Neural Network and deep learning: homework 5

of Tommaso Tabarelli

Abstract

In this homework we are asked to modify a given python script in order to train an agent moving in a given environment (which we can set arbitrarily) using reinforcement learning algorithm. The goal is to learn the main features of the learning process, trying to change the environment, comparing the Q-learning and SARSA methods performances.

Code understanding

First of all we had to undestand how the code works and how the *Agent* and the *Environment* models are implemented.

Code development

My choice was to work with jupyter notebook since I find it more familiar and cleaner; anyway a python script was finally developed as homework requests.

After having understood the code, some obstacles were added to see if the agent could learn how to handle them. After this, some different choices of the environment were made in order to try to *trick* the algorithms; *Q-learning* instead outsmarted the difficulties very well. Moreover, in the case of SARSA, a *softmax* function was added to compare the results with the situation without it. As next step, different functions for the choice of the ε parameter (which in general depends on the *episode* number) were tried. Furthermore, as final step, the *Q-learning* algorithm was tested in a more complicated environment with "increasing" difficulties.

The obstacles were chosen to be of two types:

- *inter-positional walls*, thus "line walls" blocking the acces to a position in the grid only from another position (they do not occupy a position in the grid; see blue lines in Fig.6 to better understand this statement);
- "sand", which occupies a position in the grid (or, in general, more than one) and gives a small negative reward (-0.01, arbitrary choice) when passing over it.

The walls were described by two positions, thus four coordinates, representing the two positions the wall is blocking the passage between. They are stored in the *Environment* class (they are "duplicated in a symmetric way" to be handled better, because the two positions to check the passage between can be passed in each order and of course the passage is blocked in both directions: for example, if a wall is described by [[1,1],[1,2]], then also [[1,2],[1,1]] will be added in the environment variable). When trying to moving through a wall, the agent will instead stand still and its reward (for that move) will be 0, making it learn that approaching a wall is waste of time (the *time* still goes on in the sense that the discount of future rewards is greater, making them lower).

The "sand" is instead described by a single position and it does not block the passage of the agent but it makes it gain a negative reward. In this way the agent should learn that passing on the sand is "bad".

To be sure that the agent would start near the wanted environment window the choice of the initial section was limited to be sure that the agent would explore and train "against" the obstacles in the environment (if the agent for example starts in a position in which there are no walls between it and its goal position, he would not learn to avoid walls and its behaviour would be less interesting).

For what concerns the comparison between the *Q-learning* and the *SARSA* methods, they were made on four environments using nothing, sand, both walls and sand. Results and comments are reported below. A brief check was also made using the softmax function to create a distribution for the actions and then chosing them sampling from it.

To go on with the analysis, the function to change the ε parameter during the episodes was changed. The three possible functions were chosen to be:

- linar (the "standard" one, which was already provided): the function makes ε linearly decrease from the value of 0.8 to 0.001 during the episodes;
- staircase function: manually implemented, it was chosen to change five times uniformly during the episodes development, assuming values 1.0, 0.75, 0.5, 0.25, 0.001;
- cut-hyperbole: the function starts flat around a conventional value and then decreases hyperbolically with the number of episodes.

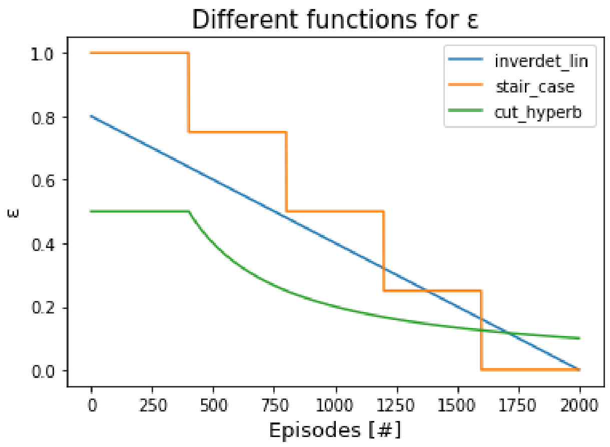

Figure 1: Different choices for "ε" function.

The agent was re-trained every time something in the environment or in the algorithms were changed, indeed it has to train and learn a specific parameters setting every time this one changes. Some parameters were not explored:

- the discount factor $\delta = 0.9$ was not changed because it was preferred to modify the obstacles parameters: walls have reward 0 and they keep the agent from moving, thus they are "not affected by the discount" factor (even if in this case the whole learning process is, indeed a different discount factor would lead to different weighing for the waste of time due to approaching the wall); sand instead is also "directly" affected by the discount factor, but in the code the preference was to tune its negative reward (which was fixed to -0.01: it was an arbitrary choice).
- the number of episodes was left to be 2000: since the initial results were "good" and the time spent for training was "sufficiently small" (around one or two minutes for each train process) it was decided to left it unmodified;

- the episode length was also left unmodified to 50 because the grid was chosen to be 10×10 and that number of "moves" was considered sufficient for a good training (also having run the code some times and having seen the results were acceptable).

In the python script the only last result is reported: when launching the script, the Agent trains in the last environment (Fig.17) using Q-learning algorithm, $\delta = 0.9$, linear function for ε.

Results

Starting with the comparison between Q-learning and SARSA algorithms in an empty environment (Fig.2, Fig.3), we can see that there is a slight difference in the Agents behaviours: knowing that SARSA tends to be more conservative, we can see that in general the SARSA path tends to stay farer from the walls with respect to the Q-learning path (anyway, this is a *biased* guess since we were told that SARSA method was more conservative: if we did not know it, that kind of guess is hard looking only to the reported environments).

Adding some sand to the environment (Fig.4, Fig.5), we can notice that SARSA algorithm is not able to learn properly this kind of environment. This may be caused by the unsufficient number of episodes, or because the algorithm gets stucked in a local extreme during the update, or by the fact that the sand is placed near the wall (in the upper section of the environment) thus while the Q-learning algorithm learns the "correct path" properly, the SARSA algorithm gets "confused" and can not evaluate the task properly.

