CS 581

Advanced Artificial Intelligence

April 15, 2024

Announcements / Reminders

- Please follow the Week 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

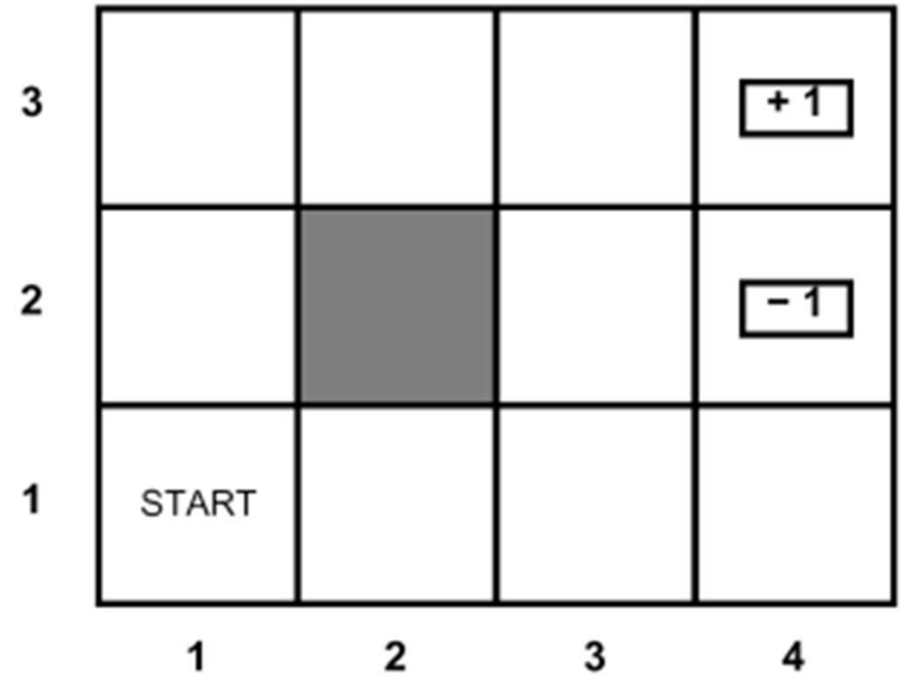
- **Reinforcement Learning: Introduction**
- **Q-Learning**

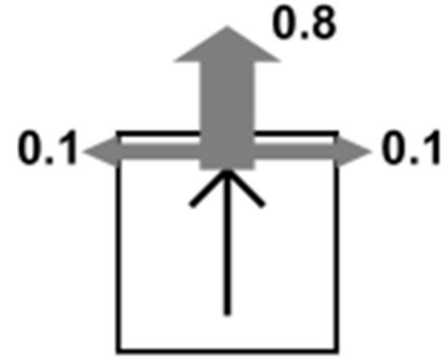
Refresher

- **MDPs**
  - **Value function** V**, action-value function** Q**, policy** π
  - **Bellman equations**
  - **Value iteration**
  - **Policy iteration**
- **Multi-armed bandits**
  - **Exploration vs exploitation trade-off**
  - ϵ**-greedy approach**

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., $P(s' \mid s,a)$
    - 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



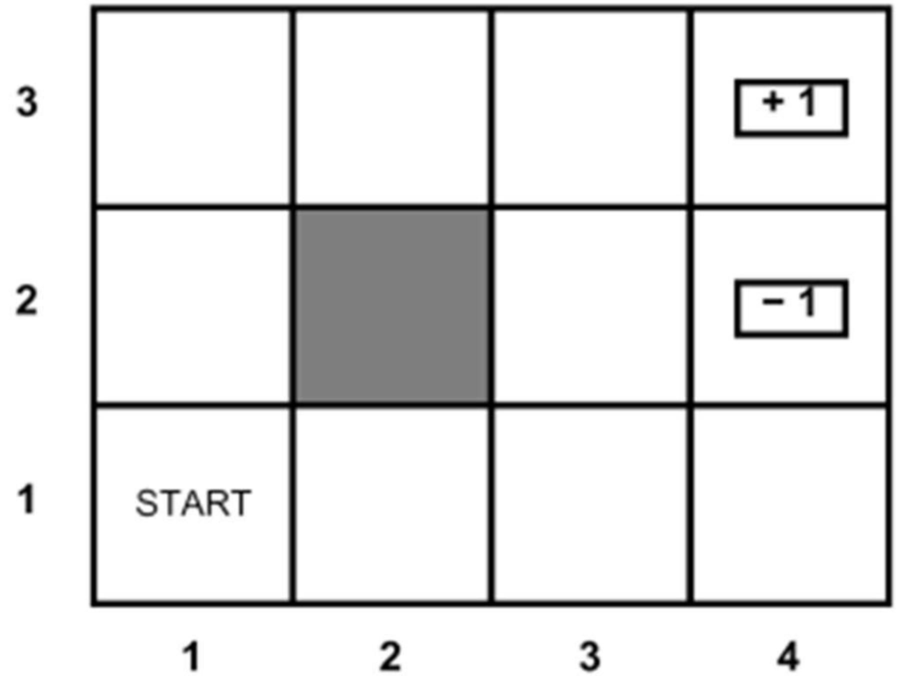


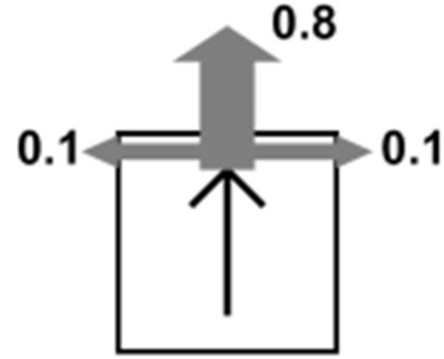
Markov Decision Process

- 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., $P(s' \mid s,a)$
    - Also called the model

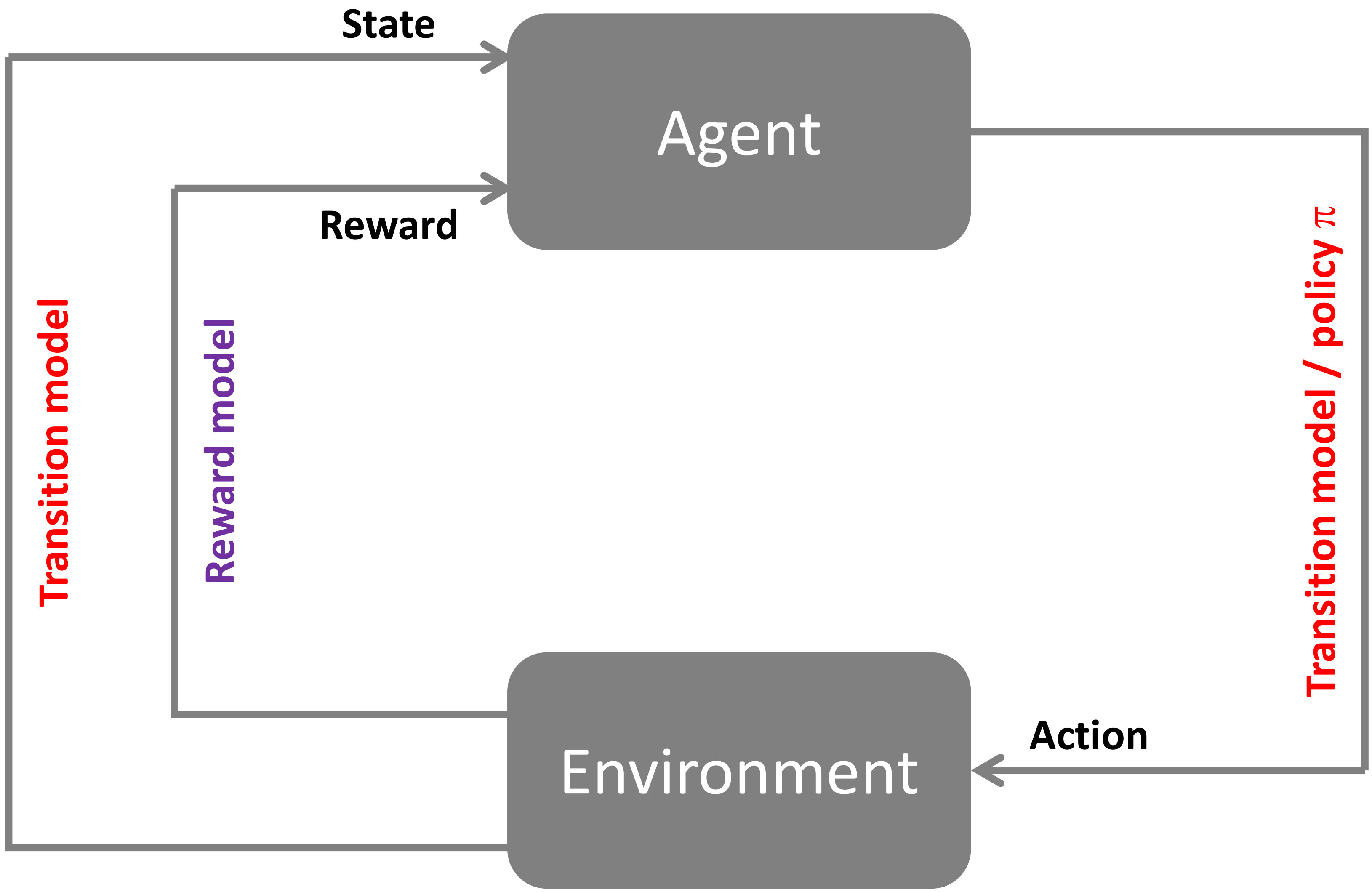
Model

  - A reward function $R(s, a, s')$
    - Sometimes just $R(s)$ or $R(s')$
  - A start state (or distribution)
  - Maybe a terminal state





Markov Decision Process



Solving MDPs

- **Offline algorithms:**
  - **Value iteration**
  - **Policy iteration**
  - **Linear programming**

- **Online algorithms:**
  - **Approximation algorithms such as Monte Carlo planning**

Bellman Equations

Bellman Equation

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

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

Bellman Equation

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

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

Utility / value of current state s

Bellman Equation

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

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

Expected "long-term" utility / value after applying ONE specific action a **[<u>Need Environment Model!</u>]**

Bellman Equation

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

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

Probability of transitioning FROM current state s TO future state s' after applying action a [<u>Need Environment Model!</u>]

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:

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

Reward after transitioning FROM current state s TO future state s' after applying action a [Need Environment Model!]

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:

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

CURRENT / SINGLE transition reward

[Need Environment Model!]

Bellman Equation

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

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

Discount factor

Bellman Equation

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

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

Future state s' utility

[can be a rough estimate at the beginning]

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:

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

Discounted future state s' utility
[can be a rough estimate at the beginning]

Bellman Equation

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

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

<u>BEST</u> Expected "long-term" utility / value after applying ONE specific <u>BEST</u> action a **[<u>Need Environment Model!</u>]**

Bellman Equation

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

Go through all possible actions a

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

<u>BEST</u> Expected "long-term" utility / value after applying ONE specific <u>BEST</u> action a [<u>Need Environment Model!</u>]

Bellman Equation

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

Go through all possible future states s'

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

Expected "long-term" utility / value after applying ONE specific action a [<u>Need Environment Model!</u>]

Expected Utility Given Policy π

The utility of a state is the <u>expected</u> reward for the next transition plus the <u>discounted</u> utility of the next state, assuming that the agent uses policy π:

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

Bellman Optimality

The utility of a state is the <u>expected</u> reward for the next transition plus the <u>discounted</u> utility of the next state, assuming that the agent uses policy π:

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

Bellman Update

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

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

Bellman Equation: Example

	1	2	3	4
3	0.812	0.868	0.918	$\boxed{+1}$
2	0.762		0.660	$\boxed{-1}$
1	0.705	0.655	0.611	0.388

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

$$
\begin{aligned}
U(1,1) = -0.04 + \gamma \max[\ & 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), && (Up)\\
& 0.9U(1,1) + 0.1U(1,2), && (Left)\\
& 0.9U(1,1) + 0.1U(2,1), && (Down)\\
& 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)\ ]. && (Right)
\end{aligned}
$$

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

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

N states: N Bellman Equations to iteratively "solve"

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

Value Iteration Algorithm

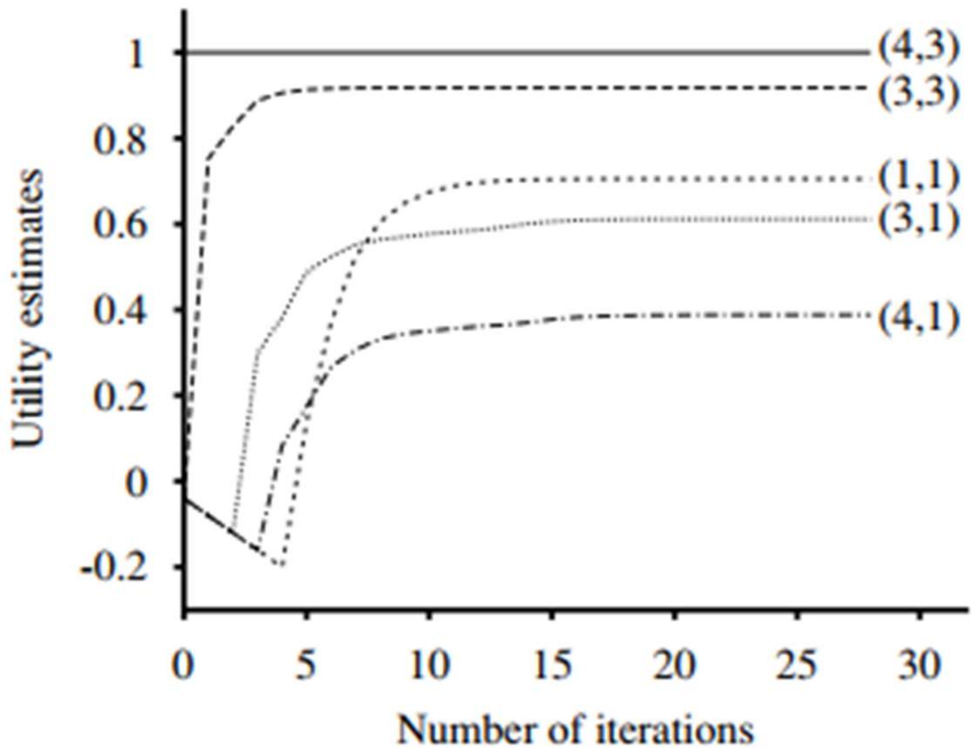
function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
rewards $R(s)$, discount γ
ϵ, the maximum error allowed in the utility of any state
local variables: U, U', vectors of utilities for states in S, initially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

