Solving an escape room with Q-learning and Expected Sarsa

Clément Cosserat, Enguerrand Monard

March 29, 2022

1 Introduction

An escape room is a game where a player is trapped in an environment and must find his way out by solving a series of riddles and puzzles in a given order with the clues s.he collected in the room. As a leisure activity, it gained a lot of popularity recently, but besides the immersive aspect, this family of games is interesting for us because it can be easily formalized as a Markov Decision Process, and is thus implementable in a reinforcement learning framework. Our goal is therefore to create an environment to simulate a very basic case of escape room and train an agent to solve it with an optimal reward.

You can find the github repository of this work on `https://github.com/Mentel1/Escape-Room-RL`.

We strongly encourage the reader to watch the slideshows we recorded for this project, enclosed within the zip file, to be opened with powerpoint.

2 Our game

Our sketch of escape room is composed of a grid with a case corresponding to the door which is the final goal, a key which is somewhere in the room at the beginning of a game and must be collected by the player to be able to open the door, and some obstacles which are inaccessible cases. The player must therefore pick up the key and go to the door as fast as possible.

3 Environment

3.1 initialization

Our environment contains a grid with height h and width w, together with a variable *gotKey* indicating if the agent has found the key yet. It has also attributes for the location of the door (which represents the goal), the initial location of the key, the initial location of the player, and a list of obstacle locations. At the function start, the current location of the agent is set to the start location (the environment stores the current position of the agent at every moment), and the $gotKey$ variable is set to $False$. All those parameters are specified in a dictionary $envInfo$ which contains the features of the room, this way we can run our experiments on many different rooms which are easy to generate.

3.2 step

The step function of the environment takes as input an action of the agent and outputs a triplet $(reward, state, term)$ where $term$ is a variable which is set to $False$ by default and activates when the door is reached by the player with the key. A state is defined as a tuple $(x, y, gotKey)$, whether the room has a key in it or not. The x and y convention is the one from 'numpy' indexing. For example, the state $(3, 0, 1)$ means that the agent is 3 tiles down, 0 right starting from the top-left corner and that it already stepped on the key.

Regarding the reward, it is set to -1 for almost all state-action couples (the player must minimize the number of steps in order to have the best possible score). For the special case of finding the key, the player gets a reward of 1, and when he reaches the final goal, he gets a reward of 10. It should be noticed that there is no stochasticity in the reward and the transition between the states is deterministic too. Finally, the last action performed by the agent is also kept as an attribute of the environment because it is used in the rendering function.

3.3 render

We added a rendering procedure which displays the position of the player in the grid, in the form of an arrow which indicates his last action. The door, the key before it is found, and the obstacles are also made visible. You can try to play this mini-game with the `play.py` script in the project folder to visualize what the environment looks like.

3.4 rooms

The rooms we use for our experiments are shown in Fig. 1:

4 Agent

The agent has four possible actions, he might go up, left, down, or right, and those actions are respectively encoded by integers between 0 and 3 in our code. Please note that if the agent tries to run through a wall or an obstacle, it will stay where it stands but still get a reward of -1.

The agent class we used is designed to support q-learning and expected sarsa, so it keeps an attribute with state-action estimated values, and a last-action attribute to be able to update these values. The implicit policy we used is ϵ-greedy, which means that our agent must also have attributes for the parameters ϵ, α which is the learning rate, and γ which is the discount factor.

