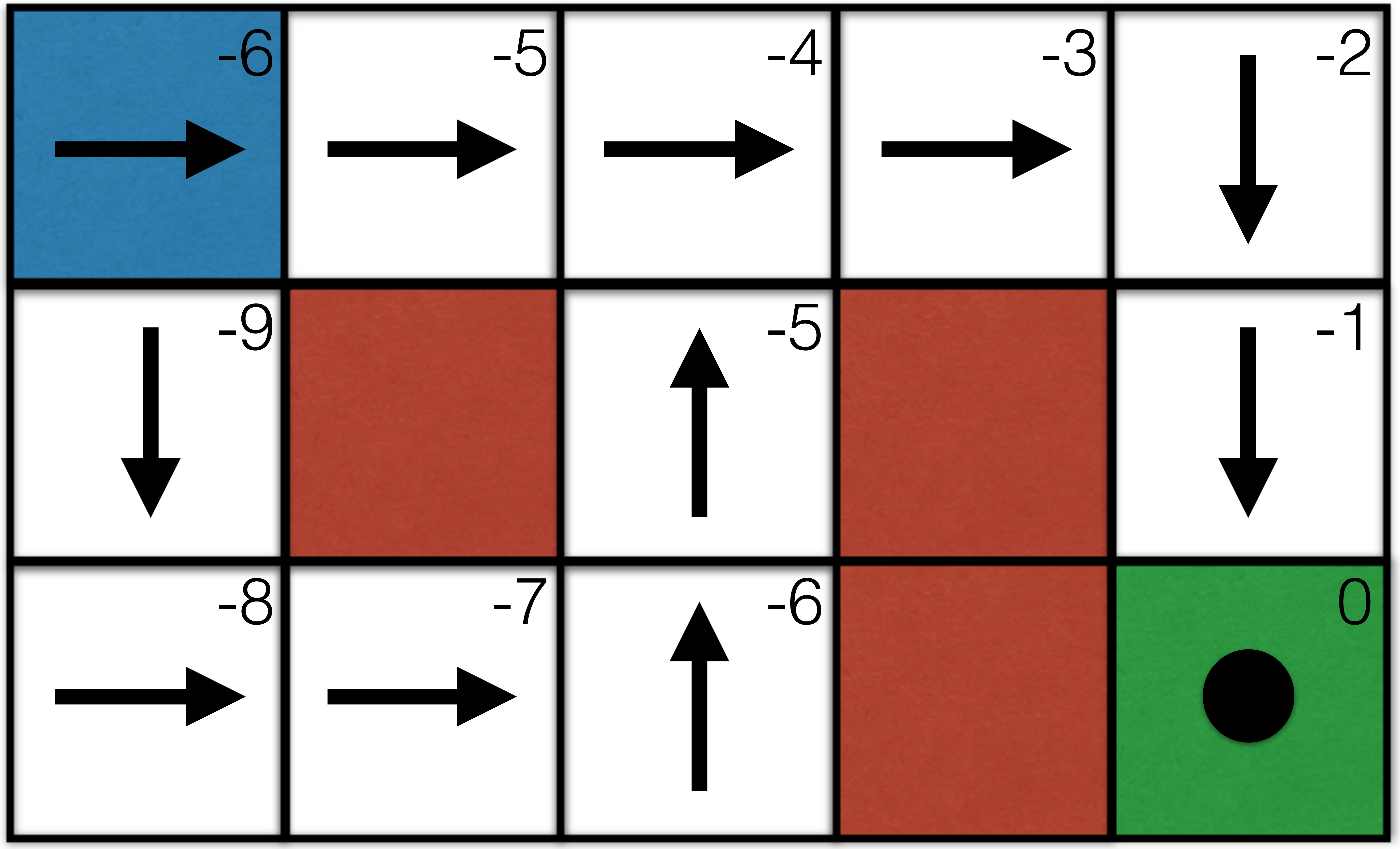
- Arrows denote the policy π.
- What are the values of the states, assuming no random movements?

Gridworld Value Function Example



- Arrows denote the policy π.
- What are the values of the states, assuming no random movements?

Gridworld Value Function Example



- Arrows denote the policy π.
- What are the values of the states, assuming no random movements?

Policy Evaluation

Policy Evaluation

- What we just did was *policy evaluation*, determining the value of states given our policy π. Simply put, we figured out how good our policy is.

Policy Evaluation

- What we just did was *policy evaluation*, determining the value of states given our policy π. Simply put, we figured out how good our policy is.

- But remember, what we are really interested in is the optimal policy $\pi*$! How do we find this?

