CS 581

Advanced Artificial Intelligence

April 10, 2024

Announcements / Reminders

- Please follow the Week 12/13 To Do List instructions (if you haven't already)

- Programming Assignment #03: OPTIONAL/NOT FOR CREDIT

- FINAL EXAM is on Monday (04/22/2024) in RE 104!
  - different room!!!
  - IGNORE Registrar's FINAL EXAM date
  - Section 02: contact Mr. Charles Scott (scott@iit.edu) to make arrangements

Plan for Today

- Making Complex Decisions
  - Markov Decision Process
- Reinforcement Learning: Introduction
- Q-Learning

Sequential Decision Process

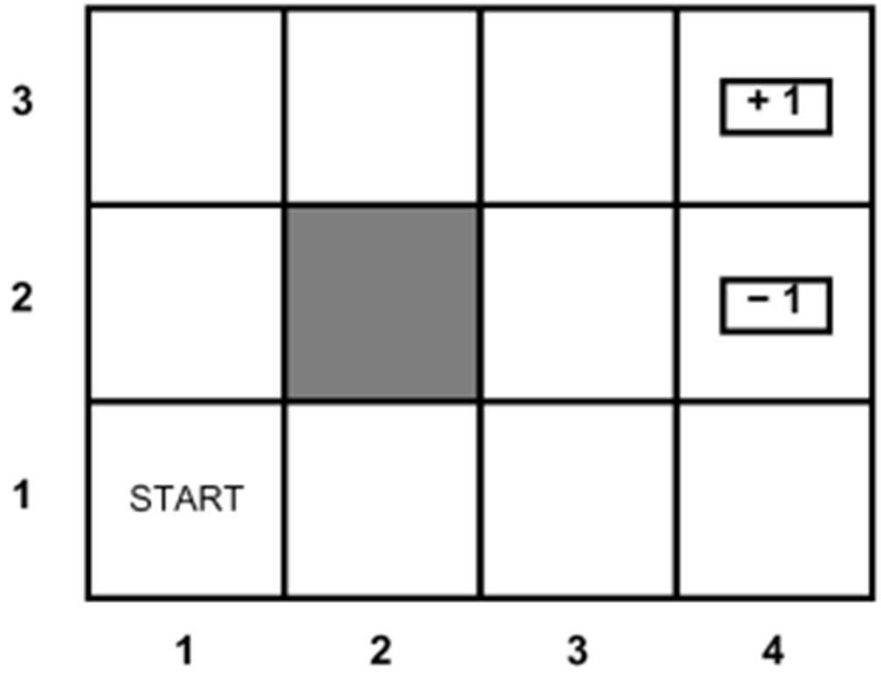
Sequential Decision Problem

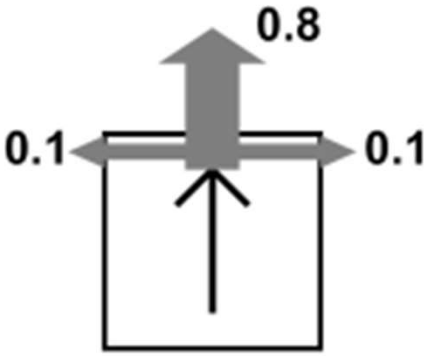
- Sequential decision problems are problems in which the agent's utility depends on a sequence of decisions.

- Sequential decision problems incorporate utilities, uncertainty, and sensing,
  - and include search and planning problems as special cases

Sequential Decision Problem

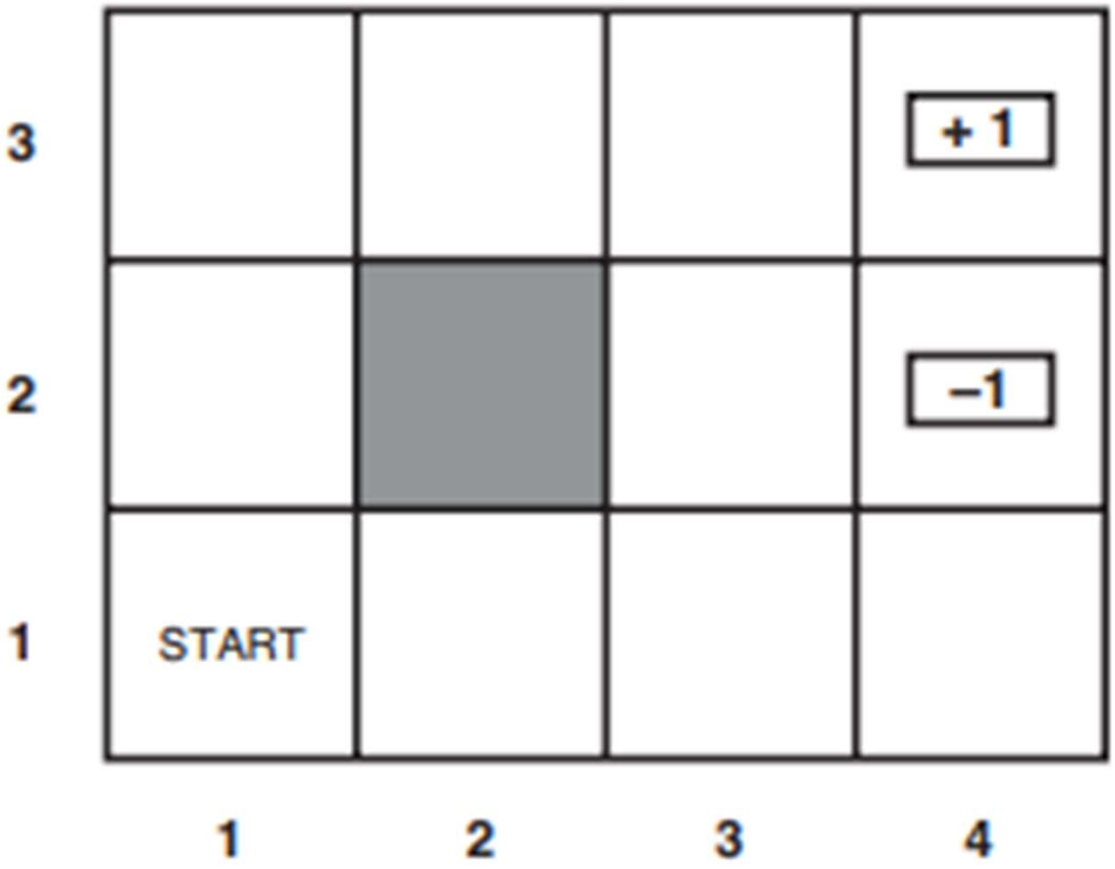
- The agent lives in a grid
- Walls block the agent's path
- The agent's actions do not always go as planned:
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- Small "living" reward each step
- Big rewards come at the end
- Goal: maximize sum of rewards*

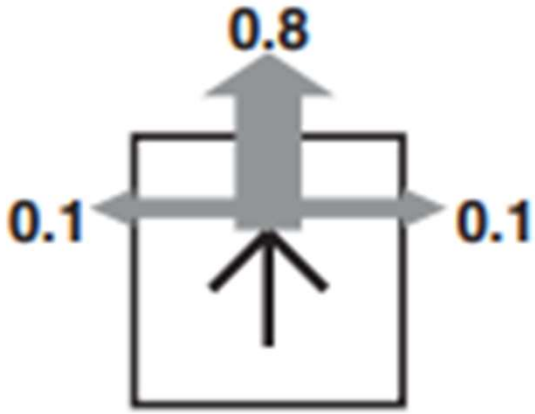




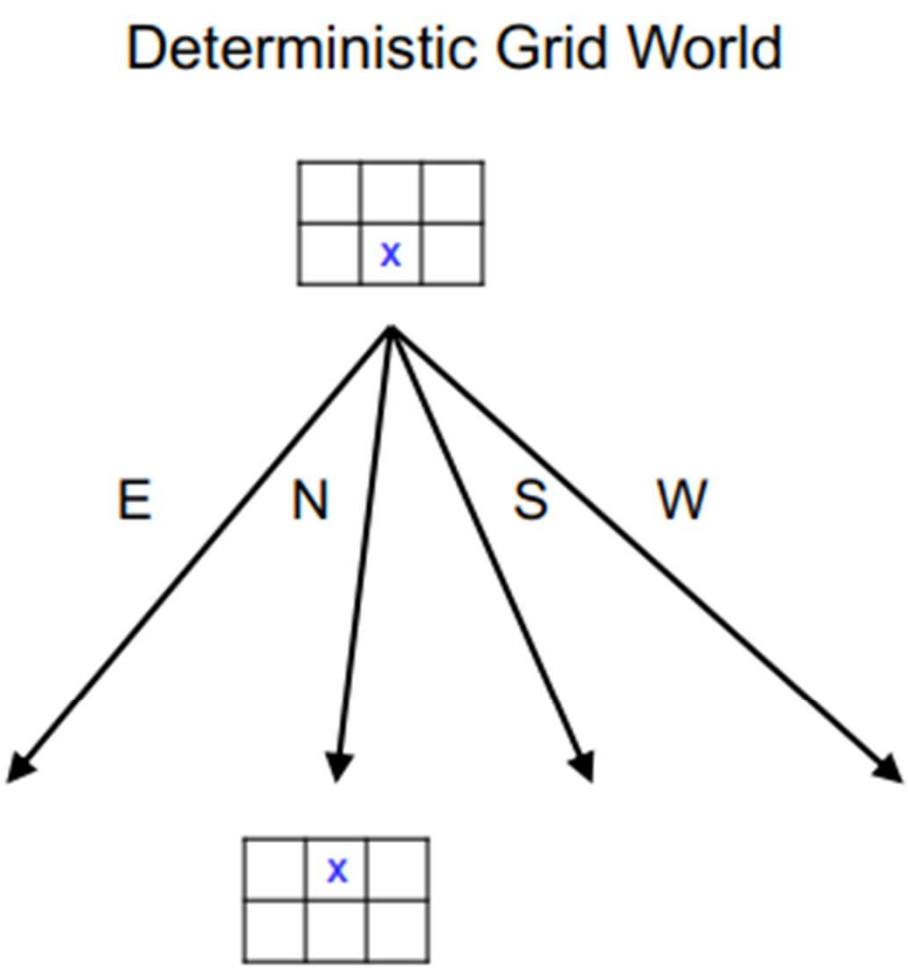
Example: Grid World

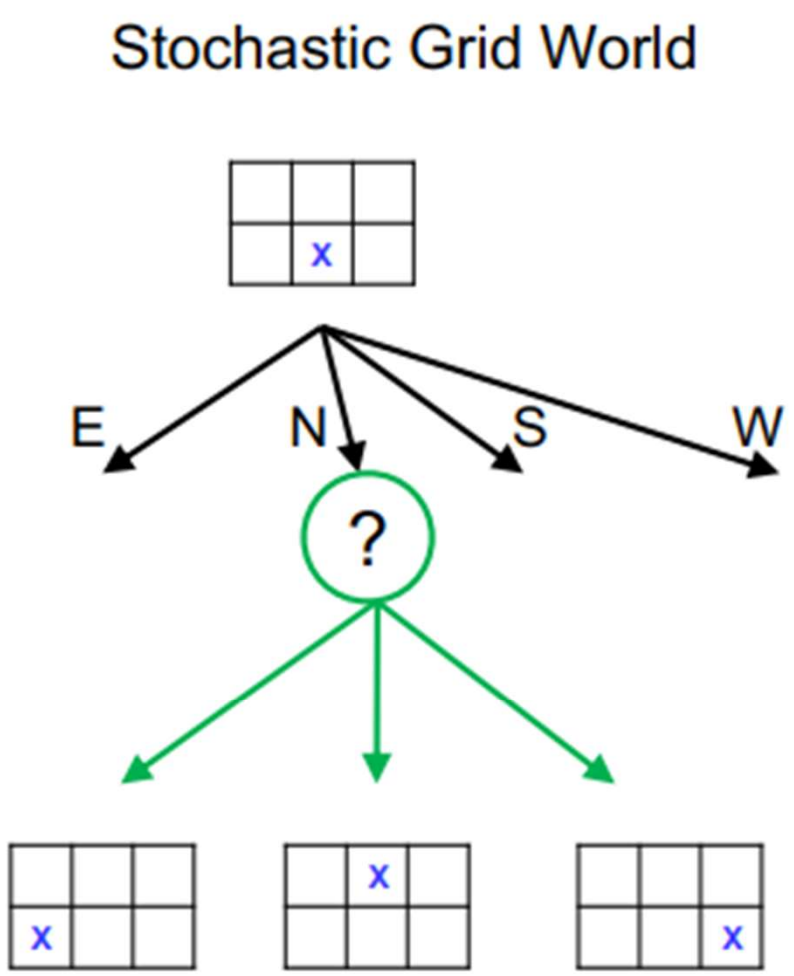
Fully-observable, non-deterministic, sequential





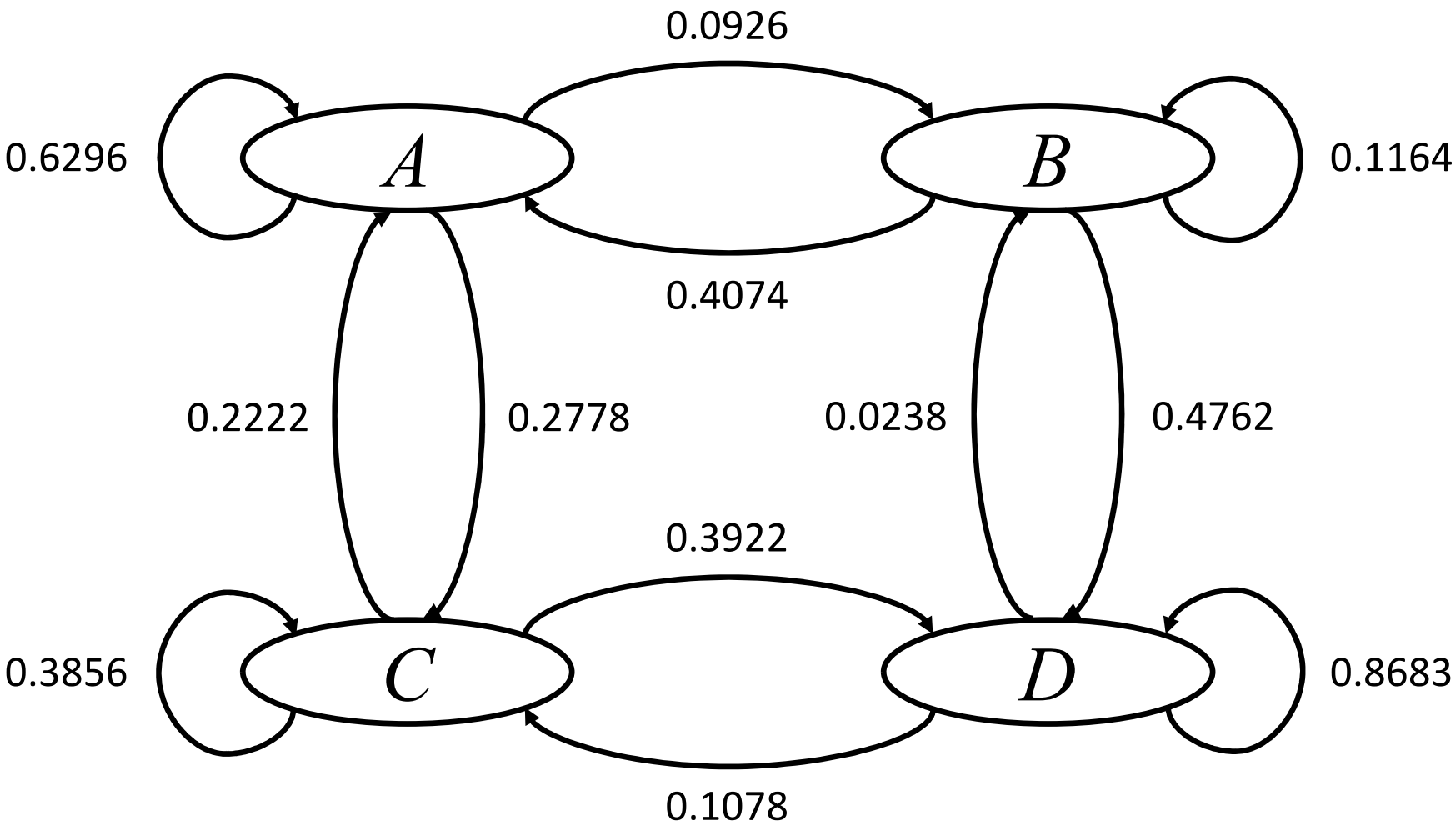
Deterministic vs. Non-Deterministic





Markov Decision Process (MDP)

State Space and Transition Model

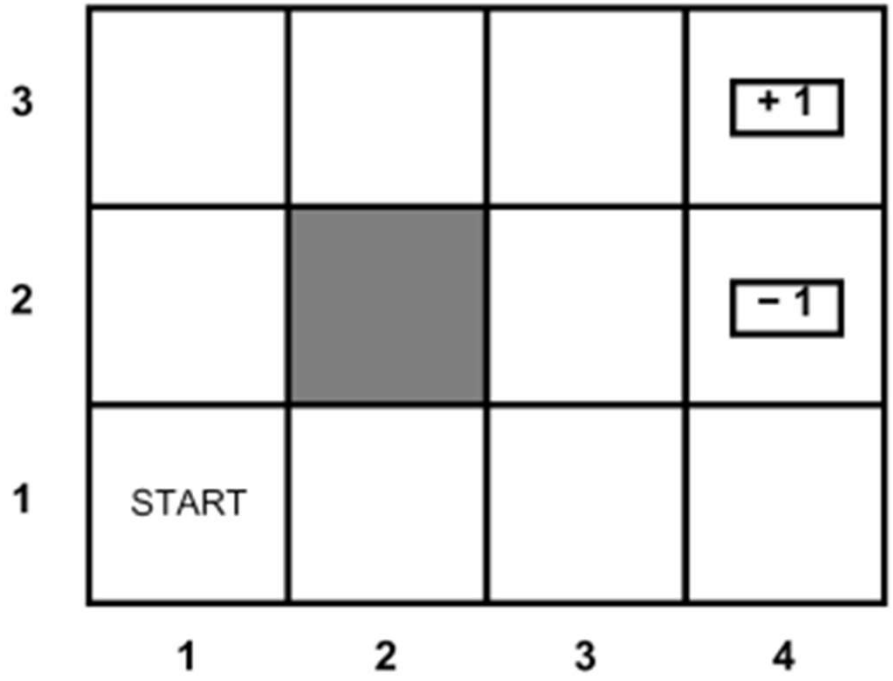


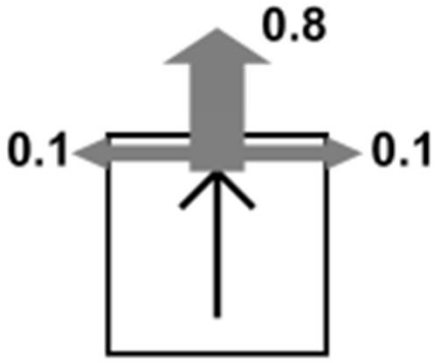
Problem Setting

- The world is represented through states
- At each state, an agent is given 0 (terminal states) or more actions to choose from
- Each action moves the agent, probabilistically, to a state (could be the current state) and results, probabilistically, in a reward (could be zero, negative, positive)
- The agent needs to maximize the sum of the rewards it accumulates over time
- Greedy strategy with respect to immediate rewards often do not work; the agent needs to consider the long-term consequences of its actions

Markov Decision Process

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function T(s,a,s')
    - Prob that a from s leads to s'
    - i.e., P(s' | s,a)
    - Also called the model
  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s')
  - A start state (or distribution)
  - Maybe a terminal state





Notation

- $P(s' \mid s, a)$ Probability of arriving at state s' given we are at state s and take action a
- $R(s, a, s')$ The reward the agent receives when it transitions from state s to state s' via action a

Markov Decision Process

- A sequential decision-making process
- Stochastic environment
- Markov transition model
- Additive rewards

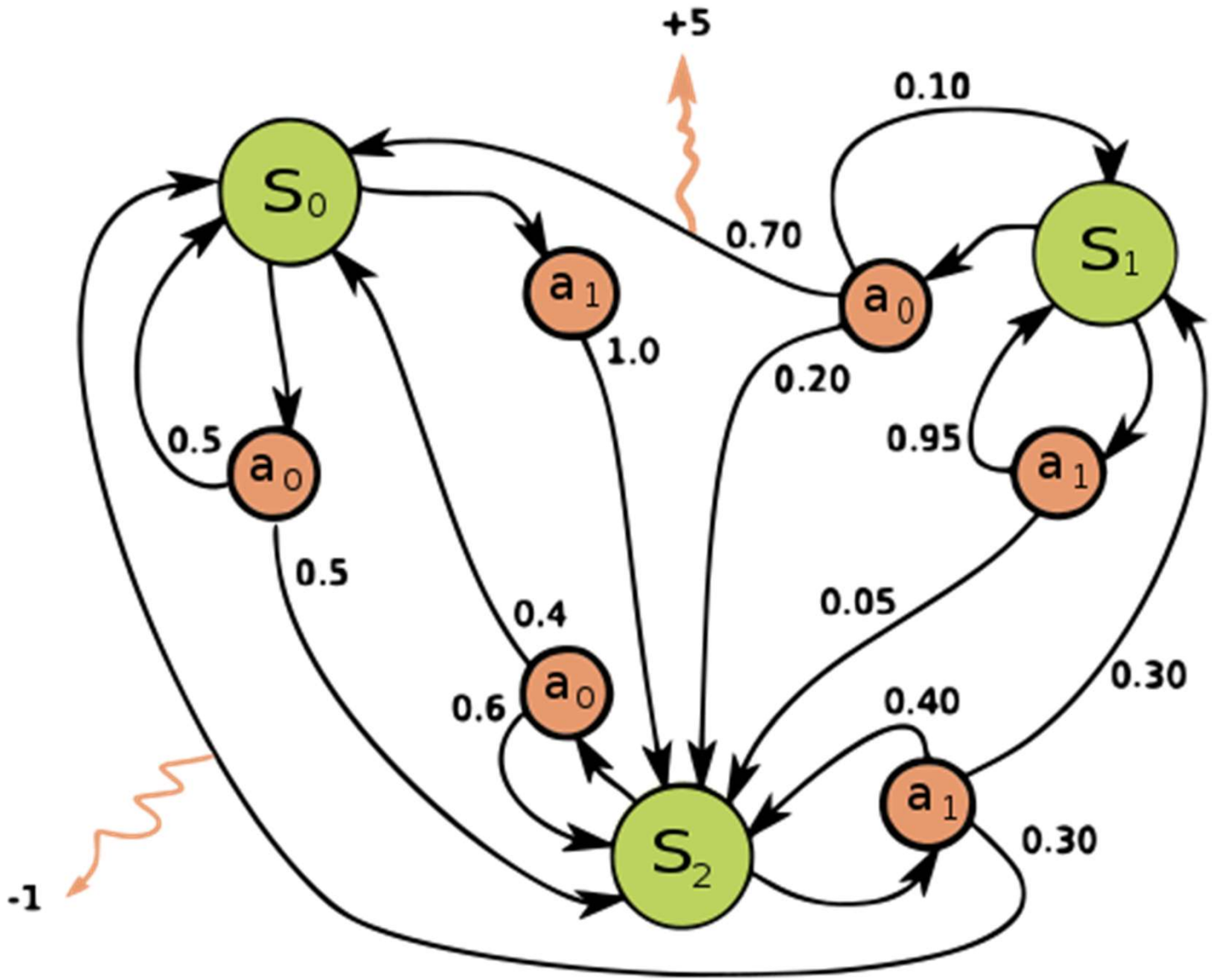
What is Markovian About MDP?