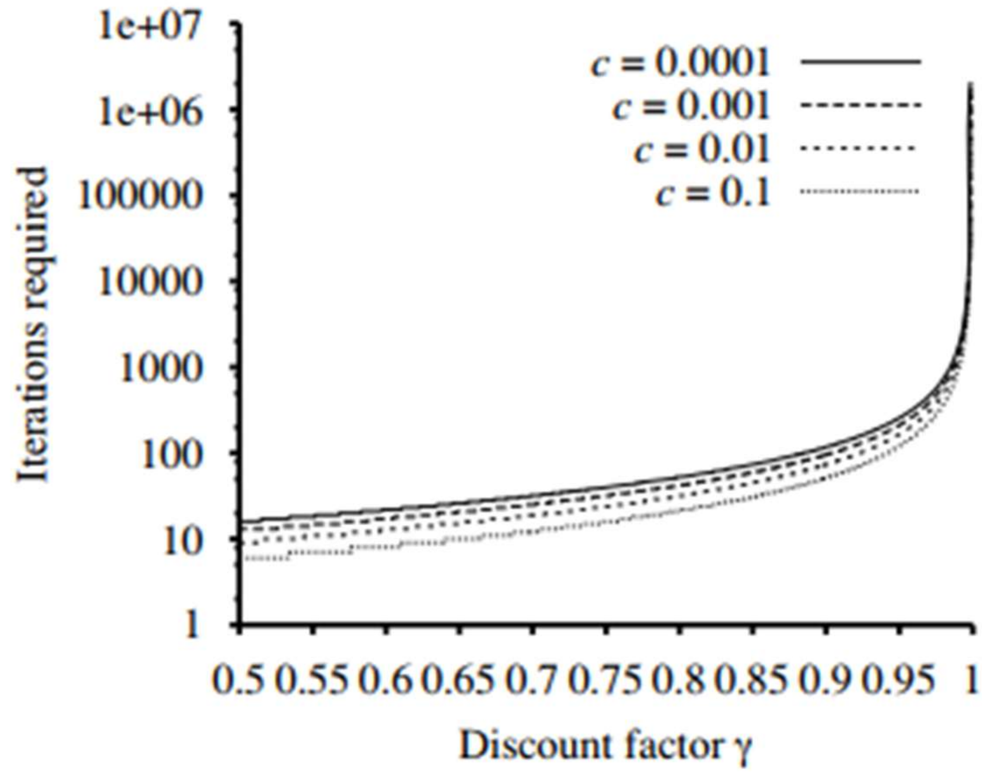
repeat

Repeat infinite times: guaranteed convergence

$U \leftarrow U'; \delta \leftarrow 0$
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

Value Iteration: Convergence





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

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

Bellman Update

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

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

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

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
    Policy evaluation
    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

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
Policy improvement
until $unchanged?$
return π

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

Recalculate policy (find new MEU policy) for all s

Bellman Equation

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

Go through all possible future states s'

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

Expected "long-term" utility / value after applying ONE specific action a [<u>Need Environment Model!</u>]

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**
Better action found
$\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
$unchanged? \leftarrow$ false
until $unchanged?$
return π

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**
        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$ Update policy for state s
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

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**
        $\pi[s] \leftarrow \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a)\ U[s']$
        $unchanged? \leftarrow$ false   Policy updated -> remember!
  until $unchanged?$
  return π

Reinforcement Learning

Main Machine Learning Categories

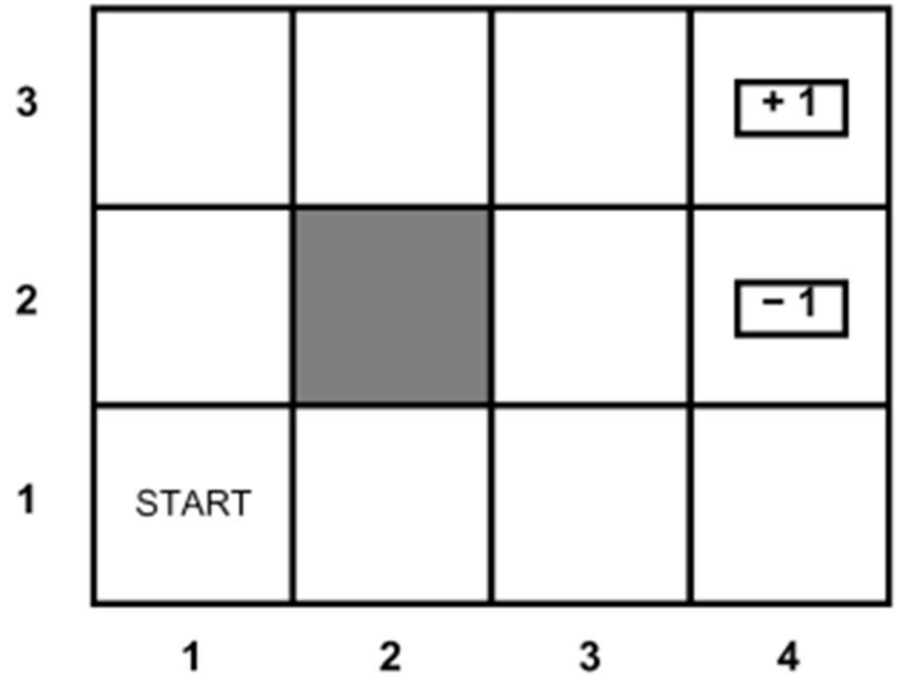
Supervised learning	Unsupervised learning	Reinforcement learning
Supervised learning is one of the most common techniques in machine learning. It is based on **known relationship(s) and patterns within data** (for example: relationship between inputs and outputs). Frequently used types: **regression**, 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.

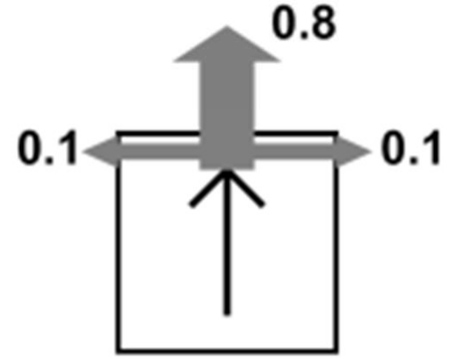
Markov Decision Process

- 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., $P(s' \mid s,a)$
    - Also called the model

Model

  - A reward function $R(s, a, s')$
    - Sometimes just $R(s)$ or $R(s')$
  - A start state (or distribution)
  - Maybe a terminal state



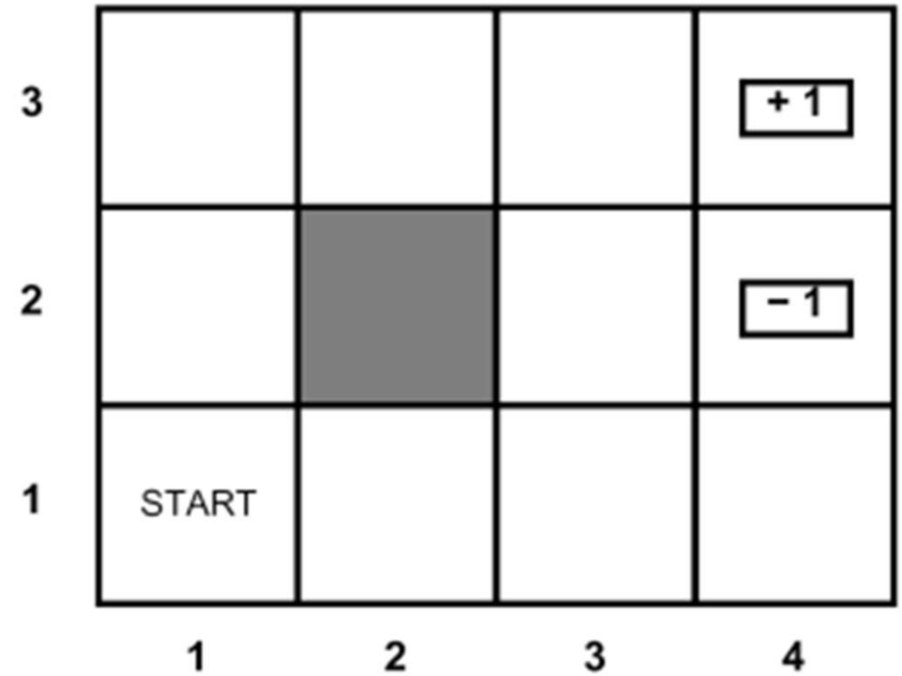


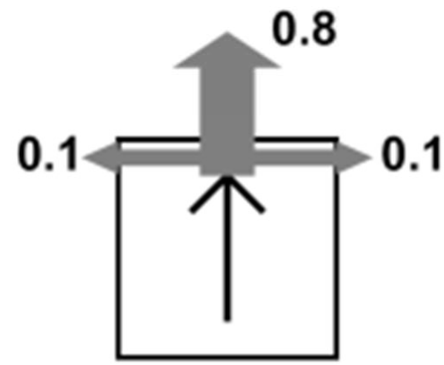
Reinforcement Learning

- An MDP is defined by:
  - A set of states s ∈ S
  - A set of actions a ∈ 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

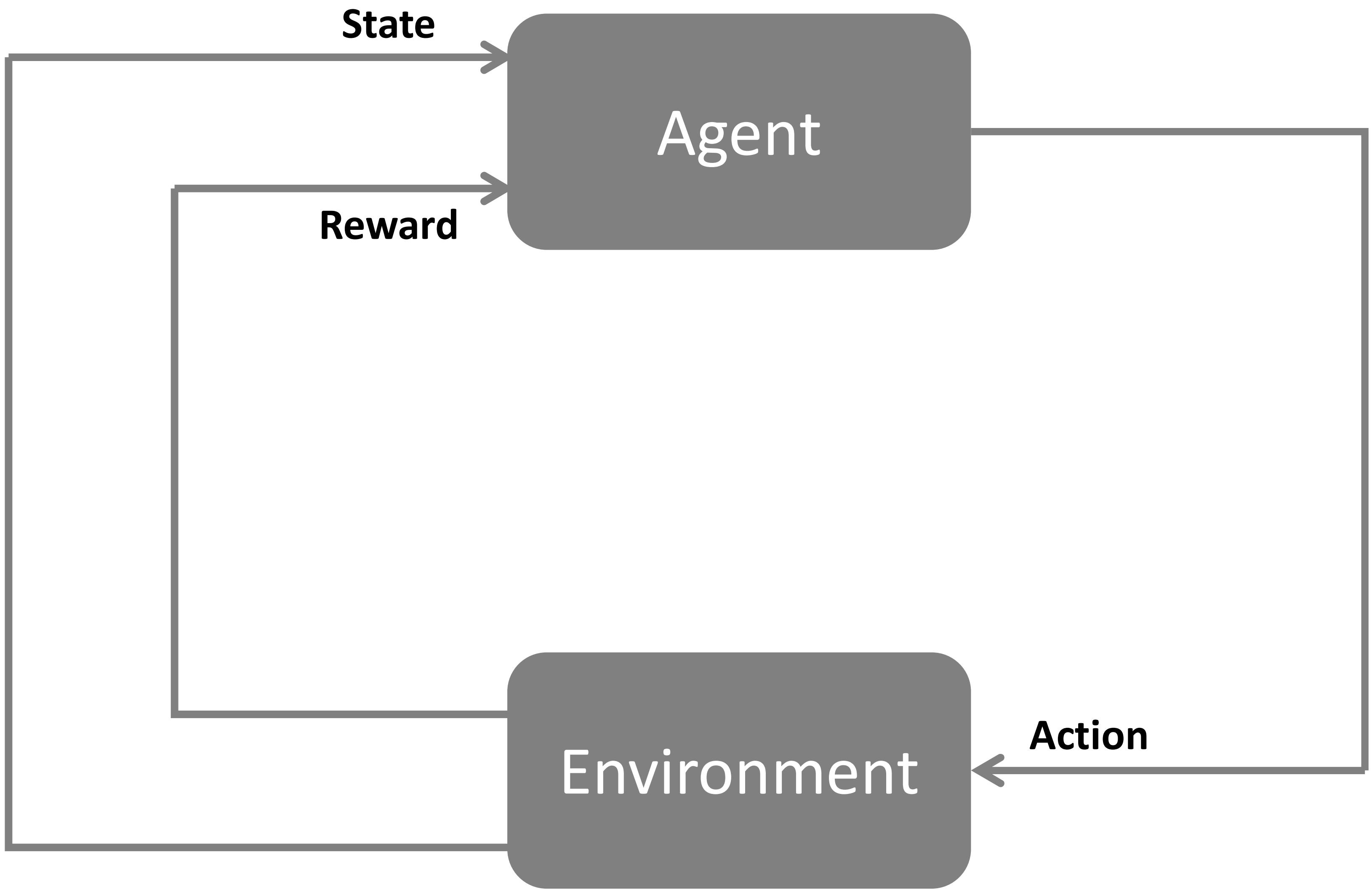
UNKNOWN

  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s')
  - A start state (or distribution)
  - Maybe a terminal state





