Dynamic Programming

CMPUT 655

Fall 2022

Ch4

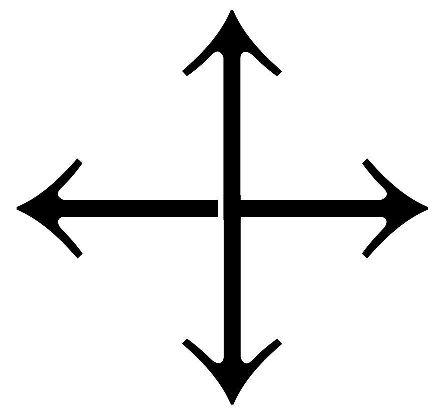
Also known as: what if we had THE model?

Models and planning

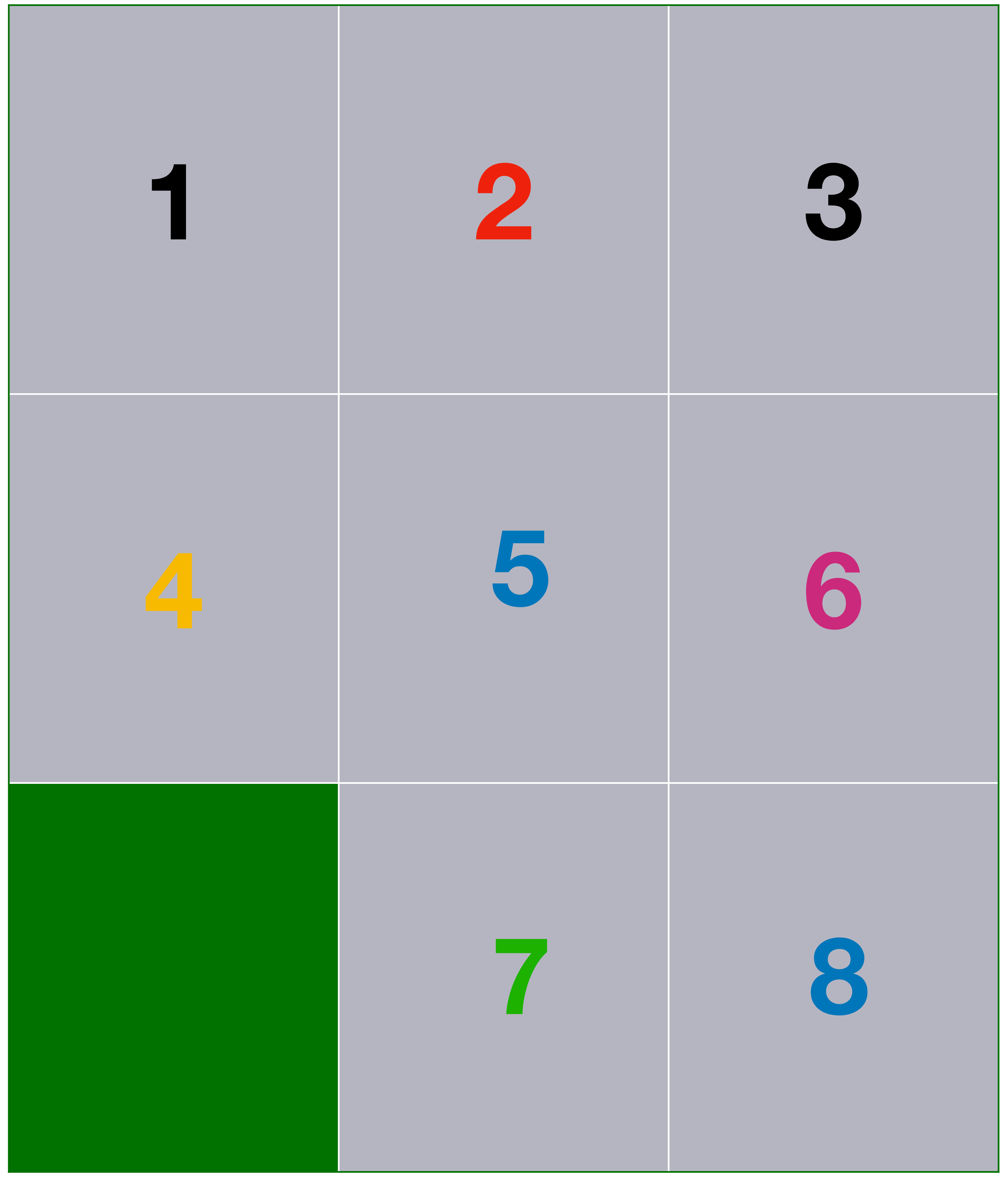
- How should we think about p(s’,r|s,a)?
- What is dynamic programming and how is it different from what we did in bandits?

Example p(s’,r|s,a)

- Consider state ‘**5**’:
  - p(2,0 | 5, up) = 1
  - p(6,0 | 5, right) = 1
  - p(7,0 | 5, down) = 1
  - p(4,0 | 5, left) = 1
  - …

