Question 1

Part 1.

Reinforcement learning is based on reward and punishment, meaning that the agent decides at each state to choose the next action based on the amount of reward it will receive for choosing that action. Sometimes the reward for an action is instant and sometimes it is delayed. Instant reward means how much reward is received as soon as an action is performed, which is the direct result of that action and moving on to the next state. Delayed reward means how much reward the agent will receive in the future by following this policy until it reaches the goal (optimal state).

Part 2.

In Reinforcement Learning, the terms "model-based" and "model-free" refers strictly as to whether, whilst during learning or acting, the agent uses predictions of the environment response. The agent can use a single prediction from the model of next reward and next state (a sample), or it can ask the model for the expected next reward, or the full distribution of next states and next rewards. These predictions can be provided entirely outside of the learning agent.
Model-based methods rely on planning as their primary component, while model-free methods primarily rely on learning. Although there are real differences between these two kinds of methods, there are also great similarities. In particular, The heart of both learning and planning methods is the estimation of value functions by backing-up update operations. The difference is that whereas planning uses simulated experience generated by a model, learning methods use real experience generated by the environment. Of course this di↵erence leads to a number of other di↵erences, for example, in how performance is assessed and in how flexibly experience can be generated. But the common structure means that many ideas and algorithms can be transferred between planning and learning. In particular, in many cases a learning algorithm can be substituted for the key update step of a planning method. Learning methods require only experience as input, and in many cases they can be applied to simulated experience just as well as to real experience.

Model-based reinforcement learning

- Pros:
  - Safe to plan exploration and can train from simulated experiences.
  - Faster to learn.
- Cons:
  - Agent only as good as the model learnt. Also, sometimes this becomes a bottleneck, as the model becomes surprisingly tricky to learn.
  - Computationally more complex than model-free methods.

Model-free reinforcement learning

- Pros :
  - Computationally less complex.
  - Occupies less space $\rightarrow$ usefull when state-space is large
  - Needs no accurate representation of the environment in order to be effective. This makes them more fundamental than model-based methods.
- Cons:
  - Actual experiences need to be gathered in order for training, which makes exploration more dangerous.
  - Cannot carry an explicit plan of how environmental dynamics affects the system, especially in response to an action previously taken.

> In summary, in problems where it is difficult to know the model of the environment or the state space is very large, the use of Model-free reinforcement learning is more effective.
> Also seen in high-dimensional problems, model-free learning has performed better and faster due to fewer parameters, while producing weaker and more inaccurate results.

Part 3.

- First iter. :

$$
\begin{aligned}
&V(1,1) = 0.2(0.6V(1,2) + 0.2V(1,1) + 0.2V(2,1)) = 0 \\
&V(1,2) = 0.2(0.6V(1,1) + 0.2V(2,2) + 0.2V(1,2)) = 0 \\
&V(1,3) = 0.2(3 + 0.2V(1,4) + 0.2(0.2V(1,2) + 0.6V(2,3))) = 0.59 \\
&V(1,4) = 3 + 0.2V(1,4) = 3.75 \\
\\
&V(2,1) = 0.2(0.6V(2,1) + 0.2V(3,1) + 0.2V(1,1)) = 0 \\
&V(2,2) = 0.2(0.6V(2,1) + 0.2V(2,2) + 0.2V(1,2)) = 0 \\
&V(2,3) = 0.2(0.2V(3,3) + 0.6V(1,3)) + 0.6(-2 + 0.2V(2,4)) = -1.48 \\
&V(2,4) = -2 + 0.2V(2,4) = -2.5 \\
\\
&V(3,1) = 0.2(0.8V(3,1) + 0.2V(2,1) = 0 \\
&V(3,3) = 0.2(0.6V(2,3) + 0.2V(3,3)) + 0.2V(3,4)) = -0.31 \\
&V(3,4) = 0.6(-2 + 0.2V(2,4)) + 0.2(0.2V(3,4) + 0.2V(1,3)) = -1.57
\end{aligned}
$$

