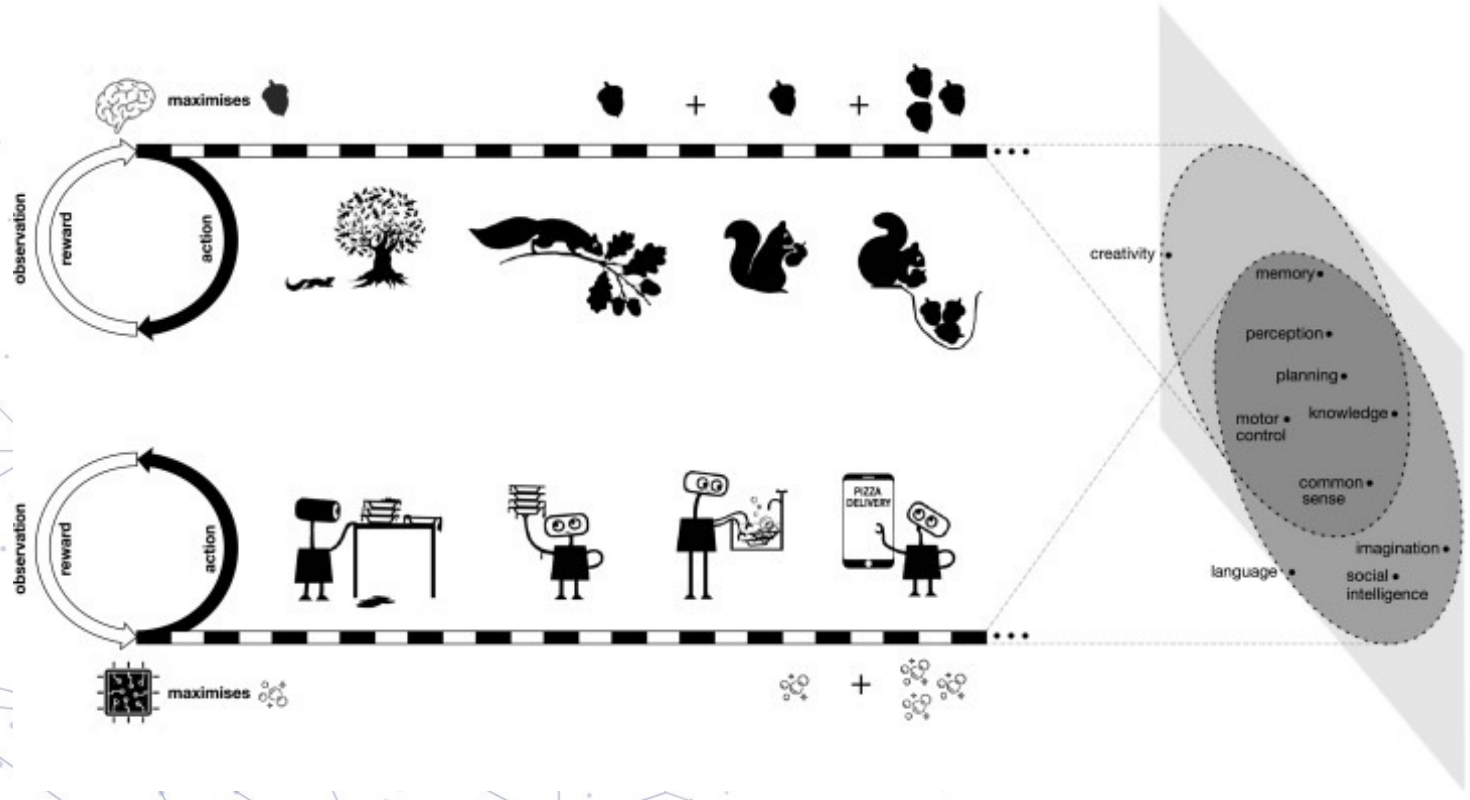
SNT

Introduction to Reinforcement Learning

Marcelo Luis Ruiz Rodríguez

UNIVERSITY OF LUXEMBOURG

What can lead agents (natural or artificial) to behave intelligently?



	Squirrel	Cleaning robot
Reward	**minimise hunger**	**A cleaning measure**
Perception	Identify good nuts	Differentiate clean and dirty utensils
Knowledge	Understand nuts	Understand utensils
Motor control	Collect nuts	Manipulate utensils
Planning	Choose where to store nuts	Recall locations of utensils
Social intelligence	Bluff about locations of the stored nuts, to ensure they are not stolen	Encourage young children to make less mess

Silver, D., Singh, S., Precup, D., & Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*, *299*, 103535.

What can lead agents (natural or artificial) to behave intelligently?

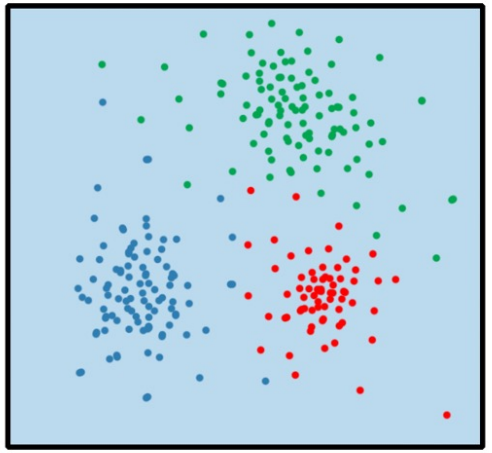
Hypothesis (Reward-is-Enough)

Intelligence, and its associated abilities, can be understood as subserving the maximisation of reward by an agent acting in its environment.

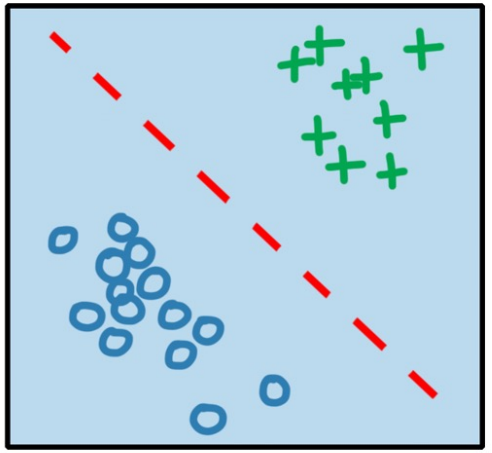
Silver, D., Singh, S., Precup, D., & Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*, *299*, 103535.

machine learning

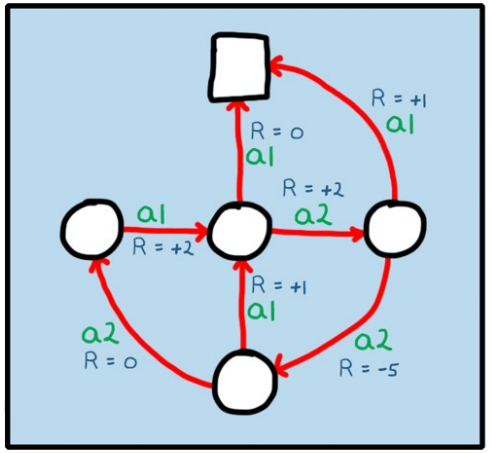
unsupervised learning



supervised learning



reinforcement learning



Applications

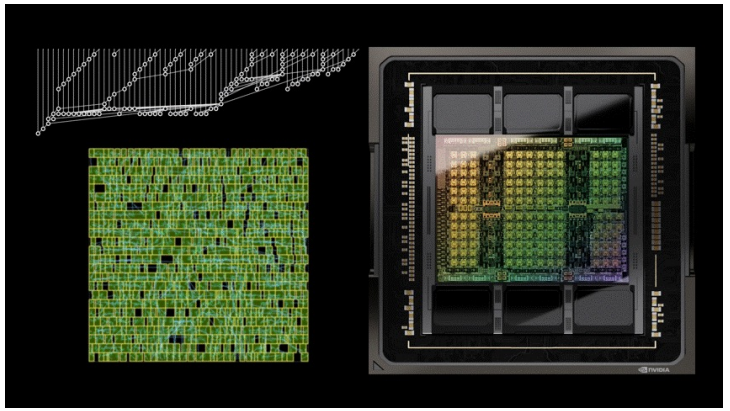
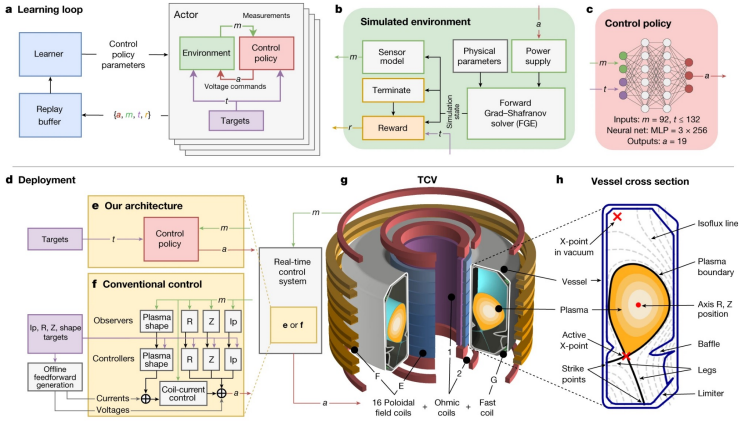


