Mini-Course 1, Module 4
Dynamic Programming

CMPUT 397
Fall 2020

Reminders: Sept 27, 2021

- Lab Session during class on Wednesday
- We imported your grades into eclass! Check them out email cmput365@ about any problems….like “I got zero for everything! What happened?”
- Any questions?

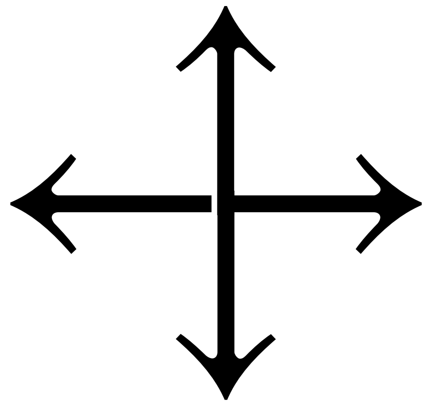
Review of C1M4
Dynamic Programming

Models and planning

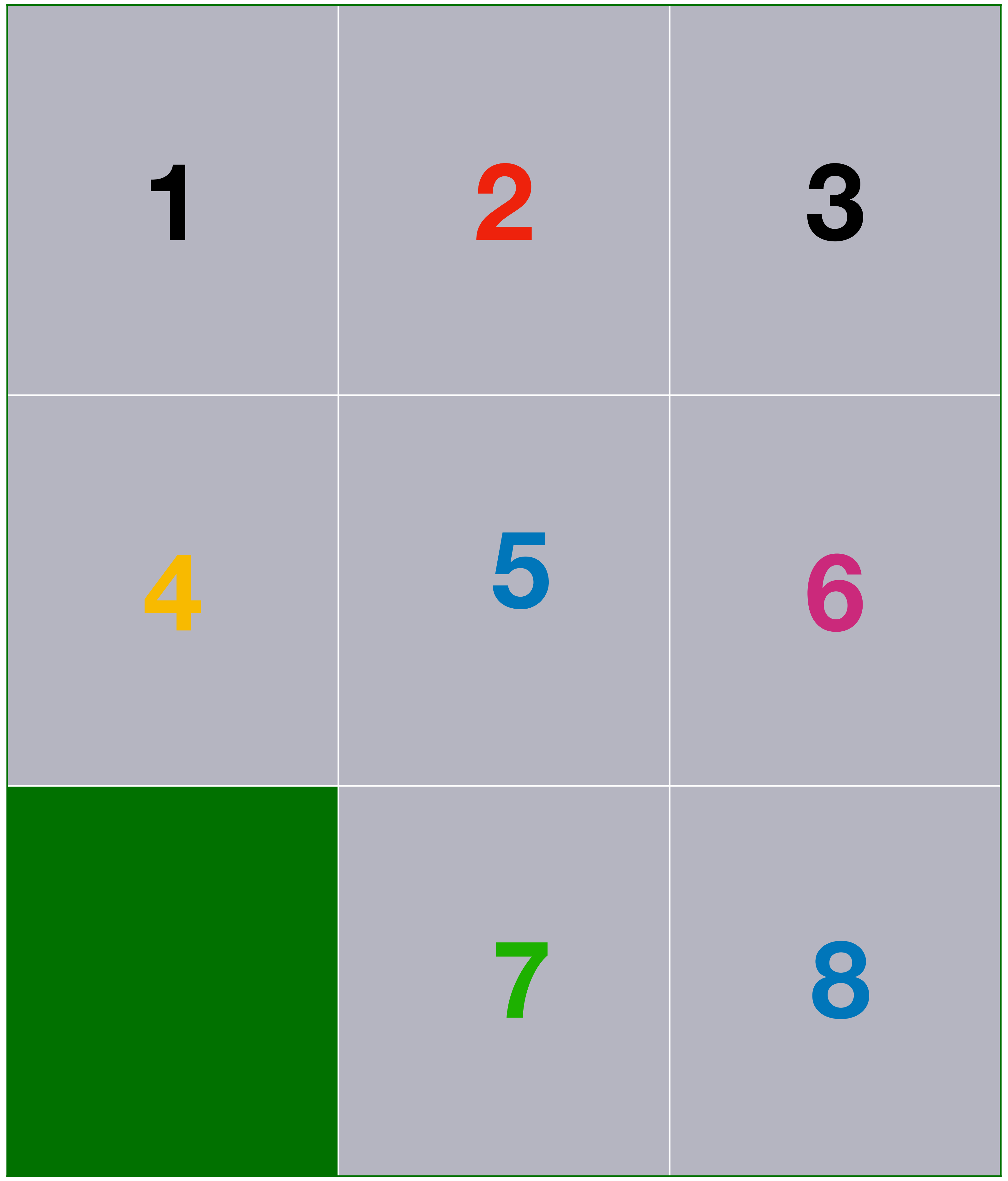
- How should we think about $p(s',r|s,a)$?
- What is dynamic programming and how is it different from what we did in bandits?

