“Q-learning vs SARSA applied to SMART CAB”

A PROJECT REPORT

Submitted by

MOHAMMAD WASIL SALEEM

MATRIKEL Nr.: 805779

[MAT-DSAM3A] Advanced Data Assimilation and Modeling A

Reinforcement Learning

<u>TITLE</u>

“Q-learning vs SARSA applied to SMART CAB”

Reinforcement Learning

In Reinforcement learning, we learn by directly interacting with the environment. It is the science of decision making. We basically have an Agent, which interacts with the environment, observes the environment, and makes a calculated decision. Let's think of a pet dog. And let us assume that the pet owner will give the dog a treat in the form of biscuits only when he rings a bell placed near him. As soon as the dog rings the bell, positive reinforcement will be provided to him in the form of biscuits. If the dog rings a bell 2-3 times, and every time he gets a biscuit, then he learns that ringing a bell would earn him a biscuit, and not ring a bell would earn him nothing.

There are several components of Reinforcement Learning:

Environment: The environment is the world where the agent moves, i.e. receive action from the agent, emits observations, and rewards to the agent. So, the environment would be the pet owner with whom the dog is interacting.

Agent: The agent takes action, interact with the environment, receives the reward. In the above example, the dog would be an agent.

Action: Action would be a set of all the possible moves an agent can make on the environment. The action would be the dog ringing the bell.

States: State is the information used to determine what happens next. An agent will always be in some states. And when an agent takes action, it moves to the next state, which might be a better state than the current one or worse. The reward we get depends on the state we are in. The state for the above example would be the dog waiting to be fed and the dog eating the biscuit.

Reward: The Reward can be positive or negative. It depends on what action we take to land on the next state. The reward would be the biscuits or no biscuits.

Policy: The policy determines the agent's behaviour, how it picks an action. It tells us for every state what action it recommends we should take in that state. There are two types of policy: stochastic policy has a probability distribution over the set of actions given a state, and Deterministic policy maps the state to actions.

Q-Value: $Q^{\pi}(s, a)$ is the action-value function. It is the expected return starting from state s, following policy π, taking action a.

Discount factor: The value of discount factor is in between 0 and 1, i.e $\gamma = [0,1]$. The total reward we get is the discounted reward at each time step, t,

$$G_t = R_{t+1} + R_{t+2} + \ldots = \sum_{k}^{\infty} \gamma^k R_{t+k+1}$$

Where G_t is the return. If γ is close to one, then we say the go for immediate rewards, and if it is close to 0, then it would go for long term rewards.

SMART CAB GAME

The Smart Cab game has been inspired by the OpenAI gym environment, Taxi-v2. The Smart Cab game has been implemented in pygame from scratch to have a better understanding of the codes and to make it as flexible as possible. The game has been created on a 2x2 grid of size 500x700 (width and height).

The main object of the cab is to pick up the passenger from one of the four possible pickup positions, taking as minimum time possible to drop off the passenger at the right location. The passenger can be dropped off only at four specific locations. The passenger cannot be dropped off from where it has been picked up.

The cab can move around the 5x5 grid freely, except it cannot go through the walls. It can go “UP”, “DOWN”, “LEFT”, and “RIGHT”. The cab cannot move diagonally. In Fig. 1, we can see the red line represent the walls. The cab also cannot go beyond a 5x5 grid. So, it if is at the top row, then it cannot go up furthermore. The same is with the bottom row and extreme left and right columns.

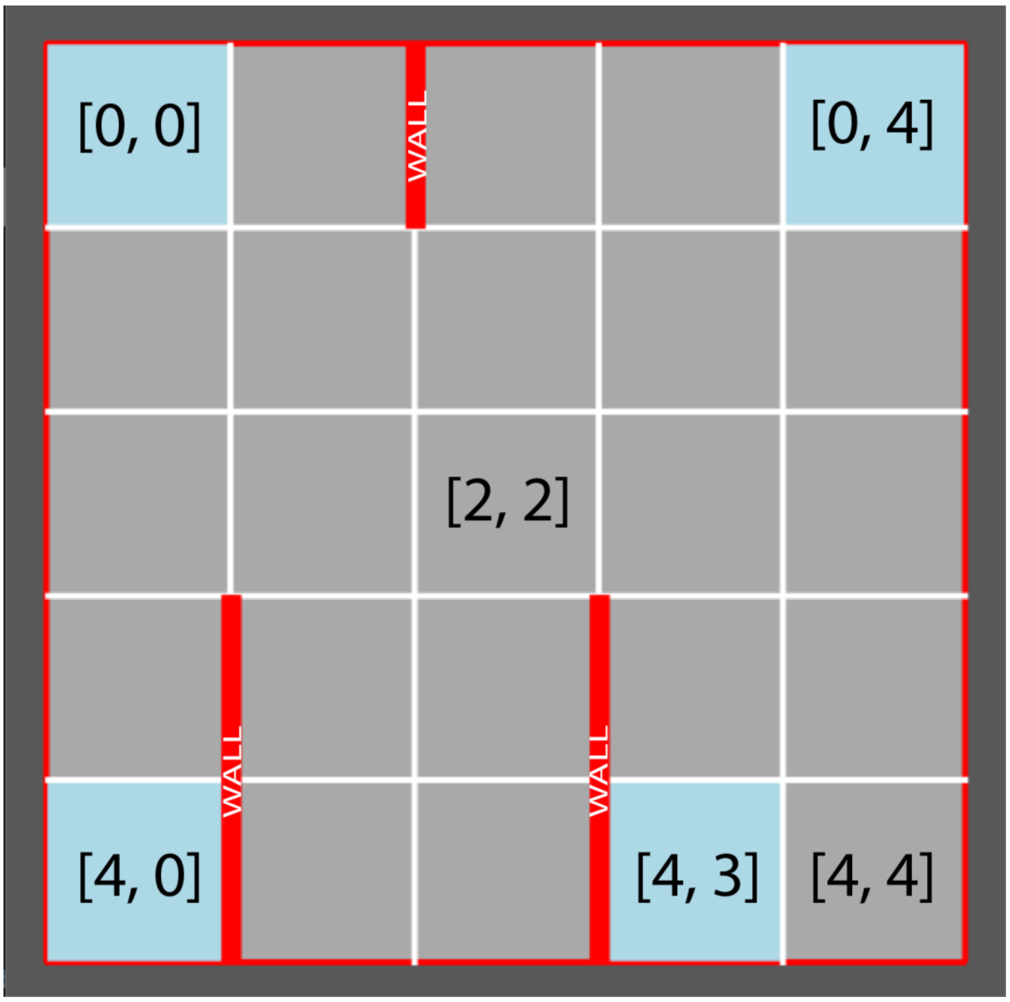


Fig 1. Basic representation of the Grid where the agent will move, does not cross the wall or the boundary and detail description of the Coordinate systems.

Coordinate system: The top left corner of the grid is represented as [0, 0]. The top right corner is represented as [0, 4]. The bottom left corner is represented as [4, 0], and the bottom right is represented as [4, 4,].

Pickup positions: The passenger can be picked up from [0, 0], [0, 4], [4, 0] and [4, 3].

Dropoff positions: The passenger can be dropped off to any of the states which are mentioned in pickup position, except one position where the passenger has been picked up.

Workflow: As soon as the game starts, the cab would get the information about the passenger pick up locations. The can would go over the location, and as the passenger location becomes equal to the cab's locations, then it would mean that the cab has picked up the passenger.

And as the cab picked up the passenger, the position of the passenger updated to the cab's location since the passenger is in the cab now.

And then, the cab would go to the destination, and when it reaches the destination, it would simply drop off the passenger.

Just after the dropoff, we would get a reward, and the game restarts.

SMART CAB with Reinforcement Learning

So, now we have set up our environment, let’s discuss components of reinforcement learning for the game.

Environment: The game would be the environment here, randomly selecting the pick-up and the drop off position, emits the observations or the reaction, and the rewards to the cab, restarting the game, etc. This is where the cab would move around.

Agent: The cab is the agent. It interacts with the game environment, receives rewards.

Action (A): The cab can take four possible actions: “UP”, “DOWN”, “LEFT”, and “RIGHT” on the grid, taking into account the rules of the game. There are no explicit actions for picking up or dropping off the passenger (as it was the case for the OpenAI gym env).

Whenever the agents aim is to pick up the passenger, and it reaches the state where the passenger is, we will assume that it has picked up the passenger. The same is the case for dropping off the passenger.

States (S): The total number of states are: 13*4 + 16*21 = 388 states. More on the next page.

