

Artificial Neural Networks

Course-8

Gavrilit Dragos

rev 1

π



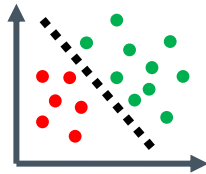

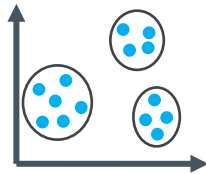


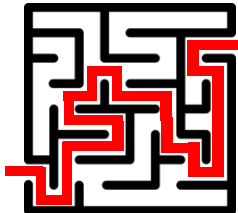
AGENDA FOR TODAY

- › Introduction to Reinforcement Learning

Introduction to Reinforcement Learning

Introduction to Reinforcement Learning

We know that machine learning can be classified into two big categories:

Type of ML	Input & Output		Expected result	Example
Supervised Learning	 Dataset	 Labels	Classification: Given an input, classify it in different categories/assign it a label	
Un-supervised Learning	 Dataset		Clustering: Given an input, assign it into a group based on similarities.	
Reinforcement Learning	 Envinmt.	 Rewards	Adapt: to an environment and a rewards (maximize reward)	

π

Introduction to Reinforcement Learning

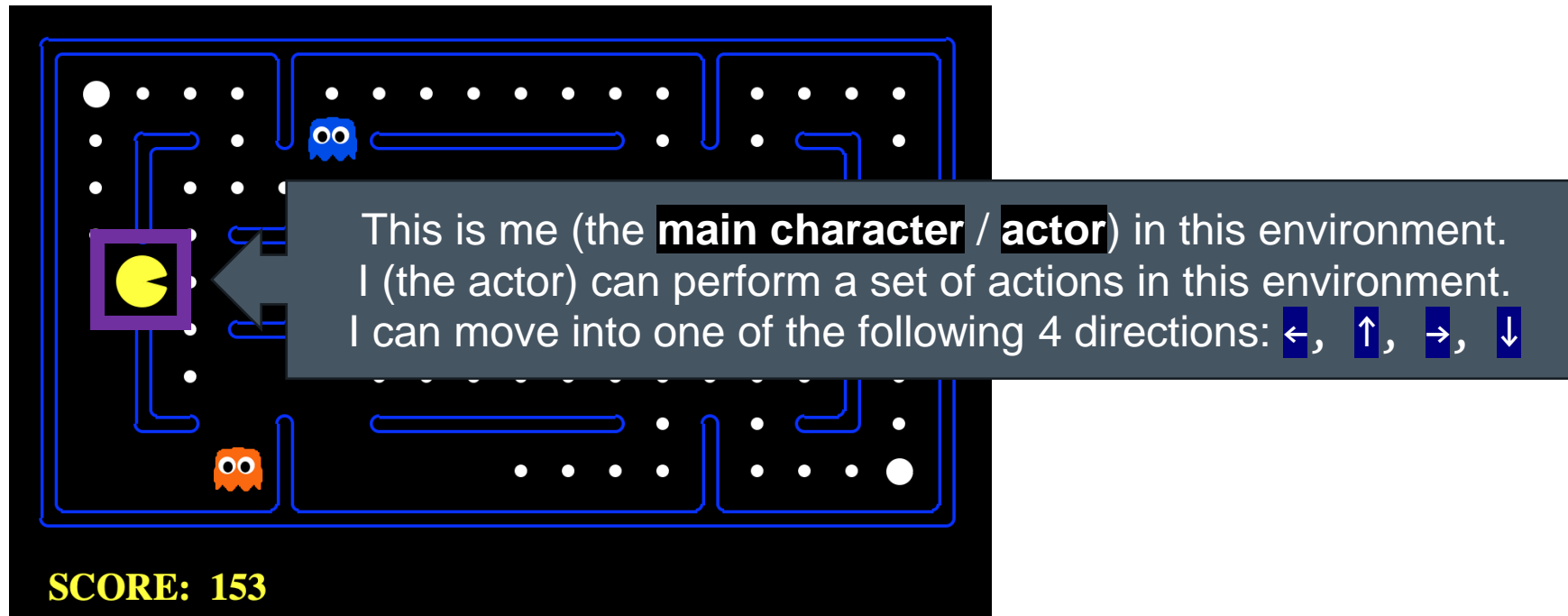
But, neural networks rely on a dataset and labels to train.

So, how do we get from:

		to			?
Envinmt.	Rewards		Dataset	Labels	

Introduction to Reinforcement Learning

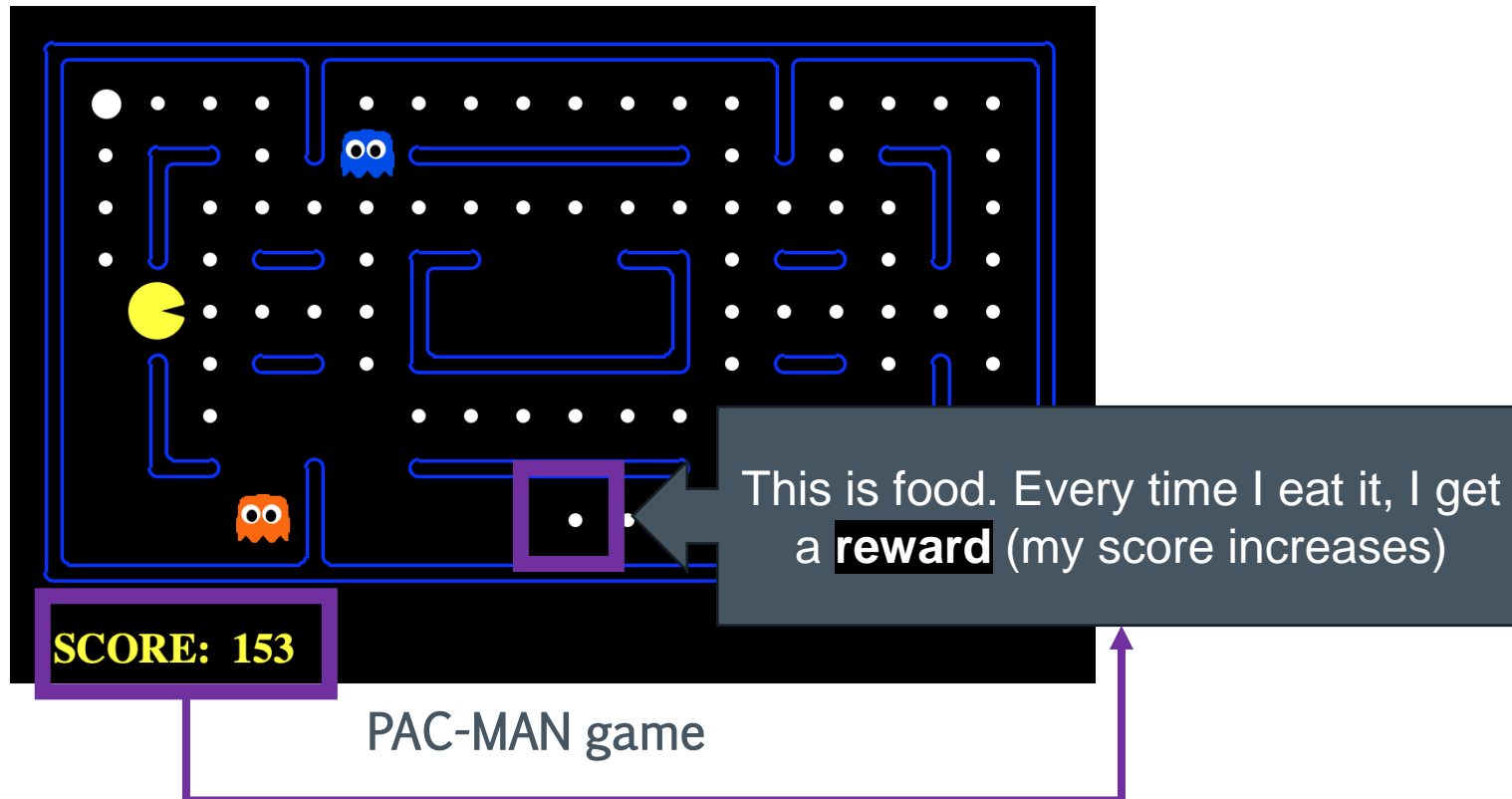
To answer the previous question, we must first understand what an **environment** is in this context.



PAC-MAN game

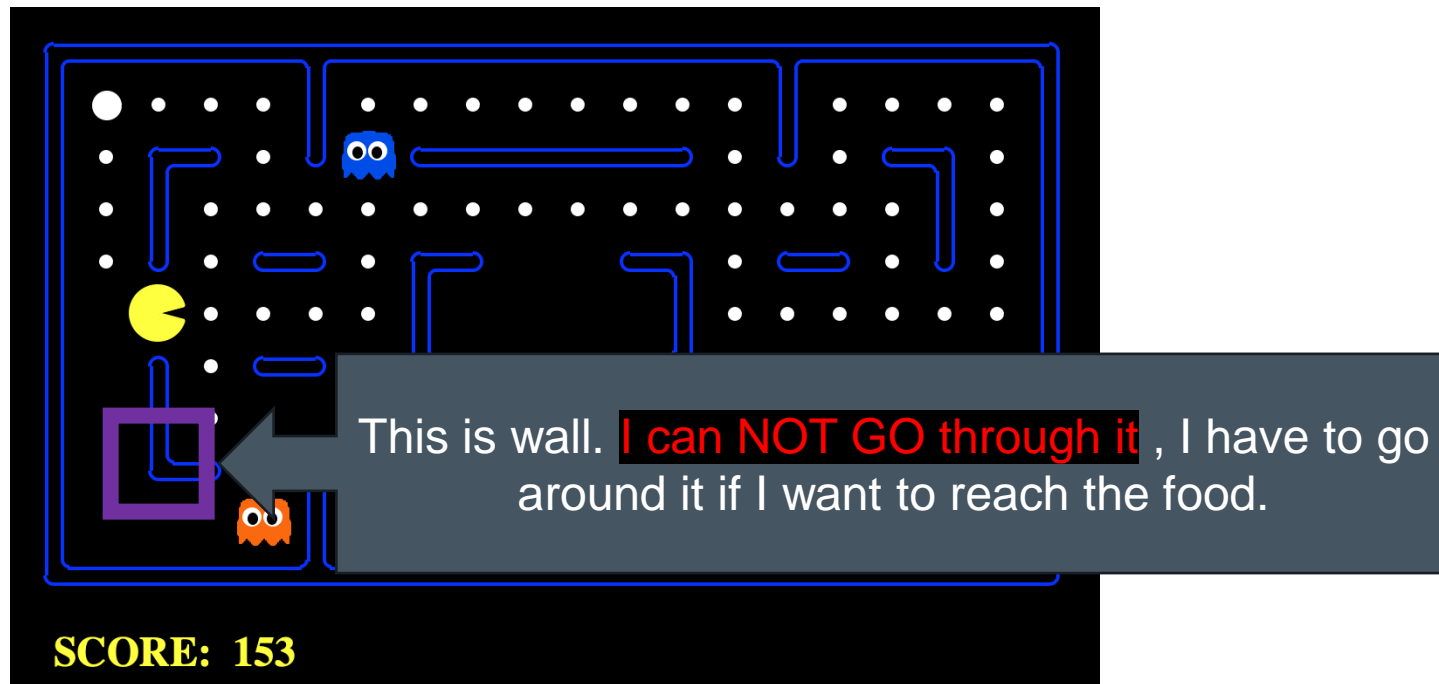
Introduction to Reinforcement Learning

To answer the previous question, we must first understand what an environment is in this context.



Introduction to Reinforcement Learning

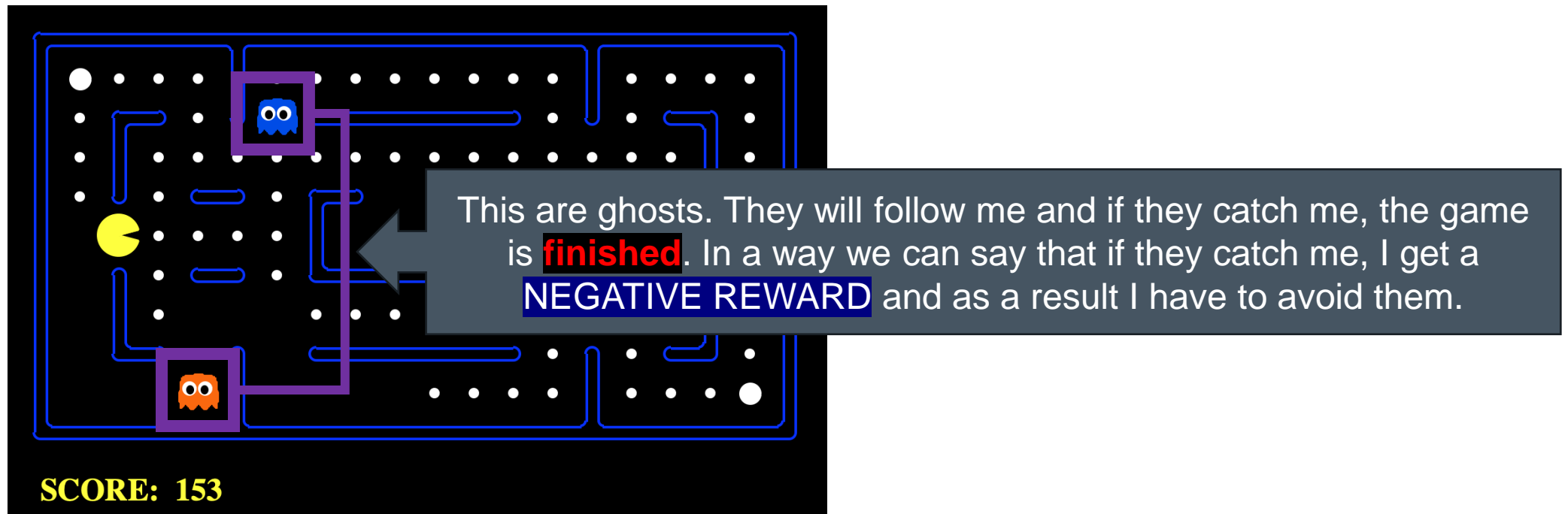
To answer the previous question, we must first understand what an environment is in this context.



PAC-MAN game

Introduction to Reinforcement Learning

To answer the previous question, we must first understand what an environment is in this context.