Policy Evaluation

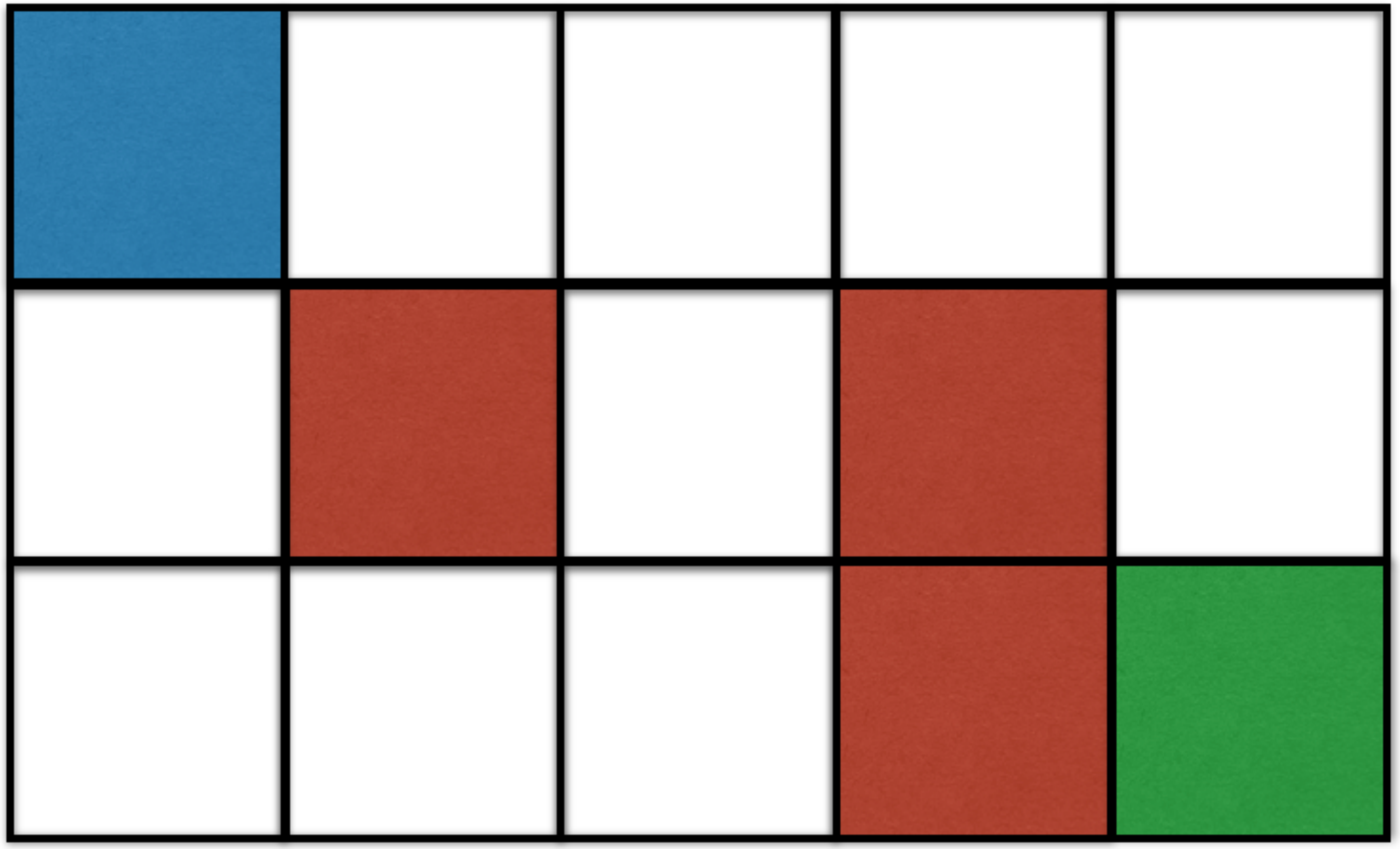
- What we just did was *policy evaluation*, determining the value of states given our policy π. Simply put, we figured out how good our policy is.
- But remember, what we are really interested in is the optimal policy π^*! How do we find this?
- We need one more step - *policy iteration*.

Policy Evaluation

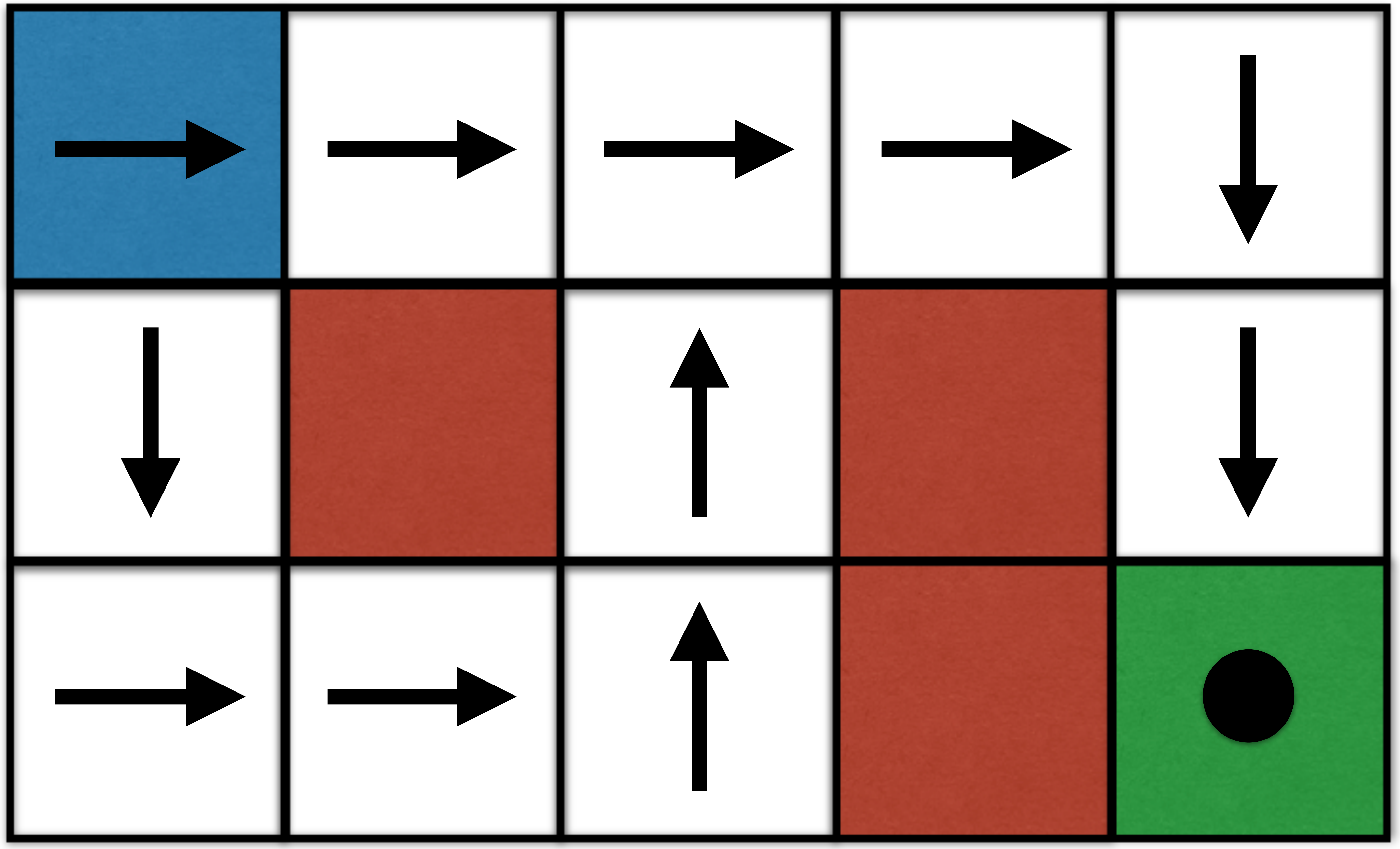
- What we just did was *policy evaluation*, determining the value of states given our policy π. Simply put, we figured out how good our policy is.
- But remember, what we are really interested in is the optimal policy π*! How do we find this?
- We need one more step - *policy iteration*.