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$

$$=$$

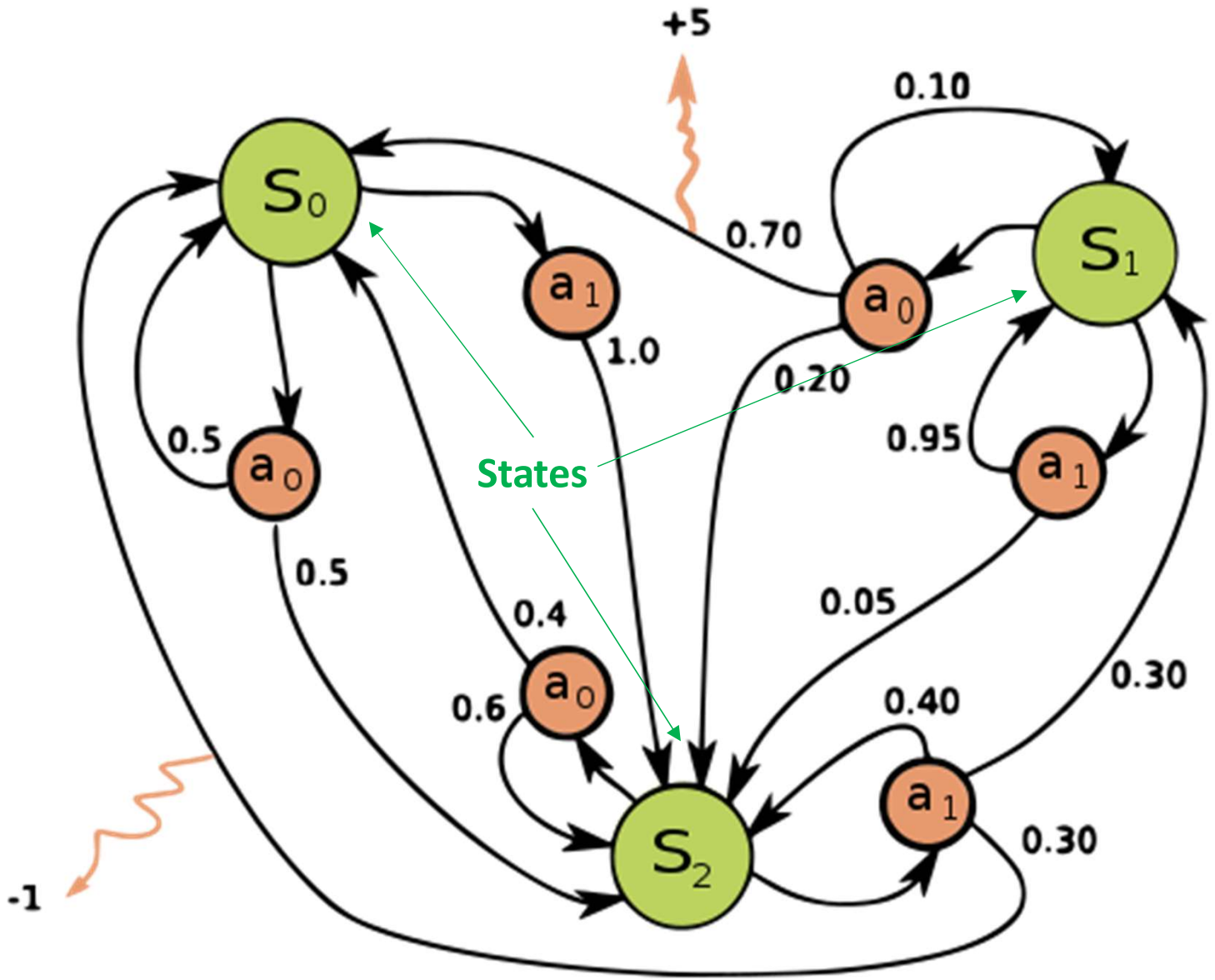
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

Markov Decision Process (MDP)



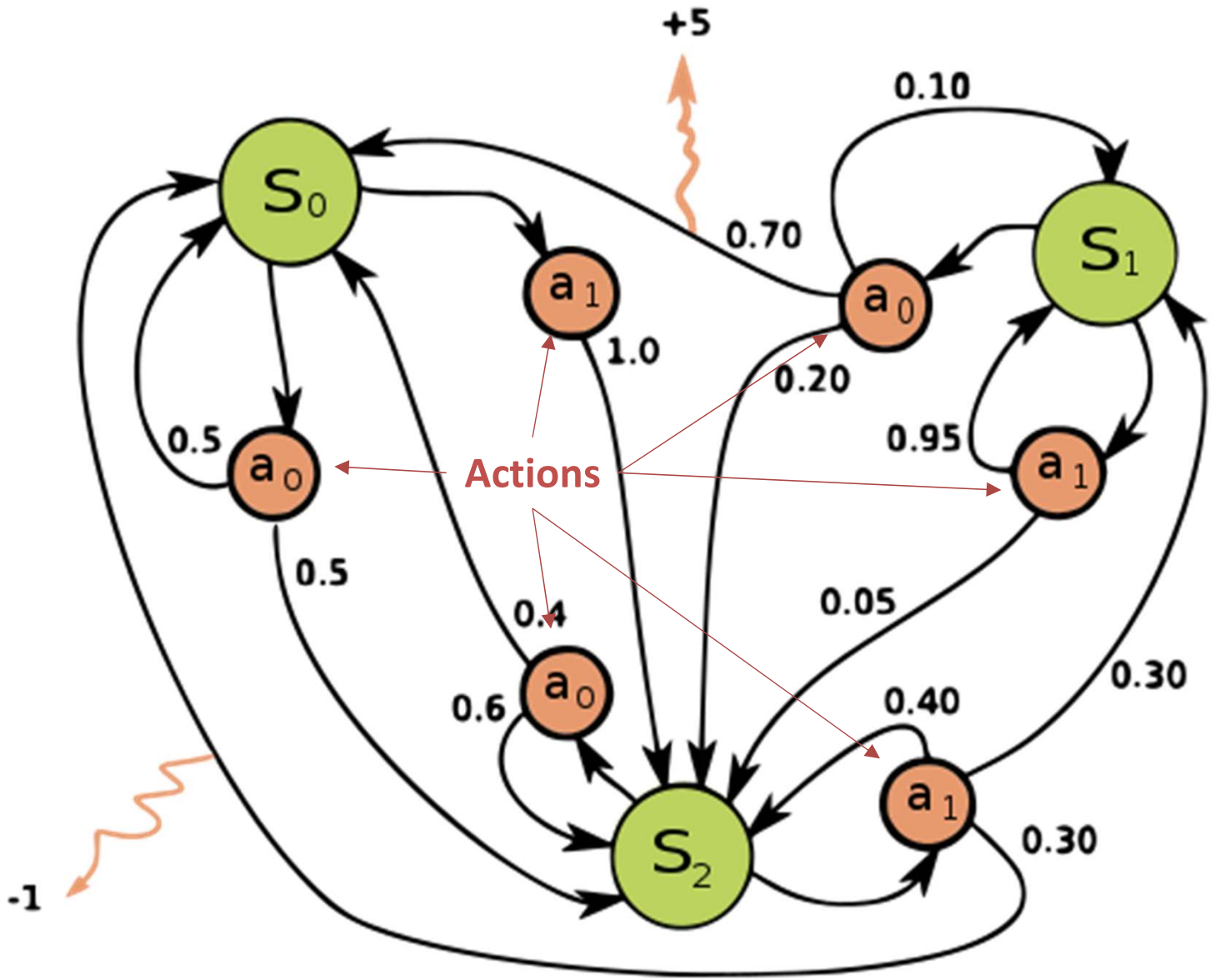
Source: Wikipedia

Markov Decision Process (MDP)



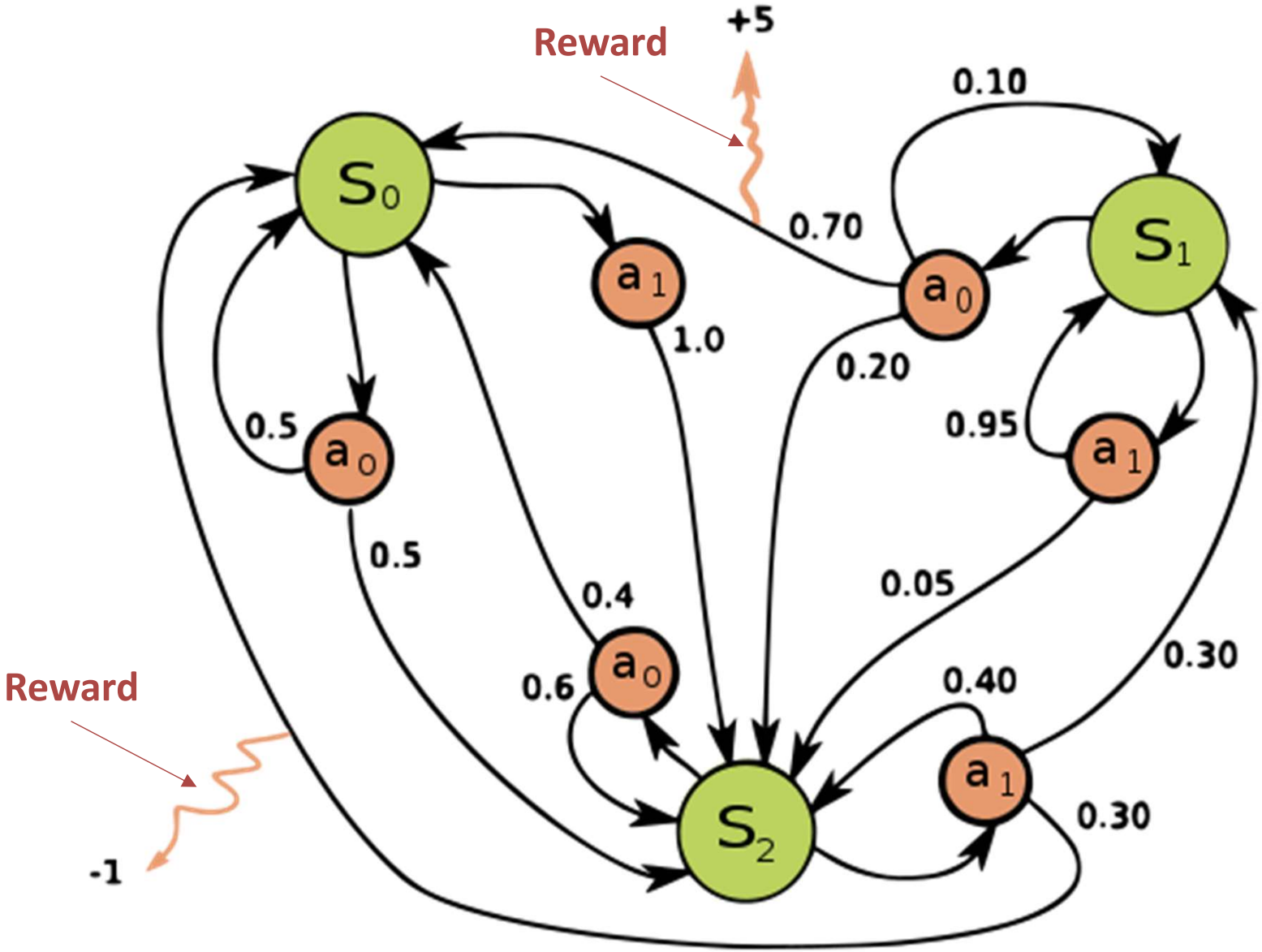
Source: Wikipedia

Markov Decision Process (MDP)



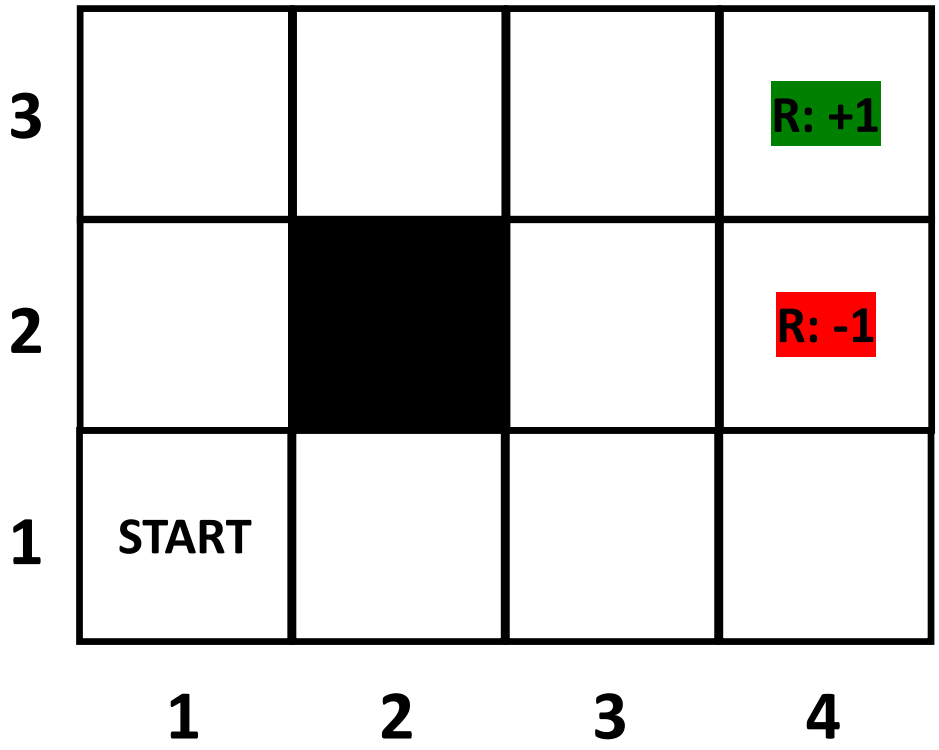
Source: Wikipedia

Markov Decision Process (MDP)