Reward (R): For each of the steps, the cab will get a penalty of -1, so that motivates the agent or the cab to go to the state with a higher reward as soon as possible. The main aim of the game is to take as minimum time as possible for the cab to pick up and drop off the passenger. I tried to implement a different reward function for picking up the passenger. I started with a reward of 0 since we considered that a reward of 0 would be much better than the reward of the sum of negative values, which are being added up for each step taken. But that does not give any satisfactory result, and hence moved to a reward of +30 for picking up the passenger. To drop off the passenger, we use a reward of +20. The aim was to assume that the passenger would be happier if the cab driver would reach the passenger as soon as possible, as the passenger does not have to wait for too long.

But if the agent tries to pick up the passenger from the wrong location, it would incur a reward of -10 as it might lead to an increase in time to pick up the passenger. The same -10 reward is penalised if the agent tries to drop off the passenger at the wrong location.

Therefore, the overall reward function looks like:

$$\boldsymbol{Reward(s,\ a) = \begin{cases} -1 & for\ every\ step \\ -10 & pickup\ from\ wrong\ location \\ -10 & drop\ off\ at\ wrong\ location \\ +30 & pick\ up\ from\ right\ location \\ +20 & drop\ off\ at\ right\ location \end{cases}}$$

Policy: The policy determines the agent's behaviour, how it picks an action. It is a deterministic policy, maps directly from the state to action.

STATE SPACES

The number of state spaces depends on the situation of the game. The coordinates where we can pick up and drop off the passenger can be different than the regular ones. We will call the coordinates of states, i.e. [[0, 0], [4, 0], [0, 4], [4, 3]] as a special state.

[0, 0] encodes integer ‘0’.
[4, 0] encodes integer ‘1’.
[0, 4] encodes integer ‘2’.
[4, 3] encodes integer ‘3’.

Fig2. Encoding for the special states, where we can drop off and pick up the passenger.

```
'0_0_0_0_0',
'0_0_0_0_1',
'0_0_0_0_2',
'0_0_0_0_3',

'0_0_0_4_0',
'0_0_0_4_1',
'0_0_0_4_3',

'0_0_4_0_0',
'0_0_4_0_2',
'0_0_4_0_3',

'0_0_4_3_0',
'0_0_4_3_1',
'0_0_4_3_2'.
```

Fig. 3 Special State space with the Cab location at [0, 0].

The first 2 coordinates is the state of the car, row and column respectively, the next 2 coordinates are the passengers coordinates, row and column respectively, and the one at the end is the Encoding of the drop off and pick up state (for reference, see the Fig. 2)

```
'1_2_0_0_1',
'1_2_0_0_2',
'1_2_0_0_3',

'1_2_0_4_0',
'1_2_0_4_1',
'1_2_0_4_3',

'1_2_1_2_0',
'1_2_1_2_1',
'1_2_1_2_2',
'1_2_1_2_3',

'1_2_4_0_0',
'1_2_4_0_2',
'1_2_4_0_3',

'1_2_4_3_0',
'1_2_4_3_1',
'1_2_4_3_2'.
```

Fig. 4 Regular state space, with [1, 2] is the cab’s location.

The first 2 coordinates is the state of the car, row and column respectively, the next 2 coordinates are the passengers coordinates, row and column respectively, and the one at the end is the eEncoding of the drop off and pick up state (for reference, see the Fig. 2).

There would be 13 possible states considering that the cab is in one of the special state s. All these possible states do not represent the possibility of choosing one action at a p articular time. This is the overall result after training the Q-learning for some number o f episodes. Let us consider the first state, '0_0_0_0_0' in Fig. 3. It says that the cab loc ation is at [0, 0], and the passenger is in the car. And the drop location is also [0,0], wh ich means that the cab dropped off the passenger here. The next state, '0_0_0_0_1' say s the same, and it will drop off the passenger at state [4, 0] after travelling for some tim

e. So, considering the cab’s location is at one of the special states, it can have 13 possi ble states to pick and drop off the passenger. That would constitute 13x4 = 52 states.

Let’s now consider that the cab’s location is at any regular state. Let us take the first st ate space. The cab’s location is [1, 2]. And the passenger’s location is at [0,0]. This me ans that the cab had to drive to that state to pick up the passenger and have to drop it of f at [4, 0]. Here, the pick-up location should not be equal to the drop off location. And when it picked up the passenger, then it can drop off the passenger in any of the states given, considering that the pick-up and drop off should not be equal. For e.g. In state 7 , the cab’s location is [1, 2] and passenger’s location is [1, 2], which means that the ca b has already picked up the passenger and is going to drop off at the location, [0,0]. So the regular state space can have 16 possible states, considering the car is in any of the r emaining 21 states other than the special states, thus constitutes 21*16 = 336 states.

Therefore, the total number of the states would be **52+336 = 388 states**.

SARSA and Q-Learning

A model-free algorithm in reinforcement learning is an algorithm that does not use transition probability distribution associated with MDPs and hence is called model free [9]. Whereas model dependent RL algorithm uses the transition probability distribution. This probability distribution has all the knowledge for the agent to transits to the next state [10].

Temporal Difference is an approach to learn how to predict a quantity that depends on future values of a given signal [18]. Temporal Difference is a model-free reinforcement learning algorithm. The agent learns from experience by interacting with the environment without any prior knowledge of the environment [11] or any other transition distributions.

In a model-free algorithm, the agent learns the policy using algorithms like Q-learning and SARSA. This is what we will discuss next.

<u>**SARSA**</u> – **S**tate **A**ction **R**eward **S**tate **A**ction.

SARSA is an on-policy Temporal difference control method [10]. A policy is defined as a map from state to action. It can be considered as a dictionary, with keys as the state and action as values []. SARSA uses an on-policy algorithm.

A policy can be on-policy and off-policy. In on-policy learning, the function is learned from the actions that we took using our current policy [], here we used epsilon greedy policy. The update step for the on-policy SARSA algorithm is as follows:

$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \left(R + \gamma \, \hat{Q}(\acute{s}, \acute{a}) - \hat{Q}(s, a)\right)$$

Th action $\acute{a}$ was taken according to the epsilon greedy algorithm.

Exploration vs Exploitation [14]

Exploration allows the agent to randomly select an action, hoping to improve its current knowledge about each action and thus benefit in the long term [14]. And sometimes, the agent using exploration goes to that state where it has never been before. And this enables the agent to make a more informed decision, as it gets more accurate estimates of action-values.

Exploitation allows the agent to choose the action in a greedy way to get the most reward[14]. But by being greedy with respect to action-value functions, it may not actually get the most reward and leads to sub optimal behaviour.

Epsilon greedy policy

To balance Exploration and exploitation, we use Epsilon greedy policy. We Choose a value of epsilon and then generate a random number between 0 and 1. And compare them both. If the random number is smaller than epsilon, then we go for Exploration, i.e., randomly selecting the action or else, we go for exploitation, taking the best action.

If epsilon is small, then there is a higher chance of selecting the maximum action, exploitation, and if epsilon is large (close to 0.9), then there is a higher chance of selecting action randomly, thus Exploration, and this will allow the agent to explore the environment even more, which helps them in a long term since they try to go to each of the states. Exploitation allows the agent to choose the action in a greedy way to get the most reward [14]. But by being greedy with respect to action-value functions, it may not actually get the most reward and leads to sub optimal behaviour.

Qlearning

Q-learning is an off-policy Temporal difference control policy [10]. It does not use any policy to find the next action, a', but chooses the action in a greedy way i.e. by taking a maximum of the $Q(s, a)$.

In off-policy learning, the $Q(s, a)$ function is learned by taking different actions, here taking the maximum of $\hat{Q}(s, a)$, where $\acute{a}$ is the current action [13].

$$\hat{Q}(s,\, a) \leftarrow \hat{Q}(s,\, a) + \alpha \left(R + \gamma\, \hat{Q}(\acute{s},\, \acute{a}) - \hat{Q}(s,\, a)\right)$$

The rest of the equation is similar to SARSA, except, taking max of $\hat{Q}(\acute{s}, \acute{a})$.

IMPLEMENTATION

The aim of the project was to compare the Q-learning with the SARSA algorithm on Smart cab. I implemented both algorithms with different rewards. I already explained both Q-learning and SARSA algorithms, and now I will give a brief overview of their implementation on the game.

The parameter values I choose are as given below:

1. Learning rate, $\alpha = 0.1$.
2. Discount factor, $\gamma = 1$.

So, I used the SARSA for training the agent for 500,000 episodes. I first used the reward of 0 for picking up the passenger, but that didn't go well with SARSA and the agent does not perform well. The total average cumulative reward was **-25.12**.

Choices of hyperparameters in SARSA: I use epsilon greedy policy with an epsilon value of 0.4. So, it would choose a random policy with a probability of 0.4 and the maximum action with a probability of 1-epsilon.

We can see from here that the cumulative reward until 1000 episodes, it did not converge very well. There is still a very high number of negative rewards.