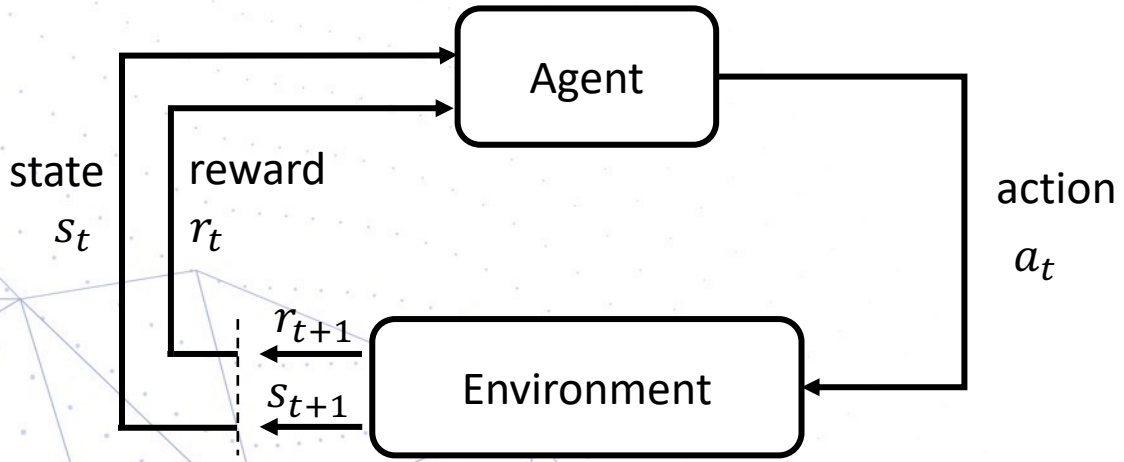
Fig. 1: Representation of the components of our controller design architecture.

From: Magnetic control of tokamak plasmas through deep reinforcement learning



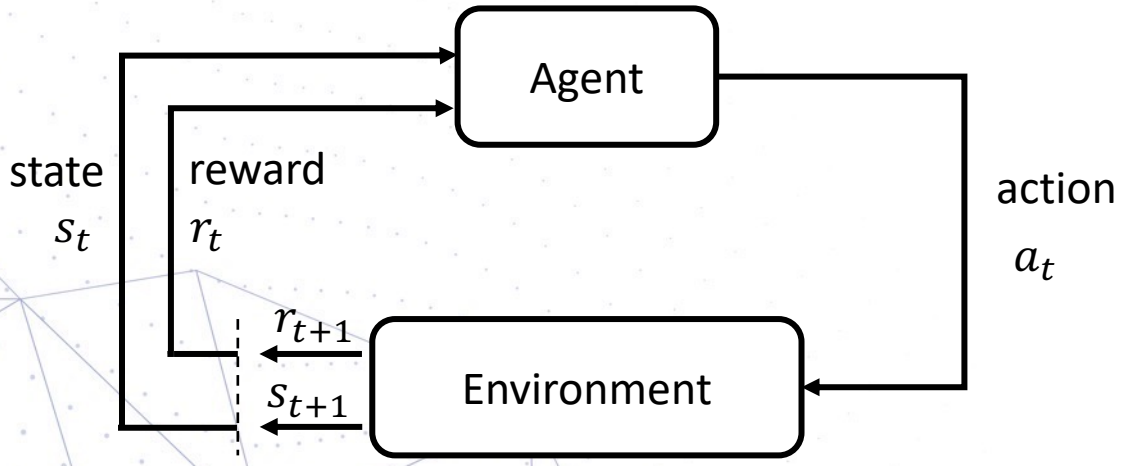
Initial Concepts

Reinforcement Learning



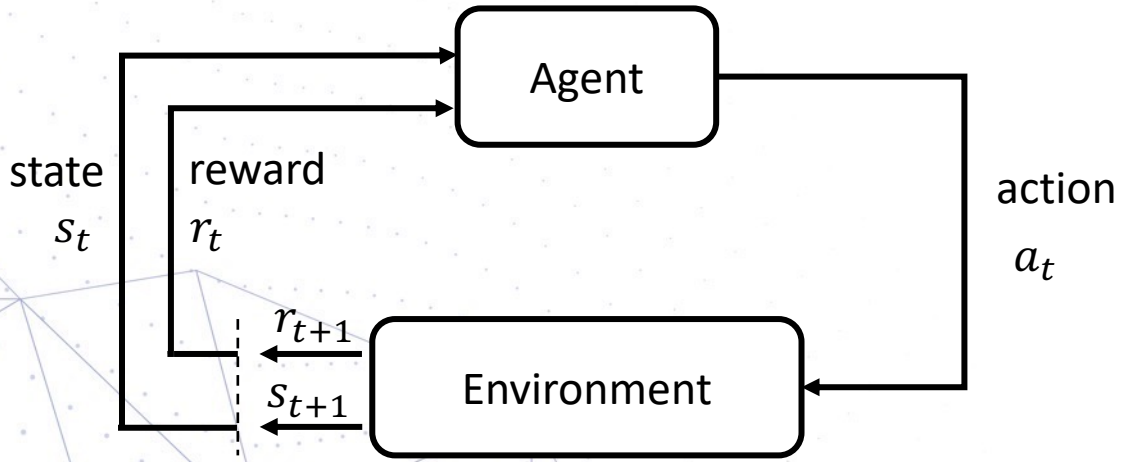
An **agent** is a system that receives at time t a representation of the environment state s_t and outputs an action a_t. The **agent** is the learner and the decision-maker.

Reinforcement Learning



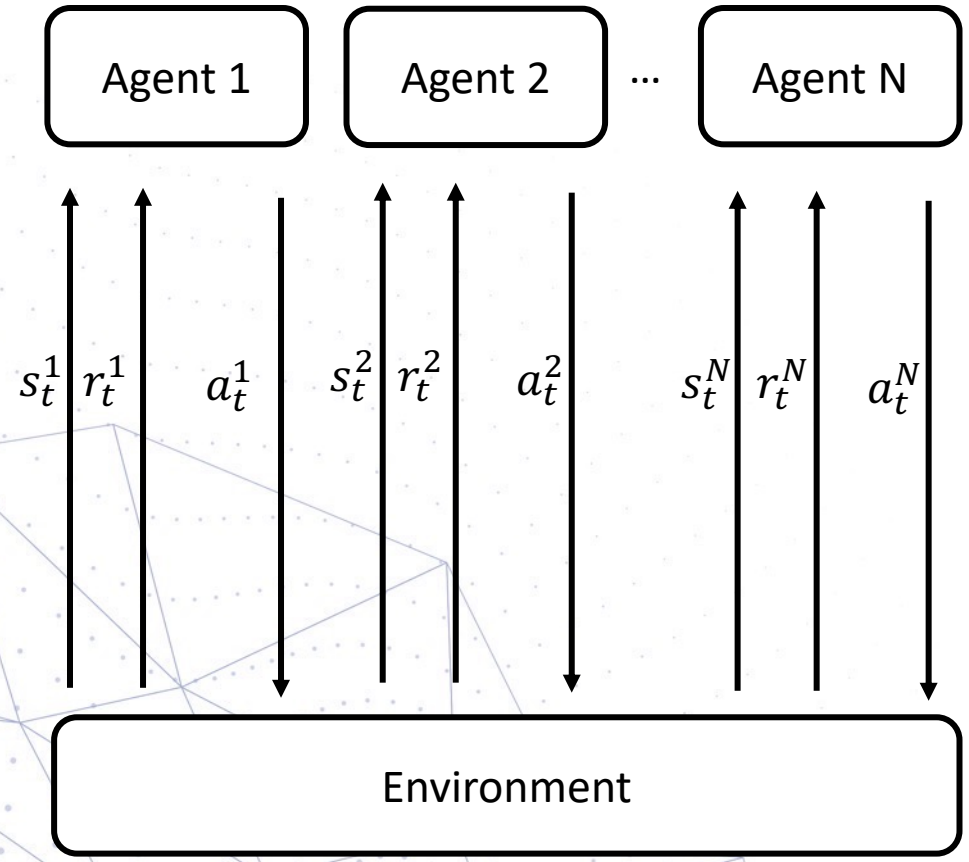
An **environment** is a system that receives action a_t at time t and responds with representation of the environment s_{t+1} at the next time step.

Reinforcement Learning

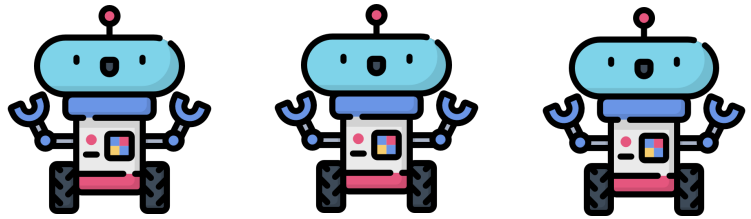


A **reward** is a special scalar observation r_t, emitted at every time-step t by a **reward signal** in the **environment**, that provides an instantaneous measurement of progress towards a **goal**.