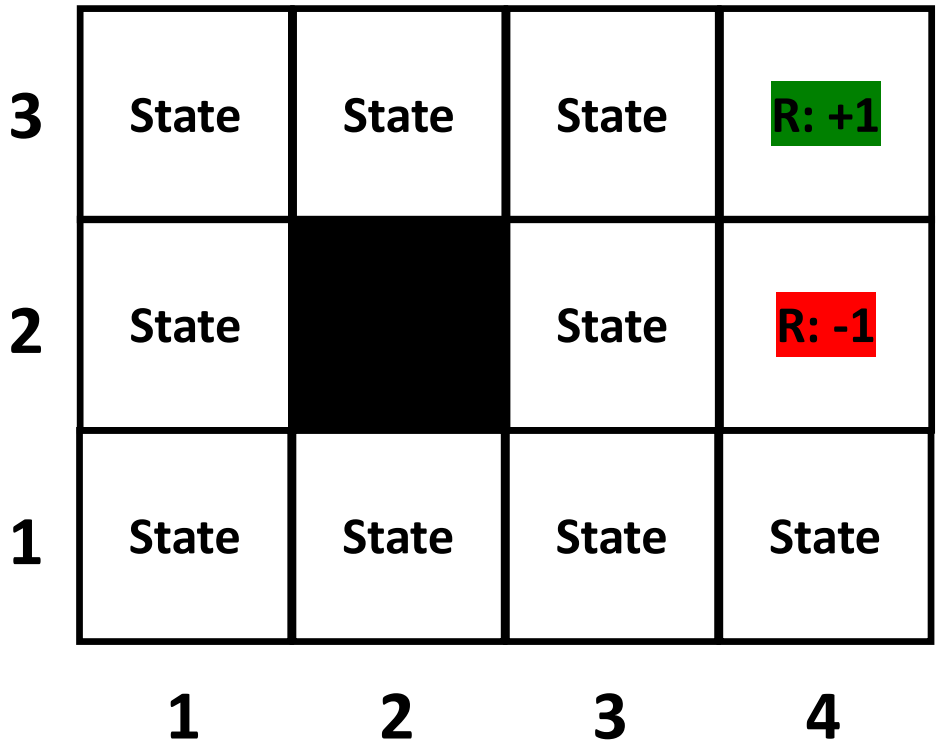


Source: Wikipedia

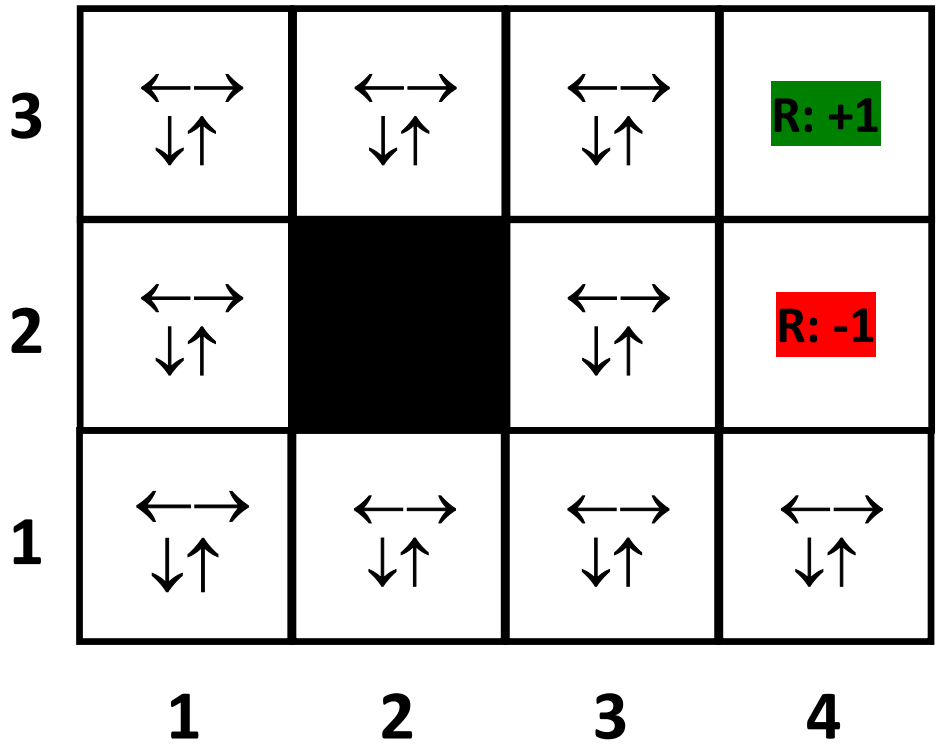
Fully Observable/Non-deterministic

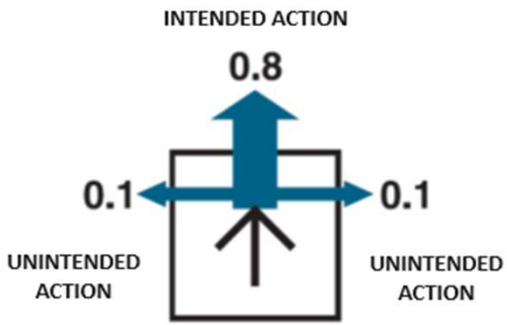


Sequential Decision Problem: States

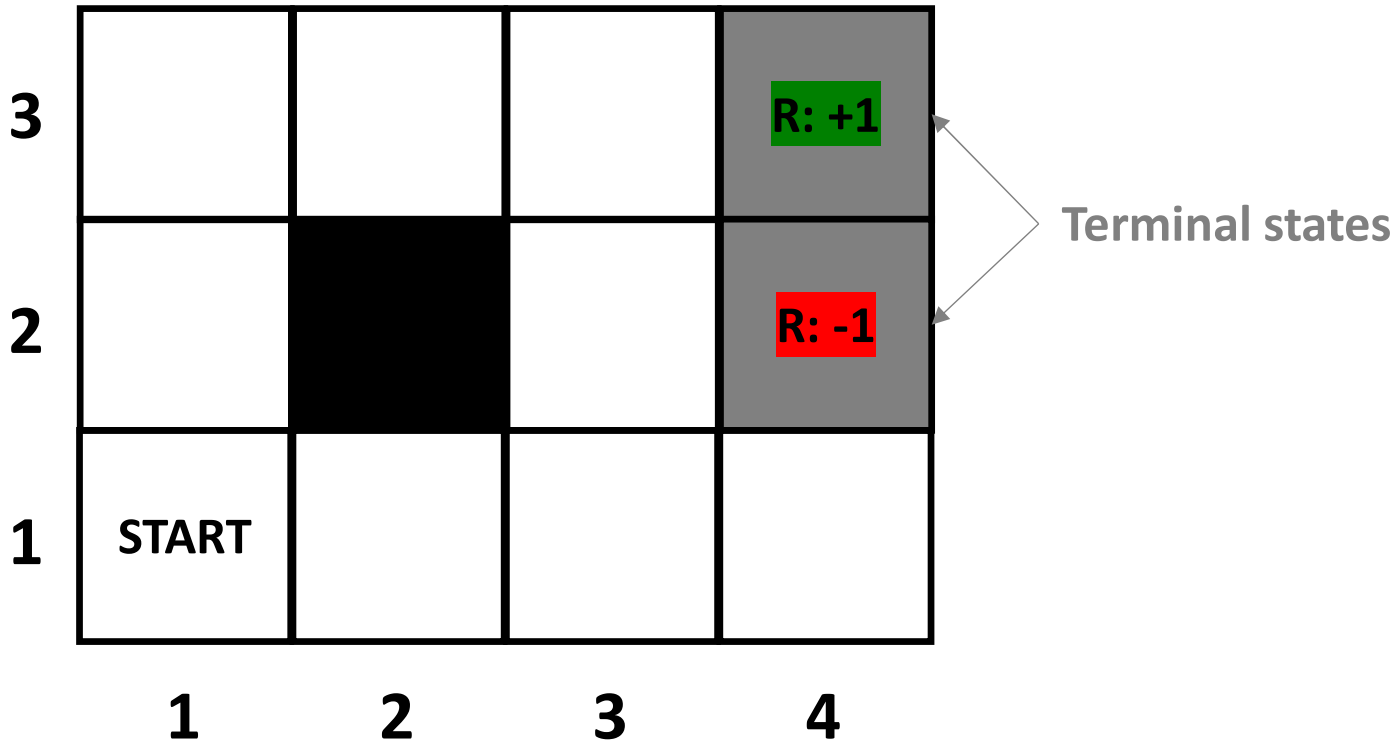


Sequential Decision Problem: Actions

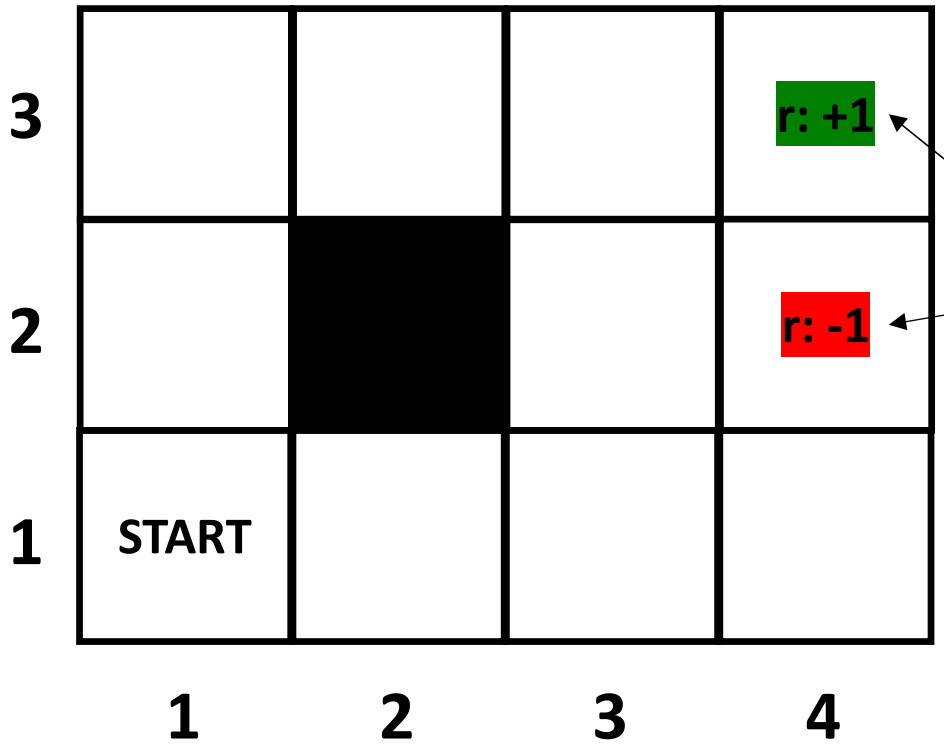




Sequential Decision Problem: Terminal



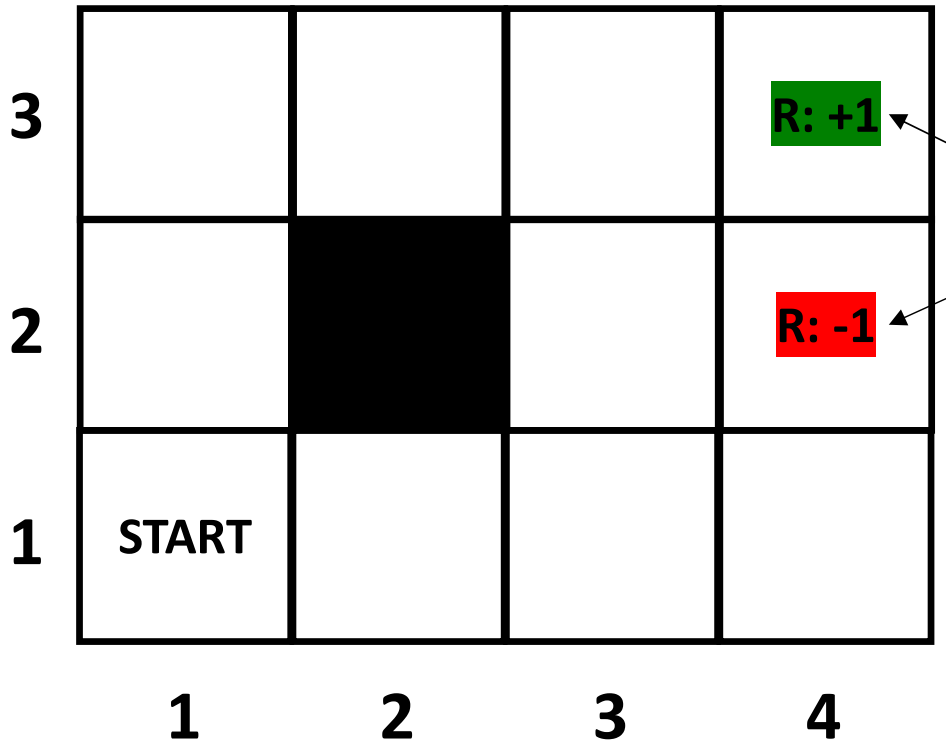
Sequential Decision Problem: Rewards



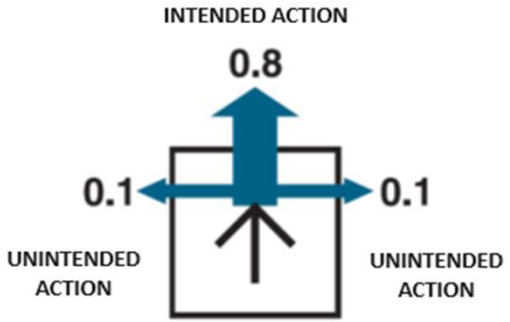
Terminal State rewards

Note: ALL other transitions have a reward r = -0.04

Fully Observable/Non-deterministic



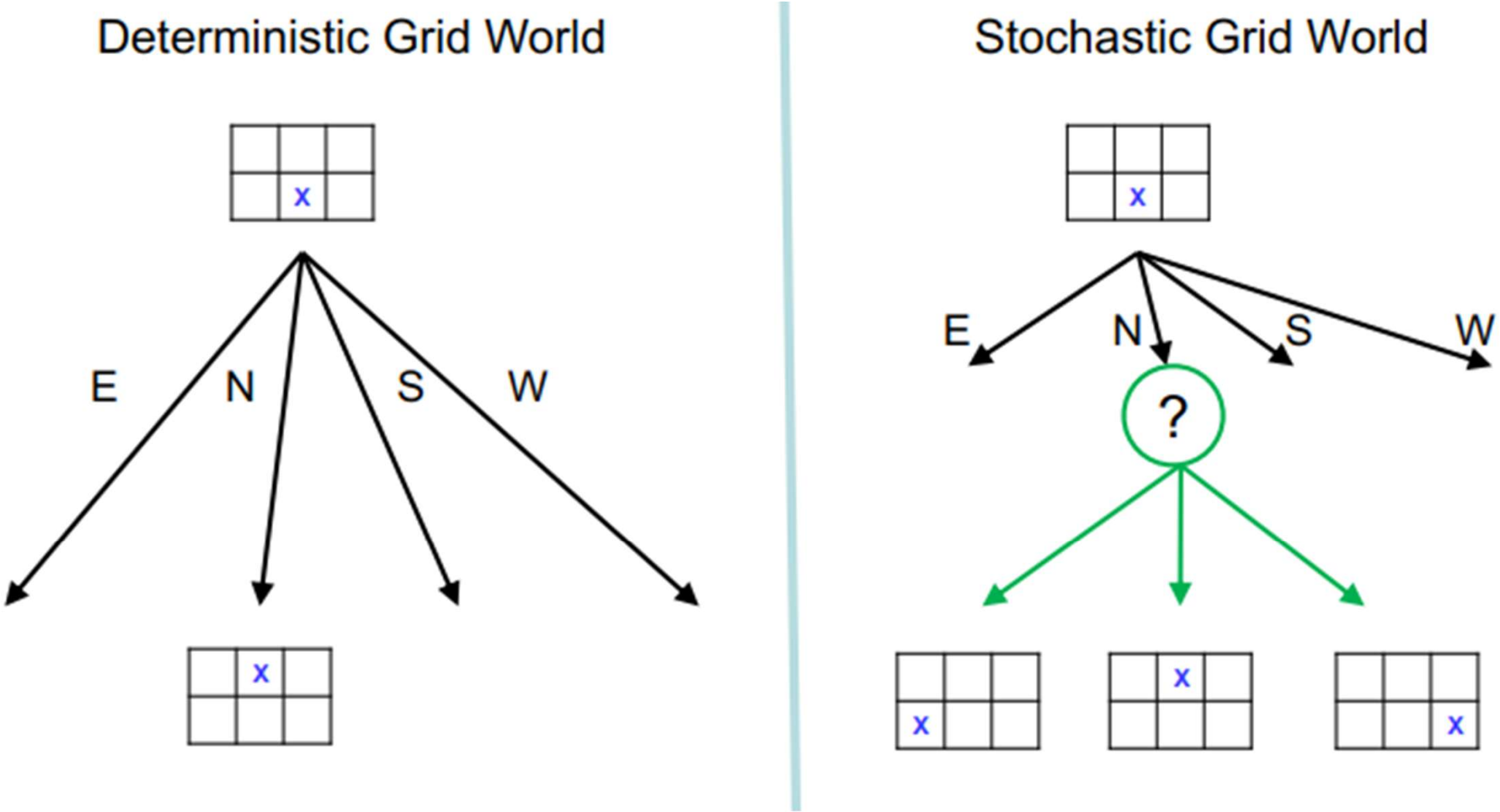
Note: ALL other transitions have a reward r = -0.04



Solution Approach

- A fixed action sequence is not the answer due to stochasticity
  - For example, [Up, Up, Right, Right, Right] is not a solution
  - It would be a solution if the environment was deterministic
- A solution must specify what the agent should do in any state that the agent might reach
  - This is called a **policy**
- Policy notation: $\boldsymbol{\pi}$
  - $\pi(s)$ specifies what action the agent should take at state s
- An **optimal policy** is the one that maximizes the expected utility → $\boldsymbol{\pi}^*$

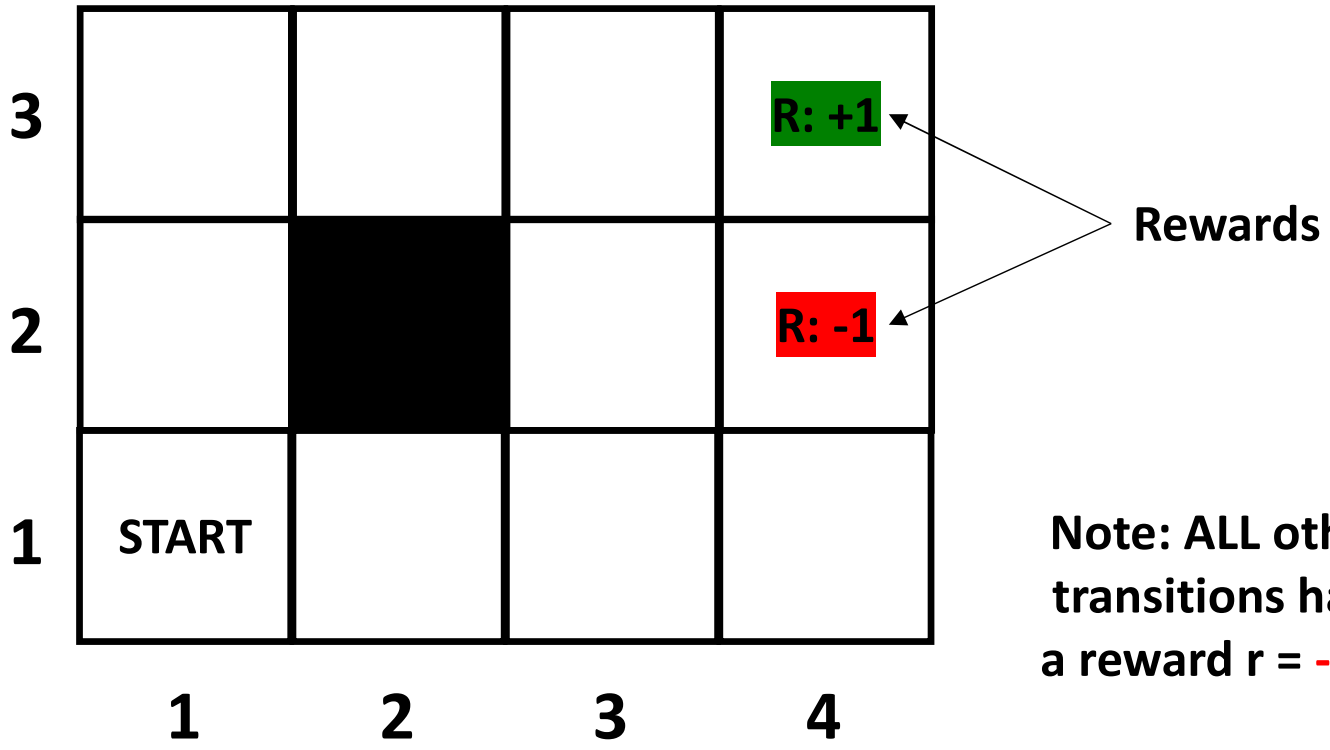
Plan vs. Policy [Solution]



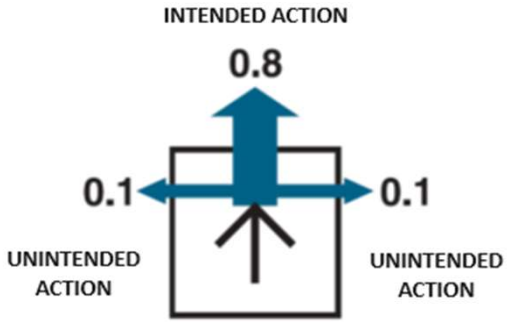
Notation

- $P(s' | s, a)$ Probability of arriving at state s' given we are at state s and take action a
- $R(s, a, s')$ The reward the agent receives when it transitions from state s to state s' via action a
- $\pi(s)$ The action recommended by policy π at state s
- π^* Optimal policy
- $U^{\pi}(s)$ The expected utility obtained via executing policy π starting at state s
- $U^{\pi^*}(s)$ is often abbreviated as $U(s)$
- $Q(s, a)$ expected utility of taking action a at state s
- γ Discount factor [0, 1]

What Are We After?



Note: ALL other transitions have a reward r = -0.04



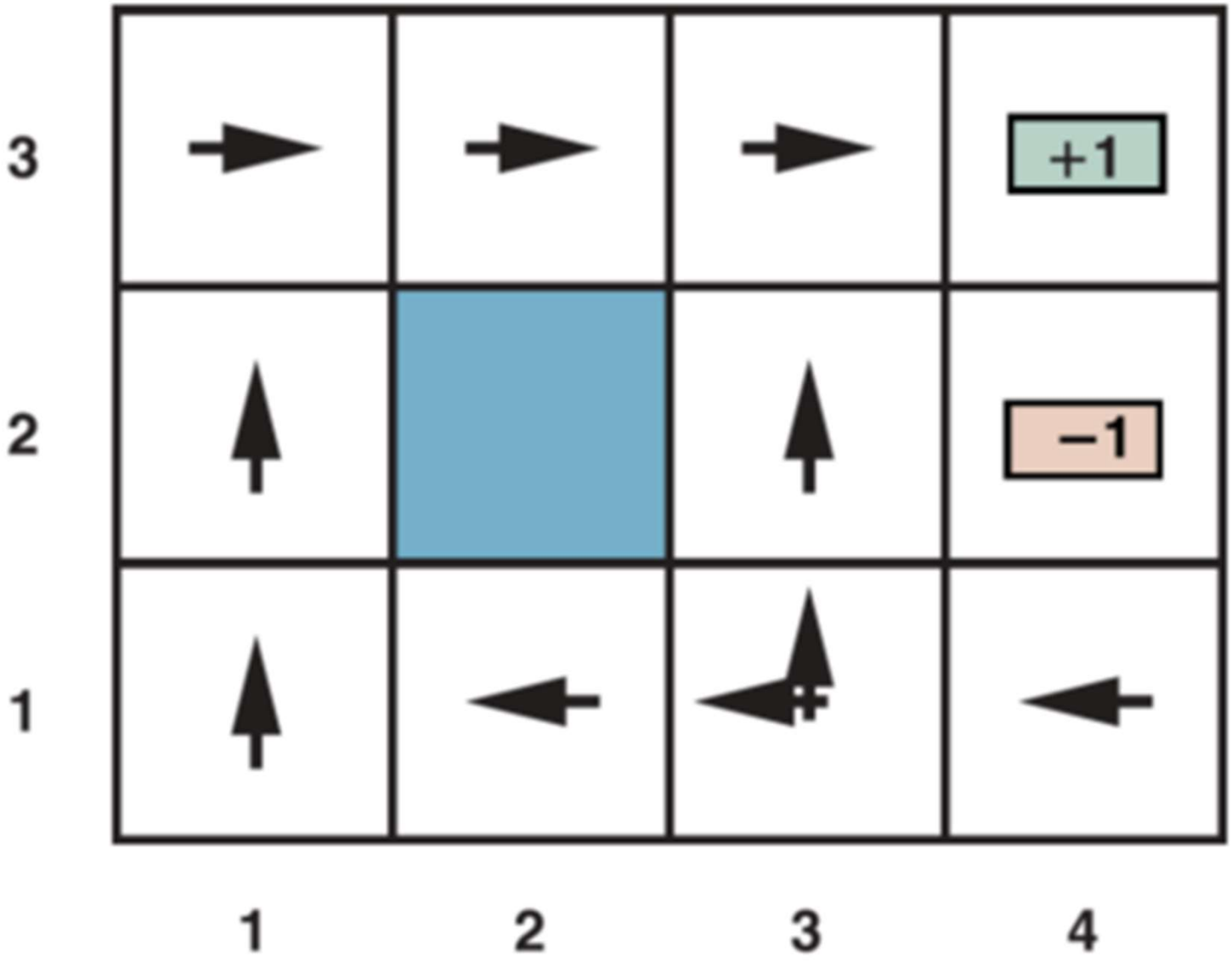
What Is the optimal policy π^*?

It **is** the mapping of actions to all reachable states that maximizes the expected utility

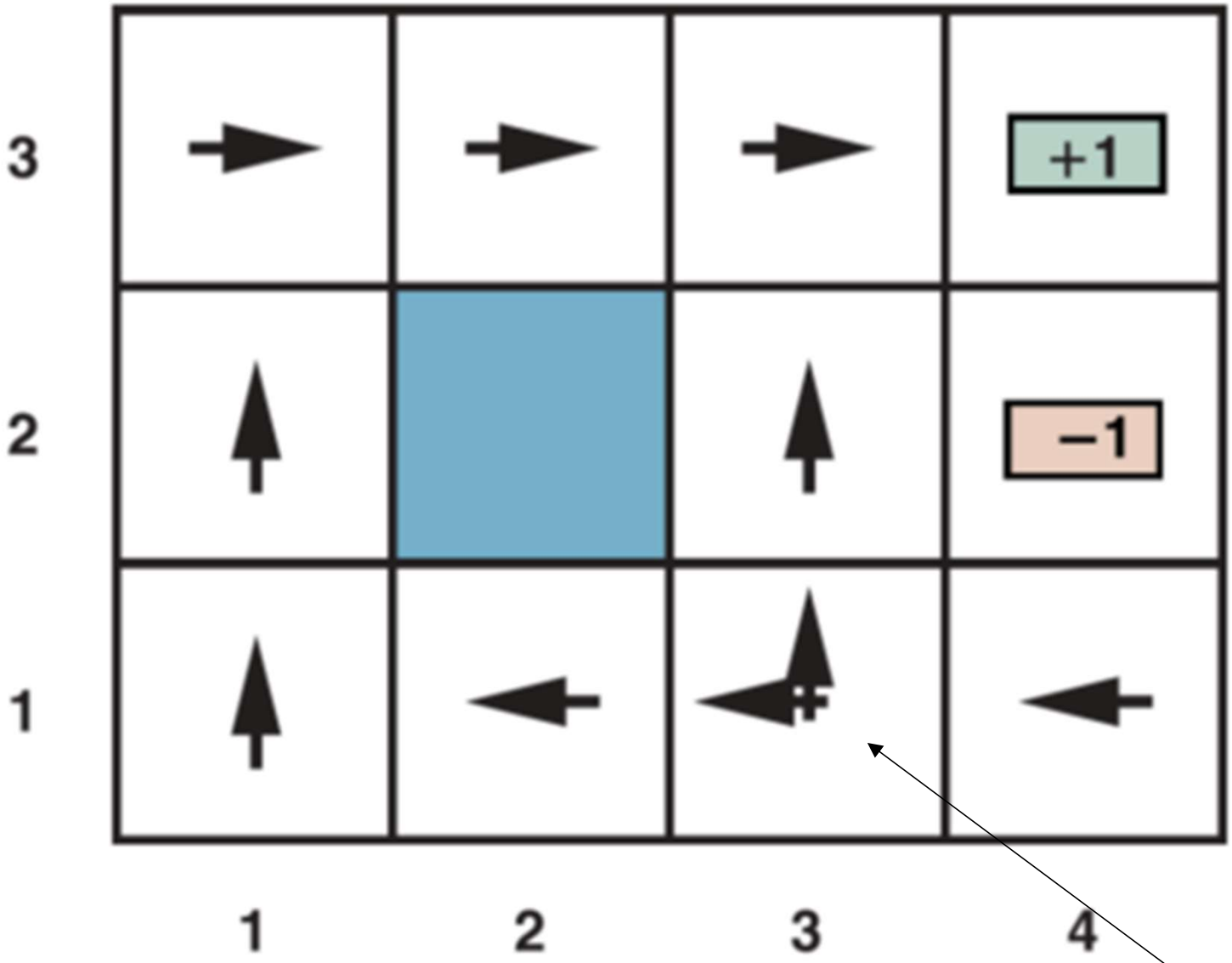
Solving MDPs

- In deterministic single-agent search problems, want an optimal plan, or sequence of actions, from start to a goal
- In an MDP, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy maximizes expected utility if followed
  - Defines a reflex agent

Optimal Policies: r = 0.04

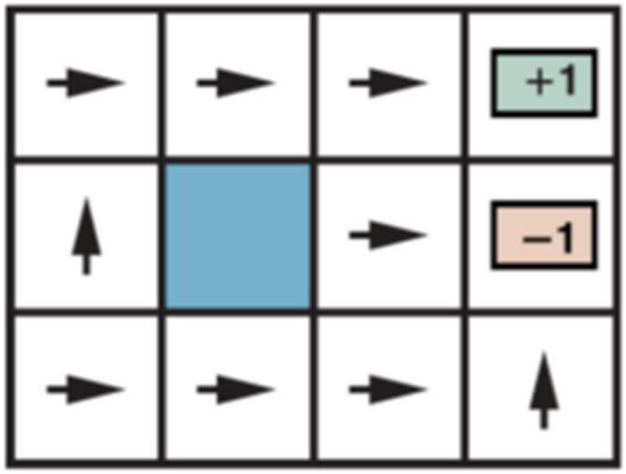


Optimal Policies: r = 0.04

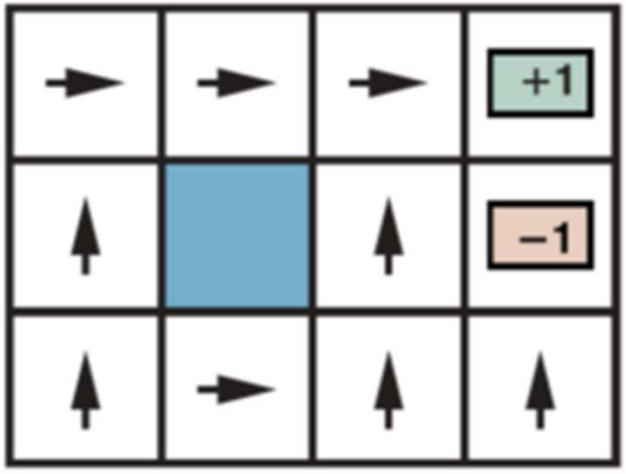


Note two EQUAL policies

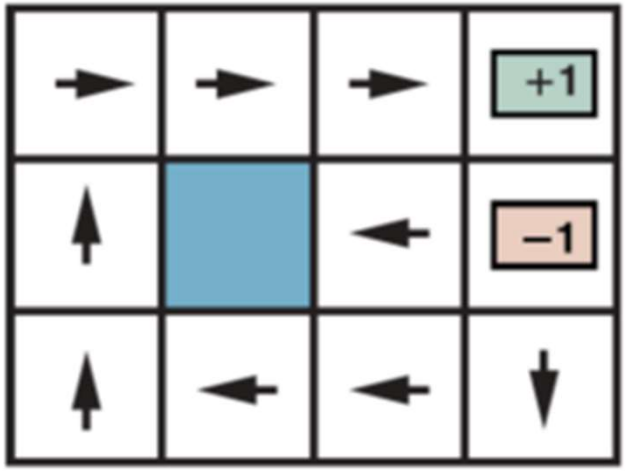
Optimal Policies: Different r Values



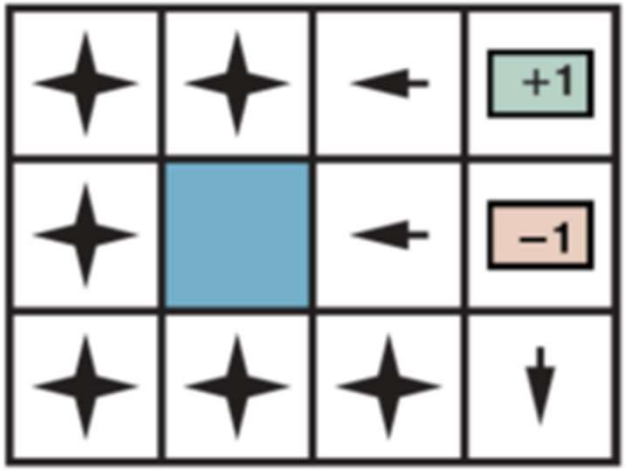
$r < -1.6497$



$-0.7311 < r < -0.4526$



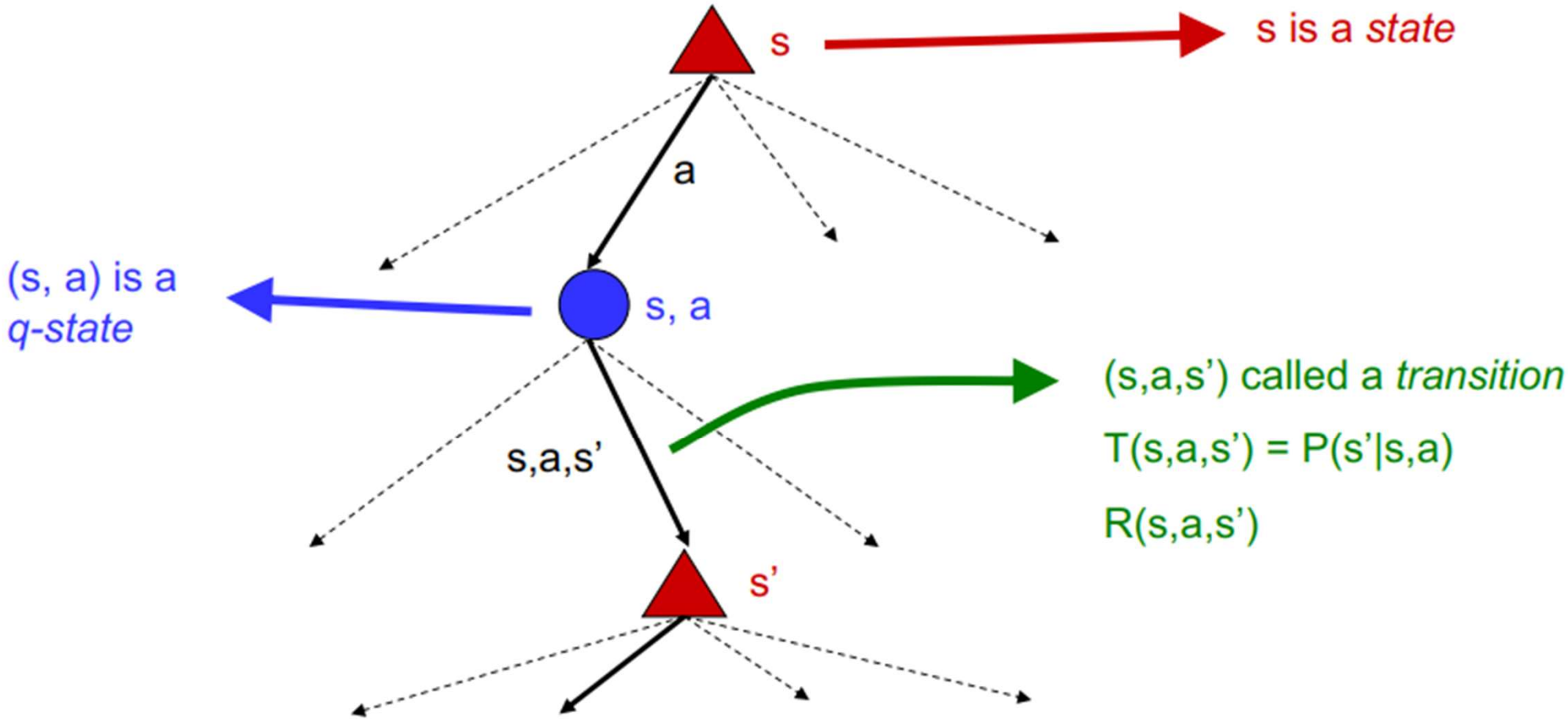
$-0.0274 < r < 0$



$r > 0$

Solve With Searching?