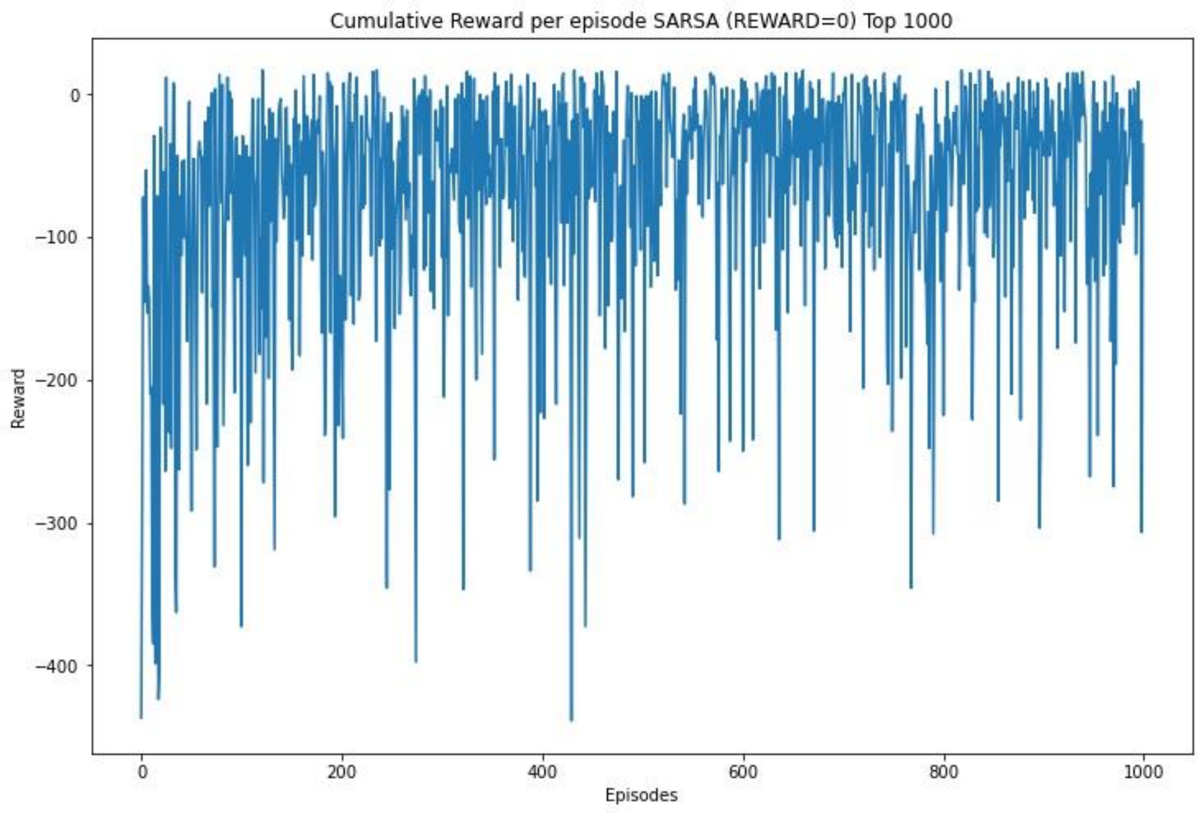


Fig 5. Cumulative Reward per episode for SARSA algorithm with reward = 0 for picking up the passenger. Only Top 100 values are displayed.

Then I move on to Q-learning with the same reward function as before, and it did work quite well. It got a total cumulative average reward of **10.92** over 500, 000 episodes.

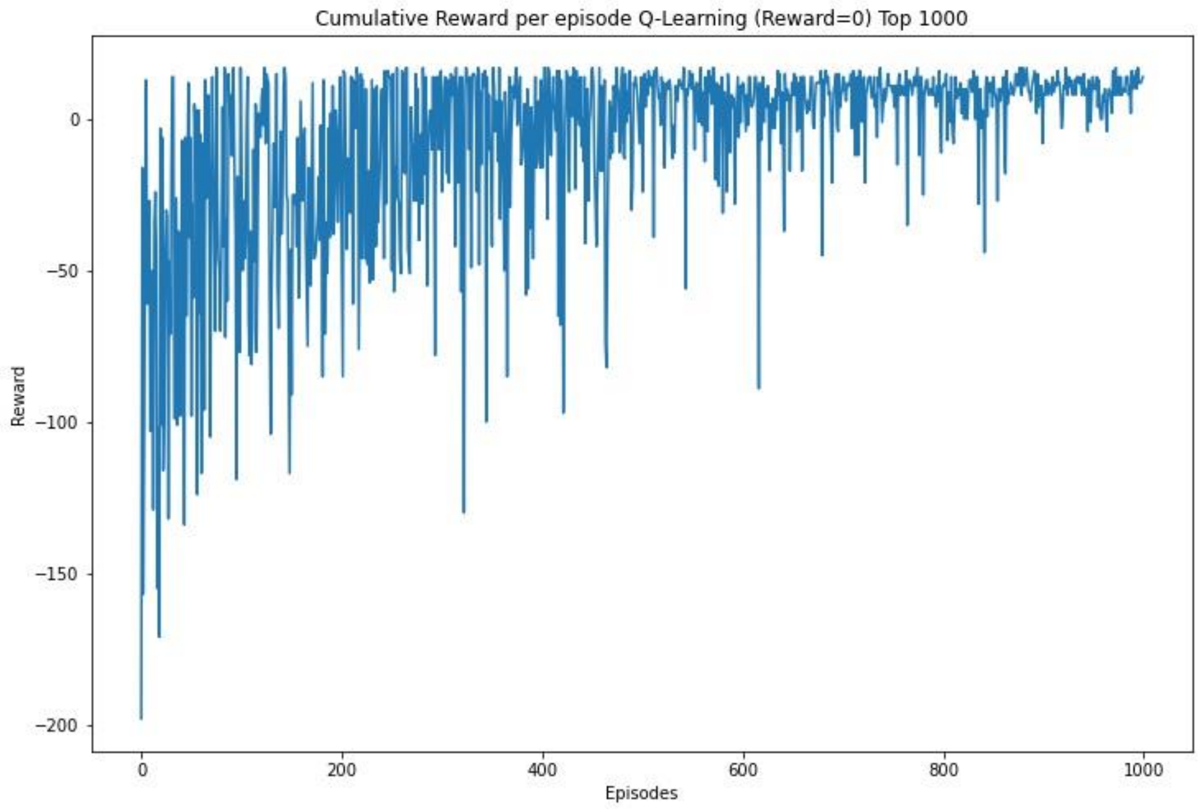


Fig 6 Cumulative Reward per episode for Q-Learning algorithm with reward = 0 for picking up the passenger. Only Top 100 values are displayed.

Even though it converges quite well, as we can see from Fig. 6, but it was still not enough for the agent to move freely in the environment. Sometimes, the agent get stuck in 4 boxes, with coordinates, [3, 1], [3, 2], [4, 1] and [4, 2]. If somehow, the agent is able to escape those four boxes, then it runs smoothly without delay.

When I compare the average cumulative reward of Q-Learning and SARSA, Fig. 7, then I saw that the average value for Q-Learning was higher than SARSA, which is not common. This will be described later.

Then I shift my focus to a reward of +30 for picking up the passenger. This drastically improves learning.

For SARSA, I use the same settings as before, except for changing the reward function. I got a cumulative average reward of **5.72**, which was a tremendous improvement over the previous SARSA model. As we can see from Fig. 8, the model already converges, but still, it has some negative reward over episodes.

Then I compared this model with Q-learning, without the exploration steps, i.e. only focus on exploitation, I got a reward of **0.9812**. It was quite low; although the game was smoothly running, the agent was taking as few steps as possible between its pickup and drop off. Fig. 9 plot the graph for Q-learning.