- Second iter.:

$$
\begin{aligned}
&V(1,1) = 0.2(0.6V(1,2) + 0.2V(1,1) + 0.2V(2,1)) = 0.03 \\
&V(1,2) = 0.2(0.6V(1,1) + 0.2V(2,2) + 0.2V(1,2)) = 0.29 \\
&V(1,3) = 0.2(3 + 0.2V(1,4) + 0.2(0.2V(1,2) + 0.6V(2,3))) = 2.34 \\
&V(1,4) = 3 + 0.2V(1,4) = 3.75 \\
\\
&V(2,1) = 0.2(0.6V(2,1) + 0.2V(3,1) + 0.2V(1,1)) = 0 \\
&V(2,2) = 0.2(0.6V(2,1) + 0.2V(2,2) + 0.2V(1,2)) = 0.01 \\
&V(2,3) = 0.2(0.2V(3,3) + 0.6V(1,3)) + 0.6(-2 + 0.2V(2,4)) = 0.09 \\
&V(2,4) = -2 + 0.2V(2,4) = -2.5 \\
\\
&V(3,1) = 0.2(0.8V(3,1) + 0.2V(2,1) = 0 \\
&V(3,3) = 0.2(0.6V(2,3) + 0.2V(3,3)) + 0.2V(3,4)) = 0 \\
&V(3,4) = 0.6(-2 + 0.2V(2,4)) + 0.2(0.2V(3,4) + 0.2V(1,3)) = -0.52
\end{aligned}
$$

- Third iter.:

$V(1,1) = 0.2(0.6V(1,2) + 0.2V(1,1) + 0.2V(2,1)) = 0.03$
$V(1,2) = 0.2(0.6V(1,1) + 0.2V(2,2) + 0.2V(1,2)) = 0.29$
$V(1,3) = 0.2(3 + 0.2V(1,4) + 0.2(0.2V(1,2) + 0.6V(2,3))) = 2.3$
$V(1,4) = 3 + 0.2V(1,4) = 3.75$

$V(2,1) = 0.2(0.6V(2,1) + 0.2V(3,1) + 0.2V(1,1)) = 0$
$V(2,2) = 0.2(0.6V(2,1) + 0.2V(2,2) + 0.2V(1,2)) = 0.02$
$V(2,3) = 0.2(0.2V(3,3) + 0.6V(1,3)) + 0.6(-2 + 0.2V(2,4)) = -0.22$
$V(2,4) = -2 + 0.2V(2,4) = -2.5$

$V(3,1) = 0.2(0.8V(3,1) + 0.2V(2,1) = 0$
$V(3,3) = 0.2(0.6V(2,3) + 0.2V(3,3)) + 0.2V(3,4)) = -0.01$
$V(3,4) = 0.6(-2 + 0.2V(2,4)) + 0.2(0.2V(3,4) + 0.2V(1,3)) = 0$