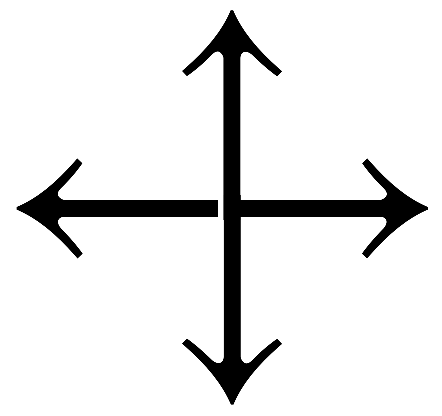


- Reward is zero on every transition

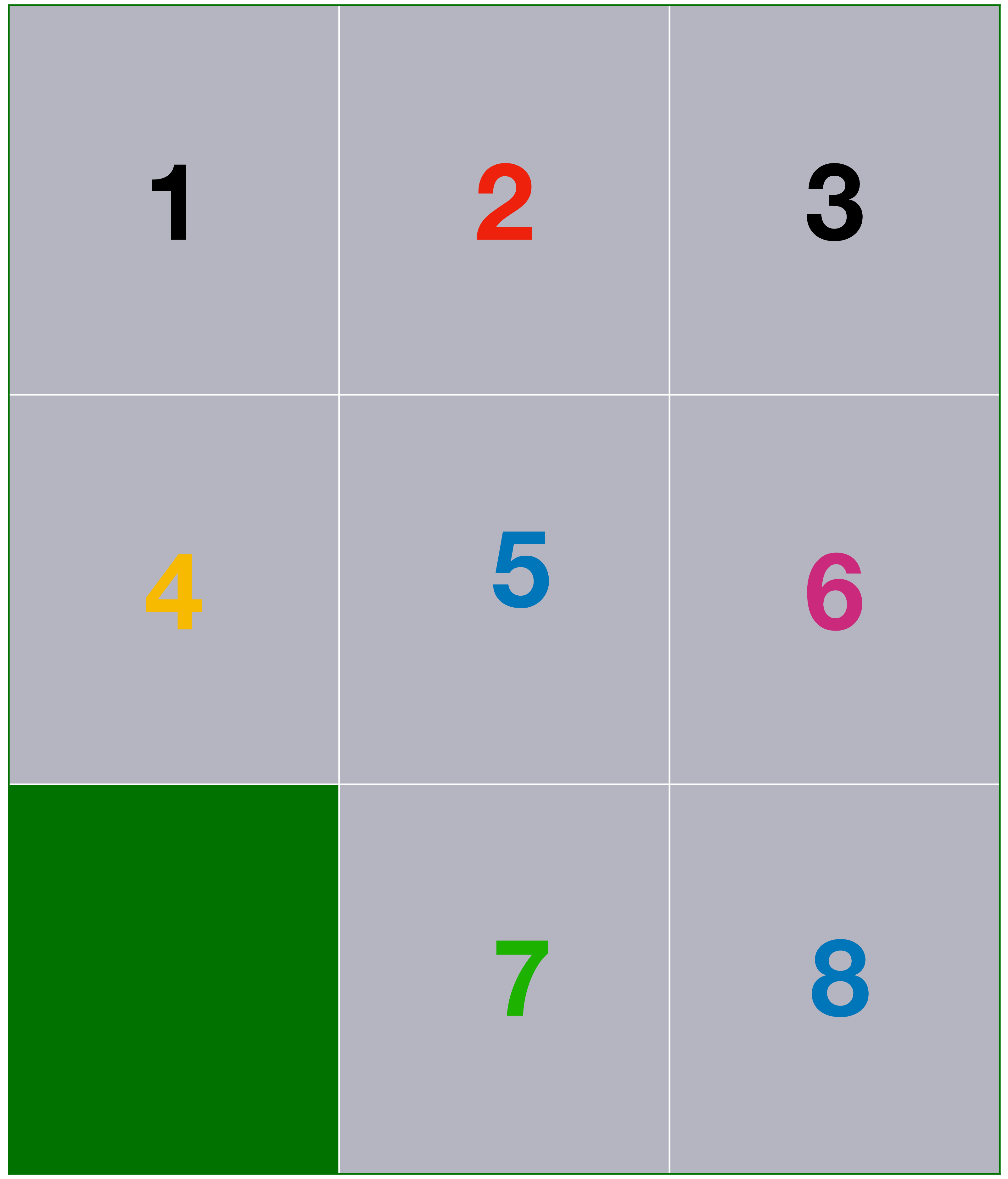


Example p(s',r|s,a)

- 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



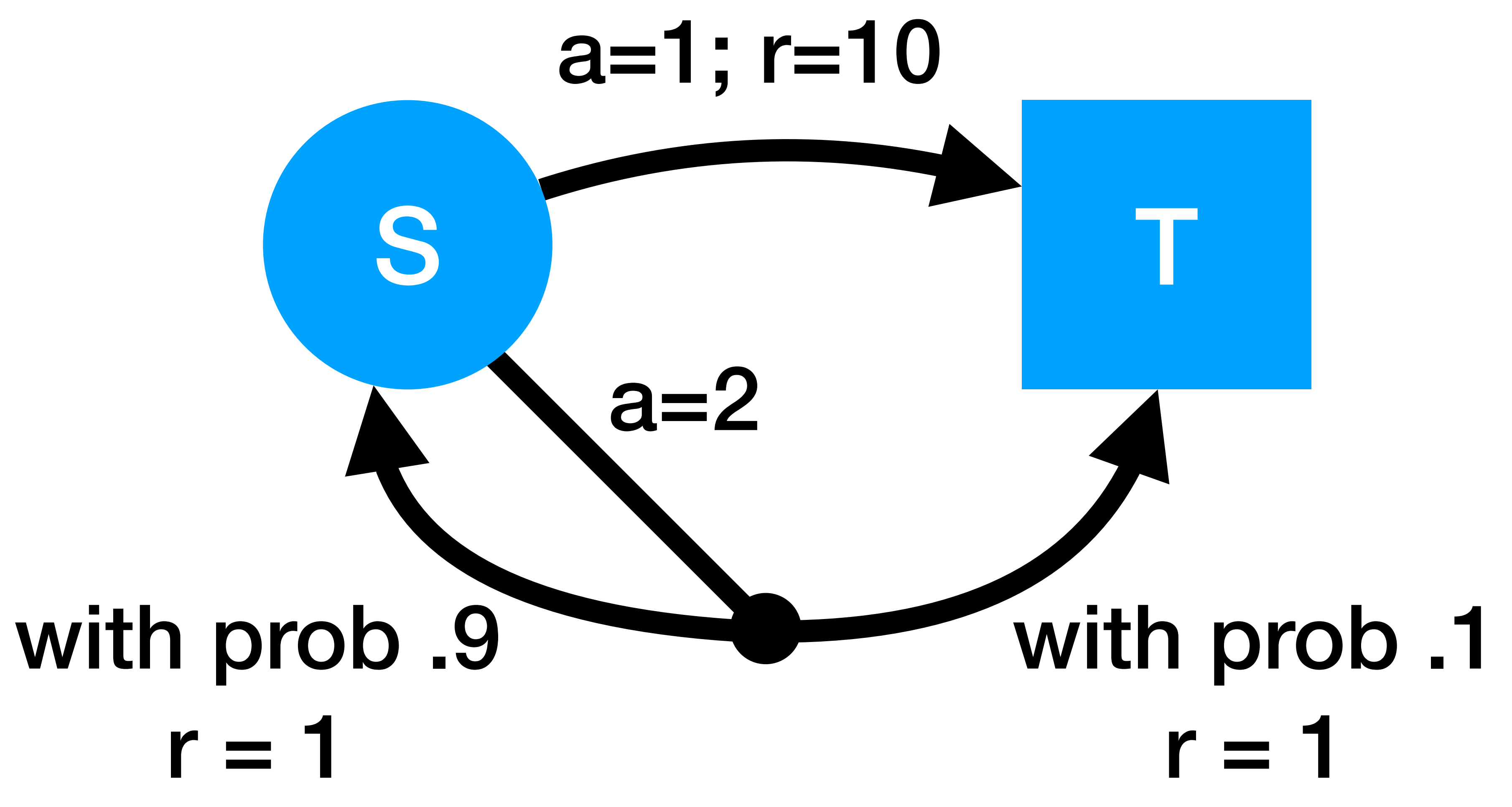
p tells us all the things that can and cannot happen in this MDP