For what concerns Q-learning algorithm, when adding walls the Agent simply learns how to bypass them (Fig.6). When adding the sand, the Agent tries to not go on it due to the negative reward. However, if the sand has a sufficiently small negative reward and it is placed near the goal, the Agent usually learns that it is convenient to approach the sand instead of outflanking it (Fig.7); indeed the sum of *goal positive reward* and *sand negative reward* can result in a better *total reward* if compared with the sum of the *goal positive reward* and *0s* (in these cases the "standard path" with no sand gives reward 0 at every step) that the Agent gains when outflanking the sand; this is caused by the increased discount due to the larger number of steps used to approach the goal in the second case.

Using the same argument, one can see that, when "sands appear in the same conditions" with respect to the goal and the approaching point, the Agent prefers to approach it when it is nearer to the goal (see Fig.7 in which a "symmetric" sand in the corridor has been placed to see if the Agent bypass all of it).

When comparing Q-learning and SARSA approaches on more complicated environments, we can see that Q-learning is usually better and that SARSA often seems to choose not optimally paths (compare paths in Fig.7 and Fig.9, and those in Fig.8 and Fig.10). This can have many causes, for example the algorithms generally performs in a different way, or maybe SARSA needs more episodes to reach "reasonable" results. Another result to notice is also that the SARSA method using softmax seems to obtain better results if compared with the method witout softmax (Fig.10 vs Fig.11).

Focusing on the different functions for ε, it seems that the *linear* (Fig.7) and the *staircase* ones (Fig.12) generally perform better with respect to the *cut-hyperbolic* one (Fig.13); this may be due to the fact that the last one has the final values of ε not close to 0, keeping the exploration present even when the "correct path/methodology" is instead already found.

To try to make more advanced tests on Q-learning algorithm, a new *ad hoc* environment was ideated and tested (Fig.14-17). It was chosen to have two symmetric corridors with respect to the starting position (which was chosen to be fixed in the upper-right corner); different situations were tested:

- in the first one (Fig.14), one of the corridors was chosen to have sand while the other no. In this situation the algorithm finds out that it is more convenient to pass the corridor without sand;
- in the second situation (Fig.15), sand was added to try to make the environment a bit more complicated and to see if the agent makes a coherent path choice compared with the previous case;
- in the third situation (Fig.16), more sand was added to build a sort of corridor in the center of the environment to see if the Agent would pass in the "gate" without sand or if it will find to be better to cut on the sand (since the sand is also near to the goal, the aforementioned argument about cutting on sand is still valid); in this case, it finds out that the sand is convenient;
- given the results in the previous point, more sand was added along the path chosen by the Agent in the previous situation (Fig.16 and Fig.17) to see if the agent will "react" in a proper manner; the *Q-learning* algorithm operates well and finds out that cutting before is more convenient than outflanking the sand (notice that the reward of this new path is not equivalent to the previous one, it is indeed lower since the sand is reached in one less step, thus the discount factor is higher and the negative payoff is larger).

Appendix

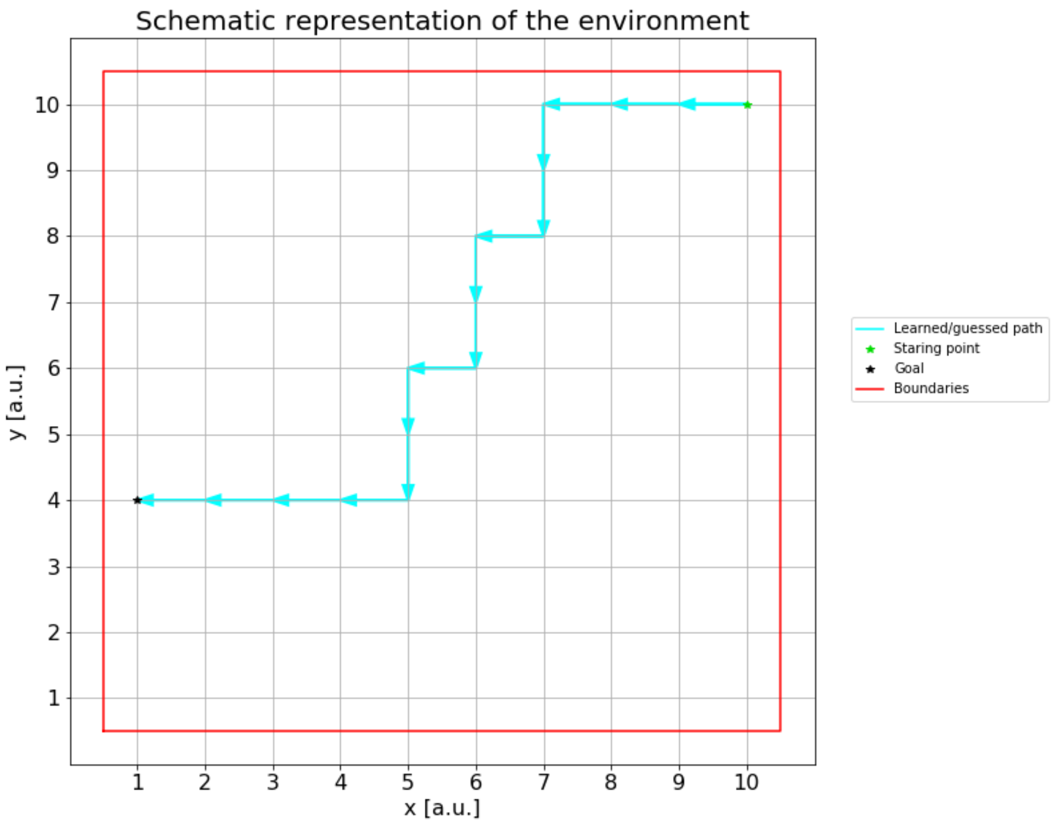

Figure 2: Environment without obstacles. Algorithm: Q-learning. $\delta = 0.9$. *ε-function: linear (given). Episode: 2000*

