MACHINE LEARNING

MEI/1

University of Beira Interior,
Department of Informatics

Hugo Pedro Proença,
hugomcp@di.ubi.pt, 2024/2025

Machine Learning

[09]

Syllabus

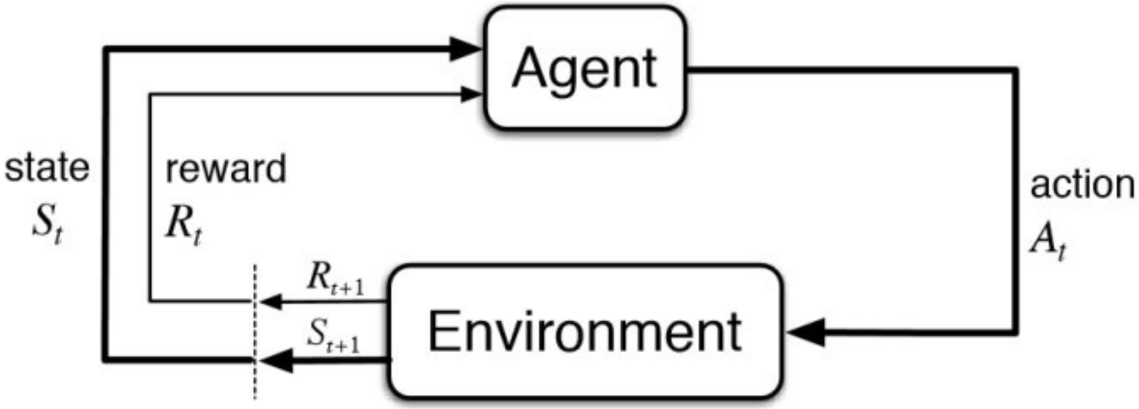
- Reinforcement Learning

Reinforcement Learning

- This is the area of Machine Learning concerned with how a computational agent should take the **best actions** in an **environment**, so to **maximize** its cumulative reward.
- The most intuitive analogy is "**pet training**". Suppose that we have a dog that, upon our request, can "sit", "lay down" or "do nothing".
- Each time we give an order to the dog, and he makes the right action (i.e., obeys) leads to a reward.
- After a while, the dog starts to "understand" that the best thing is to act according to our command

Reinforcement Learning

- In this analogy, the owner is the environment, which gives to the dog a current state (**S**$_t$). The dig is the agent that sees the state and should take a corresponding action (**A**$_t$).
- The owner reacts to the action taken with a reward (**R**$_{t+1}$) and s new state (**S**$_{t+1}$).



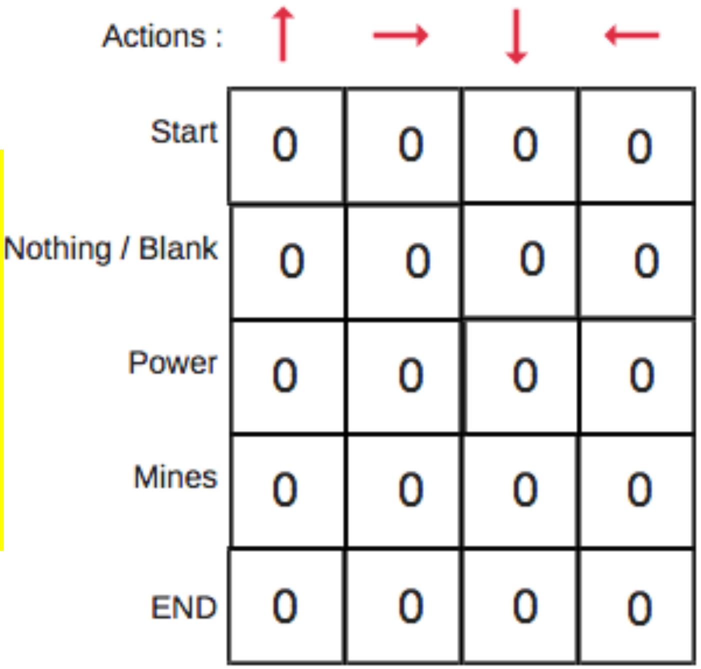
- Reinforcement Learning is one of the three major paradigms of Machine Learning, along with Supervised Learning and Unsupervised Learning.
  - Though both **supervised** and **reinforcement learning** use some kind of mapping between the input/output, in supervised learning the *feedback* provided is the explicit set of actions for performing a task. Instead, Reinforcement Learning uses rewards and punishment as signals for positive and negative behavior.