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 | s, 1) = 1
- p(T,1 | s, 1) = 0
- p(s,1 | s, 1) = 0
- p(s,10 | s, 1) = 0

- p(T,1 | s, 2) = 0.1
- p(s,1 | s, 2) = .9
- p(T,10 | s, 2) = 0
- p(s,10 | s, 2) = 0

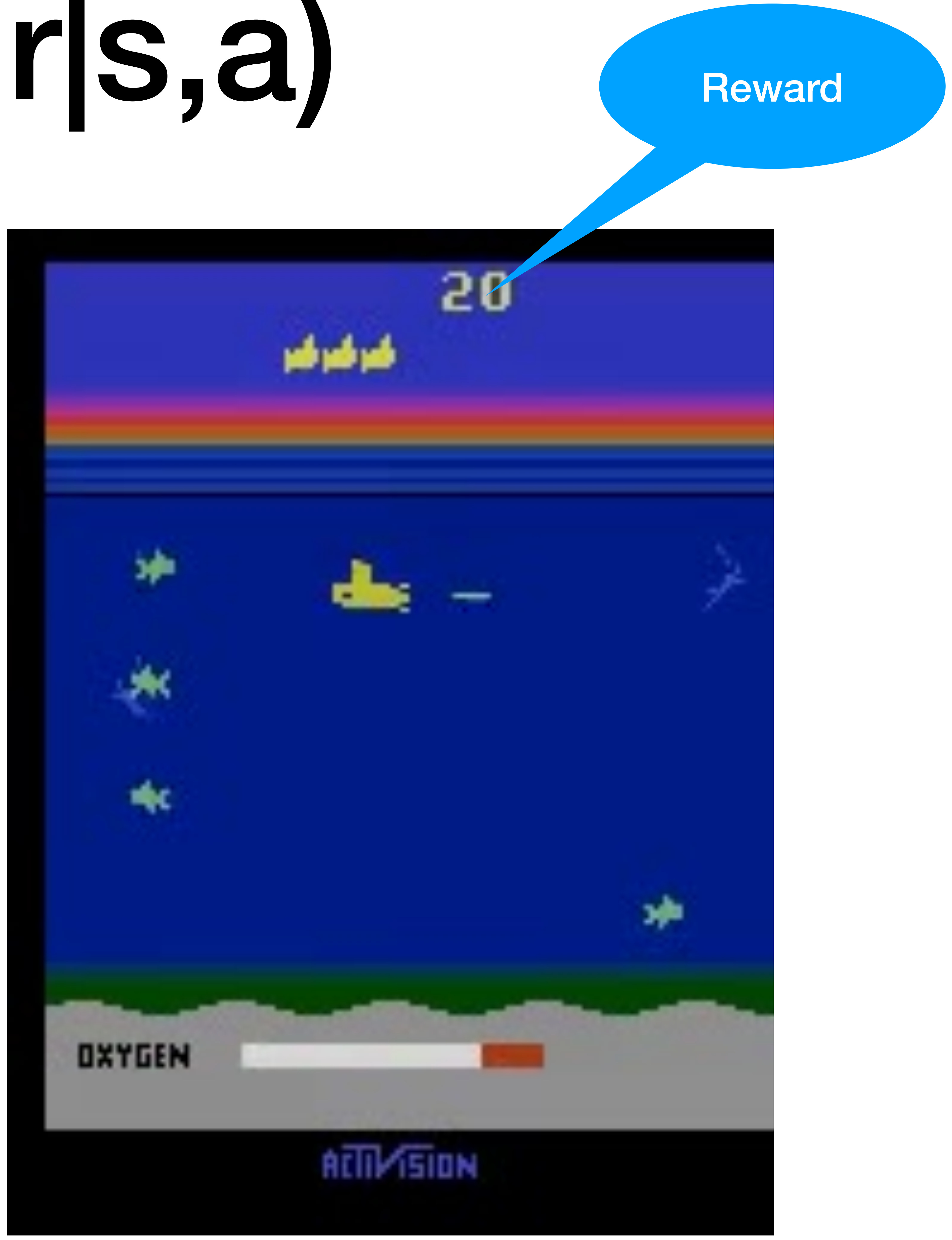


- Set of possible rewards = {1,10}
- Set of possible actions = {1,2}
- Set of possible states = {s,T}