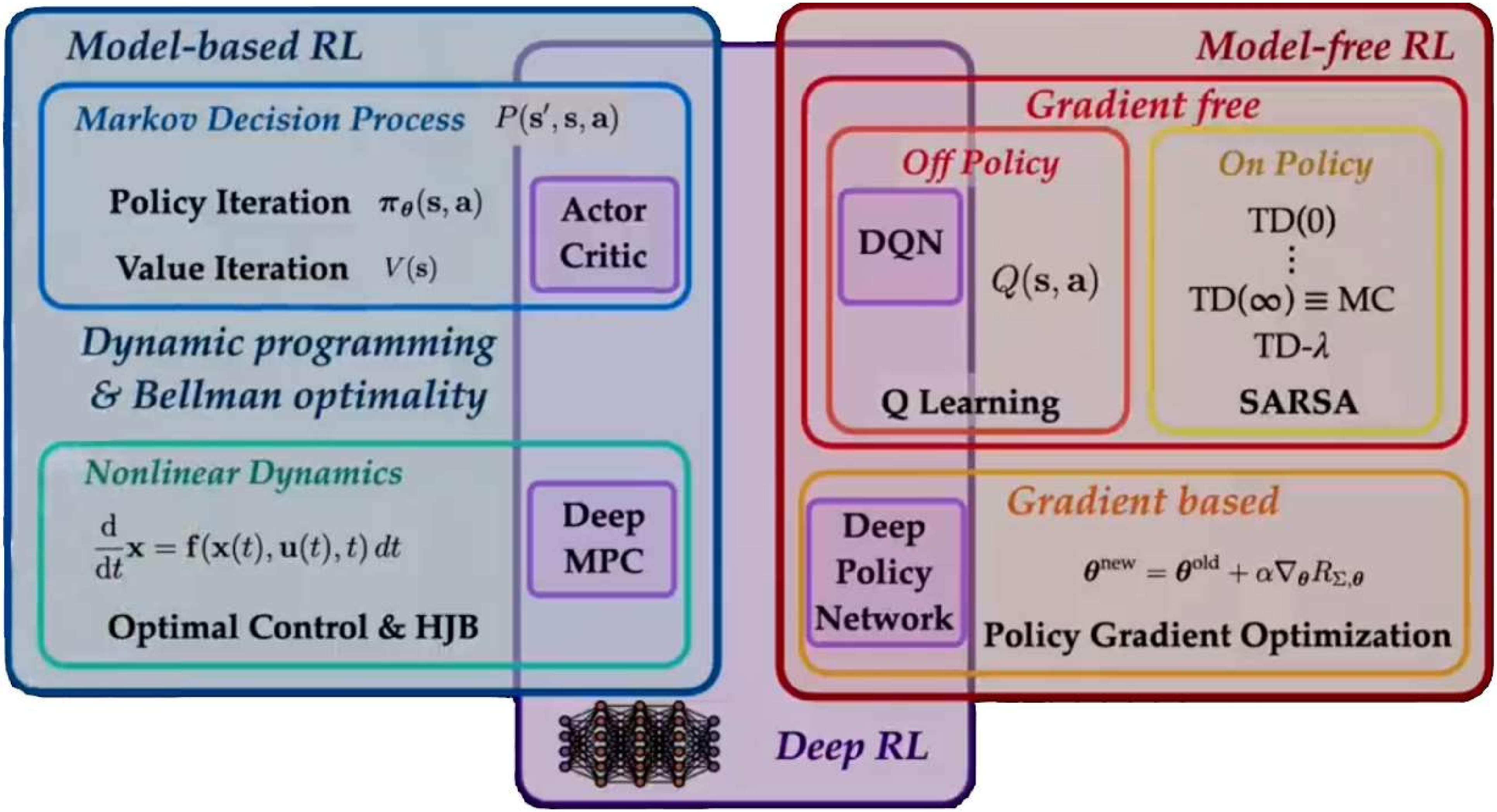
RL: Agents and Environments



MDPs vs. Reinforcement Learning

- MDPs are building blocks for RL
- RL has the additional complexity that the agent does not have access to the full specification of the MDP. For example:
  - Transition probabilities are often unknown
  - Reward function is often unknown

Reinforcement Learning Approaches



HJB: Hamilton-Jacobi-Bellman

RL: Prediction and Control

- **Prediction**
  - **Given a policy, estimate the utility/value function**
- **Control**
  - **Learn the optimal policy**

Model-free vs. Model-based

- **Model-free:**
  - **The agent does not have and does not learn a model of the how the environment works**
- **Model-based:**
  - **The agent learns/improves a model of the environment**
- **Note: we are not talking about an approximate "model" of a state representation**
  - **rather, we mean model of the environment, such as transition probabilities**

RL: On-Policy and Off-Policy

- **On-Policy RL: the agent consistently follows its current policy while exploring the environment (even if suboptimal)**
  - **SARSA**

- **Off-Policy RL: on the other hand, allows the agent to deviate from its current policy and try different actions (even if suboptimal)**
  - **Q-Learning**

RL: On-Policy and Off-Policy

- **On-Policy RL: The behavior/experience is generated by the same policy π that we are trying to improve**
-
- **Off-Policy RL: The behavior/experience is generated by a behavior policy b and we are trying to learn/improve policy π**

Passive vs. Active RL

- **Passive Reinforcement Learning**
  - **agent policy π is known and fixed**
    - **= agent knows which action to pick NOW**
  - **learning state utilities $U(s)$**
    - **possibly environment model (transition function, reward function, etc.) as well**
- **Active Reinforcement Learning**
  - **agent has learn what to do as well**

Approaches

- Prediction
  - Monte Carlo methods
  - Temporal-difference learning, specifically TD(0)
  - Unified view: TD(λ)
- Control
  - Monte Carlo methods
  - Temporal-difference learning: SARSA, N-step TD, TD(λ)
  - Q-learning
- Approximate methods
  - MC prediction
  - TD prediction
  - Semi-gradient SARSA control

Q - Learning

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:

Go through all possible actions a

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

BEST Expected "long-term" utility / value after applying ONE specific BEST action a [Need Environment Model!]

Bellman Equation

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

Go through all possible future states s'

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

Expected "long-term" utility / value after applying ONE specific action a [<u>Need Environment Model!</u>]

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]

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:

Go through all possible future states s'

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

$$Q(s, a)$$

Quality-function: expected utility of taking action a at state s

Utility of State and Q-Function

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

$$\Downarrow$$

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

Q-Function/Utility of State/Policy

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

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

I DON'T NEED to know the TRANSITION function.

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

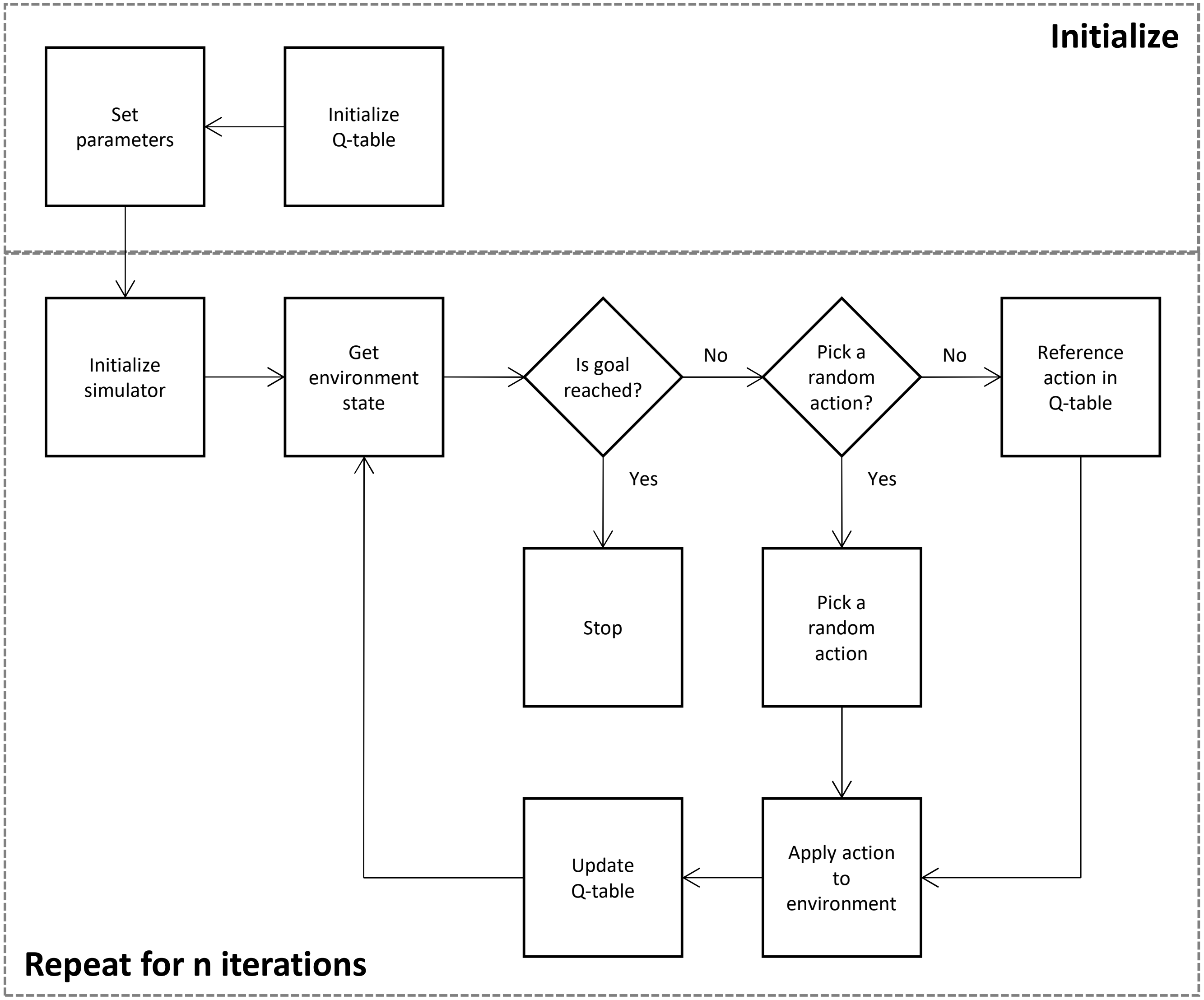
<u>NO need to know what the next state s' is</u>

Q-Learning Agent

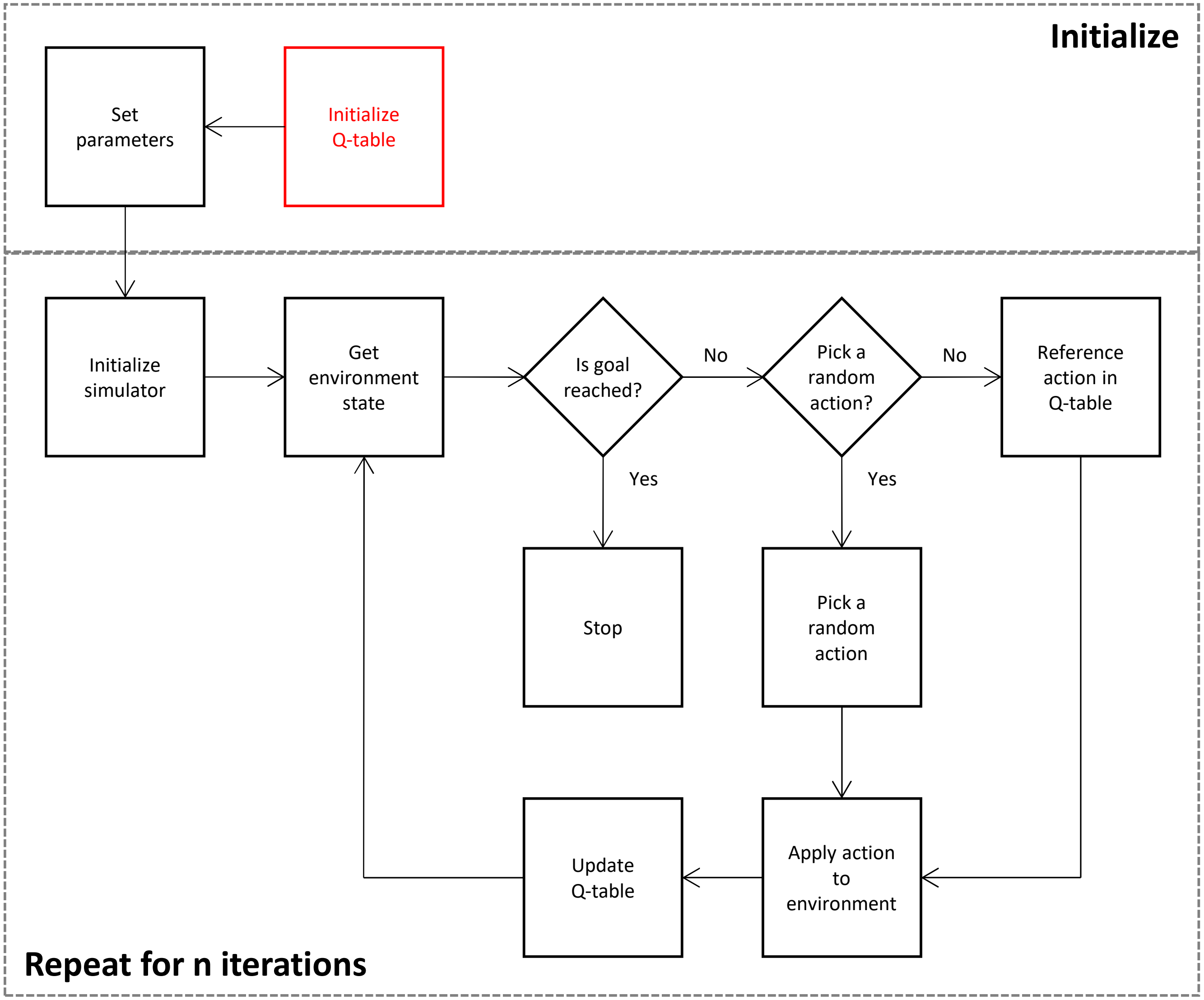
function Q-LEARNING-AGENT($percept$) **returns** an action
  inputs: $percept$, a percept indicating the current state s' and reward signal r'
  persistent: Q, a table of action values indexed by state and action, initially zero
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if TERMINAL?(s) **then** $Q[s, None] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s, a]$
    $Q[s,a] \leftarrow Q[s,a] + \alpha(N_{sa}[s,a])(r + \gamma \max_{a'} Q[s',a'] - Q[s,a])$
  $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s',a'], N_{sa}[s',a']), r'$
  return a