→	→	→	3
↑	↑	↑	-2
↑		→	↓

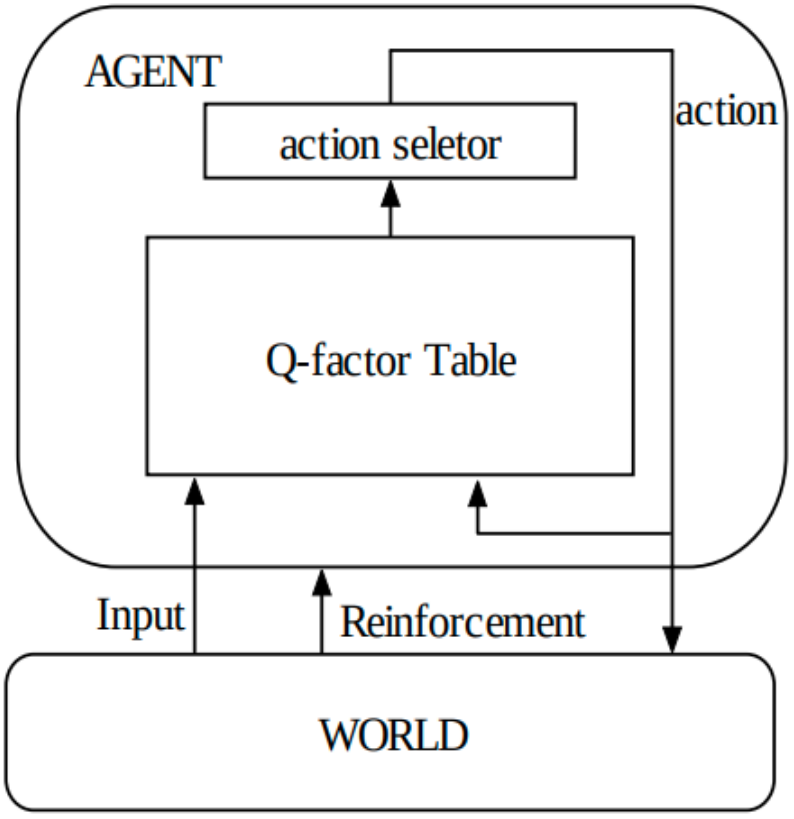
Question 2

Q-learning is a form of model-free reinforcement learning (i.e. agent does not need an internal model of environment to work with it). Since Q-learning is an active reinforcement technique, it generates and improves the agent's policy on thefly. The Q-learning algorithm works by estimating the values of state-action pairs.

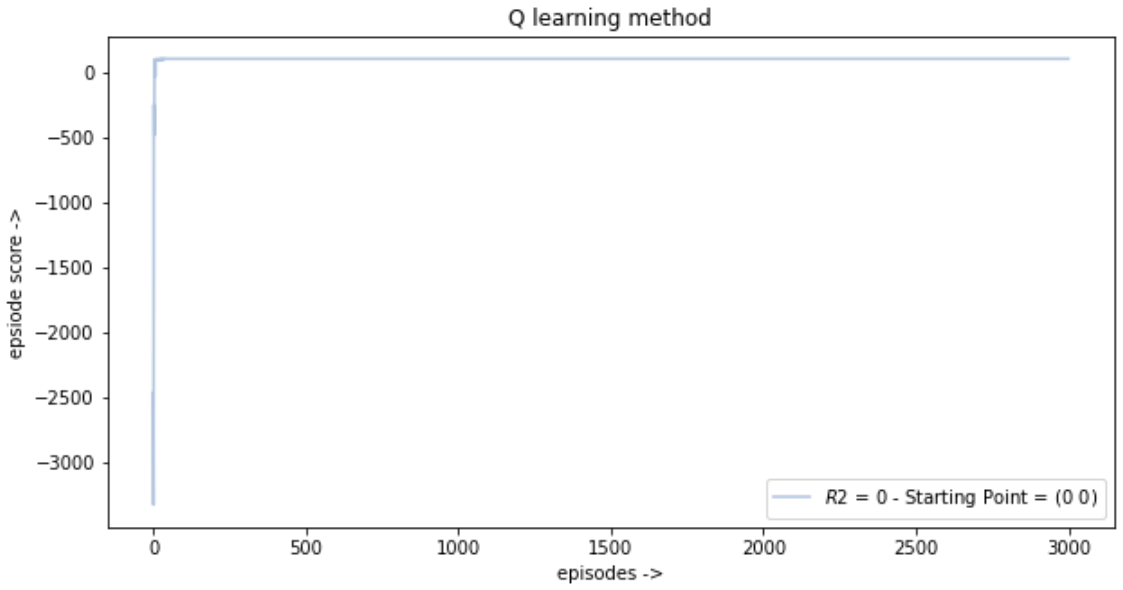
The purpose of Q-learning is to generate the Q-table, Q(s,a), which uses state-action pairs to index a Q-value, or expected utility of that pair. The Q-value is defined as the expected discounted future reward of taking action a in state s, assuming the agent continues to follow the optimal policy. For everypossible state, every possible action is assigned a value which is a function of both the immediate reward for taking that action and the expected reward in the future based on the new state that is the result of taking that action. This is expressed by the one-step Q-update equation