PAC-MAN game

Introduction to Reinforcement Learning

To answer the previous question, we must first understand what an **environment** is in this context.

We can say that an environment is a set of rules. For example, in case of **PacMan** game:

- Food give you rewards
- You need to avoid ghosts
- You can no go through walls

Introduction to Reinforcement Learning

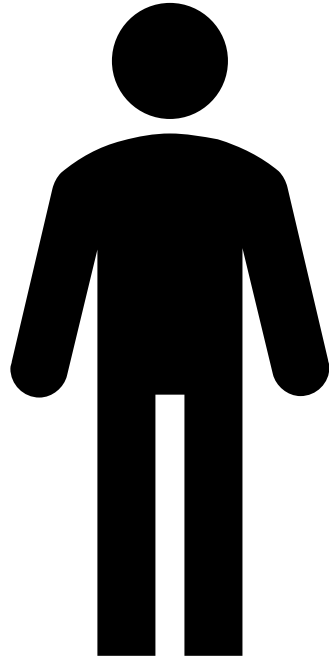
But we (the main character/the actor) **don't know the rules** (or at least not all of them).

What we can perceive is ***rewards*** (when we get them). A rewards can be **positive** or **negative** (think in the case where a ghost catches us in the PacMan game).

And we also have a set of **sensors** that allows us to perceive a part (or the entire) environment.

Introduction to Reinforcement Learning

What is a sensor ?



Eyes

Vision sensor (allows us to see our surroundings)



Ears

Audio sensor (allows us to listen for sounds)



Tongue

Taste sensor (allows us taste our food)



Nose

Smell sensor (allows us evaluate air quality)



Skin

Tactile sensor (allows us to evaluate surroundings texture)

Introduction to Reinforcement Learning

Sensors are one of our adaptation that help us survive our environment.

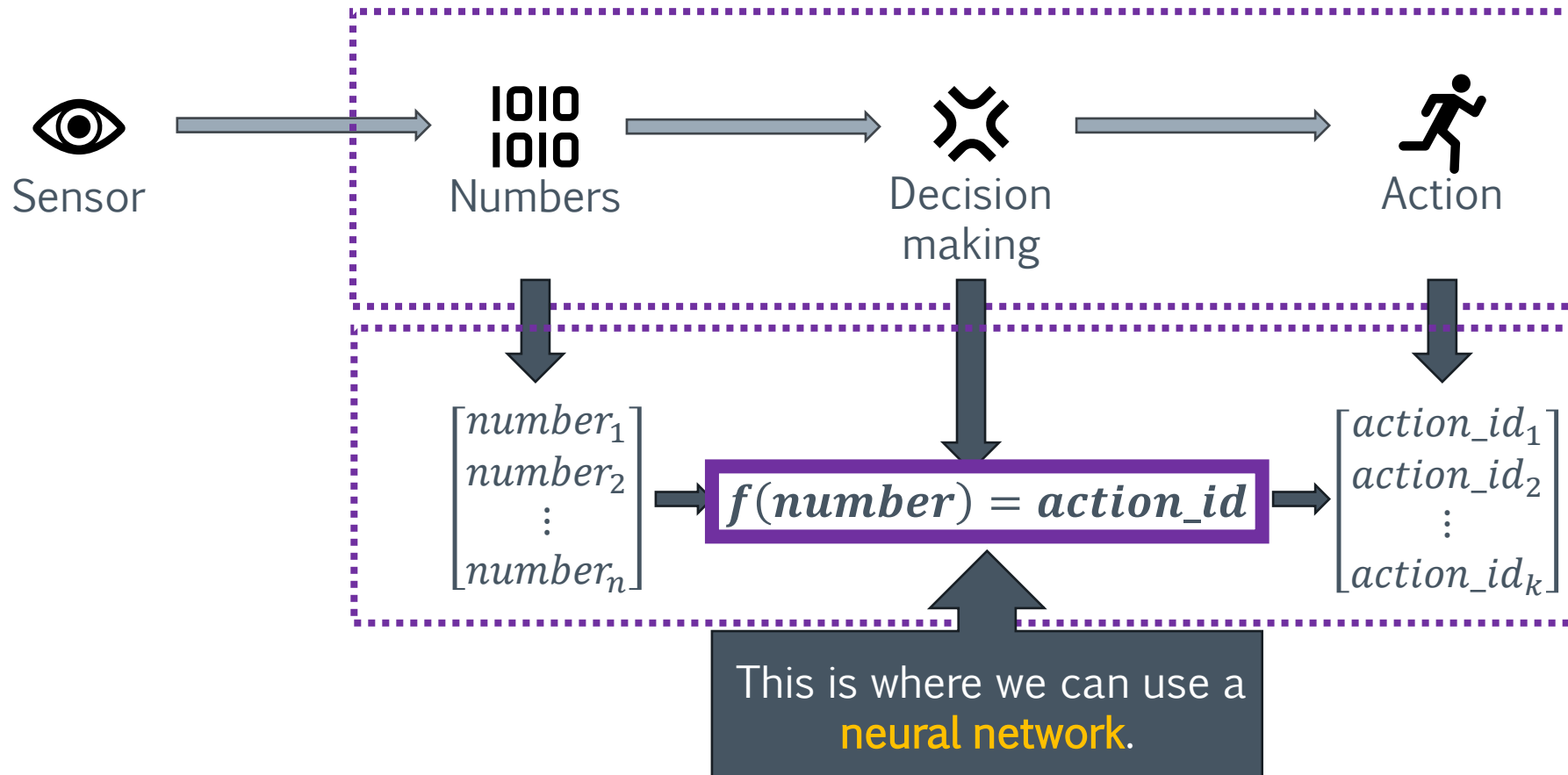
- **Visual, audio** and **tactile** sensors → help us avoid danger (so in a way help us avoid a **negative reward**).
- **Smell** sensors → help us identify possible food sources (so in a way they help us find a **positive reward**)
- Taste sensor → has can decide if a food source is good or not (so in a way is capable of both finding **positive rewards** and avoiding **negative ones**)

Another key aspect to understand here is that the better the sensor, the more likely it is to survive (get a higher reward).

π

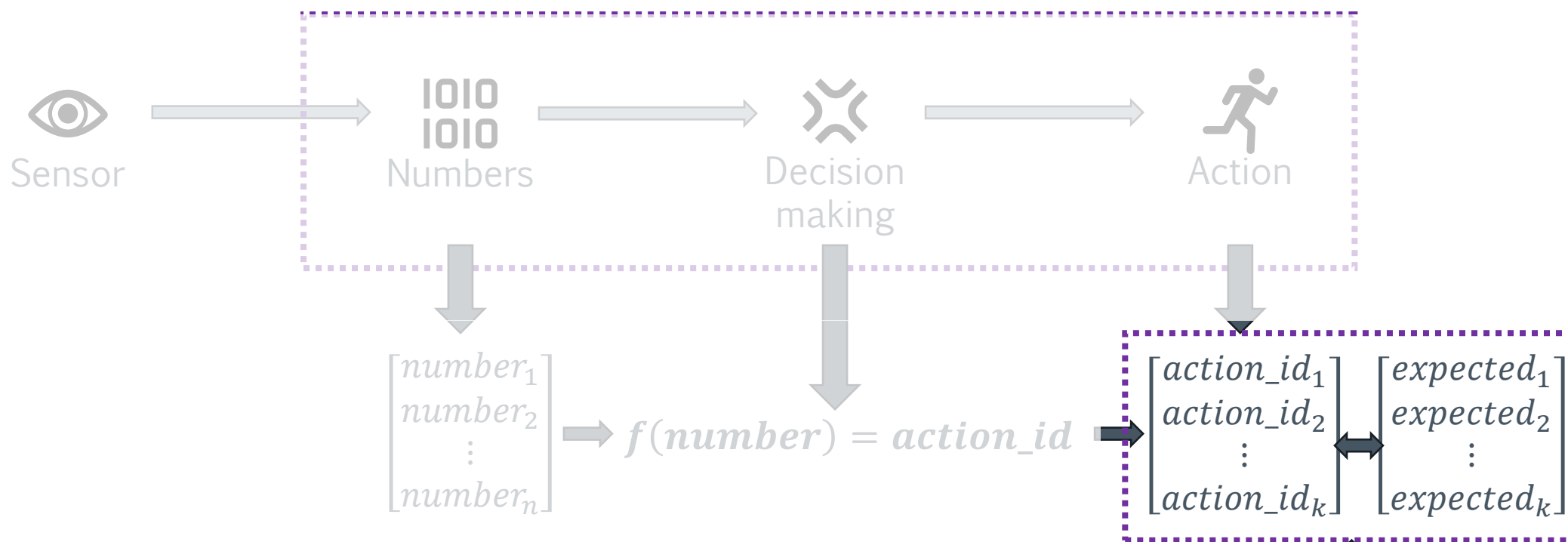
Introduction to Reinforcement Learning

So ... we can look into this approach in a different way:



Introduction to Reinforcement Learning

So ... we can look into this approach in a different way:



For the next part of the course, we will discuss this aspect
(how can we know/find the expected output)

The only question is what
is the **expected output** ?

Q-Learning

π

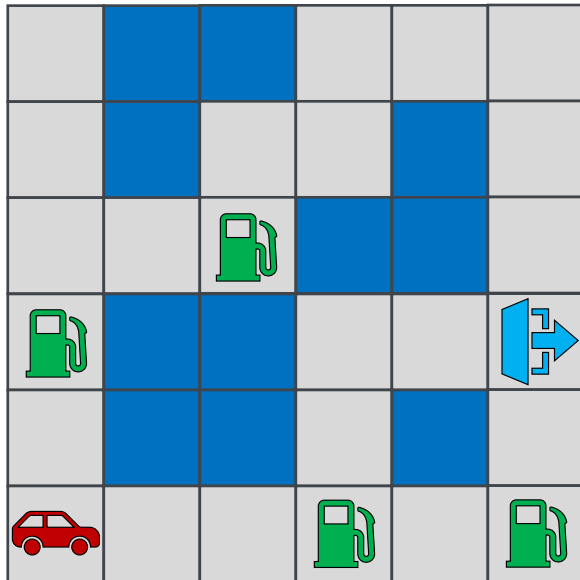
Q-Learning

Before we start talking about Q-Learning, let's imagine the following problem:

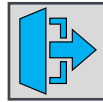
- You are a car and need to reach a destination
- You can move into 4 directions (\leftarrow \uparrow \rightarrow \downarrow)
- You can move you consume fuel (1 unit of fuel per move)
- You start with 3 units of fuel
- On your road to the destination, you can encounter fuel stations that will increase your fuel by 3
- Once you get fuel from a fuel station you can not charge there anymore (you can only get fuel from a fuel station once)
- Your overall goal is to reach the destination with a maximum amount of fuel available.

Q-Learning

Before we start talking about Q-Learning, let's imagine the following problem:



The **gas station** (reaching this point will increase your fuel by 5)



The **exit point** (this is where we need to arrive – the destination point)



Road (you can drive your car). Every time you move from one square of road to another you lose 1 unit of fuel.



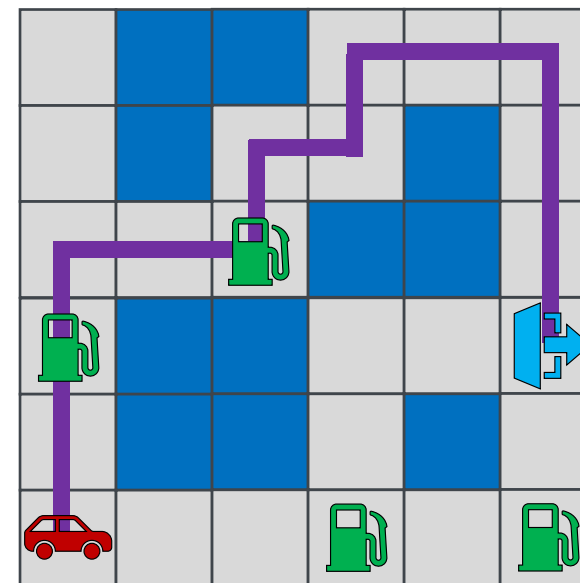
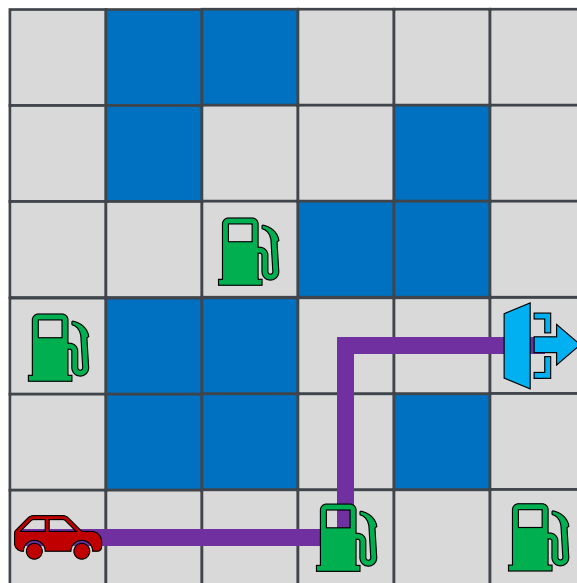
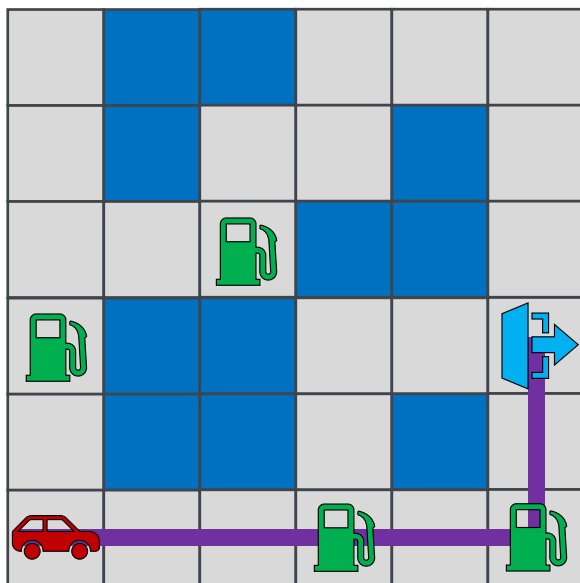
Water
if you end up here, you lose the game.