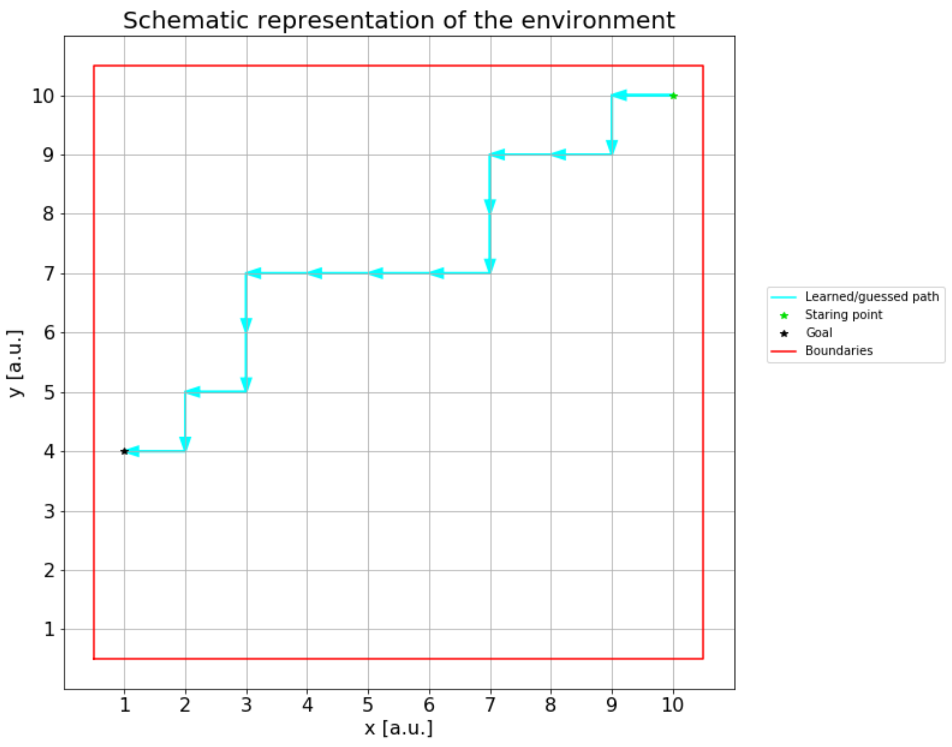

Figure 3: Environment without obstacles. Algorithm: SARSA. $\delta = 0.9$. *ε-function: linear (given). Episode: 2000*

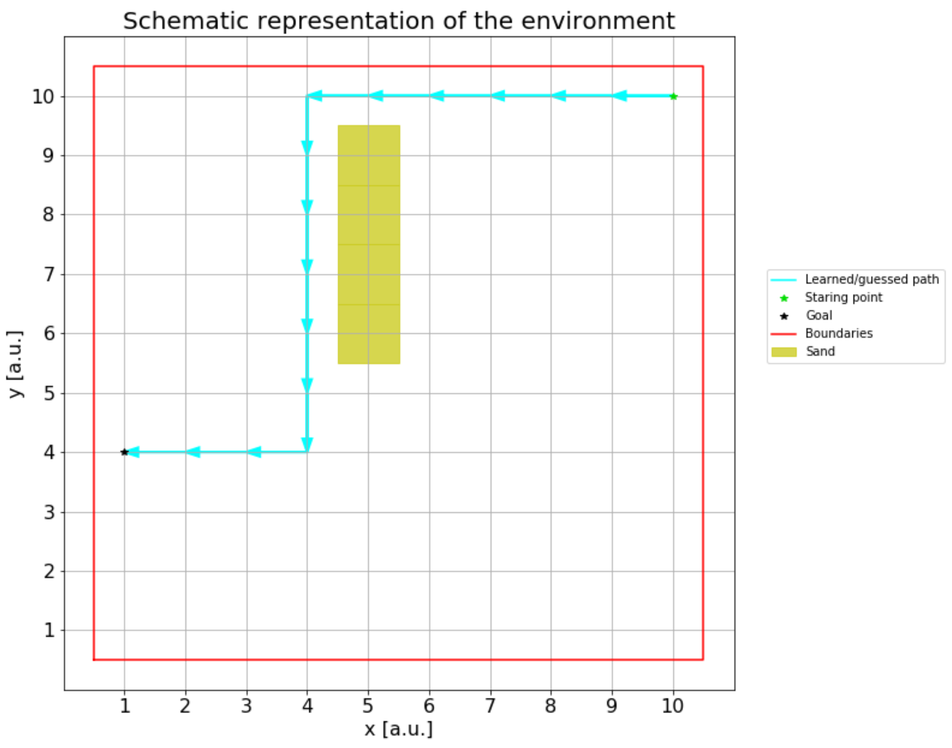

Figure 4: Environment with some sand. Algorithm: Q-learning. $\delta = 0.9$. *ε-function: linear (given). Episode: 2000*