- Each MDP state gives an expectimax-like search tree



Utility of States

- The agent receives a reward at each state
- Utility of a state s given a policy π is the expected reward that the agent will get starting from state s and taking actions according to policy π. Let s_t denote the state that the agent reaches at time t

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t * R(s_t, \pi(s_t), s_{t+1})\right]$$

- The expectation is with respect to the transition probabilities

Utility vs. Reward

- R(s, a, s') is the short-term immediate reward the agent receives when it transitions from state s to state s' via action a
- U(s) is the long-term cumulative reward from s onward
- Utility of state s given policy π:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^{t} * R(S_t, \pi(S_t), S_{t+1})\right]$$

- γ^{t} - discount factor at time t

State Utility and Optimal Policy

- **Utility of state s given policy π:**

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^{t} * R(S_t, \pi(S_t), S_{t+1})\right]$$

- **There can be multiple policies available to the agent when at state s. Optimal policy π^* at s:**

$$\pi_s^* = \underset{\pi}{\operatorname{argmax}}\, U^{\pi}(s)$$

Utilities of Sequences of Rewards

- In order to formalize optimality of a policy, need to understand utilities of sequences of rewards
- Typically consider stationary preferences:

$$[r, r_0, r_1, r_2, \ldots] \succ [r, r'_0, r'_1, r'_2, \ldots]$$

$$\Leftrightarrow$$

$$[r_0, r_1, r_2, \ldots] \succ [r'_0, r'_1, r'_2, \ldots]$$

Preference symbol

- Theorem: only two ways to define stationary utilities
  - Additive utility:

$$U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$$

  - Discounted utility:

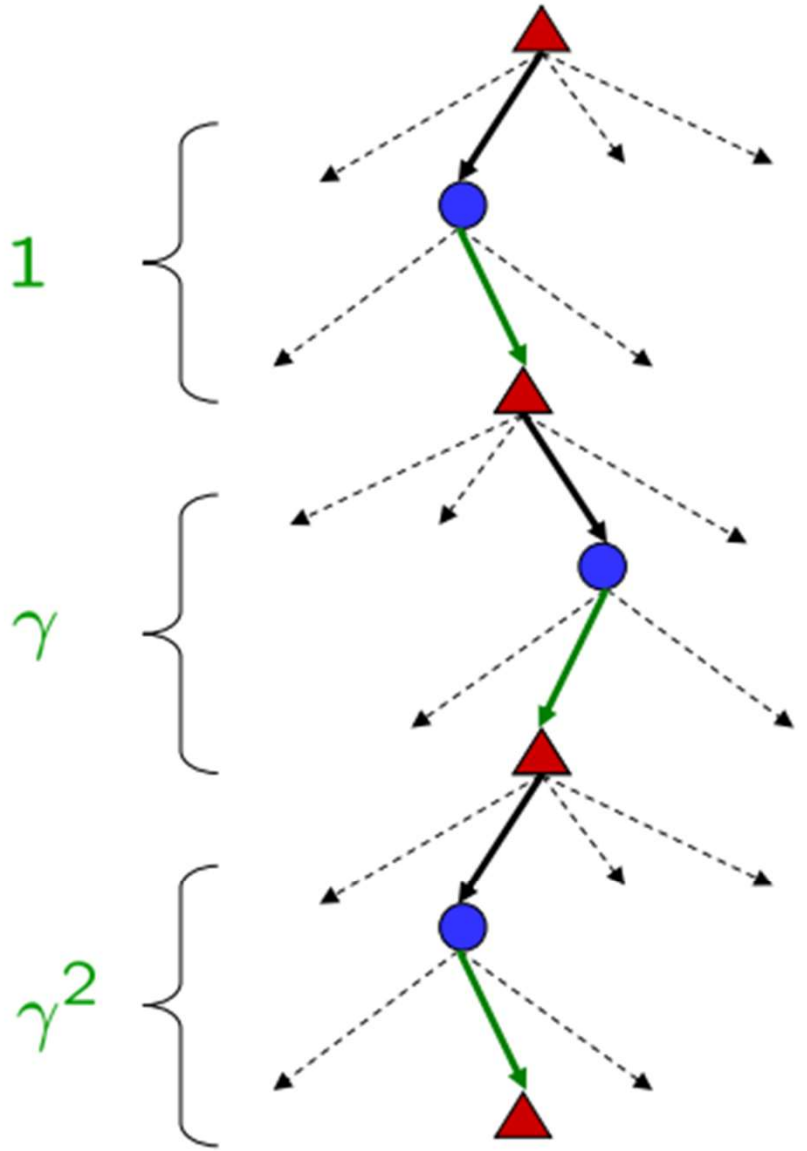
$$U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$$

Discount Factor γ

- The discount factor γ is a number between 0 and 1.
- The discount factor describes the preference of an agent for current rewards over future rewards.
- When γ is close to 0, rewards in the distant future are viewed as insignificant.
- When γ is 1, discounted rewards are exactly equivalent to additive rewards, so additive rewards are a special case of discounted rewards.
- Discounting appears to be a good model of both animal and human preferences over time.

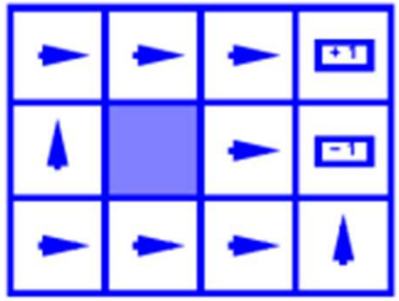
Discounting

- Typically discount rewards by $\gamma < 1$ each time step
  - Sooner rewards have higher utility than later rewards
  - Also helps the algorithms converge



Infinite Utilities

- Problem: infinite state sequences have infinite rewards



- Solutions:
  - Finite horizon:
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives nonstationary policies (π depends on time left)
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached
  - Discounting: for $0 < \gamma < 1$

$$U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max}/(1-\gamma)$$

    - Smaller γ means smaller "horizon" – shorter term focus

Infinite Discounted Sequence/Finite U

With discounted rewards, the utility of an infinite sequence is finite.

In fact, if $\gamma < 1$ and rewards are bounded by ± R_{max}, we have

$$U_h([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = R_{\max}/(1-\gamma)$$

using the standard formula for the sum of an infinite geometric series

Proper Policy

If the environment contains terminal states and if the agent is guaranteed to get to one eventually, then we will never need to compare infinite sequences.

A policy that is guaranteed to reach a terminal state is called a proper policy. With proper policies, we can use γ = 1 (i.e., additive rewards)

Average Reward [Infinite Sequences]

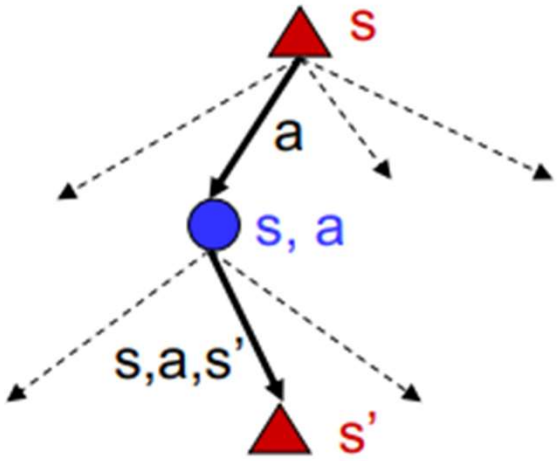
Infinite sequences can be compared in terms of the average reward obtained per time step.

Suppose that square (1,1) in the 4 x 3 world has a reward of 0.1 while the other nonterminal states have a reward of 0.01.

Then a policy that does its best to stay in (1,1) will have higher average reward than one that stays elsewhere.

MDP Recap

- Markov decision processes:
  - States S
  - Start state s_0
  - Actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)



- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility (or return) = sum of discounted rewards

Expected Utility Given Policy π

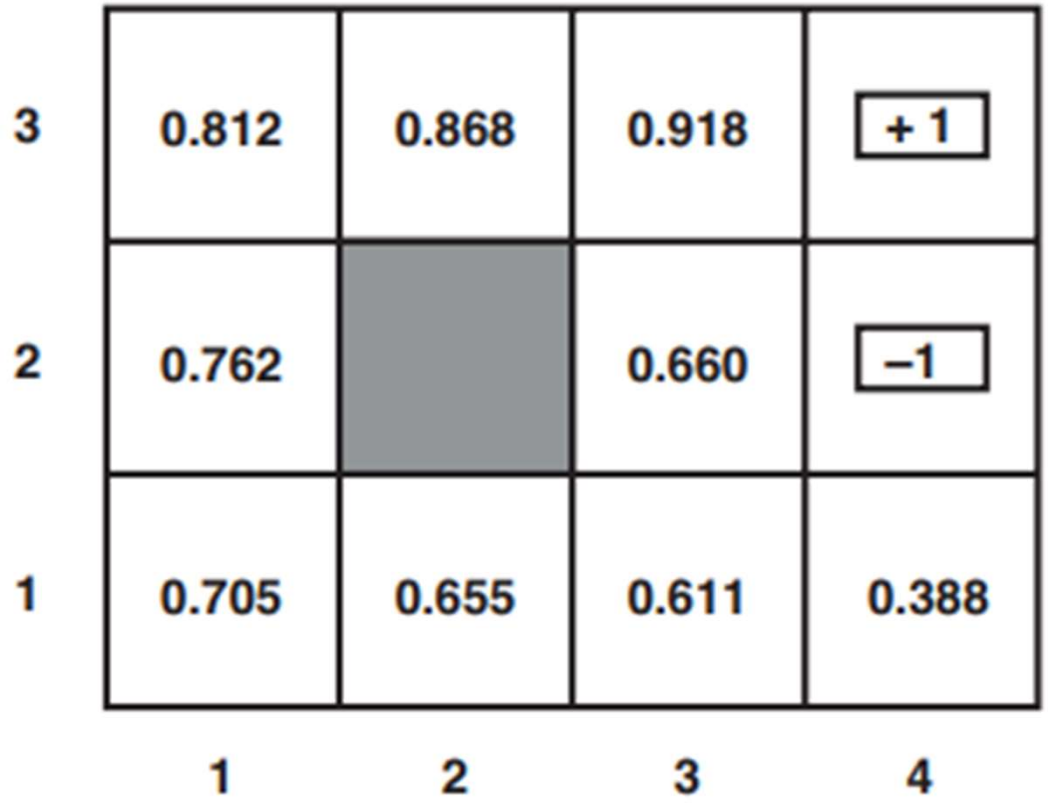
$$U^{\pi}(s) = \sum_{s'} P(s' \mid s, \pi(s)) * [R(s, \pi(s), s') + \gamma * U^{\pi}(s')]$$

Bellman Equation

The utility of a state is the expected reward for the next transition plus the discounted utility of the next state, assuming that the agent chooses the optimal action:

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) * [R(s, a, s') + \gamma * U(s')]$$

Bellman Equation: Example



Note: ALL non-terminal transitions have a reward r = -0.04

$$U(1,1) = -0.04 + \gamma \max[\ 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), \quad (Up)$$
$$0.9U(1,1) + 0.1U(1,2), \quad (Left)$$
$$0.9U(1,1) + 0.1U(2,1), \quad (Down)$$
$$0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)\ ]. \quad (Right)$$

Q-Function / Action-Utility Function

The utility of a state is the expected reward for the next transition plus the discounted utility of the next state, assuming that the agent chooses the optimal action:

$$U(s) = \max_{a \in A(s)} Q(s, a)$$

Q-Function / Action-Utility Function

The utility of a state is the expected reward for the next transition plus the discounted utility of the next state, assuming that the agent chooses the optimal action:

$$U(s) = \max_{a \in A(s)} Q(s, a)$$

From that:

$$\pi_s^* = \operatorname*{argmax}_{a \in A(s)} Q(s, a)$$

Solving MDPs

- Offline algorithms:
  - Value iteration
  - Policy iteration
  - Linear programming
- Online algorithms:
  - Approximation algorithms such as Monte Carlo planning

Value Iteration Algorithm

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

  repeat
    $U \leftarrow U'$; $\delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

Bellman Update

Iterative utility update at i+1 iteration can be calculated with:

$$U_{i+1}(s) = \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) * [R(s, a, s') + \gamma * U_i(s')]$$

Policy Iteration:

- **Start with initial policy π_0**
- **Policy iteration algorithm involves (alternates between) two steps**
  - **Policy evaluation: given a policy π_i, calculate $U_i = U^{\pi i}$, the utility of each state if π_i were to be executed**
  - **Policy improvement: calculate a new MEU policy π_{i+1}, using a one step look-ahead based on U_i**

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a) * [R(s, a, s') + \gamma * U(s')]$$

Policy Iteration Algorithm

function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

Reinforcement Learning

Main Machine Learning Categories

Supervised learning	Unsupervised learning	Reinforcement learning
Supervised learning is one of the most common techniques in machine learning. It is based on **known relationship(s) and patterns within data** (for example: relationship between inputs and outputs). Frequently used types: **regression**, and **classification**.	**Unsupervised learning** involves finding underlying patterns within data. Typically used in **clustering** data points (similar customers, etc.)	Reinforcement learning is inspired by behavioral psychology. It is **based on a rewarding / punishing an algorithm**. Rewards and punishments are based on algorithm's action within its environment.

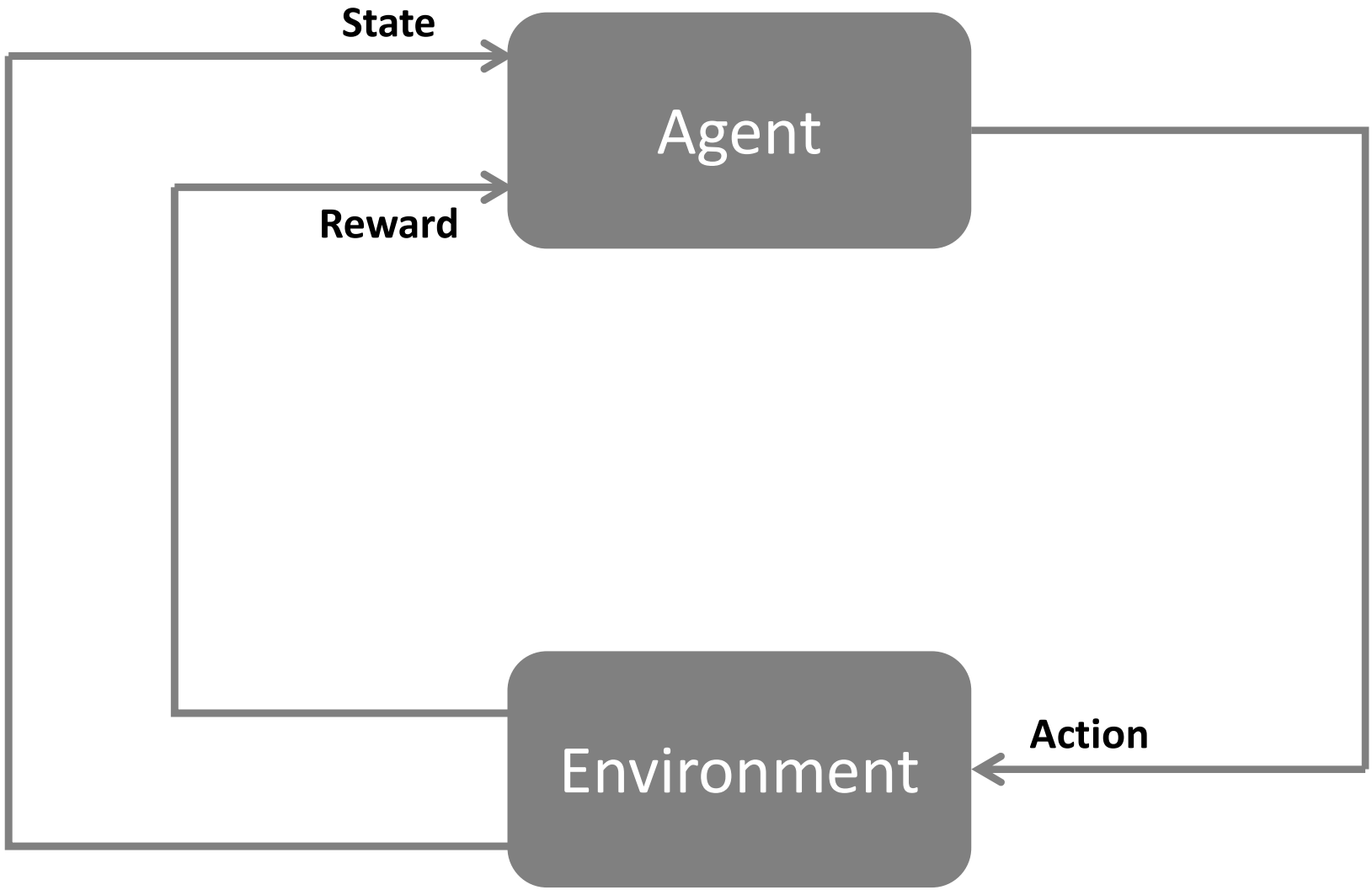
What is Reinforcement Learning?

Idea:

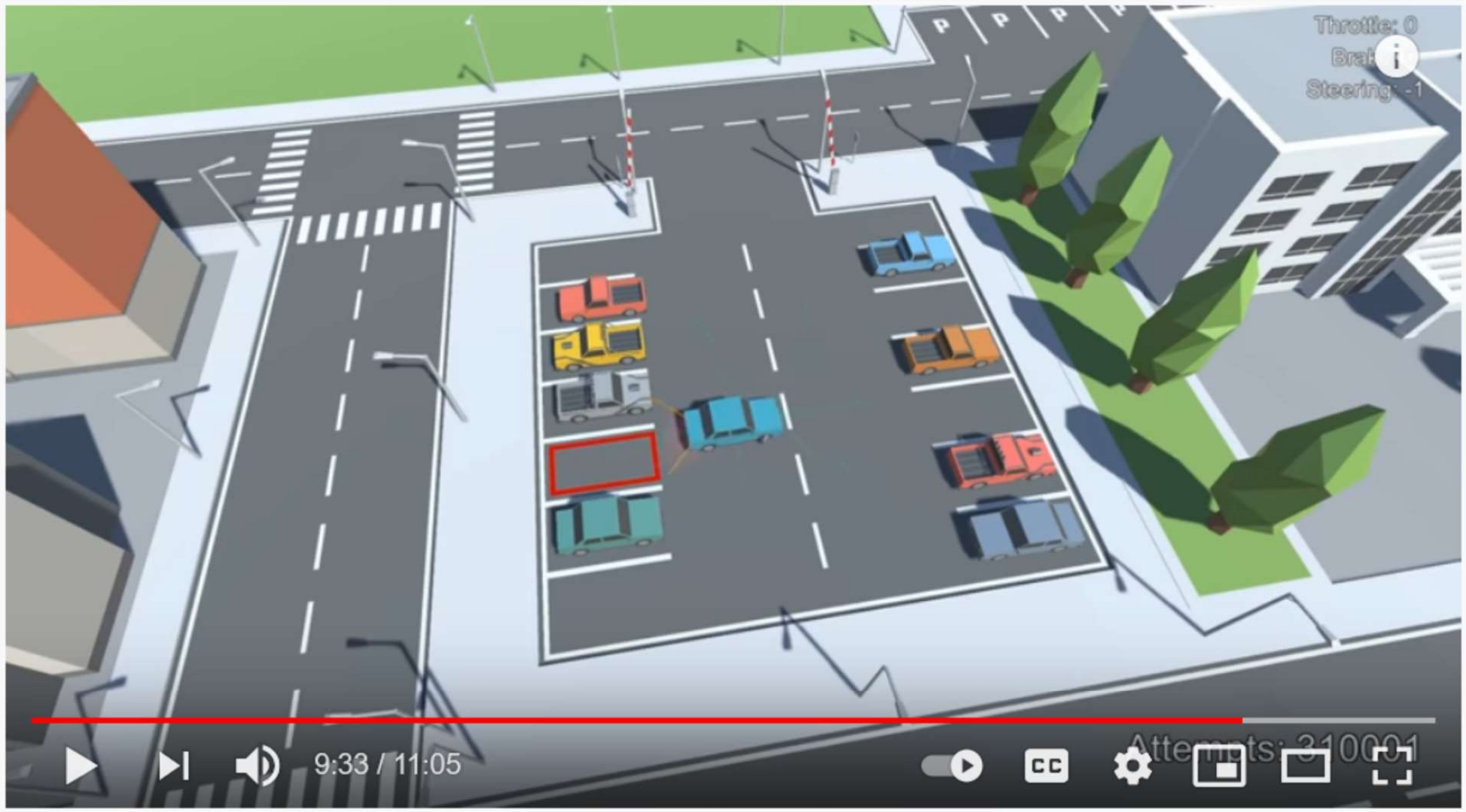
Reinforcement learning is inspired by behavioral psychology. It is **based on a rewarding / punishing an algorithm**.

Rewards and punishments are based on algorithm's action within its environment.