Q-learning Algorithm

- The most classical solution for Reinforcement Learning is the **Q-learning** algorithm, which gives to the agent a **knowledge set** , in form of a **Q-table**.
- A Q-table is a bidimensional structure with size "#total states" x "#total actions", storing the value for each pair state/action (**Q-values**)
- Each Q-value represents the "**quality**" of an action taken at that state.
- **Higher Q-values** correspond to **higher chances** of getting greater rewards in the future

4 possible actions

5 possible states



Actions :	↑	→	↓	←
Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0

Q-learning

- Q-values are calculated according to the formula:

$$Q_{new}(s_t, at) \leftarrow (1-\alpha)Q(s_t, at) + \ \alpha(Rt + \gamma\ maxa\ Q(S_{t\ +\ 1,}\ a))$$

Old value

Immediate reward

Discounted estimate of optimal future reward

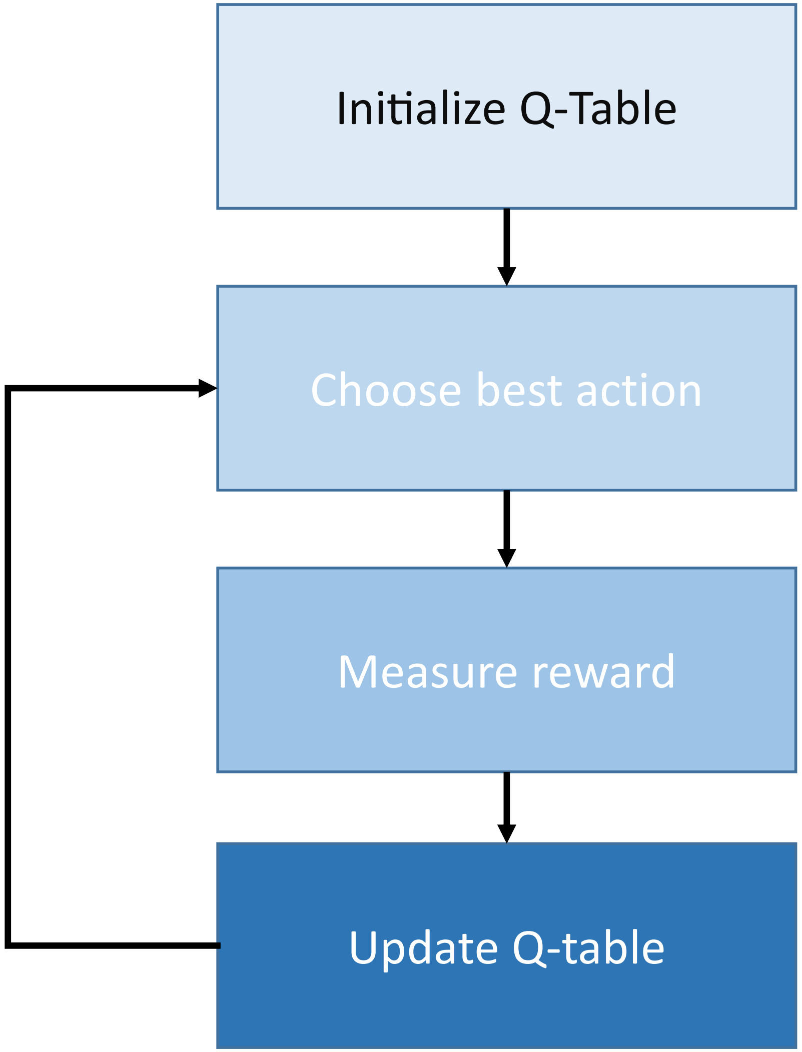
- The γ parameter controls the balance between the immediate/future rewards weight.
- It is important because we want our agent to focus more on the immediate rewards while not fully ignoring the future rewards.
- The α parameter is the learning rate, and determines how fast the values in the Q-table change.

Q-learning

- Supposing that at the beginning, all Q-values were set to 0, there is no best action in the start.
- In these circumstances we have to choose randomly.
  - That will be problematic once a positive Q-value was found, as the agent will perform that action indefinitely.
- We don't want that solution, as there might be even higher Q-values in the future, if that momentaneous optimal action is not taken at the first place.
  - That's where the γ comes into play.
  - It decides whether we should take the best local action (**exploitation**) or should instead take a risky random sample of the space of actions (**exploration**).
- This exploitation/exploration strategy has the advantage of never stopping to explore.
- Typically, we set a high exploration rate at the beginning, and then decrease it gradually.

