Understanding Deep Learning

Chapter 19: Reinforcement learning

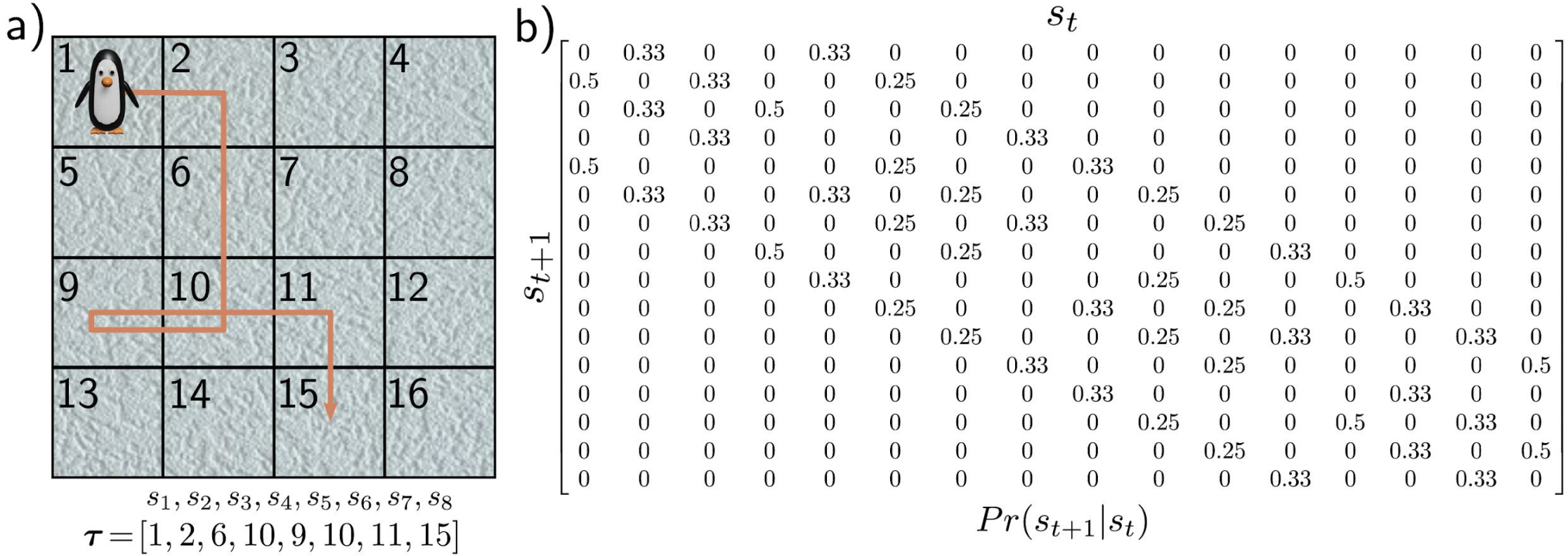


Figure 19.1 Markov process. A Markov process consists of a set of states and transition probabilities $Pr(s_{t+1}|s_t)$ that define the probability of moving to state s_{t+1} given the current state is s_t. a) The penguin can visit 16 different positions (states) on the ice. b) The ice is slippery, so at each time, it has an equal probability of moving to any adjacent state. For example, in position 6, it has a 25% chance of moving to states 2, 5, 7, and 10. A trajectory $\tau = [s_1, s_2, s_3, \ldots]$ from this process consists of a sequence of states.

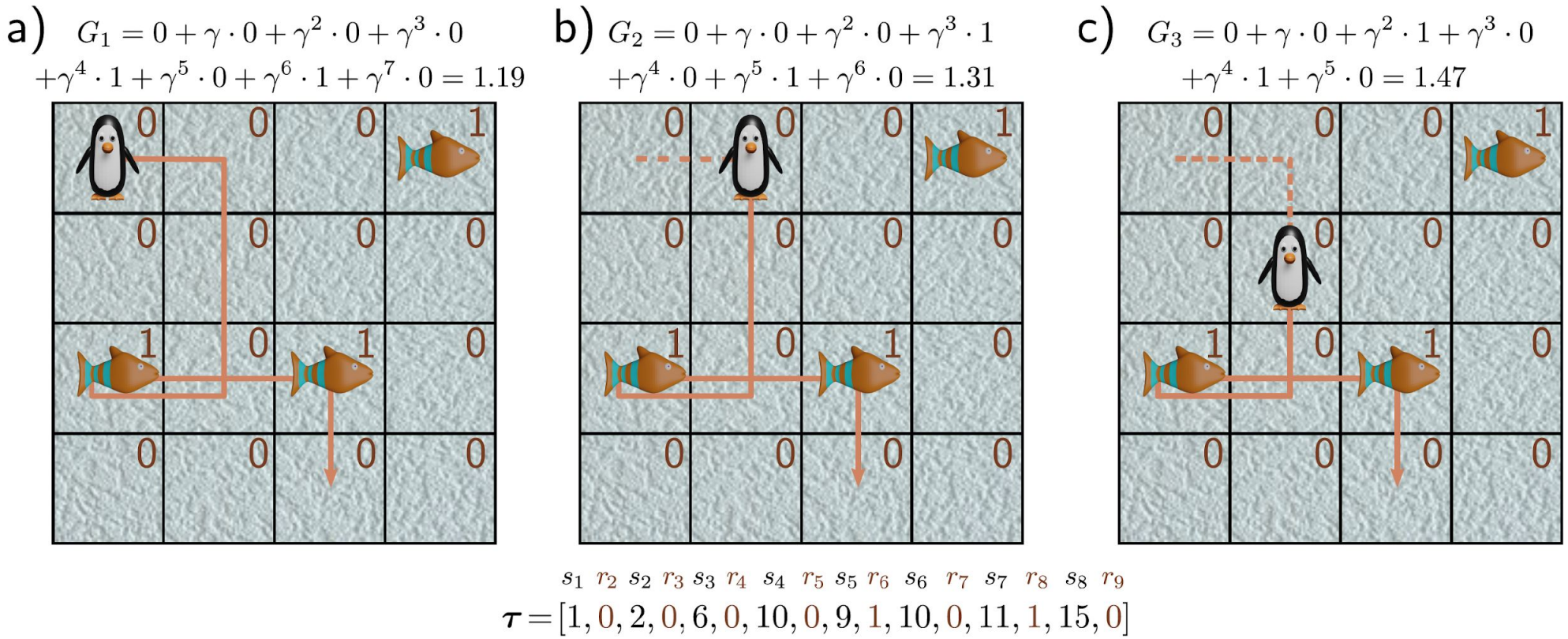


Figure 19.2 Markov reward process. This associates a distribution $Pr(r_{t+1}|s_t)$ of rewards r_{t+1} with each state s_t. a) Here, the rewards are deterministic; the penguin will receive a reward of $+1$ if it lands on a fish and 0 otherwise. The trajectory τ now consists of a sequence $s_1, r_2, s_2, r_3, s_3, r_4 \ldots$ of alternating states and rewards, terminating after eight steps. The return G_t of the sequence is the sum of discounted future rewards, here with discount factor $\gamma = 0.9$. b-c) As the penguin proceeds along the trajectory and gets closer to reaching the rewards, the return increases.