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

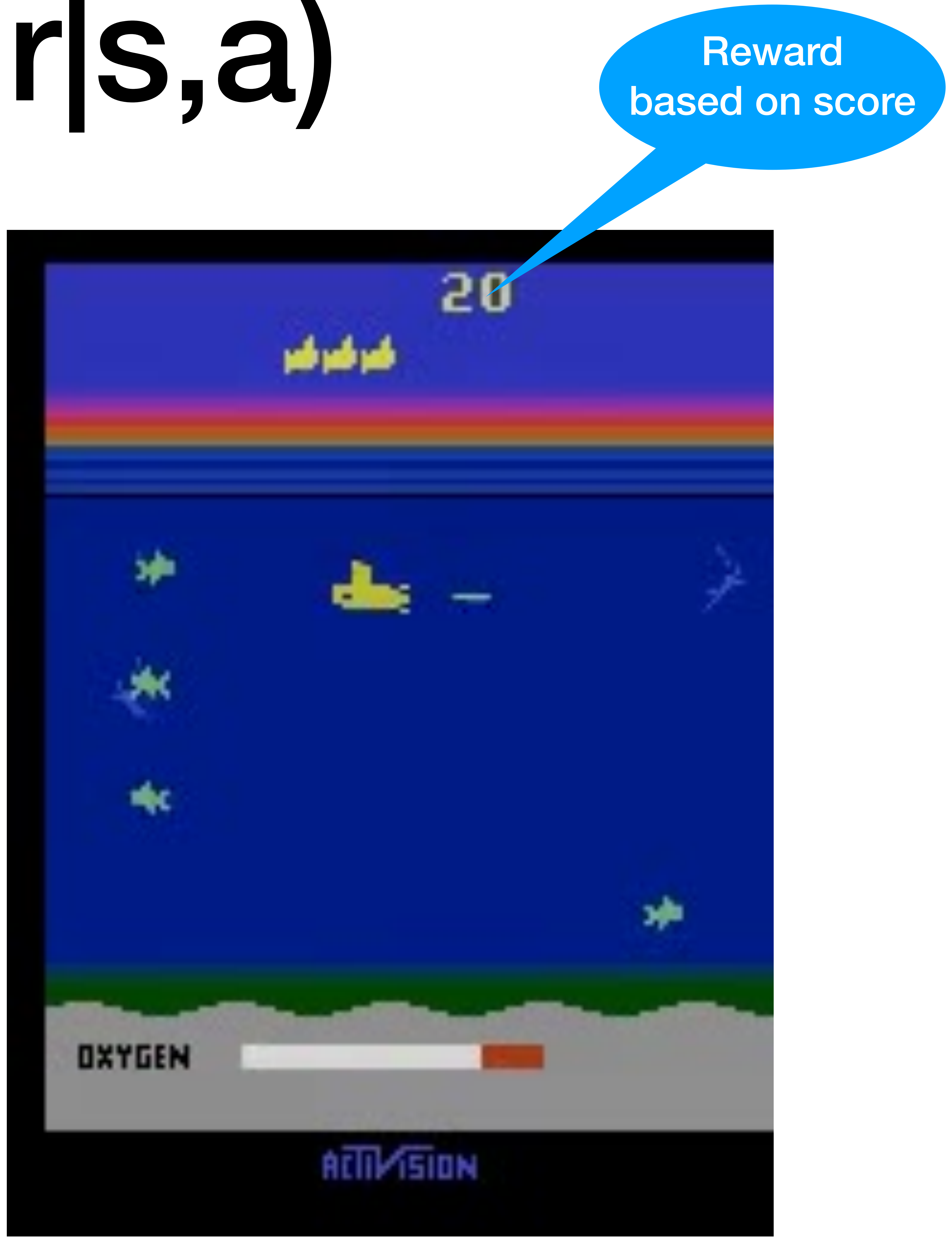
- Let the state be the RAM state of the game console
- Let the actions be the joystick actions (discrete)
- Reward is change score

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



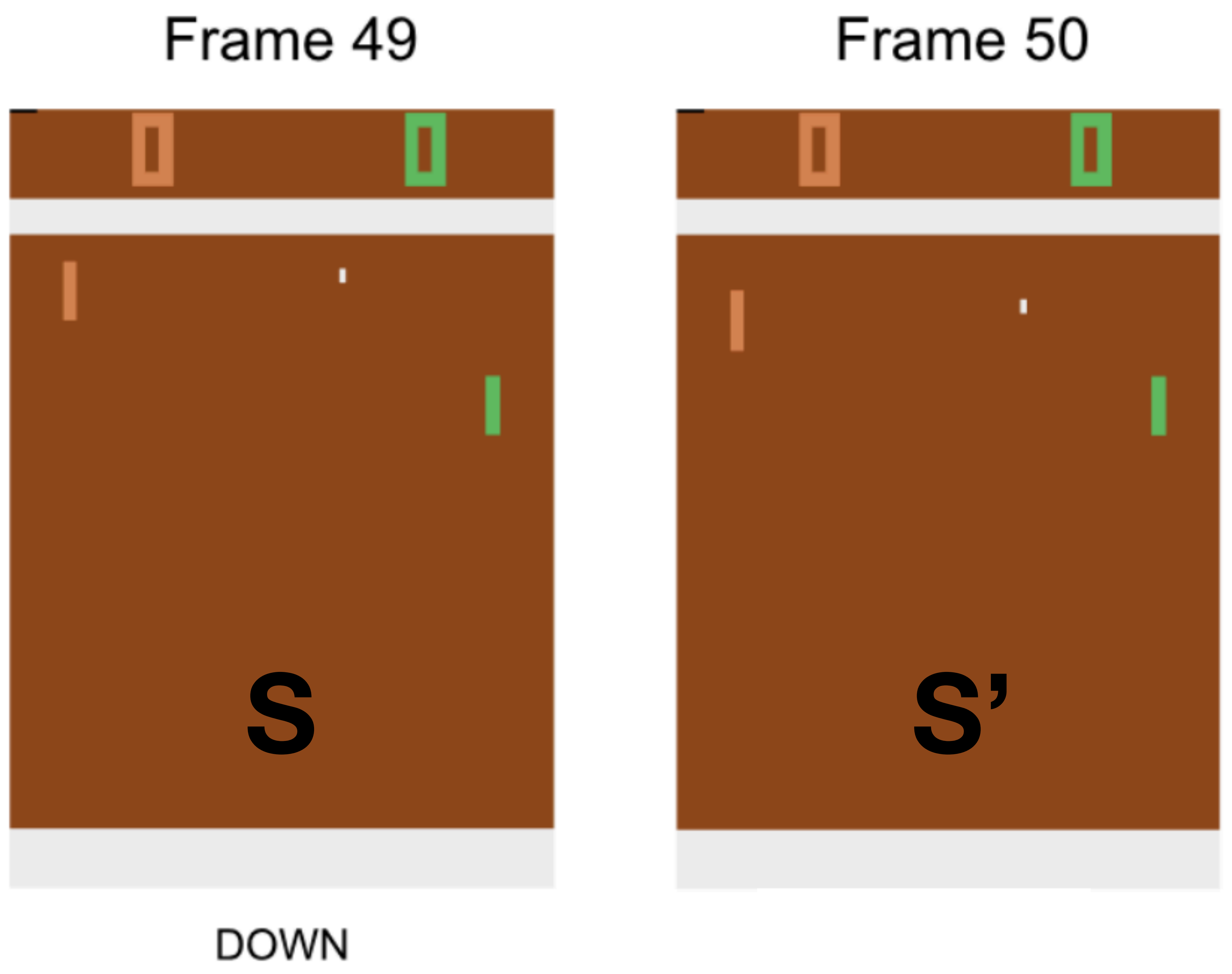
Atari p(s’,r|s,a)