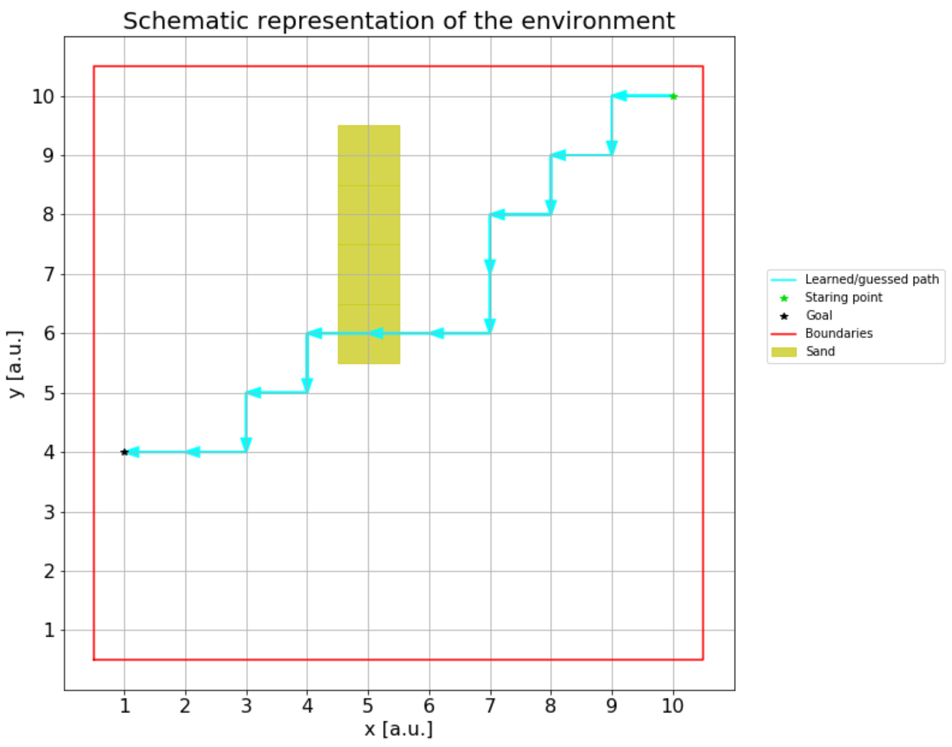

Figure 5: Environment with some sand. Algorithm: SARSA. $\delta = 0.9$. ε-function: linear (given). Episode: 2000

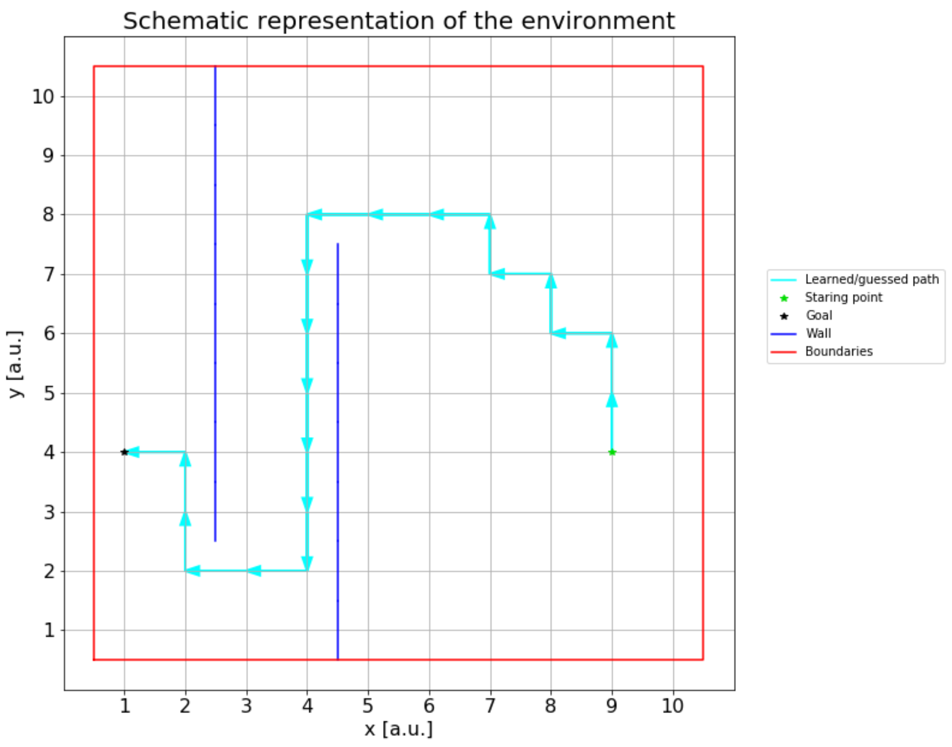

Figure 6: First environment, without sand. Algorithm: Q-learning. $\delta = 0.9$. ε-function: linear (given). Episode: 2000

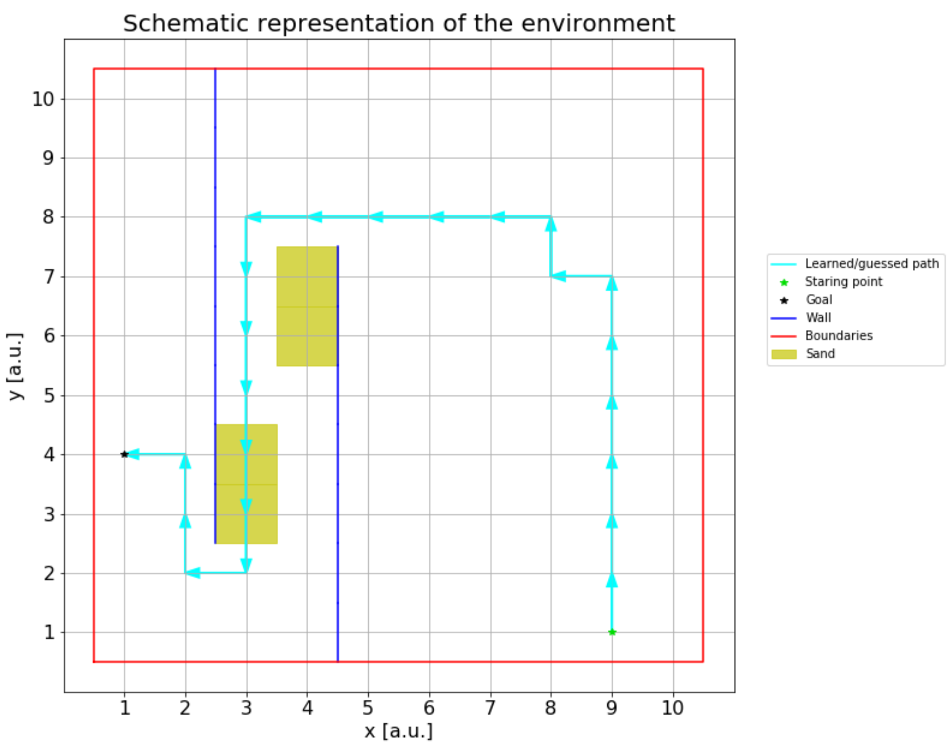

Figure 7: First environment, with first sand disposition. Algorithm: Q-learning. $\delta = 0.9$. *ε-function: linear (given). Episode: 2000*