Q-Learning Algorithm



Q-Learning Algorithm

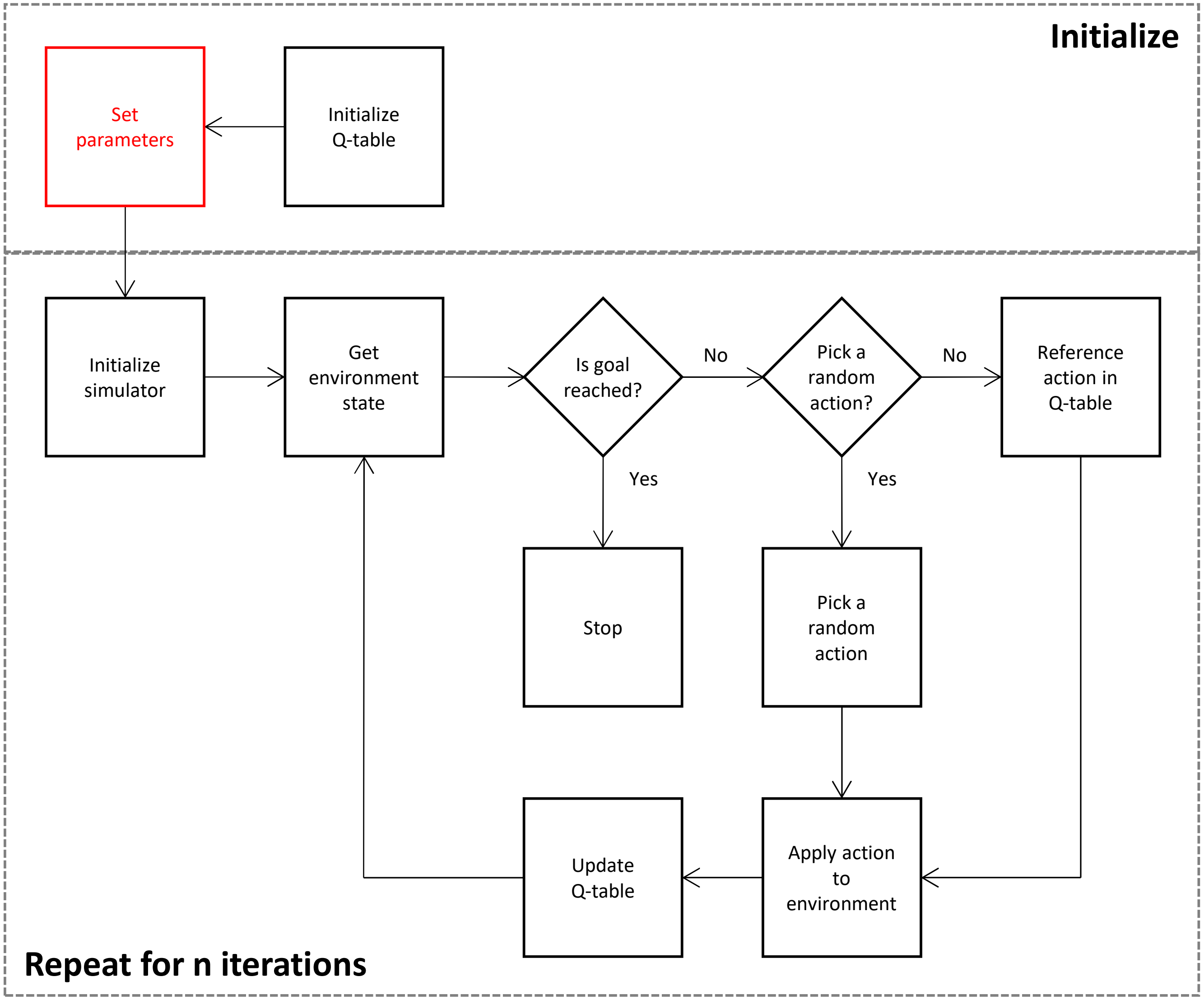


Initialize Q-table:
Set up and initialize (**all values set to 0**) a table where:

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

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

Q-Learning Algorithm

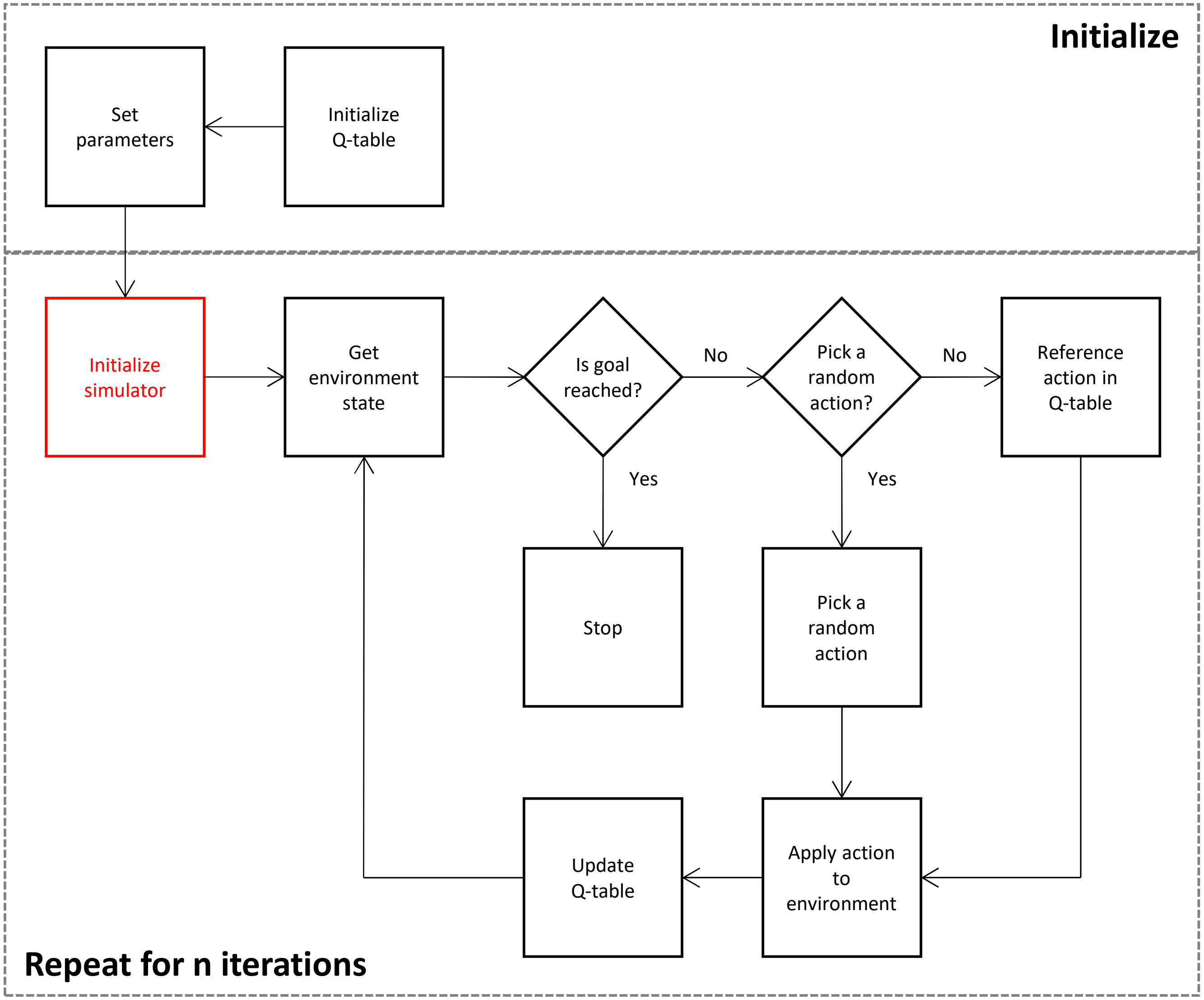


Set parameters:
Set and initialize **hyperparameters** for the Q-learning process.

Hyperparemeters include:

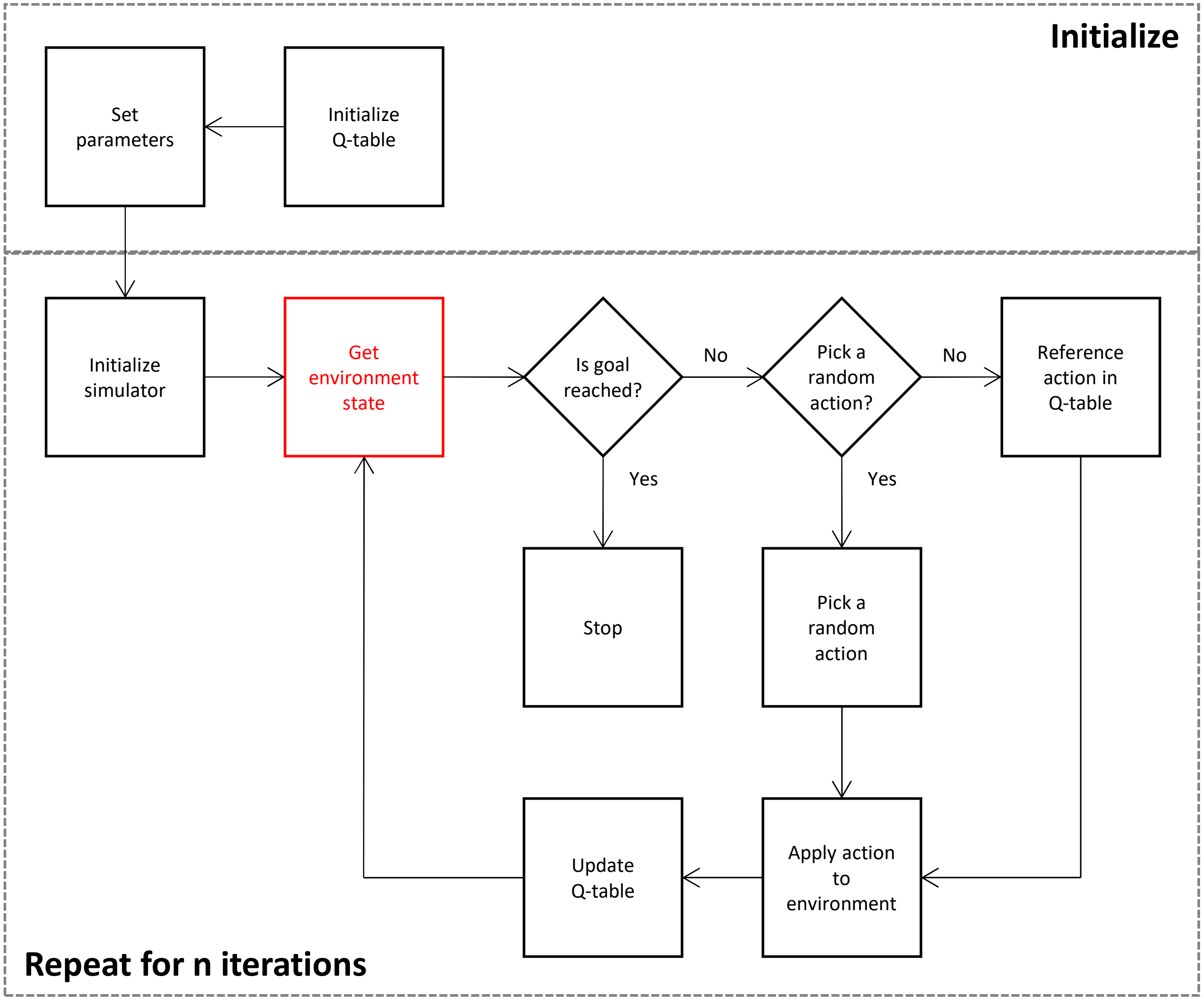
- **chance of choosing a random action**: a threshold for choosing a random action over an action from the Q-table
- **learning rate**: a parameter that describes how quickly the algorithm should learn from rewards in different states
  - high: faster learning with erratic Q-table changes
  - low: gradual 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



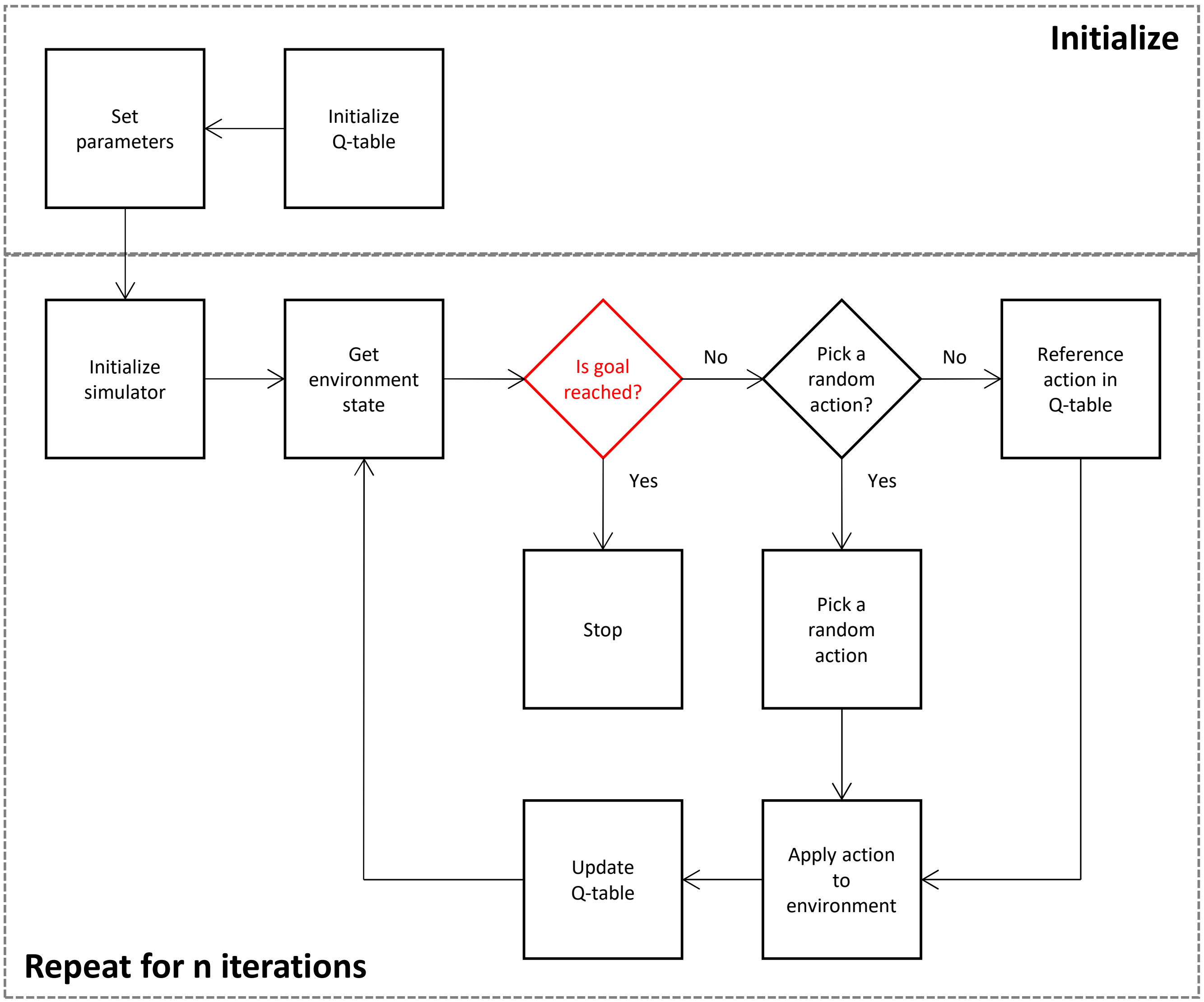
Initialize simulator:
Reset the simulated environment to its initial state and place the agent in a neutral state.