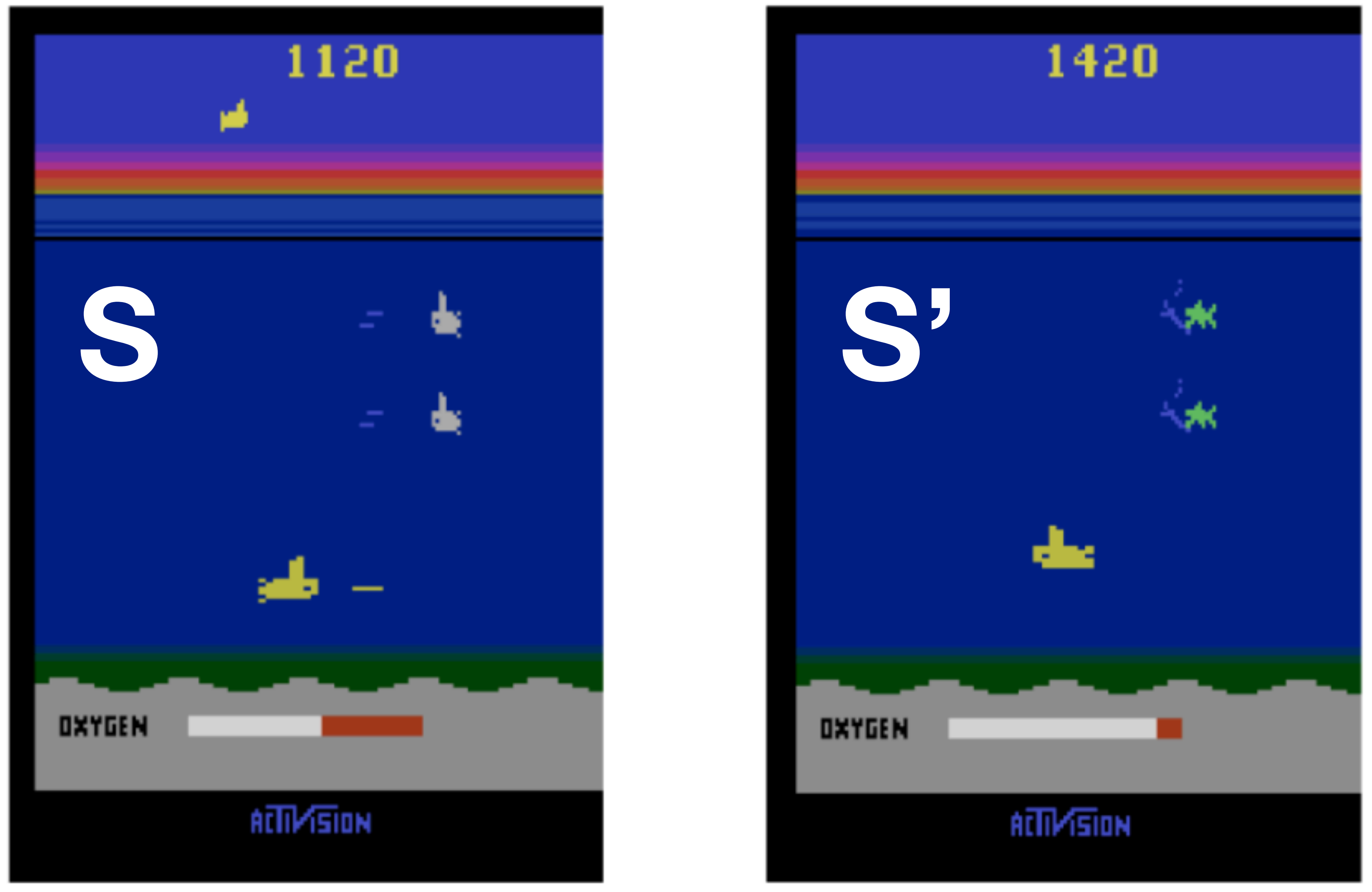
✅ p(s’,0 | s, down) = 1

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

❌ p(s’,300 | s, downrightfire) = 0.0

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





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)