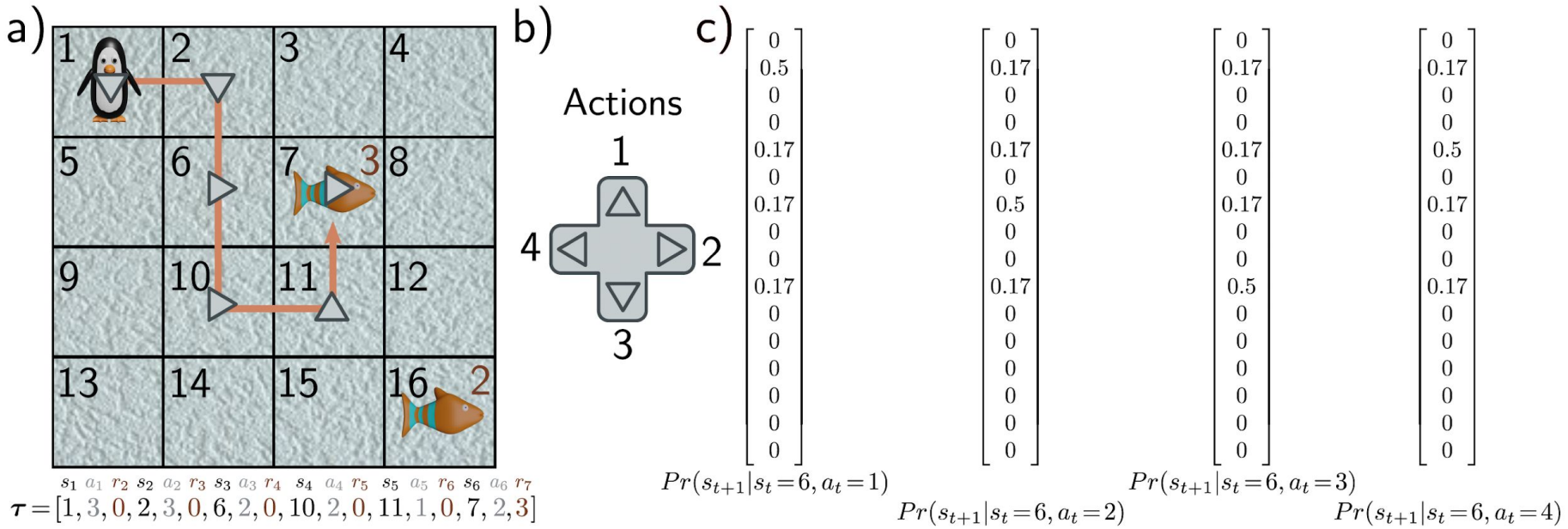
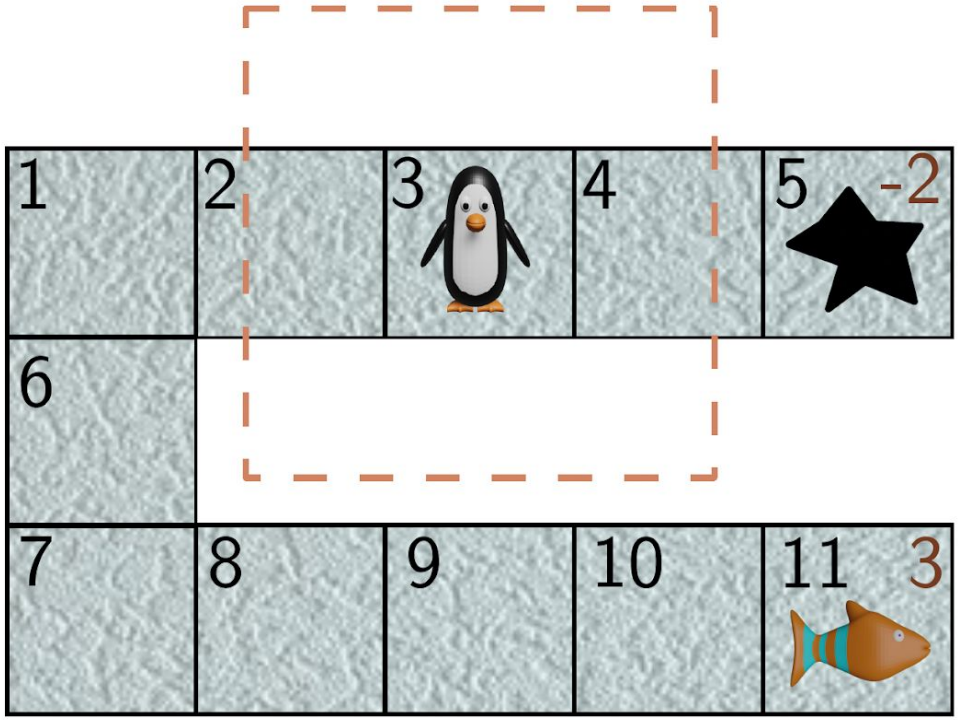


Figure 19.3 Markov decision process. a) The agent (penguin) can perform one of a set of actions in each state. The action influences both the probability of moving to the successor state and the probability of receiving rewards. b) Here, the four actions correspond to moving up, right, down, and left. c) For any state (here, state 6), the action changes the probability of moving to the next state. The penguin moves in the intended direction with 50% probability, but the ice is slippery, so it may slide to one of the other adjacent positions with equal probability. Accordingly, in panel (a), the action taken (gray arrows) doesn't always line up with the trajectory (orange line). Here, the action does not affect the reward, so $Pr(r_{t+1}|s_t, a_t) = Pr(r_{t+1}|s_t)$. The trajectory $\boldsymbol{\tau}$ from an MDP consists of a sequence $s_1, a_1, r_2, s_2, a_2, r_3, s_3, a_3, r_4 \ldots$ of alternating states s_t, actions a_t, and rewards, r_{t+1}. Note that here the penguin receives the reward when it *leaves* a state with a fish (i.e., the reward is received for passing through the fish square, regardless of whether the penguin arrived there intentionally or not).

Figure 19.4 Partially observable Markov decision process (POMDP). In a POMDP, the agent does not have access to the entire state. Here, the penguin is in state three and can only see the region in the dashed box. This is indistinguishable from what it would see in state nine. In the first case, moving right leads to the hole in the ice (with -2 reward) and, in the latter, to the fish (with +3 reward).