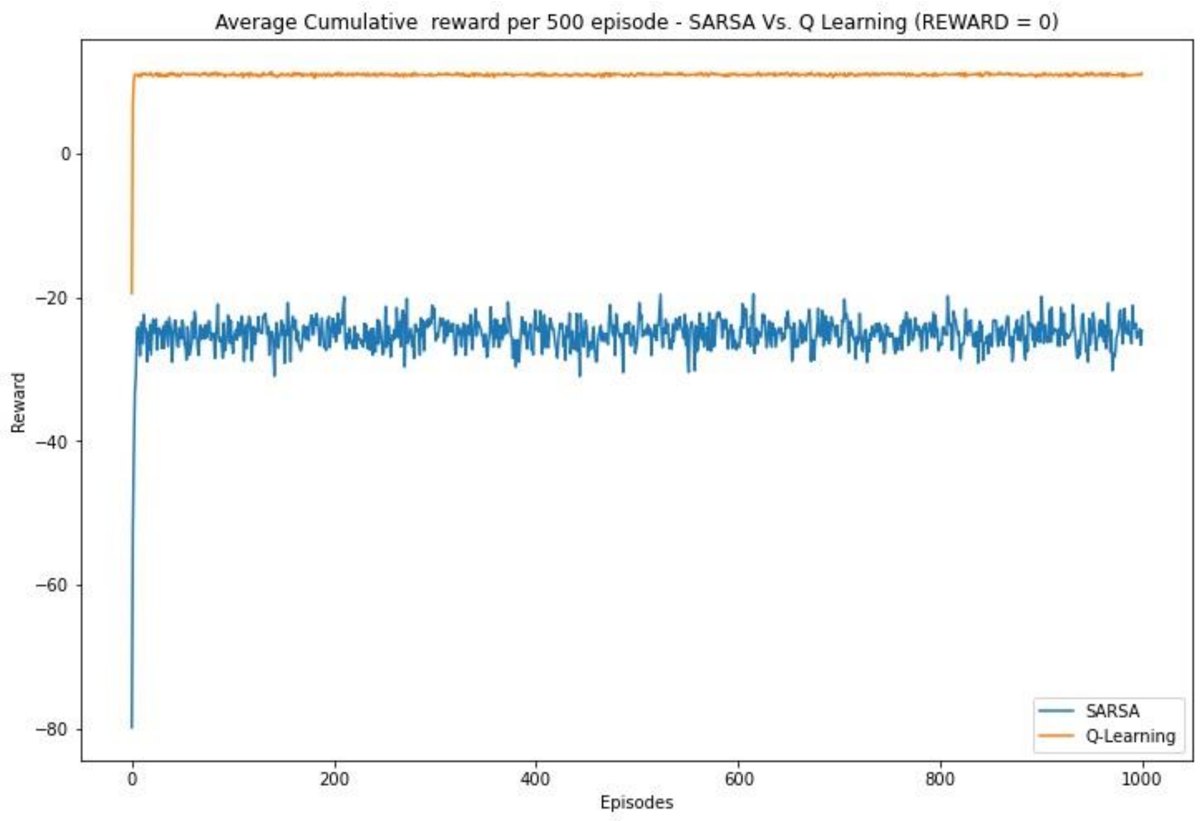


Fig 7. Average Cumulative reward per 500 episodes between SARSA and Q-Learning with Reward = 0 for picking up the passenger.

Both SARSA and Q-Learning seems to be similar, so I plot a comparison plot, Fig. 10.

Because SARSA learns the safe path, it actually receives a higher average reward per episode than Q-Learning even though it does not walk the optimal path since we use the exploration step with 0.4 probability [15].

We can also see that less change was noticed in SARSA as compared to Q-Learning in the average cumulative reward per 500 episodes, so in a sense, we can say that it converges quicker than Q-Learning. Also the average cumulative reward for SARSA was much better than Q-Learning algorithm.

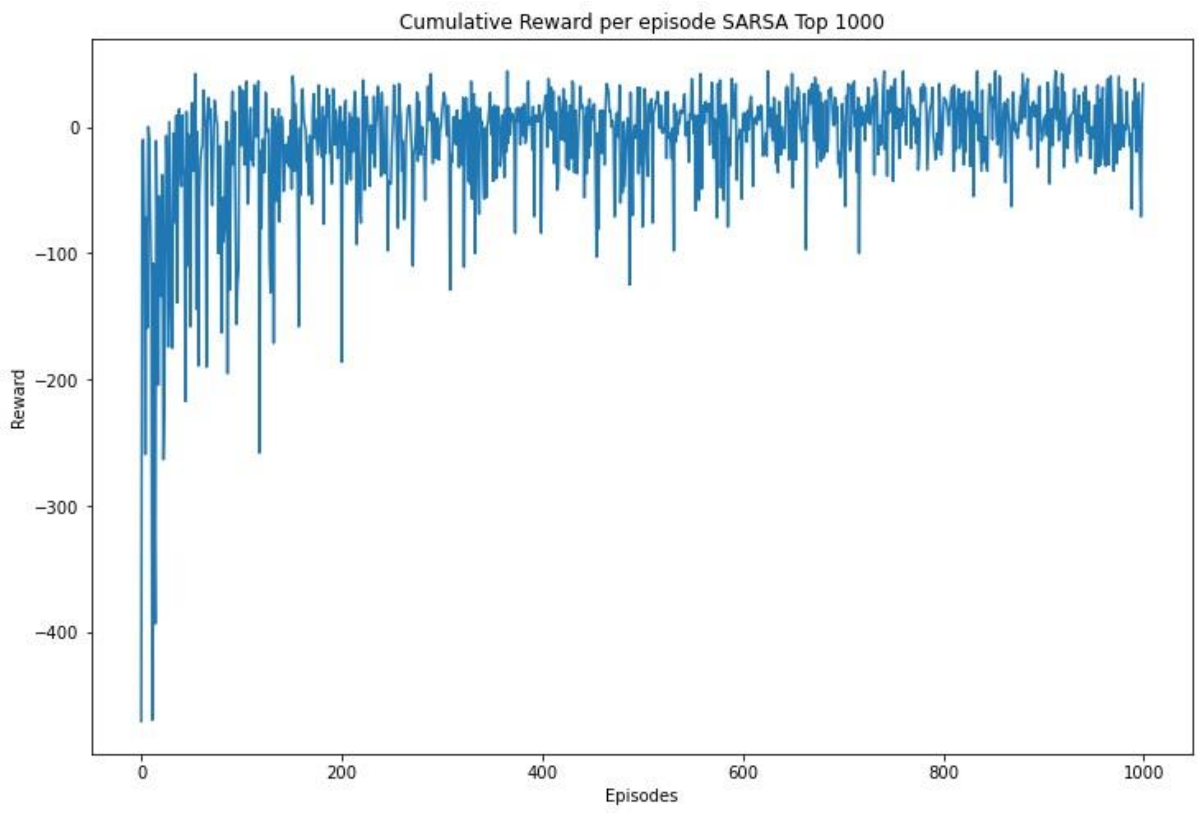


Fig 8. Cumulative Reward per episode for SARSA algorithm with reward = +30 for picking up the passenger. Only Top 100 values are displayed.

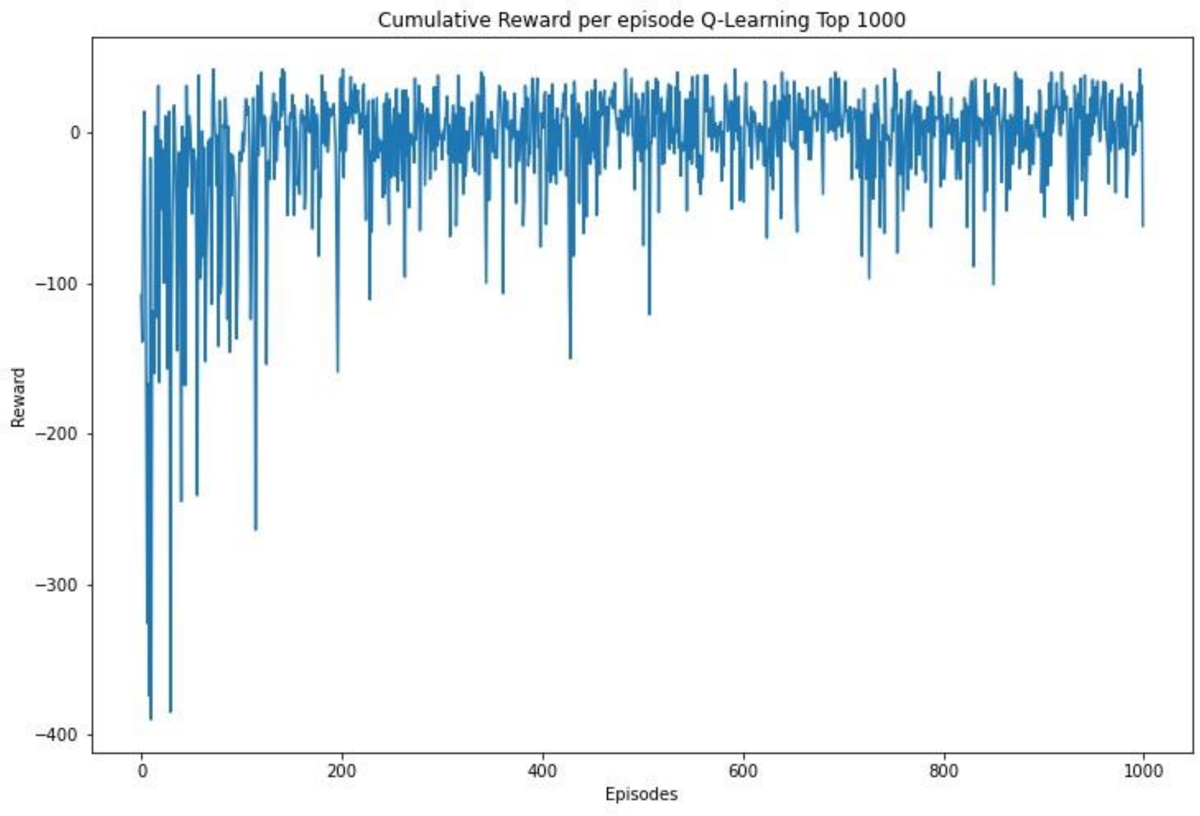


Fig 9. Cumulative Reward per episode for Q-Learning Algorithm with reward = +30 for picking up the passenger. Only Top 100 values are displayed.