RL: Agents and Environments



Reinforcement Learning in Action

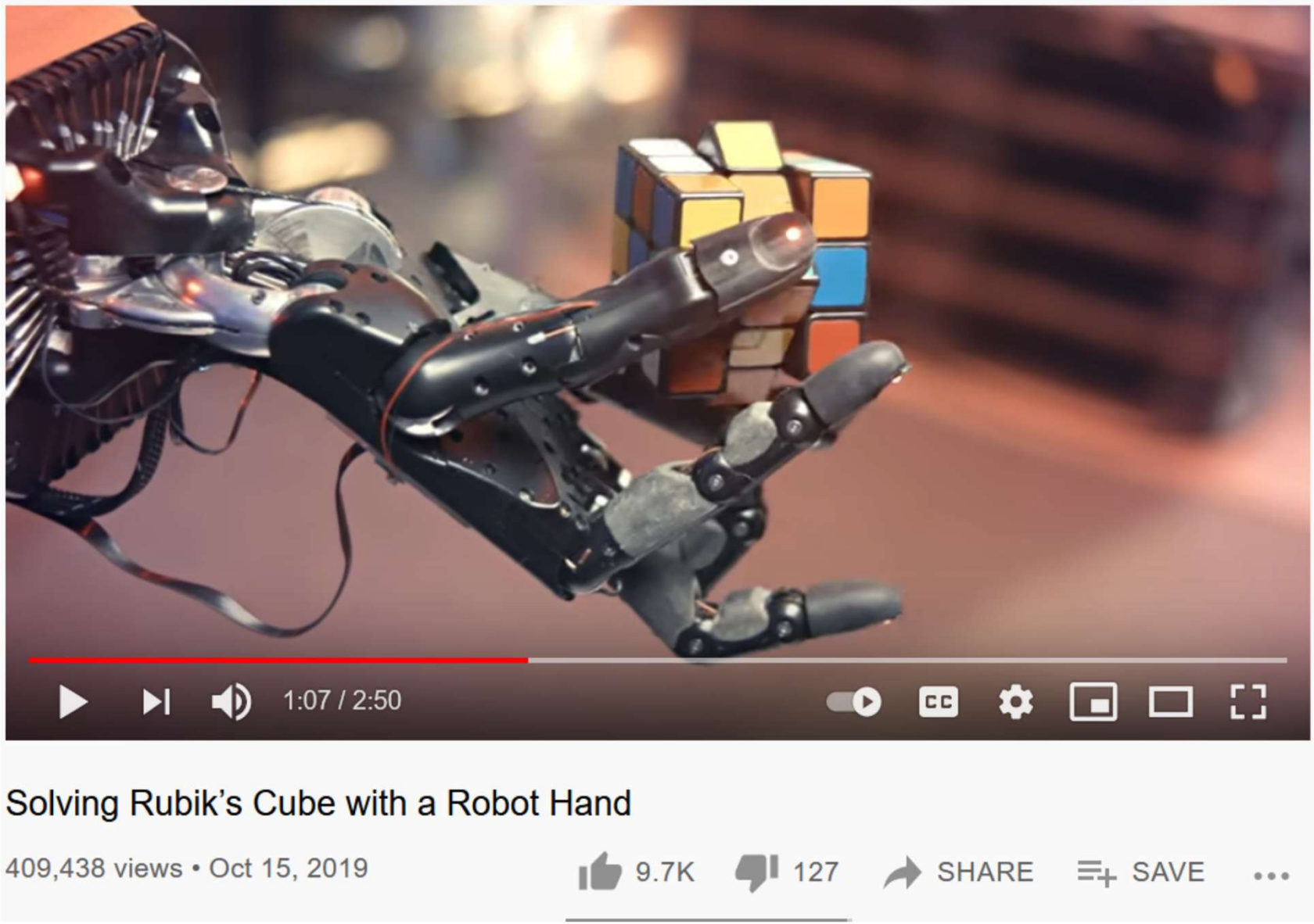


#ArtificialIntelligence #MachineLearning #ReinforcementLearning

AI Learns to Park - Deep Reinforcement Learning

1,744,342 views • Aug 23, 2019 28K 1.1K SHARE SAVE

Reinforcement Learning in Action

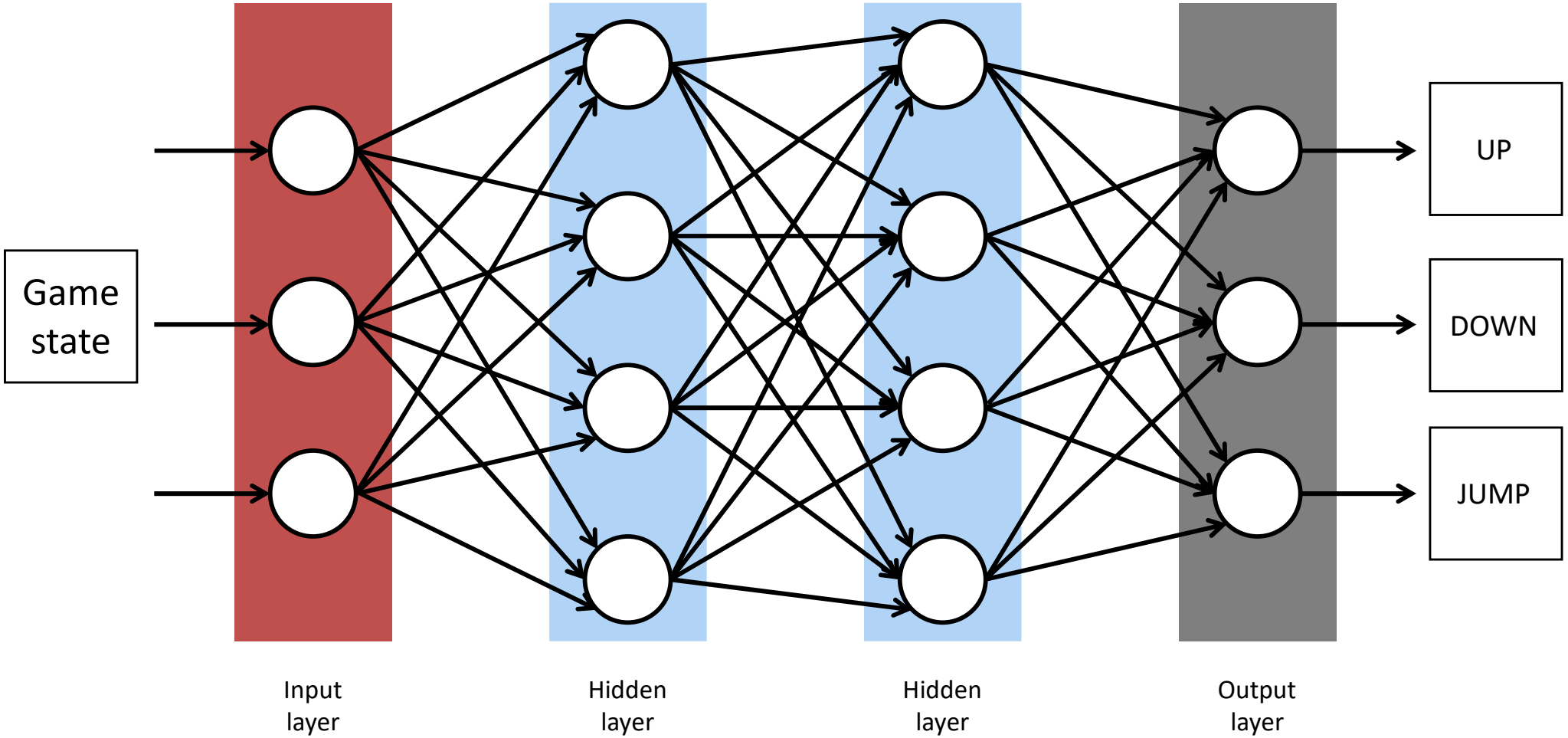


Source: https://www.youtube.com/watch?v=x4O8pojMF0w

Reinforcement Learning in Action

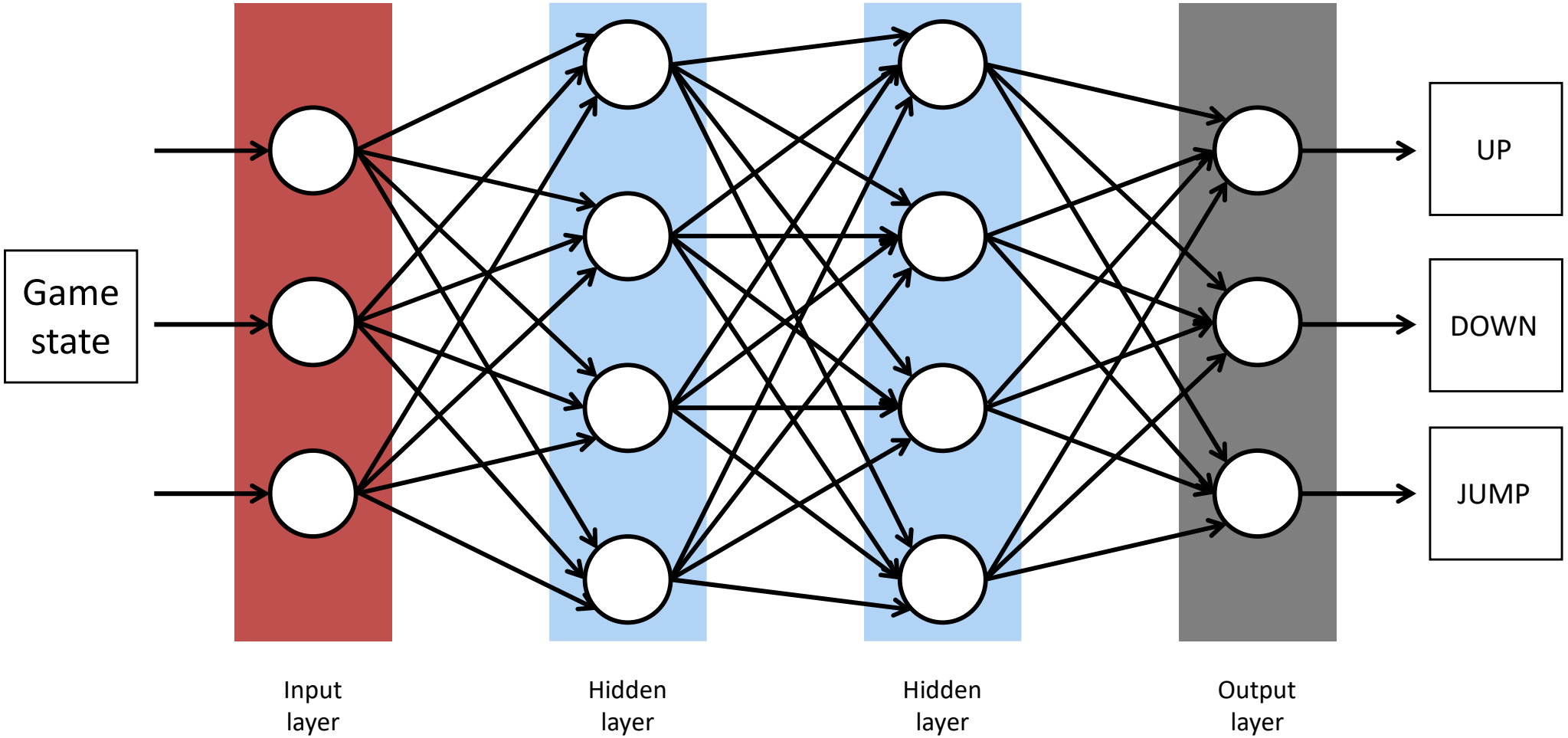
Source: https://www.youtube.com/watch?v=kopoLzvh5jY

ANN for Simple Game Playing



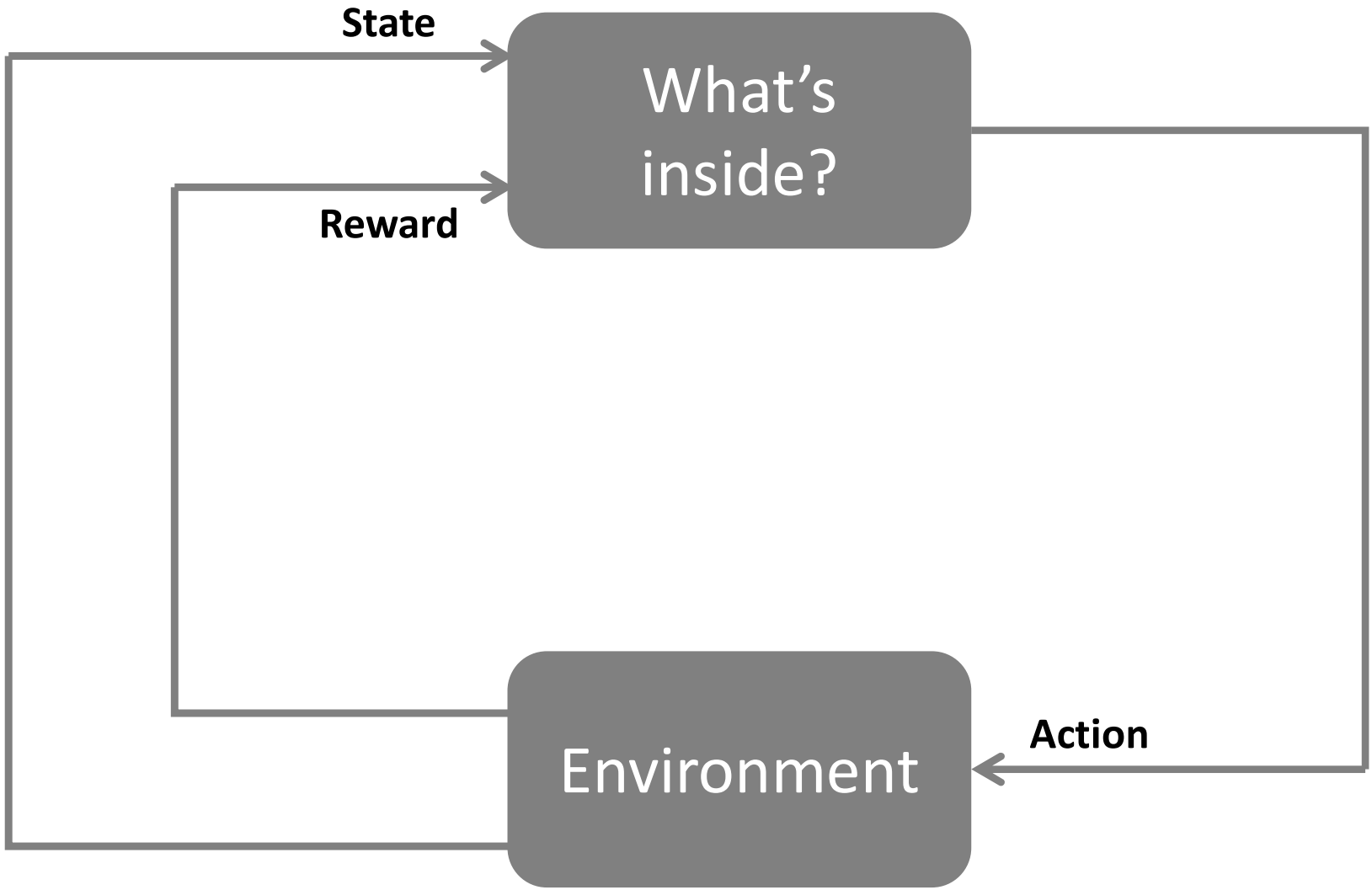
ANN for Simple Game Playing

Current game is an input. Decisions (UP/DOWN/JUMP) are rewarded/punished.

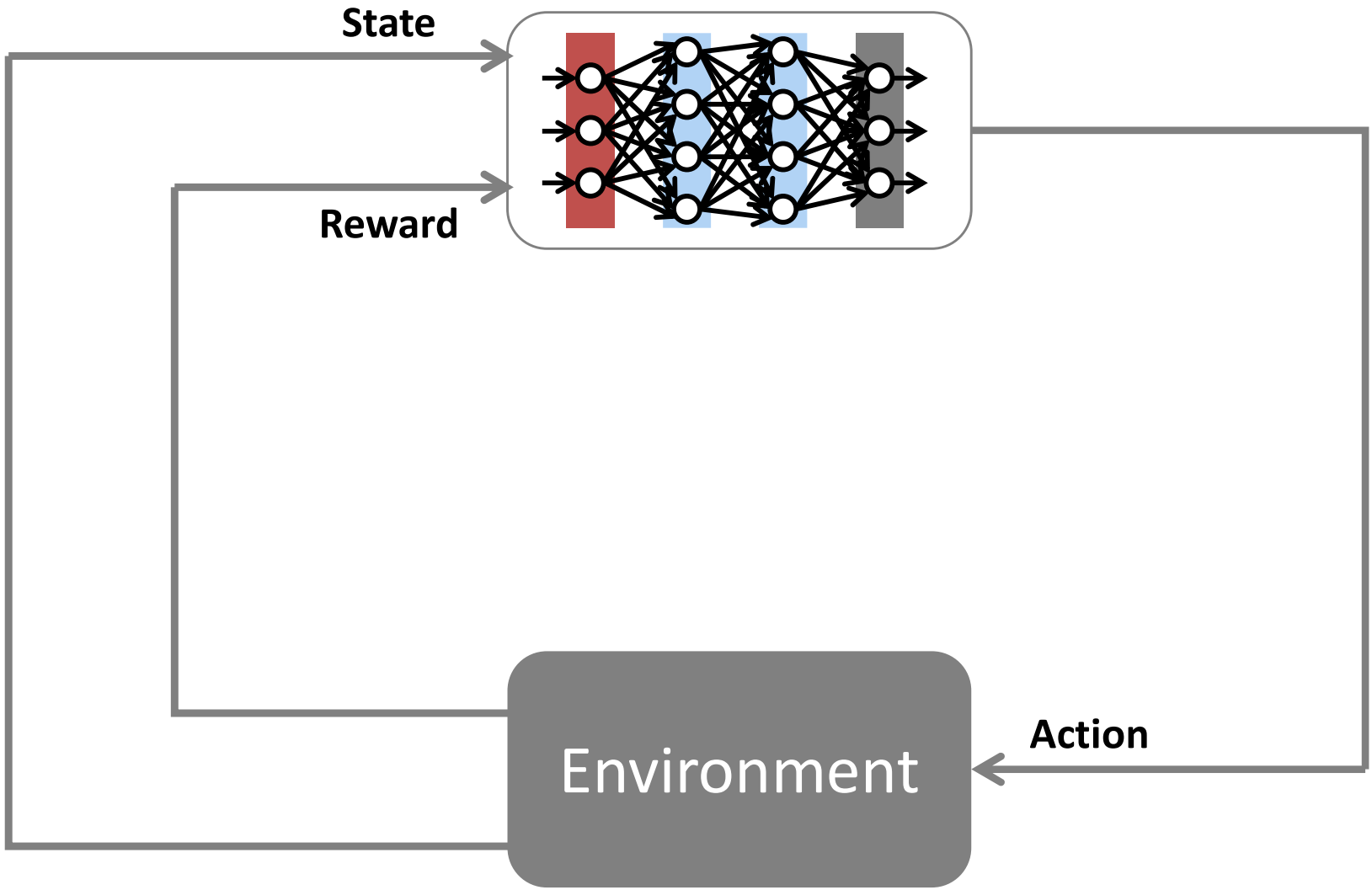


Correct all the weights using Reinforcement Learning.