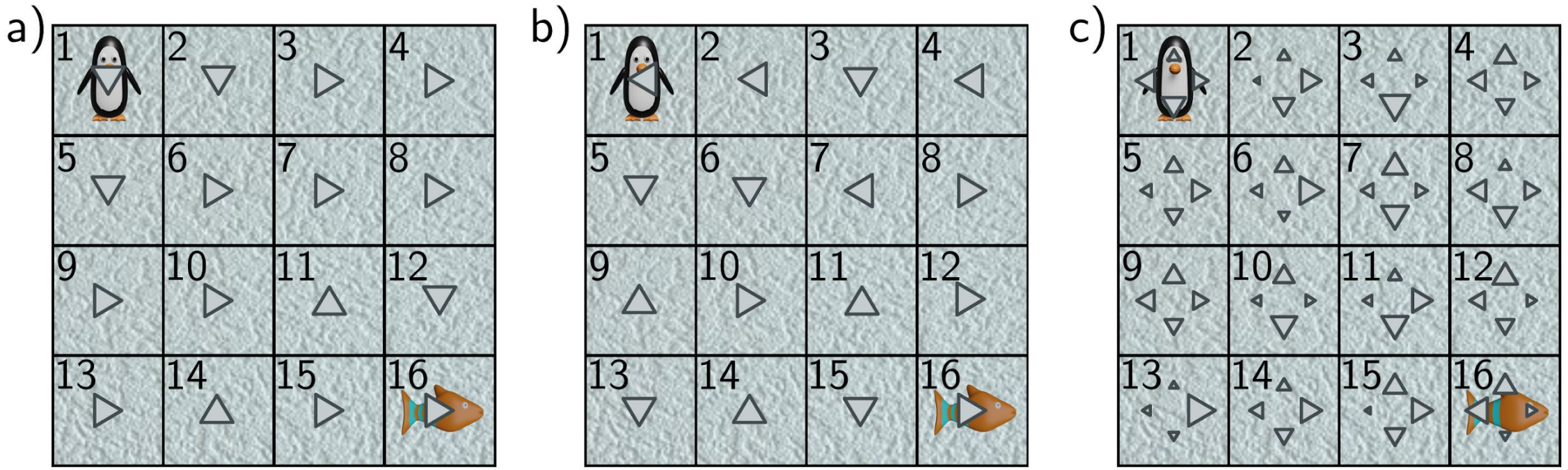
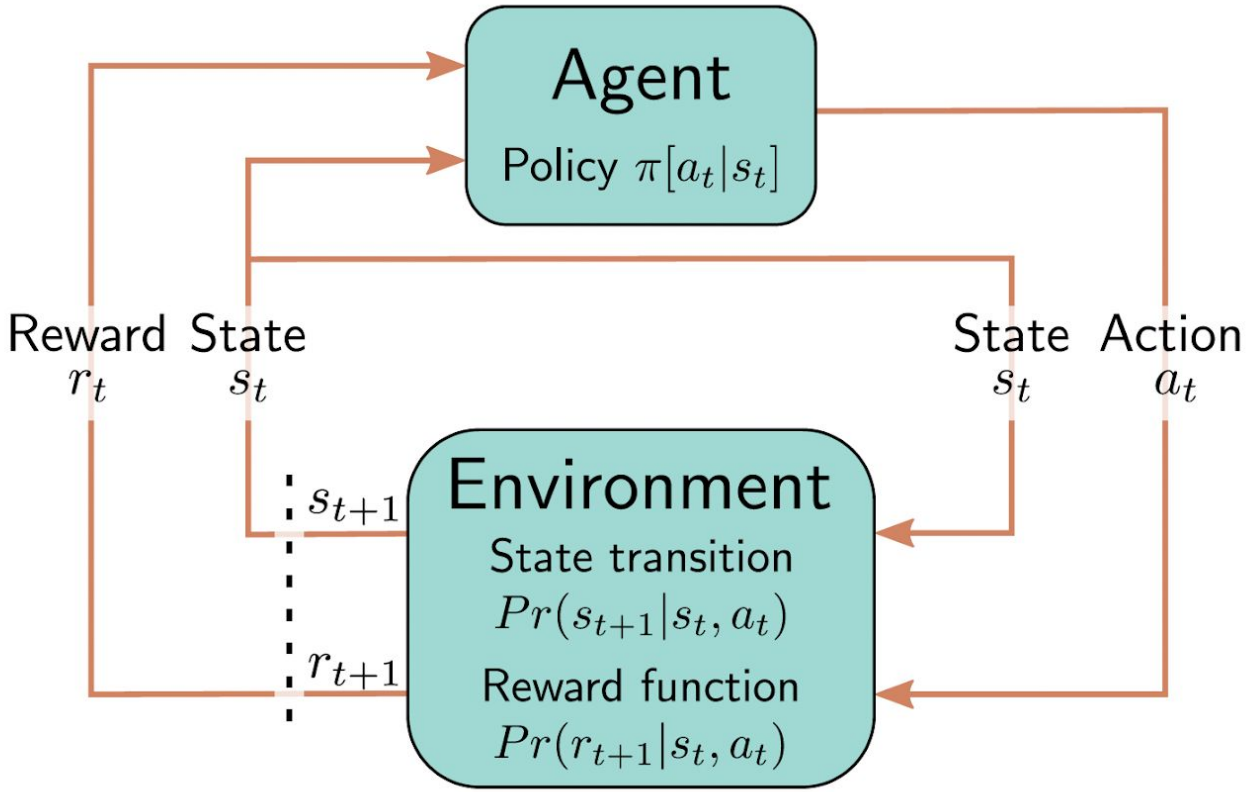


Figure 19.5 Policies. a) A deterministic policy always chooses the same action in each state (indicated by arrow). Some policies are better than others. This policy is not optimal but still generally steers the penguin from top-left to bottom-right where the reward lies. b) This policy is more random. c) A stochastic policy has a probability distribution over actions for each state (probability indicated by size of arrows). This has the advantage that the agent explores the states more thoroughly and can be necessary for optimal performance in partially observable Markov decision processes.

Figure 19.6 Reinforcement learning loop. The agent takes an action a_t at time t based on the state s_t, according to the policy $\pi[a_t|s_t]$. This triggers the generation of a new state s_{t+1} (via the state transition function) and a reward r_{t+1} (via the reward function). Both are passed back to the agent, which then chooses a new action.



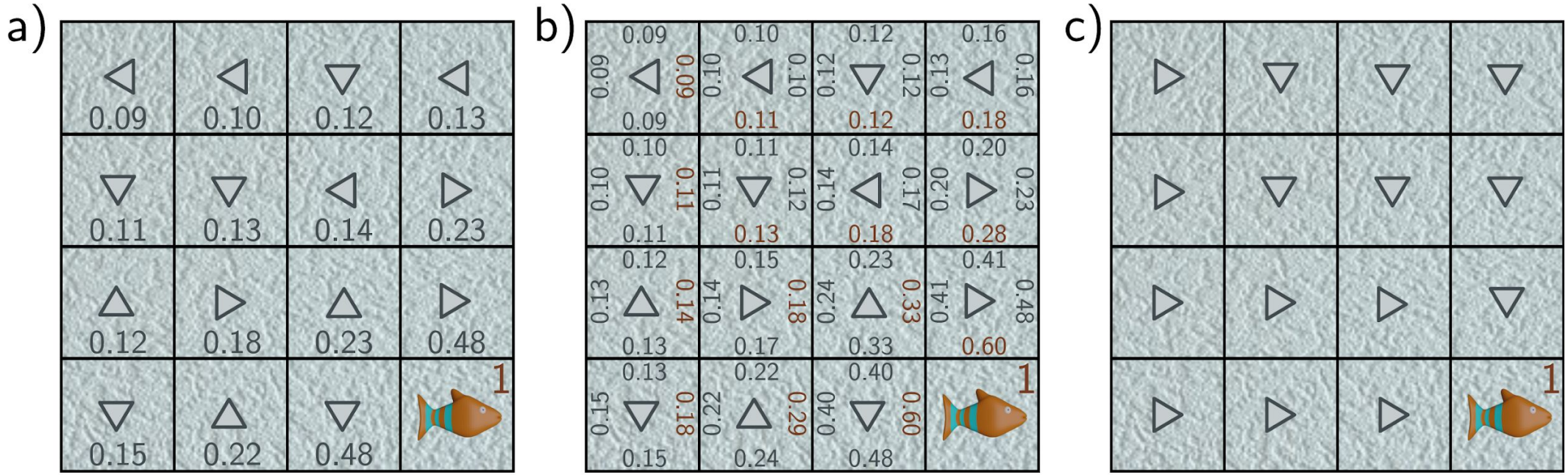


Figure 19.7 State and action values. a) The value $v[s_t|\pi]$ of a state s_t (number at each position) is the expected return for this state for a given policy π (gray arrows). It is the average sum of discounted rewards received over many trajectories started from this state. Here, states closer to the fish are more valuable. b) The value $q[s_t, a_t, \pi]$ of an action a_t in state s_t (four numbers at each position/state corresponding to four actions) is the expected return given that this particular action is taken in this state. In this case, it gets larger as we get closer to the fish and is larger for actions that head in the direction of the fish. c) If we know the action values at a state, then the policy can be modified so that it chooses the maximum of these values (red numbers in panel b).