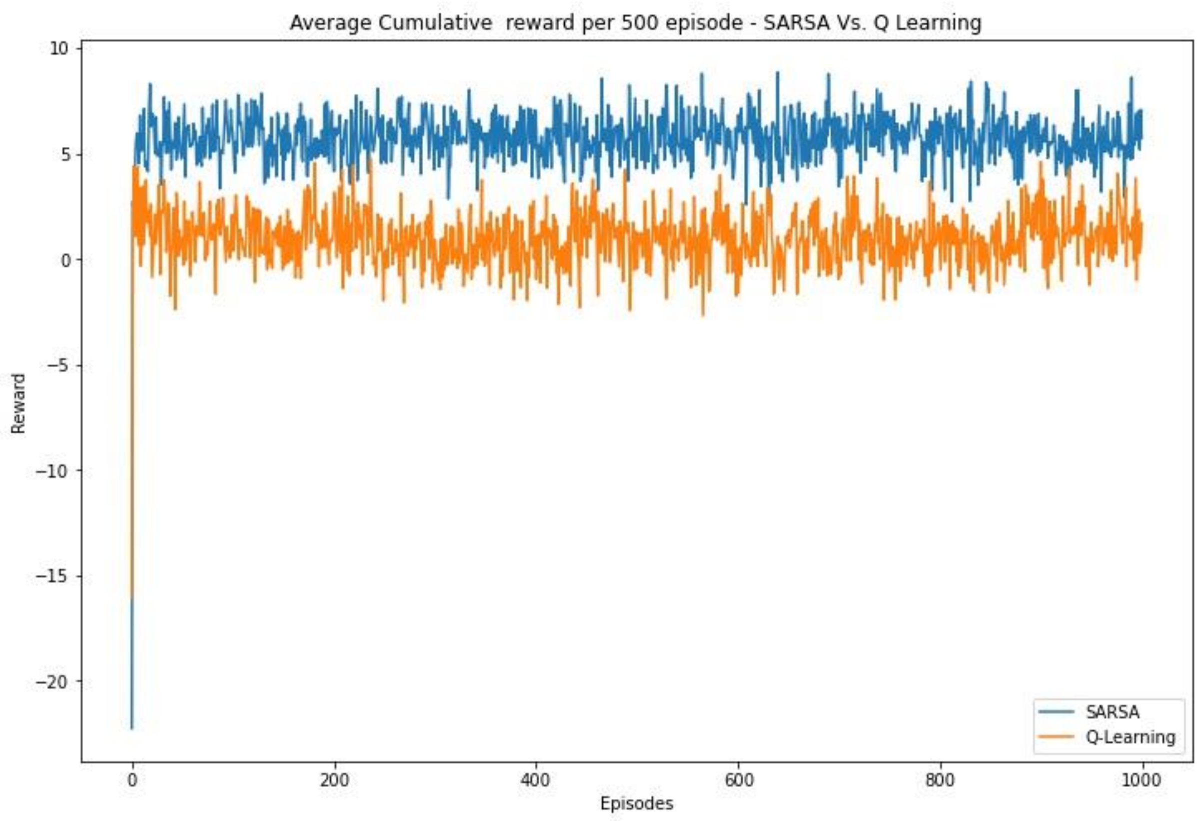


Fig 10. Average Cumulative reward per 500 episodes between SARSA and Q-Learning with Reward = +30 for picking up the passenger.

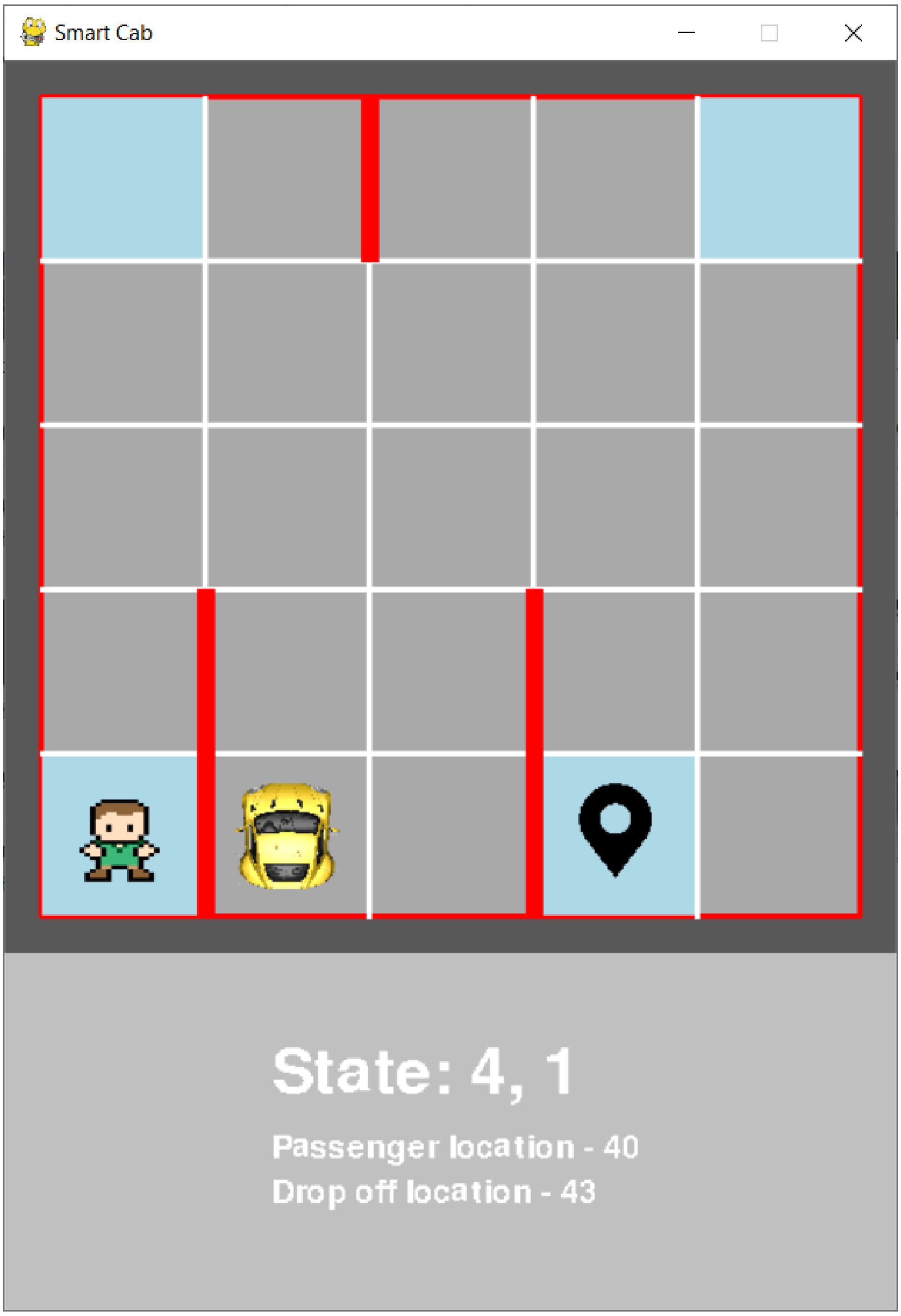
CONCLUSION

- Both are excellent approaches for Reinforcement Learning problems [16].
- Q-Learning learns optimal policy, it takes the shortest path [16].
- SARSA learns “near” optimal policy and takes a longer and safer path [16].

FUTURE WORK

1. We can try different hyperparameters, alpha and gamma.
2. We can also use different reward function.
3. And we also have an option to go more into inverse RL.