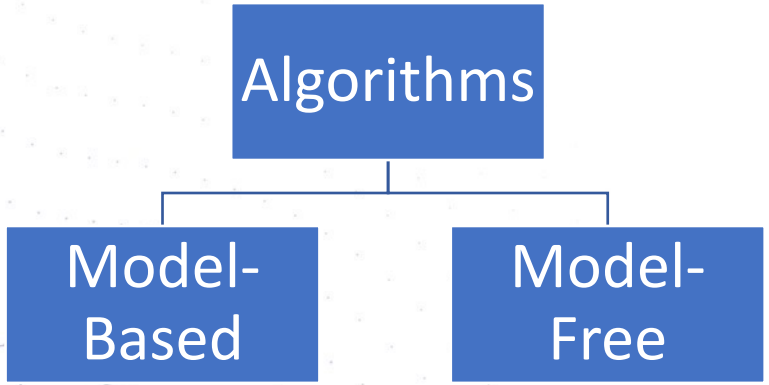
Reinforcement Learning



A **multi-agent system** involves the interaction of multiple agents where each agent receives a representation of the environment and performs its actions independently guided by its own reward signal

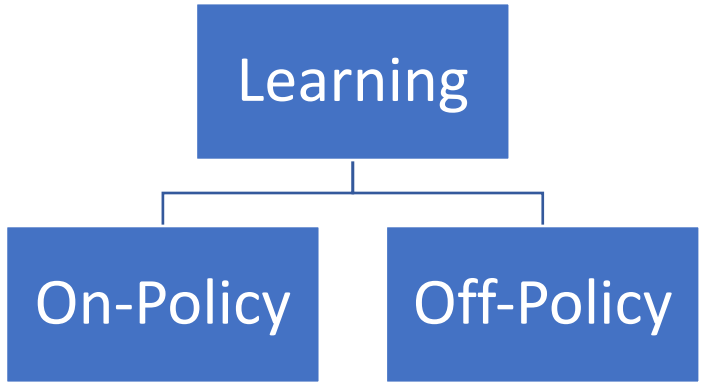


Reinforcement Learning



Methods which use a model to plan actions before they are taken

Methods without a model. Learning to associate actions to returns.



$b = \pi$ $b \neq \pi$

Behavior Policy: $b(a|s)$
Target Policy: $\pi(a|s)$

Markov Decision Process

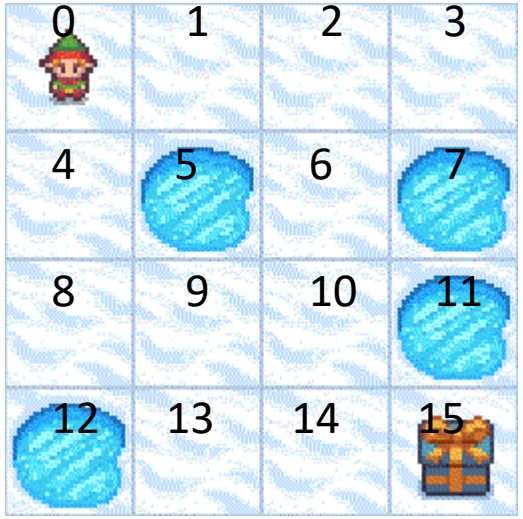
A Markov Decision Process (MDP) is a problem formulation that defines how an agent takes sequential actions to transit between different states under uncertainty while being guided by rewards.

Variable	Description
S	State space
A	Action space
$P : S \times A \rightarrow S$	Transition Function
$R : S \times A \times S \rightarrow \mathbb{R}$	Reward Function
$\gamma \in [0,1]$	Discount Factor

State Space

A set of all possible states an agent can be in (discrete or continuous).

$S = \{0,1,2, \dots, 15\}$

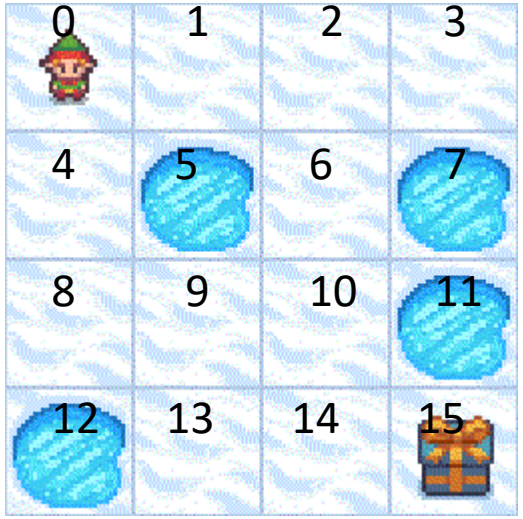


Frozen Lake
16 states

Action Space

A set of all possible action an agent can take (discrete or continuous).

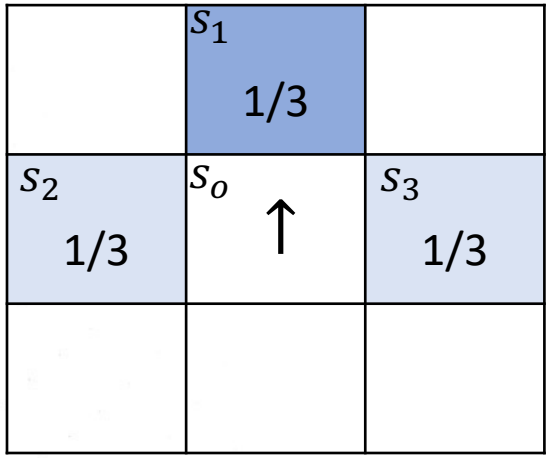
$A = \{\uparrow\downarrow\leftarrow\rightarrow\}$
Up, down, left, right



Frozen Lake
16 states

Transition function

Defines how an agent transits from state s to state s' when taking action a.



$P(s_1|s_0, \uparrow)$=.33

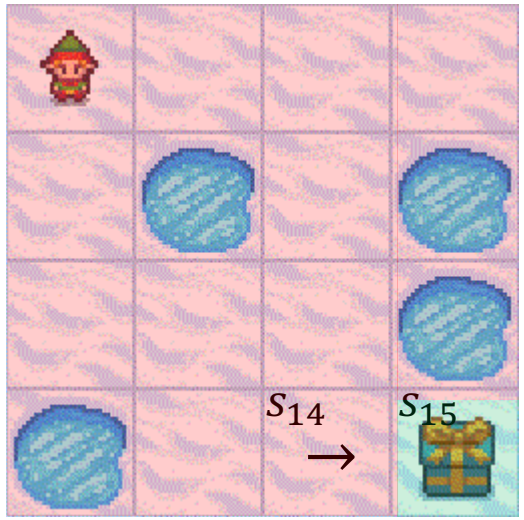
$P(s_2|s_0, \uparrow)$=.33

$P(s_3|s_0, \uparrow)$=.33

Reward Function

Defines the reward that the agent receives when taking action a, to transit from state s to state s'.

$$R(s_{14}, \rightarrow, s_{15}) = 1$$



Discount Factor (γ)

Because of the possibility of infinite sequences of time steps in infinite horizon tasks, we need a way to discount the value of rewards over time; that is, we need a way to tell the agent that getting +1's is better sooner than later.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

The sum of all rewards obtained during an episode is referred to as **the return,**

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_T$$

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

but we can also have **the discounted return.**

$$G_t = R_{t+1} + \gamma G_{t+1}$$

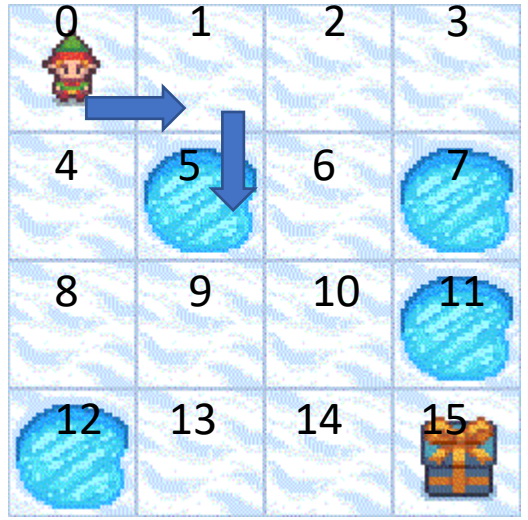
Episodes

A *time step*, is also referred to as epoch, cycle, iteration, is an interaction in the environment.

A trajectory τ is a sequence of states, actions and rewards that the agent followed until reaching a **terminal state**.

$$\tau = (s_0, a_1, r_1, s_1, a_2, r_2 \ldots)$$

$$\tau = (0, \rightarrow, 0, 1, \downarrow, 0{,}5)$$



Frozen Lake

Policy

A policy is the rules used by the agent to determine what actions to take based on a particular observation.