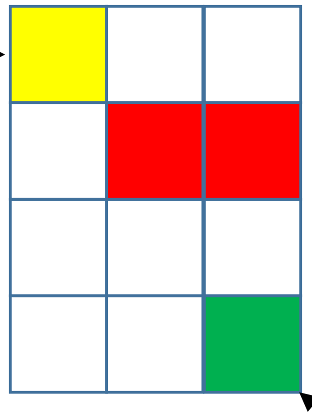
Q-learning

- The Q-Learning algorithm is pretty simple to run, and is composed of one initialization plus 3 iterative steps
- After a sufficient number of iterations, a good Q-Table is ready, and the agent has learned how to behave in a particular problem.
- This ”naïve” approach works well in practice for problems where it is realistic to keep a **list of all possible states**



Q-learning Example

- Suppose we want to learn an agent able to find the best path between two positions in a rectangular grid, avoiding the “red” positions, which will “kill” the agent.



The initial position

Dangerous positions

The goal

Considering “**N, S, E, W**” movements, there are only 4 actions

The board has 12 positions. Hence we have 12 different states

The Q-Table will have dimensions **12 x 4**

Q-learning Example

- We start by defining some reward/penalty values for each type of cells. Let's consider that **blank cells** have a small penalty (**-1**, to assure that we will not move indefinitely between cells), while **red** cells should be avoided (penalty=**-100**). Finally, the **goal** cell has a positive reward (**100**).
- We start by initializing all cells to 0.

- As we are starting the training now,

we simply perform a random action.

- Suppose we move "EAST".
- We will update the **[1, 3] cell**

	N	S	E	W
1,1			🟦	
1,2	🟦	🟦	🟦	🟦
1,3				
2,1				
2,2				
2,3				
3,1				
3,2				
3,3				
4,1				
4,2				
4,3				

🟨		
	🟥	🟥
		🟩

Learning rate

Immediate reward (-1)

Maximum expected future reward

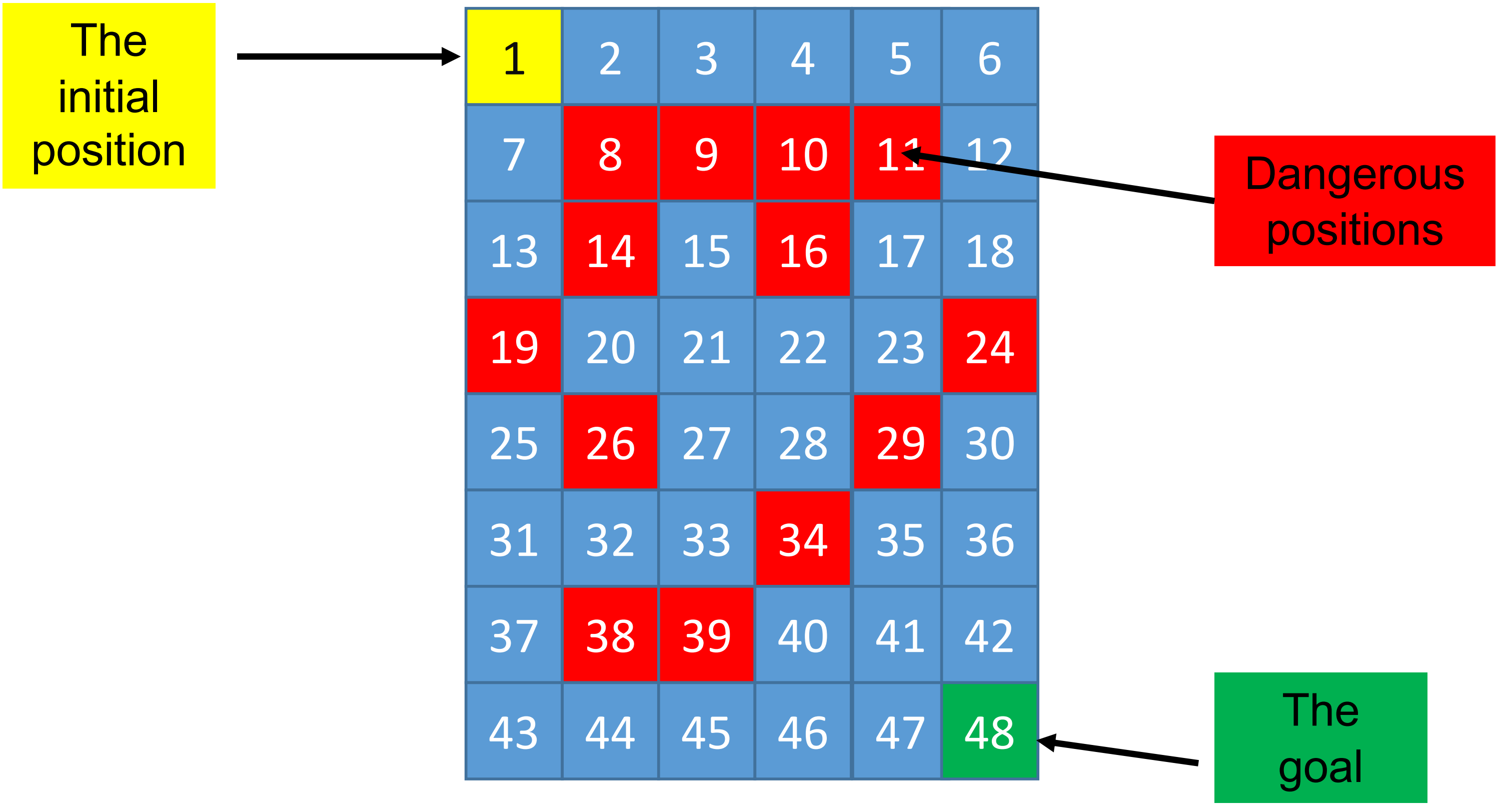
$$Q[1,3] = Q[1,3] + \alpha[R[1,3] + \gamma \max Q'[s', a'] - Q[1,3]]$$