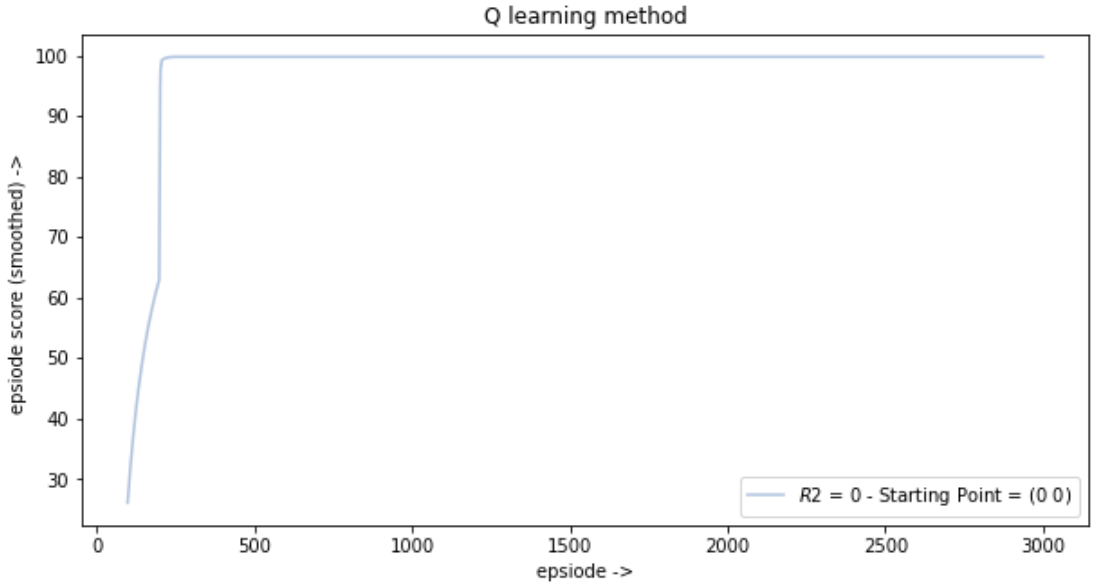
$$Q(a,s) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$$

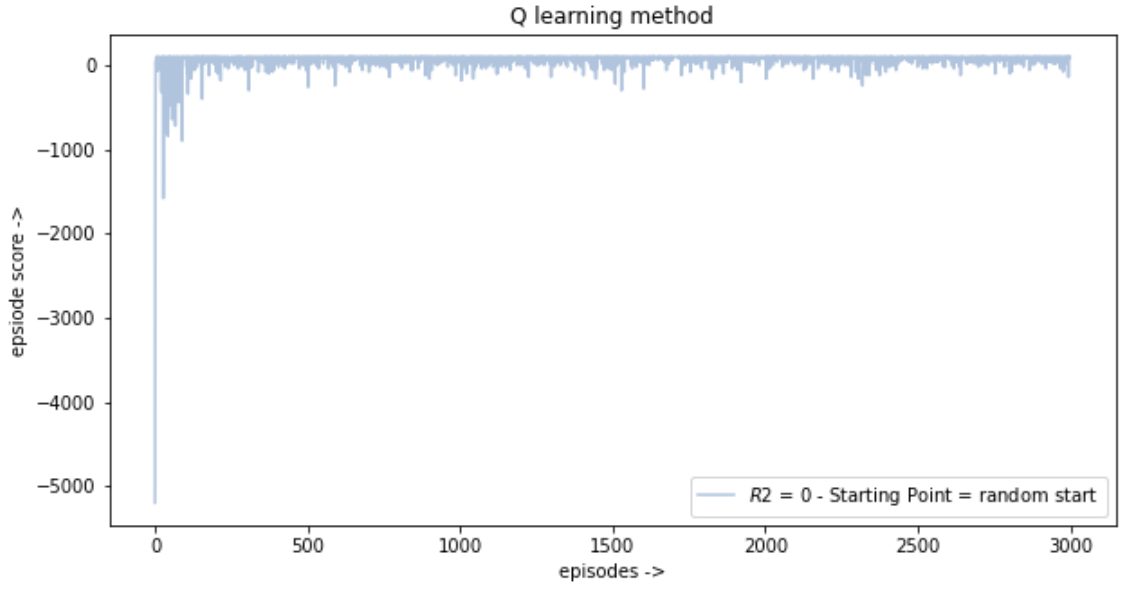
Where α is the learning factor and γ is the discount factor.These values are positive decimals less than 1 and are set through experimentation to affect the rate at which the agent attempts to learn the environment. The variables s and a represent the current state and action of the agent, r is the reward from performing s' and a', the previous state and action, respectively. The discount factor makes rewards earned earlier more valuable than those received later. This method learns the values of all actions, rather than just finding the optimal policy. This knowledge is expensive in terms of the amount of information that has to be stored, but it does bring benefits. Qlearning is exploration insensitive, any action can be carried out at any time and information is gained from this experience.