Example p(s',r|s,a)

- Consider state '**5**':
  - $p(2,0 \mid 5, \text{up}) = 1$
  - $p(6,0 \mid 5, \text{right}) = 1$
  - $p(7,0 \mid 5, \text{down}) = 1$
  - $p(4,0 \mid 5, \text{left}) = 1$
  - …

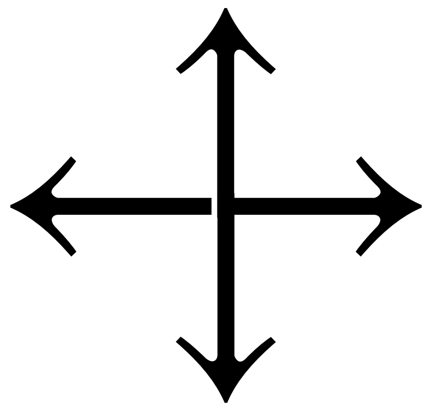


- Reward is zero on every transition

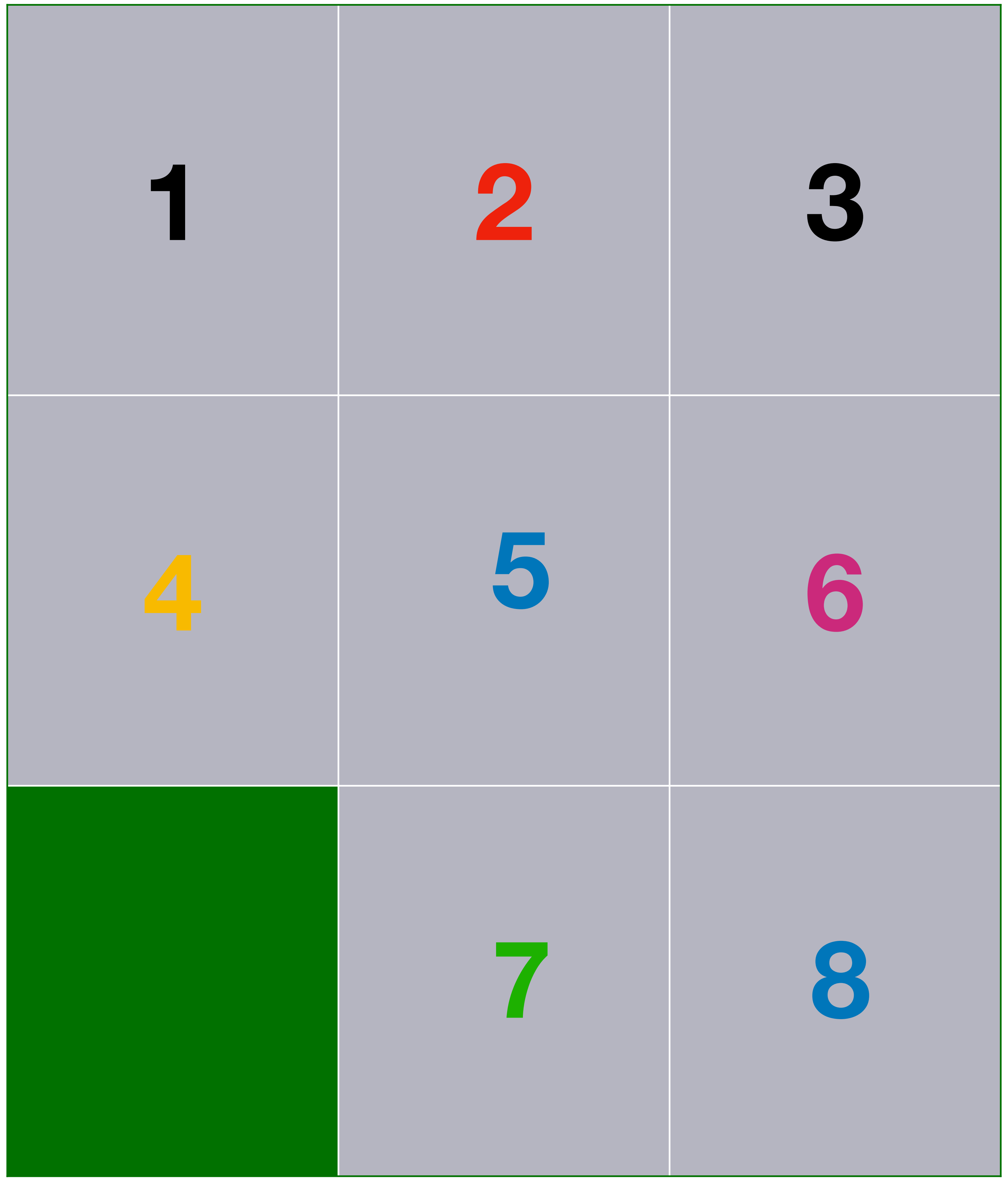


Example p(s',r|s,a)

- p(2,0 | 5, up) = 1
- p(6,0 | 5, right) = 1