Discount rate

The process is repeated (depth N), with each move being either the max Q value action (exploitation factor) or a random move (exploration, i.e., 1-exploitation)

Q-learning Example 2

- Now, suppose we want to learn an agent able to find the best path between two positions in a rectangular grid, avoiding the “red” positions, which will “kill” the agent.



The initial position

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36
37	38	39	40	41	42
43	44	45	46	47	48

Dangerous positions

The goal

Q-learning Example

- The Q-table will have 48 rows (# states) and 4 columns (# actions: LEFT, RIGHT, UP, DOWN).
- Initially, we should incorporate all the physical constraints in this problem, and create a Reward Matrix **R** (of size 48 x 48), that provides the R values for all possible transitions between states.
  - For example, from State 1, only States 2 and 7 can be reached.
  - Hence the first row of the Reward Matrix will have "**0**" value in positions (1,2) and (1,7) and "**-1**" in all others.
  - We should define an infinite reward to all transitions between states that led to State 48 (our goal).
  - Also, we should define a large penalty to all transitions that led to a **red position**.
- Finally, as we are interested in finding the short path, every transition between states should have a small penalty, to avoid unnecessary movements.
- Next, we initialize all the values in the Q-table to "0".
  - Note that some actions are not possible for some states. For instance, for state "1", only "RIGHT" and "DOWN" actions are valid.

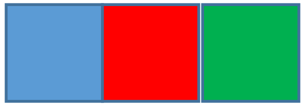
Q-learning Example

- When the process starts, suppose that by random selection, we choose State 7.
- The next step will be to predict what can happen if the agent was in State 7:
  - It can move to State 13 (”DOWN” action)
  - It can move to State 8 (“RIGHT” action)
- The transition 7 to 13 R(7,13) has cost -1
- However, R(7,8) has a much worse penalty ($-\inf$), as it corresponds to a dangerous position.
- This way, the future component of the equation will have value equal to “-1” , i.e., max ([-1, -inf]).
- The following update in the Q-table should be done:

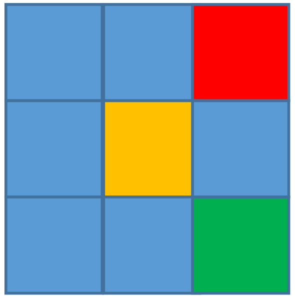
$$Q_{new}(1, \text{“}DOWN\text{”}) \leftarrow (1-\alpha)Q(1, \text{”}DOWN\text{”}) + \alpha(-1+\gamma\ -1)$$