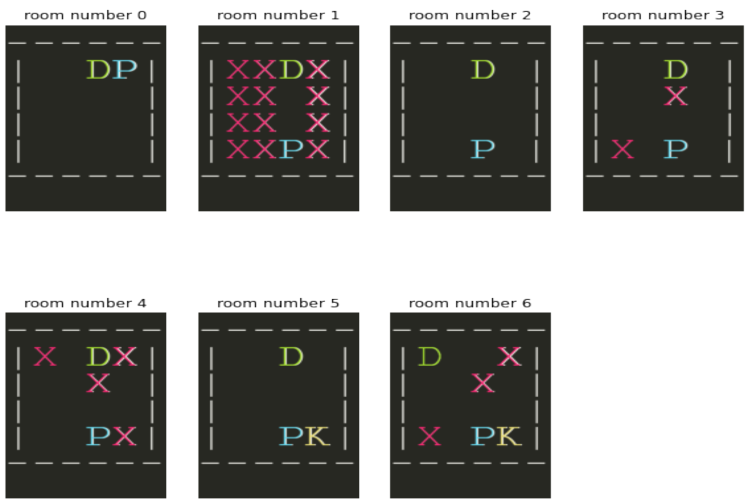


Figure 1: The rooms where the player evolves, from the simplest one to the most complex one

5 Optimal solution

We wrote an iteration value algorithm which we can run on environments and which outputs an optimal policy and a value map of the states. As the problem is fully deterministic and has a finite number of states, the value iteration converges exactly after a finite number of steps and we can use the criteria $newValues == Values$ to decide when to stop it. This optimal solution can be used as a comparison point for the agents' performances. Fig. 2 shows the value maps for the rooms which do not require the key to open the door as an example.

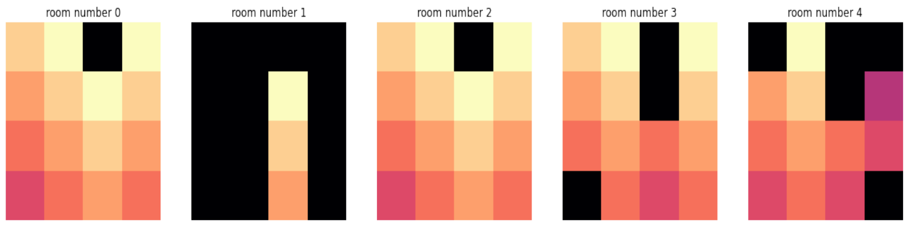


Figure 2: values maps for rooms 0 to 4

On Fig. 3 we display the policy computed for the last room. The interpretation is quite straightforward, except for the arrow in the right-hand bottom corner which is actually on the case with the key and does not correspond to a true action but rather an initial value in the value iteration algorithm which has not been updated during the computation of the best policy.

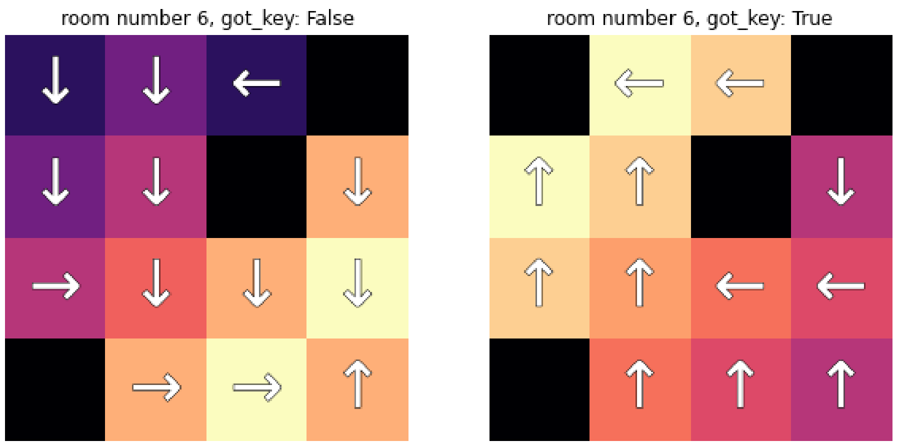


Figure 3: the policy computed by value iteration for the last room

6 Learning agents

6.1 Q-learning agent

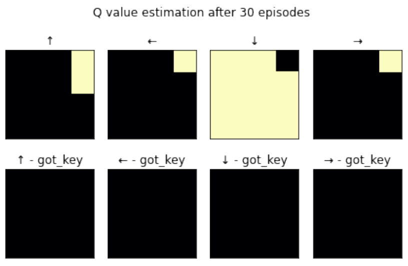
6.1.1 Principle

The Q-learning agent updates with a step α the estimated value of the last action-state where it has been with the reward it obtained plus the discounted value of the current state assumed to be the max of the values for the possible actions in this state:

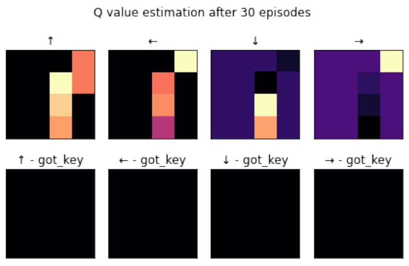
$$q[s_{old}][a_{old}] = (1-\alpha)q[s_{old}][a_{old}] + \alpha(R_{s_{old},a_{old}} + \gamma \max_a(q[s_{new}][a]))$$

6.1.2 Results

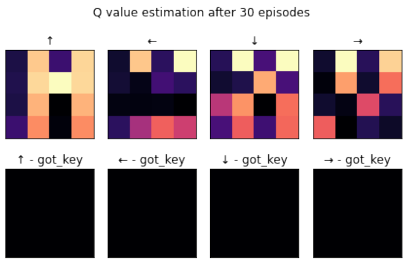
The results we get for a learning agent without any assumption on the model (that is, with an action-state value map), takes the form of a collection of 2 x 4 plots displaying the values of each location in function of the action considered and of the state of the variable *gotKey*. We usually perform learning sessions over 30 episodes, as it turned out that the agents learn really fast for those simple games. Also, the results do not vary much so we did not find useful to average the results over several runs even if we added the possibility to do so in our code.



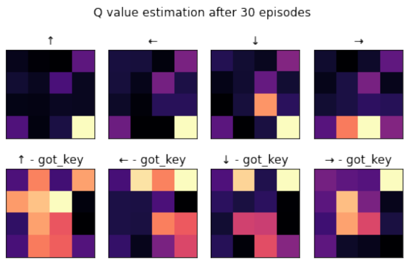
(a) Room 0



(b) Room 1



(c) Room 4



(d) Room 6

Figure 4: q value map for rooms 0, 1, 4 and 6

Fig. 4 shows those maps for rooms 0, 1, 4 and 6.

Then, we display this information in the form of arrows which represent which actions have higher values for each state, which gives the more compact plots of Fig. 5.

We finally show which tiles are the most visited during the 10 last episodes for the same agents in the same rooms on Fig. 6.

6.2 expected SARSA

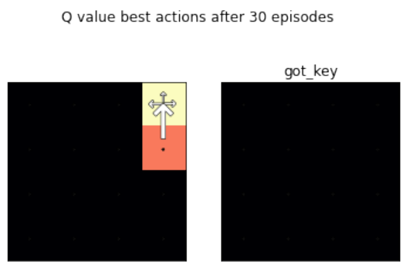
6.2.1 Principle

This time, the updates are given by the formula

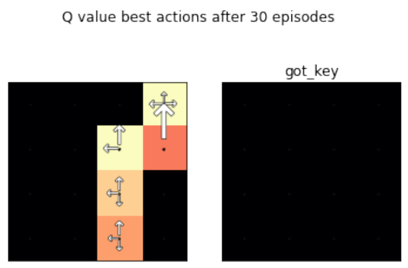
$$q[s_{old}][a_{old}] = (1-\alpha)q[s_{old}][a_{old}] + \alpha(R_{s_{old},a_{old}} + \gamma E_{a\sim\epsilon-greedy}(q[s_{new}][a]))$$

. The difference with q-learning is that the agent evaluates the value of the next states taking into account the possibility of doing a sub-par action due to the ϵ probability of taking a random action. In a sense, this agent is more realistic on its future behavior and incorporates this consideration in its assessment of the state-action values.