SCREENSHOTS

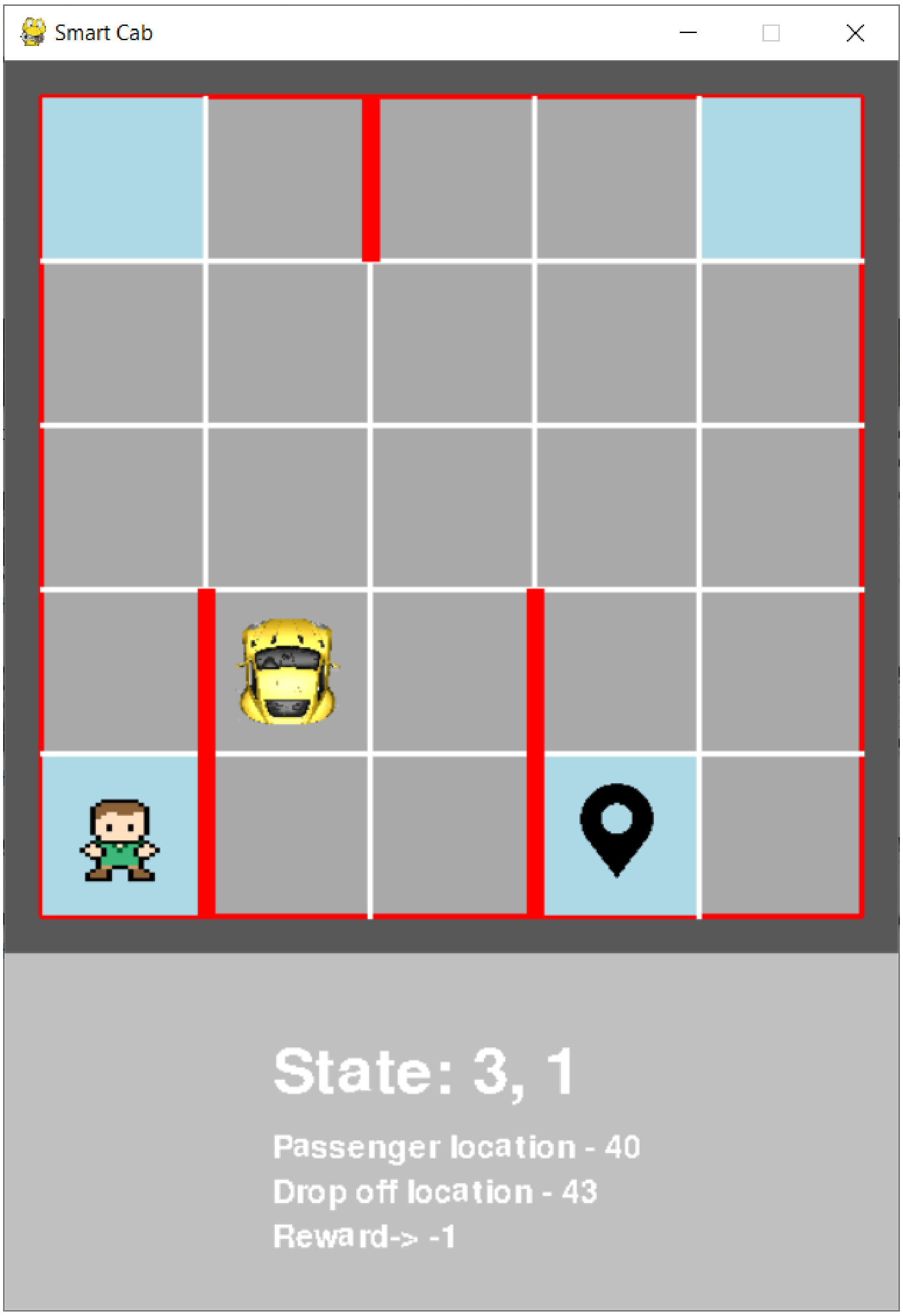


State: 4, 1

Passenger location - 40

Drop off location - 43

Fig 11. At the initial step, the game will randomly initialize the pickup and drop off position on the grid. The agent will be initialized at location [4, 1].