Q-learning Example

- The most sensitive issue in Q-learning is the definition of “state”. What characterizes a state?
  - If we are too vague, and consider a broad definition of state, the algorithm will have difficulties to simulate an intelligent behavior.
    - E.g., in the previous example, consider simply the “type of cell” as state.
    - That will imply to have only three rows in the Q-Table, which will reduce the complexity.
    - However, the algorithm will produce the same action for all “blue cells”, “red cells”,…

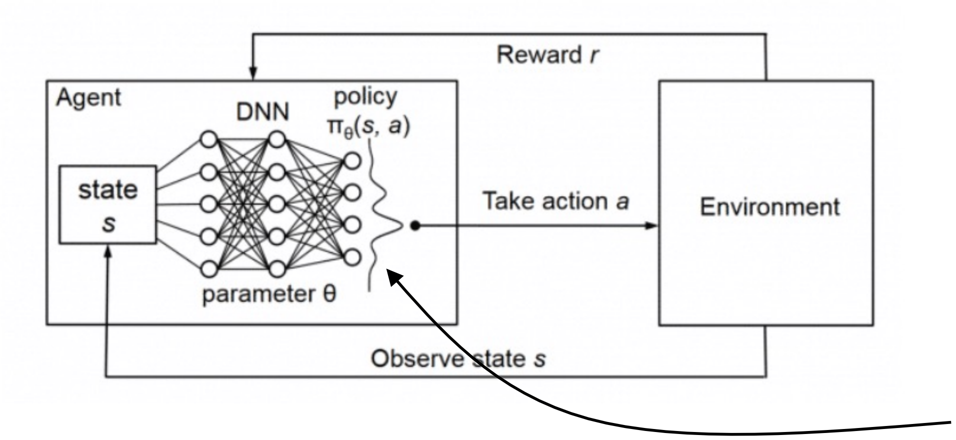


    - At the opposite side, we can consider each actual cell in the board as a state, which will yield 48 rows in the Q-Table and a 48x48 reward matrix.
    - What would happen in case of a board with 1,000,000 cells?
  - There are intermediate solutions for defining a state, such as the current position and a neighborhood of radius “r”.
    - Enables to infer the correct behavior in a more sophisticated way.
    - Augments the computational complexity (each of the free 8 positions can have 3 types, i.e., 3^8 = 6561 states



Deep Q-learning

- Q-learning is known to be able to attain decent results in problems with **small dimension** in terms of the "space of all states".
- The problem is that even relatively simple problems have an intractable number of possible states, where Q-learning cannot be applied.
- A possible solution is the **Deep Q-learning** algorithm, where the idea is to use a CNN to analyze the current state of the world and return the policy of actions.