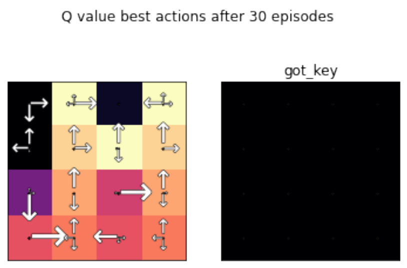
Like we did for the Q-learning agent, we display on Fig. 7, the values map as well as the best actions and the most visited cases of this room during the 10 last episodes. This time we do it only for room 6.



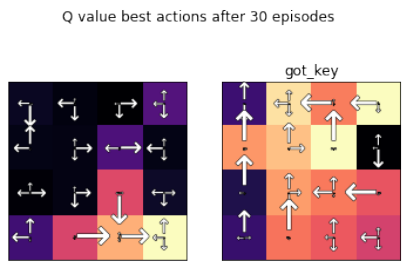
(a) Room 0



(b) Room 1



(c) Room 4



(d) Room 6

Figure 5: Q value and estimation and best actions for rooms 0, 1, 4 and 6

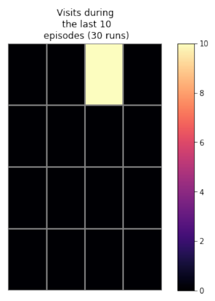
6.2.2 Results

6.3 experiment

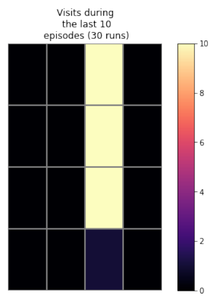
To conclude, we did a final experiment: we initialized the two agents in the first room, they spend 30 episodes learning in this room before we move them to the second room where they spend again 30 episodes and so on. This experiment aims at observing how the agents adapt to a change of environment. The result is shown in Fig. 8 where we plot the reward earned for each episode. What we note is that there does not seem to have a big difference between the two agents. Also, the rooms with a key seem harder to learn for the agents, as their reward takes about 10 episodes to become stable in those rooms whereas the learning is almost instantaneous in the first rooms.

7 Discussion

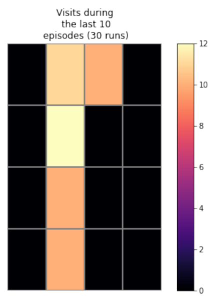
There are many other features we could have added to this set-up. To name a few examples we thought of, we could have put some holes giving the agent the possibility to die and get a very negative reward. Maybe it would have enable to see a clearer difference of behaviour between the two kinds of agents (like we saw with the cliffworld environment).