This is you (the actor / the car) that has to move within the current environment.

Q-Learning

What are the possible solutions to this problem ?

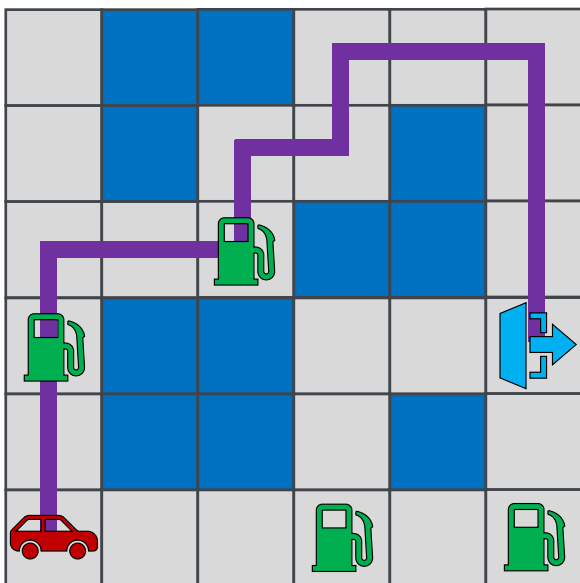


OBS: All of these solutions actually manage to reach the destination

π

Q-Learning

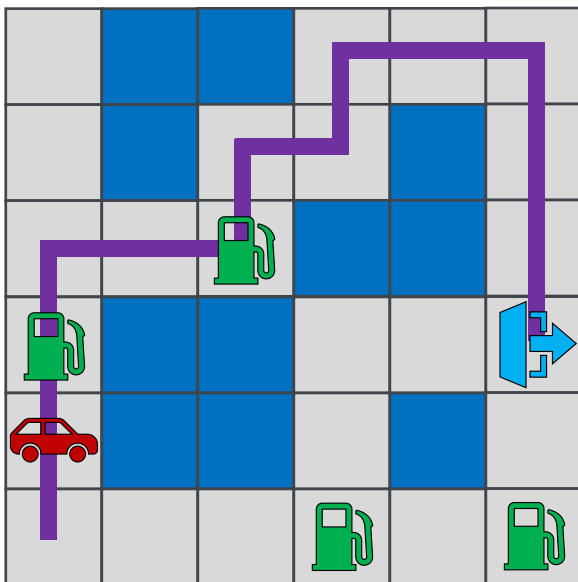
Let's analyze one of these solutions to see the result:



Step	Next Action	Fuel	Observation
1	Move ↑	3	-1 fuel for moving

Q-Learning

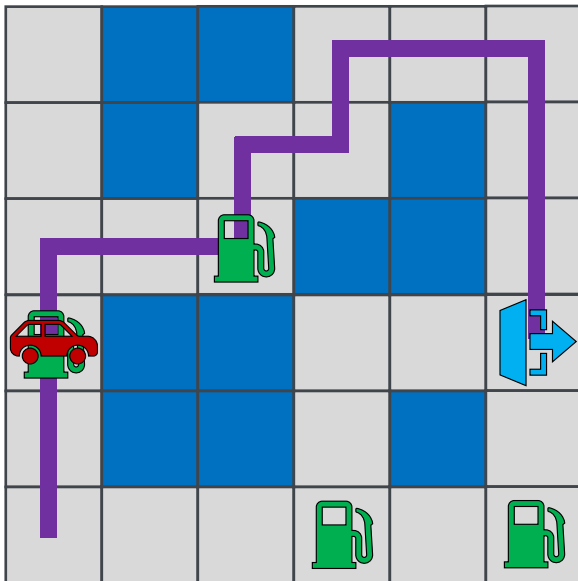
Let's analyze one of these solutions to see the result:



Step	Next Action	Fuel	Observation
1	Move \uparrow	3	-1 fuel for moving
2	Move \uparrow	$3-1=2$	-1 fuel for moving

Q-Learning

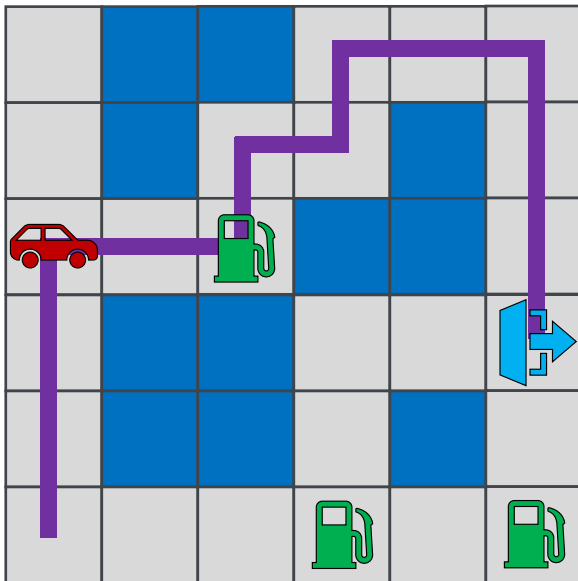
Let's analyze one of these solutions to see the result:



Step	Next Action	Fuel	Observation
1	Move \uparrow	3	-1 fuel for moving
2	Move \uparrow	$3-1=2$	-1 fuel for moving
3	Move \uparrow	$2-1+5=6$	-1 fuel for moving, reach a fuel station (+5)

Q-Learning

Let's analyze one of these solutions to see the result:

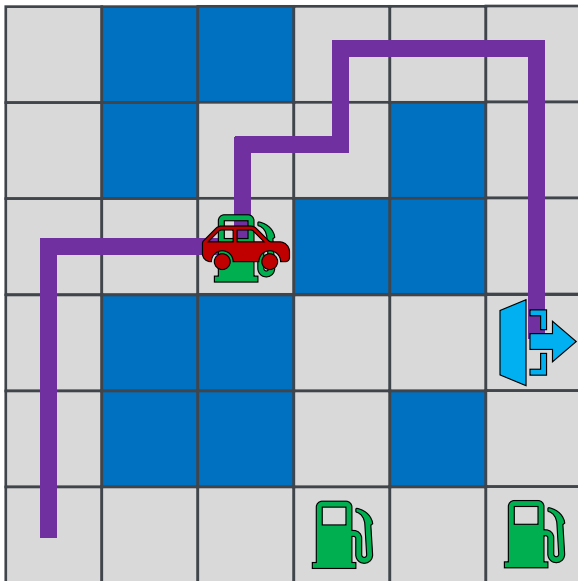


Step	Next Action	Fuel	Observation
1	Move \uparrow	3	-1 fuel for moving
2	Move \uparrow	$3-1=2$	-1 fuel for moving
3	Move \uparrow	$2-1+5=6$	-1 fuel for moving, reach a fuel station (+5)
4	Move \rightarrow	$6-1=5$	-1 fuel for moving

Notice that the fuel station from the previous step is gone (meaning that I can use it anymore).

Q-Learning

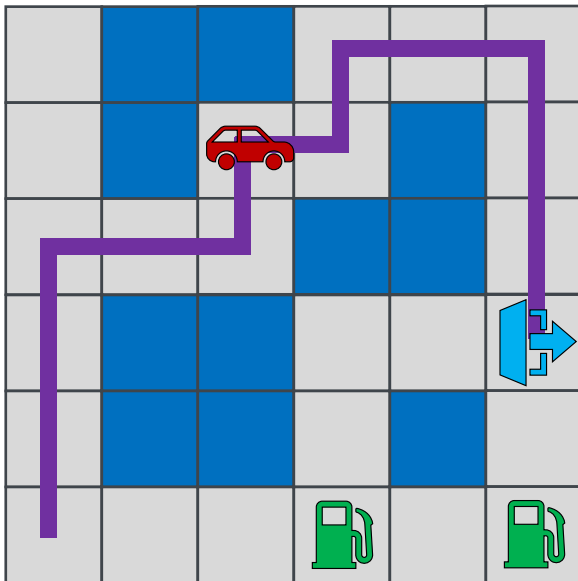
Let's analyze one of these solutions to see the result:



Step	Next Action	Fuel	Observation
1	Move \uparrow	3	-1 fuel for moving
2	Move \uparrow	$3-1=2$	-1 fuel for moving
3	Move \uparrow	$2-1+5=6$	-1 fuel for moving, reach a fuel station (+5)
4	Move \rightarrow	$6-1=5$	-1 fuel for moving
5	Move \rightarrow	$5-1=4$	-1 fuel for moving
6	Move \uparrow	$4-1+5=8$	-1 fuel for moving, reach a fuel station (+5)

Q-Learning

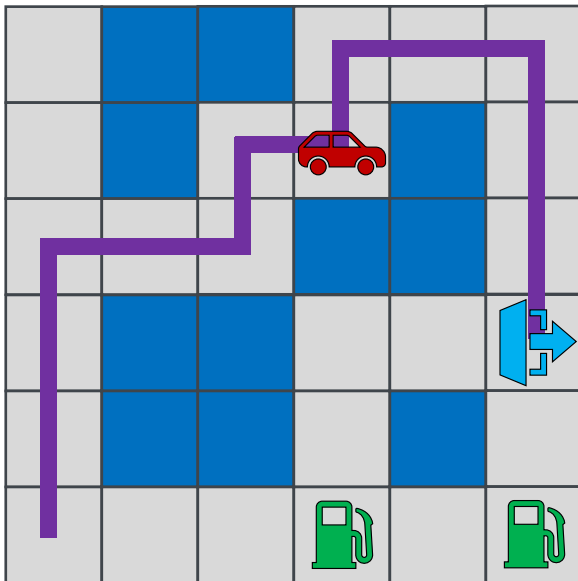
Let's analyze one of these solutions to see the result:



Step	Next Action	Fuel	Observation
1	Move \uparrow	3	-1 fuel for moving
2	Move \uparrow	$3-1=2$	-1 fuel for moving
3	Move \uparrow	$2-1+5=6$	-1 fuel for moving, reach a fuel station (+5)
4	Move \rightarrow	$6-1=5$	-1 fuel for moving
5	Move \rightarrow	$5-1=4$	-1 fuel for moving
6	Move \uparrow	$4-1+5=8$	-1 fuel for moving, reach a fuel station (+5)
7	Move \rightarrow	$8-1=7$	-1 fuel for moving

Q-Learning

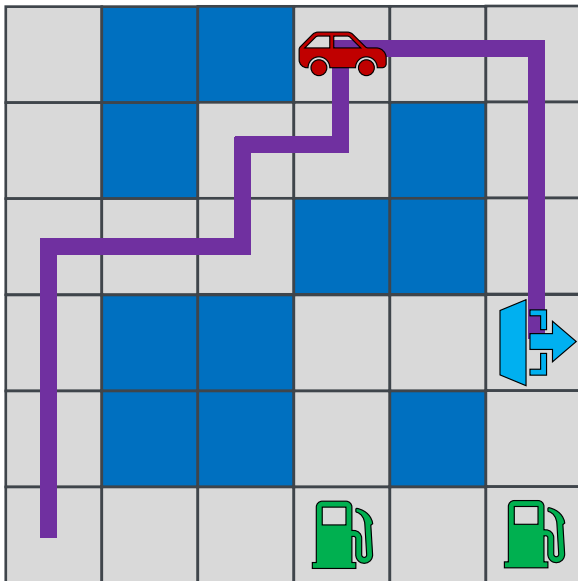
Let's analyze one of these solutions to see the result:



Step	Next Action	Fuel	Observation
1	Move \uparrow	3	-1 fuel for moving
2	Move \uparrow	$3-1=2$	-1 fuel for moving
3	Move \uparrow	$2-1+5=6$	-1 fuel for moving, reach a fuel station (+5)
4	Move \rightarrow	$6-1=5$	-1 fuel for moving
5	Move \rightarrow	$5-1=4$	-1 fuel for moving
6	Move \uparrow	$4-1+5=8$	-1 fuel for moving, reach a fuel station (+5)
7	Move \rightarrow	$8-1=7$	-1 fuel for moving
8	Move \uparrow	$7-1=6$	-1 fuel for moving

Q-Learning

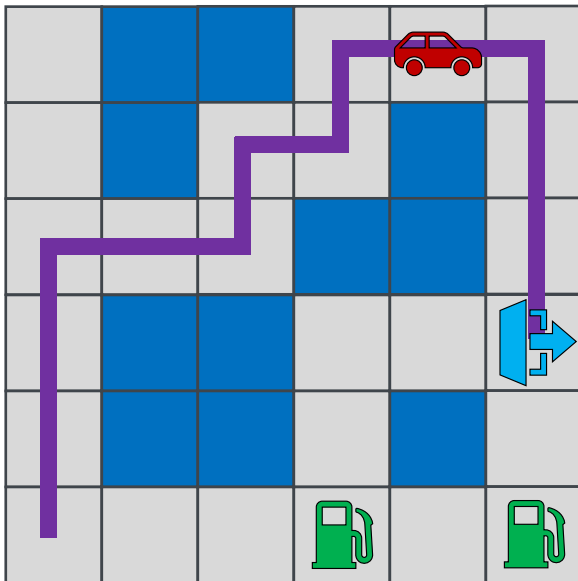
Let's analyze one of these solutions to see the result:



Step	Next Action	Fuel	Observation
1	Move ↑	3	-1 fuel for moving
2	Move ↑	$3-1=2$	-1 fuel for moving
3	Move ↑	$2-1+5=6$	-1 fuel for moving, reach a fuel station (+5)
4	Move →	$6-1=5$	-1 fuel for moving
5	Move →	$5-1=4$	-1 fuel for moving
6	Move ↑	$4-1+5=8$	-1 fuel for moving, reach a fuel station (+5)
7	Move →	$8-1=7$	-1 fuel for moving
8	Move ↑	$7-1=6$	-1 fuel for moving
9	Move →	$6-1=5$	-1 fuel for moving

Q-Learning

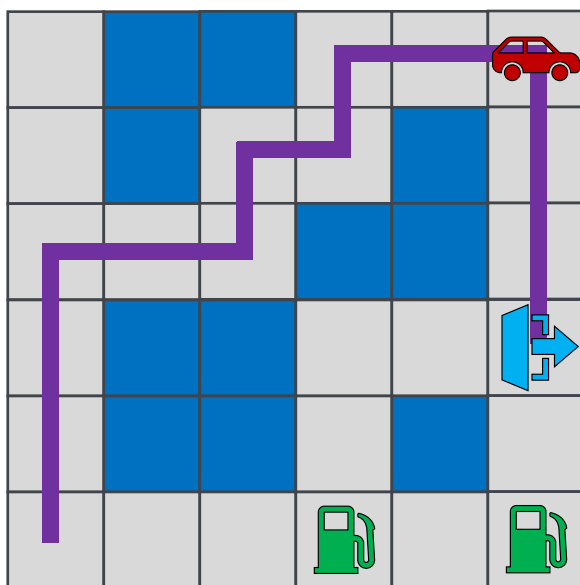
Let's analyze one of these solutions to see the result:



Step	Next Action	Fuel	Observation
1	Move \uparrow	3	-1 fuel for moving
2	Move \uparrow	$3-1=2$	-1 fuel for moving
3	Move \uparrow	$2-1+5=6$	-1 fuel for moving, reach a fuel station (+5)
4	Move \rightarrow	$6-1=5$	-1 fuel for moving
5	Move \rightarrow	$5-1=4$	-1 fuel for moving
6	Move \uparrow	$4-1+5=8$	-1 fuel for moving, reach a fuel station (+5)
7	Move \rightarrow	$8-1=7$	-1 fuel for moving
8	Move \uparrow	$7-1=6$	-1 fuel for moving
9	Move \rightarrow	$6-1=5$	-1 fuel for moving
10	Move \rightarrow	$5-1=4$	-1 fuel for moving

Q-Learning

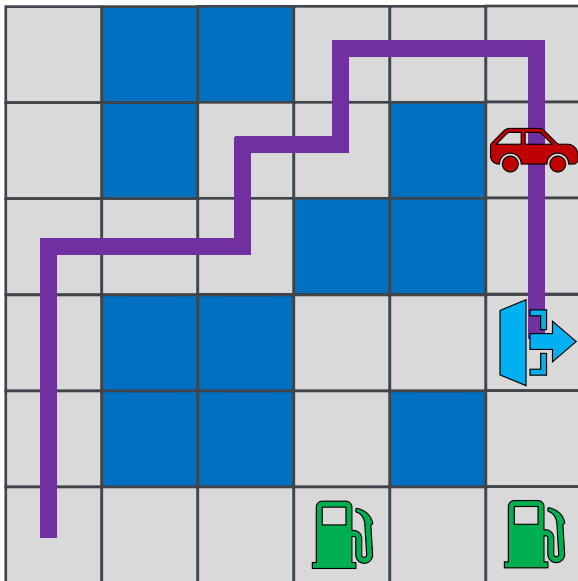
Let's analyze one of these solutions to see the result:



Step	Next Action	Fuel	Observation
1	Move ↑	3	-1 fuel for moving
2	Move ↑	$3-1=2$	-1 fuel for moving
3	Move ↑	$2-1+5=6$	-1 fuel for moving, reach a fuel station (+5)
4	Move →	$6-1=5$	-1 fuel for moving
5	Move →	$5-1=4$	-1 fuel for moving
6	Move ↑	$4-1+5=8$	-1 fuel for moving, reach a fuel station (+5)
7	Move →	$8-1=7$	-1 fuel for moving
8	Move ↑	$7-1=6$	-1 fuel for moving
9	Move →	$6-1=5$	-1 fuel for moving
10	Move →	$5-1=4$	-1 fuel for moving
11	Move ↓	$4-1=3$	-1 fuel for moving

Q-Learning

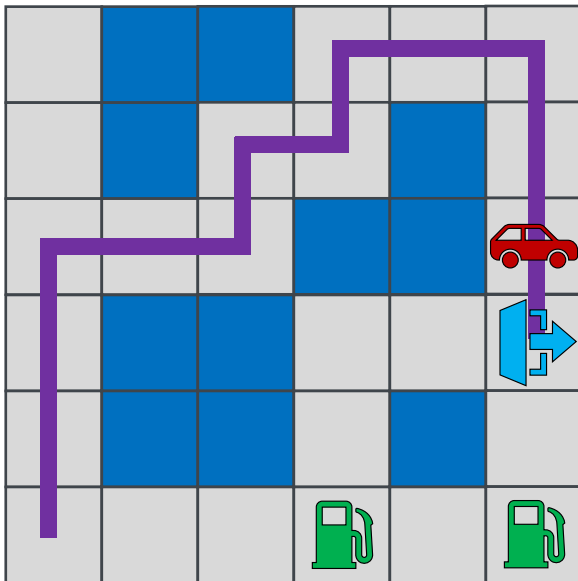
Let's analyze one of these solutions to see the result:



Step	Next Action	Fuel	Observation
1	Move \uparrow	3	-1 fuel for moving
2	Move \uparrow	$3-1=2$	-1 fuel for moving
3	Move \uparrow	$2-1+5=6$	-1 fuel for moving, reach a fuel station (+5)
4	Move \rightarrow	$6-1=5$	-1 fuel for moving
5	Move \rightarrow	$5-1=4$	-1 fuel for moving
6	Move \uparrow	$4-1+5=8$	-1 fuel for moving, reach a fuel station (+5)
7	Move \rightarrow	$8-1=7$	-1 fuel for moving
8	Move \uparrow	$7-1=6$	-1 fuel for moving
9	Move \rightarrow	$6-1=5$	-1 fuel for moving
10	Move \rightarrow	$5-1=4$	-1 fuel for moving
11	Move \downarrow	$4-1=3$	-1 fuel for moving
12	Move \downarrow	$3-1=2$	-1 fuel for moving

Q-Learning

Let's analyze one of these solutions to see the result:

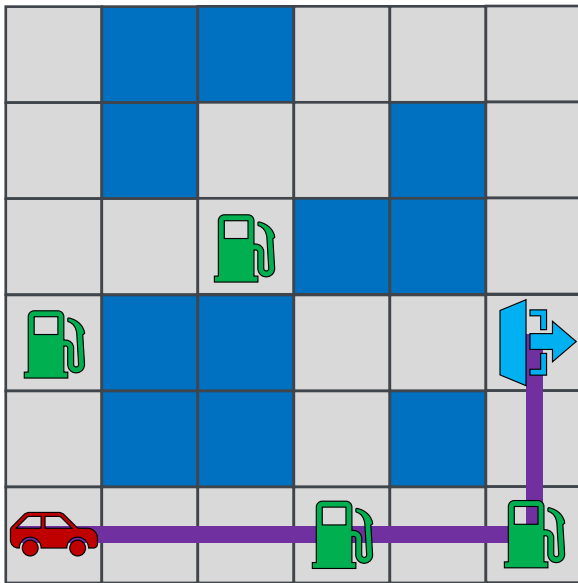


Step	Next Action	Fuel	Observation
1	Move ↑	3	-1 fuel for moving
2	Move ↑	$3-1=2$	-1 fuel for moving
3	Move ↑	$2-1+5=6$	-1 fuel for moving, reach a fuel station (+5)
4	Move →	$6-1=5$	-1 fuel for moving
5	Move →	$5-1=4$	-1 fuel for moving
6	Move ↑	$4-1+5=8$	-1 fuel for moving, reach a fuel station (+5)
7	Move →	$8-1=7$	-1 fuel for moving
8	Move ↑	$7-1=6$	-1 fuel for moving
9	Move →	$6-1=5$	-1 fuel for moving
10	Move →	$5-1=4$	-1 fuel for moving
11	Move ↓	$4-1=3$	-1 fuel for moving
12	Move ↓	$3-1=2$	-1 fuel for moving
13	Move ↓	$2-1=1$	-1 fuel for moving

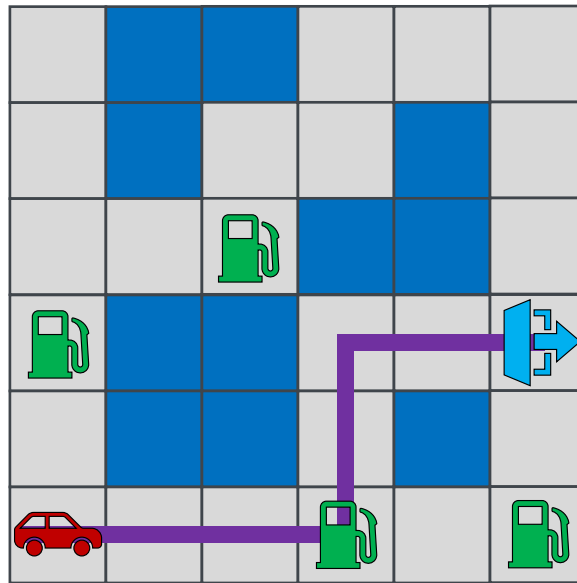
π

Q-Learning

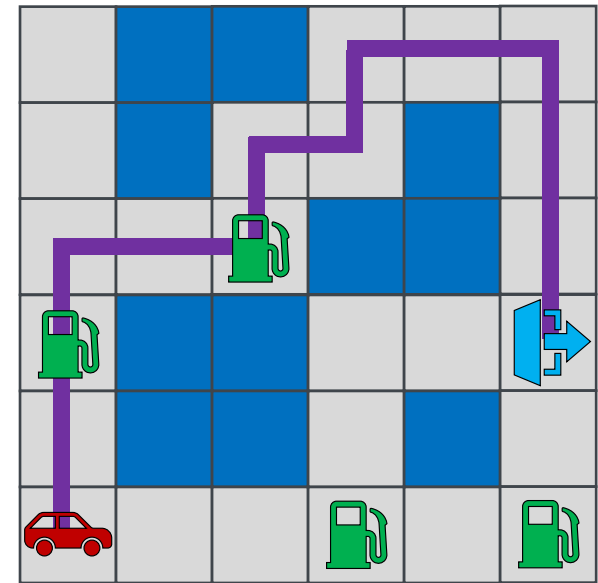
With this in mind, we can compute the fuel consumption for all of the possible scenarios:



6 fuel units left



1 fuel unit left



0 fuel unit left

Best solution

Q-Learning

Another way we can look into things is to consider that the car moves from one state to another state, based on the selected action.

19			12	11	10
18		14	13		9
17	16	15			8
20			23	24	7
21			22		6
0	1	2	3	4	5

3 → 4

Considering we are on state **3**, and we select action → (move to right), then the new state will be **4**

3 ↑ 22

Considering we are on state **3**, and we select action ↑ (move up), then the new state will be **22**

3 ← 2

Considering we are on state **3**, and we select action ← (move to left), then the new state will be **2**

Q-Learning

Another way we can look into things is to consider that the car moves from one state to another state, based on the selected action.

$$f(\mathbf{state}) = \mathbf{action},$$

state = a vector with values associated with the state,

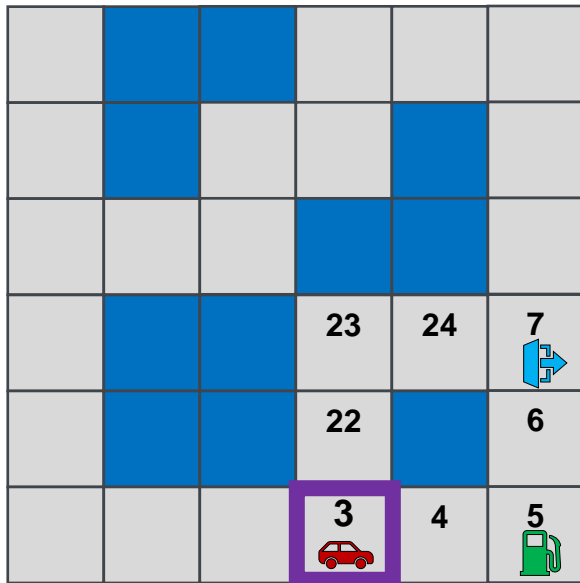
action = a scalar that reflects the action

In our case we can say that *action*=1 means move left, *action*=2 means move right, *action*=3 means move up, ...

OBS: *action* can also be a vector with all possible actions and we can use a **softmax** method to select the next action

Q-Learning

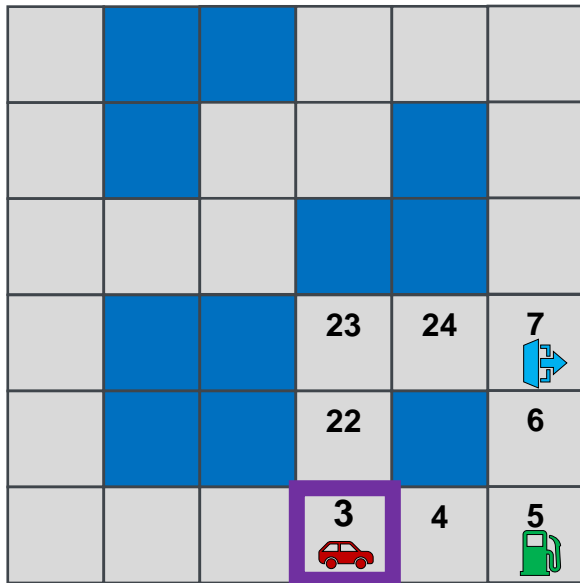
Now, let's discuss a little bit how we can define the function f :



Let's consider that we are at state **3** and we need to decide the next action. One way of looking into this case, is to think that we have two possible solutions (go to state **4** or go to state **22**) and choose the best one.