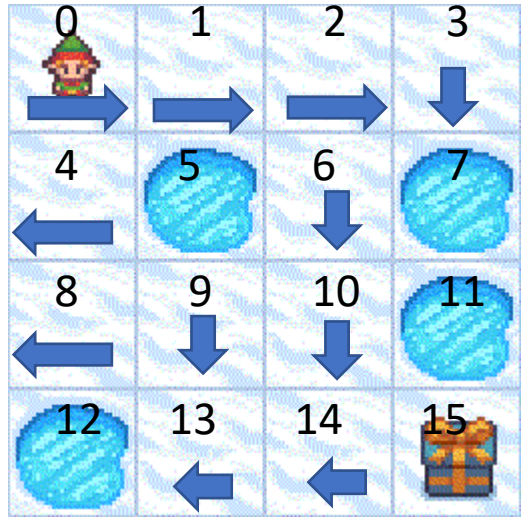
$$a_t = \pi(s)$$

For deterministic policies it can give us the action to follow depending on the state.

$$\downarrow = \pi(6) \qquad 1.0 = \pi(\downarrow|6)$$

But we can have stochastic policies that give us the probability of each action.

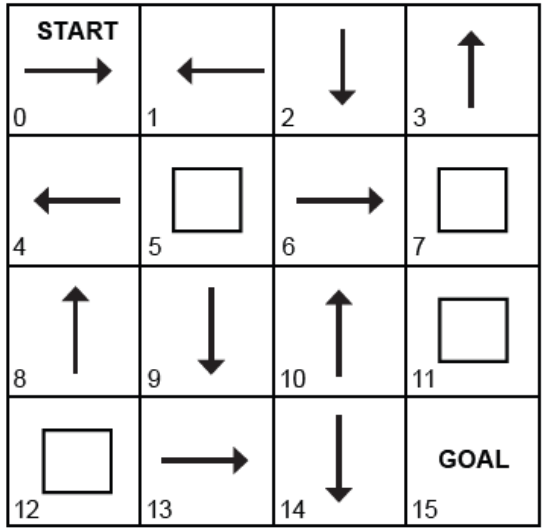
$$.7 = \pi(\downarrow|6)$$
$$.1 = \pi(\uparrow|6)$$
$$.1 = \pi(\leftarrow|6)$$
$$.1 = \pi(\rightarrow|6)$$



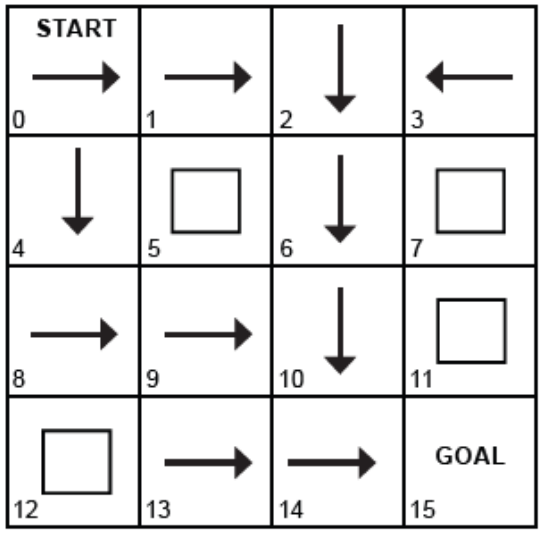
Frozen Lake

Policy

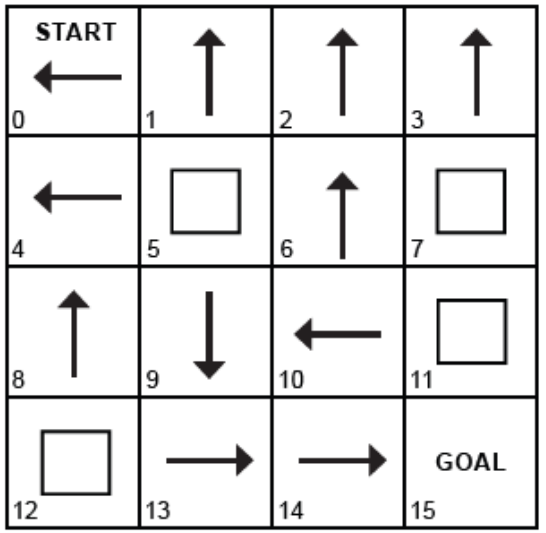
A policy is the rules used by the agent to determine what actions to take based on a particular observation.



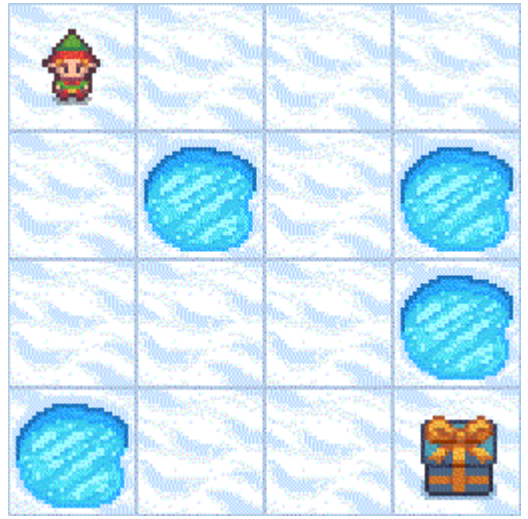
Random generated policy



Policy go-get it



Policy careful



Frozen Lake

We can have different policies
But how do we know which policy is better?

State-value function

We now define the value of a state *s* when following a policy π:

The value of a state *s* under policy π is the expectation of returns if the agent follows policy π starting from state *s*.

$$v_\pi(s) = \mathbb{E}[G_t|S_t = s] \qquad G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_T$$

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |S_t = s]$$

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s]$$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'r} p(s', r|s, a)[r + \gamma v_\pi(s')], \forall s \in S$$

Bellman Equation tell us how to find the value of a state

State-value function

Bellman Equation tell us how to find the value of a state

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'r} p(s', r|s, a)[r + \gamma v_\pi(s')], \forall s \in S$$

We add the reward and the discounted value of the landing state, then weight that by the probability of that transition occurring.

We get the action (or actions, if the policy is stochastic) prescribed for state s and do a weighted sum.

We also weight the sum over the probability of next states and rewards

Do this for all states in the state space.

Action-value function

We now define the value of a state *s* when following a policy π after taking action a:

The value of a state *s* under policy π is the expectation of returns if the agent follows policy π starting from state *s*.

$$q_\pi(s,a) = \mathbb{E}[G_t | S_t = s, A_t = a] \qquad G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_T$$

$$q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s, A_t = a]$$

$$q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a]$$

$$q_\pi(s,a) = \sum_{s'r} p(s', r|s,a)[r + \gamma v_\pi(s')], \forall s \in S, \forall a \in A(s)$$

We do that for all state-action pairs.

Bellman Equation for action-values

What do we weigh? The sum of the reward and the discounted value of the next state.

We don't weigh over actions because we're interested only in a specific action.

We do weigh, however, by the probabilities of next states and rewards

Action-value function

Bellman Equation for action-values

What do we weigh? The sum of the reward and the discounted value of the next state.

$$q_\pi(s,a) = \sum_{s'r} p(s',r|s,a)[r + \gamma v_\pi(s')], \forall s \in S, \forall a \in A(s)$$

We don't weigh over actions because we're interested only in a specific action.

We do weigh, however, by the probabilities of next states and rewards

We do that for all state-action pairs.

Model-Based

SnT

uni.lu
UNIVERSITY OF LUXEMBOURG

Policy Evaluation

We established that policy π is better than or equal to policy π′ if the expected return is better than or equal to π′ for all states.

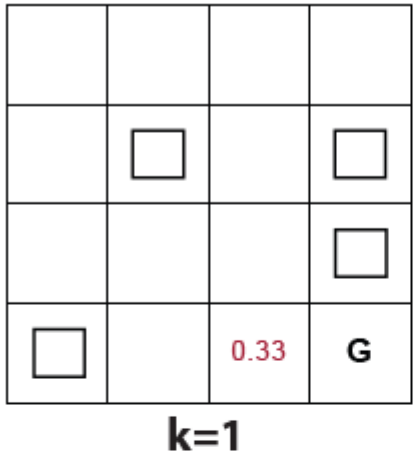
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'r} p(s'|s,a)[r + \gamma v_\pi(s')]$$

Policy Evaluation

We have a deterministic policy, so this part is 1

Let's use $\gamma = 1$

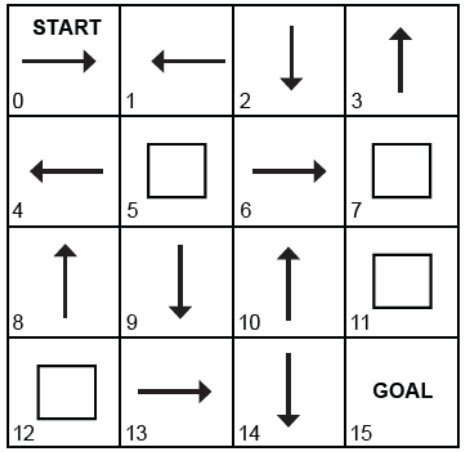
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'r} p(s'|s,a)[r + \gamma v_\pi(s')]$$