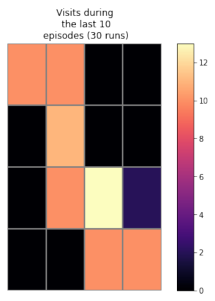
(a) Room 0



(b) Room 1



(c) Room 4



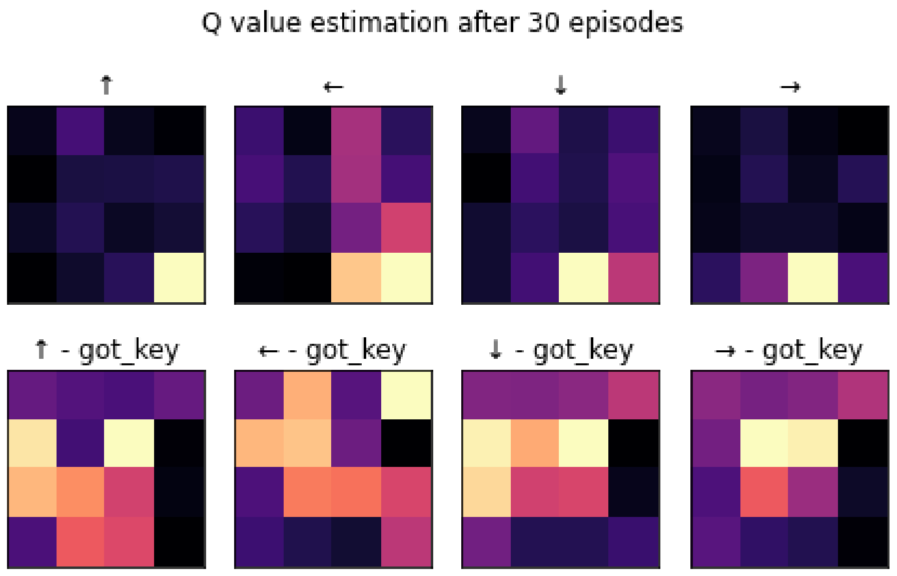
(d) Room 6

Figure 6: 10 last visits for rooms 0, 1, 4 and 6

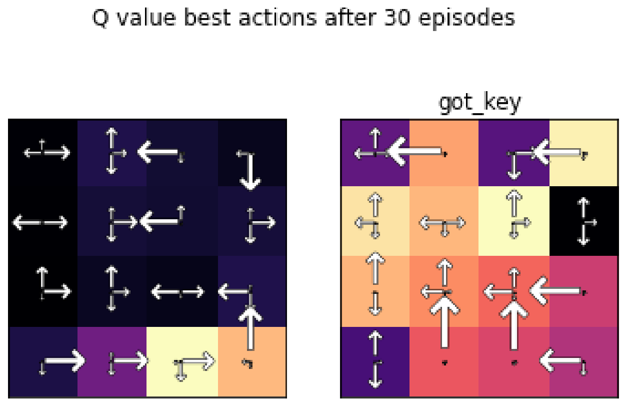
Also, we could have augmented the size of the game, using broader rooms, more keys, or even doors to change rooms and set the goal in a different room than the starting location. With a more complex environment, it would have been interesting to change the rewards when an agent finds a key to see which influence intermediate rewards have on the time necessary to learn.

An other option would have been to make the agent die after a certain time to make him find the solution faster and avoid extremely long episodes at times.

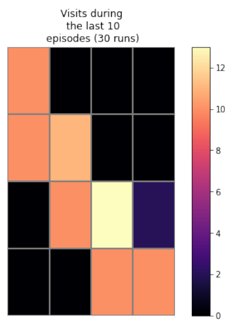
Finally, let's remember that these agents do not solve the game the way we do. To achieve human-like performance, it is in this case mandatory to use as a state the whole screen, and maybe to use deep-Q-learning to process the screen as we do.



(a) Q-value map



(b) Actions



(c) Visits

Figure 7: Expected Sarsa in room 6

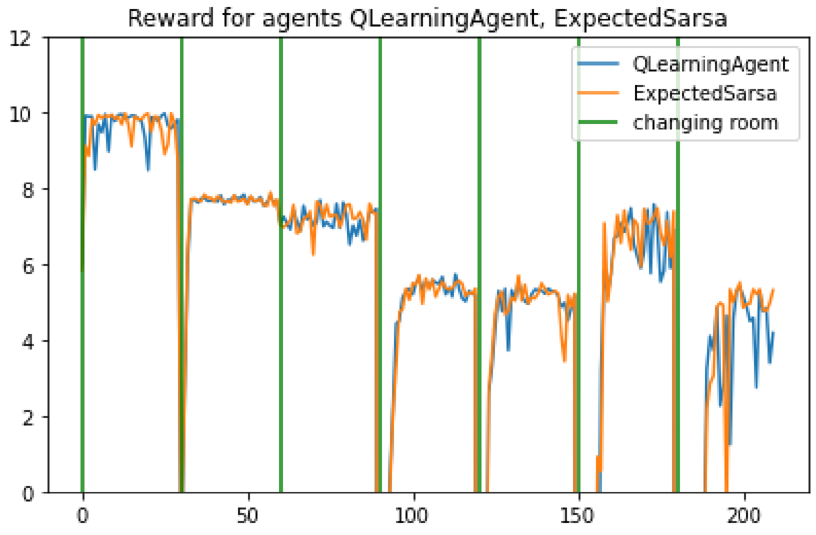


Figure 8: reward obtained along the episodes