Gridworld Policy Iteration Example

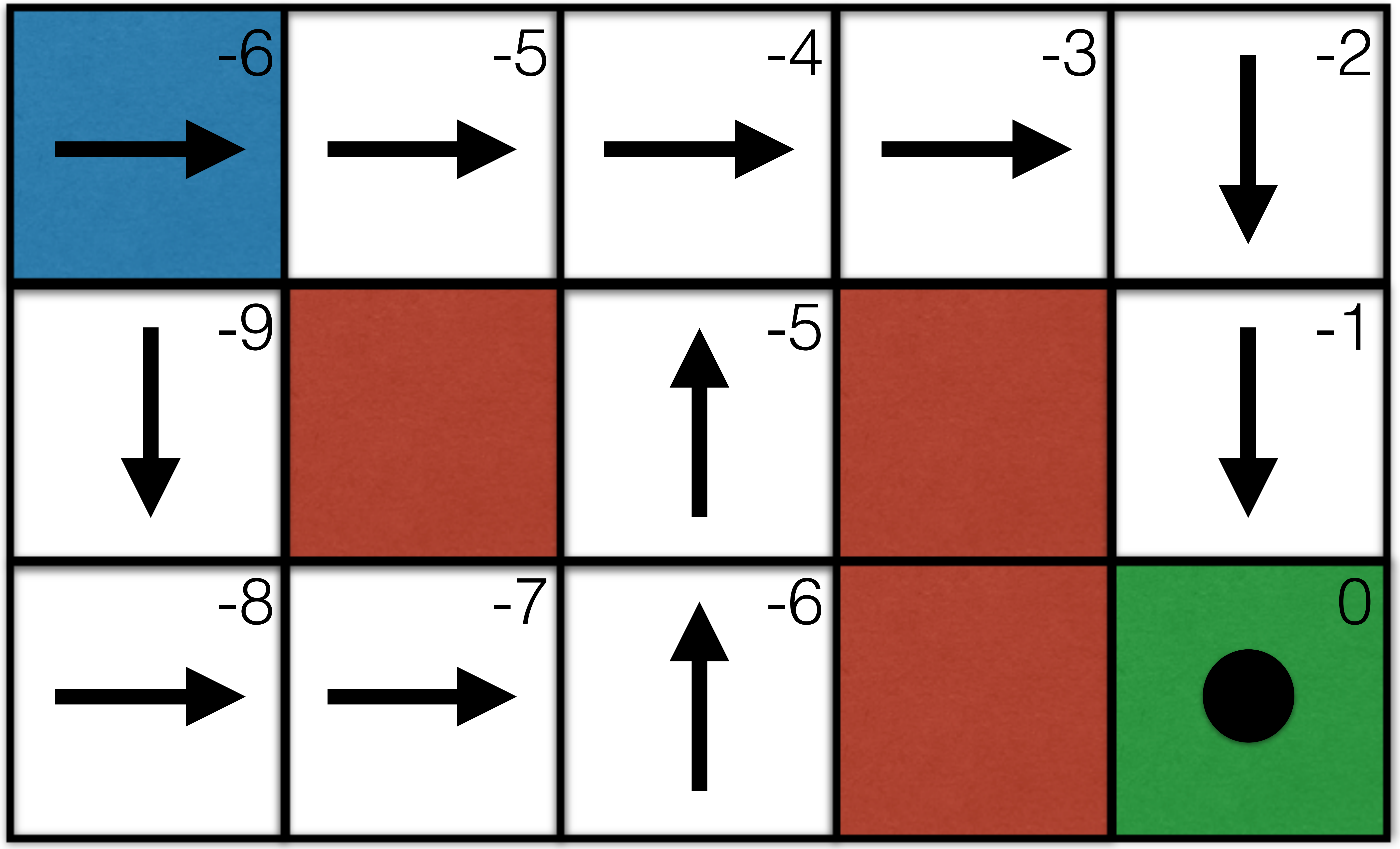
Gridworld Policy Iteration Example



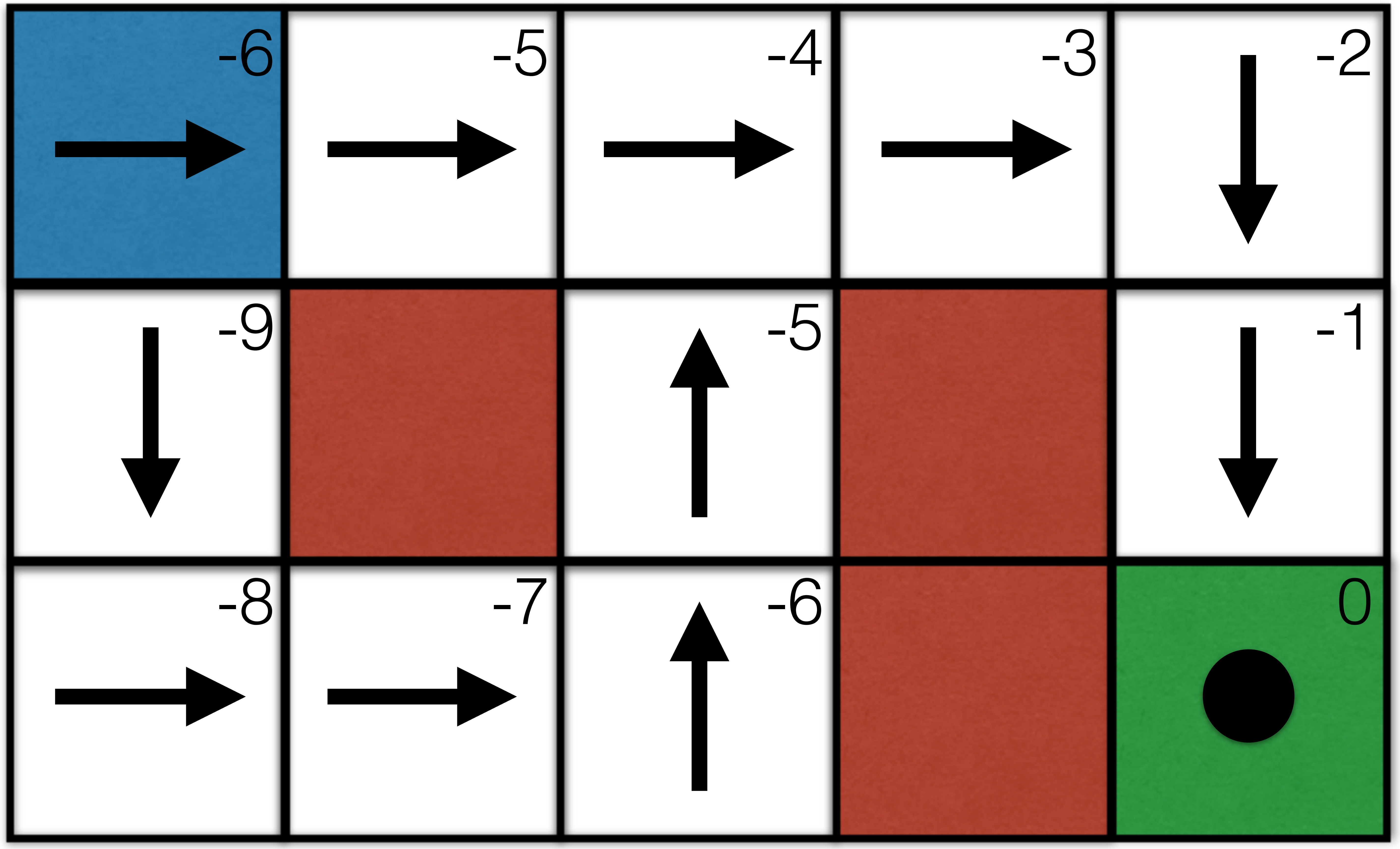
Gridworld Policy Iteration Example



Gridworld Policy Iteration Example

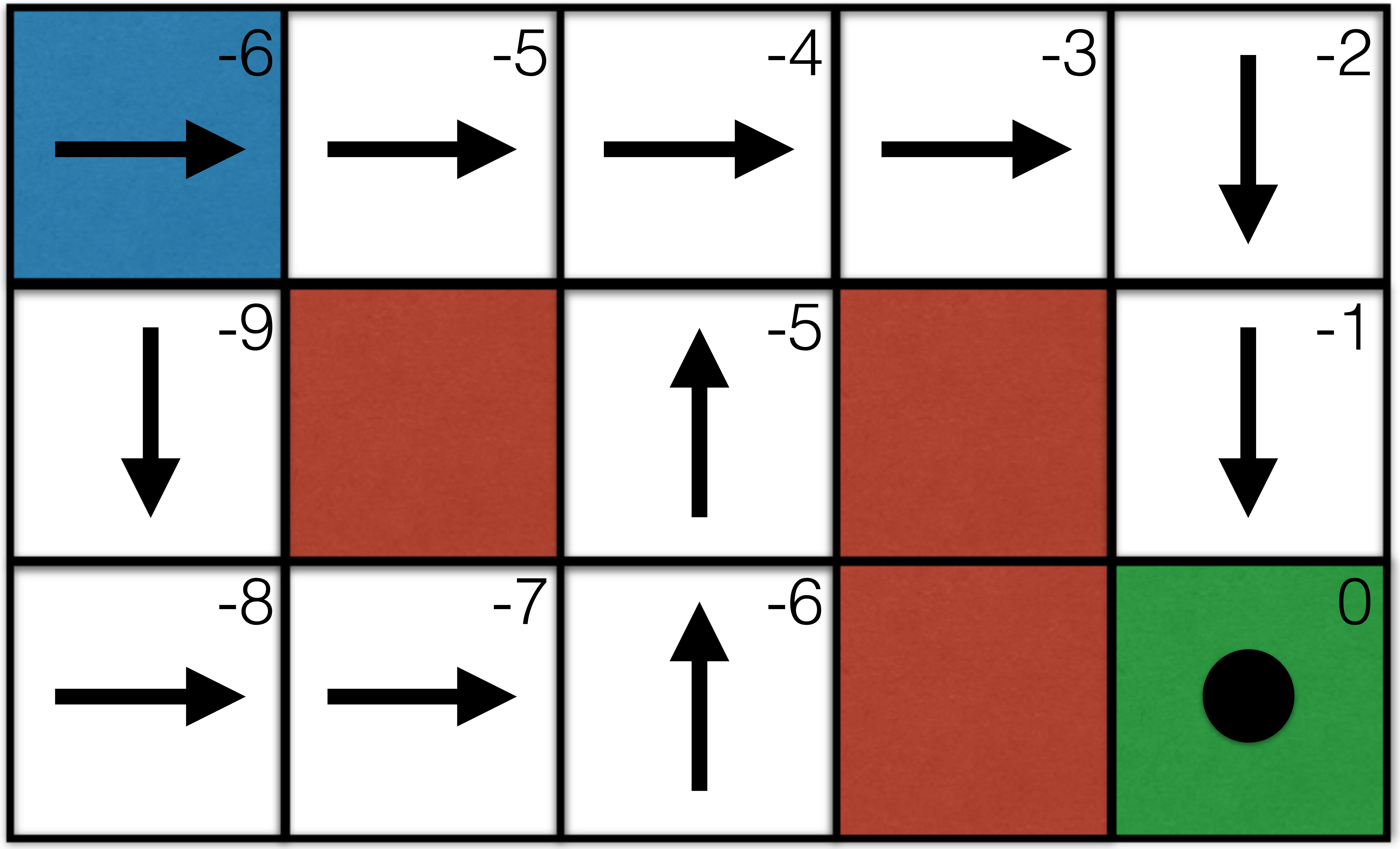


Gridworld Policy Iteration Example



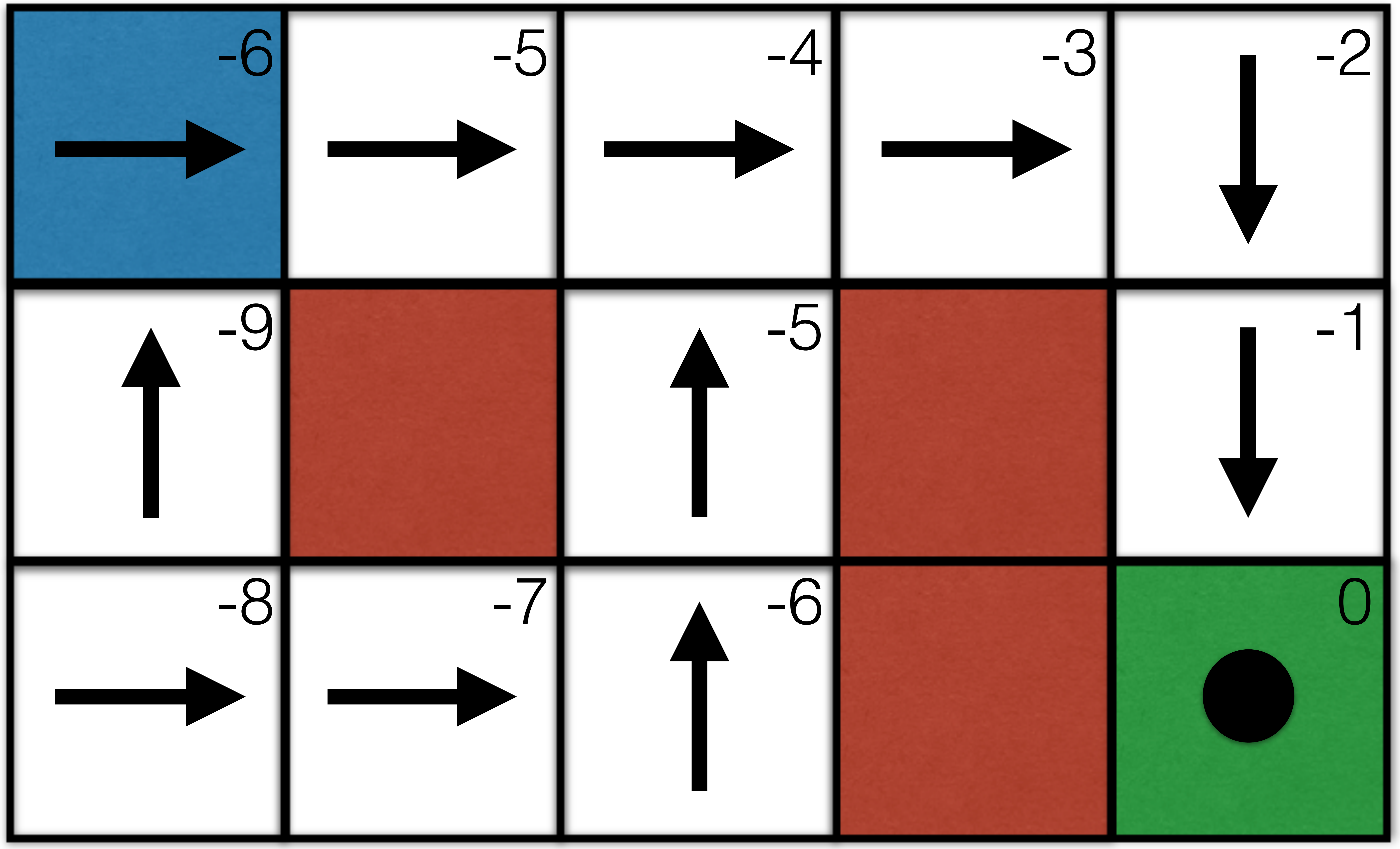
- Arrows denote the policy.

Gridworld Policy Iteration Example



- Arrows denote the policy.
- Based on the value function, which action of the current policy should we change?

Gridworld Policy Iteration Example



- Arrows denote the policy.
- Based on the value function, which action of the current policy should we change?

Policy Iteration

Policy Iteration

- Now that we know V(s), we improve our policy π to a new policy π' as follows:

Policy Iteration

- Now that we know V(s), we improve our policy π to a new policy π’ as follows:
  - For every state s, π’ picks the action that leads to the next state s’ with the highest value.