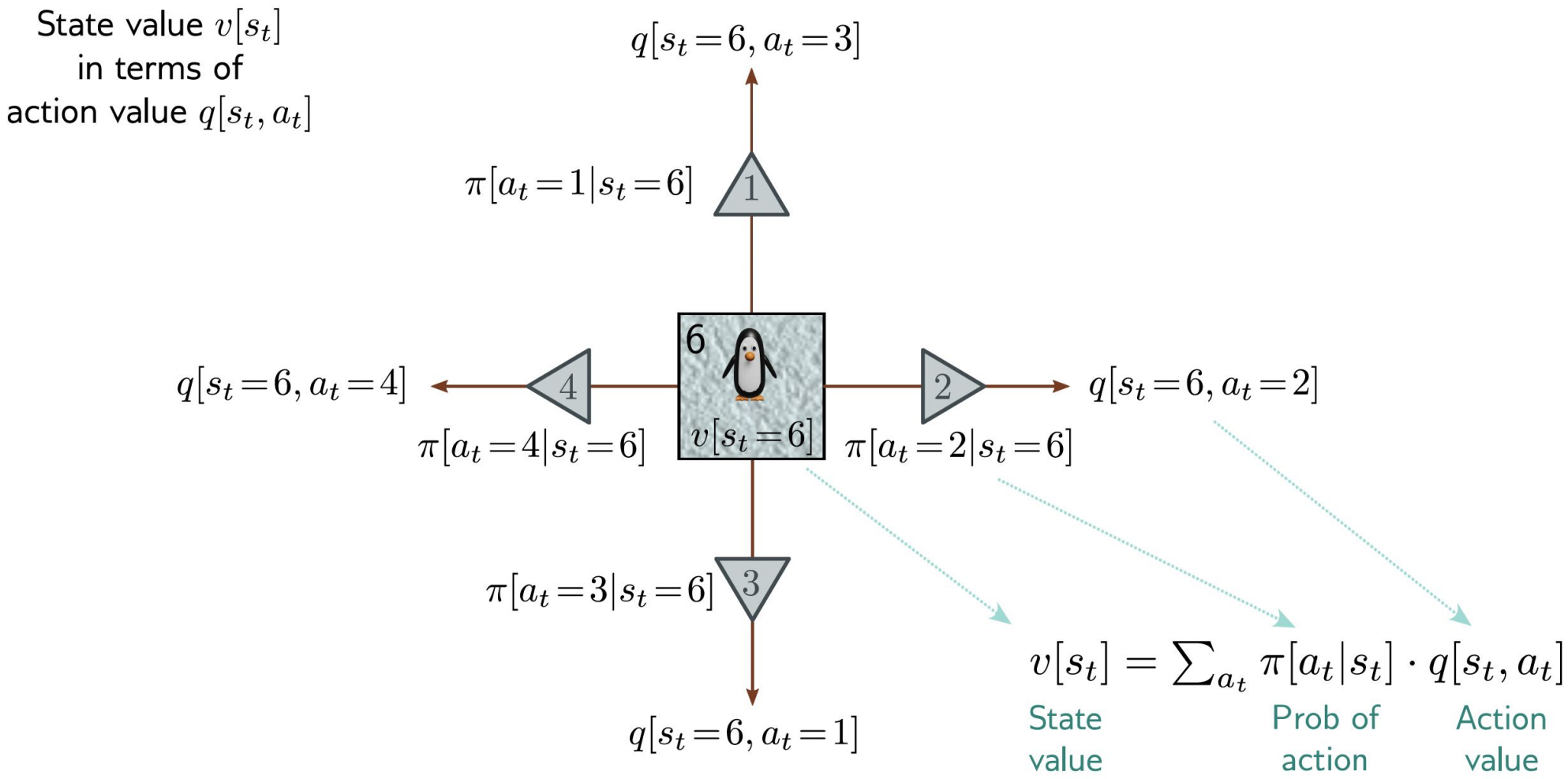


Figure 19.8 Relationship between state values and action values. The value of state six $v[s_t=6]$ is a weighted sum of the action values $q[s_t=6, a_t]$ at state six, where the weights are the policy probabilities $\pi[a_t|s_t=6]$ of taking that action.

