Elias Asskali Assakali
ARTIFICIAL INTELLIGENCE Joel Lamata i Martin
Adam Dbouk Alberich

**Comments about exercise 1:**

First of all, we want to compute every iteration of our Value Iteration, where a Value Iteration takes a Markov decision process on initialization and runs value iteration for a given number of iterations using the supplied discount factor.

Secondly, we will compute the Q-value of action in state from the value stored.

And finally, we want the best policy for each state, where a policy is the best action in the given state according to the values currently stored.

**Comments about exercise 2:**

We know that:

**Discount** (0<disc<1):

GREATER: future reward more important,

LOWER: more important immediate reward.

**Noise (0<=noise<=1):**

GREATER: more uncertainty in next state

**Living reward:**

POSITIVE: each step has a reward (long path),

NEGATIVE: take the shortest path.

So, we want the agent to not end in an unintended state so the noise should be 0.

**Comments about exercise 3:**

a) Closest exit (negative living reward, low discount), risky path (no noise)
b) Close exit (low discount, negative living reward), avoid cliffs (some noise)
c) Distant exit (high discount, positive living reward), risky path (no noise)
d) Distant exit (high discount, positive living reward), avoid cliffs (some noise)
e) avoid exits (living reward higher than terminal nodes >1 and >10)

**Comments about exercise 4:**

To implement q-learning we have to apply the action that minimizes Q(a,s), we observe the resulting state s' and collect the cost c, then we update Q(a,s). We break ties choosing a random action between the Q(a,s) with the same value. In this case we don't implement epsilon greedy as this will be implemented in the next exercise.

Elias Asskali Assakali
Joel Lamata i Martin
Adam Dbouk Alberich

**Comments about exercise 5:**
In this case we will choose a random action with probability epsilon using flipCoin(epsilon). Otherwise, with probability 1-epsilon, we will choose the best policy iteration. If the state is a terminal state we return None.

**Comments about exercise 6:**
In this case, 50 iterations are not enough with any epsilon and learning rate so we would need many more iterations, so we return “NOT POSSIBLE”.

**Comments about exercise 7:**
As our code above works without exceptions and our agent wins at least 80% of the time, we don’t need to do anything in this question, as this question uses the qleaningAgent already implemented to find a solution for a pacman game.

**Comments about exercise 8:**
As we have the command “PacmanQAgent.__init__(self, **args)” we can inherit the functions we did before in order to solve this exercise.
GetQvalue: we need the vector of weight and the feature vector in order to get the Q value for the (state, action) doing a dot product.
Update function: we apply the formula received in theory where we need to get the max value for the next state, also we need the feature vector and now we can update directly the weights.