- p(7,0 | 5, down) = 1
- p(4,0 | 5, left) = 1
- p(2,+10 | 5, up) = 0
- p(2,0 | 5, down) = 0
- p(2,0 | 5, left) = 0
- p(2,0 | 5, right) = 0
- p(1,0 | 5, up) = 0
- p(1,0 | 5, down) = 0
- p(1,0 | 5, left) = 0
- p(1,0 | 5, right) = 0
- …



- Reward is zero on every transition



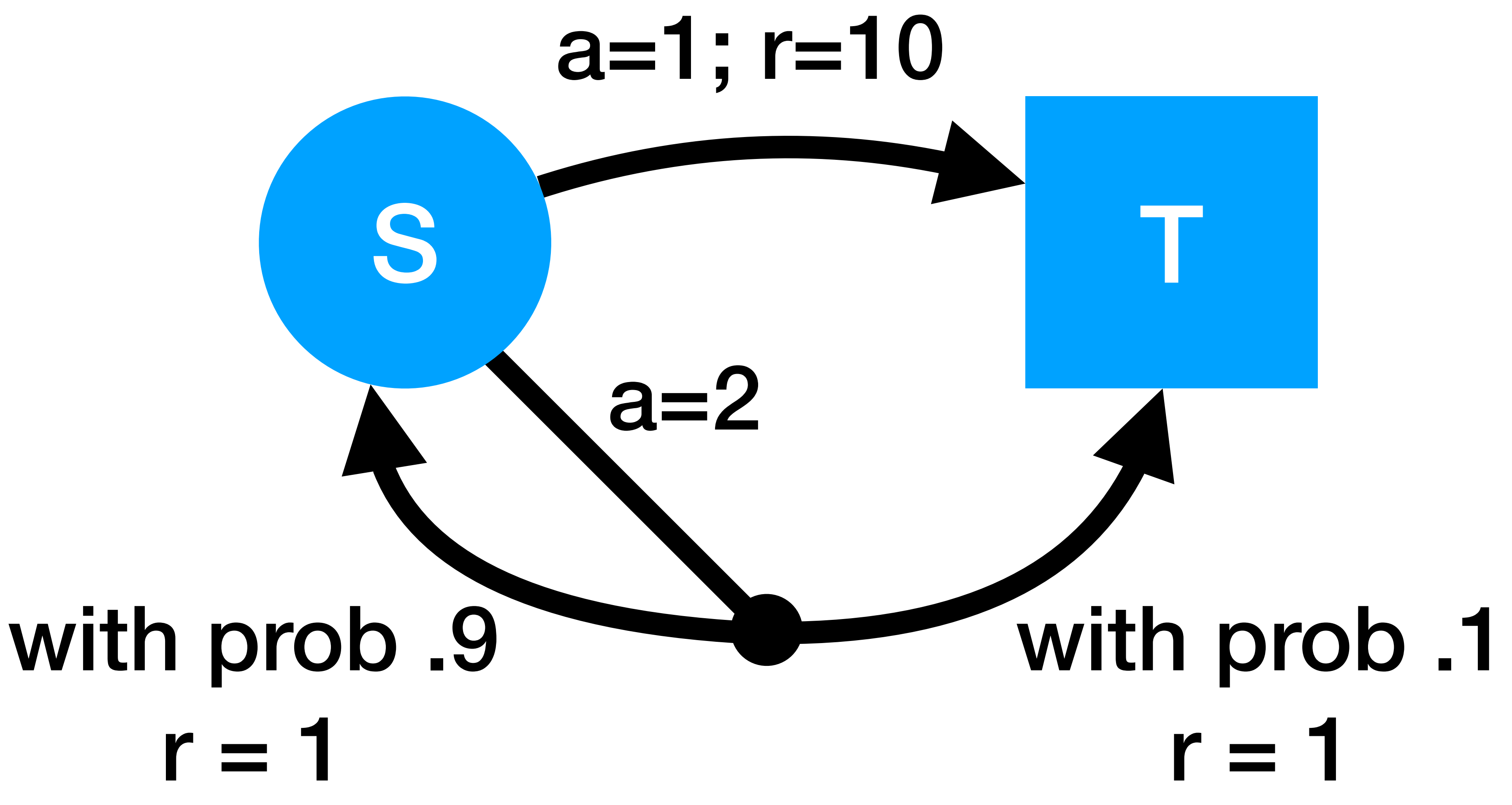
p tells us all the things that can and cannot happen in this MDP