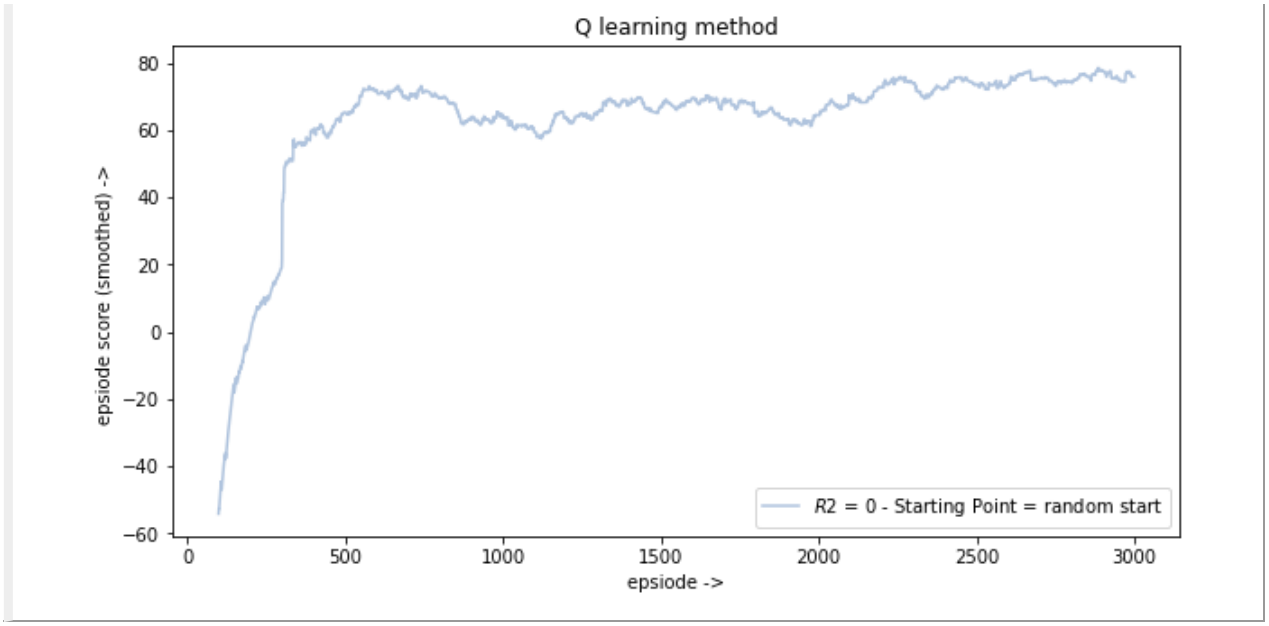


Note : I tried both *random start* and starting only from *starting point i.e. (0 0)* for $R_2 = 0$.˙ also drew smoothed cumulative reward(window size = 100) to have better visualization .