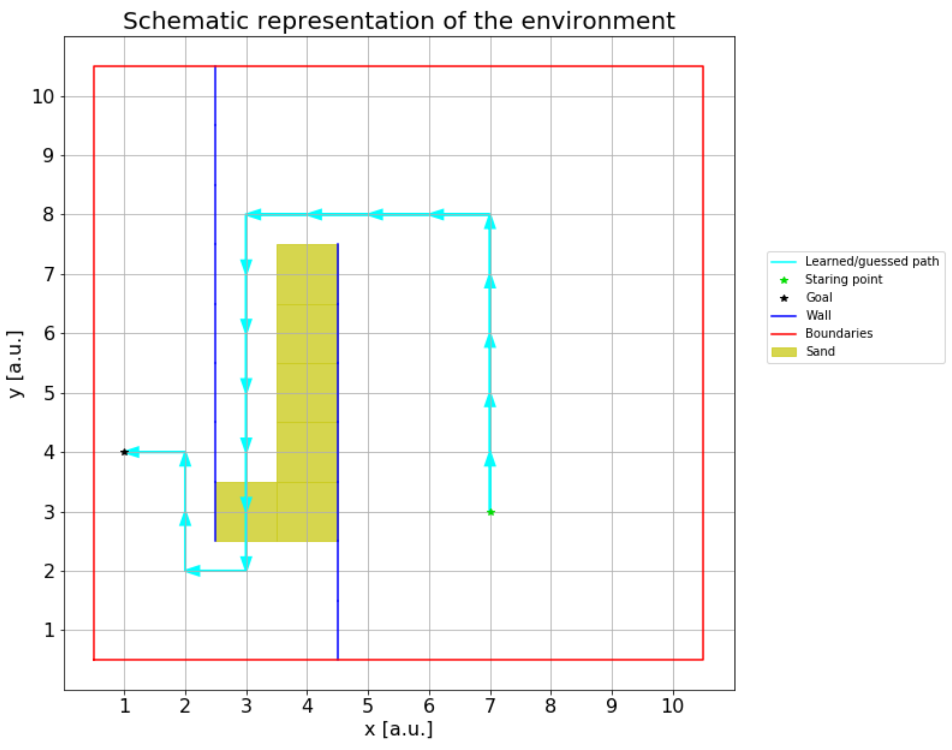

Figure 8: First environment, with second sand disposition. Algorithm: Q-learning. $\delta = 0.9$. *ε-function: linear (given). Episode: 2000*

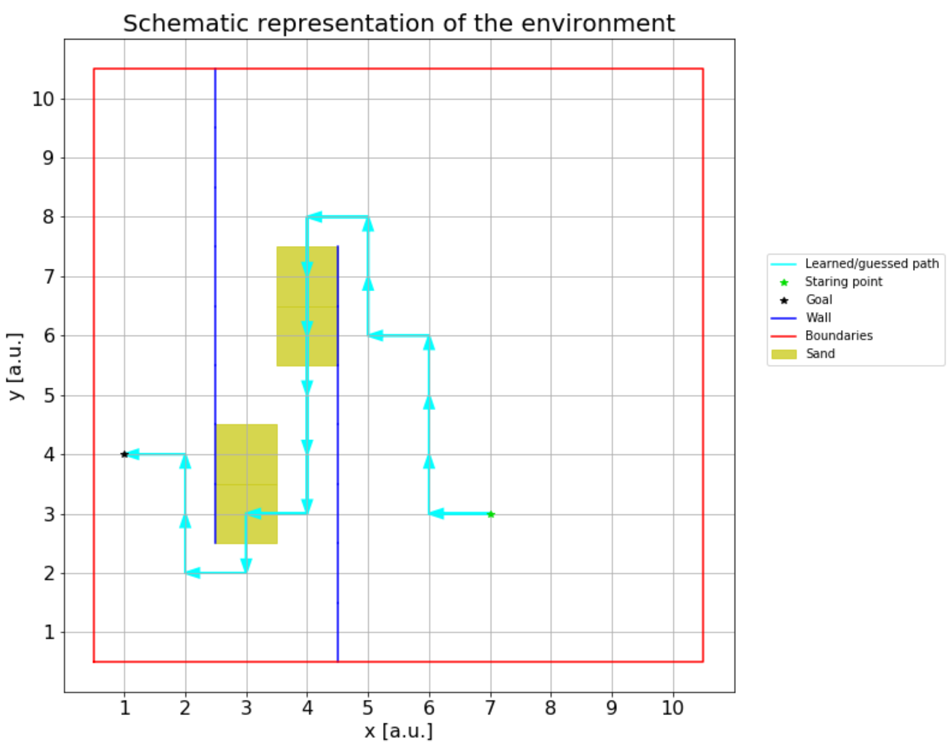

Figure 9: First environment, with first sand disposition. Algorithm: SARSA. $\delta = 0.9$. ε-function: linear (given). Episode: 2000