Example p(s',r|s,a)

In this MDP the outcome of action 2 is stochastic: different possible next state and reward given action

- $p(T,10 \mid s, 1) = 1$
- $p(T,1 \mid s, 1) = 0$
- $p(s,1 \mid s, 1) = 0$
- $p(s,10 \mid s, 1) = 0$

- $p(T,1 \mid s, 2) = 0.1$
- $p(s,1 \mid s, 2) = .9$
- $p(T,10 \mid s, 2) = 0$
- $p(s,10 \mid s, 2) = 0$



- Set of possible rewards = {1,10}
- Set of possible actions = {1,2}
- Set of possible states = {s,T}

Atari p(s’,r|s,a)

Reward

- Let the state be the RAM state of the game console
- Let the actions be the joystick actions (discrete)
- Reward is change score

16 actions

Atari p(s’,r|s,a)

- Atari is deterministic
- Given the current RAM state **s** and the player’s action …
- The game engine:
  - Outputs a new state s’
  - And a change to the score
- There is literally a function:
  - p(s’,r|s,a) that is binary under the hood

16 actions

Reward based on score

Atari $p(s',r|s,a)$

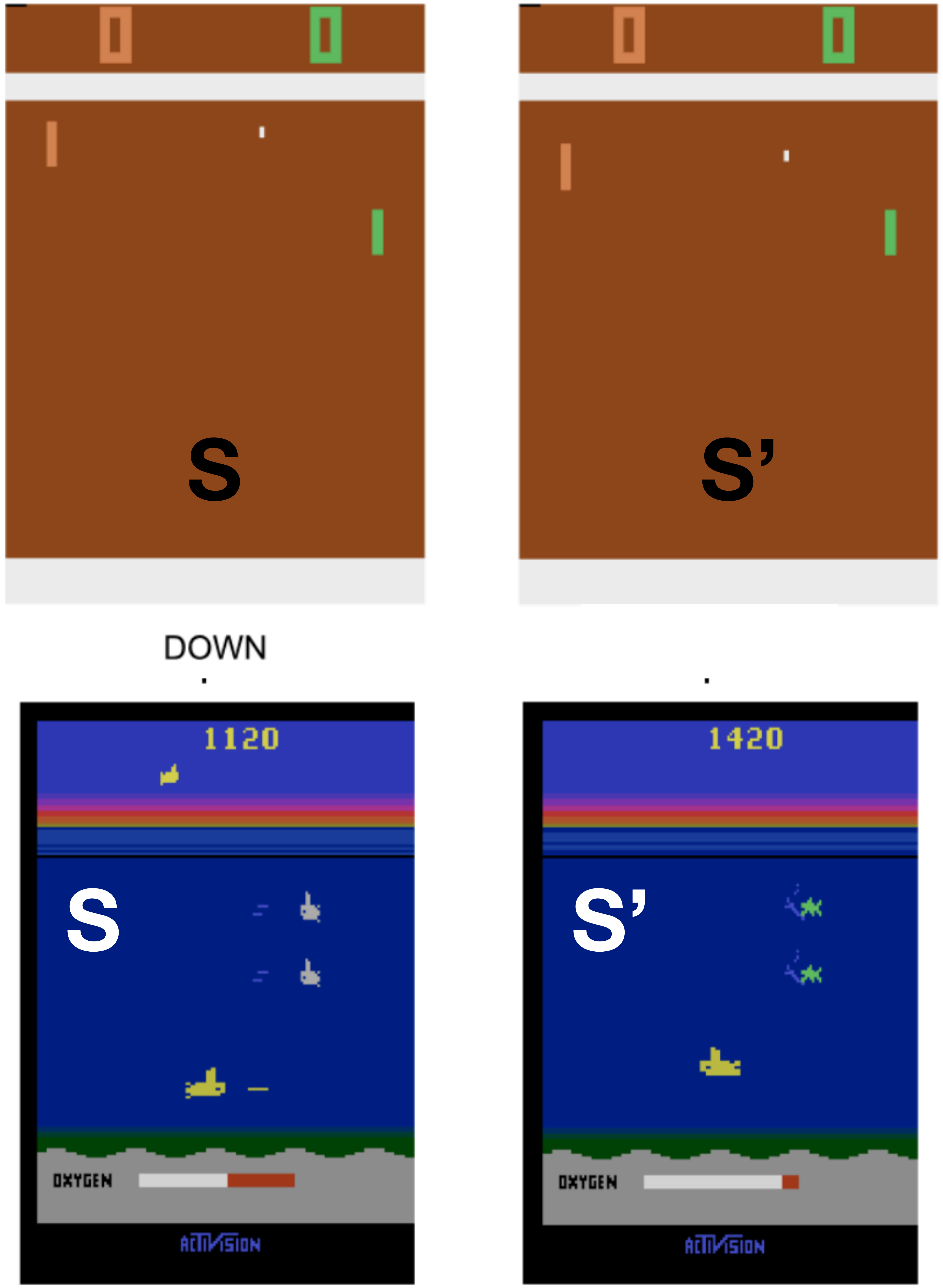
✅ $p(s',0 \mid s, \text{down}) = 1$

(paddle moves down, balls moves forward down, no score change)

❌ $p(s',300 \mid s, \text{downrightfire}) = 0.0$

(Boats gone, ship flipped, more oxygen, …)

Frame 49 Frame 50



DOWNRIGHTFIRE

Planning vs learning from interaction

- Imagine the universe consists of you, a chess board and pieces AND me (but I only play chess and I never talk, never respond to you)
  - The only thing you can do in life is play chess against me!!
- There are only two ways you could figure out how to beat me:
  - Play me over and over and figure out how the game works; figure out my play. **Trial and error learning** (like in a bandit)
  - -OR- If you had a **book** describing the rules & how I play (Adam is part of the environment)
    - You could sit there and **THINK** about how to beat me. You could **REASON** about the rules and how I play. You could **PLAN**

Planning vs learning from interaction

- $p(s', r|s, a)$ is like the **book** describing the rules and how I play
- You could use it to image different board configurations and how I would react to your moves
- You would be using the **book** to simulate playing against me:
  - You could imagine whole games in your mind
- Without ever picking up a chess piece or touching the board, you could figure out how to beat me. **Assuming the book was correct!**