The CNN will have as many neurons in the output layer as possible actions

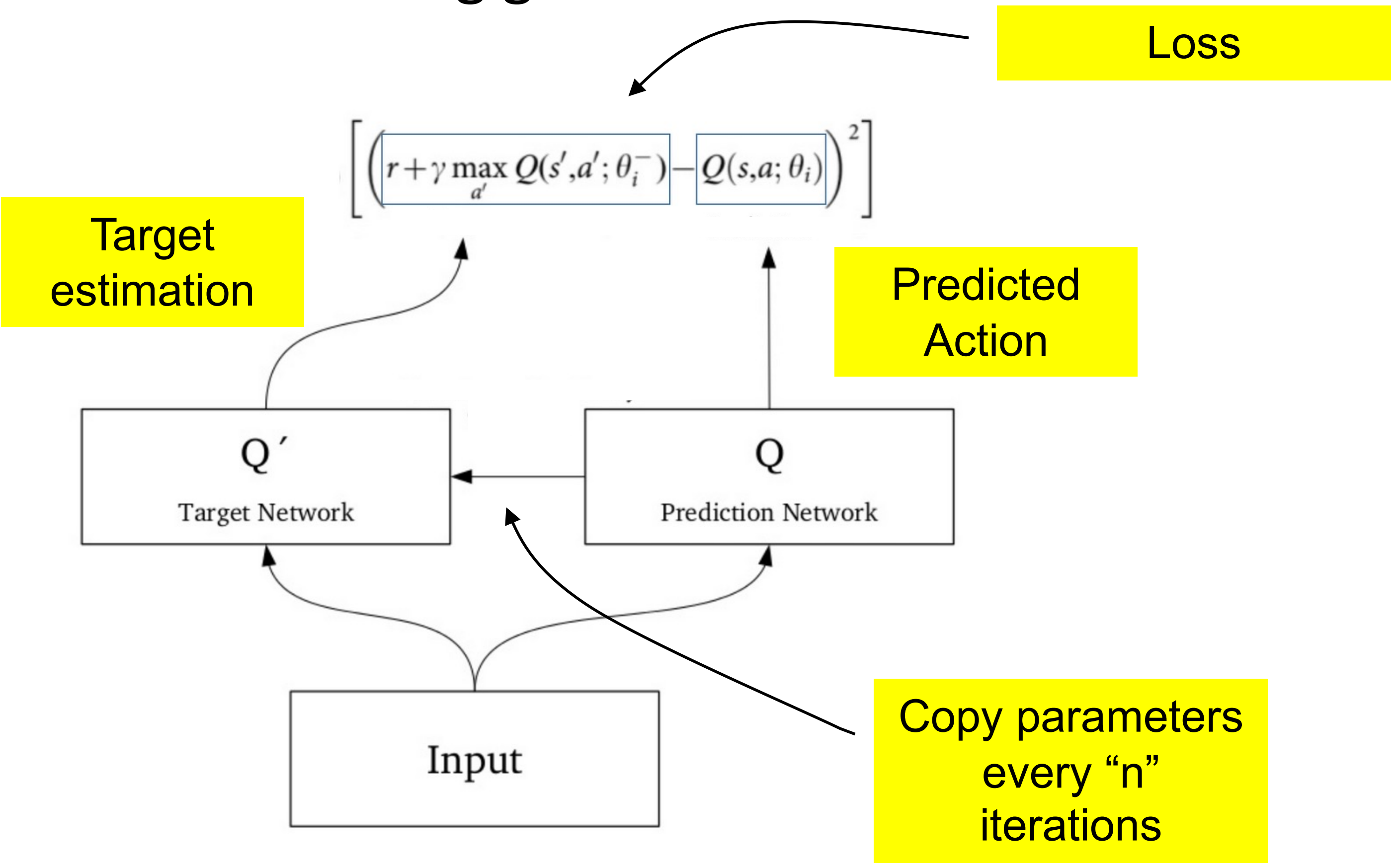
The optimal action corresponds to the most activated output neuron

Deep Q-learning

- In practice terms, the main difference between using a CNN for Reinforcement Learning purposes and the traditional way classification CNNs work is the fact in the latter models, the target variable does not change over the time.
  - For a specific instance (state), the ground truth (or desired label) is always the same.
- In the Reinforcement Learning setting, we depend on the policy or value functions to sample actions. However, this policy changes as we continuously learn what to explore. During the learning process, we start to know more about the ground truth values of the states/actions and hence, the desired output should also change accordingly.
- At the end, we are trying to learn a map for a constantly changing input/output. Is this feasible?

Deep Q-learning

- The solution is to use slightly different **two twin networks**, from slightly different learning generations.



Loss

$$\left[\left(\boxed{r+\gamma \max_{a'} Q(s',a';\theta_i^-)}-\boxed{Q(s,a;\theta_i)}\right)^2\right]$$

Target estimation

Predicted Action

Copy parameters every "n" iterations

Q-learning Practical Work: Labyrinths

- Create a Pyhton script that generates random “N x N” boards, with each cell having only two possible types: “**Safe**” (penalty=1) or “**Dangerous**” (penalty=100).
- Generate two random cell positions (each one provided in a (row, column) format), corresponding to the current position of the agent and to the final position.
- Using Reinforcement Learning techniques (Q-learning), design and implement a solution for moving from the current state to the final position in as few movements as possible, avoiding the Dangerous cells as much as possible.
- Compare different strategies for defining the “state” in terms of the spatial computational cost of the algorithm (the amount of storage required) and the quality of the solutions generated (the cost of the overall path, i.e., the sum of the penalties for the cells composing the best path).
- Perceive the variations in the results, with respect to the α, γ values used in the Q-Table update formula:

$$Q_{new}(s_t, at) \leftarrow (1 - \alpha)Q(s_t, at) + \ \alpha(Rt + \gamma\ maxa\ Q(S_{t\ +\ 1,}\ a))$$

- Prepare a small report, describing your main findings.