Q-Learning Algorithm



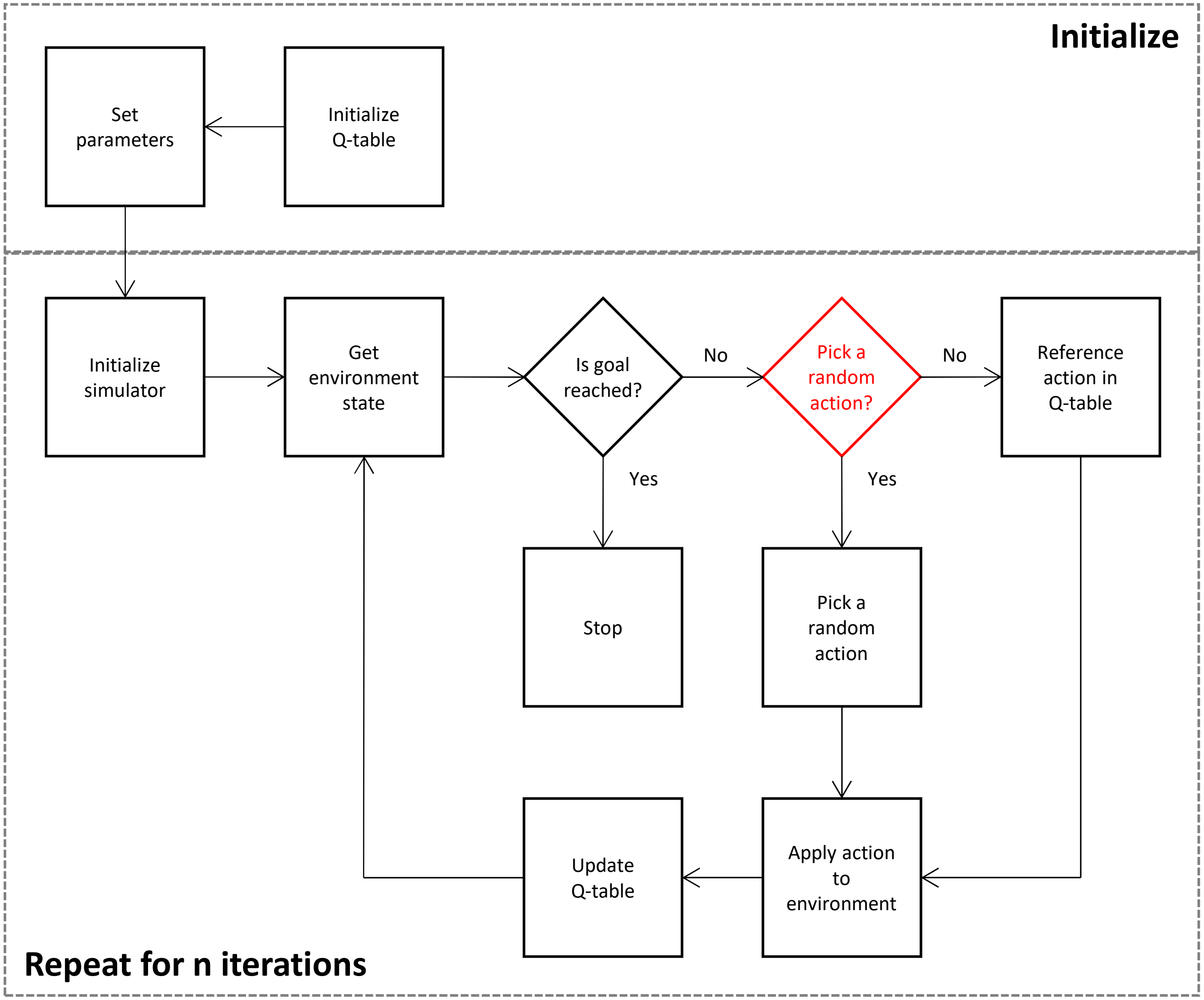
Get environment state:
Report the current state of the environment. Typically a vector of values representing all relevant variables.

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.

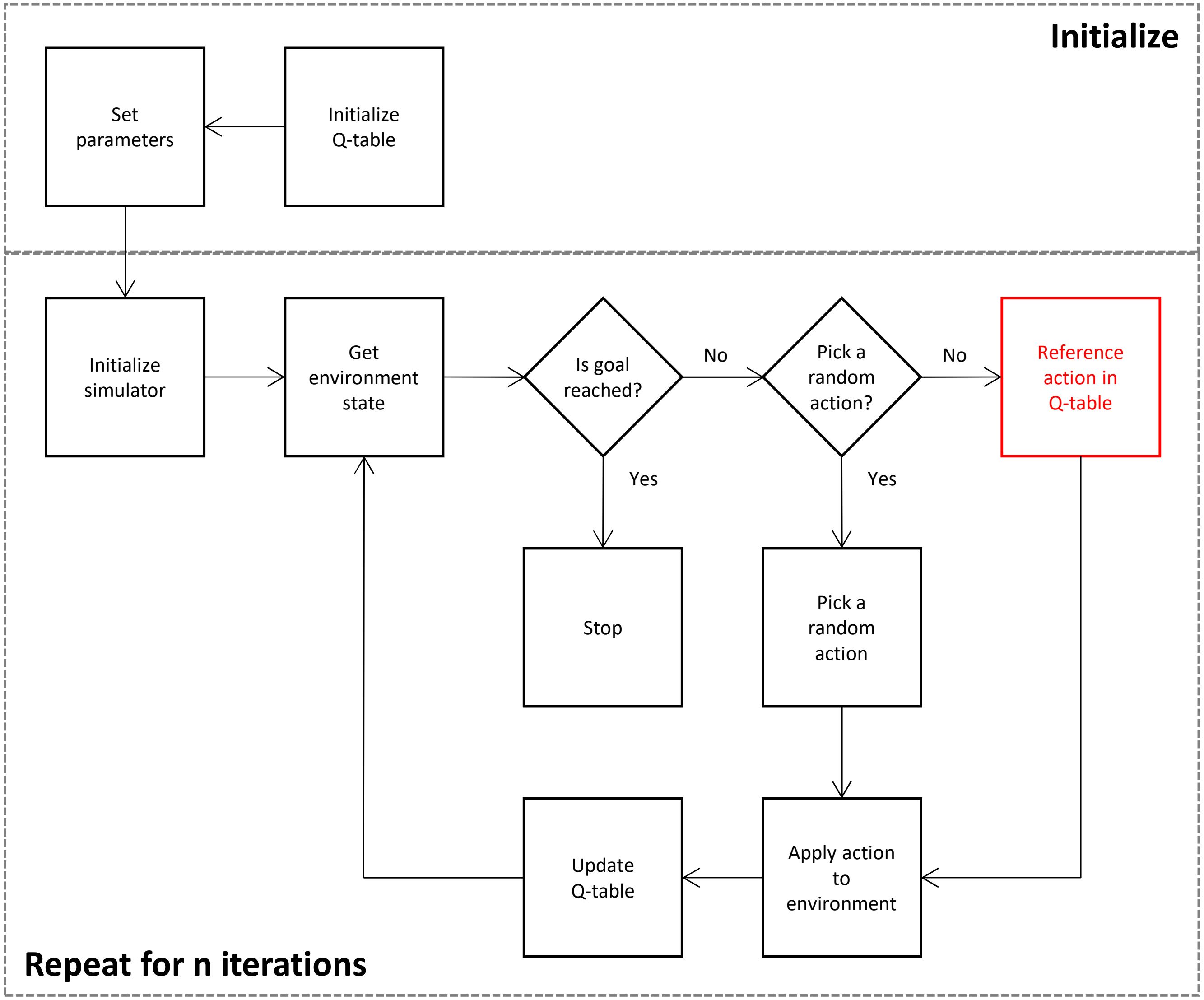
Q-Learning Algorithm



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).

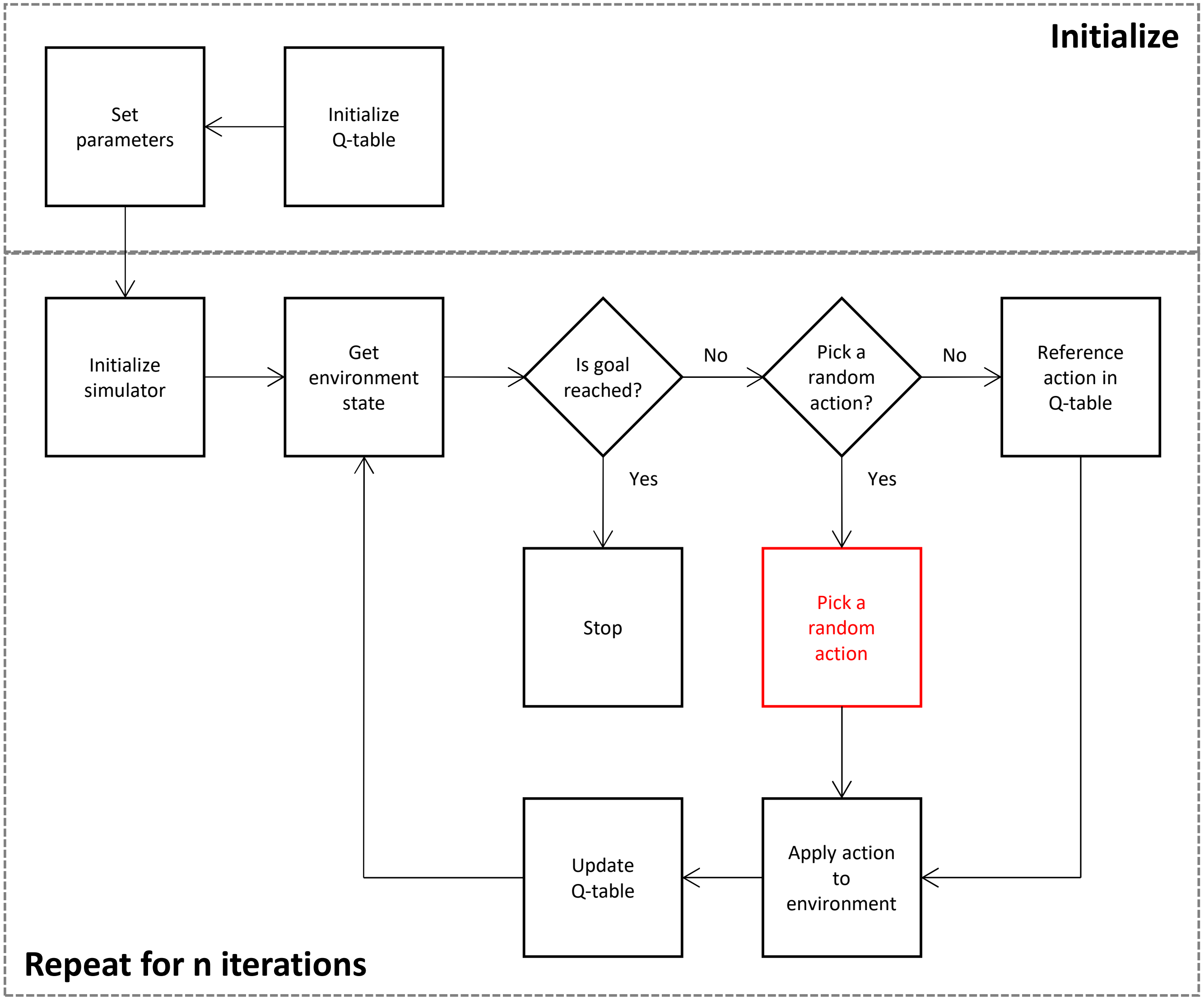
Use the **chance of choosing a random action** **hyperparameter** to decide.

Q-Learning Algorithm



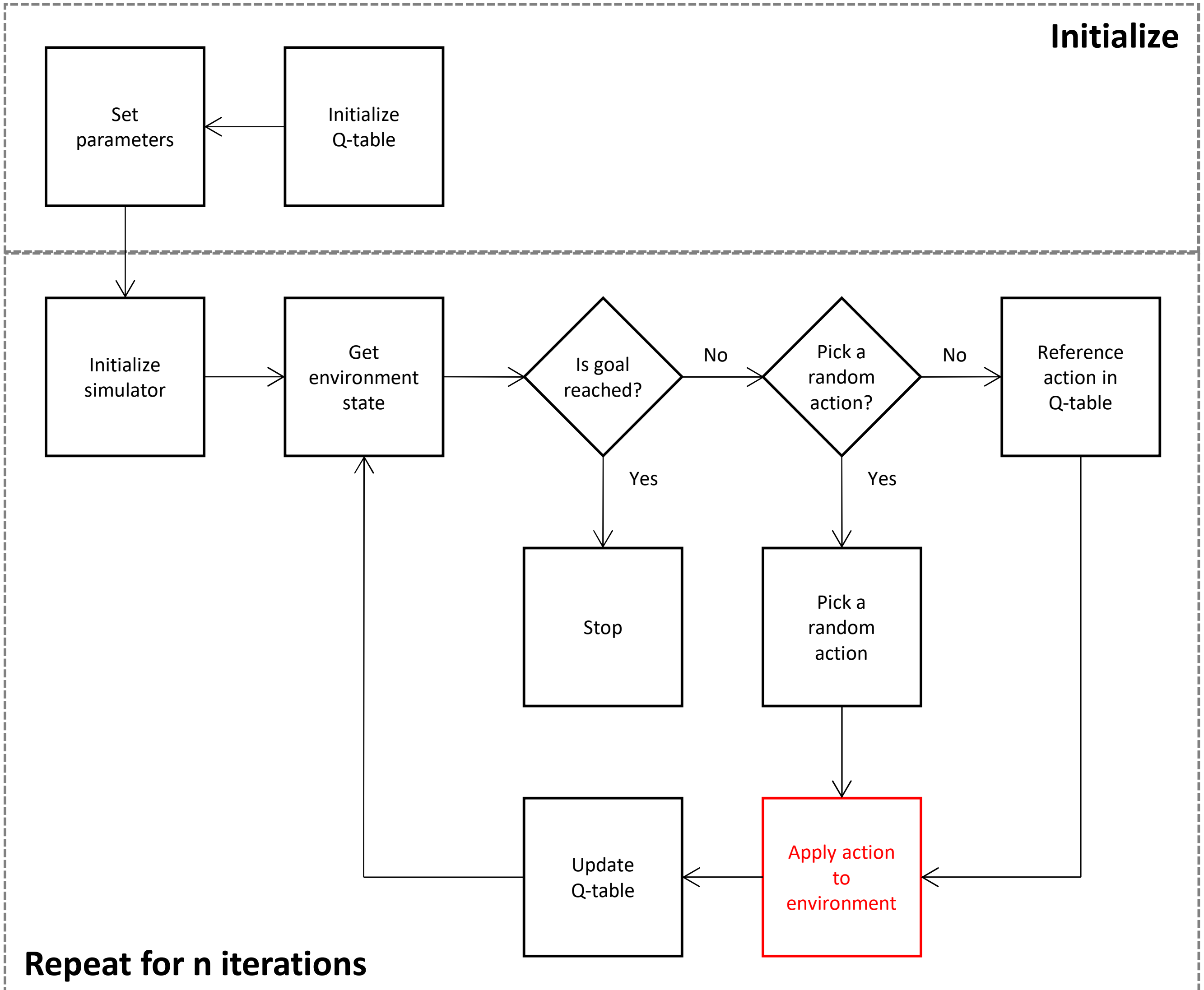
Reference action in Q-table:
Next action decision will be based on data from the Q-table given the current state of the environment.