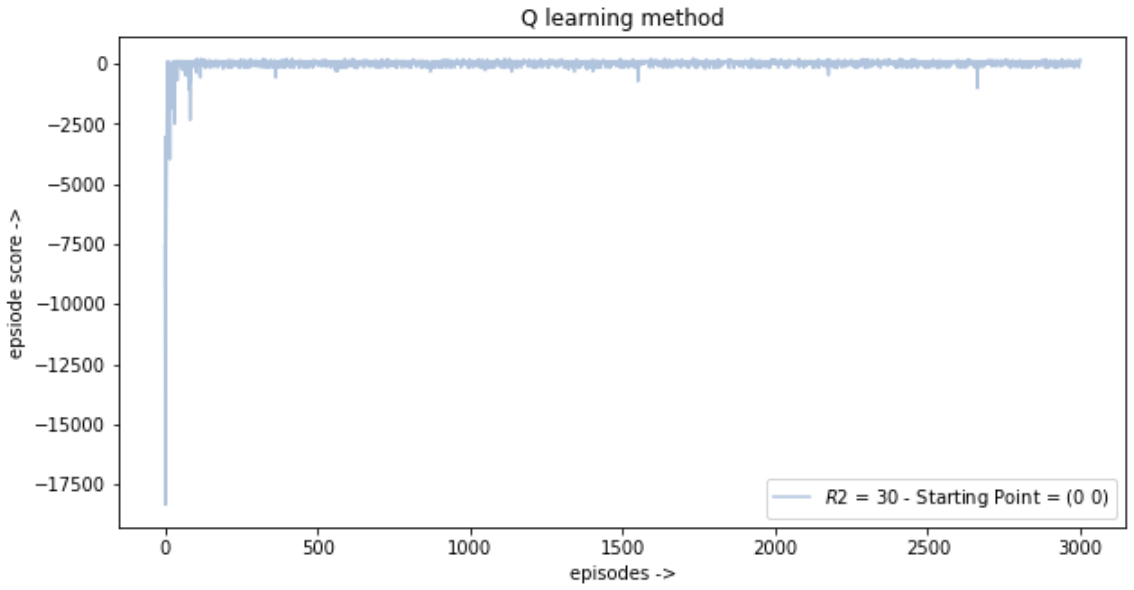


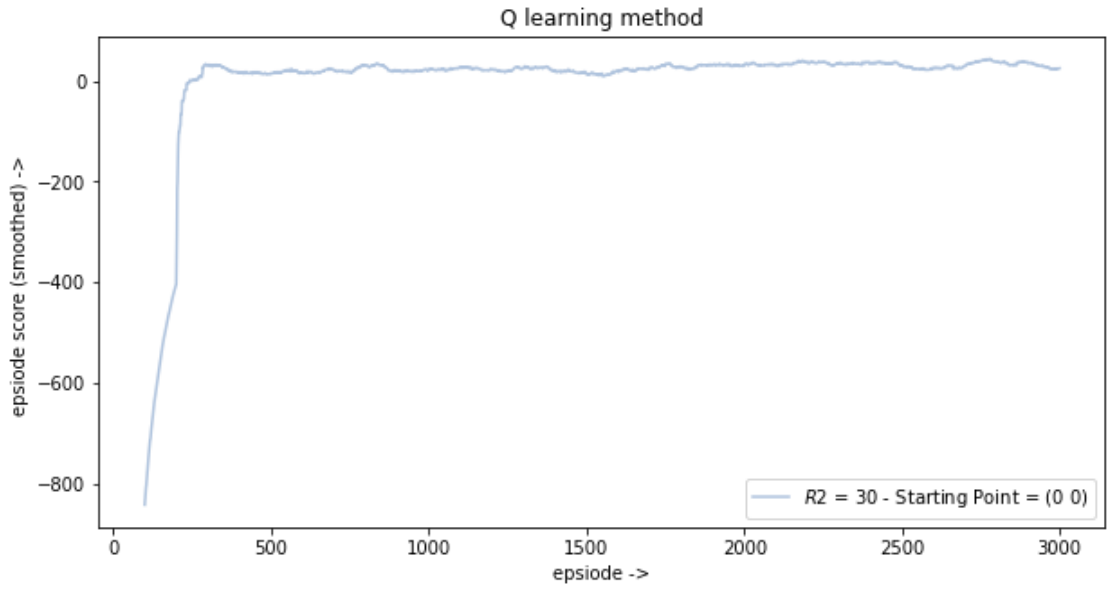


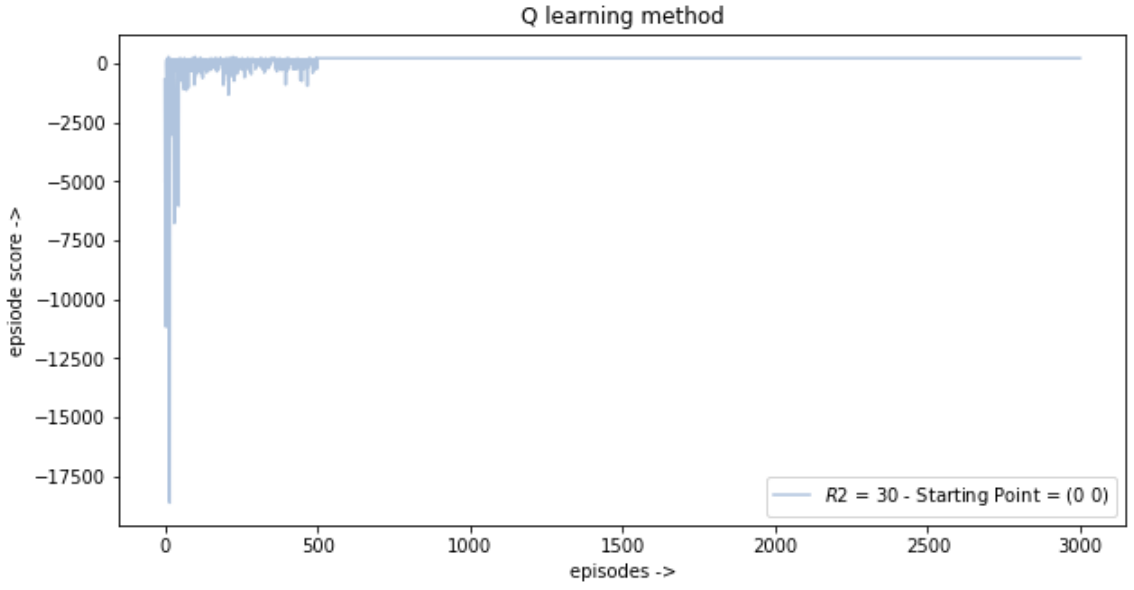