RL: Agents and Environments



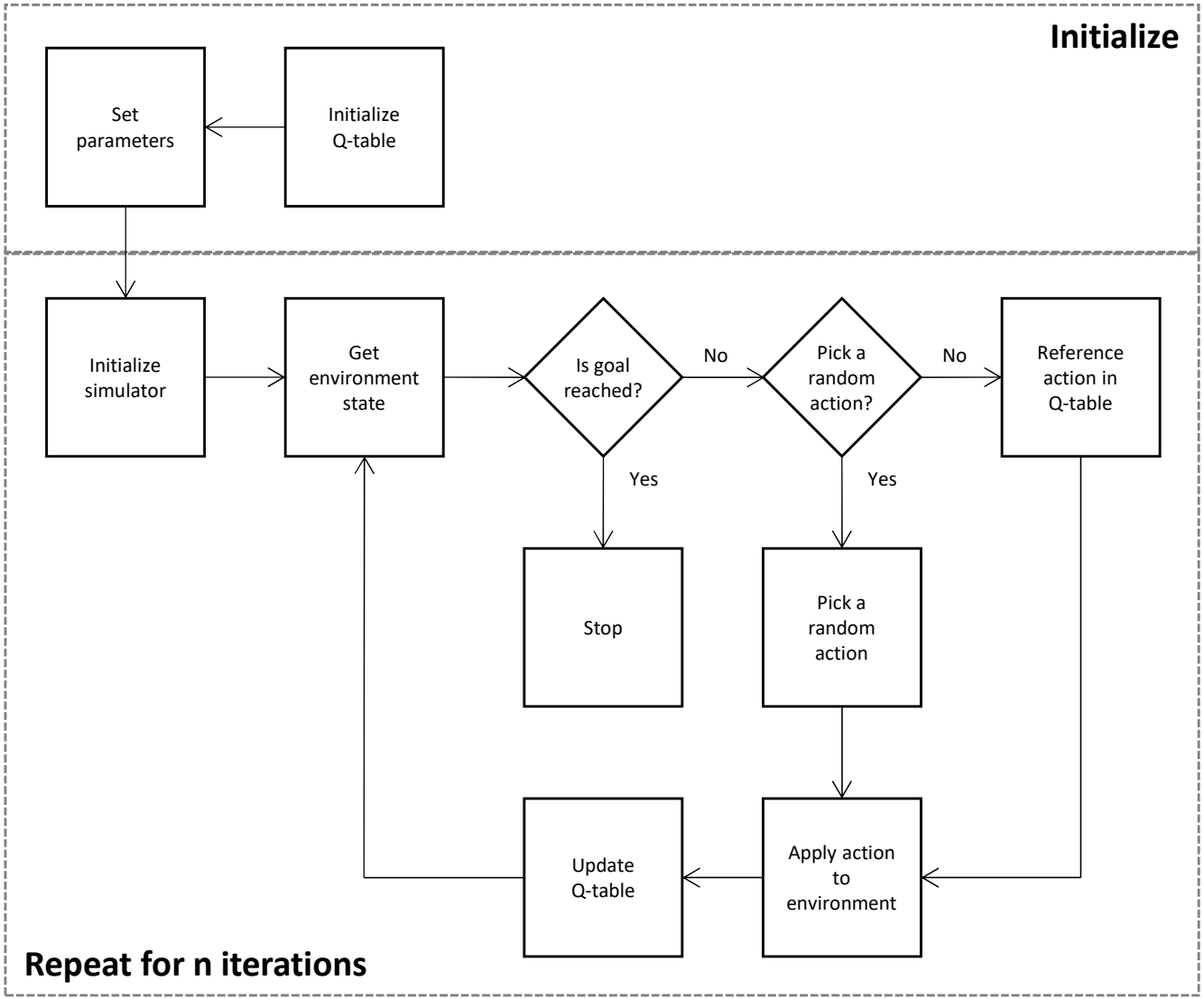
RL: Agents and Environments



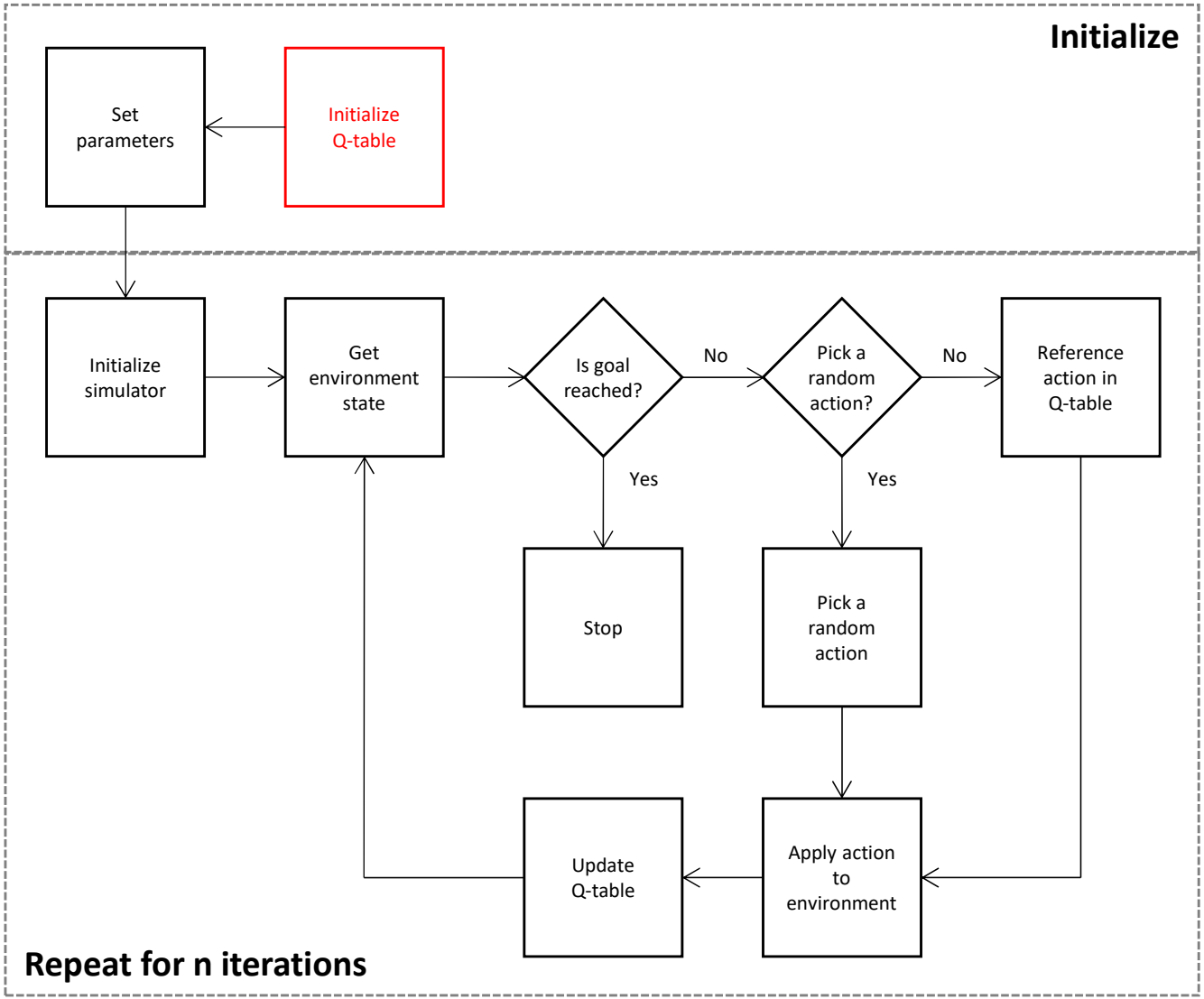
Reinforcement Learning in Action

Source: https://www.youtube.com/watch?v=Tnu4O_xEmVk

Q-Learning Algorithm



Q-Learning Algorithm

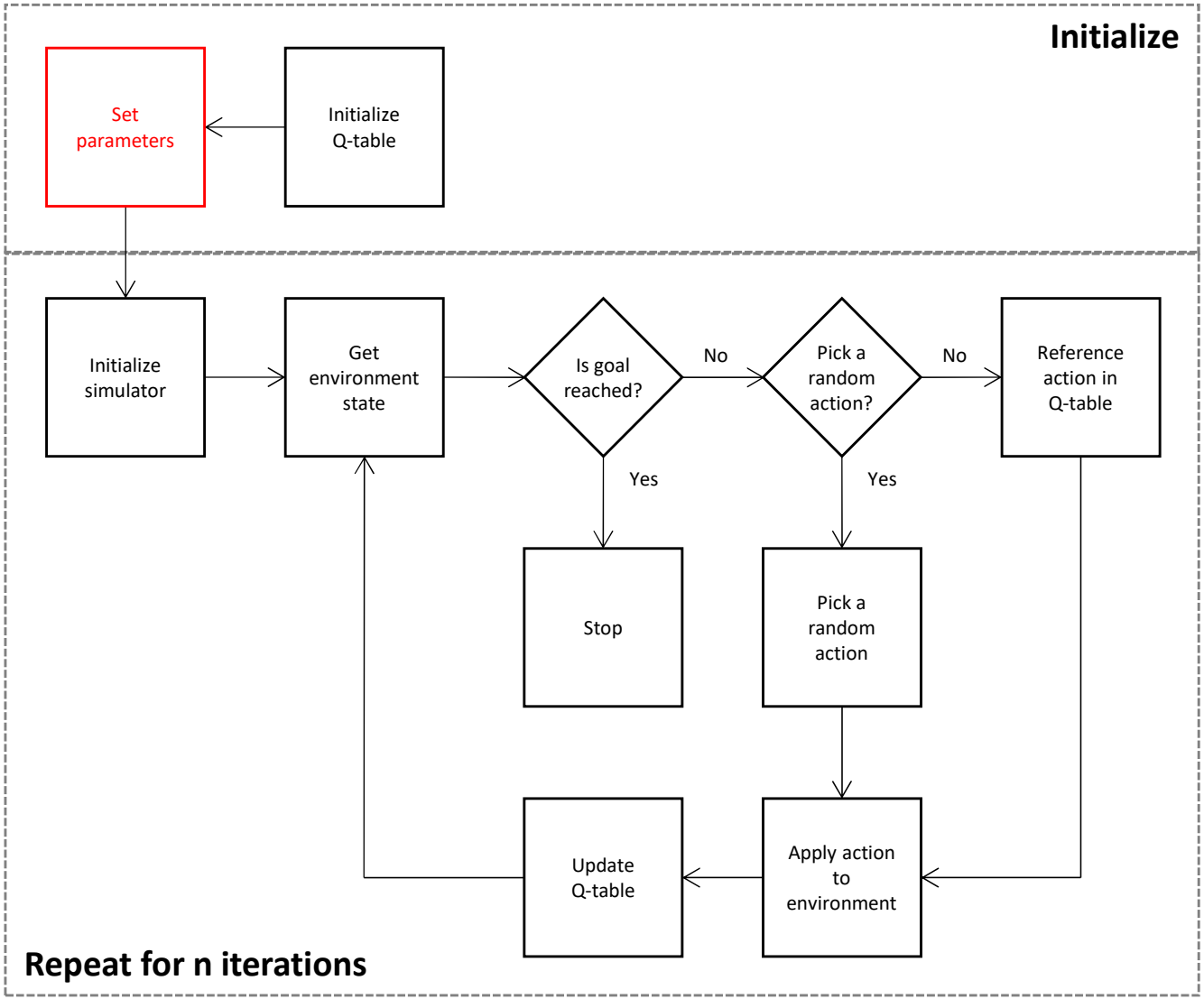


Initialize Q-table:
Set up and initialize (**all values set to 0**) a table where:

- rows represent **possible states**
- columns represent **actions**

Note that additional states can be added to the table when encountered.

Q-Learning Algorithm

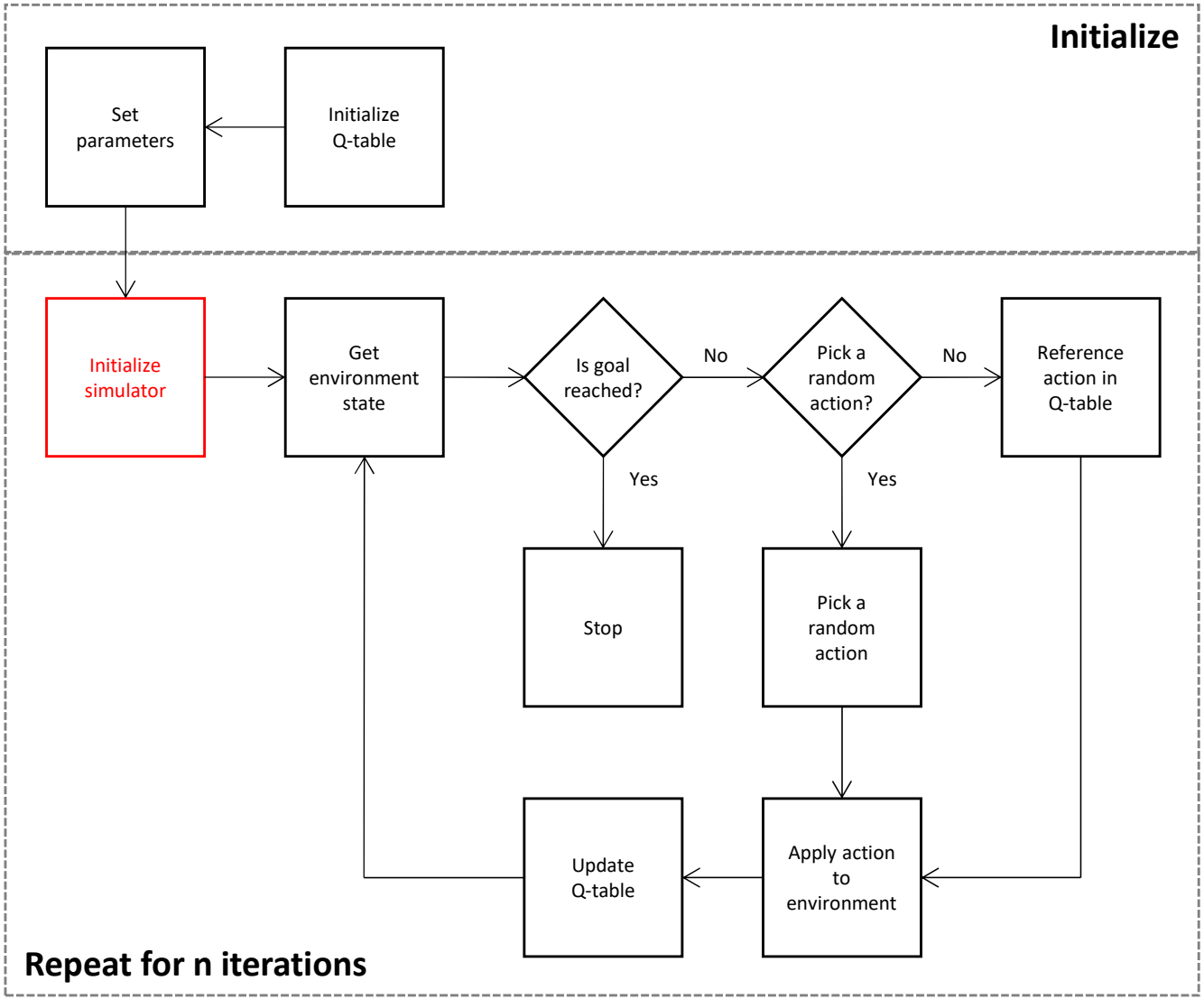


Set parameters:
Set and initialize **hyperparameters** for the Q-learning process.

Hyperparemeters include:

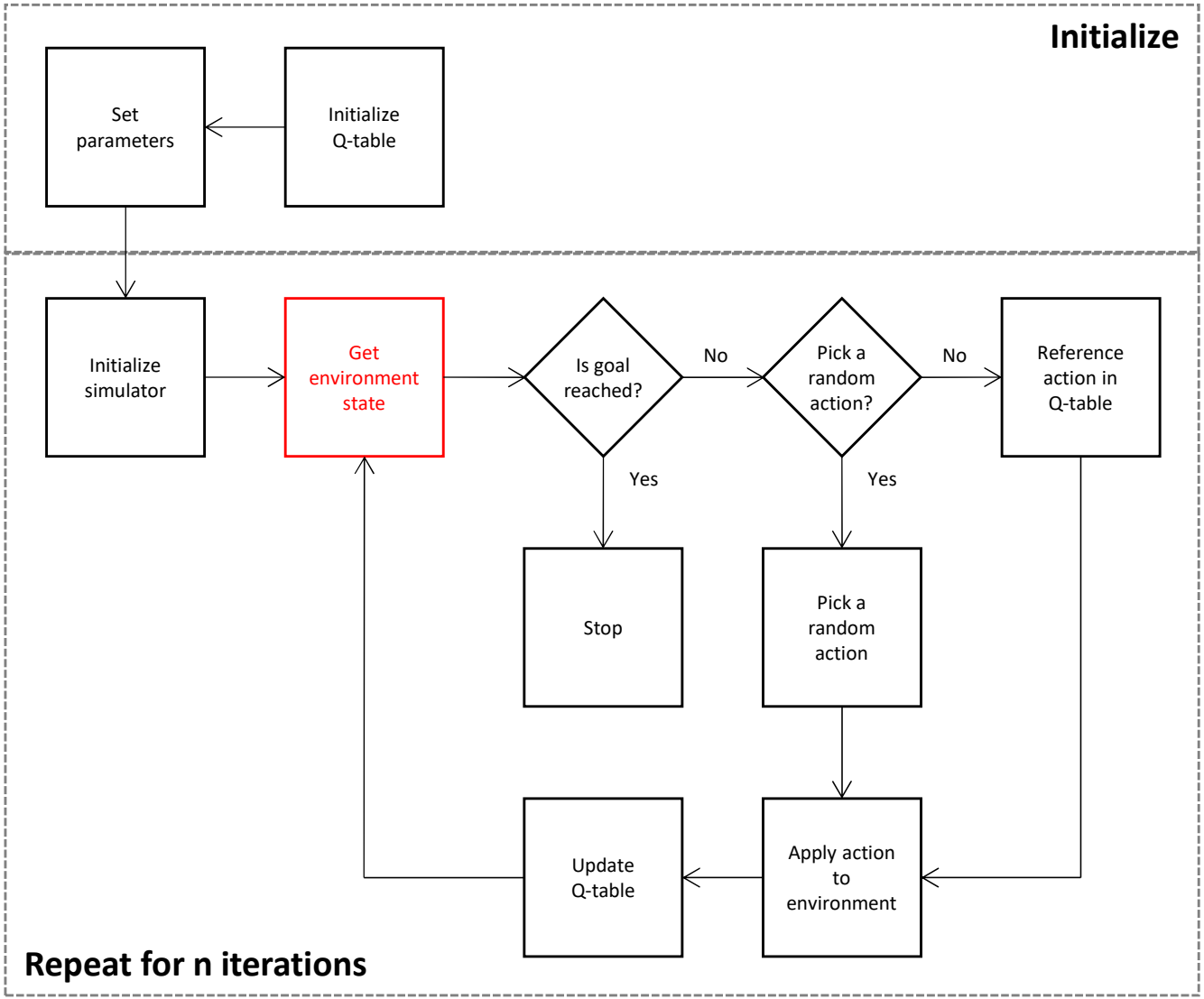
- **chance of choosing a random action**: a threshold for choosing a random action over an action from the Q-table
- **learning rate**: a parameter that describes how quickly the algorithm should learn from rewards in different states
  - high: faster learning with erratic Q-table changes
  - low: gradual learning with possibly more iterations
- **discount factor**: a parameter that describes how valuable are future rewards. It tells the algorithm whether it should seek "immediate gratification" (small) or "long-term reward" (large)

Q-Learning Algorithm



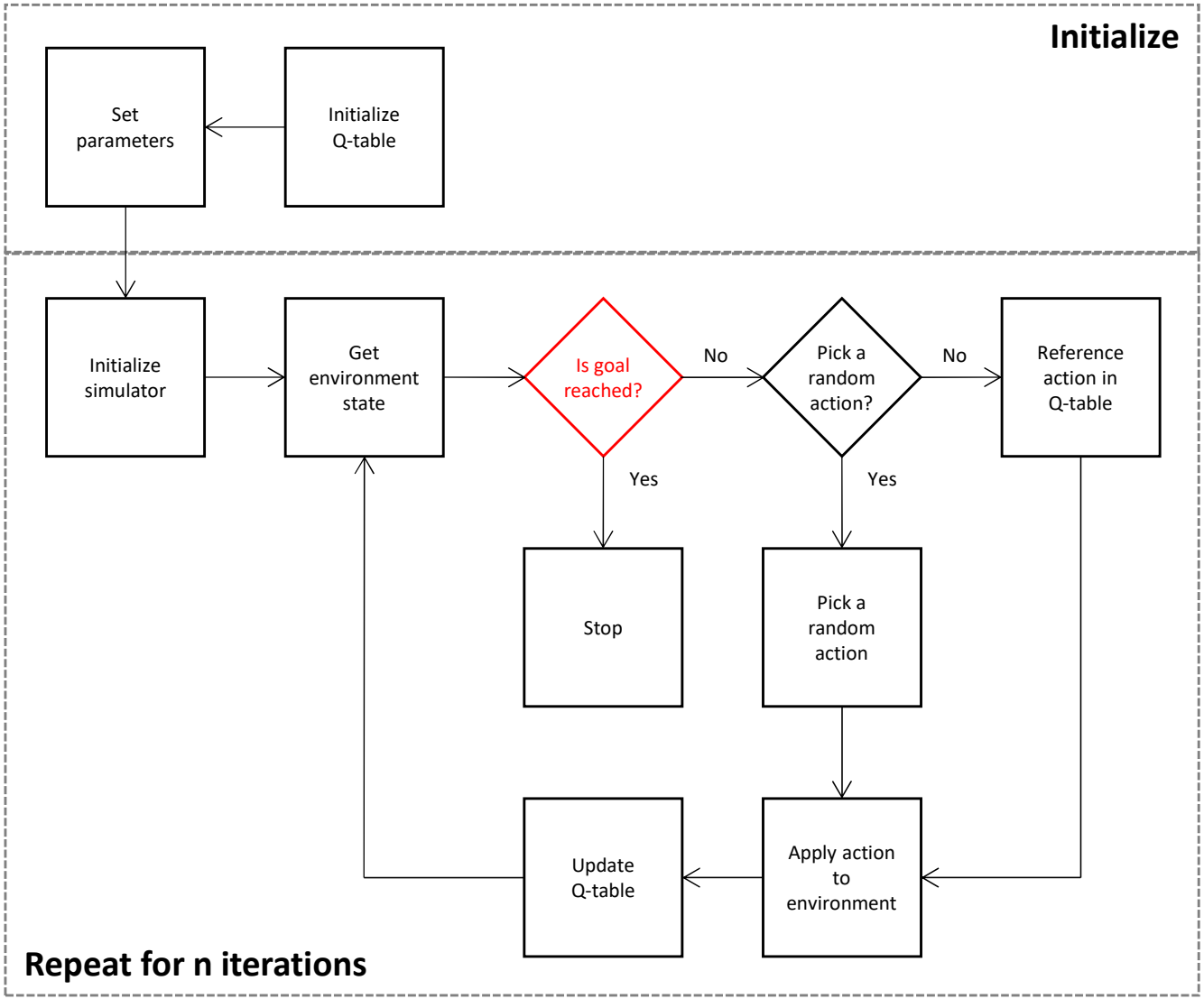
Initialize simulator:
Reset the simulated environment to its initial state and place the agent in a neutral state.

Q-Learning Algorithm



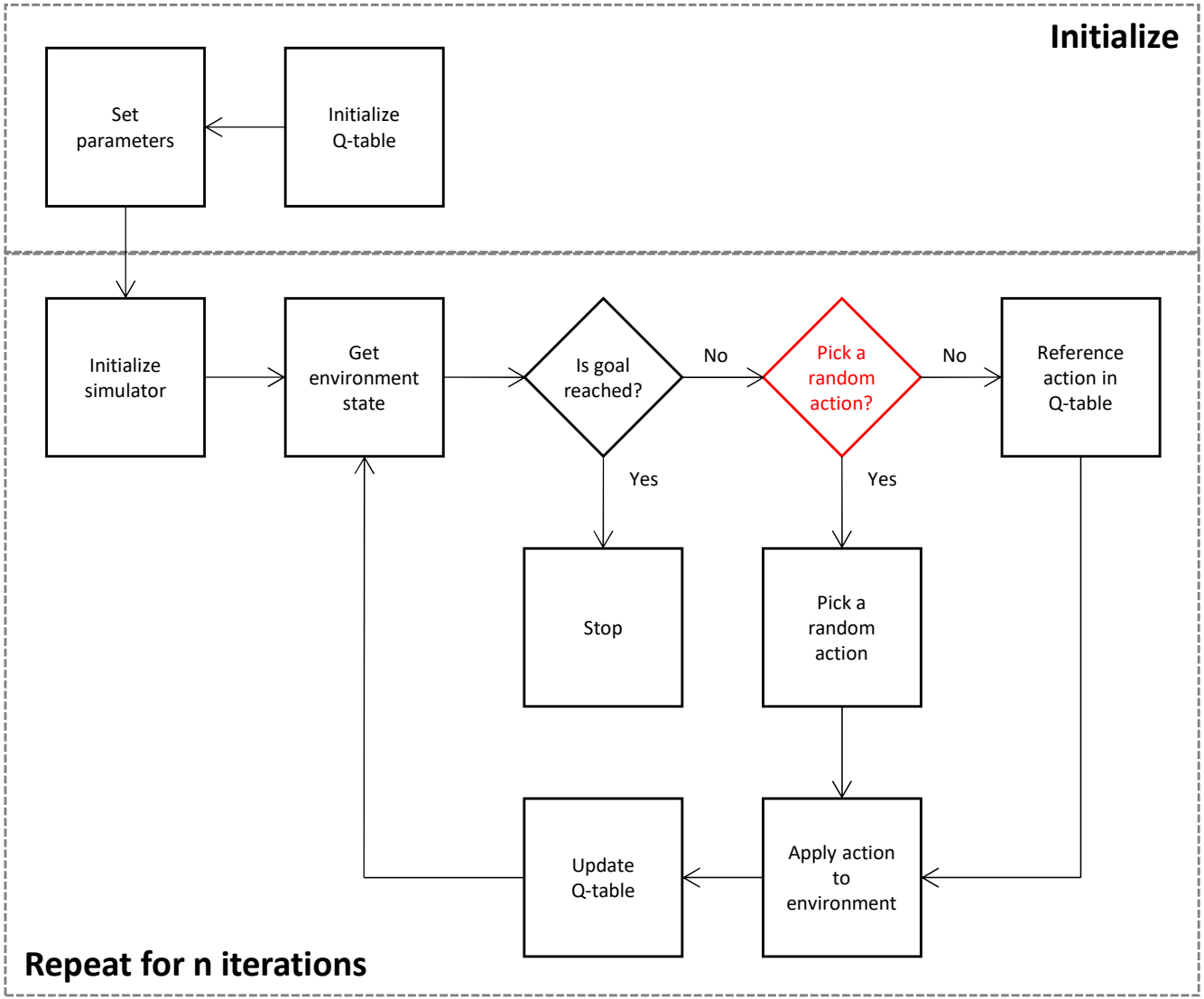
Get environment state:
Report the current state of the environment. Typically a vector of values representing all relevant variables.

Q-Learning Algorithm



Is goal reached?:
Verify if the goal of the simulation has been achieved. It could be decided with the agent arriving in expected final state or by some simulation parameter.

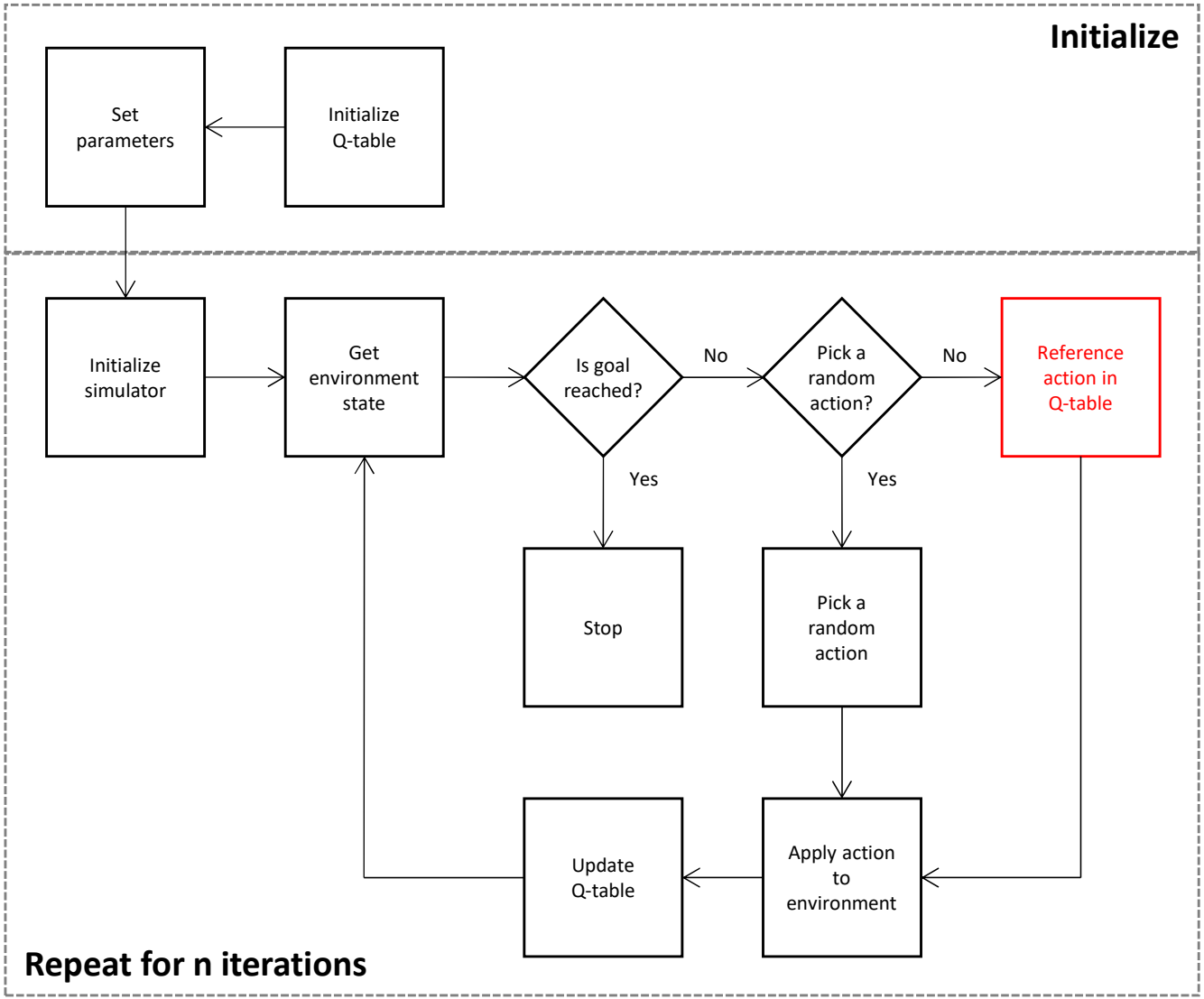
Q-Learning Algorithm



Pick a random action?:
Decide whether next action should be picked at random or not (it will be selected based on Q-table data then).