Q-Learning

Now, let's discuss a little bit how we can define the function f :



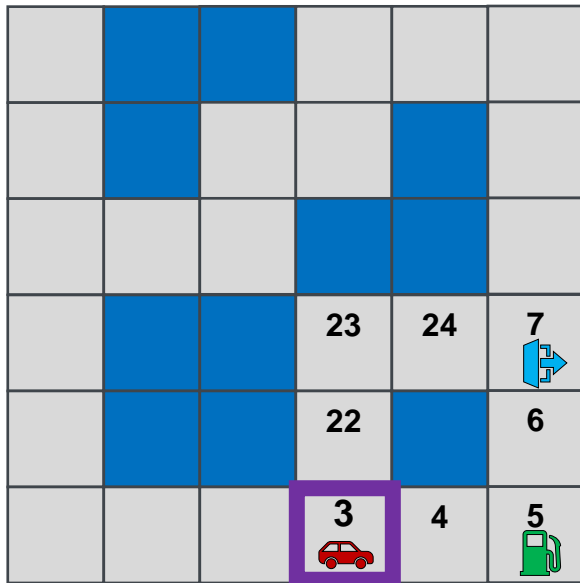
With this in mind, let's discuss solution to get from state 3 to state 7:

Solution no. 1 (3 → 4 → 5 ↑ 6 ↑ 7)

$$\begin{aligned}
 TotalReward &= R(3, \rightarrow, 4) + R(4, \rightarrow, 5) + R(5, \uparrow, 6) + R(6, \uparrow, 7) \\
 &= (-1) + (-1 + 5) + (-1) + (-1 + 1000) \\
 &= -1 + 4 - 1 + 999 = \mathbf{1001}
 \end{aligned}$$

Q-Learning

Now, let's discuss a little bit how we can define the function f :



With this in mind, let's discuss solution to get from state 3 to state 7:

Solution no. 1 (3 → 4 → 5 ↑ 6 ↑ 7)

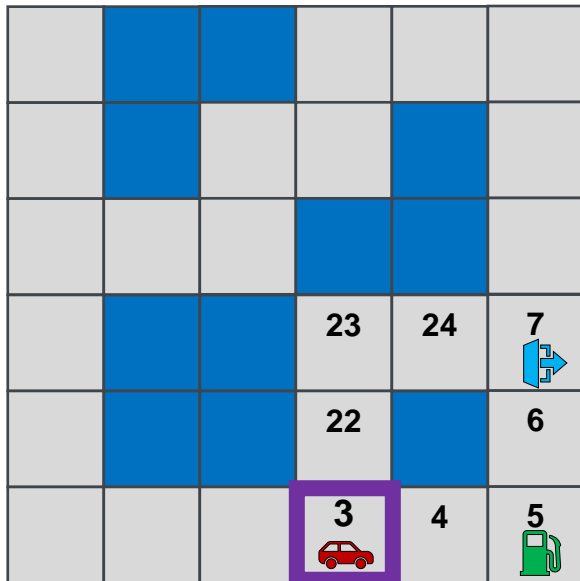
$$\begin{aligned} \text{TotalReward} &= R(3, \rightarrow, 4) + R(4, \rightarrow, 5) + R(5, \uparrow, 6) + R(6, \uparrow, 7) \\ &= (-1) + (-1 + 5) + (-1) + (-1 + 1000) \\ &= -1 + 4 - 1 + 999 = \mathbf{1001} \end{aligned}$$

Solution no. 2 (3 ↑ 22 ↑ 23 → 24 → 7)

$$\begin{aligned} \text{TotalReward} &= R(3, \uparrow, 22) + R(22, \uparrow, 23) + R(23, \rightarrow, 24) \\ &\quad + R(24, \rightarrow, 7) = (-1) + (-1) + (-1) + (-1 + 1000) \\ &= -1 - 1 - 1 + 999 = \mathbf{996} \end{aligned}$$

Q-Learning

Now, let's discuss a little bit how we can define the function f :



With this in mind, let's discuss a solution to get from state 3 to state 7:

Solution no. 1 $(3 \rightarrow 4 \rightarrow 5 \uparrow 6 \uparrow 7) = 1001$

Solution no. 2 $(3 \uparrow 22 \uparrow 23 \rightarrow 24 \rightarrow 7) = 996$

Obviously, the best solution is solution no. 1 (with a total reward of 1001). What is more important to deduce from this case is that **if we would know the value of all solutions from one state, we could choose the MAXIMAL ONE** and find the best solution.

Q-Learning

With this in mind, we can define a Quality function **Q** that reflects the reward we can obtain if we get if we apply an action (a) to a state (s)

$$Q(s, a) = R(s, a, s') + \max_{1 \leq i \leq n} Q(s', a_i), \text{ where}$$

s' = the new state we obtain if we apply action a over state s ,

n = total number of possible actions available on state s' ,

$R(s, a, s')$ = the reward we get from going applying action a over state s

Q-Learning

Let's compute the Quality function for some state:

[illegible]

$$Q(24, \rightarrow) = R(24, \rightarrow, 7) + \max_{1 \leq i \leq n} Q(7, a_i)$$

In this case “n” is 0 (once we reach state 7 there are no other possible actions as we reach the destination point).

This means that:

$$\begin{aligned} Q(24, \rightarrow) &= R(24, \rightarrow, 7) \\ &= (-1) + 1000 \\ &= 999 \end{aligned}$$

Q-Learning

Let's compute the Quality function for some state:

			23 Q(23,→) 998	24 Q(24,→) 999	7 ➡
			22		6
		2	3 ⛽	4	5 ⛽

$$Q(23, \rightarrow) = R(23, \rightarrow, 24) + \max_{1 \leq i \leq n} Q(24, a_i)$$

In this case “n” is 1 (the possible state from 24 is → (to left)).

In reality, there is also a possibility to move from state 23 to state 22. However, since we have reach state 23 moving up from state 22, going back will just create a circular scenario. We will discuss more about this scenario in the next slides.

This means that:

$$\begin{aligned} Q(23, \rightarrow) &= R(23, \rightarrow, 24) + Q(23, \rightarrow) \\ &= (-1) + 999 \\ &= 998 \end{aligned}$$

Q-Learning

Let's compute the Quality function for some state:

[illegible]

$$Q(6, \uparrow) = R(6, \uparrow, 7) + \max_{1 \leq i \leq n} Q(7, a_i)$$

In this case “n” is 0 (once we reach state 7 there are no other possible actions as we reach the destination point).

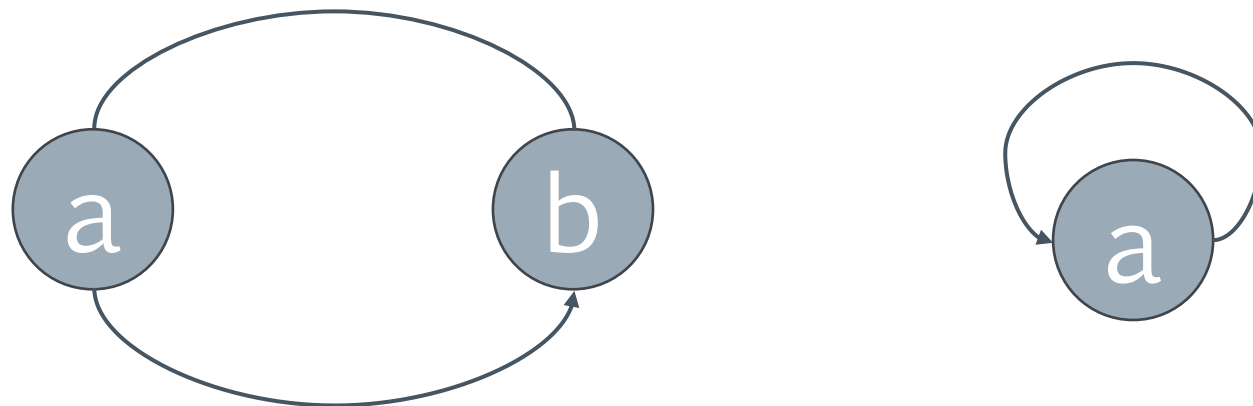
This means that:

$$\begin{aligned} Q(6, \uparrow) &= R(6, \uparrow, 7) \\ &= (-1) + 1000 \\ &= 999 \end{aligned}$$

Q-Learning

What happens if we can go from state (a) to state (b), but also from state (b) to state (a) ?

Or if we can choose to remain in state (a) because this type of action is possible.



Q-Learning

If this is the case, we can end up in a circular loop (an infinite loop) that will end up in not finding the optimal result. The solution is to introduce a new term (γ) called discount factor into the equation:

$$Q(s, a) = R(s, a, s') + \gamma \times \max_{1 \leq i \leq n} Q(s', a_i), \text{ where}$$

s' = the new state we obtain if we apply action a over state s ,

n = total number of possible actions available on state s' ,

$R(s, a, s')$ = the reward we get from going applying action a over state s ,

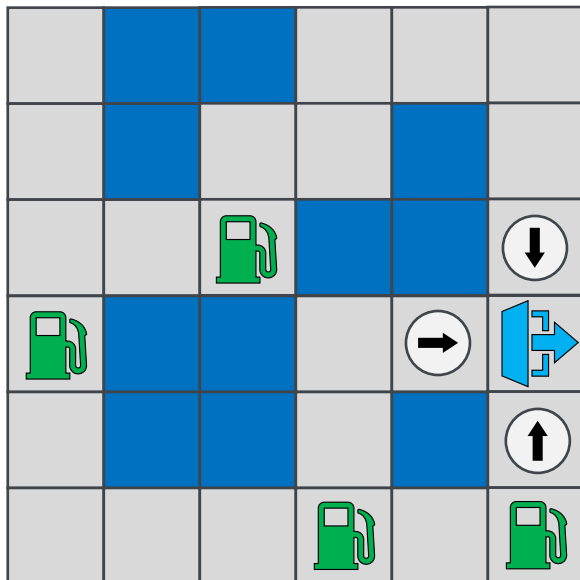
γ = discount factor, $0 \leq \gamma < 1$



While the main purpose of the discount factor is to avoid circular loops, we can also look at it as a way to reflect how valuable the current reward is against future ones.

Q-Learning

But where neural networks come in place ? Up to this moment we've only discussed solution that don't require any machine learning algorithm.



For example, in this case we can start from the end (the destination) and compute the Q function for all possible states that can go through a specific action to the final state.

Q-Learning

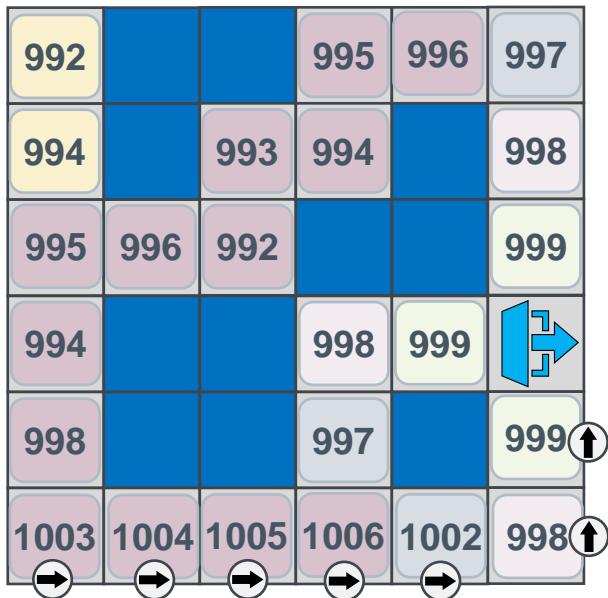
But where neural networks come in place ? Up to this moment we've only discussed solution that don't require any machine learning algorithm.

					↓
					998
		⛽			999
⛽			998	999	→
			↑		999
			⛽	→	998

We can repeat the previous process until Q functions are computed.

Q-Learning

But where neural networks come in place ? Up to this moment we've only discussed solution that don't require any machine learning algorithm.



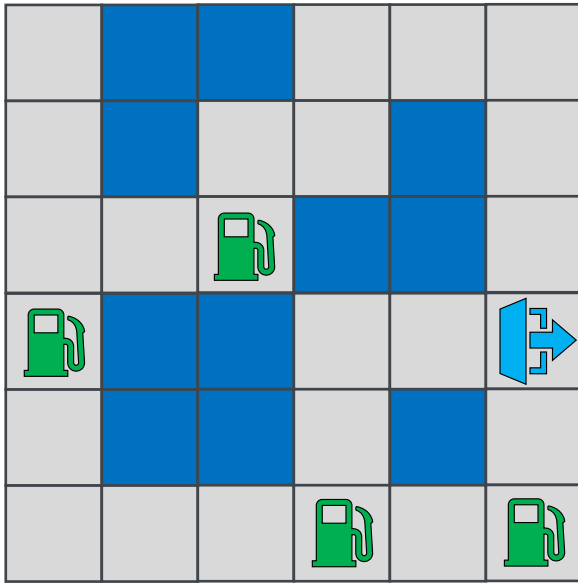
We can repeat the previous process until Q functions are computed.

Once we complete computed all Q functions and the associated actions we just need to follow the Q function results and its action (in our case move to → for 5 times and them move up two times) and we will reach our destination.

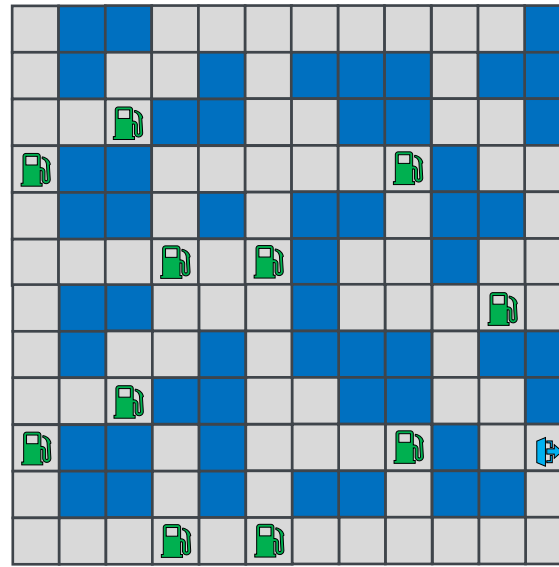
π

Q-Learning

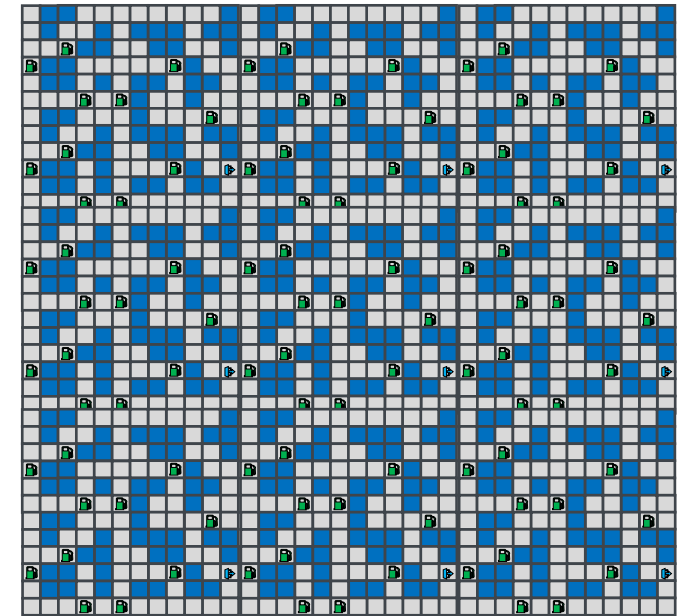
So, we don't really need any neural networks for this case !