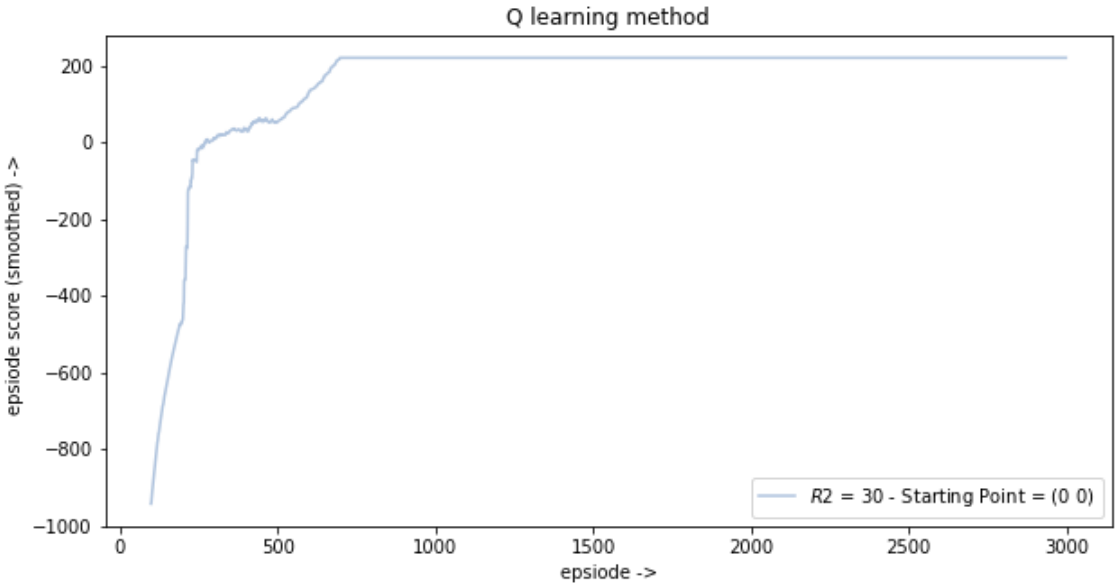
As expected, cumulative reward : $100 - \#steps \times 0.01 \approx 100$