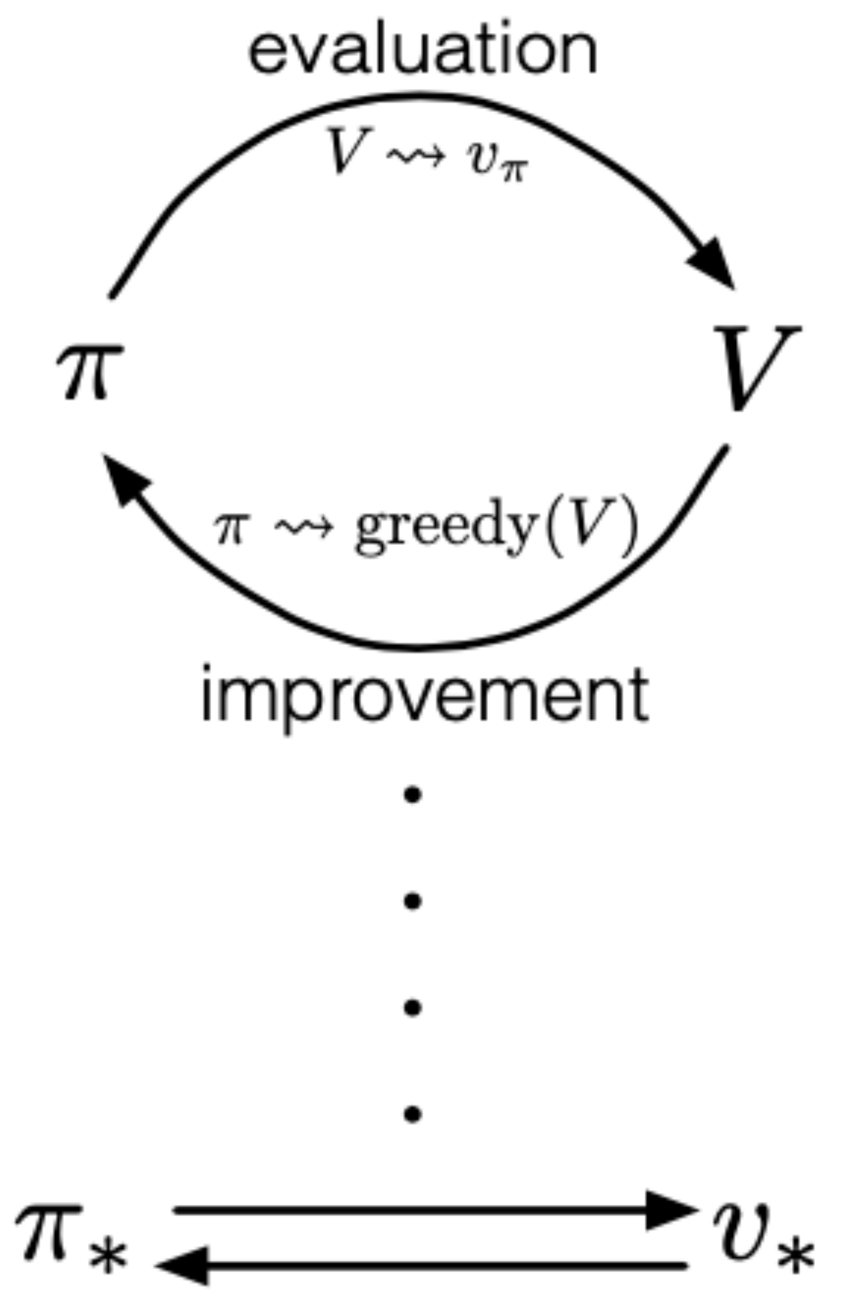
- All kinds of fun questions arise:
  - What should we compute? v_\pi, q_\pi, v*, \pi*
  - How should select states to imagine about? And in what order?
  - How much computation does it take to figure out \pi* using p?
    - How many imaginings do we need to do to figure out the optimal policy?
- This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

Things that don’t typically happen in classic DP algorithms

- **Visiting** states (**updating** a state is not the same as **visiting** a state)
- Exploration
- More terminology:
  - What is a sweep?
  - What is a full backup?

Discussion Posts

- What is the Generalized in Generalized Policy Iteration?
    - Is Iterative Policy an instance of GPI?
    - Is Value Iteration an instance of GPI?
    - Is Asynchronous DP and instance of GPI?
    - Is Q-learning an instances of GPI?


evaluation
$V \rightsquigarrow v_\pi$
π
V
$\pi \rightsquigarrow \mathrm{greedy}(V)$
improvement
π_*
v_*

How do we break the curse of dimensionality?

- Are DP methods efficient?
- How do we scale DP methods?
- Does scaling require approximation?
- Do people use DP in practice?
- How can we combine DP methods and methods that interact with the world?

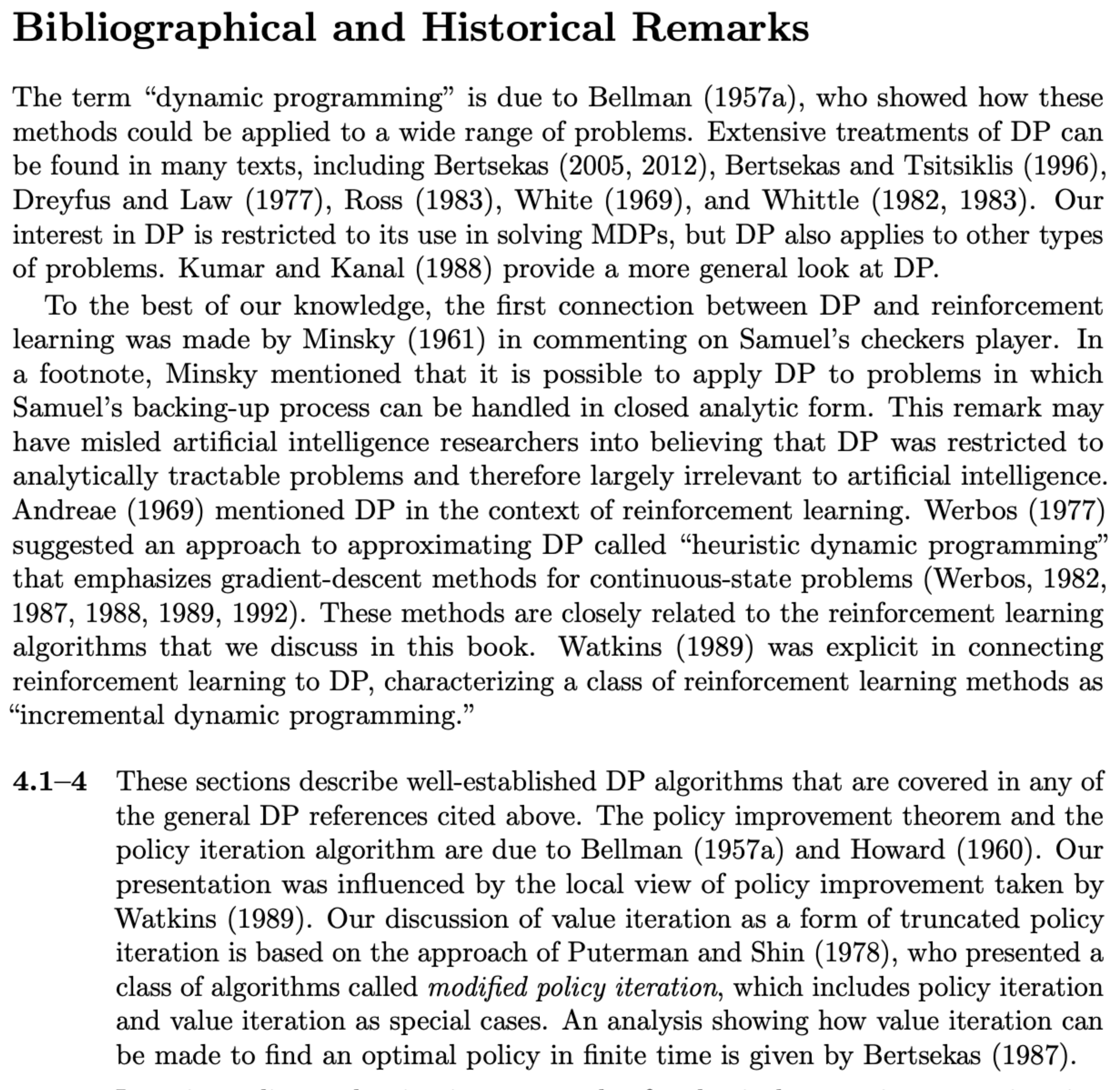
How do we break the curse of dimensionality?