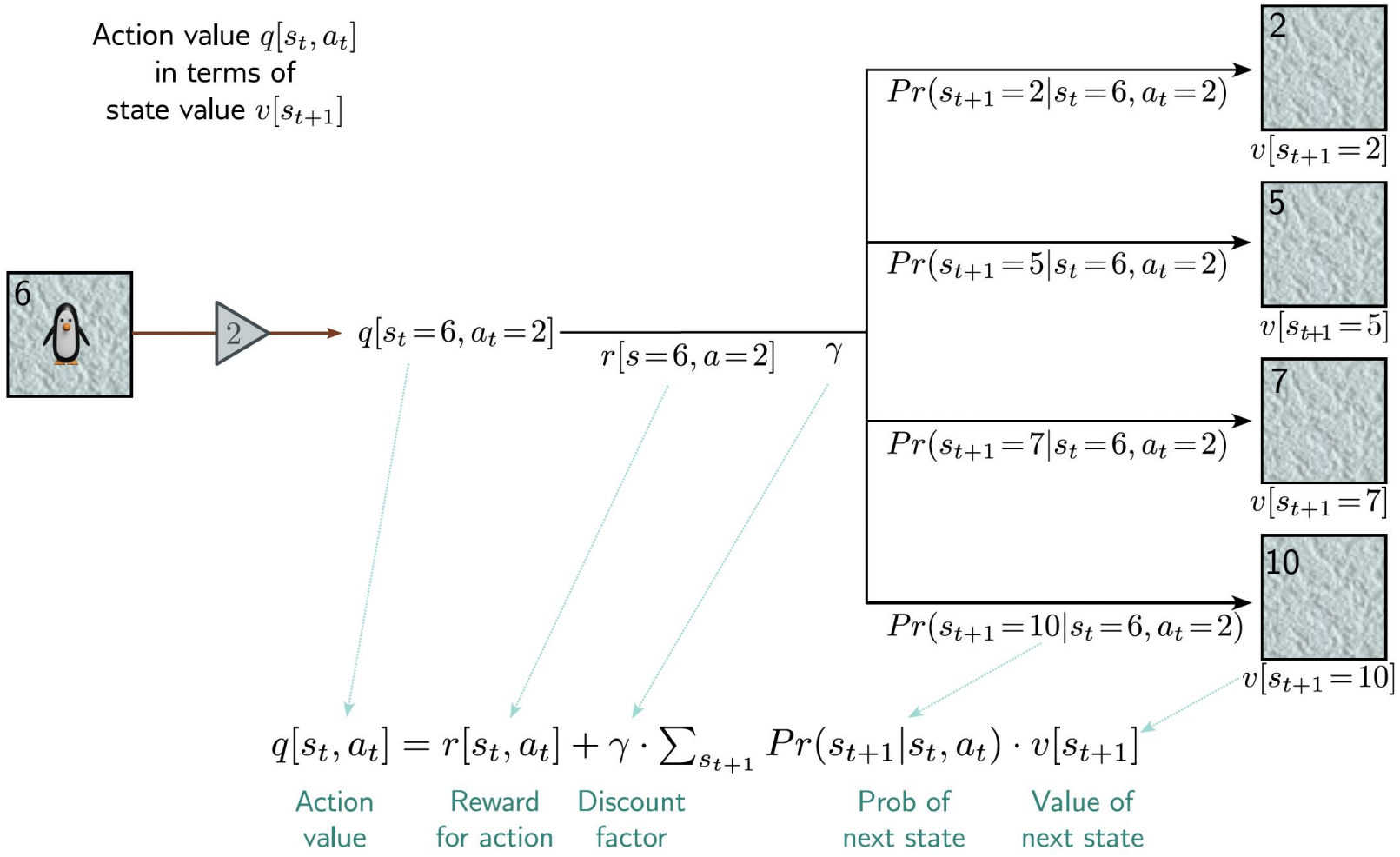


Figure 19.9 Relationship between action values and state values. The value $q[s_t = 6, a_t = 2]$ of taking action two in state six is the reward $r[s_t = 6, a_t = 2]$ from taking that action plus a weighted sum of the discounted values $v[s_{t+1}]$ of being in successor states, where the weights are the transition probabilities $Pr(s_{t+1}|s_t = 6, a_t = 2)$. The Bellman equations chain this relation with that of figure 19.8 to link the current and next (i) state values and (ii) action values.

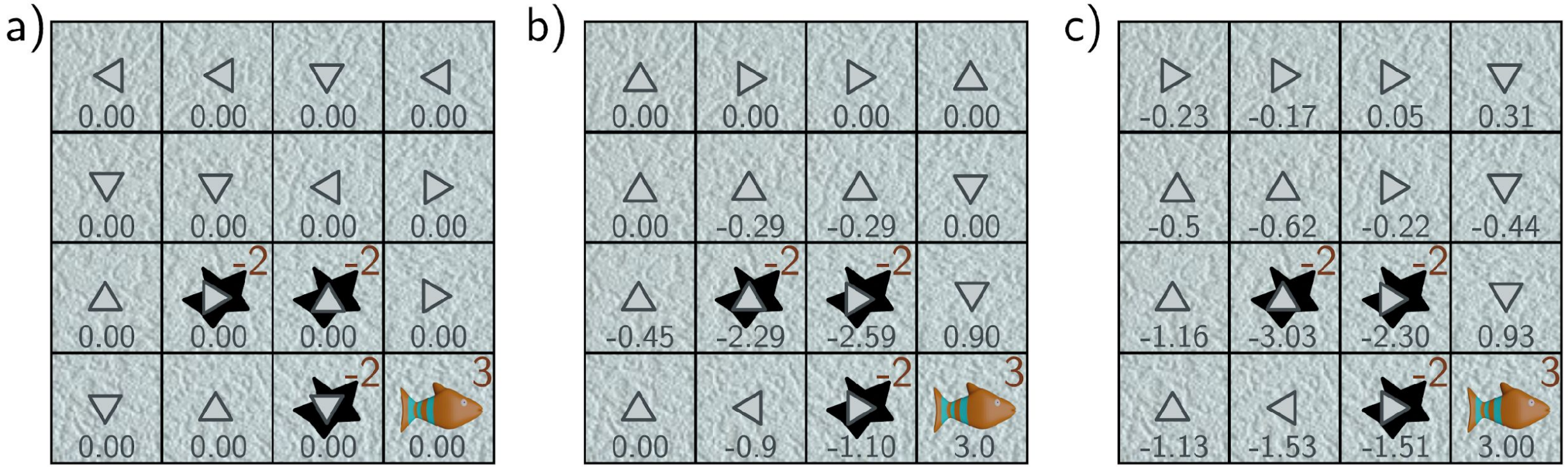


Figure 19.10 Dynamic programming. a) The state values are initialized to zero, and the policy (arrows) is chosen randomly. b) The state values are updated to be consistent with their neighbors (equation 19.11, shown after two iterations). The policy is updated to move the agent to states with the highest value (equation 19.12). c) After several iterations, the algorithm converges to the optimal policy, in which the penguin tries to avoid the holes and reach the fish.

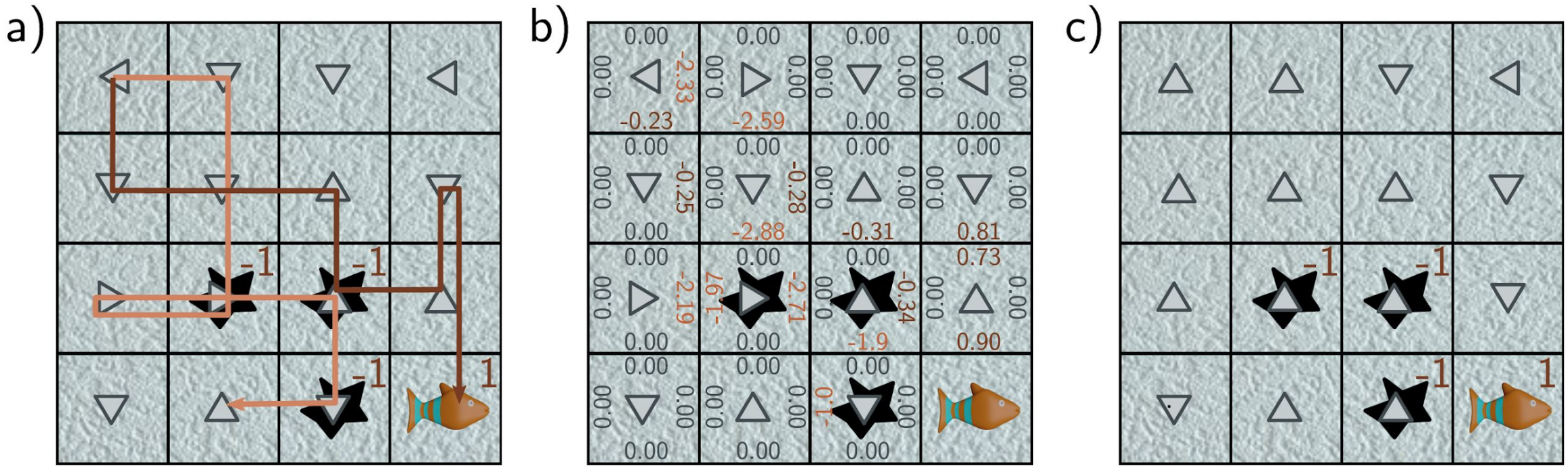
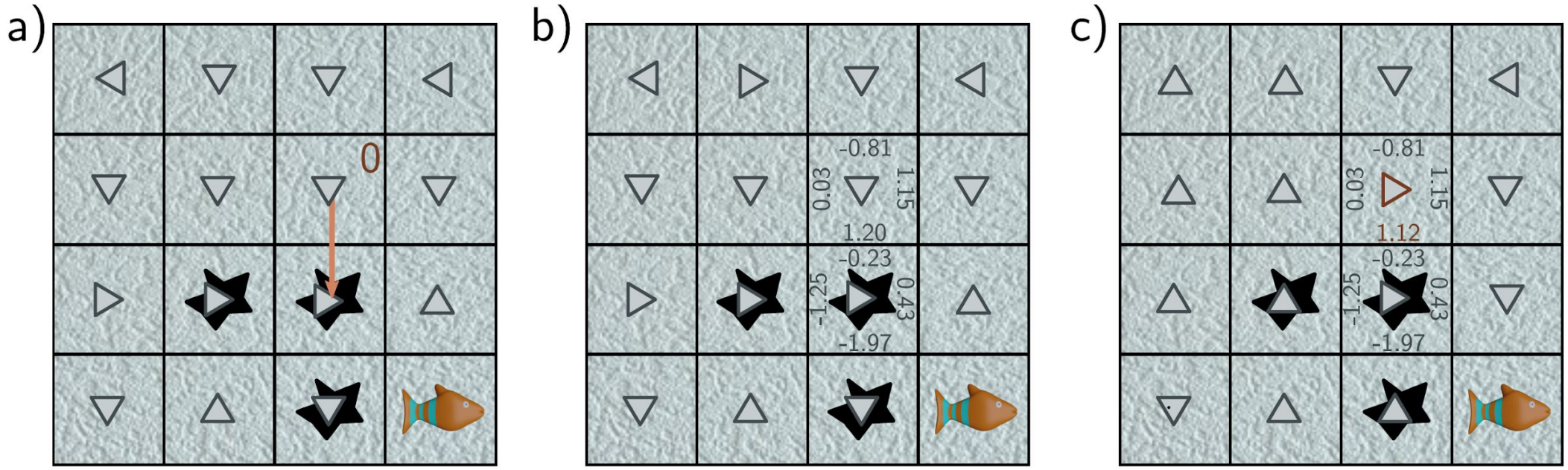