Q-Learning Algorithm



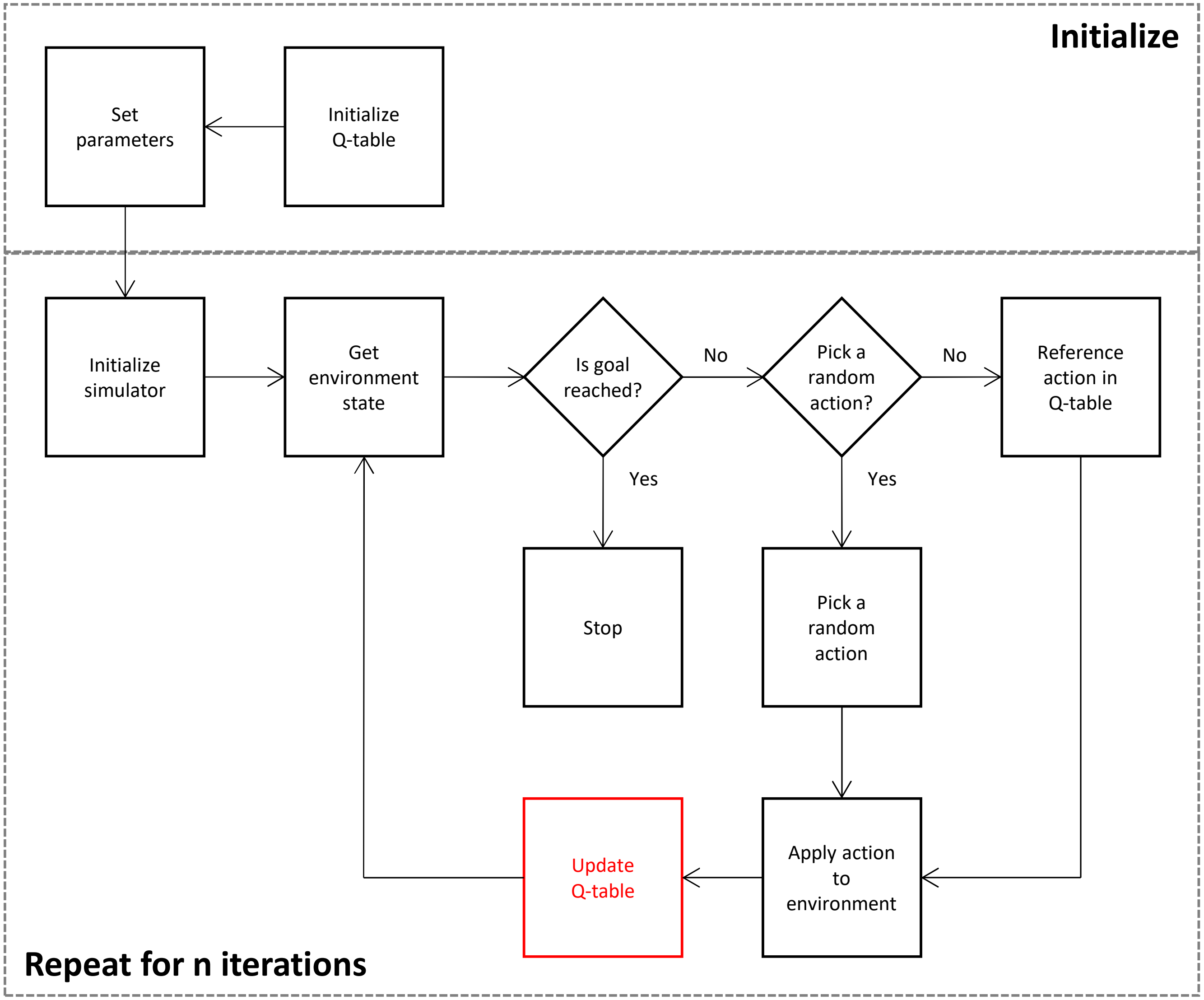
Pick a random action:
Pick any of the available actions at random. Helpful with exploration of the environment.

Q-Learning Algorithm



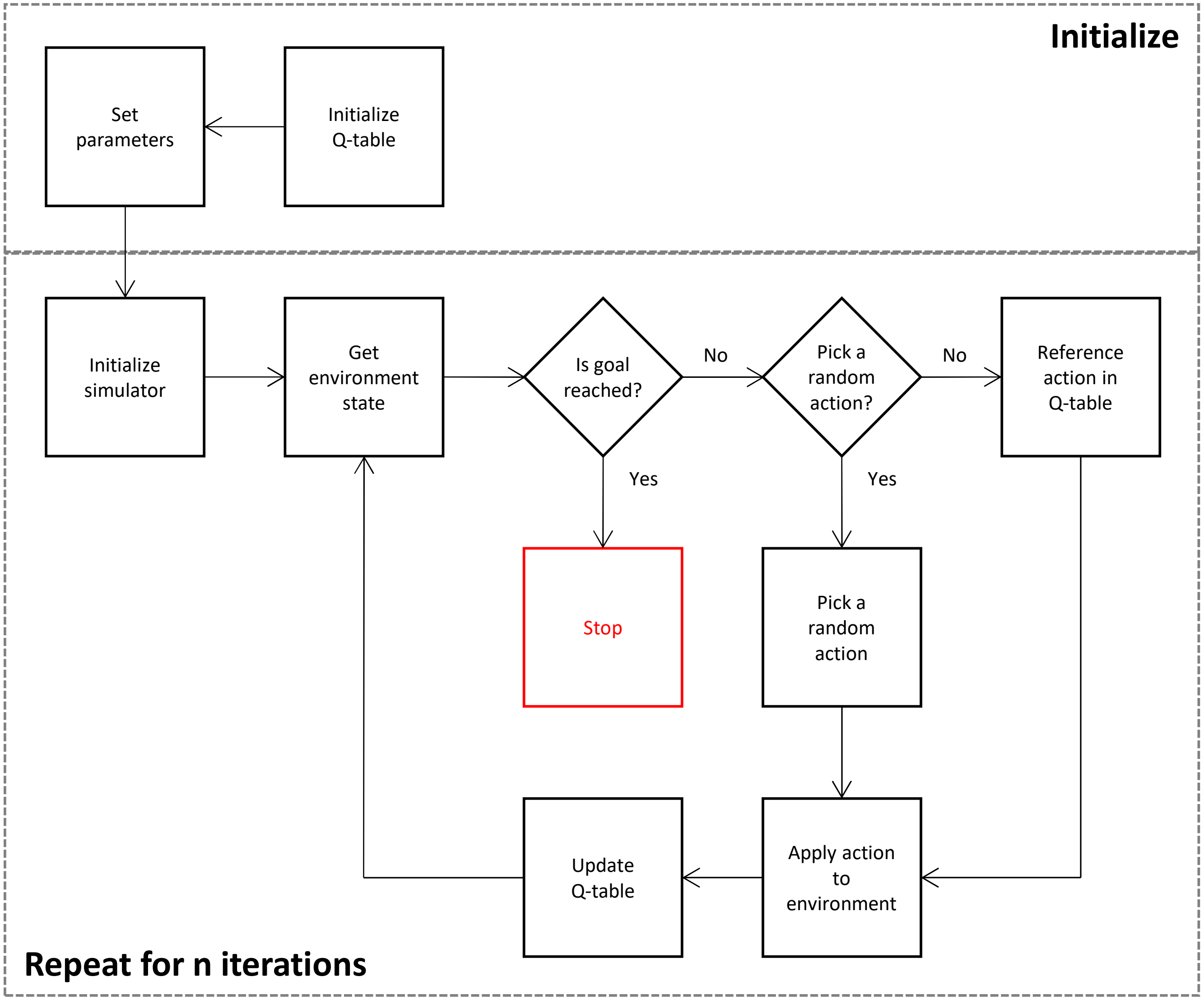
Apply action to environment: Apply the action to the environment to change it. Each action will have its own reward.

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: Example

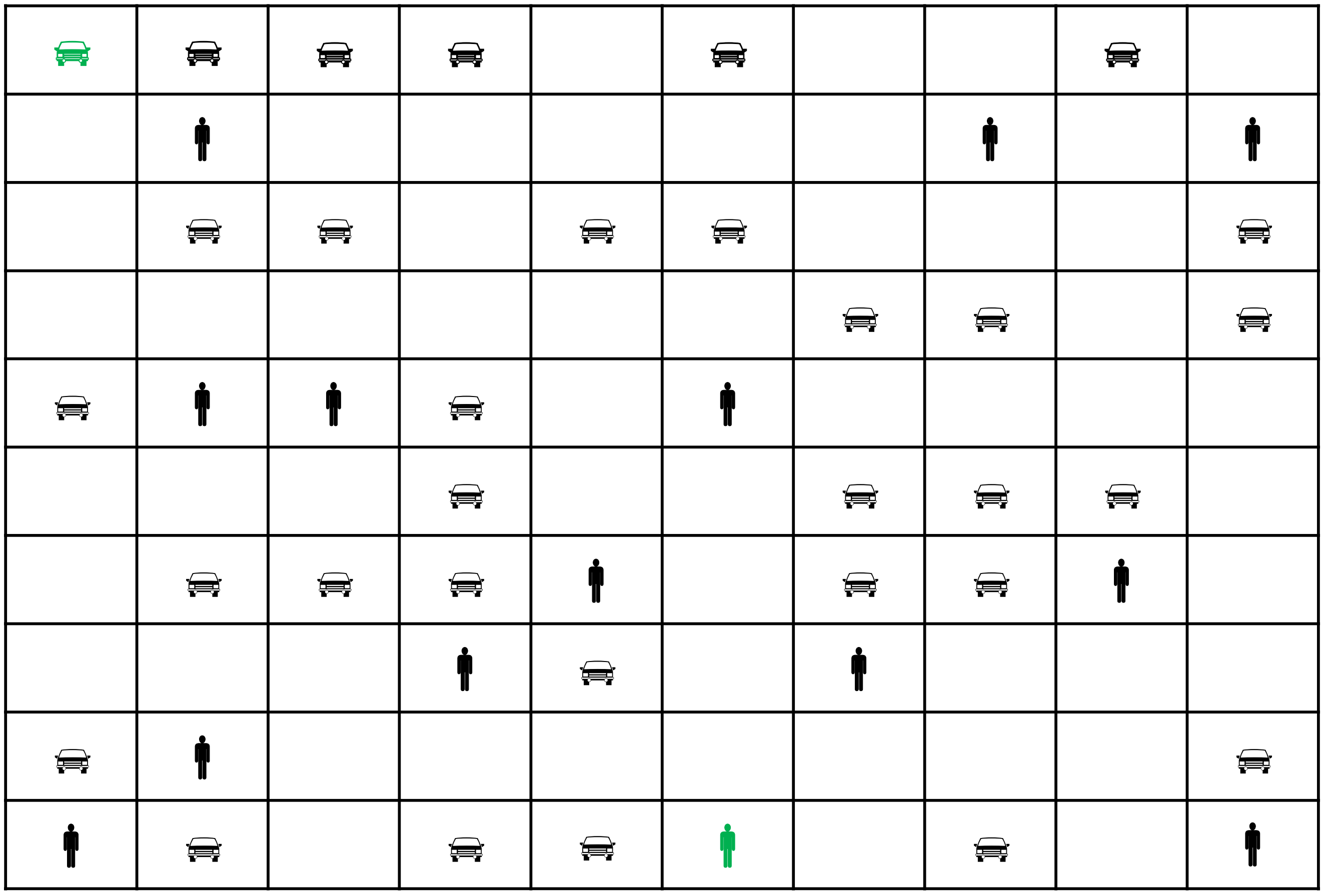
Q-Learning Agent

function Q-LEARNING-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: Q, a table of action values indexed by state and action, initially zero
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if TERMINAL?(s) **then** $Q[s, None] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s, a]$
    $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
  $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a']), r'$
  return a

ASSUME: 1

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:

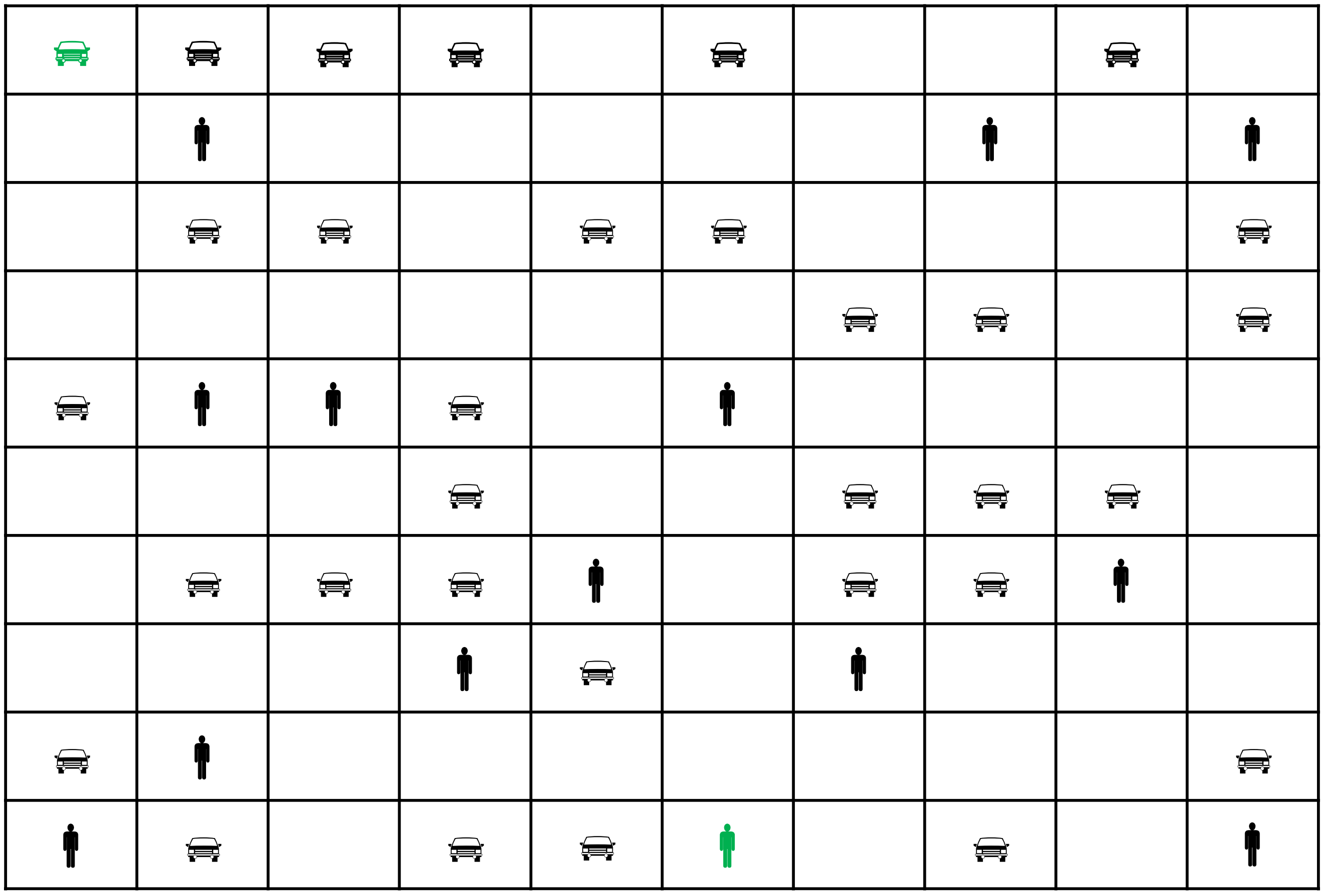
ASSUME: 1

$$Q[s,a] \leftarrow Q[s,a] + \alpha(\boxed{N_{sa}[s,a]})(r + \gamma \max_{a'} Q[s',a'] - Q[s,a])$$

