LABO IA ML: TP2

January 9, 2020

1 Installation

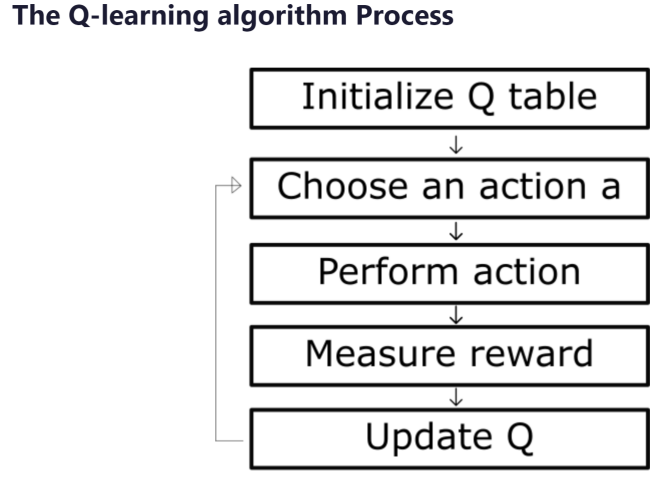
Clone the source code at : `https://github.com/HadarakDev/LabESGI-IA-ML/`
Then navigate to the folder $TP2\_Q\_Learning$. You will need *pygame* to run the code and complete the functions in the $fix\_me.py$

Install pygame in an existing env or new env.

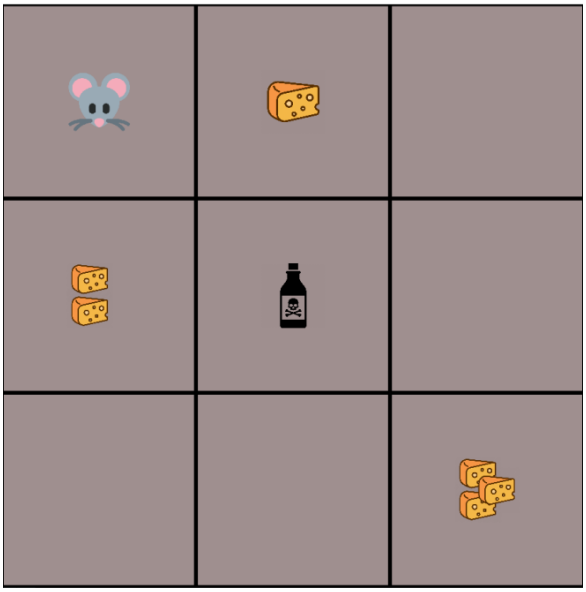
```
pip install pygame
```

2 The Q-learning algorithm process

In this tutorial, we will see the first reinforcement learning algorithm called Q-learning. The main objective of this algorithm is to list all the action possible of the agent and to compute it's rewards. The agent will try each actions at each step and will update the reward in a table.



In this exercise, we will make a mouse learn to eat all the cheese on a board without drinking the poison. Here is what the board looks like:



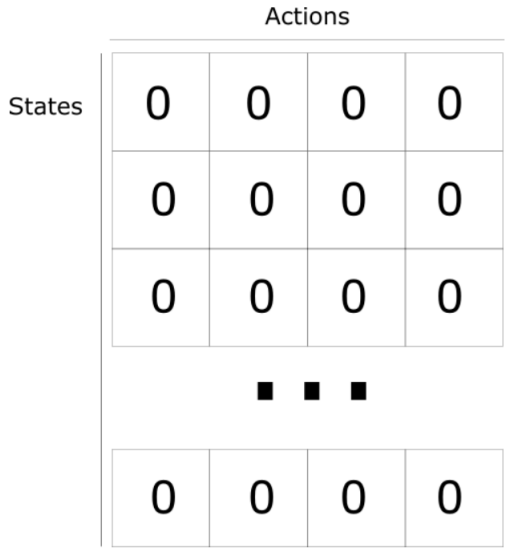
2.1 Initialize Q-values

The first steps is to create a table to store each potential reward of each actions at each states. Your objective here is to complete the *create_table*() function in the source code given.

How many states they are[1]? How many actions they are[2]?

[1] 9 cells on the board: 9 states

[2] the mouse can move, up, down, left, right: 4 actions



Here is what should look like your table after creation. An empty SxA matrix. S are the number of States and A the number of actions possible for the agent.

2.2 Staying Alive (is_alive())

You will have to train your mouse for a certain amount of epochs, and for each epochs the mouse will try to get the highest reward possible. The mouse stops the epochs when it has eaten the biggest cheese on the board. Eating the poison could stop the game too. You can choose to stop the game if you consider the mouse dead after eating the poison. Also the mouse will die after X actions.

You can edit the *is_alive*() function if you want to stop the game after the mouse eats the poison. Else it will stop the game when it eats the biggest cheese.

2.3 Choose an action

In this step, you will have to give an order to the mouse. Careful, you can kill it!

FIX the function *choose_action*().

How do you chose an action?

Choose an action a in the current state S based on the current Q-value estimates. Recall the Q-values are stored in your matrix. At the beginning the matrix is empty, so you will have to take a totally random action. And at each steps you will take less and less random actions.

- We specify an exploration rate "epsilon," which we set to 1 in the beginning. This is the rate of steps that we'll do randomly. In the beginning, this rate must be at its highest value, because we don't know anything about the values in Q-table. This means we need to do a lot of exploration, by randomly choosing our actions.
- We generate a random number. If this number ¿ epsilon, then we will do "exploitation" (this means we use what we already know to select the best action at each step). Else, we'll do exploration.
- The idea is that we must have a big epsilon at the beginning of the training of the Q-function. Then, reduce it progressively as the agent becomes more confident at estimating Q-values.
- If you take a non random action, you will have to take the action which gives you the best rewards. Return the index of the highest reward in the list of action on your state row in the matrix.

The list of the actions are the indexes of your Q-table. [0, 1, 2, 3].

2.4 Measure reward

Once you have you have your next action, you can use the function *step* to make the action, this function will give you the

```
x, y, stateNext, reward, done, score_map_edit = step(action, x, y,
                                    score_map_edit, item_map_edit)
```

You get the new position of the mouse. The *stateNext* is the state were the mouse gets after your action. The *reward*. The *done* is the variable that will say if the mouse is alive or dead. *score_map_edit* is the updated map matrix of the rewards.

2.4.1 Update the table

Once you have the reward, you can update the Q-table regarding the reward. Fix the function *update*().

Here is how you update the Q-table:

$$NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma max Q'(s',a') - Q(s,a)]$$

New Q value for that state and that action

Current Q value

Learning Rate

Reward for taking that action at that state

Discount rate

Maximum expected future reward **given the new s' and all possible actions at that new state**

```
New Q value = Current Q value + lr * [Reward + discount_rate * (
                                    highest Q value between possible
                                    actions from the new state s' ) -
                                     Current Q value ]
```

discount_rate is the importance you give to the highest Q value from the actual and the next state.

lr is the learning rate, it will affect the final new Q-value in the Q-table.