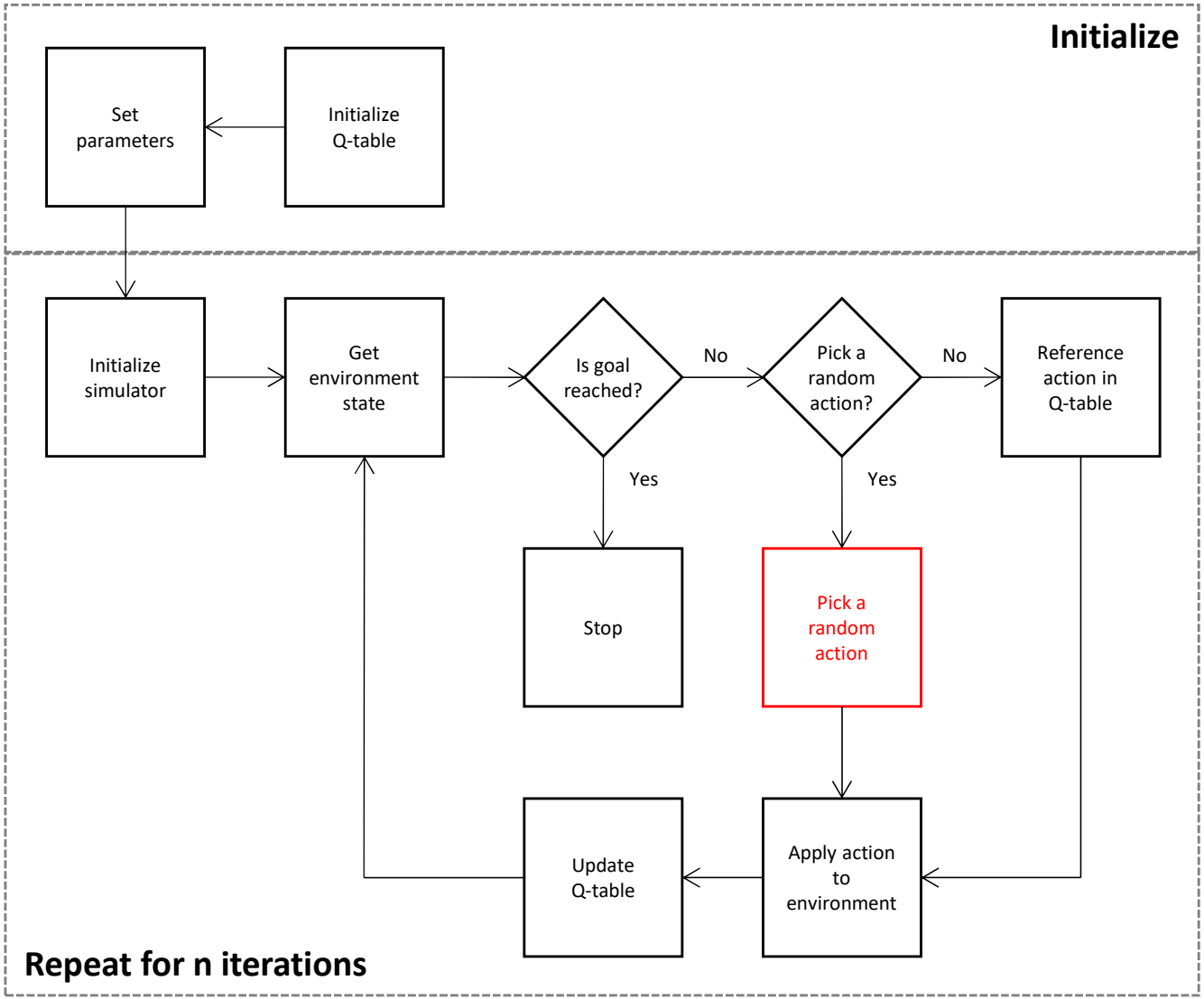
Use the **chance of choosing a random action hyperparameter** to decide.

Q-Learning Algorithm



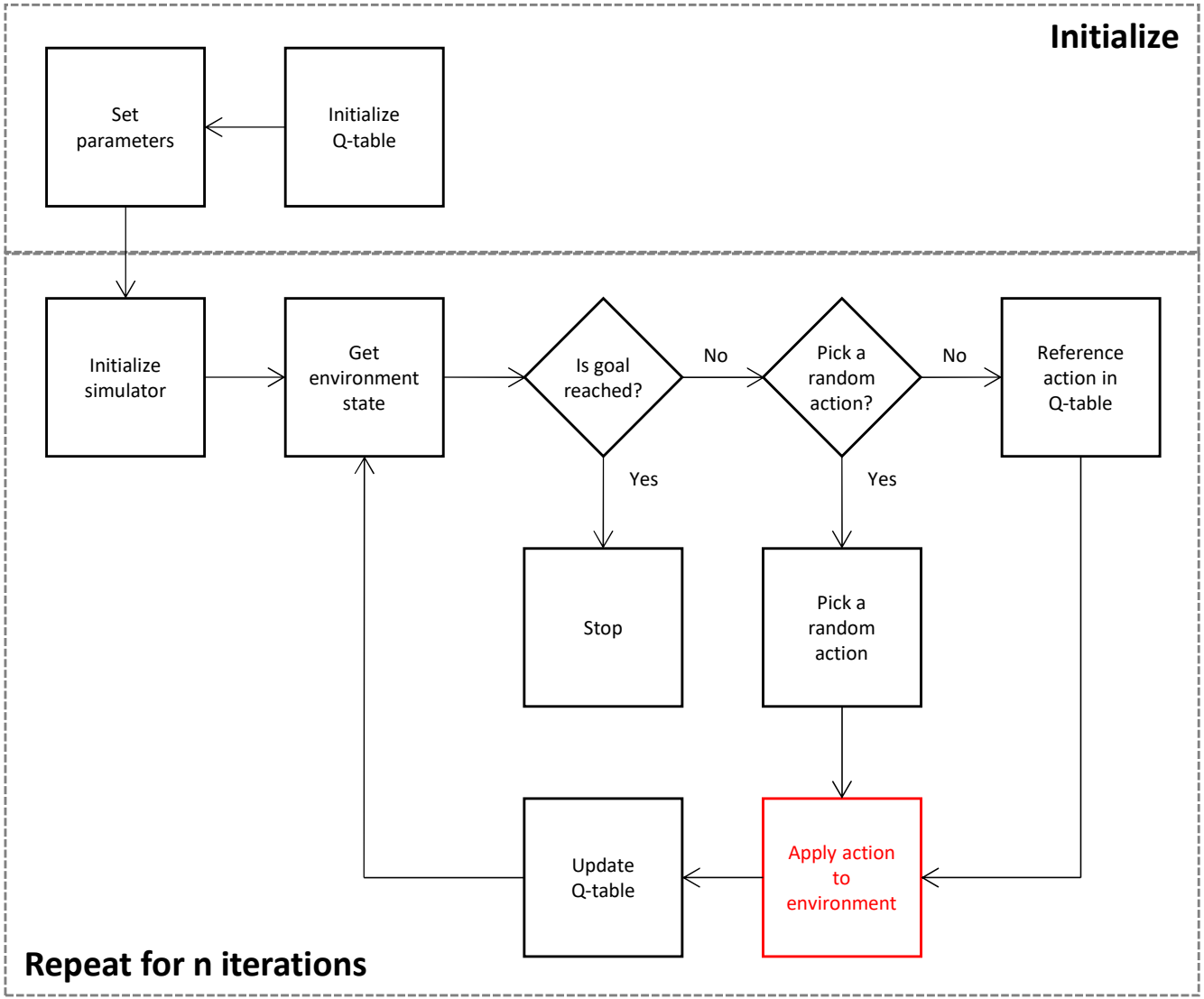
Reference action in Q-table:
Next action decision will be based on data from the Q-table given the current state of the environment.

Q-Learning Algorithm



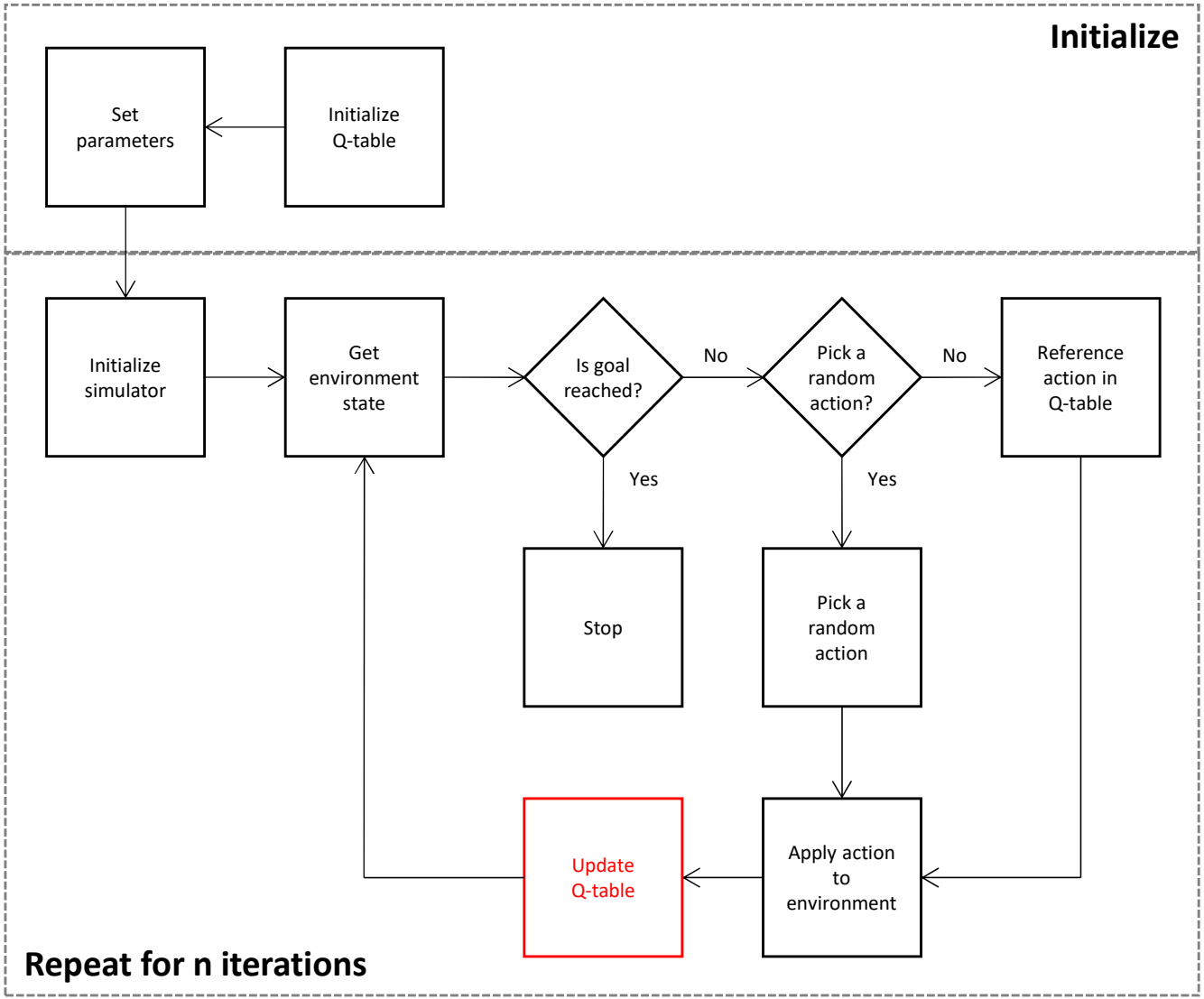
Pick a random action:
Pick any of the available actions at random. Helpful with exploration of the environment.

Q-Learning Algorithm



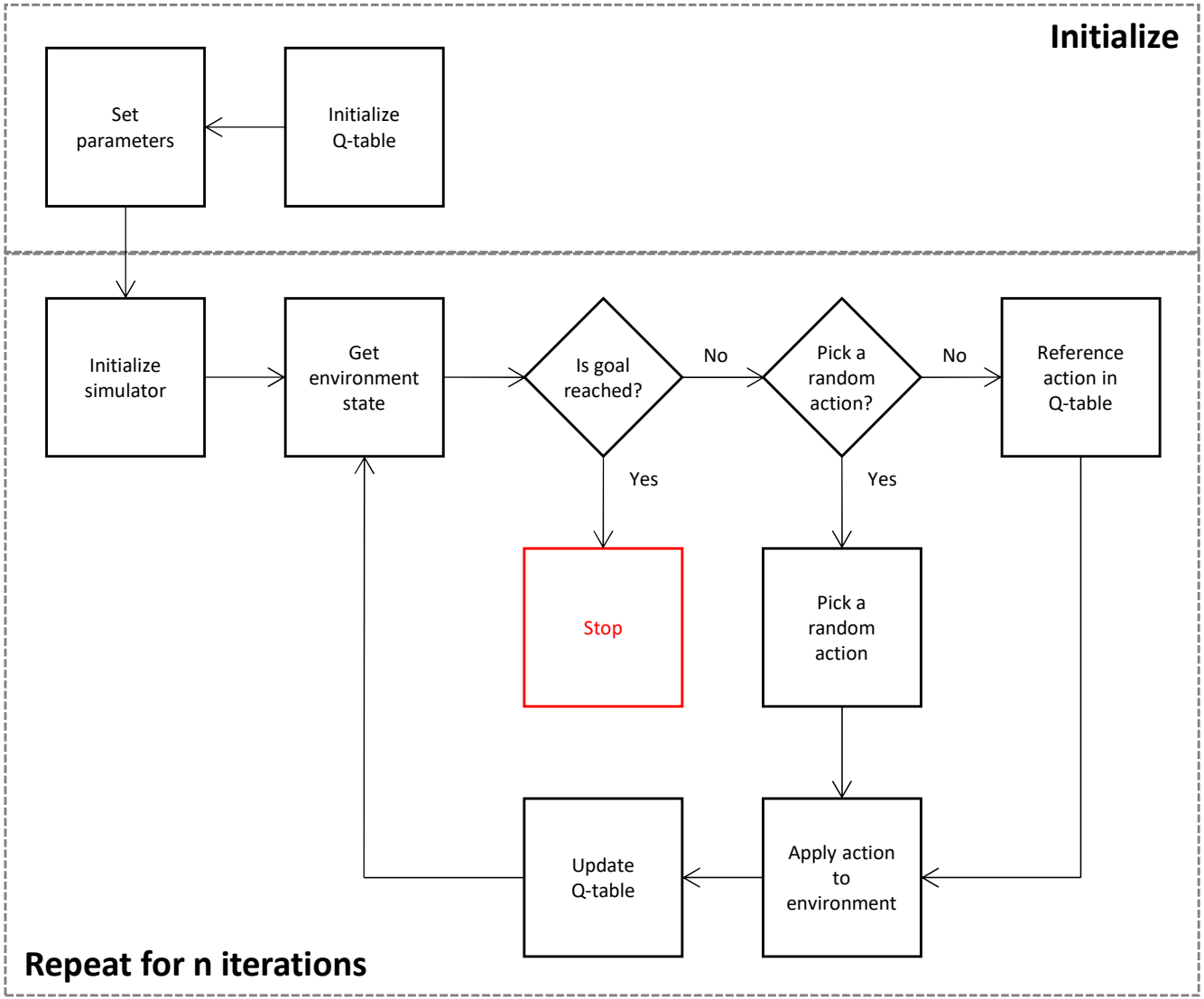
Apply action to environment:
Apply the action to the environment to change it. Each action will have its own reward.

Q-Learning Algorithm



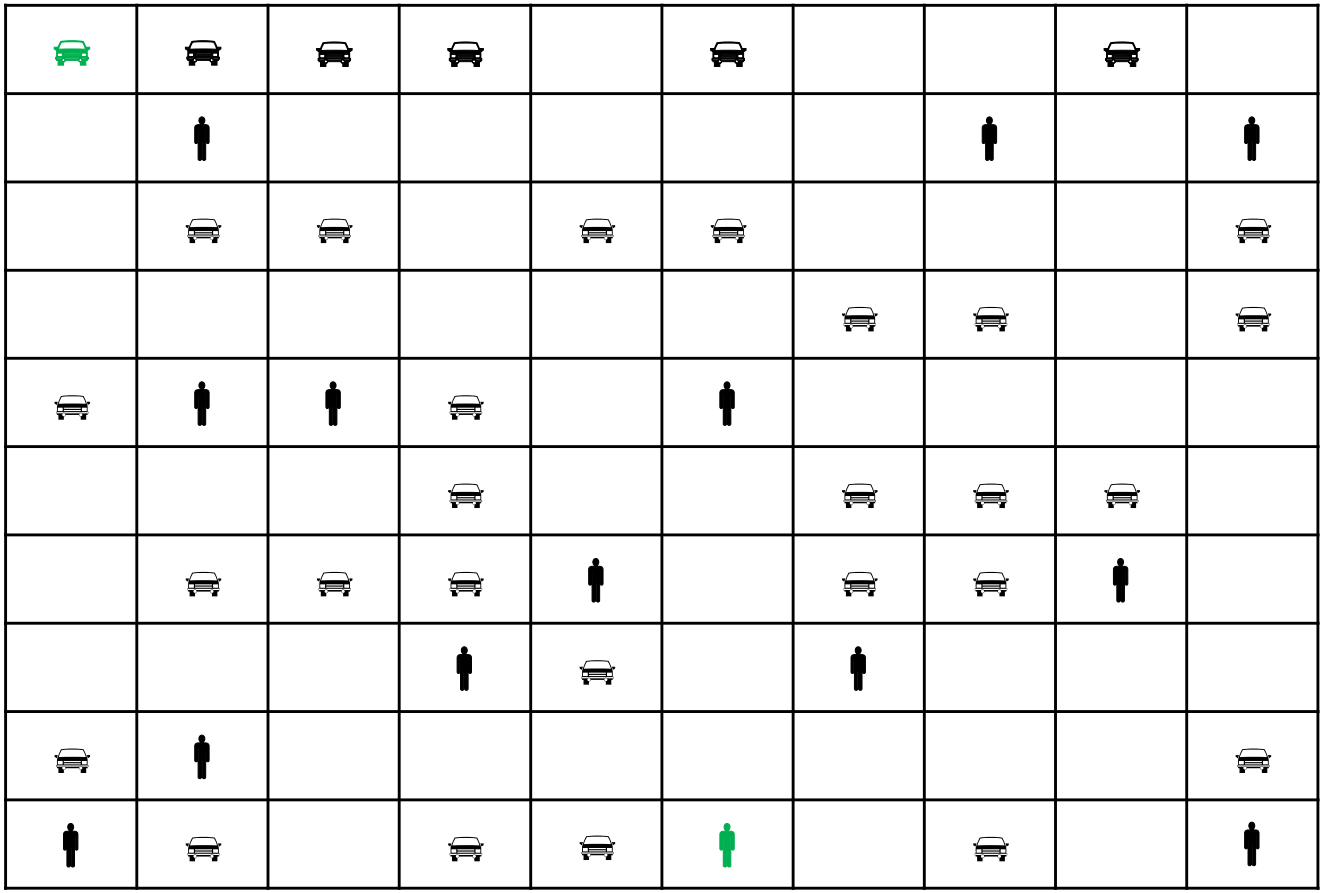
Update Q-table:
Update the Q-table given the reward resulting from recently applied action (feedback from the environment).

Q-Learning Algorithm



Stop:
Stop the learning process

Q-Learning Algorithm



Q-table		Actions			
		↑	↓	→	←
States	1	0	0	0	0
	2	0	0	0	0
	...	...	...	...	...
	n	0	0	0	0

Rewards:

Move into car: -100

Move into pedestrian: -1000

Move into empty space: 100

Move into goal: 500

Action:

Reward:

Q-table value:

Learning rate

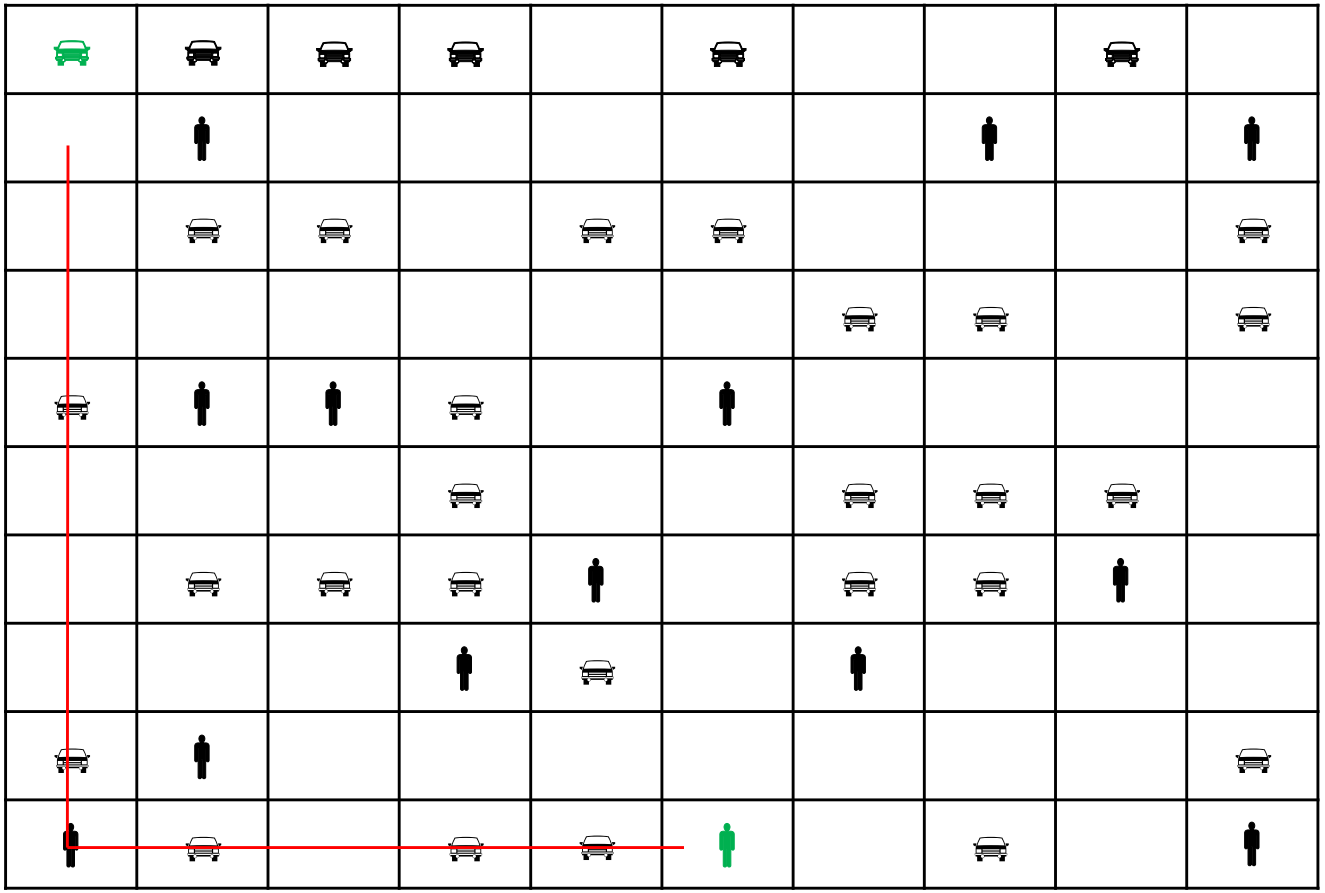
Discount

$$Q(\text{state}, \text{action}) = (1 - \text{alpha}) * Q(\text{state}, \text{action}) + \text{alpha} * (\text{reward} + \text{gamma} * Q(\text{next state}, \text{all actions}))$$

Current value

Maximum value of all actions on next state

Q-Learning Algorithm



Q-table		Actions			
		↑	↓	→	←
States	1	0	0	0	0
	2	0	0	0	0
	...	...	...	...	...
	n	0	0	0	0

Rewards:

Move into car: -100

Move into pedestrian: -1000

Move into empty space: 100

Move into goal: 500

Action: **Reward:**

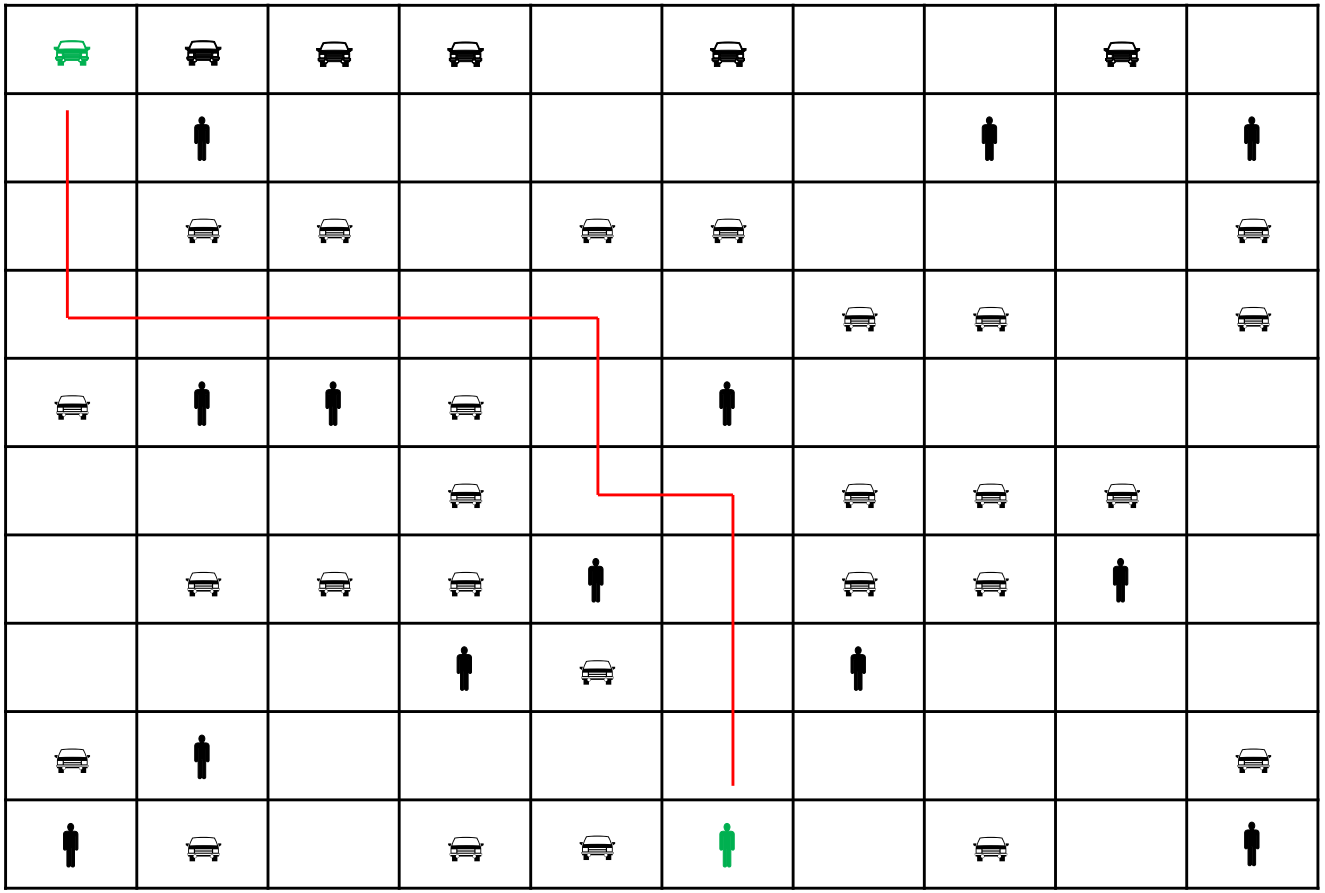
Q-table value:

Learning rate — Discount

$$Q(\text{state}, \text{action}) = (1 - \text{alpha}) * Q(\text{state}, \text{action}) + \text{alpha} * (\text{reward} + \text{gamma} * Q(\text{next state}, \text{all actions}))$$

Current value — Maximum value of all actions on next state

Q-Learning Algorithm



Q-table		Actions			
		↑	↓	→	←
States	1	0	0	0	0
	2	0	0	0	0
	...	...	...	...	...
	n	0	0	0	0

Rewards:

Move into car: -100

Move into pedestrian: -1000

Move into empty space: 100

Move into goal: 500

Action:

Reward:

Q-table value:

Learning rate

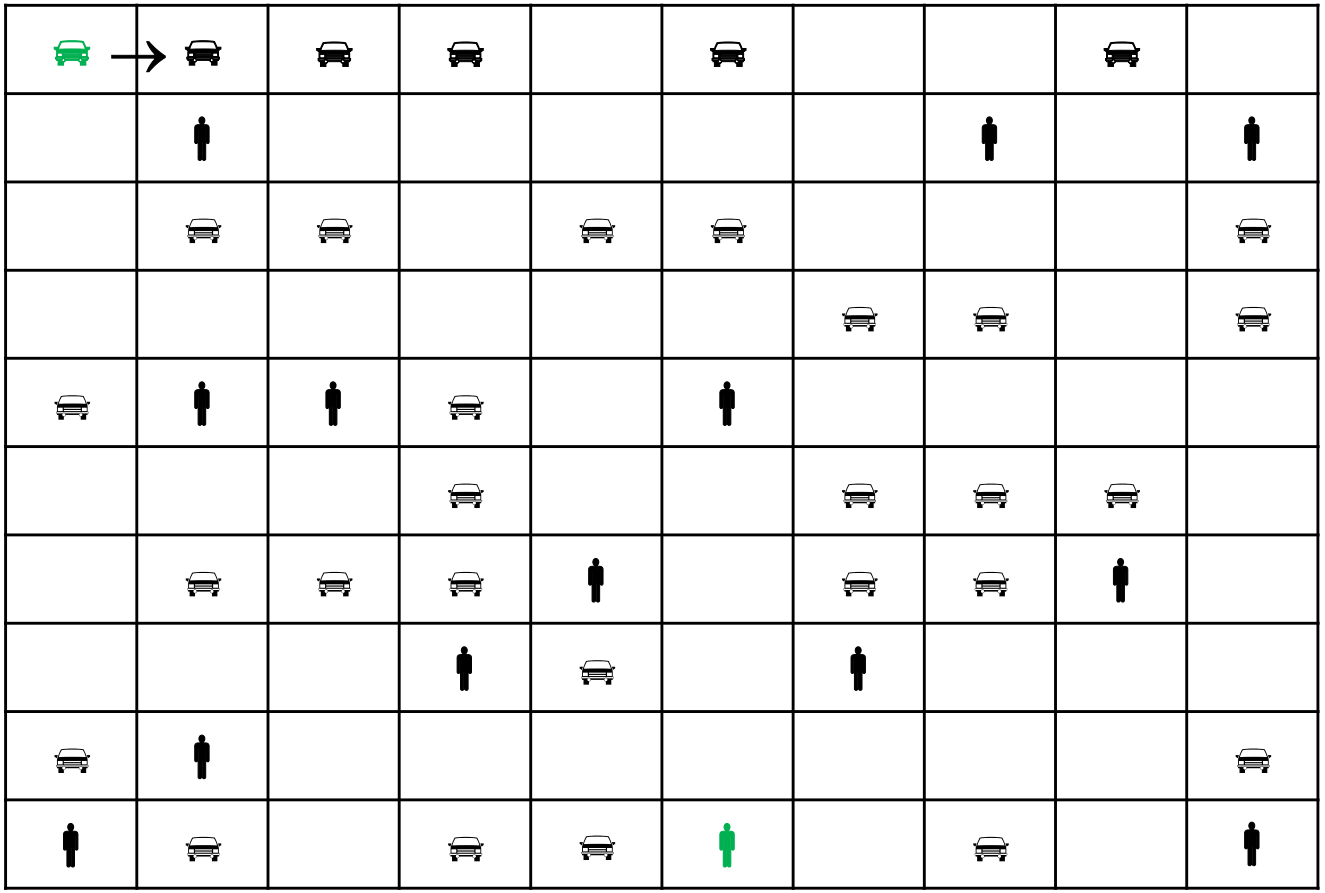
Discount

$$Q(\text{state}, \text{action}) = (1 - \text{alpha}) * Q(\text{state}, \text{action}) + \text{alpha} * (\text{reward} + \text{gamma} * Q(\text{next state}, \text{all actions}))$$

Current value

Maximum value of all actions on next state

Q-Learning Algorithm



Q-table		Actions			
		↑	↓	→	←
States	1	0	0	0	0
	2	0	0	0	0
	...	...	...	...	...
	n	0	0	0	0

Rewards:
Move into car: -100
Move into pedestrian: -1000
Move into empty space: 100
Move into goal: 500

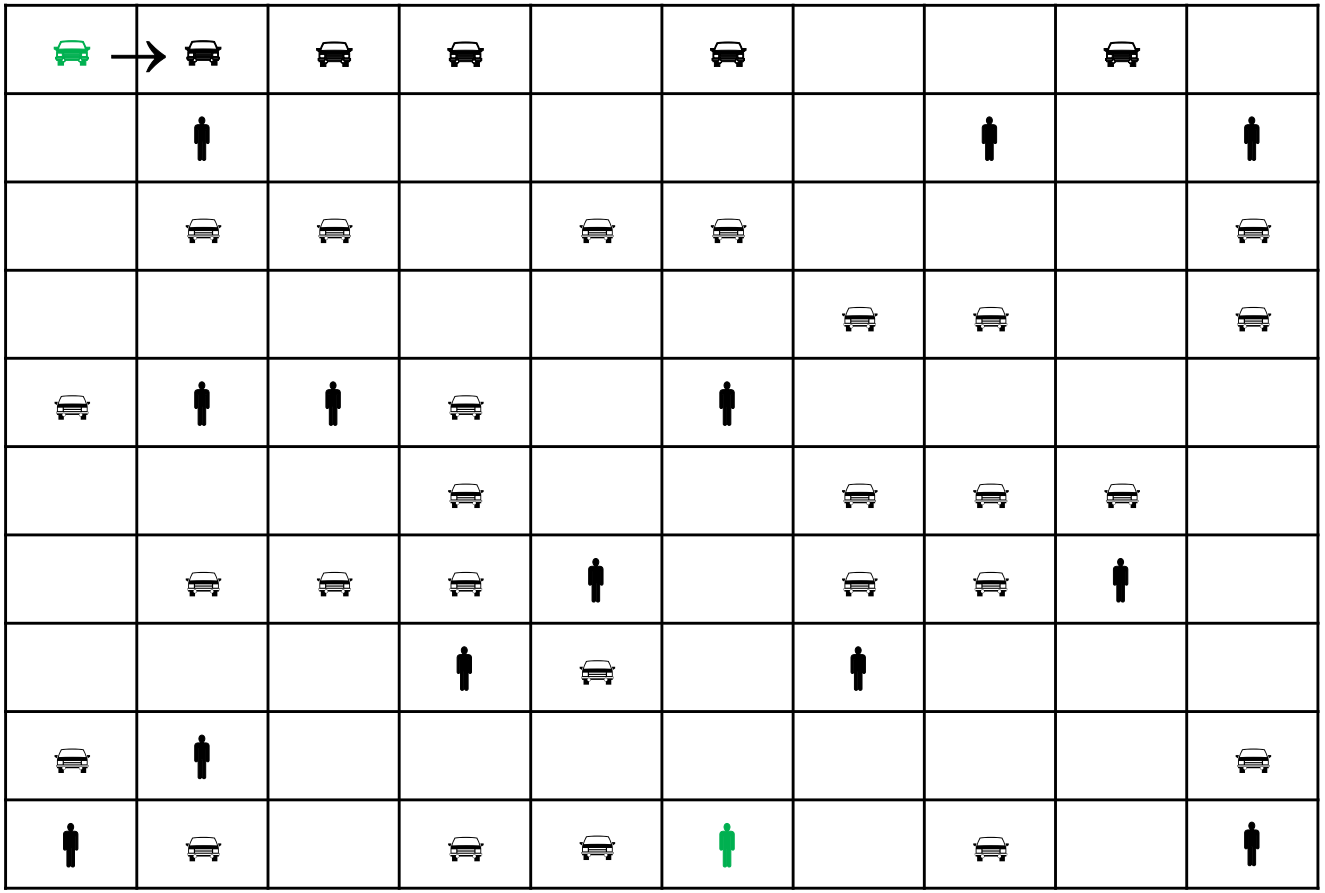
Action: →

Reward: -100

Q-table value:

$$Q(1, \text{east}) = (1 - 0.1) * 0 + 0.1 * (-100 + 0.6 * \max \text{ of } Q(2, \text{all actions}))$$

Q-Learning Algorithm



Q-table		Actions			
		↑	↓	→	←
States	1	0	0	-10	0
	2	0	0	0	0
	...	...	...	...	...
	n	0	0	0	0

Rewards:
Move into car: -100
Move into pedestrian: -1000
Move into empty space: 100
Move into goal: 500

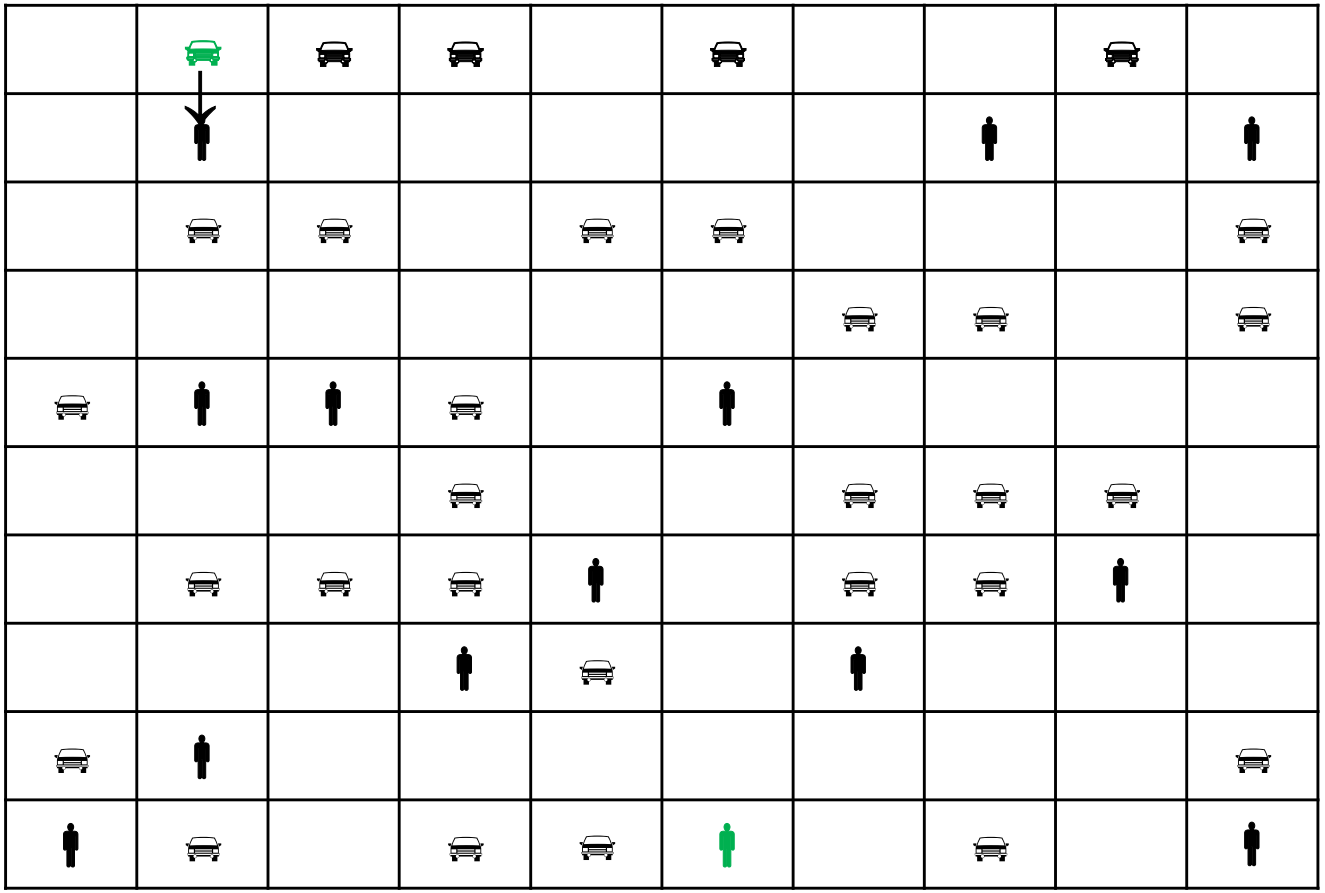
Action: →

Reward: -100

Q-table value:

$$Q(1, \text{east}) = (1 - 0.1) * 0 + 0.1 * (-100 + 0.6 * 0) = -10$$

Q-Learning Algorithm



Q-table		Actions			
		↑	↓	→	←
States	1	0	0	-10	0
	2	0	0	0	0
	...	...	...	...	...
	n	0	0	0	0

Rewards:
Move into car: -100
Move into pedestrian: -1000
Move into empty space: 100
Move into goal: 500

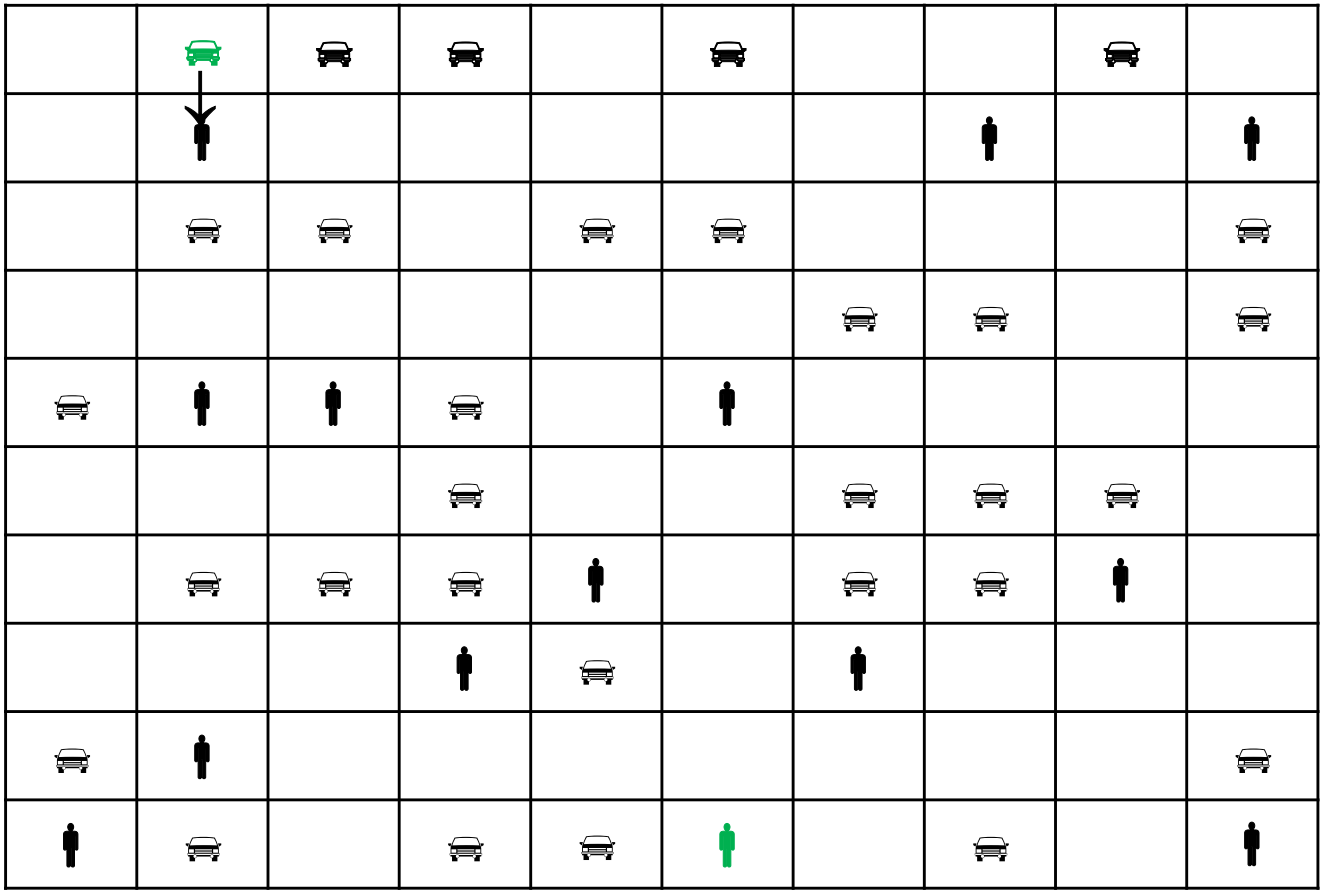
Action: →

Reward: -1000

Q-table value:

$$Q(2, \text{south}) = (1 - 0.1) * 0 + 0.1 * (-1000 + 0.6 * \max \text{ of } Q(3, \text{all actions}))$$

Q-Learning Algorithm



Q-table		Actions			
		↑	↓	→	←
States	1	0	0	-10	0
	2	0	-100	0	0
	...	...	...	...	...
	n	0	0	0	0

Rewards:
Move into car: -100
Move into pedestrian: -1000
Move into empty space: 100
Move into goal: 500

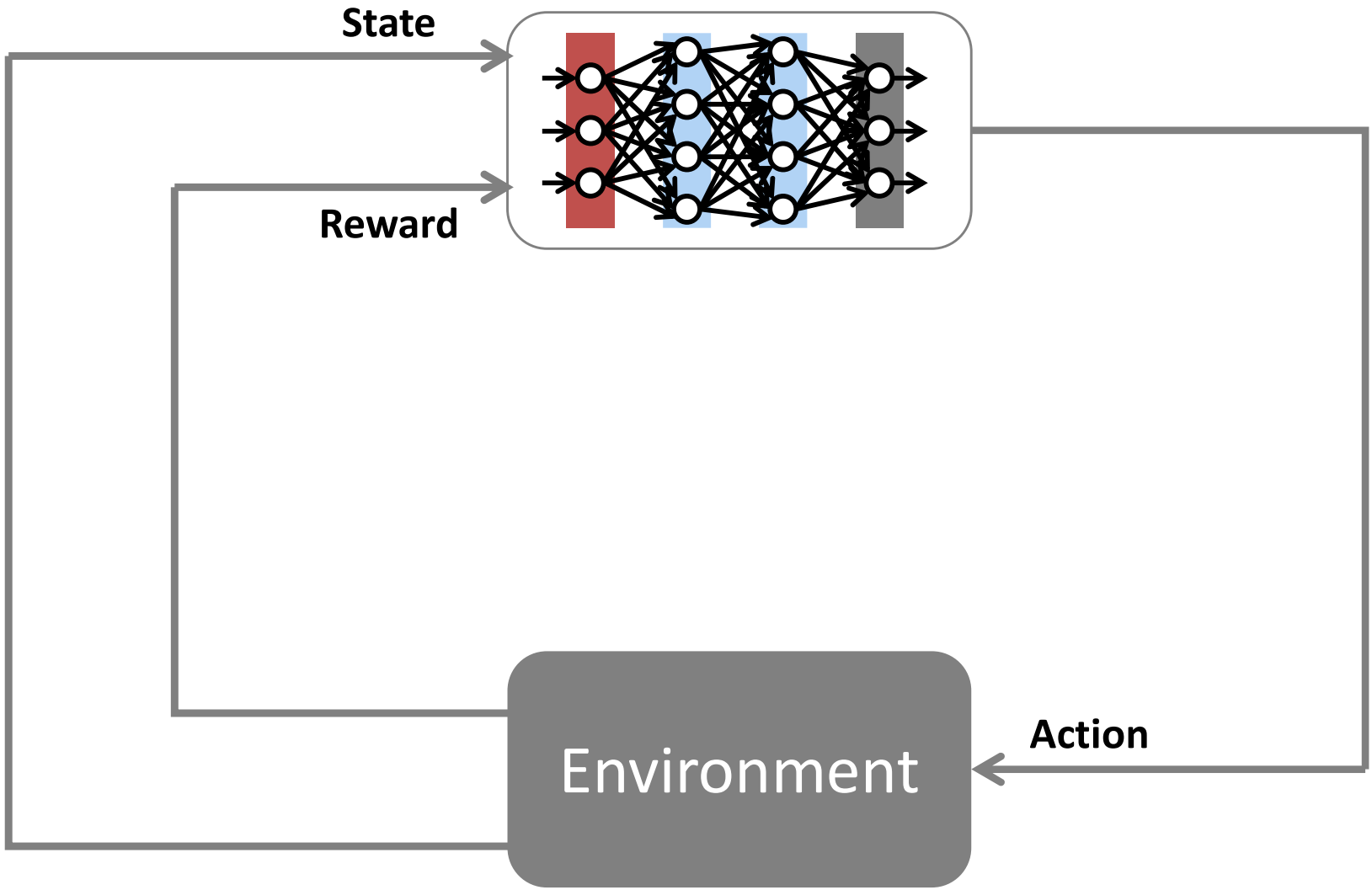
Action: →

Reward:  -1000

Q-table value:

$$Q(2, \text{south}) = (1 - 0.1) * 0 + 0.1 * (-1000 + 0.6 * 0) = -100$$

Deep Reinforcement Learning



RL: Agents and Environments