Planning vs learning from interaction

- Without ever picking up a chess piece or touching the board, you could figure out how to beat me
    - THAT'S **PLANNING**!!!
    - Using a description of how the world works—$p(s',r|s,a)$—to figure out an **optimal policy**
    - *NO interacting with the world required*!!
    - We will assume access to the correct/perfect $p(s',r|s,a)$
        - Where does the model come from?
        - For now don't worry about it. It is there. We will come back to this question in CH8

Computing value functions and optimal policies using p(s’,r|s,a)

- All kinds of fun questions arise:
  - What should we compute? v_\pi, q_\pi, v*, \pi*
  - How should select states to imagine about? And in what order?
  - How much computation does it take to figure out \pi* using p?
    - How many imaginings do we need to do to figure out the optimal policy?
- This process of computing value functions and \pi* from p (with no interaction) is called **Dynamic Programming**

Video 1: Policy Evaluation vs. Control

- Introduce the two classic problems of RL: prediction and control. Classic assumptions of DP
- Goals:
  - Understand the distinction between policy **evaluation** and **control**
  - Explain the **setting** in which dynamic programming can be **applied**, as well as its limitations
- *What is the main limitation of DP?*

Video 2: Iterative Policy Evaluation

- How to turn Bellman equations into algorithms for **computing** value functions and policies
- Goals:
  - Outline the iterative policy evaluation algorithm for estimating state values for a given policy
  - Apply iterative policy evaluation to compute value functions, in an example MDP
- *How do we create a DP algorithm from the Bellman equation?*

Video 3: Policy Improvement

- **Key theoretical result** in RL and DP! How to make the policy better using the value function
- Goals:
  - Understand the **policy improvement theorem**; and how it can be used to construct improved policies
  - And use the value function for a policy to **produce a better policy**
- *Why are such theoretical results important? Aren't experiments enough?*

Video 4: Policy Iteration

- Our first control algorithm. Why sequencing evaluation and improvement works!
- Goals:
  - Outline the **policy iteration algorithm** for finding the optimal policy;
  - Understand **“the dance of policy and value”**, how policy iteration reaches the optimal policy by alternating between evaluating a policy and improving it
  - Apply policy iteration to compute optimal policies and optimal value functions
- *What are the two parts if the iterative policy evaluation algorithm?*

Video 5: Flexibility of the Policy Iteration Framework

- Generalized Policy Iteration: a general framework for control
- **Goals**:
  - Understand the framework of generalized policy iteration
  - Outline value iteration, an important special case of generalized policy iteration
  - Differentiate synchronous and asynchronous dynamic programming methods
- *Could we mix Dynamic Programming (planning) with interacting with the world?*

Video 6: Efficiency of Dynamic Programming

- DP is actually pretty good, compared to other approaches! What's the deal with **Bootstrapping**?
- **Goals**:
  - Describe Monte-Carlo sampling as an **alternative** method for learning a value function
  - Describe brute force search as an **alternative** method for finding an optimal policy; and
  - Understand the advantages of Dynamic programming and **bootstrapping** over these alternatives.
- *Where have we seen bootstrapping before?*

Key Terminology

- Policy evaluation
- Policy improvement
- Policy iteration
- Value iteration
- Generalized policy iteration

Quiz review

- https://www.coursera.org/learn/fundamentals-of-reinforcement-learning/quiz/5dph6/dynamic-programming/

More Definitions and Terminology

- “I'm confused about what v_k is, my interpretation is its the state-value function for an arbitrary policy. I don't believe that is correct though. What is v_k?”
  - —> It is our value estimate on the k-th step of Iterative Policy Evaluation

Difference between v and q

- “Why does the lower golf example (figure 3.3) which is supposed to be optimal have a -2 field over most of the green, where the above example with the putter has that area marked as only -1? Isn't q*() supposed to be optimal? There should be no areas where q*() has a worse result than v putt, right?”