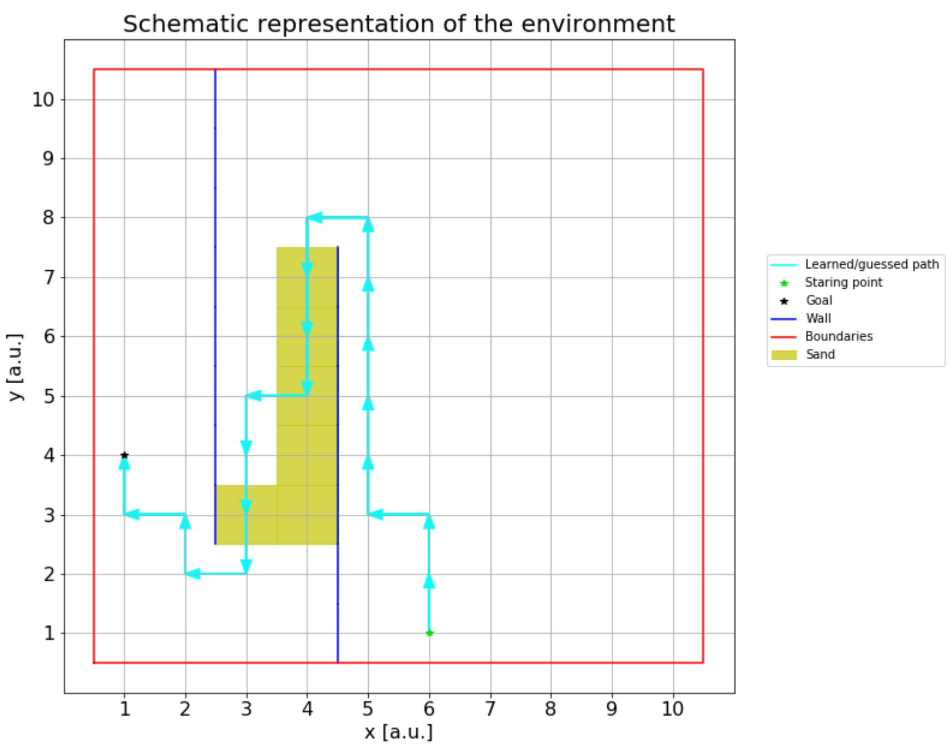

Figure 10: First environment, with second sand disposition. Algorithm: SARSA. $\delta = 0.9$. ε-function: linear (given). Episode: 2000

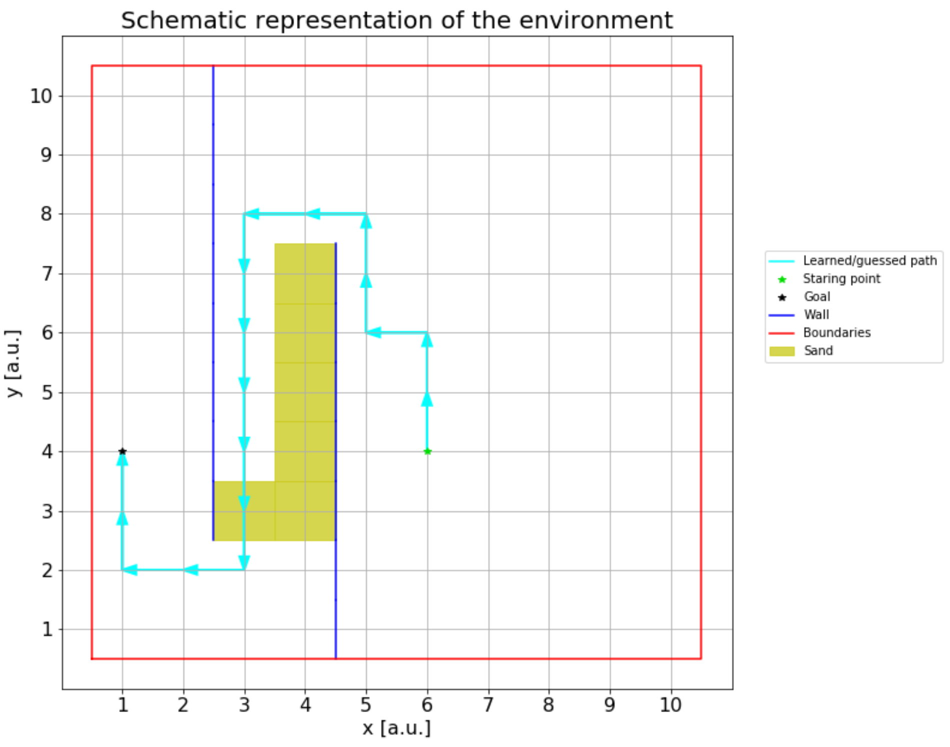

Figure 11: First environment, with second sand disposition. Algorithm: SARSA with softmax. $\delta = 0.9$. ε-*function: linear (given). Episode: 2000*

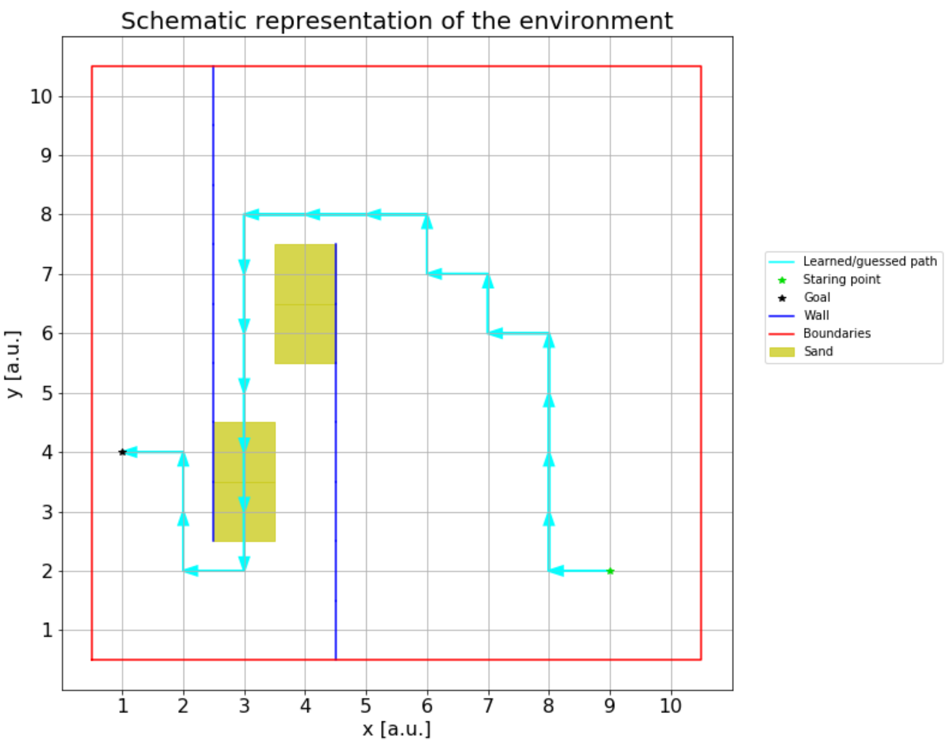

Figure 12: First environment, with first sand disposition. Algorithm: Q-learning. $\delta = 0.9$. ε-*function: staircase. Episode: 2000*