- Are DP methods efficient?
- How do we scale DP methods?
- Does scaling require approximation?
- Do people use DP in practice?
- How can we combine DP methods and methods that interact with the world?

Where do we get the model?

- Does it need to be perfect?
- Can we learn the model?
- Would planning with a learned model be better than a model-free approach like Q-learning?
  - In what settings might the model-based method be better—with a learned model?

Check the end of each chapter



Bibliographical and Historical Remarks

The term "dynamic programming" is due to Bellman (1957a), who showed how these methods could be applied to a wide range of problems. Extensive treatments of DP can be found in many texts, including Bertsekas (2005, 2012), Bertsekas and Tsitsiklis (1996), Dreyfus and Law (1977), Ross (1983), White (1969), and Whittle (1982, 1983). Our interest in DP is restricted to its use in solving MDPs, but DP also applies to other types of problems. Kumar and Kanal (1988) provide a more general look at DP.

To the best of our knowledge, the first connection between DP and reinforcement learning was made by Minsky (1961) in commenting on Samuel's checkers player. In a footnote, Minsky mentioned that it is possible to apply DP to problems in which Samuel's backing-up process can be handled in closed analytic form. This remark may have misled artificial intelligence researchers into believing that DP was restricted to analytically tractable problems and therefore largely irrelevant to artificial intelligence. Andreae (1969) mentioned DP in the context of reinforcement learning. Werbos (1977) suggested an approach to approximating DP called "heuristic dynamic programming" that emphasizes gradient-descent methods for continuous-state problems (Werbos, 1982, 1987, 1988, 1989, 1992). These methods are closely related to the reinforcement learning algorithms that we discuss in this book. Watkins (1989) was explicit in connecting reinforcement learning to DP, characterizing a class of reinforcement learning methods as "incremental dynamic programming."

4.1–4 These sections describe well-established DP algorithms that are covered in any of the general DP references cited above. The policy improvement theorem and the policy iteration algorithm are due to Bellman (1957a) and Howard (1960). Our presentation was influenced by the local view of policy improvement taken by Watkins (1989). Our discussion of value iteration as a form of truncated policy iteration is based on the approach of Puterman and Shin (1978), who presented a class of algorithms called *modified policy iteration*, which includes policy iteration and value iteration as special cases. An analysis showing how value iteration can be made to find an optimal policy in finite time is given by Bertsekas (1987).

Policy/value initialization

- How would a good value function initialization speed up DP?
- Is it safe? What if I get it wrong?
    - Recall optimistic initial values in bandits!
- Can the initial value function really be arbitrary?

Projects

- Groups of 2 to 4: minimum 2, maximum 4
- You must pick a project from the list
- The idea is that you project provides novel insight into an interesting empirical question
    - This might lead to a paper
    - This might be a small part of a paper

Challenge Q

5. **(Challenge Question)** A gambler has the opportunity to make bets on the outcomes of a sequence of coin flips. If the coin comes up heads, she wins as many dollars as she has staked on that flip; if it is tails, she loses her stake. The game ends when the gambler wins by reaching her goal of \$100, or loses by running out of money. On each flip, the gambler must decide what portion of her capital to stake, in integer numbers of dollars. This problem can be formulated as an undiscounted, episodic, finite MDP. The state is the gambler's capital, $s \in \{1, 2, ..., 99\}$ and the actions are stakes, $a \in \{0, 1, ..., \min(s, 100 - s)\}$. The reward is $+1$ when reaching the goal of \$100 and zero on all other transitions. The probability of seeing heads is $p_h = 0.4$.

5. **(Challenge Question)** A gambler has the opportunity to make bets on the outcomes of a sequence of coin flips. If the coin comes up heads, she wins as many dollars as she has staked on that flip; if it is tails, she loses her stake. The game ends when the gambler wins by reaching her goal of \$100, or loses by running out of money. On each flip, the gambler must decide what portion of her capital to stake, in integer numbers of dollars. This problem can be formulated as an undiscounted, episodic, finite MDP. The state is the gambler's capital, $s \in \{1, 2, ..., 99\}$ and the actions are stakes, $a \in \{0, 1, ..., \min(s, 100 - s)\}$. The reward is $+1$ when reaching the goal of \$100 and zero on all other transitions. The probability of seeing heads is $p_h = 0.4$.

(a) What does the value of a state mean in this problem? For example, in a gridworld where the value of 1 per step, the value represents the expected number of steps to goal. What does the value of state mean in the gambler's problem? Think about the minimum and maximum possible values.