No need for NN
(only **24** states to compute)



What about now ?
(**100+** states to compute)



How about now ?
(**1000+** states to compute)

Q-Learning

The observation here is that we can solve these type of problems if the number of states and actions is finite (or at least a reasonable number of states).

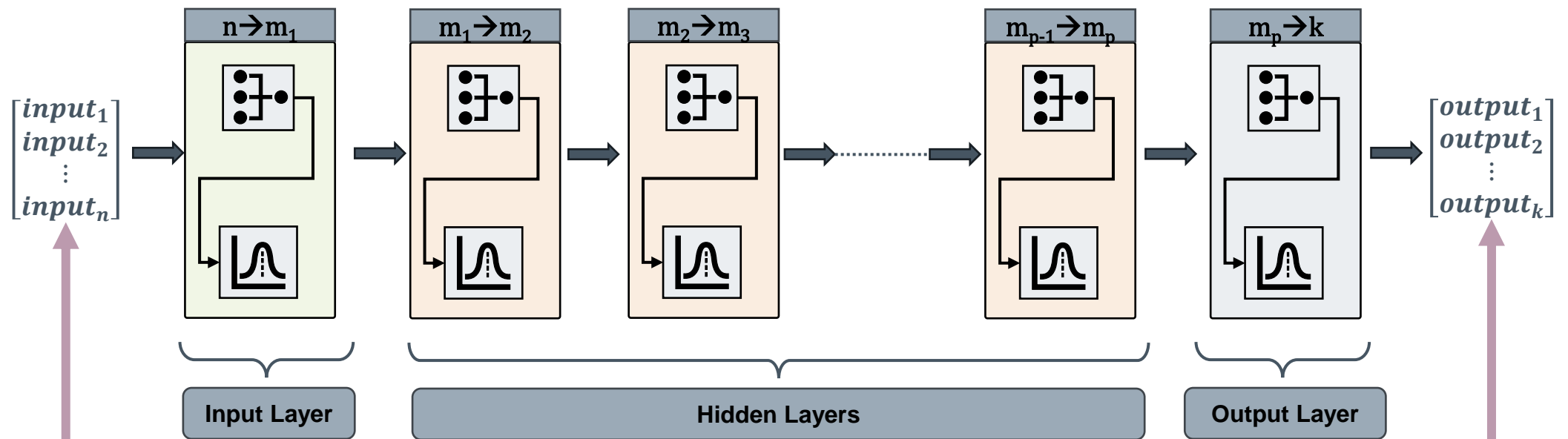
However :

- If the number of states is too large
- If we don't know all possible states
- If we don't know where the final end point is

... We might need a different approach ...

Q-Learning

So ... how would a neural network look like in this case ?

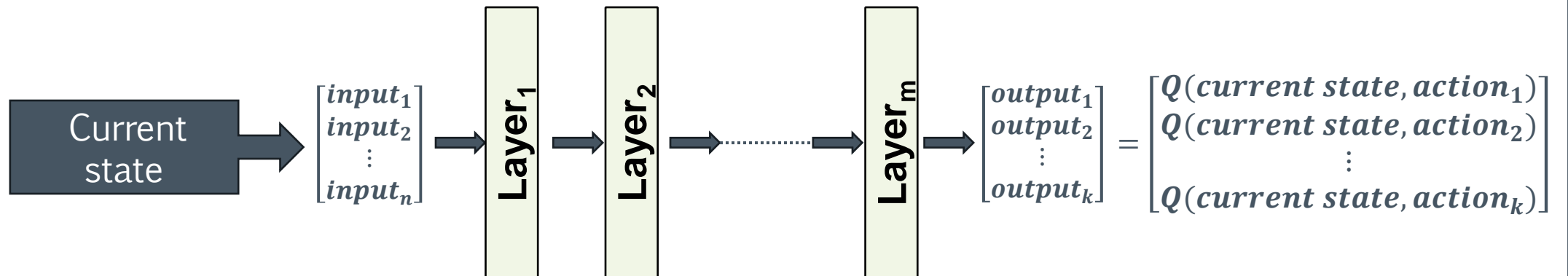


But, what is the **Input** and the **Output** vector ?

Q-Learning

What is the **input** and the **output** ?

- › The **input** of the network should be current state (by state we refer to all information associated with a stated).
- › The **output** should be a vector with the approximation of Q value for every possible action



Q-Learning

But what is a state in this case ? To better understand this let's imagine that we want to train an AI to 1990 Prince of Persia



Q-Learning

Let's take the following state of the game (noticed that I have used the word state):



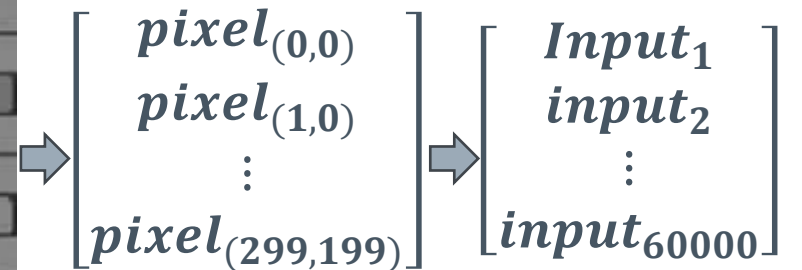
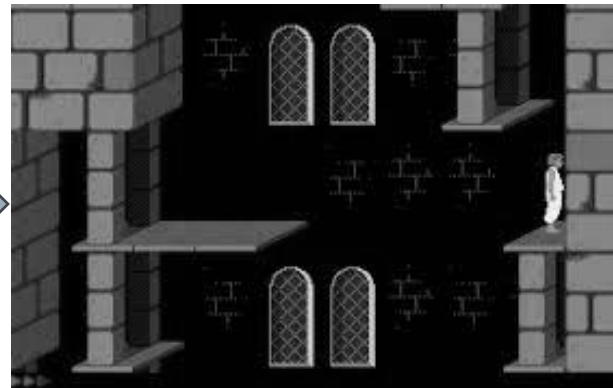
This is the main character that you can control with your keyboard.

ID	Action
1	Go Left
2	Go Right
3	Jump Left
4	Jump Right
5	Climb Up
6	Go Down

Q-Learning

So... how can we convert the state into a vector of numbers.

Solution no 1: Take a snapshot of the screen, convert it into pixels and then create a vector of values (we may use another prefilter to convert a pixel into a gray value from **0(black)** to **1(white)**).

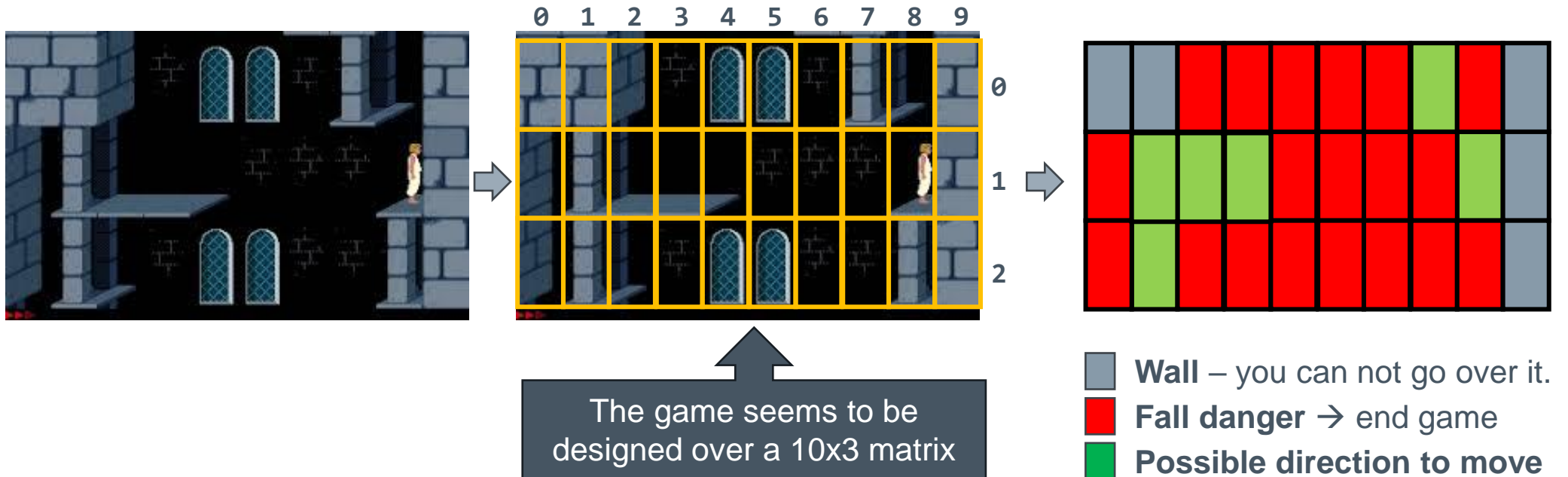


The game has a screen of
300x200 pixels

Q-Learning

So... how can we convert the state into a vector of numbers.

Solution no 2: Understand some elements from the game (where walls and empty spaces are, etc). This implies having an extra set of methods (**a smarter sensor**) that extracts these information.



Q-Learning

So... how can we convert the state into a vector of numbers.

Solution no 2: Understand some elements from the game (where walls and empty spaces are, etc). This implies having an extra set of methods (**a smarter sensor**) that extracts these information.



0	0	1	1	1	1	1	2	1	0
1	2	2	2	1	1	1	1	2	0
1	2	1	1	1	1	1	1	1	0

- Wall – value 0.
- Fall danger → value 1
- Possible direction to move (2)

$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 0 \\ 0 \\ 8 \\ 1 \end{bmatrix}$

Character coordinates (x,y)

Q-Learning

So... how can we convert the state into a vector of numbers.

Solution no 3: combine the previous 2 solutions

Solution no 4: use solution no 1, but only for a small area (let's say 100 x 100 pixels) around the character

Solution no 5: use solution no 2, but only for a small area (+/- one rectangle in each direction).

... what's important to understand is that there are practically an indefinite number of solutions

Q-Learning

A couple of observations for this example:

1. Using the entire screen is simpler (as you don't need to do much, just process the image). On the other hand, the amount of data that needs to be processed is quite high and as a result, the neural network might converge slowly.
2. Using a preprocess version (like in solution no 2) is more complicated (implies understanding how the game works and to write specialized converters and image identification). On the other hand, the input is more relevant, and the convergence will be faster.

Q-Learning

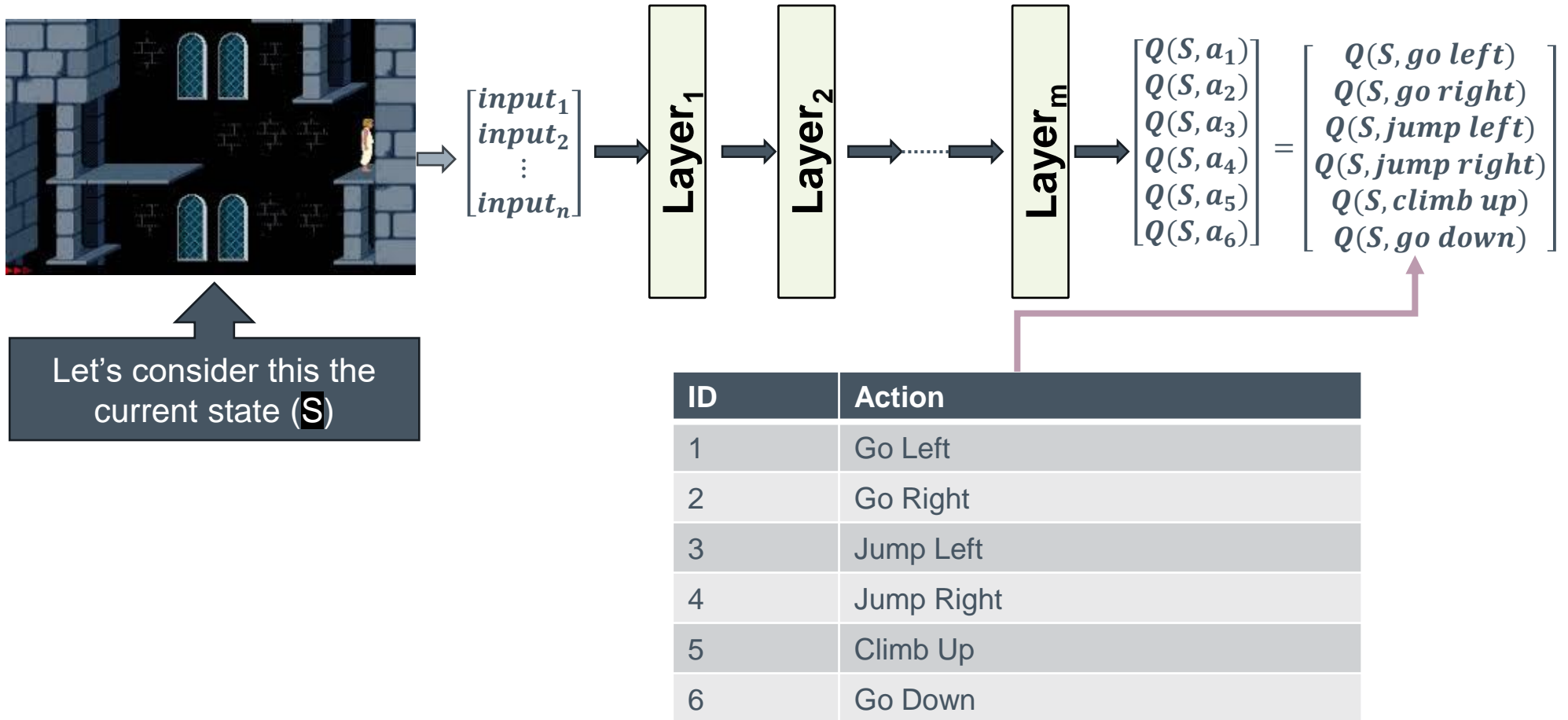
And as a general observation:

The better the sensors (the way the environment is read and interpreted) the better the Q-Learning algorithm will be (in terms of performance, accuracy)

π

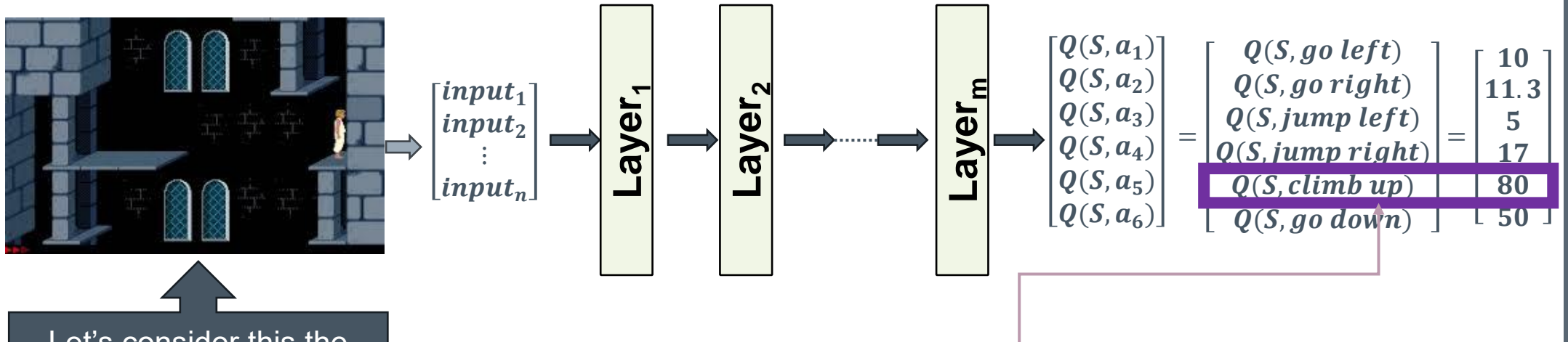
Q-Learning

What about the output ?