Figure 19.11 Monte Carlo methods. a) The policy (arrows) is initialized randomly. The MDP is repeatedly simulated, and the trajectories of these episodes are stored (orange and brown paths represent two trajectories). b) The action values are empirically estimated based on the observed returns averaged over these trajectories. In this case, the action values were all initially zero and have been updated where an action was observed. c) The policy can then be updated according to the action which received the best (or least bad) reward.



$$q[s_t, a_t] \leftarrow q[s_t, a_t] + \alpha\Big(r[s_t, a_t] + \gamma \cdot \max_a\big[q[s_{t+1}, a]\big] - q[s_t, a_t]\Big)$$

$$1.12 \leftarrow 1.20 + 0.1\Big(0.0 + 0.9 \cdot \max\big[-0.23, 0.43, -1.97, -1.25\big] - 1.20\Big)$$

Figure 19.12 Q-learning. a) The agent starts in state s_t and takes action $a_t = 2$ according to the policy. It does not slip on the ice and moves downward, receiving reward $r[s_t, a_t] = 0$ for leaving the original state. b) The maximum action value at the new state is found (here 0.43). c) The action value for action 2 in the original state is updated to 1.12 based on the current estimate of the maximum action value at the subsequent state, the reward, discount factor $\gamma = 0.9$, and learning rate $\alpha = 0.1$. This changes the highest action value at the original state, so the policy changes.

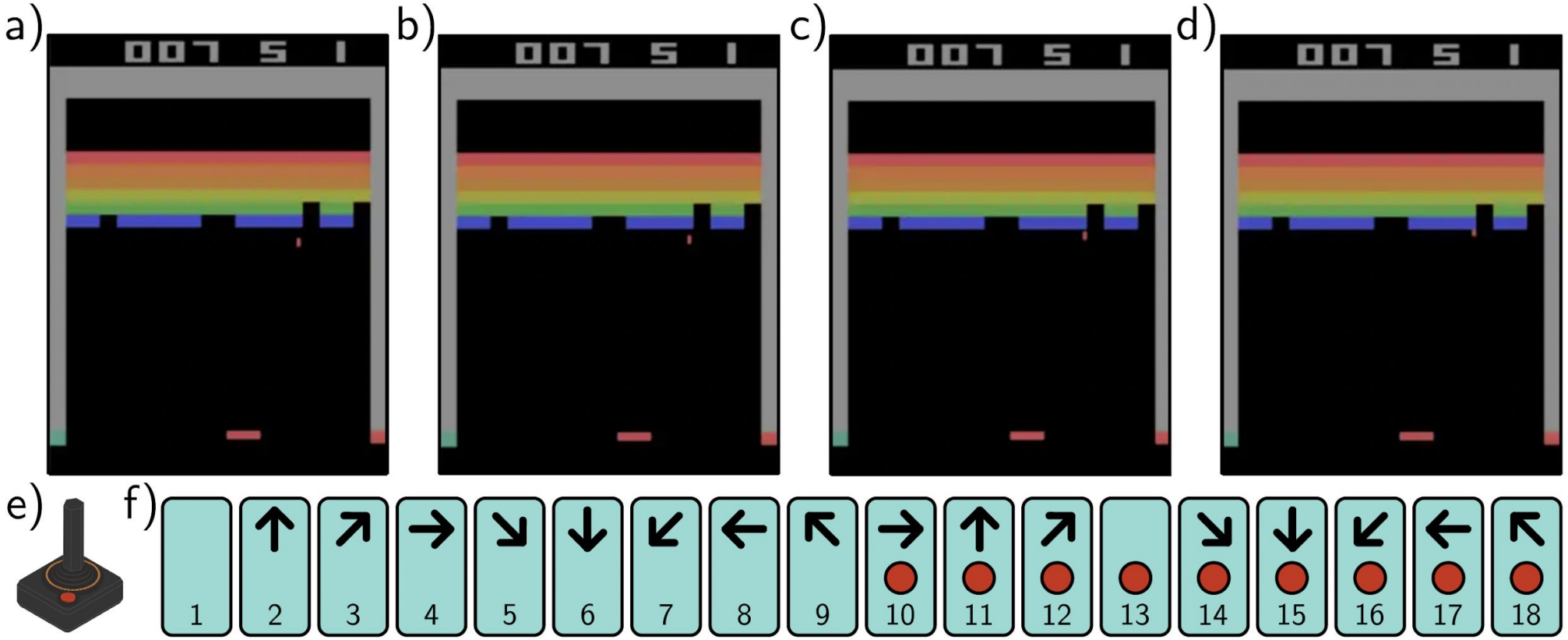


Figure 19.13 Atari Benchmark. The Atari benchmark consists of 49 Atari 2600 games, including Breakout (pictured), Pong, and various shoot-em-up, platform, and other types of games. a-d) Even for games with a single screen, the state is not fully observable from a single frame because the velocity of the objects is unknown. Consequently, it is usual to use several adjacent frames (here, four) to represent the state. e) The action simulates the user input via a joystick. f) There are eighteen actions corresponding to eight directions of movement or no movement, and for each of these nine cases, the button being pressed or not.