Policy Iteration

- Now that we know V(s), we improve our policy π to a new policy π' as follows:
  - For every state s, π' picks the action that leads to the next state s' with the highest value.

$$\pi'(s) = \arg\max_a \sum_{s'} P(s,a,s') V^{\pi}(s')$$

Policy Iteration

- Now that we know V(s), we improve our policy π to a new policy π' as follows:
  - For every state s, π' picks the action that leads to the next state s' with the highest value.

$$\pi'(s) = \arg\max_a \sum_{s'} P(s,a,s') V^{\pi}(s')$$

- This is called *policy iteration*.

Policy Iteration

- Now that we know V(s), we improve our policy π to a new policy π' as follows:
  - For every state s, π' picks the action that leads to the next state s' with the highest value.

$$\pi'(s) = \arg\max_a \sum_{s'} P(s, a, s') V^{\pi}(s')$$

- This is called *policy iteration*.

```
def new_policy(s):
    return max(actions,
        key=lambda a: sum([P(s, a, n_s) * V(n_s) for n_s in states]))
```

Policy Iteration

- Now that we know V(s), we improve our policy π to a new policy π' as follows:
  - For every state s, π' picks the action that leads to the next state s' with the highest value.

$$\pi'(s) = \arg\max_a \sum_{s'} P(s, a, s') V^{\pi}(s')$$

- This is called *policy iteration*.

```
def new_policy(s):
    return max(actions,
        key=lambda a: sum([P(s, a, n_s) * V(n_s) for n_s in states]))
```

Optimal Policy

Optimal Policy

- So, to find the optimal policy:

Optimal Policy

- So, to find the optimal policy:
    - Initialize some policy π.

Optimal Policy

- So, to find the optimal policy:
  - Initialize some policy π.
  - Repeat:

Optimal Policy

- So, to find the optimal policy:
  - Initialize some policy π.
  - Repeat:
    - Determine V(s) using *policy evaluation*.

Optimal Policy

- So, to find the optimal policy:
  - Initialize some policy π.
  - Repeat:
    - Determine V(s) using *policy evaluation*.

$$V^{\pi}(s) = R(s) + \sum_{s'} P(s, \pi(s), s') V^{\pi}(s')$$

Optimal Policy

- So, to find the optimal policy:
  - Initialize some policy π.
  - Repeat:
    - Determine V(s) using *policy evaluation*.

$$V^{\pi}(s) = R(s) + \sum_{s'} P(s, \pi(s), s') V^{\pi}(s')$$

    - Find a better policy π' using *policy iteration*.

Optimal Policy

- So, to find the optimal policy:
  - Initialize some policy π.
  - Repeat:
    - Determine V(s) using *policy evaluation*.

$$V^{\pi}(s) = R(s) + \sum_{s'} P(s, \pi(s), s') V^{\pi}(s')$$

    - Find a better policy π' using *policy iteration*.

$$\pi'(s) = \arg\max_{a} \sum_{s'} P(s, a, s') V^{\pi}(s')$$

Optimal Policy

- So, to find the optimal policy:
  - Initialize some policy π.
  - Repeat:
    - Determine V(s) using *policy evaluation*.

$$V^\pi(s) = R(s) + \sum_{s'} P(s, \pi(s), s') V^\pi(s')$$

    - Find a better policy π' using *policy iteration*.

$$\pi'(s) = \arg\max_a \sum_{s'} P(s, a, s') V^\pi(s')$$

    - If π == π', return π.

Optimal Policy

- So, to find the optimal policy:
  - Initialize some policy π.
  - Repeat:
    - Determine V(s) using *policy evaluation*.

$$V^{\pi}(s) = R(s) + \sum_{s'} P(s, \pi(s), s') V^{\pi}(s')$$

    - Find a better policy π' using *policy iteration*.

$$\pi'(s) = \arg\max_{a} \sum_{s'} P(s, a, s') V^{\pi}(s')$$

    - If π == π', return π.
    - Otherwise, set π equal to π'.

Optimal Policy

- So, to find the optimal policy:
  - Initialize some policy π.
  - Repeat:
    - Determine V(s) using *policy evaluation*.

$$V^\pi(s) = R(s) + \sum_{s'} P(s, \pi(s), s') V^\pi(s')$$

    - Find a better policy π' using *policy iteration*.

$$\pi'(s) = \arg\max_a \sum_{s'} P(s, a, s') V^\pi(s')$$

    - If $\pi == \pi'$, return π.
    - Otherwise, set π equal to π'.

- We can prove that this π we return is optimal, i.e. $\pi == \pi^*$!
  We won't do the math, though.

Optimal Policy

- So, to find the optimal policy:
  - Initialize some policy π.
  - Repeat:
    - Determine V(s) using *policy evaluation*.

$$V^{\pi}(s) = R(s) + \sum_{s'} P(s, \pi(s), s') V^{\pi}(s')$$

    - Find a better policy π’ using *policy iteration*.

$$\pi'(s) = \arg\max_{a} \sum_{s'} P(s, a, s') V^{\pi}(s')$$

    - If π == π’, return π.
    - Otherwise, set π equal to π’.

- We can prove that this π we return is optimal, i.e. π == π*!
  We won’t do the math, though.

Terminology So Far

Terminology So Far

- Reward function (R): how good is a state? (*short-term*)

Terminology So Far

- Reward function (R): how good is a state? (*short-term*)
- Value function (V): how good is a state? (*long-term*)

Terminology So Far

- Reward function (R): how good is a state? (*short-term*)

- Value function (V): how good is a state? (*long-term*)
  - Measures long-term expected reward.

Terminology So Far

- Reward function (R): how good is a state? (*short-term*)
- Value function (V): how good is a state? (*long-term*)
  - Measures long-term expected reward.
  - Depends on the current policy π.