Q-Learning

What about the output ?



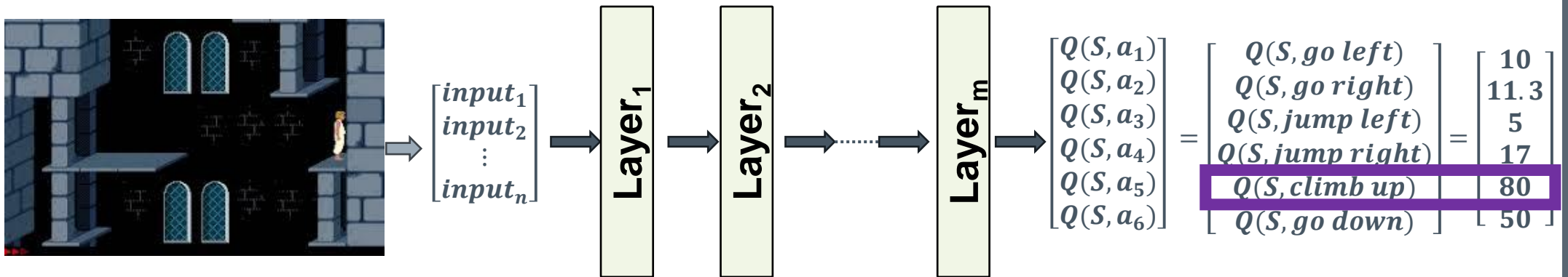
We can look at this result in two ways:

1. The highest result is 80 (the corresponds to climbing up) so **we should climb up**
2. We know the value of each Q function for different actions (e.g. $Q(S, go\ right) = 11.3$, $Q(S, go\ down) = 50$)

Q-Learning

Exploitation vs Exploration.

Let's consider the previous example and let's that for that's specific state the best value (the highest Q value is 80 that corresponds to the action of climbing up). **Should we always take this action ?**



Q-Learning

Exploitation vs Exploration.

In practice, if we always take the best possible action, we will never explore anything (or in other words, we might end up in a local minimum and never get out of it). As such, it is recommended that from time to time we don't take the best action (**exploitation**) but instead we chose another one (**exploration**).

Q-Learning

Exploitation vs Exploration.

	Exploration	Exploitation
Definition	Exploration involves trying out different actions and visiting new states to gather more information about the environment. The idea is to learn something new that wasn't known before.	Exploitation involves using the current knowledge to choose actions that yield the highest rewards / leveraging what was already learned.
When to use	Exploration is essential in the early stages of learning or in any situation where the agent doesn't have enough information about the environment.	Exploitation is beneficial when the agent has sufficient knowledge about the environment and wants to maximize its performance or rewards.

Q-Learning

Exploitation vs Exploration.

There are several ways to balance exploitation vs exploration in a Q-Learning algorithm. The most popular one is Epsilon-Greedy defined as follows:

$$\text{given } q = \begin{bmatrix} Q(S, a_1) \\ Q(S, a_2) \\ \vdots \\ Q(S, a_n) \end{bmatrix}, \text{ and } \epsilon \text{ a probability } (0 \leq \epsilon \leq 1), \text{ then}$$
$$\text{selected action} = \begin{cases} i, Q_i = \max_{1 \leq j \leq n} Q_j, & \text{with probability } (1 - \epsilon) \\ \text{random}(n), & \text{with probability } \epsilon \end{cases}$$

Q-Learning

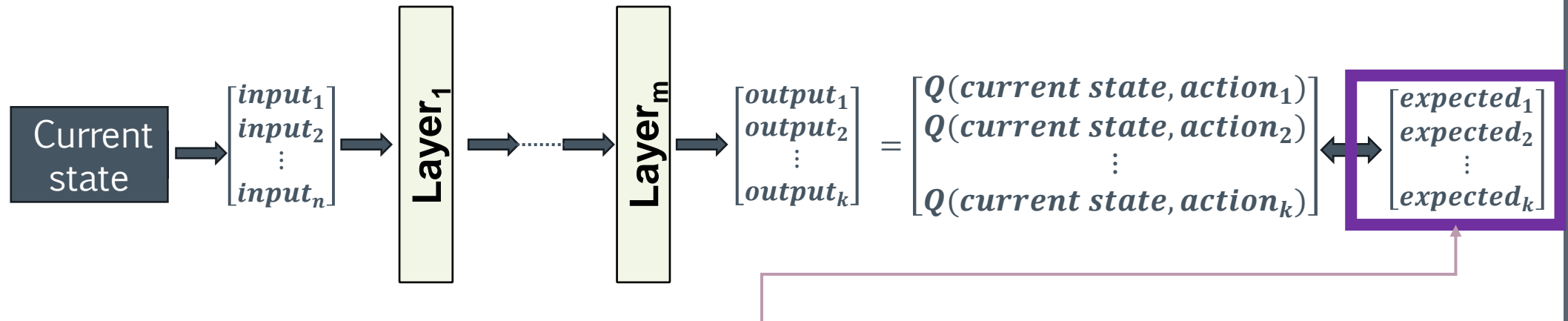
Exploitation vs Exploration.

Usually, the ϵ value is selected between 0.1 and 0.2 (10 to 20% chance to chose something random then the best solution).

It is also recommended to apply a decay rate (reduce the ϵ with each iteration). This makes sure that at the beginning, we will **explore** more, and after we learn the environment (after several iterations), we will **exploit** more.

Q-Learning

Right now, we know how to build a neural network and how to interpret its output into taking an action. **But how do we train this neural network ?**



The problem is how we can find the expected (the actual best actions) to compare them with what our network predicted. If we can do this, we just need to apply the backpropagation and the network will train itself.

KEEP IN MIND that we are in a scenario with a lot of states where we can not store all possible states and actions !

Q-Learning

The solution for the previous problem is to use Bellman equation:

$$Q_{target}(s, a) = R(s, a, s') + \gamma \times \max_{1 \leq i \leq n} Q_{predicted}(s', a_i), \text{ where}$$

s' = the new state we obtain if we apply action a over state s ,

n = total number of possible actions available on state s' ,

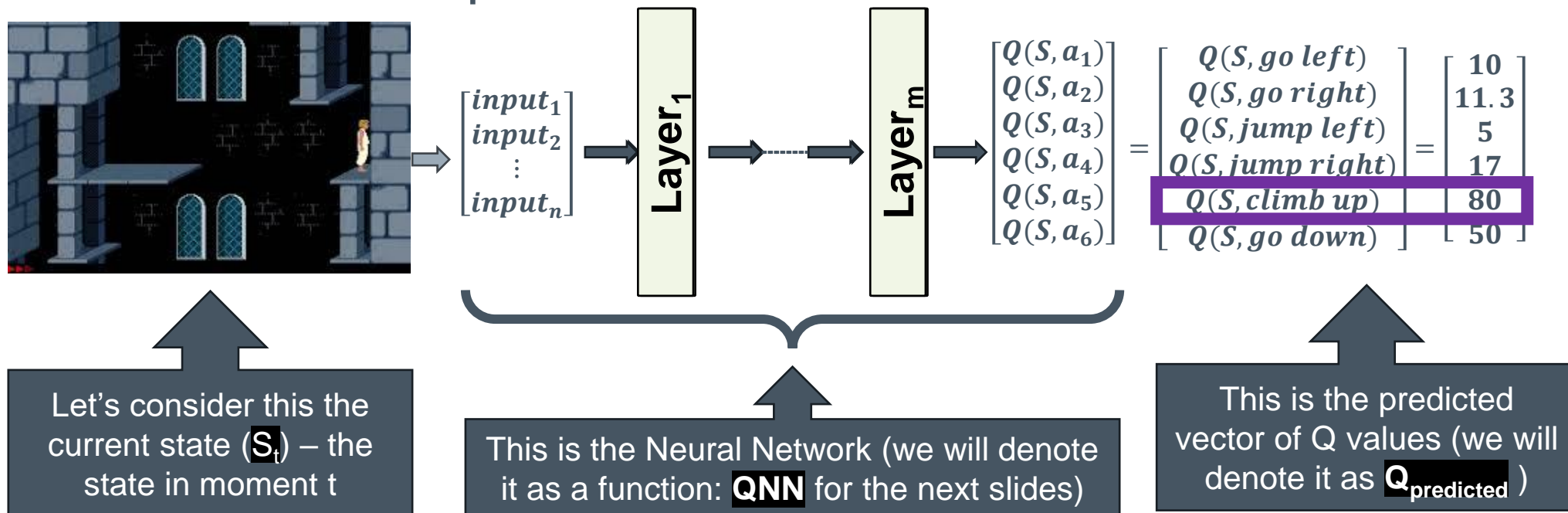
$R(s, a, s')$ = the reward we get from going applying action a over state s ,

γ = discount factor, $0 \leq \gamma < 1$,

$Q_{predicted}$ = the predicted result of the neural network in states s'

Q-Learning

Let's see an example to better understand how this works:

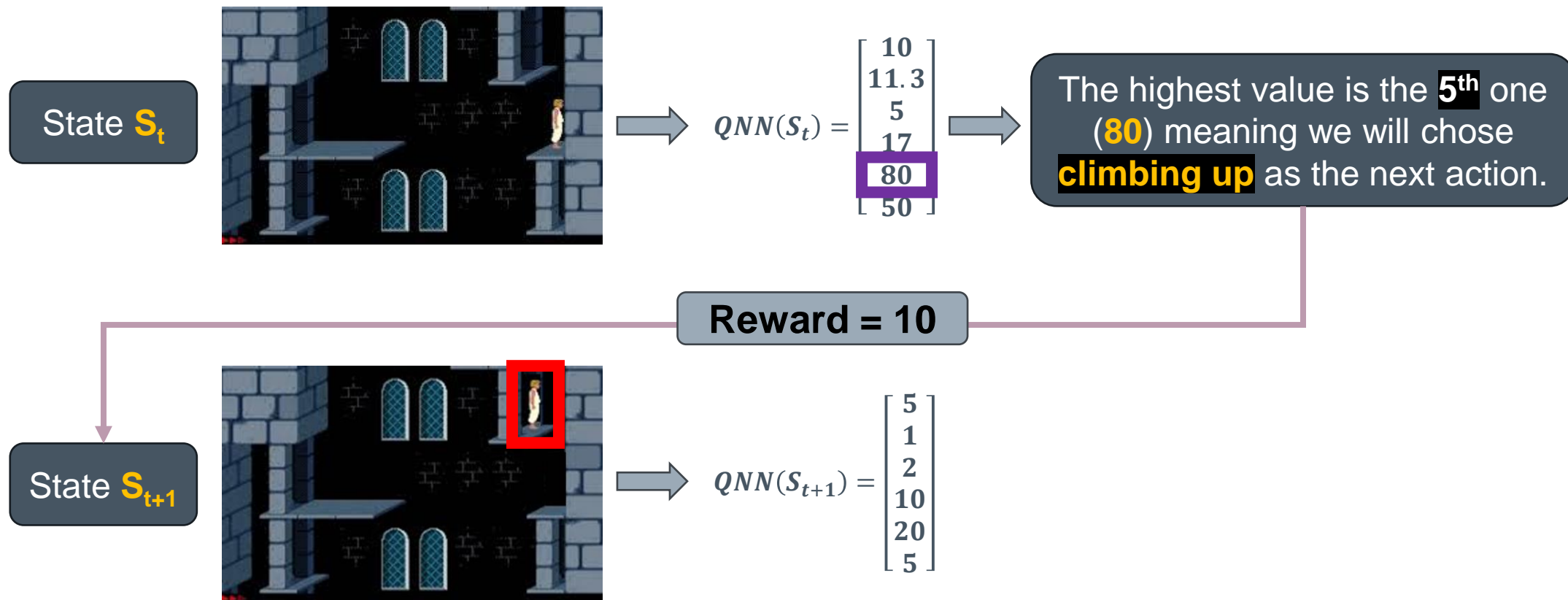


Then, we can simplify the process and say that:

$$Q_{predicted} = QNN(S), \text{ or in our case } QNN \left(\text{state image} \right) = QNN(S_t) = \begin{bmatrix} 10 \\ 11.3 \\ 5 \\ 17 \\ 80 \\ 50 \end{bmatrix}$$

Q-Learning

Let's see an example to better understand how this works:



Q-Learning

So ... we have the following:

$$QNN(S_t) = \begin{bmatrix} 10 \\ 11.3 \\ 5 \\ 17 \\ 80 \\ 50 \end{bmatrix}, QNN(S_{t+1}) = \begin{bmatrix} 5 \\ 1 \\ 2 \\ 10 \\ 20 \\ 5 \end{bmatrix} \text{ with } \max_{1 \leq i \leq 6} QNN(S_{t+1}) = 20, \text{Reward} = R(S_t, 5 \text{ (climb up)}, S_{t+1}) = 10$$