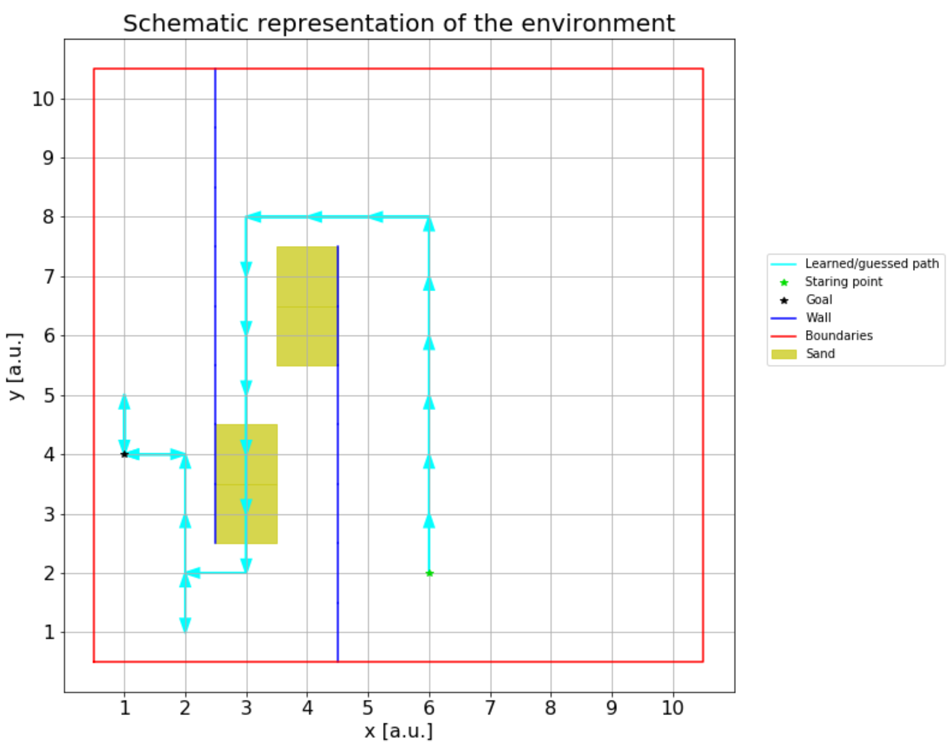

Figure 13: First environment, with first sand disposition. Algorithm: Q-learning. $\delta = 0.9$. *ε-function: cut-hyperbole. Episode: 2000*

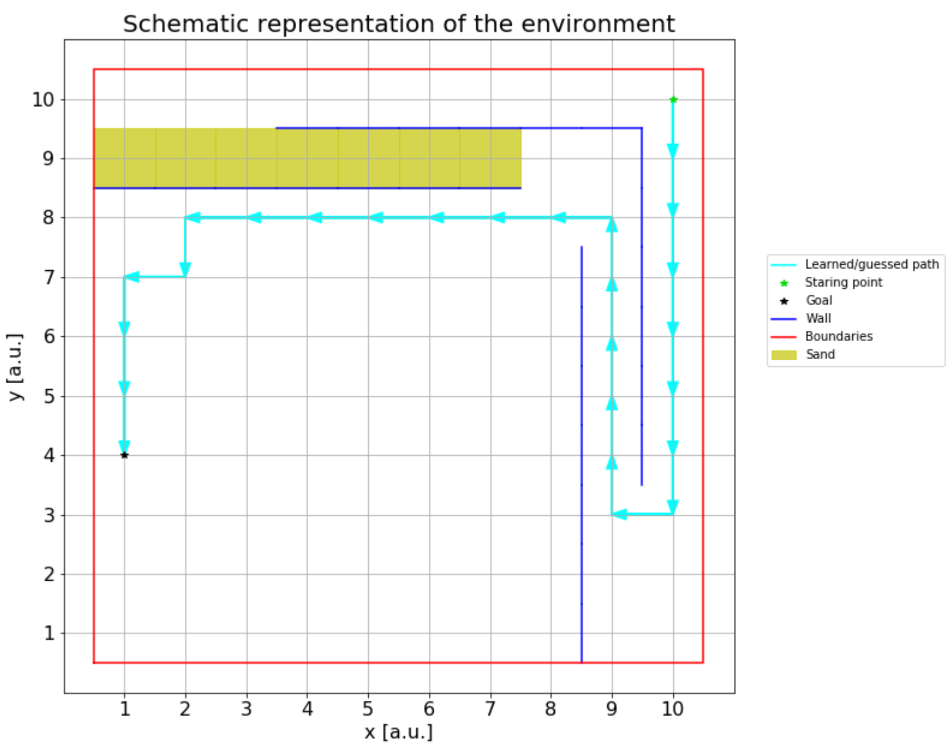

Figure 14: Second environment, first sand disposition. Algorithm: Q-learning. $\delta = 0.9$. *ε-function: linear (given). Episode: 2000*