Terminology So Far

- Reward function (R): how good is a state? (*short-term*)

- Value function (V): how good is a state? (*long-term*)
  - Measures long-term expected reward.
  - Depends on the current policy π.

- Policy evaluation:

Terminology So Far

- Reward function (R): how good is a state? (*short-term*)
- Value function (V): how good is a state? (*long-term*)
  - Measures long-term expected reward.
  - Depends on the current policy π.
- Policy evaluation:
  - Evaluating our current policy π to get a value function V.

Terminology So Far

- Reward function (R): how good is a state? (*short-term*)

- Value function (V): how good is a state? (*long-term*)
  - Measures long-term expected reward.
  - Depends on the current policy π.

- Policy evaluation:
  - Evaluating our current policy π to get a value function V.

$$V^\pi(s) = R(s) + \sum_{s'} P(s, \pi(s), s') V^\pi(s')$$

Terminology So Far

- Reward function (R): how good is a state? (*short-term*)

- Value function (V): how good is a state? (*long-term*)
  - Measures long-term expected reward.
  - Depends on the current policy π.

- Policy evaluation:
  - Evaluating our current policy π to get a value function V.

$$V^\pi(s) = R(s) + \sum_{s'} P(s, \pi(s), s') V^\pi(s')$$

- Policy iteration:

Terminology So Far

- Reward function (R): how good is a state? (*short-term*)

- Value function (V): how good is a state? (*long-term*)
  - Measures long-term expected reward.
  - Depends on the current policy π.

- Policy evaluation:
  - Evaluating our current policy π to get a value function V.

$$V^\pi(s) = R(s) + \sum_{s'} P(s, \pi(s), s') V^\pi(s')$$

- Policy iteration:
  - Using our value function V to get a better policy π'.

Terminology So Far

- Reward function (R): how good is a state? (*short-term*)

- Value function (V): how good is a state? (*long-term*)
  - Measures long-term expected reward.
  - Depends on the current policy π.

- Policy evaluation:
  - Evaluating our current policy π to get a value function V.

$$V^\pi(s) = R(s) + \sum_{s'} P(s, \pi(s), s') V^\pi(s')$$

- Policy iteration:
  - Using our value function V to get a better policy π'.

$$\pi'(s) = \arg\max_a \sum_{s'} P(s, a, s') V^\pi(s')$$

Terminology So Far

- Reward function (R): how good is a state? (*short-term*)

- Value function (V): how good is a state? (*long-term*)
  - Measures long-term expected reward.
  - Depends on the current policy π.

- Policy evaluation:
  - Evaluating our current policy π to get a value function V.

$$V^\pi(s) = R(s) + \sum_{s'} P(s, \pi(s), s')V^\pi(s')$$

- Policy iteration:
  - Using our value function V to get a better policy π'.

$$\pi'(s) = \arg\max_a \sum_{s'} P(s, a, s')V^\pi(s')$$

- Basically every RL algorithm is a combination of policy evaluation and policy iteration! Let's take a closer look at two such algorithms.

Terminology So Far

- Reward function (R): how good is a state? (*short-term*)

- Value function (V): how good is a state? (*long-term*)
  - Measures long-term expected reward.
  - Depends on the current policy π.

- Policy evaluation:
  - Evaluating our current policy π to get a value function V.

$$V^{\pi}(s) = R(s) + \sum_{s'} P(s, \pi(s), s') V^{\pi}(s')$$

- Policy iteration:
  - Using our value function V to get a better policy π'.

$$\pi'(s) = \arg\max_{a} \sum_{s'} P(s, a, s') V^{\pi}(s')$$

- Basically every RL algorithm is a combination of policy evaluation and policy iteration! Let's take a closer look at two such algorithms.

Value Iteration

Value Iteration

- Value iteration is an algorithm that *combines* the policy evaluation and policy iteration steps into one single step.

Value Iteration

- Value iteration is an algorithm that *combines* the policy evaluation and policy iteration steps into one single step.

- Repeat:

Value Iteration

- Value iteration is an algorithm that *combines* the policy evaluation and policy iteration steps into one single step.

- Repeat:
  - For all states s, determine V(s), and set V(s) to its maximum possible value.

Value Iteration

- Value iteration is an algorithm that *combines* the policy evaluation and policy iteration steps into one single step.

- Repeat:
  - For all states s, determine V(s), and set V(s) to its maximum possible value.

$$V(s) = R(s) + \max_a \sum_{s'} P(s, a, s')V(s')$$

Value Iteration

- Value iteration is an algorithm that *combines* the policy evaluation and policy iteration steps into one single step.

- Repeat:
  - For all states s, determine V(s), and set V(s) to its maximum possible value.

$$V(s) = R(s) + \max_a \sum_{s'} P(s, a, s')V(s')$$

  - If V doesn't change, return the policy π that acts according to the maximum value of V.

Value Iteration

- Value iteration is an algorithm that *combines* the policy evaluation and policy iteration steps into one single step.

- Repeat:
  - For all states s, determine V(s), and set V(s) to its maximum possible value.

$$V(s) = R(s) + \max_a \sum_{s'} P(s, a, s')V(s')$$

  - If V doesn't change, return the policy π that acts according to the maximum value of V.

$$\pi(s) = \arg\max_a \sum_{s'} P(s, a, s')V(s')$$

Value Iteration

- Value iteration is an algorithm that *combines* the policy evaluation and policy iteration steps into one single step.

- Repeat:
  - For all states s, determine V(s), and set V(s) to its maximum possible value.

$$V(s) = R(s) + \max_a \sum_{s'} P(s, a, s')V(s')$$

  - If V doesn't change, return the policy π that acts according to the maximum value of V.

$$\pi(s) = \arg\max_a \sum_{s'} P(s, a, s')V(s')$$

- Again, we can show that this policy is optimal, i.e. π == π*! Again, let's not do the math.