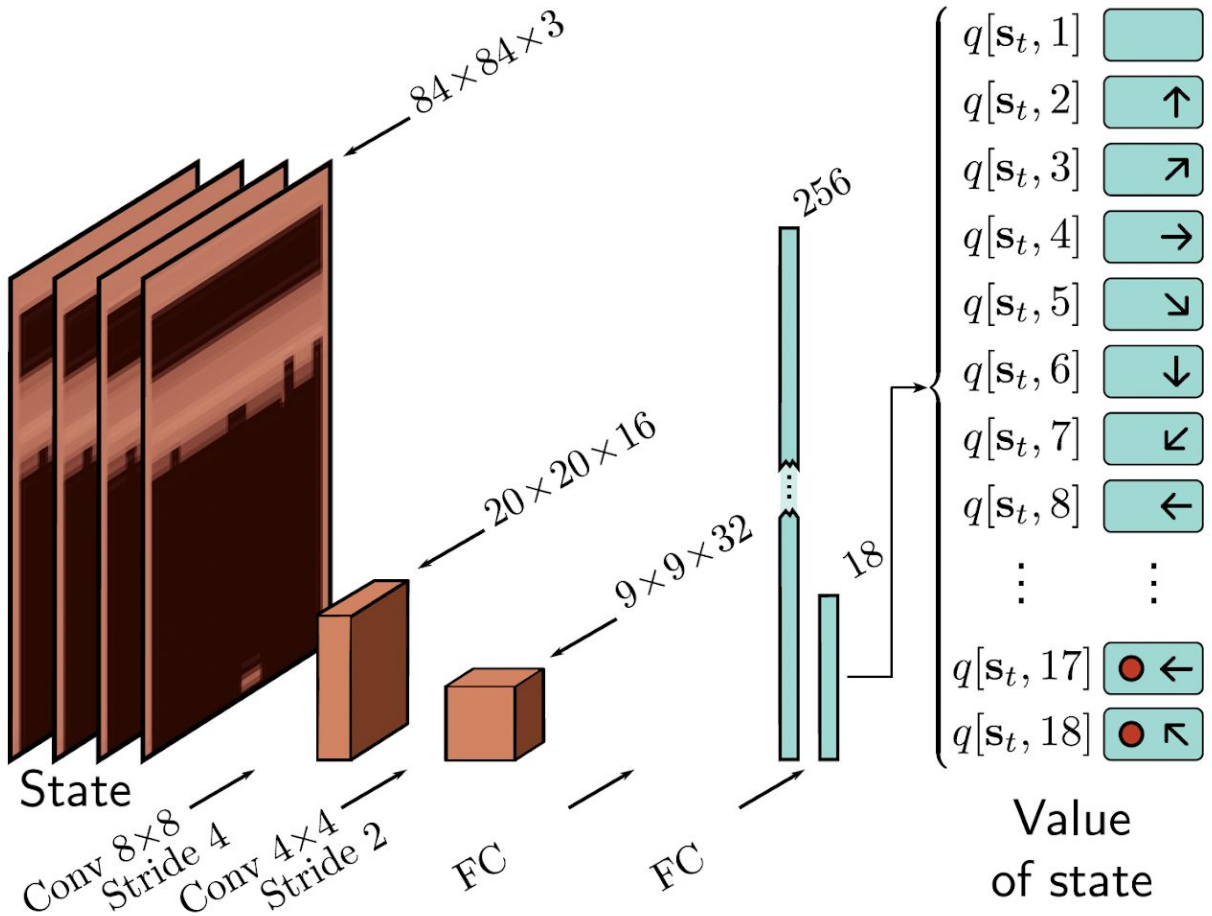
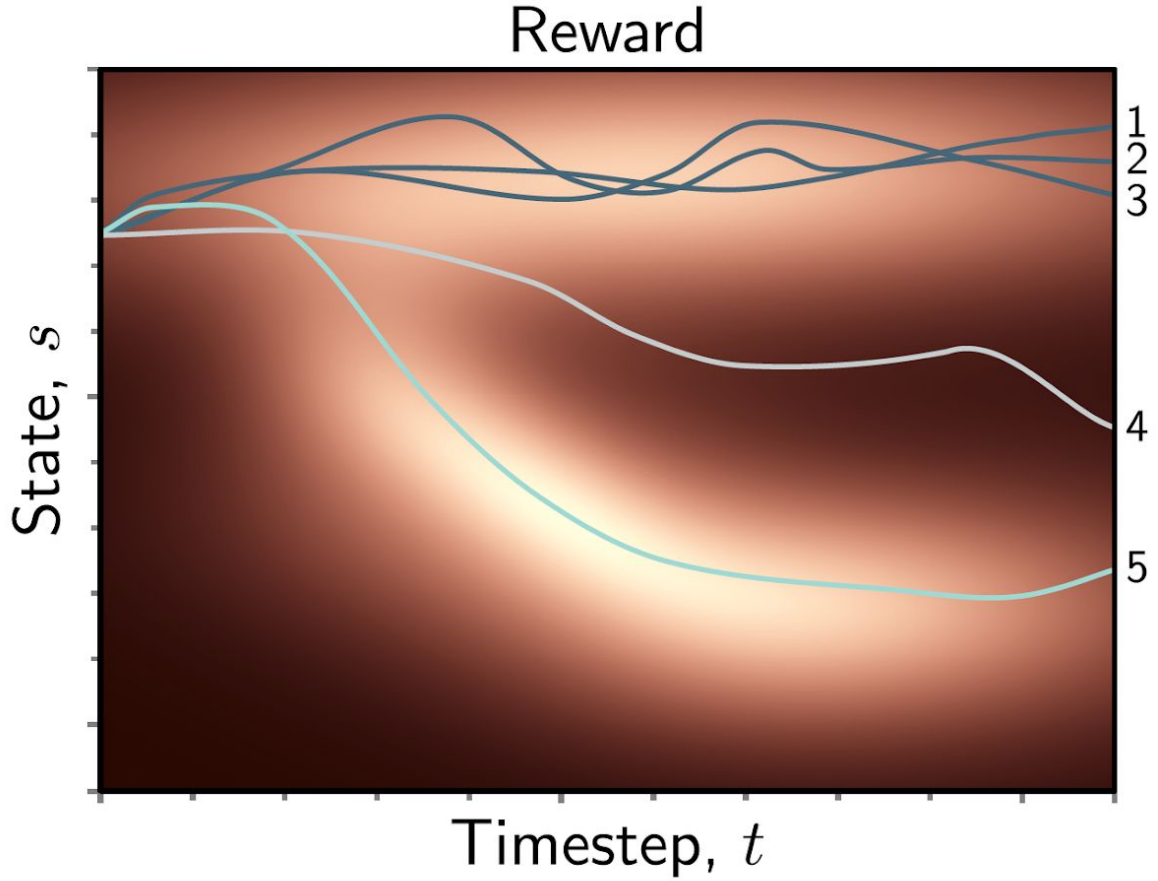


Figure 19.14 Deep Q-network architecture. The input $\mathbf{s}_t$ consists of four adjacent frames of the ATARI game. Each is resized to 84×84 and converted to grayscale. These frames are represented as four channels and processed by an 8×8 convolution with stride four, followed by a 4×4 convolution with stride 2, followed by two fully connected layers. The final output predicts the action value $q[\mathbf{s}_t, a_t]$ for each of the 18 actions in this state.

Figure 19.15 Policy gradients. Five episodes for the same policy (brighter indicates higher reward). Trajectories 1, 2, and 3 generate consistently high rewards, but similar trajectories already frequently occur with this policy, so there is no impetus for change. Trajectory 4 receives low rewards, so the policy should be modified to avoid producing similar trajectories. Trajectory 5 receives high rewards *and* is unusual. This will cause the largest change to the policy under equation 19.25.