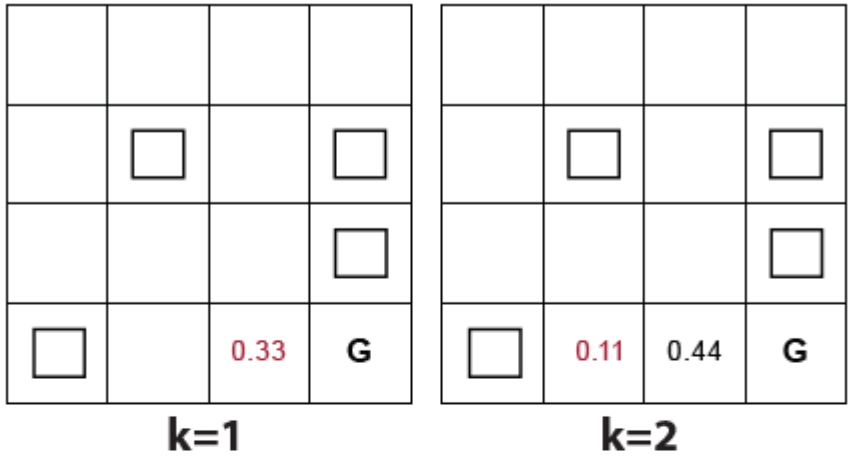
k=1

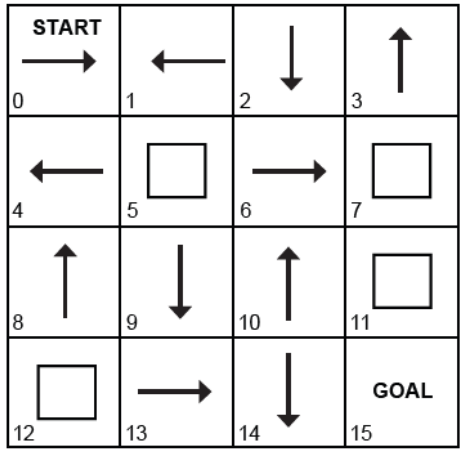
$$\begin{aligned} v_1^\pi(14) = \; & p(s'=15|s=14,a=\downarrow) * [R(14,\downarrow,15) + v_0^\pi(15)] + \\ & p(s'=14|s=14,a=\downarrow) * [R(14,\downarrow,14) + v_0^\pi(14)] + \\ & p(s'=13|s=14,a=\downarrow) * [R(14,\downarrow,13) + v_0^\pi(13)] \end{aligned}$$

$$\begin{aligned} v_1^\pi(14) = \; & 1/3 * [1+0] + \\ & 1/3 * [0+0] + \\ & 1/3 * [0+0] = .33\ldots \end{aligned}$$



Policy Evaluation





$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'r} p(s'|s,a)[r + \gamma v_\pi(s')]$$

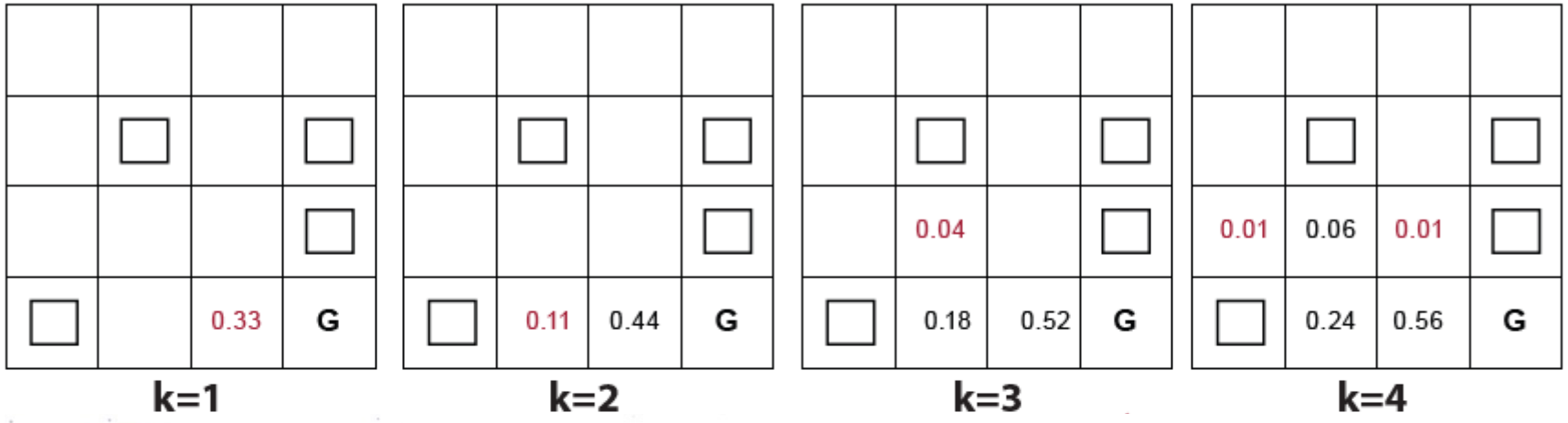
$$\begin{aligned} v_2^\pi(14) = \; & p(s'=15|s=14,a=\downarrow) * [R(14,\downarrow,15) + v_1^\pi(15)] + \\ & p(s'=14|s=14,a=\downarrow) * [R(14,\downarrow,14) + v_1^\pi(14)] + \\ & p(s'=13|s=14,a=\downarrow) * [R(14,\downarrow,13) + v_1^\pi(13)] \end{aligned}$$

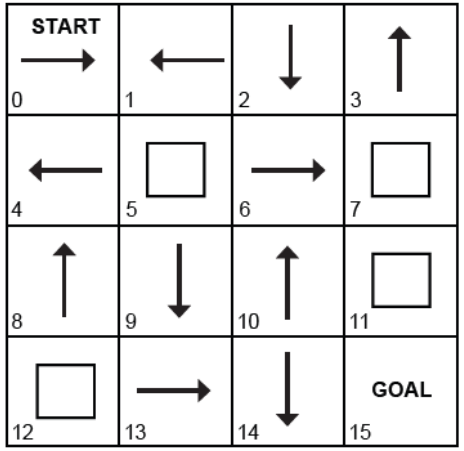
$$\begin{aligned} v_2^\pi(14) = \; & 1/3 * [1+0] + \\ & 1/3 * [0+.33] + \\ & 1/3 * [0+0] = .44\ldots \end{aligned}$$

$$\begin{aligned} v_2^\pi(13) = \; & p(s'=14|s=13,a=\rightarrow) * [R(13,\rightarrow,14) + v_1^\pi(14)] + \\ & p(s'=13|s=13,a=\rightarrow) * [R(13,\rightarrow,13) + v_1^\pi(13)] + \\ & p(s'=9|s=13,a=\rightarrow) * [R(13,\rightarrow,9) + v_1^\pi(9)] \end{aligned}$$

$$\begin{aligned} v_2^\pi(14) = \; & 1/3 * [0+.33] + \\ & 1/3 * [0+0] + \\ & 1/3 * [0+0] = .11\ldots \end{aligned}$$

Policy Evaluation





Policy Evaluation

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
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 $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Policy Improvement

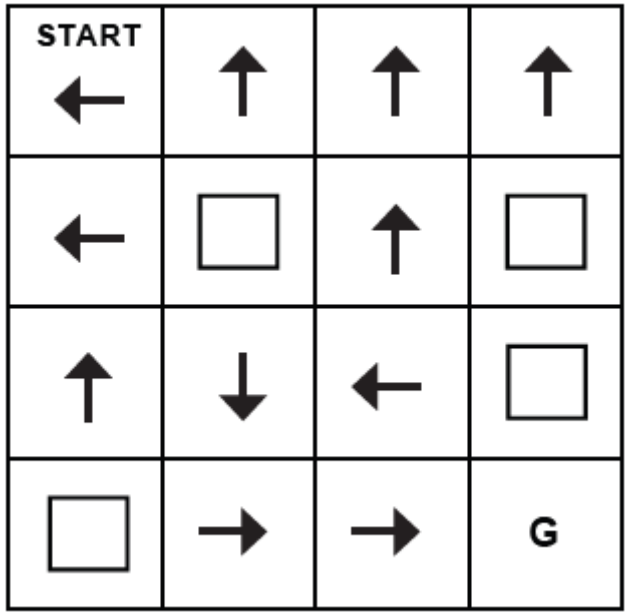
To improve a policy, we use a state-value function and an MDP to get a one-step look-ahead and determine which of the actions lead to the highest value. This is the policy-improvement equation

$$\pi'(s) = \underset{a}{\operatorname{argmax}} \sum_{s'r} p(s'|s,a)[r + \gamma v_\pi(s')]$$

We obtain a new policy π by taking the highest-valued action.