Let's see how we can apply Bellman equation in this case:

$$\text{Bellman equation: } Q_{target}(s, a) = R(s, a, s') + \gamma \times \max_{1 \leq i \leq n} Q_{predicted}(s', a_i)$$

In our case, this will translate as follows:

$$\begin{aligned} Q_{target}(S_t, 5(\text{climb up})) &= R(S_t, 5 \text{ (climb up)}, S_{t+1}) + \gamma \times \max_{1 \leq i \leq 6} QNN(S_{t+1}) = 10 + \gamma \times 20 \\ &= 10 + 0.1 \times 20 = \mathbf{12} \text{ (considering the discount factor } \gamma \text{ is 0.1)} \end{aligned}$$

Q-Learning

This means that for a state S_t we have

$$Q_{NN}(S_t) = Q_{predicted}(S_t) = \begin{bmatrix} 10 \\ 11.3 \\ 5 \\ 17 \\ \mathbf{80} \\ 50 \end{bmatrix}, \text{ with the 5th value (80) being the highest}$$

And we know the expected (target Q value) for the 5th with a decay rate of 0.1 is **12** and a reward of +10 from moving from S_t to S_{t+1} .
This means that:

$$Q_{expected}(S_t) = \begin{bmatrix} 10 \\ 11.3 \\ 5 \\ 17 \\ \mathbf{12} \\ 50 \end{bmatrix}$$

Notice that we have copied all values from the $Q_{predicted}$ vector except the 5th one

And now we can use the expected (target) result and the predicted one to train the network 😊

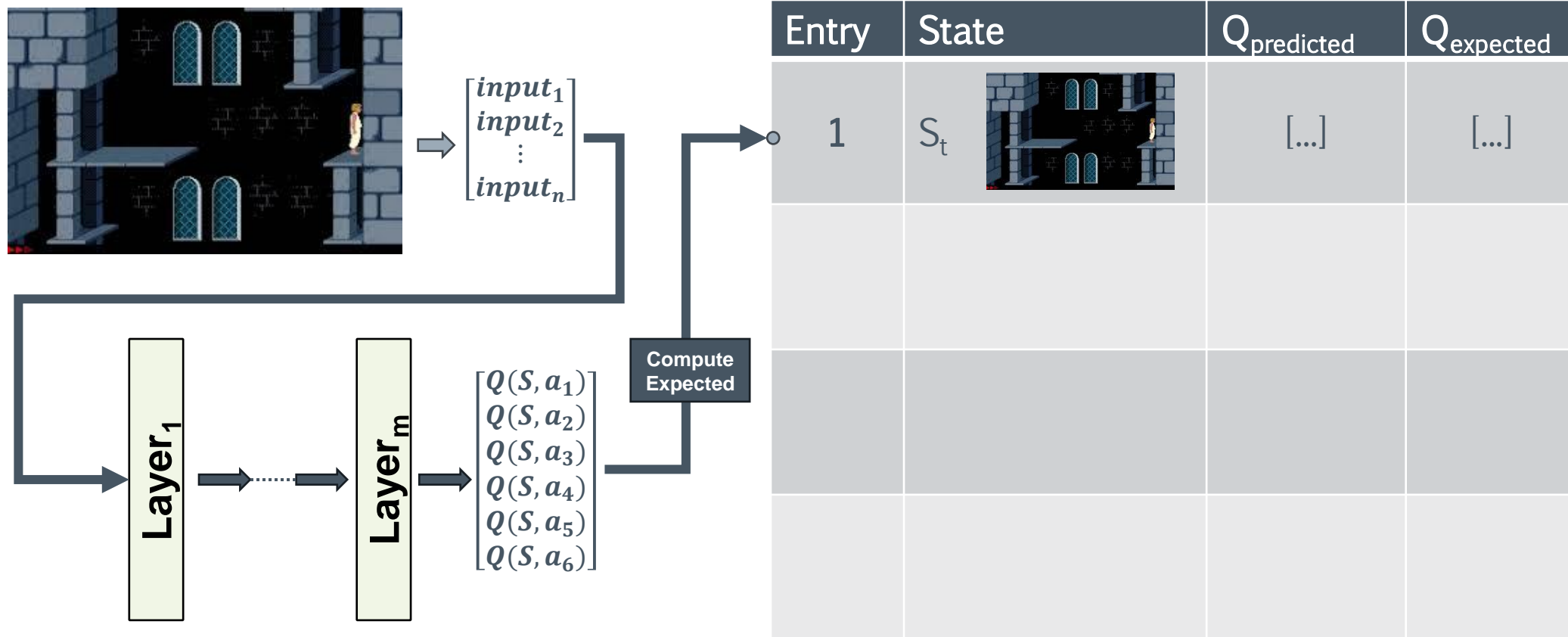
Q-Learning

Observations:

1. Instead of training each case, it is better to collect several pairs (state, prediction and expected) value and train the network with all of them.
2. And since many states are related one to another one (S_{t+1} is clearly related with S_t as we can only get to S_{t+1} from state S_t) it is better to shuffle the collection.
3. Often the collection from point 1 is a circular dataset (meaning that it has a limited number of entries, and adding a new one will remove the oldest one).
4. This procedure is called **replay**

Q-Learning

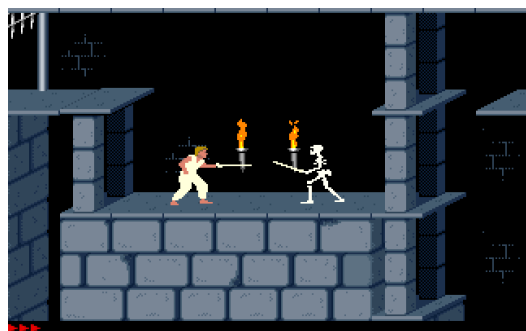
Building a dataset:



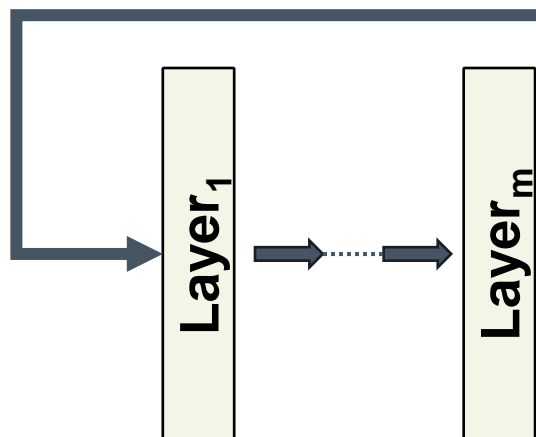
π

Q-Learning

Building a dataset:



$\begin{bmatrix} input_1 \\ input_2 \\ \vdots \\ input_n \end{bmatrix}$



$\begin{bmatrix} Q(S, a_1) \\ Q(S, a_2) \\ Q(S, a_3) \\ Q(S, a_4) \\ Q(S, a_5) \\ Q(S, a_6) \end{bmatrix}$

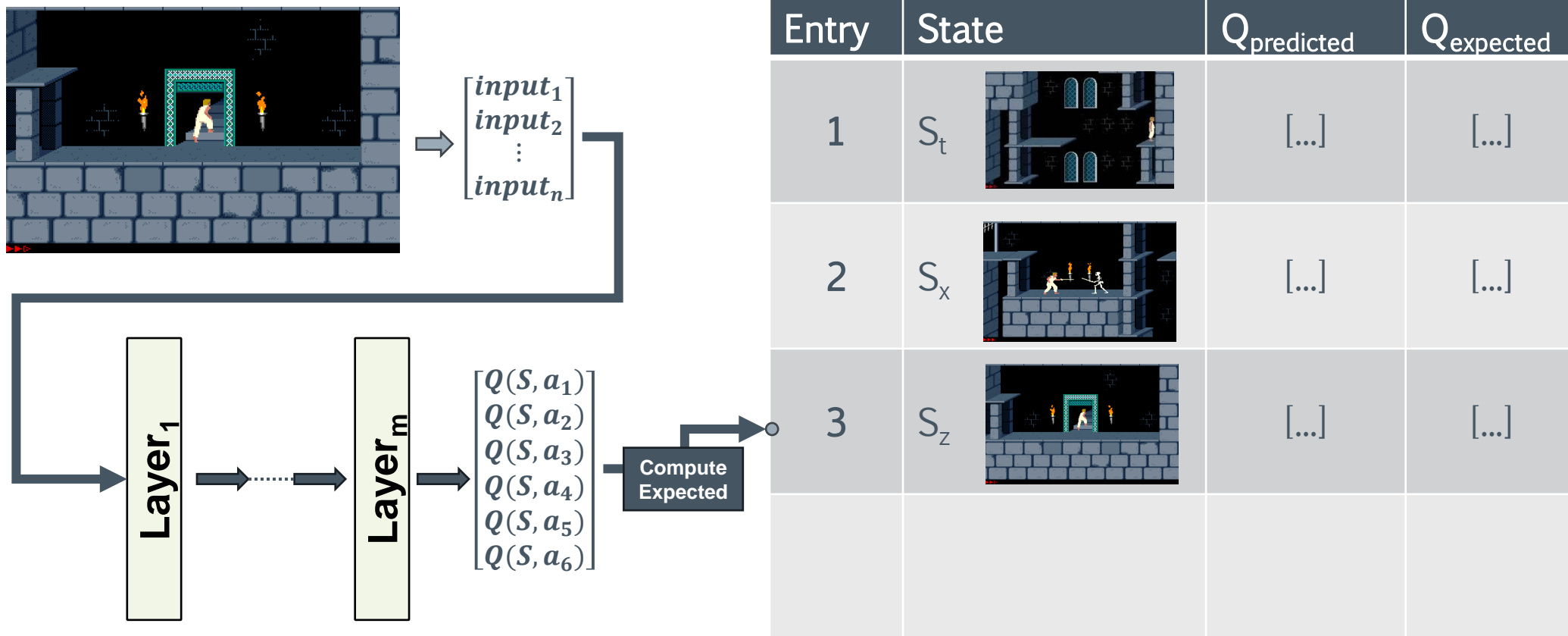
Compute
Expected

Entry	State	$Q_{\text{predicted}}$	Q_{expected}
1	S_t 	[...]	[...]
2	S_x 	[...]	[...]

π

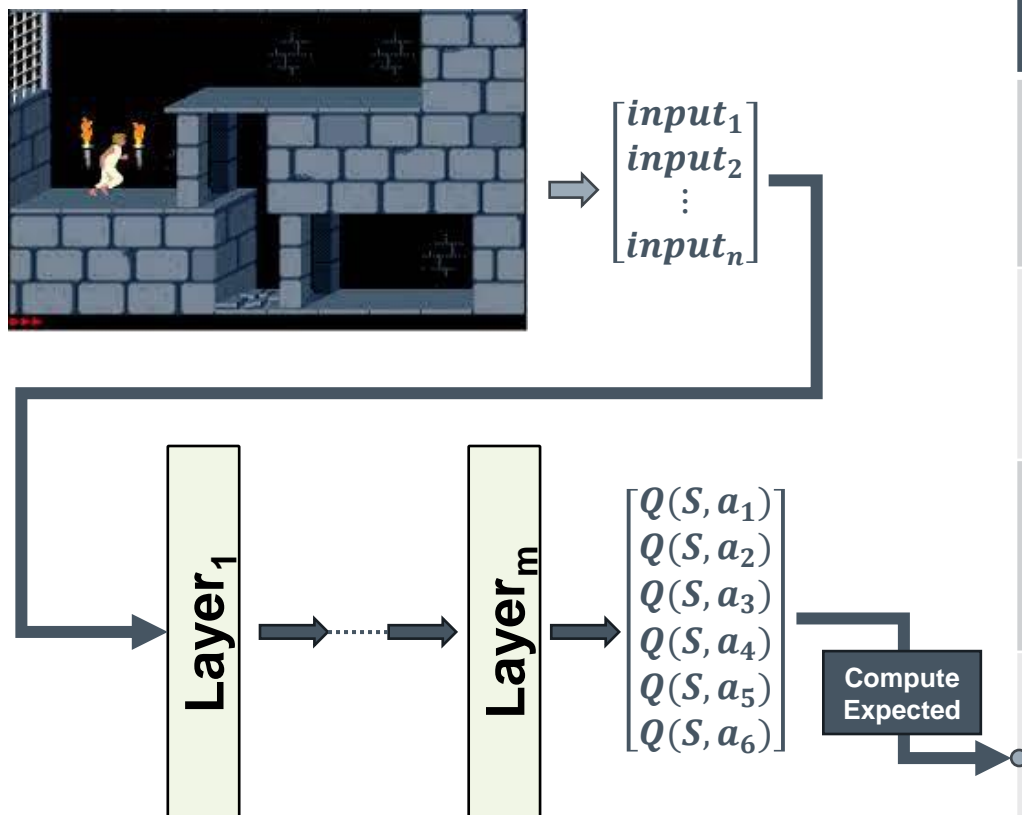
Q-Learning

Building a dataset:



Q-Learning

Building a dataset:



Entry	State	$Q_{predicted}$	$Q_{expected}$
1	S_t 	[...]	[...]
2	S_x 	[...]	[...]
3	S_z 	[...]	[...]
4	S_p 	[...]	[...]

Q-Learning

Building a dataset:

Now we have a data set, with input values (the states) and expected values (Q_{expected}) that we can use for training.

The process is repeated after each training iteration, by adding in the dataset new entries and removing old ones.

Entry	State	$Q_{\text{predicted}}$	Q_{expected}
1	S_t 	[...]	[...]
2	S_x 	[...]	[...]
3	S_z 	[...]	[...]
4	S_p 	[...]	[...]

Tip & Tricks

- › Its best **if your initial training is done with some success scenarios** (e.g. in case of Prince of Persia this translates into playing your game yourself and use your decisions as a base for your neural network).
- › **You should choose the rewards carefully**. Its also important to point out that identifying rewards implies preprocessing (e.g. in case of Prince of Persia you will need somehow to understand when the game ends).

Tip & Tricks

- › Usually, a good **loss function is MSE** (one reason for this is how Q_{expected} is computed by copying values from $Q_{\text{predicted}}$). Those values if subtracted will result in 0, and as a result, only the value that corresponds to a specific action will count.
- › **Exploration vs Exploitation** → it is important to maintain a balance in this case if you want your agent to be “smart enough”.

π

Q & A