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.**

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

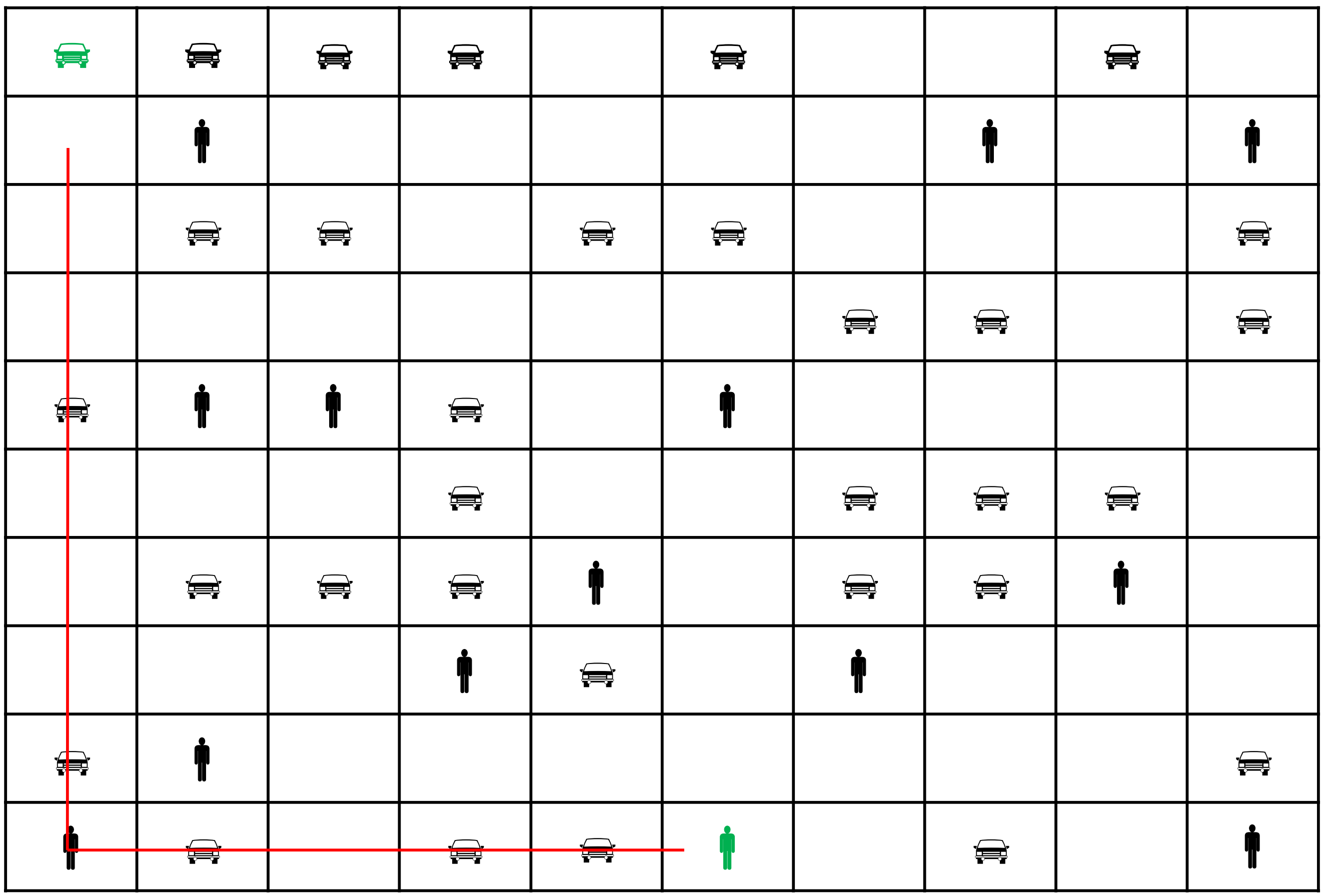
Discount

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

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

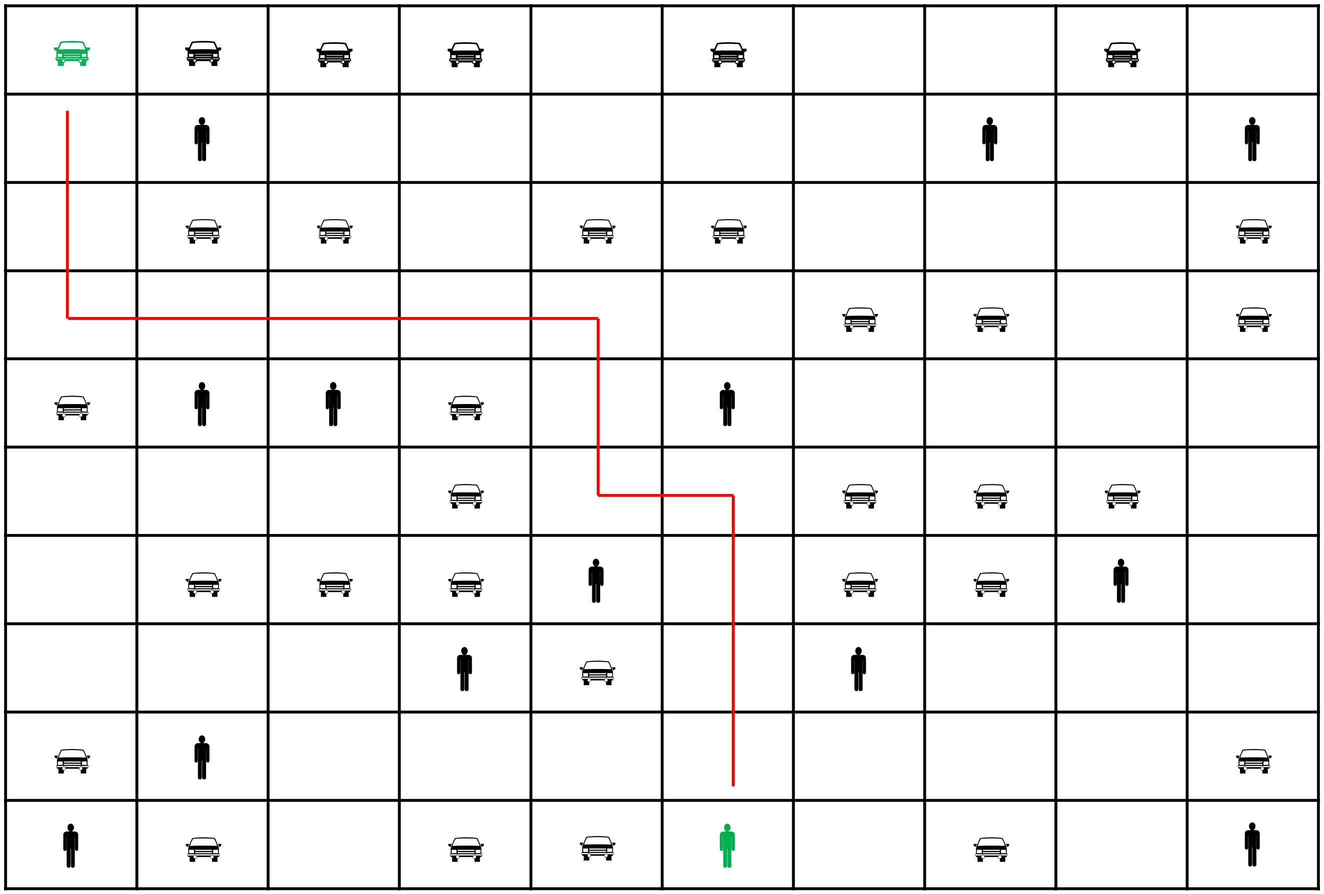
Discount

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

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

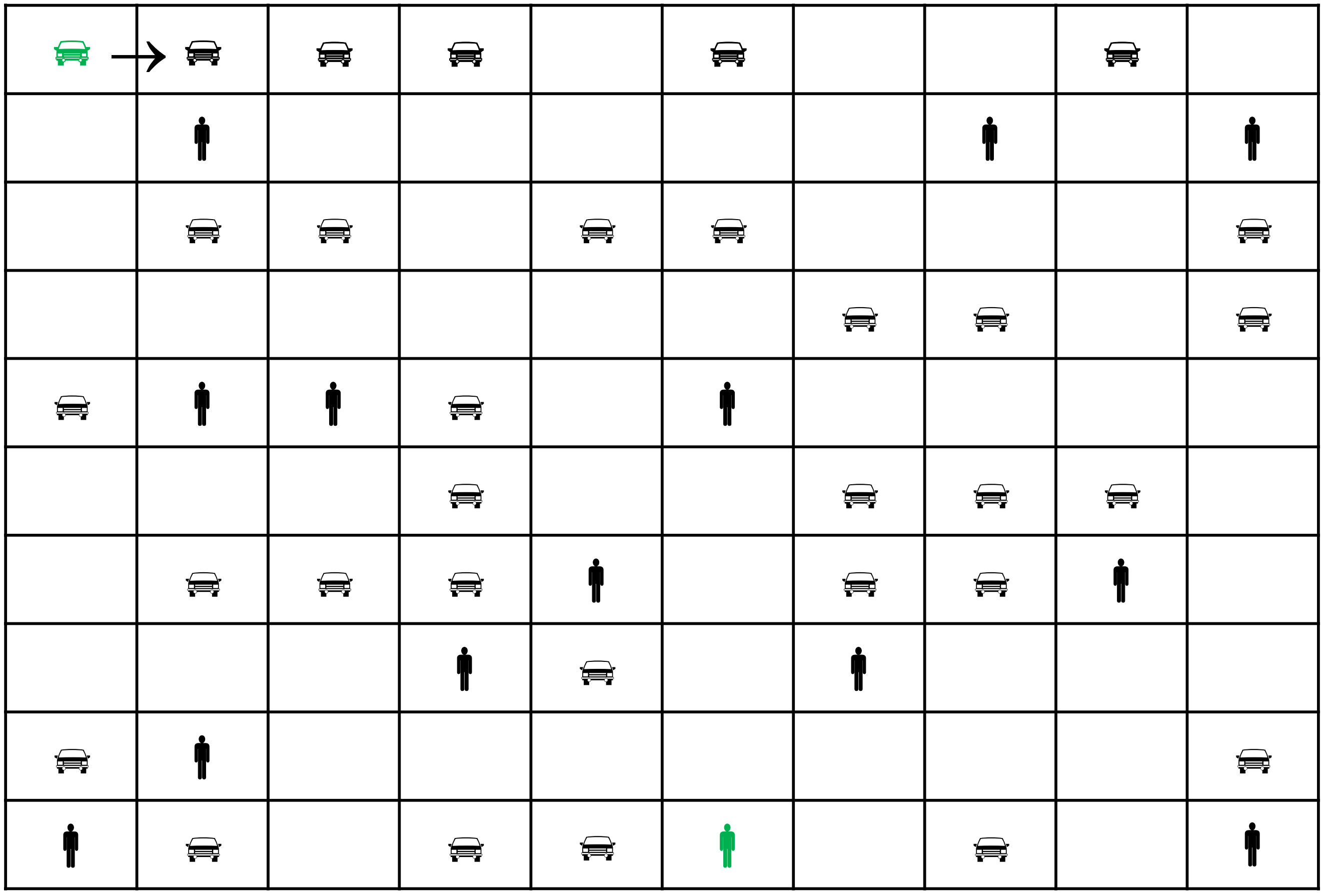
Discount

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

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

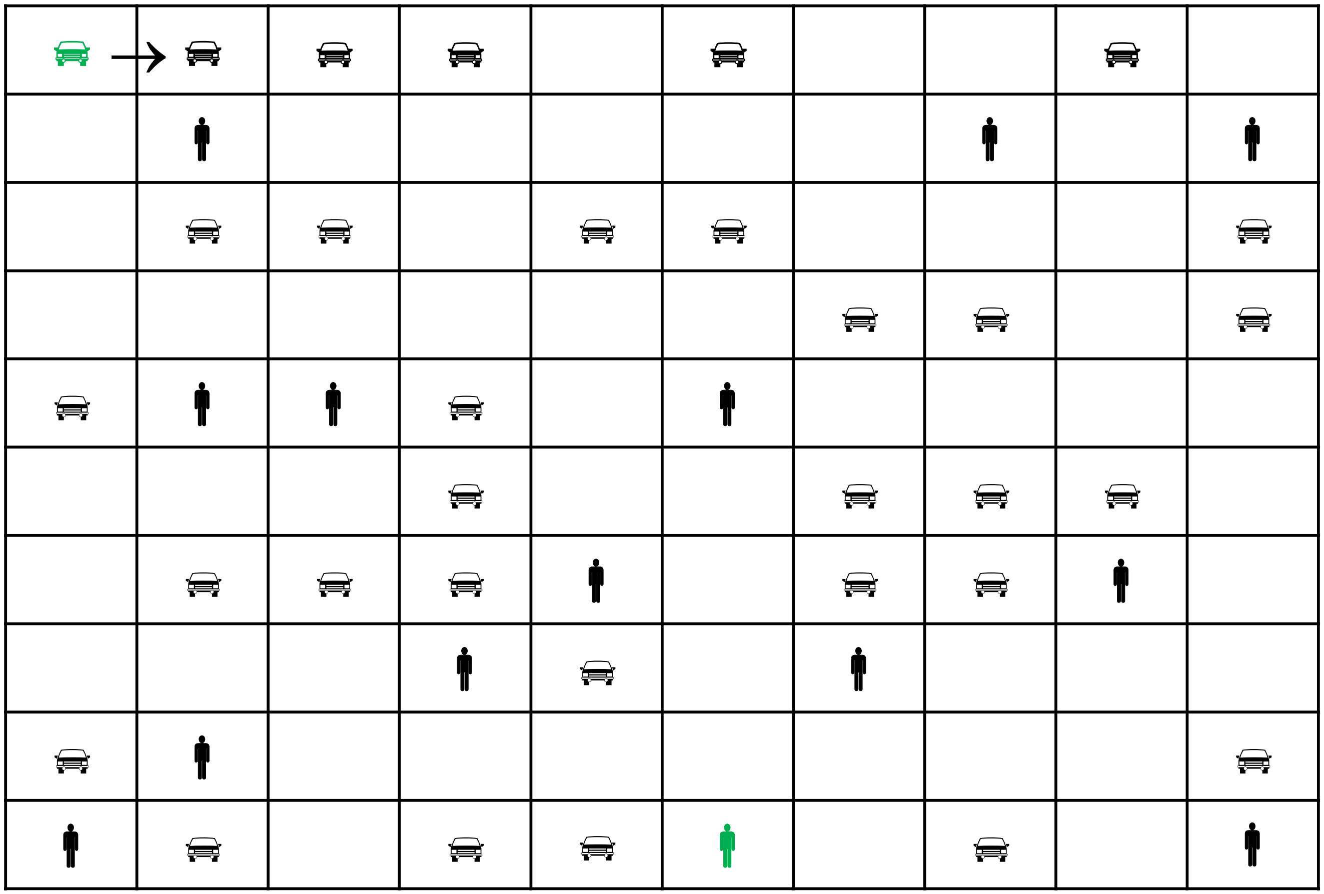
Action: →

Reward: -100

Q-table value:

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

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

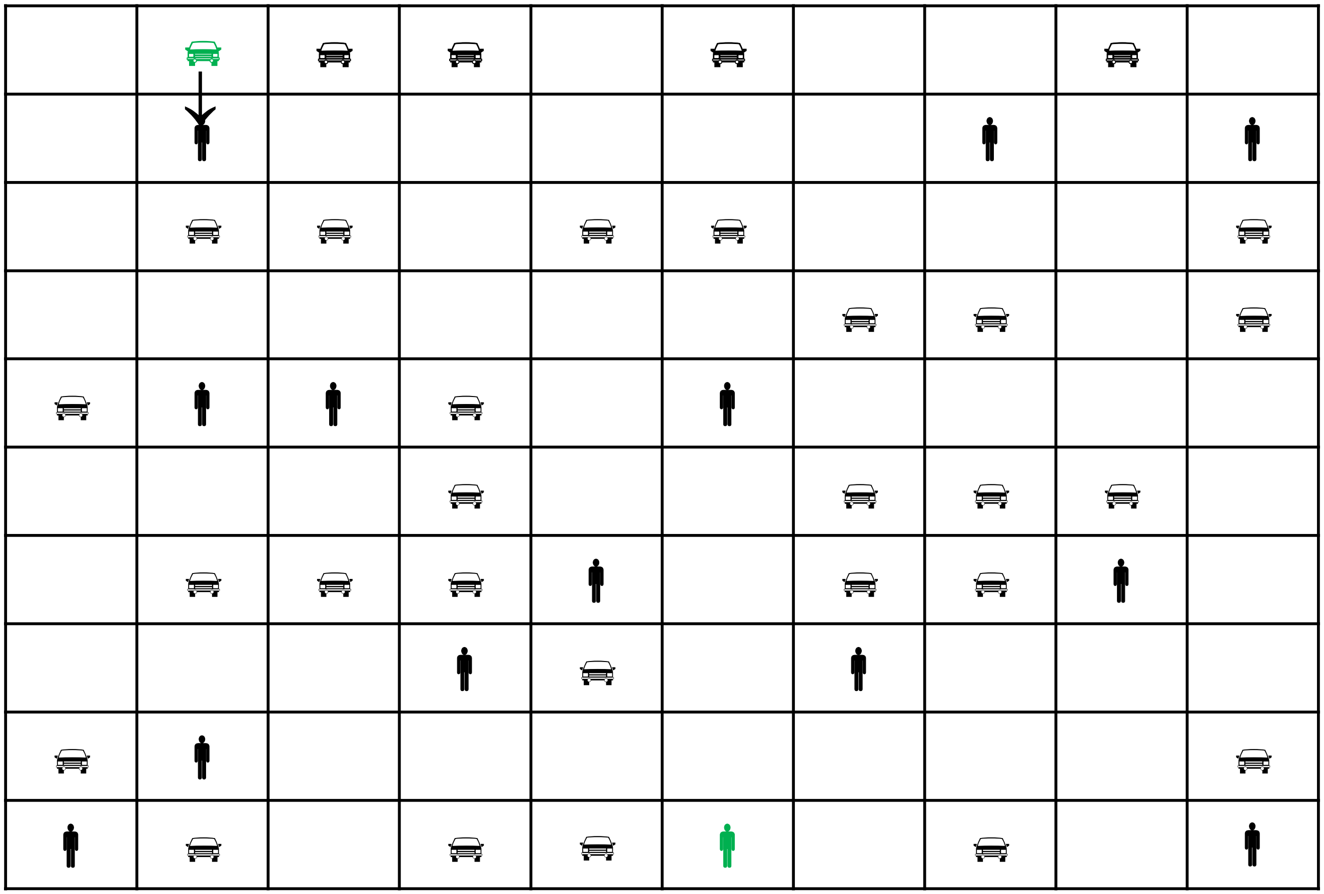
Action: →

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

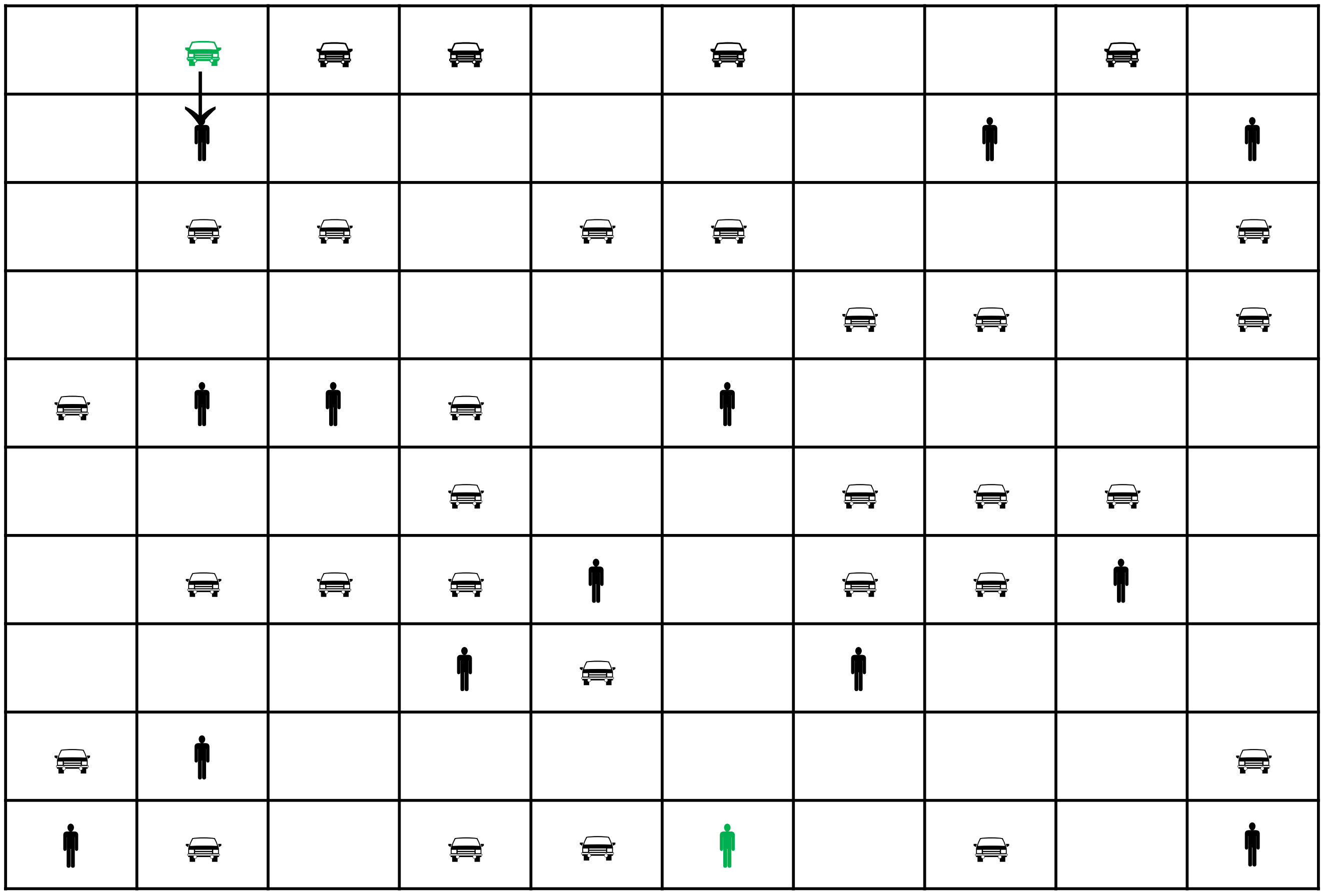
Action: →

Reward: -1000

Q-table value:

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

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

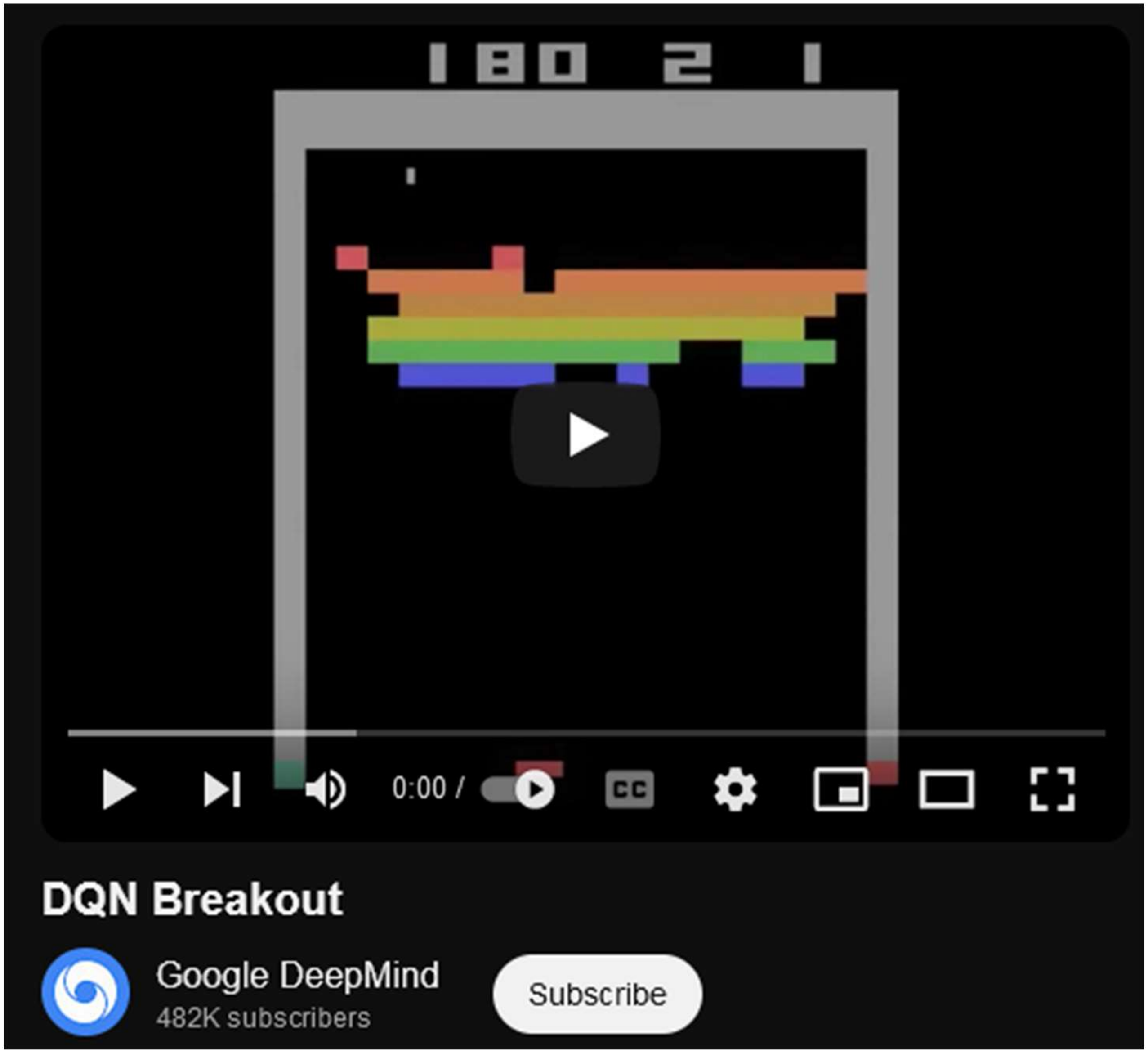
Action: →

Reward: -1000

Q-table value:

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

Deep Q-Learning



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