**Q-learning Robot**

* In this project we will implement the Q-Learning and Dyna-Q solutions to the reinforcement learning problem. We will apply them to a navigation problem in this project. In next step we will apply them to trading. Note that your Q-Learning code really shouldn't care which problem it is solving. The difference is that you need to wrap the learner in different code that frames the problem for the learner as necessary.
* The reinforcement learning methods are applied to optimize the portfolios with asset allocation between risky and riskless instruments in this paper. We use classic reinforcement algorithm, Q-learning.
* ***Q*-learning** is a [reinforcement learning](https://en.wikipedia.org/wiki/Reinforcement_learning) technique used in [machine learning](https://en.wikipedia.org/wiki/Machine_learning). The technique does not require a model of the environment. *Q*-learning can handle problems with stochastic transitions and rewards, without requiring adaptations.
* For any finite [Markov decision process](https://en.wikipedia.org/wiki/Markov_decision_process) (FMDP), *Q*-learning eventually finds an optimal policy, in the sense that the expected value of the total reward return over all successive steps, starting from the current state, is the maximum achievable. *Q*-learning can identify an optimal action-selection policy for any given FMDP
* "Q" names the function or equivalent that returns the reward used to provide the reinforcement and can be said to stand for the "quality" of an action taken in a given state.
* There’s how it works:

Imagine you have an agent that’s expected to take autonomous actions in a changing, unpredictable environment. Q-learning involves repeating the following 4 steps until the agent’s task is done:

1. Agent senses its environment, using this information to determine its current state
2. Agent takes an action and obtain a penalty or reward
3. Agent senses its environment again - to see what effect its chosen action had
4. Agent learns from its experience (and so makes ‘better’ decisions next time)

* Main Constructor we made in our task to implement Q-learning:
* **The constructor QLearner()** should reserve space for keeping track of Q[s, a] for the number of states and actions. It should initialize Q[] with all zeros. Details on the input arguments to the constructor
* **query(s\_prime, r)** is the core method of the Q-Learner. It should keep track of the last state s and the last action a, then use the new information s\_prime and r to update the Q table. The learning instance, or experience tuple is <s, a, s\_prime, r>. query() should return an integer, which is the next action to take. Note that it choose a random action with probability rar, and that it should update rar according to the decay rate radr at each step. Details on the arguments.
* **querysetstate(s)** A special version of the query method that sets the state to s, and returns an integer action according to the same rules as query() (including choosing a random action sometimes), but it does not execute an update to the Q-table. It also does not update rar. There are two main uses for this method: 1) To set the initial state, and 2) when using a learned policy, but not updating it.
* We will test our Q-Learner with a already defined navigation problem as follows.
* The navigation task takes place in a 10 x 10 grid world. The particular environment is expressed in a CSV file of integers, where the value in each position is interpreted as follows:
  1. 0: blank space
  2. 1:an obstacle
  3. 2:  the starting location for the robot.
  4. 3:  the goal location.
  5. 5: quicksand
* In this example the robot starts at the bottom center, and must navigate to the top left. Note that a wall of obstacles blocks its path, and there is some quicksand along the left side. The objective is for the robot to learn how to navigate from the starting location to the goal with the highest total reward. We define the reward for each step as:
  1. -1 if the robot moves to an empty or blank space, or attempts to move into a wall.
  2. 100 if the robot moves to a quicksand space.
  3. 1 if the robot moves to the goal space.
* Overall, we will assess the performance of a policy as the median reward it incurs to travel from the start to the goal (higher reward is better). We assess a learner in terms of the reward it converges to over a given number of training epochs (trips from start to goal).