How do we get the highest-valued action? By calculating, for each action, the weighted sum of all rewards and values of all possible next states

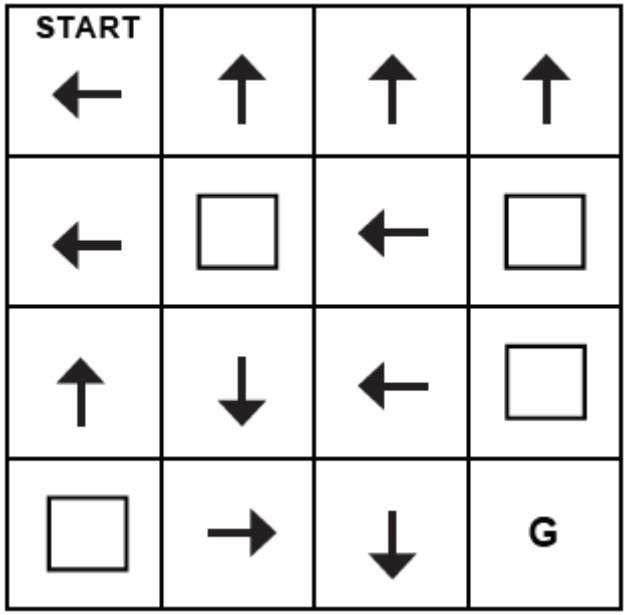
Policy Improvement



Careful Policy

Action-value function of the Careful

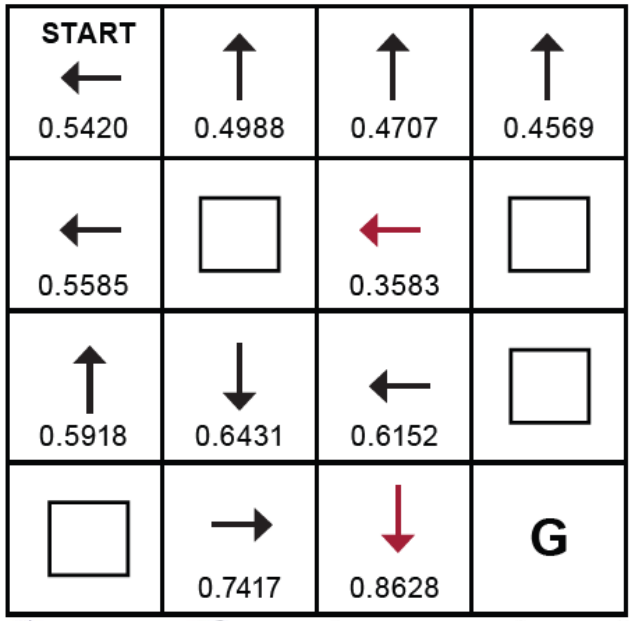
START 0.39 0.41 0.40 0.40	0.38 0.26 0.24 0.25	0.35 0.28 0.27 0.28	0.34 0.23 0.23 0.23
0.27 0.42 0.28 0.29		0.12 0.26 0.26 0.14	
0.45 0.29 0.30 0.31	0.29 0.34 0.34 0.48	0.2 0.43 0.27 0.39	
	0.39 0.35 0.59 0.43	0.67 0.57 0.71 0.76	GOAL



Careful+ Policy

$$\pi'(s) = \underset{a}{\operatorname{argmax}} \sum_{s'r} p(s'|s,a)[r + \gamma v_\pi(s')]$$

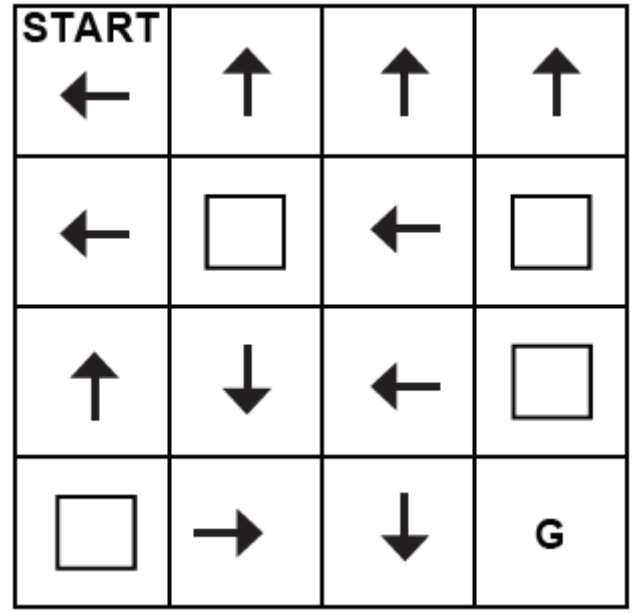
Policy Improvement



Careful+ Policy
State value function

Action-value function of the Careful+

START 0.52 0.54 0.53 0.53	0.50 0.34 0.32 0.33	0.47 0.44 0.42 0.43	0.46 0.31 0.30 0.31
0.36 0.56 0.37 0.38		0.16 0.36 0.36 0.20	
0.59 0.38 0.40 0.41	0.40 0.44 0.45 0.64	0.33 0.62 0.40 0.50	
	0.50 0.46 0.74 0.53	0.78 0.73 0.82 0.86	GOAL



Careful+ = Careful++

$$\pi'(s) = \underset{a}{\operatorname{argmax}} \sum_{s'r} p(s'|s,a)[r + \gamma v_\pi(s')]$$

SNT

Exercise 1

UNIVERSITY OF LUXEMBOURG

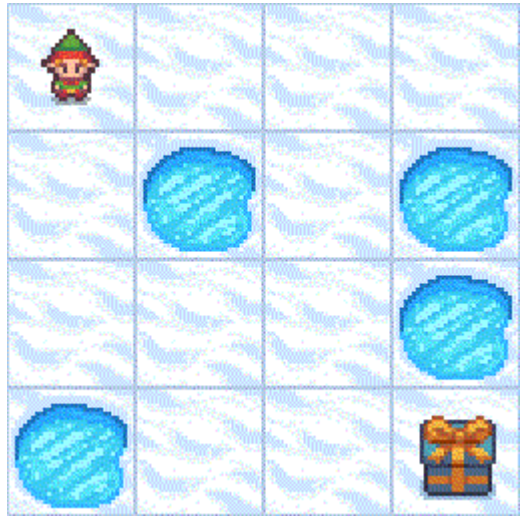
Exercise 1.1

1. Define, at least, four deterministic policies to solve the Frozen Lake environment.
    1. One (or more) policy should be an adversarial policy
2. Implement the Policy evaluation algorithm to determine the State-value function.
3. Evaluate the four different policies.

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
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 $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

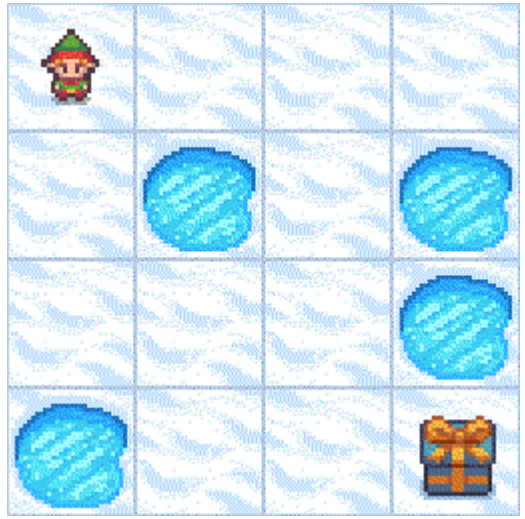


Exercise 1.2

1. Improve every policy and determine the new V function
2. Can an improved policy be improved?
3. If so, can you design an algorithm to keep improving a policy?
4. Does the policy converge?
5. Is the policy optimal?

Policy Improvement:

- For an optimal policy π_*
  - $\pi_*(\mathrm{s}) = \underset{a}{\operatorname{argmax}}\, q_*(s,a)$
- Define a new policy π'
  - $\pi'(\mathrm{s}) = \underset{a}{\operatorname{argmax}}\, q_\pi(s,a)$



$$q_\pi(s,a) = \sum_{s'r} p(s',r|s,a)[r + \gamma v_\pi(s')], \forall s \in S, \forall a \in A(s)$$

Model-Free

SnT

UNIVERSITY OF LUXEMBOURG

But what if we don’t have the full knowledge of the MDP?

The goal is to estimate the value of a policy, that is, to learn how much total reward to expect from a policy. The simplest approach might be to run several episodes with this policy collecting hundreds of trajectories, and then calculate the averages for each state. This method of estimating value functions is called **Monte Carlo (MC) prediction**.

But what if we don’t have the full knowledge of the MDP?

The two main approaches for this are called **Monte Carlo** methods and **Temporal Difference** methods.

Monte Carlo

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

+1

Monte Carlo

Every-visit

~~First visit~~ MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

~~Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$~~:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow$ average$(Returns(S_t))$

+1

Monte Carlo Example

Episode 1: A+3->A+2->B-4->A+4->B-3->Terminate
Episode 2: B-2->A+3->B-3

First-visit

$$V(A) = \frac{1}{2}(2 + 0)$$

$$V(B) = \frac{1}{2}(-3 - 2)$$

Every-visit

$$V(A) = \frac{1}{4}(2 - 1 + 1 + 0)$$

$$V(B) = \frac{1}{4}(-3 - 3 - 2 - 3)$$

+1

Temporal-difference (TD) learning

One of the main drawbacks of MC is the fact that the agent must wait until the end of an episode when it can obtain the actual return $G_{t:T}$ before it can update the state-value function estimate $V_T(S_t)$

Temporal-difference (TD) learning is a combination of learning directly from experience by sampling, but also bootstraps.