Value Iteration (demo)

- Value iteration is an algorithm that *combines* the policy evaluation and policy iteration steps into one single step.

- Repeat:
  - For all states s, determine V(s), and set V(s) to its maximum possible value.

$$V(s) = R(s) + \max_a \sum_{s'} P(s,a,s')V(s')$$

  - If V doesn't change, return the policy π that acts according to the maximum value of V.

$$\pi(s) = \arg\max_a \sum_{s'} P(s,a,s')V(s')$$

- Again, we can show that this policy is optimal, i.e. π == π*! Again, let's not do the math.

Value Iteration

(demo)

- Value iteration is an algorithm that *combines* the policy evaluation and policy iteration steps into one single step.

- Repeat:
  - For all states s, determine V(s), and set V(s) to its maximum possible value.

$$V(s) = R(s) + \max_a \sum_{s'} P(s, a, s')V(s')$$

  - If V doesn't change, return the policy π that acts according to the maximum value of V.

$$\pi(s) = \arg\max_a \sum_{s'} P(s, a, s')V(s')$$

- Again, we can show that this policy is optimal, i.e. π == π*! Again, let's not do the math.

Questions and Limitations

Questions and Limitations

- What if there are way too many states and actions to try?

Questions and Limitations

- What if there are way too many states and actions to try?
  - We have to find a way to only look at a *subset* of states and actions, and we also need to reasonably *approximate* their values.

Questions and Limitations

- What if there are way too many states and actions to try?
  - We have to find a way to only look at a *subset* of states and actions, and we also need to reasonably *approximate* their values.

- What if we don’t know how the environment works?

Questions and Limitations

- What if there are way too many states and actions to try?
  - We have to find a way to only look at a *subset* of states and actions, and we also need to reasonably *approximate* their values.

- What if we don’t know how the environment works?
  - It’s reasonable to think that the agent doesn’t completely understand what the next possible states are from taking an action.

Questions and Limitations

- What if there are way too many states and actions to try?
  - We have to find a way to only look at a *subset* of states and actions, and we also need to reasonably *approximate* their values.

- What if we don’t know how the environment works?
  - It’s reasonable to think that the agent doesn’t completely understand what the next possible states are from taking an action.
  - In this case, we have to try different things in order to figure out our environment - this is called *exploration*.

Questions and Limitations

- What if there are way too many states and actions to try?
  - We have to find a way to only look at a *subset* of states and actions, and we also need to reasonably *approximate* their values.

- What if we don't know how the environment works?
  - It's reasonable to think that the agent doesn't completely understand what the next possible states are from taking an action.
  - In this case, we have to try different things in order to figure out our environment - this is called *exploration*.
  - Sometimes, we also want to just keep doing what we know is good - this is called *exploitation*.

Questions and Limitations

- What if there are way too many states and actions to try?
  - We have to find a way to only look at a *subset* of states and actions, and we also need to reasonably *approximate* their values.

- What if we don't know how the environment works?
  - It's reasonable to think that the agent doesn't completely understand what the next possible states are from taking an action.
  - In this case, we have to try different things in order to figure out our environment - this is called *exploration*.
  - Sometimes, we also want to just keep doing what we know is good - this is called *exploitation*.

Rollout-based Policy Iteration

Rollout-based Policy Iteration

- In RBPI, instead of full policy evaluation, we run simulations, or *rollouts*, to approximate our value function.

Rollout-based Policy Iteration

- In RBPI, instead of full policy evaluation, we run simulations, or *rollouts*, to approximate our value function.

- This addresses the limitations on the previous slide.

Rollout-based Policy Iteration

- In RBPI, instead of full policy evaluation, we run simulations, or *rollouts*, to approximate our value function.

- This addresses the limitations on the previous slide.
  - The *subset* of states that we look at are the states we encounter during our rollouts.

Rollout-based Policy Iteration

- In RBPI, instead of full policy evaluation, we run simulations, or *rollouts*, to approximate our value function.

- This addresses the limitations on the previous slide.
  - The *subset* of states that we look at are the states we encounter during our rollouts.
  - The *approximation* of the value of the states is by measuring our total reward over of course of our rollout.

Rollout-based Policy Iteration

- In RBPI, instead of full policy evaluation, we run simulations, or *rollouts*, to approximate our value function.

- This addresses the limitations on the previous slide.
  - The *subset* of states that we look at are the states we encounter during our rollouts.
  - The *approximation* of the value of the states is by measuring our total reward over of course of our rollout.
  - We balance *exploration* and *exploitation* by sometimes randomly selecting our action.

Rollout-based Policy Iteration (demo)

- In RBPI, instead of full policy evaluation, we run simulations, or *rollouts*, to approximate our value function.

- This addresses the limitations on the previous slide.
  - The *subset* of states that we look at are the states we encounter during our rollouts.
  - The *approximation* of the value of the states is by measuring our total reward over of course of our rollout.
  - We balance *exploration* and *exploitation* by sometimes randomly selecting our action.

Rollout-based Policy Iteration (demo)

- In RBPI, instead of full policy evaluation, we run simulations, or *rollouts*, to approximate our value function.

- This addresses the limitations on the previous slide.
  - The *subset* of states that we look at are the states we encounter during our rollouts.
  - The *approximation* of the value of the states is by measuring our total reward over of course of our rollout.
  - We balance *exploration* and *exploitation* by sometimes randomly selecting our action.

A Parting Thought

A Parting Thought

- We’ve talked about three of the projects in this class, and how machine learning applies to them.

A Parting Thought

- We’ve talked about three of the projects in this class, and how machine learning applies to them.
- What about the last one? How might machine learning apply to the Scheme project?

A Parting Thought

- We’ve talked about three of the projects in this class, and how machine learning applies to them.
- What about the last one? How might machine learning apply to the Scheme project?