Gambler’s problem

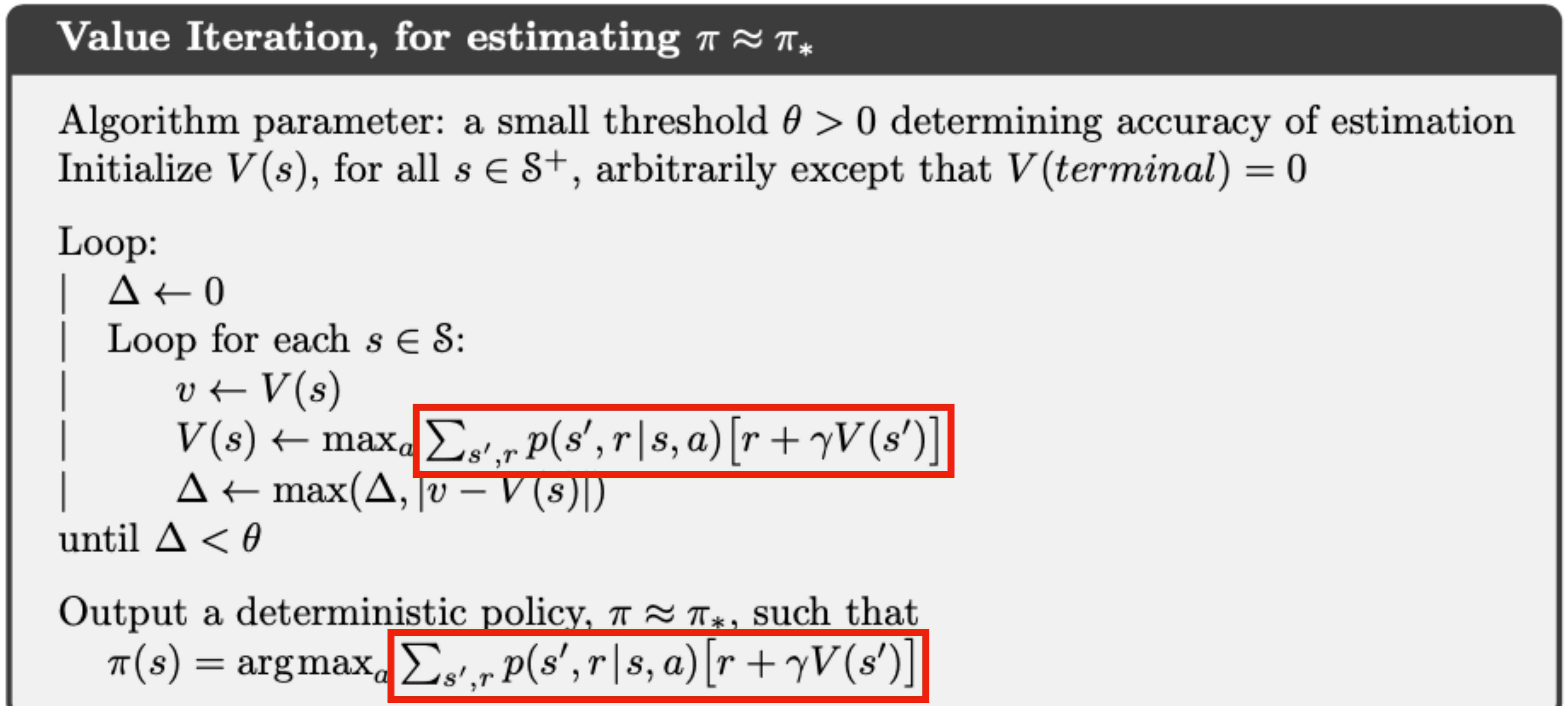
- What are the states?
  - gambler’s capital, s $\in$ {1,2, …, 99}
  - terminal states 0 and 100; Let $V_0 = 0$
- What are the rewards?
  - +1 win, zero otherwise
- What are the actions:
  - a = [1:min(s,100-s)]

Gambler’s problem

- For each action there are two possible outcomes, heads and tails
- For each action, compute $p(s', r \mid s, a)[r + V(s')]$ for each outcome, each $r + V(s')$

5. **(Challenge Question)** A gambler has the opportunity to make bets on the outcomes of a sequence of coin flips. If the coin comes up heads, she wins as many dollars as she has staked on that flip; if it is tails, she loses her stake. The game ends when the gambler wins by reaching her goal of \$100, or loses by running out of money. On each flip, the gambler must decide what portion of her capital to stake, in integer numbers of dollars. This problem can be formulated as an undiscounted, episodic, finite MDP. The state is the gambler's capital, $s \in \{1, 2, ..., 99\}$ and the actions are stakes, $a \in \{0, 1, ..., \min(s, 100 - s)\}$. The reward is $+1$ when reaching the goal of \$100 and zero on all other transitions. The probability of seeing heads is $p_h = 0.4$.

   (b) Modify the pseudocode for value iteration to more efficiently solve this specific problem, by exploiting your knowledge of the dynamics. *Hint: Not all states transition to every other state. For example, can you transition from state* 1 *to state* 99*?*



Value Iteration, for estimating $\pi \approx \pi_*$

```
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
|   $\Delta \leftarrow 0$
|   Loop for each $s \in \mathcal{S}$:
|       $v \leftarrow V(s)$
|       $V(s) \leftarrow \max_a \boxed{\sum_{s',r} p(s', r | s, a)\big[r + \gamma V(s')\big]}$
|       $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
    $\pi(s) = \operatorname{argmax}_a \boxed{\sum_{s',r} p(s', r | s, a)\big[r + \gamma V(s')\big]}$
```

Answer:

a) $v_\pi(s) = E[G_t|S_t = s]$ and G_t can have a value of 0 or 1. $G_t = 1$ if the game is won and 0 otherwise. Therefore, the value of a state determines the probability of winning from that state.

b) We know that given that action a is taken in state s, the set of possible next states are $\{s - a, s + a\}$. Therefore, we can update $V(s)$ as shown below:

$$V(s) \leftarrow \max_a \sum_{s' \in \{s-a, s+a\}, r} p(s', r|s, a) \left[r + \gamma v_k(s')\right]$$

And also compute $\pi(s)$ as shown below:

$$\pi(s) \leftarrow \text{argmax}_a \sum_{s' \in \{s-a, s+a\}, r} p(s', r|s, a) \left[r + \gamma v_k(s')\right]$$