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

Vanilla MC Update

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Vanilla TD Update

TD learning is the precursor of methods such as SARSA, Q-learning, double Q-learning, deep Q-networks (DQN), double deep Q-networks (DDQN)…

Q-Learning

One of the early breakthroughs in reinforcement learning was the development of an off-policy TD control algorithm known as Q-learning (Watkins, 1989), defined by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} - \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

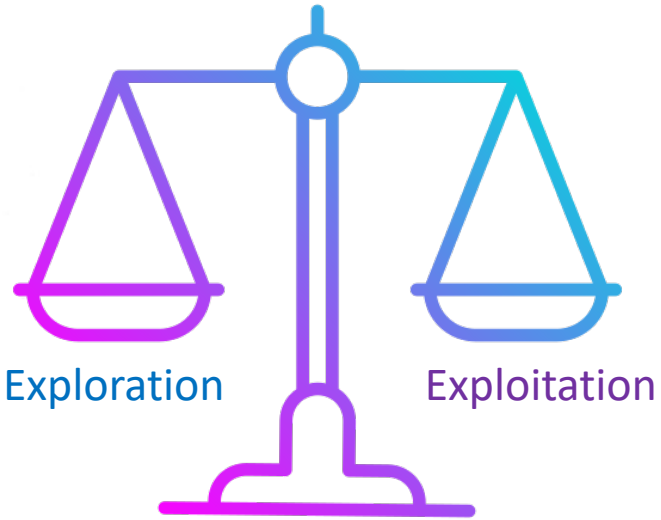
In this case, the learned action-value function, Q, directly approximates $q*$ (the optimal action-value function), **independent of the policy being followed**.

Q-Learning

One of the early breakthroughs in reinforcement learning was the development of an off-policy TD control algorithm known as Q-learning (Watkins, 1989), defined by

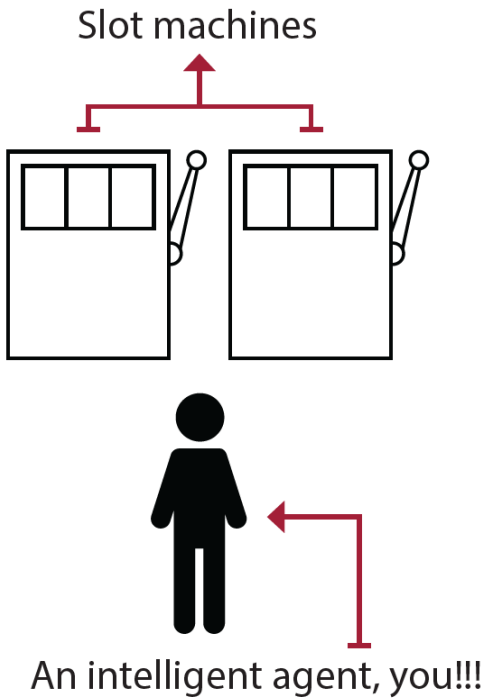
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} - \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

In this case, the learned action-value function, Q, directly approximates $q*$ (the optimal action-value function), **independent of the policy being followed**.



Exploration vs Exploitation

When you go to a casino and play the slot machines, your goal is to find the machine that pays the most, and then stick to that arm.



Q-Learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal

$$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Sarsa

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

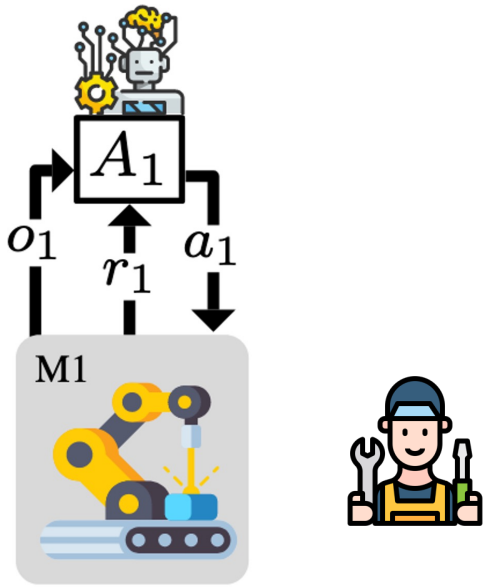
SNT

Exercise 2

UNIVERSITY OF LUXEMBOURG

Exercise 2

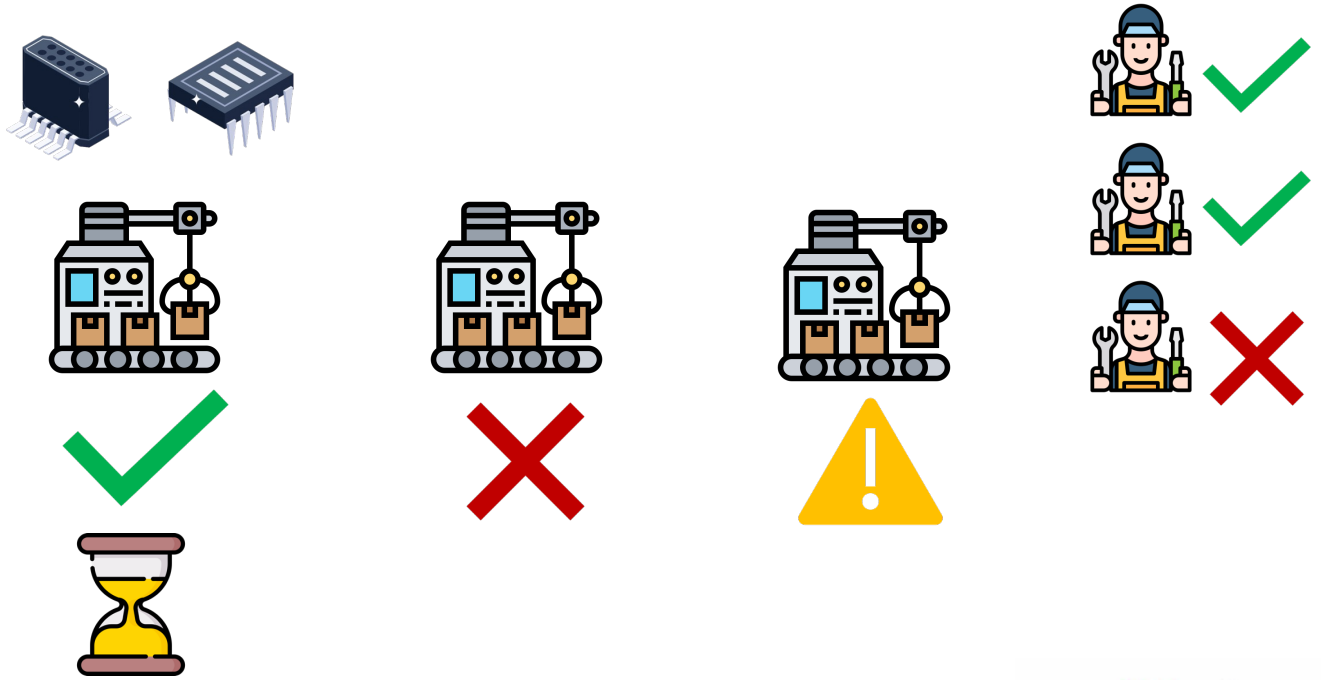
There is an agent who is in charge of maintaining a machine in a manufacturing company. The machine has 2 different components that can fail following a different Weibull distribution.



Exercise 2
Observation

Notation	Description
$\mathbf{o}_m$	Local observation obtained by agent in charge of m
M	Set of machines
T	Set of Technicians
C_m	Set of components of $m \in M$
z_m	State of $m \in M$
g_t	State of $t \in T$
w_c	State of $c \in C_m$
l_c	Lifespan of component $c \in C_m$
e_m	Remaining maintenance time of m

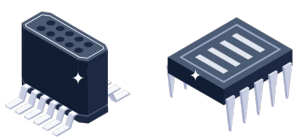
$$\boldsymbol{o}_m = [z_1, \dots, z_{|M|}, g_1, \dots, g_{|T|}, w_1, \dots, w_{|C_m|}, l_1, \dots, l_{|C_m|}, e_m]$$

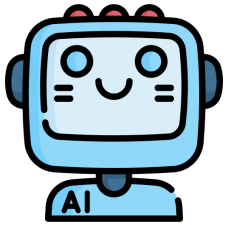


Exercise 2
Action

Notation	Description
$\boldsymbol{a}_m$	Action taken by the agent in charge of m
T	Set of Technicians
C_m	Set of components of $m \in M$
$\boldsymbol{d}$	No action

$$\boldsymbol{a}_m \in T \times C_m \cup \{\boldsymbol{d}\}$$



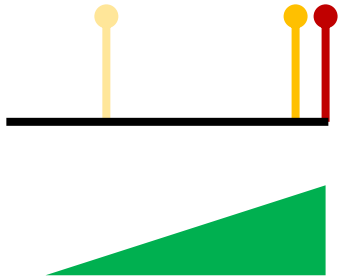


Exercise 2
Reward

Notation	Description
r_m	Reward obtained by agent in charge of m
$\mathbf{o}_m$	Local observation obtained by agent in charge of m
$\boldsymbol{a}_m$	Action taken by the agent in charge of m
f_c	Failure time of $c \in C_m$
l_c	Lifespan of component $c \in C_m$
η	Reward penalty due to breakdown prediction

$$r_m(\mathbf{o}_m, \boldsymbol{a}_m, \mathbf{o'}_m) = \begin{cases} 1 & if \ \mathbf{o'}_m = working \\ 1-\eta & if \ \mathbf{o'}_m = maintenance \\ 0 & if \ \mathbf{o'}_m = breakdown \\ -2 & if \ \boldsymbol{a} = invalid \end{cases}$$

$$\eta = \begin{cases} \dfrac{f_c - l_c}{f_c + l_c} & if \ f_c > l_c \\ 1 & otherwise \end{cases}$$