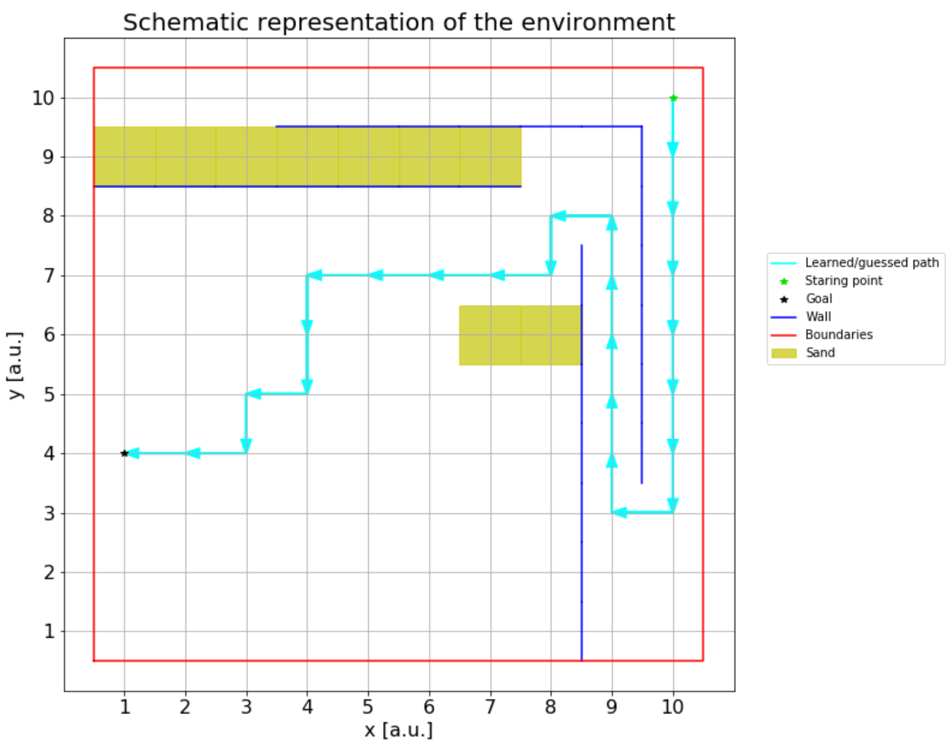

Figure 15: Second environment, second sand disposition. Algorithm: Q-learning. $\delta = 0.9$. ε-function: linear (given). Episode: 2000

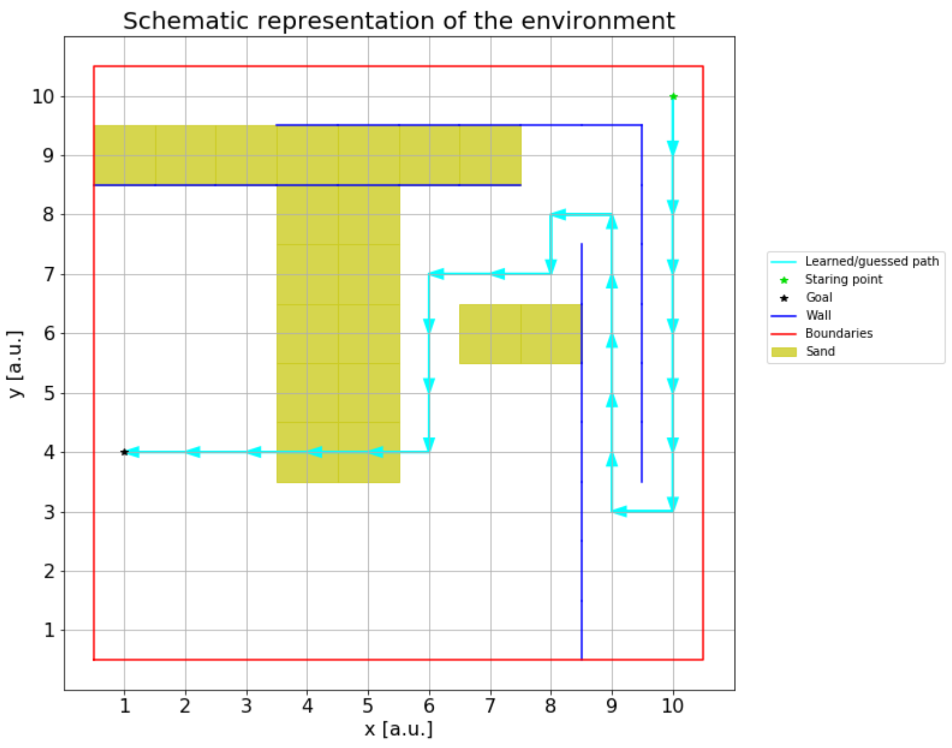

Figure 16: Second environment, third sand disposition. Algorithm: Q-learning. $\delta = 0.9$. ε-function: linear (given). Episode: 2000

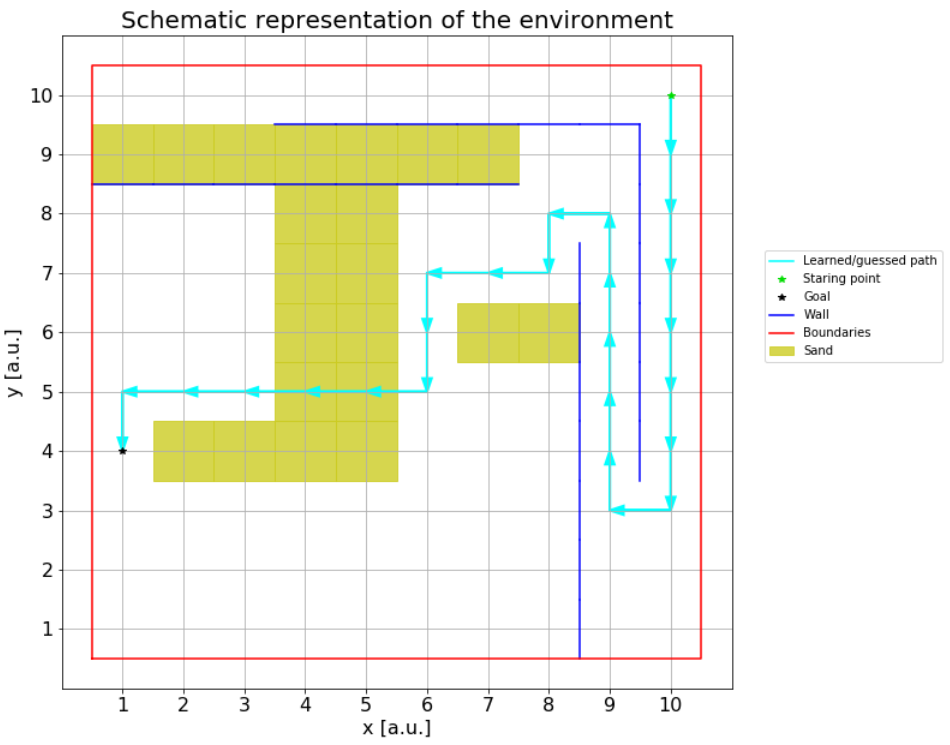

Figure 17: Second environment, fourth sand disposition. Algorithm: Q-learning. $\delta = 0.9$. ε*-function: linear (given). Episode: 2000*