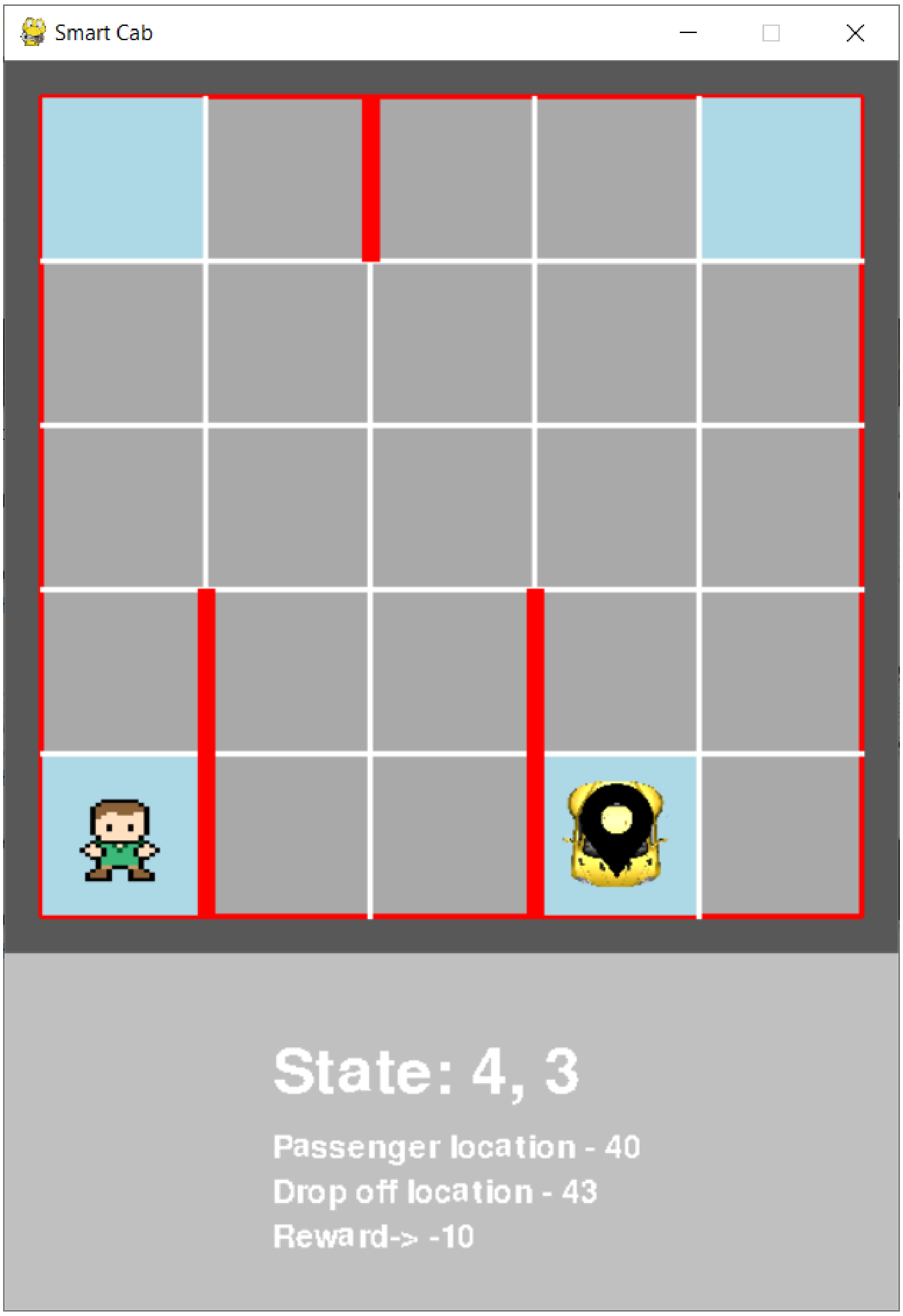
State: 3, 1

Passenger location - 40

Drop off location - 43

Reward-> -1

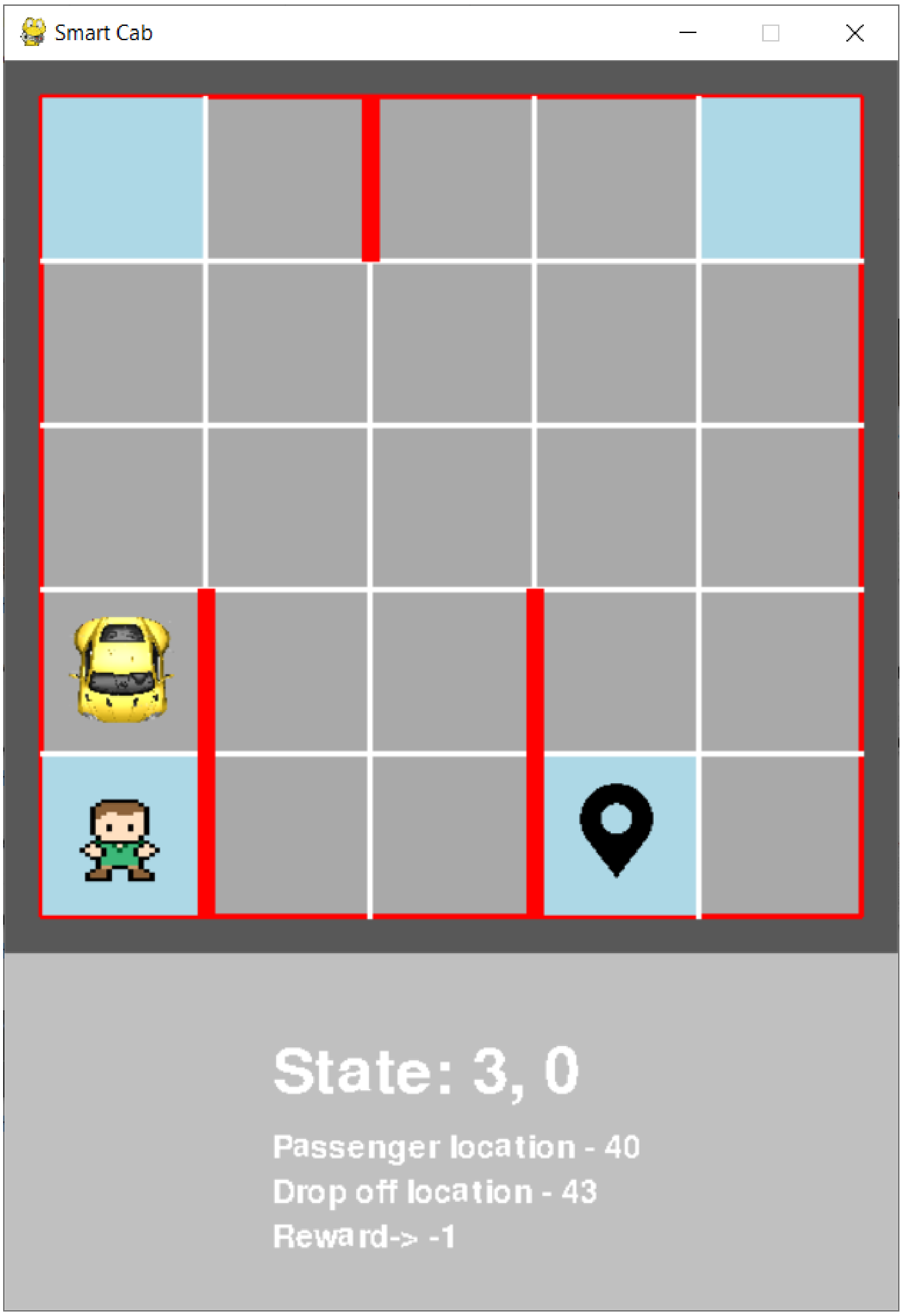
Fig 12. When we press the keyboard key, "UP", the agent will go one step UP, and it will incur a negative reward of -1. The state of the car changes accordingly.



State: 4, 3

Passenger location - 40
Drop off location - 43
Reward-> -10

Fig 13. The Agent wants to pick up the passenger. If it tries to pick up from wrong position, like here it is trying to pick up the passenger from the drop off location, then I would incur a negative reward of -10.



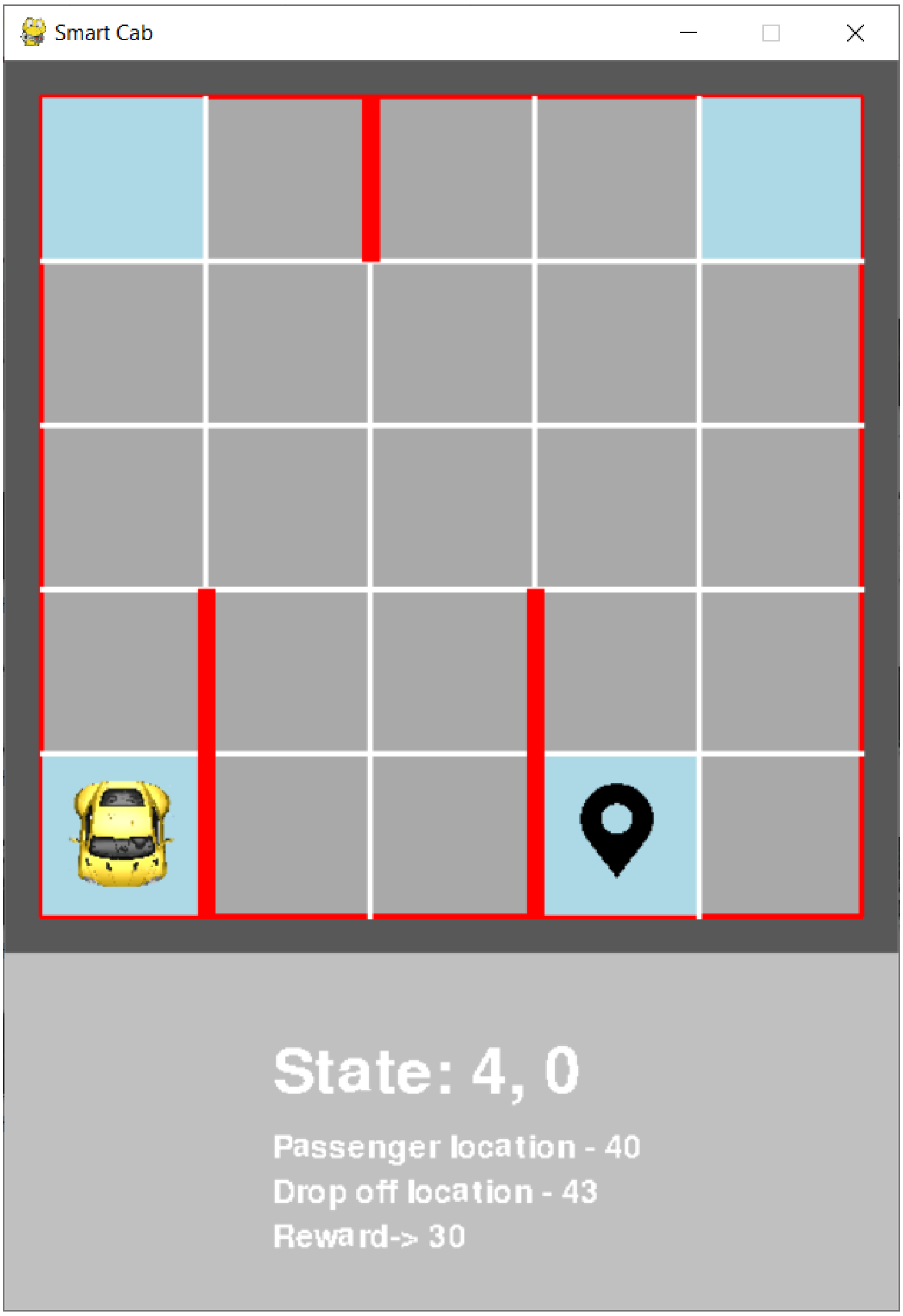
State: 3, 0

Passenger location - 40

Drop off location - 43

Reward-> -1

Fig 14. After this next step, the agent will pick up the passenger, if the next action is "DOWN".