Note :To have better exploration i used exploring start in the first 500 episodes, also to have better and faster convergance, the agent will be punished if it visits one state twice in each episdoe for $R_2 = 30$. i also drew smoothed cumulative reward to have better visualization .





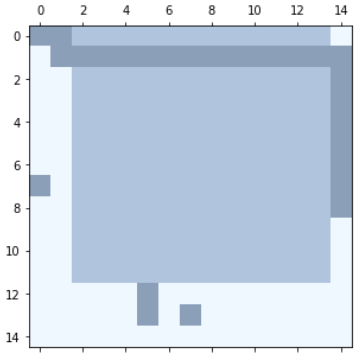




As expected, cumulative reward : $4 \times 30 + 100 - t\#steps \times 0.01 \approx 220$

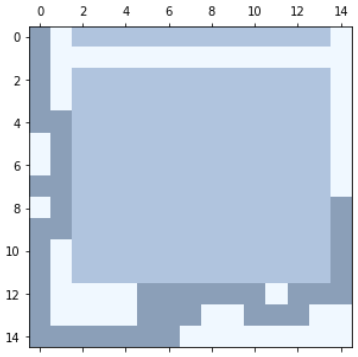
Part 2.

```
R2 = 0 - Starting Point = (0 0)
Path taken :
[0 1]->[1 1]->[1 2]->[1 3]->[1 4]->[1 5]->[1 6]->[1 7]->[1 8]->[1 9]->[ 1 10]->[ 1 11]->[ 1 12]->[ 1 13]->[ 1 14]->[ 2 14]->[ 3 14]->[ 4 14]->[ 5 14]->[ 6 14]->[ 7 14]->[ 8 14]
```



```
R2 = 30 - Starting Point = (0 0)
Path taken :
[0 0]->[1 0]->[2 0]->[3 0]->[4 0]->[4 1]->[5 1]->[6 1]->[7 1]->[8 1]->[9 1]->[9 0]->[10  0]->[11  0]->[12  0]->[13  0]->[14  0]->[14  1]->[14  2]->[14  3]->[14  4]->[14  5]->[14  6]->[13  6]->

[12  6]->[12  7]->[12  8]->[12  9]->[12 10]->[13 10]->[13 11]->[13 12]->[12 12]->[12 13]->[12 14]->[11 14]->[10 14]->[ 9 14]->[ 8 14]
```



Part 3.

Although ε-greedy action selection is an effective and popular means of balancing exploration and exploitation in reinforcement learning, one drawback is that when it explores it chooses equally among all actions. This means that it is as likely to choose the worst-appearing action as it is to choose the next-to-best action. In tasks where the worst actions are very bad, this may be unsatisfactory. The obvious solution is to vary the action probabilities as a graded function of estimated value. The greedy action is still given the highest selection probability, but all the others are ranked and weighted according to their value estimates. These are called softmax action selection rules. The most common softmax method uses a Gibbs, or Boltzmann, distribution. It chooses action on the th play with probability.
Whether softmax action selection or ε-greedy action selection is better is unclear and may depend on the task and on human factors. Both methods have only one parameter that must be set.

- ε-greedy policy:

$$action\ at\ time\ t: \begin{cases} maxQ_t(a) & with\ probability\ 1-\varepsilon \\ anyaction & with\ probability\ \varepsilon \end{cases}$$

- softmax policy :

$$\frac{e^{Q_t(a)}}{\sum_{b\epsilon allactions}\ e^{Q_t(b)}}$$