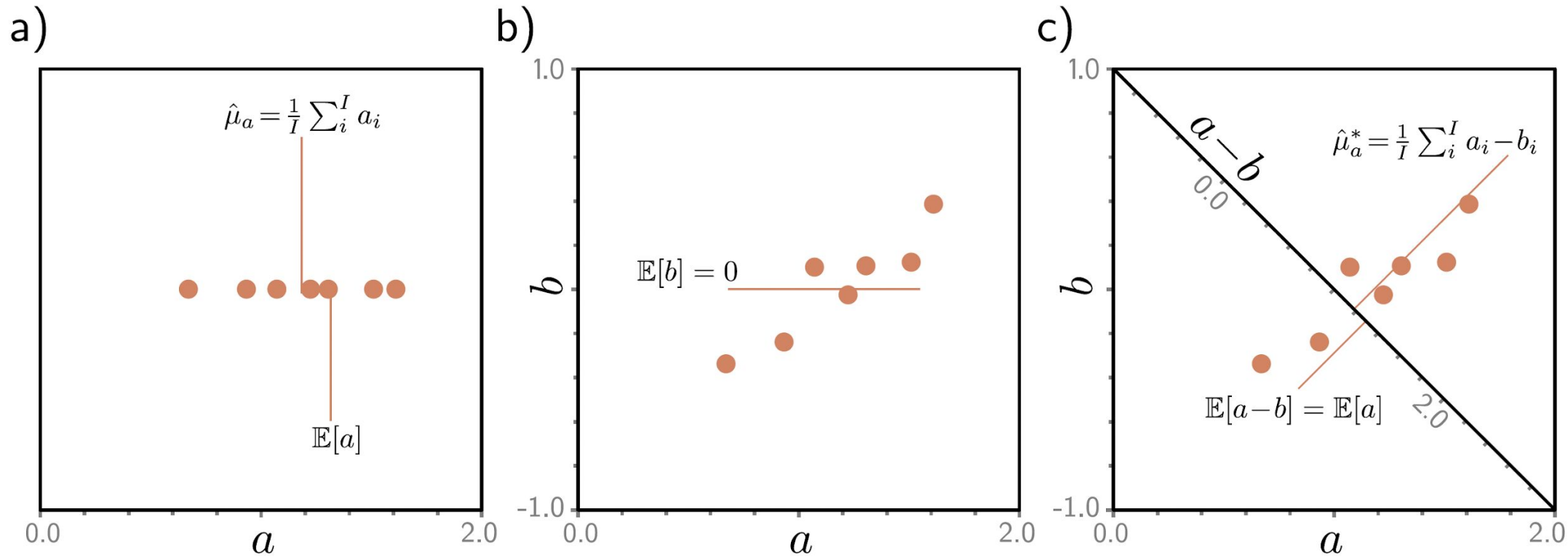


Figure 19.16 Decreasing variance of estimates using control variates. a) Consider trying to estimate $\mathbb{E}[a]$ from a small number of samples. The estimate (the mean of the samples) will vary based on the number of samples and the variance of those samples. b) Now consider observing another variable b that co-varies with a and has $\mathbb{E}[b] = 0$ and the same variance as a. c) The variance of the samples of $a - b$ is much less than that of a, but the expected value $\mathbb{E}[a - b] = \mathbb{E}[a]$, so we get an estimator with lower variance.

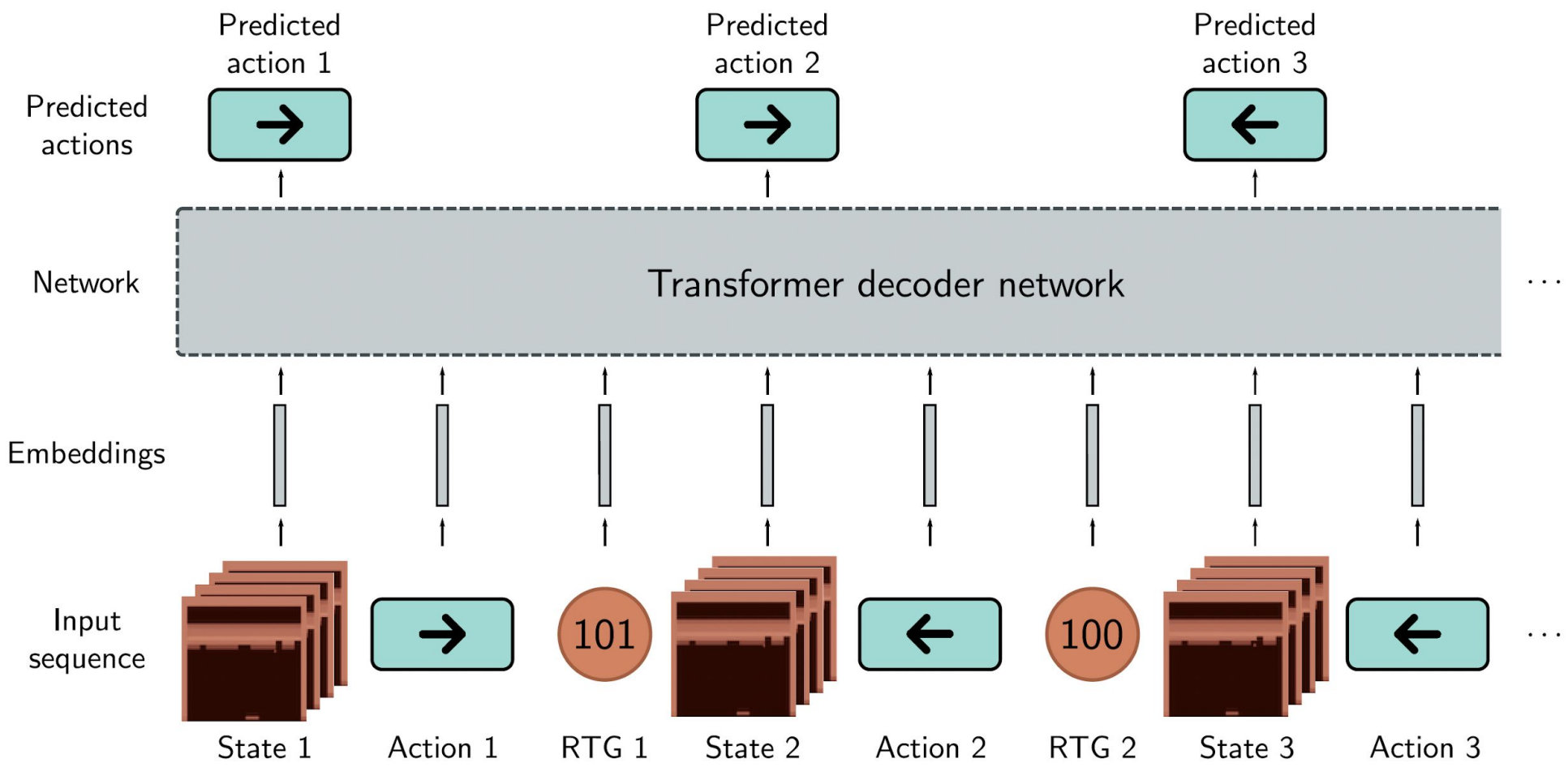


Figure 19.17 Decision transformer. The decision transformer treats offline reinforcement learning as a sequence prediction task. The input is a sequence of states, actions, and returns-to-go (remaining rewards in the episode), each of which is mapped to a fixed-size embedding. At each time step, the network predicts the next action. During testing, the returns-to-go are unknown; in practice, an initial estimate is made from which subsequent observed rewards are subtracted.

Figure 19.18 One trajectory through an MDP. The penguin receives a reward of $+1$ when it reaches the first fish tile, -2 when it falls in the hole, and $+1$ for reaching the second fish tile. The discount factor γ is 0.9.