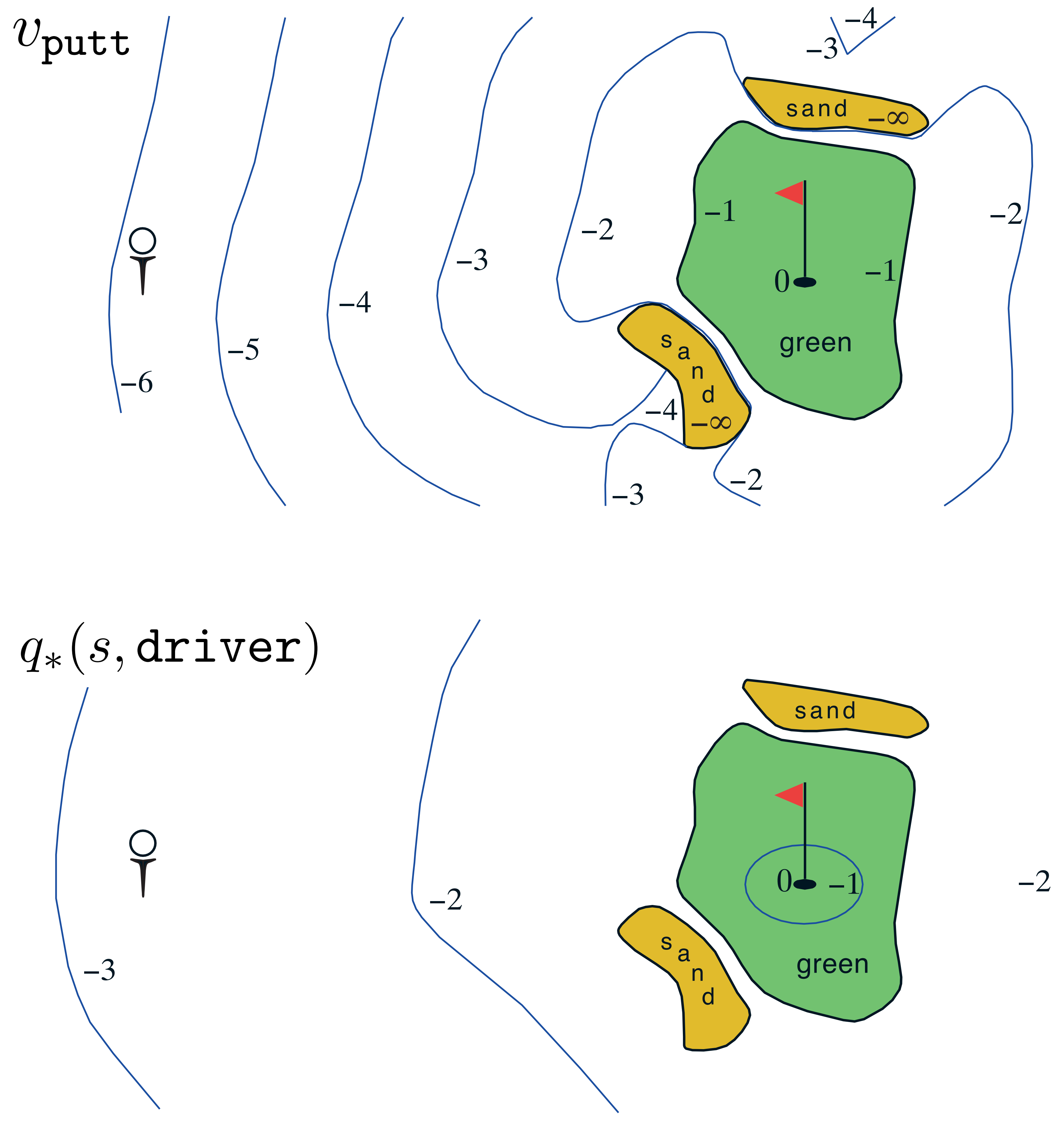


Figure 3.3: A golf example: the state-value function for putting (upper) and the optimal action-value function for using the driver (lower). ■

Additional Clarifications

- “is it possible to use DP in non-episodic models?” —> Yes
- “The Monte Carlo method which is quite famous is described to be a optimization of averages of the policy taken over a lot of instances. This seems to me a very unsophisticated method? So, why is such a method so widely used in RL?” —> Its actually not very widely used
- “How do you make sure the optimal solution found by value iteration is global maximum instead of local maximum?” —> we have not talked about having a (smooth) optimization surface that could have local maxima. Value iteration is guaranteed to converge to the optimal solution (the global max)