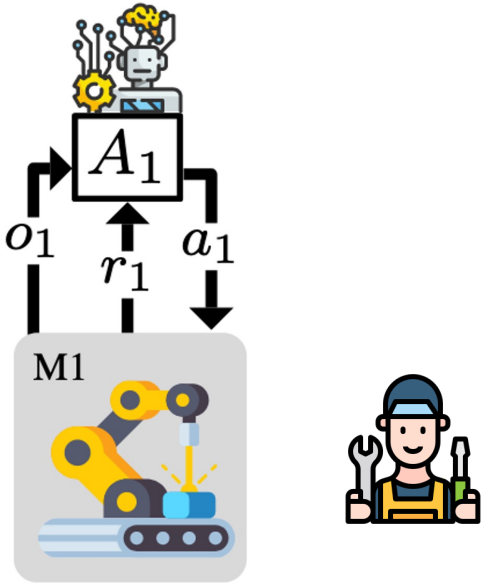


Closer the maintenance to the failure

Greater the reward

Exercise 2

1. Train the agent using Q-Learning to learn the best maintenance policy.
2. Improve the tradeoff exploration-exploitation using an epsilon decay
    1. (Plus, design a different exploration strategy)
3. Define which hyperparameters fit best the problem



SNT

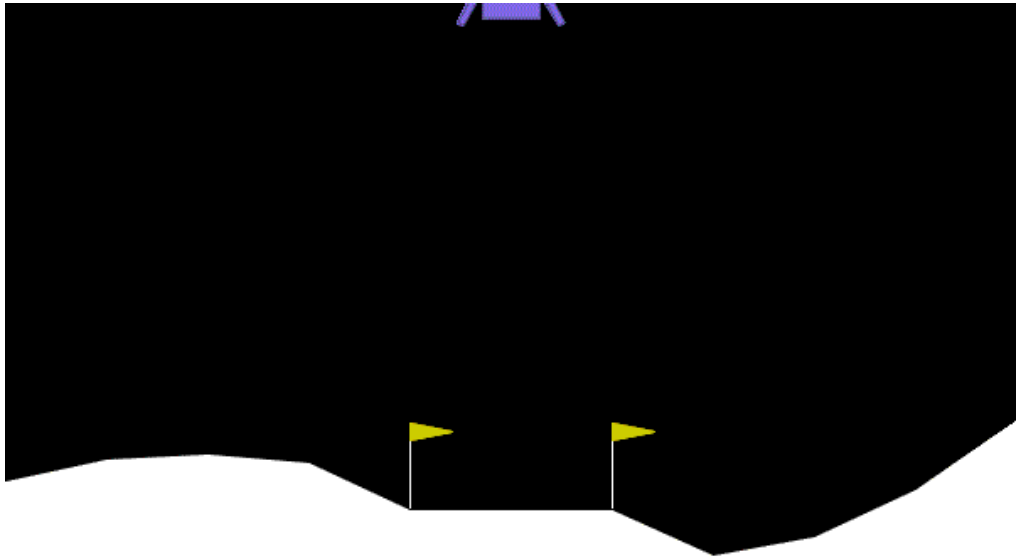
Exercise 3

UNIVERSITY OF LUXEMBOURG

Deep Q Learning

https://www.gymlibrary.dev/environments/box2d/lunar_lander/

Action Space	Discrete(4)
Observation Shape	(8,)
Observation High	[1.5 1.5 5. 5. 3.14 5. 1. 1.]
Observation Low	[-1.5 -1.5 -5. -5. -3.14 -5. -0. -0.]
Import	`gym.make("LunarLander-v2")`



Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Q-function optimization without a target network

(1) At first everything will look normal. We just chase the target.

(2) But the target will move as our Q-function improves.

(3) Then, things go bad.

(4) And the moving targets could create divergence.

Q-function approximation with a target network

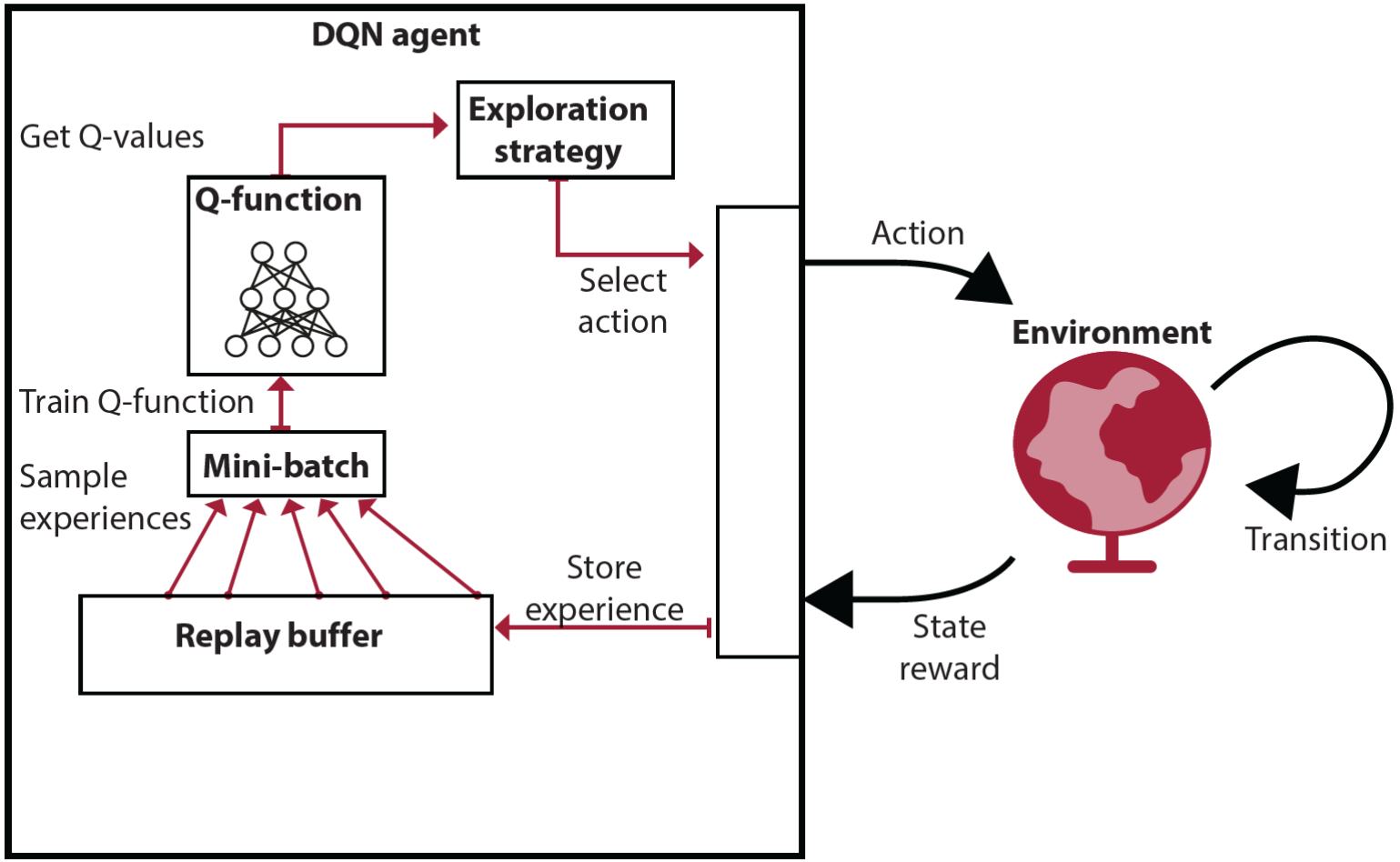
(1) Suppose we freeze the target for a few steps.

(2) That way, the optimizer can make stable progress toward it.

(3) We eventually update the target, and repeat.

(4) This allows the algorithm to make stable progress.

DQN with a replay buffer



Notebook

References

Sutton, R. S., & Barto, A. G. (2012). An Reinforcement Learning: Introduction. Mit Press.

Morales, M. (2020). Grokking deep reinforcement learning. Manning Publications.

Winder, P. (2020). Reinforcement learning: Industrial applications of intelligent agents. O'Reilly Media.

SnT

Marcelo Luis Ruiz Rodríguez
marcelo.ruiz@uni.lu

uni.lu
UNIVERSITY OF LUXEMBOURG