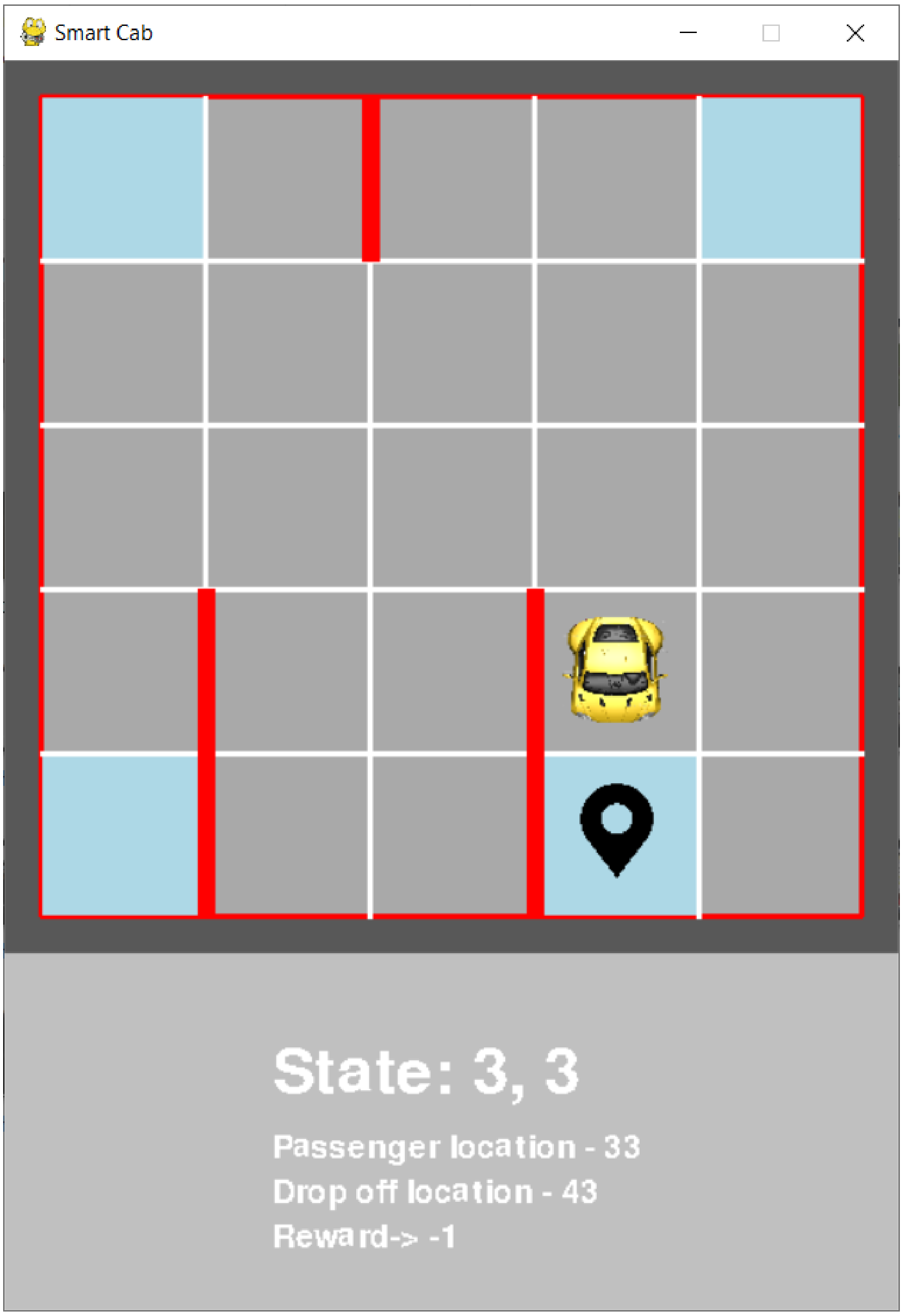
State: 4, 0

Passenger location - 40

Drop off location - 43

Reward-> 30

Fig 15. The agent picked up the passenger, and it will receive a reward of +30. Now the location of the passenger also changes along with the cab.



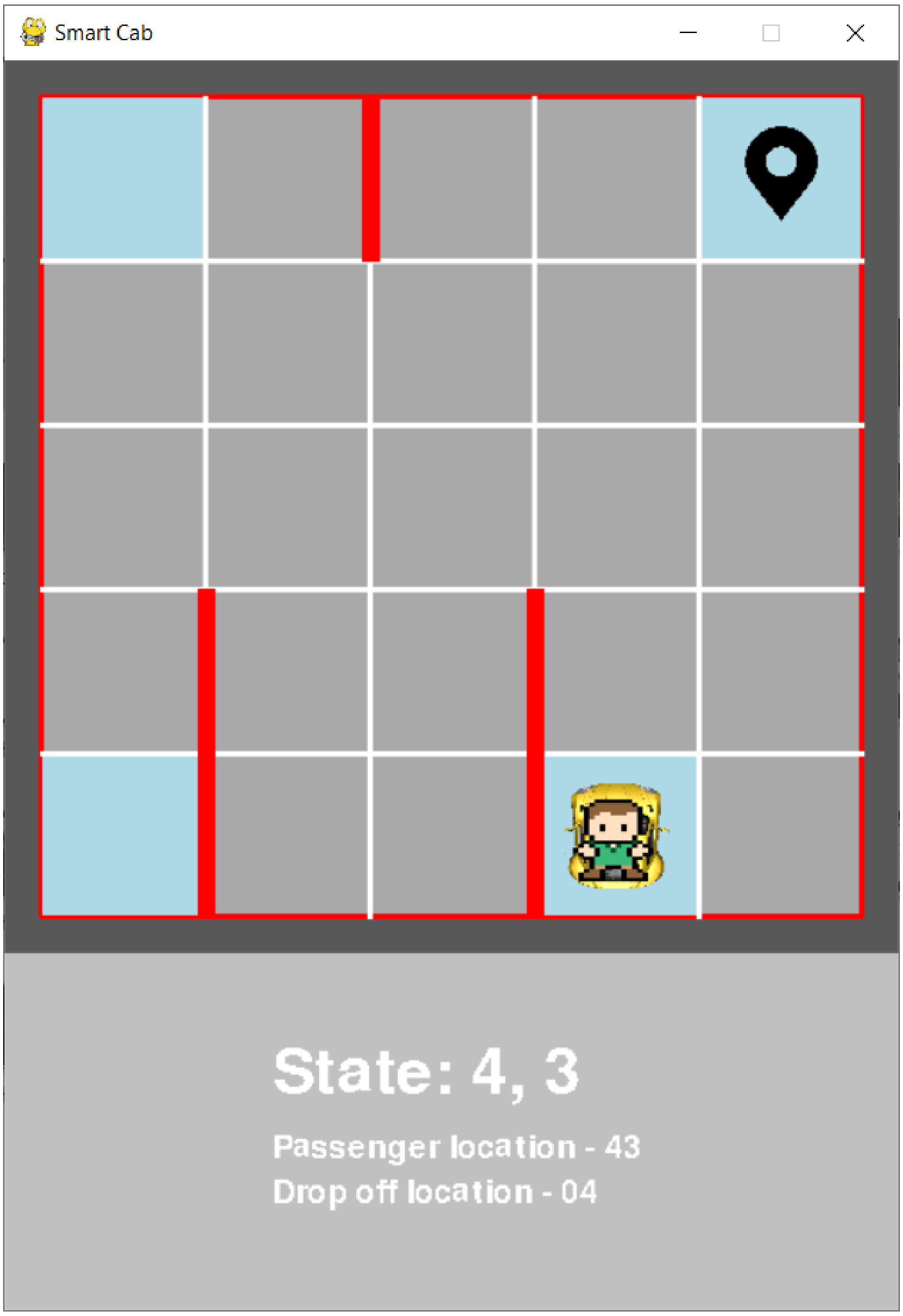
State: 3, 3

Passenger location - 33

Drop off location - 43

Reward-> -1

Fig 16 The cab has picked up the passenger and is driving towards the drop off location. If the next action is "DOWN" then it will drop off the passenger.



State: 4, 3

Passenger location - 43

Drop off location - 04

Fig 17 The cab has dropped off the passenger and received a reward of +20. As soon as it drops off the passenger, a new game starts, and new random locations of pick up and drop off are instantiated.

REFERENCES

1. https://www.101computing.net/getting-started-with-pygame/
2. http://www.pygame.org/wiki/RotateCenter?parent=CookBook
3. https://gym.openai.com/envs/Taxi-v2/
4. https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/
5. https://github.com/openai/gym/blob/master/gym/envs/toy_text/taxi.py
6. https://aakash94.github.io/Reward-Based-Epsilon-Decay/
7. https://stackoverflow.com/questions/53198503/epsilon-and-learning-rate-decay-in-epsilon-greedy-q-learning
8. https://wiki.pathmind.com/deep-reinforcement-learning
9. https://en.wikipedia.org/wiki/Model-free_(reinforcement_learning)
10. https://towardsdatascience.com/reinforcement-learning-temporal-difference-sarsa-q-learning-expected-sarsa-on-python-9fecfda7467e
11. https://medium.com/@violante.andre/simple-reinforcement-learning-temporal-difference-learning-e883ea0d65b0
12. https://towardsdatascience.com/reinforcement-learning-temporal-difference-sarsa-q-learning-expected-sarsa-on-python-9fecfda7467e
13. https://stats.stackexchange.com/questions/184657/what-is-the-difference-between-off-policy-and-on-policy-learning/376830#376830?newreg=703c24a8ebae4e75873f87dbd271717f
14. https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/
15. https://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html
16. https://towardsdatascience.com/intro-to-reinforcement-learning-temporal-difference-learning-sarsa-vs-q-learning-8b4184bb4978
17. https://datascience.stackexchange.com/questions/9832/what-is-the-q-function-and-what-is-the-v-function-in-reinforcement-learning#:~:text=Q%CF%80(s%2Ca)%20is%20the%20action%2Dvalue,policy%20%CF%80%2C%20taking%20action%20a.
18. http://www.scholarpedia.org/article/Temporal_difference_learning#:~:text=Temporal%20difference%20(TD)%20learning%20is,to%20drive%20the%20learning%20process.