Assignment 5

Reinforcement Learning Analysis: Environment Navigation via Q-Learning

Theodore Fitch

Department of Data Analytics, University of Maryland Global Campus

DATA 640: Predictive Modeling

Dr. Steven Knode

March 26th, Spring 2024

Introduction:

This brief report covers the analysis made by Sayak (2019) which demonstrates Q-Learning, a subset of Reinforcement Learning (RL), via generating a robot that learns to navigate a guitar manufacturing facility. As previously discussed (Fitch, 2024a), Q-Learning is a model free form of RL which keeps a table of rewards associated with certain actions and teaches and agent to learn the optimal actions to take from each state in order to maximize cumulative rewards (which is an MDP). Definitions of common terms used through the report are defined in Table 2. The factory is defined as a 3x3 grid with defined walls and the robot can only navigate horizontally and vertically (Figure 1). Parameters of the code were explored such as training iteration number, Alpha, Gamma, different routes, and expanding the environment.

Overview of Q-Learning Code:

A brief overview of the code will be discussed (full code can be seen at: (Sayak, 2019)). Numpy is imported as the only library, and the hyperparameters are set: Alpha (α) the learning rate and Gamma ($\boldsymbol{\gamma}$) the discount factor. Next, the states of the environment and the actions are set (arrays of 0-8 each). The rewards table is defined where a reward of 1 is given for a possible action, and a 0 given for an impossible action (accounting for proximity and walls). Then, the dictionary of location_to_state is inversely flipped to state_to_location to recall the reverse value as needed. Next, the primary function is defined, get_optimal_route such that the agent is trained to find the optimal route between 2 given states. The rewards table is copied to a new table and the ending state is given a reward of 999 so the agent learns this state is the ultimate goal. A matrix of Q-values (which represent expected future rewards for taking an action in a certain state) is then generated (beginning with all zeroes). The Q-Learning process can then begin with 1,000 iterations where it randomly selects a current state and a next state of all available actions. It computes the Q-value for the current state and action using the Bellman Equation and the TD

(Temporal Difference) which represents the discrepancy between the current state-action pair and the sum of the immediate rewards plus future expected rewards (Sayak, 2019). The route is initialized as the starting location, and the next_location is set as the starting location to begin. With all the needed tools defined, a final loop is generated that runs as long as the next_location does not equal the end_location. The loop looks at the starting location, finds the highest Q-value associated with that state, takes that action, updates its current state, and iterates the loop. Finally, the primary function is closed (get_optimal_route) and the action to find the optimal route using that function from L9 to L1 is requested. This returns: ['L9', 'L8', 'L5', 'L2', 'L1']

Discussion:

The printout returned from the code for running function from L9 to L1 returns: ['L9', 'L8', 'L5', 'L2', 'L1'](Question 3; Figure 2; Table 1). This is the most efficient route as it only takes 4 actions to arrive (L9 to L8, L8 to L5, L5 to L2, and L2 to L1)(Figure 6). The agent could attempt to go into the walled off corners (L4 and L6) or it could have moved forward, and then backward in any allowable direction. However, it wouldn't do this as the Q-values associated with those actions would be too low. Once the agent trains in the 1,000-iteration loop, it would have enough data to know the Q-values associated with each action-state pair. Thus, it would know the best action starting from L9 would be to move to L8 (as this is the only allowable action). However, at L8 where there are options (L7, L5, and L9), the Q-value for L5 would be the highest and thus the agent would choose to move there. In order to see the effects of α (learning rate) and $\boldsymbol{\gamma}$ (discount factor), the code was run using 3 variations: 1. α = 0.05 $\boldsymbol{\gamma}$ = 0.05 (Figure *3*); 2. α = 0.05 $\boldsymbol{\gamma}$ = 0.9 (Figure *4*); 3. α = 0.75 $\boldsymbol{\gamma}$ = 0.05 (Figure *5*)(Question 5; Table *1*). Alpha specifically denotes how much new information overrides old information. Too high of an alpha may cause divergence while too low may cause the agent to be too slow to learn. Gamma denotes how the agent considers immediate rewards versus future (0=immediate only, 1=future

only)(Sayak, 2019). The Q-values reflect this. When only gamma is low (Figure *5*), only the last two action Q-values are high (1,054 and 54); all other actions are $\leq$ 4 even for the correct actions. This demonstrates how a low gamma forces the agent to only focus on immediate rewards since it doesn't think about the route to arrive at L1, only the rewards it receives immediately once it is there. In a more complicated environment, this might cause the agent to take more inefficient routes. In contrast, having only a low alpha causes the agent to learn slower causing the Q-values to be lower but have similar proportions as the 0.75 alpha output (Figure *4*). If there wasn't much computing power and only 50 iterations could be used to train the agent, then this could cause issues where the agent does not learn well. We see having both a low alpha and gamma causes both trends to converge: all Q-values are much lower, and the values for the last 2 actions are significantly higher than all the rest (830 and 35, where all others are 1-2)(Figure *3*). While the step count of the while loop was 4 for each of these, this may not be the cause in a more complicated environment. This clearly shows how an elevated alpha and gamma are important for this analysis as they help ensure the agent learns quickly and doesn't focus too much on immediate reward; thus, this causes the Q-values to be higher in general and have a better linear relationship instead of exponential jumps for immediate rewards.

The code was slightly edited in order to show the number of times the while loop (in the get_optimal_route function) was run in order to find the optimal route (Question 6; Table 1). The code output 4 as the number of while loop iterations which corresponds to the number of actions the agent needs to take to move from L9 to L1. The loop only executes while the starting location of that loop does not equal the ending location. As such, it executes from L9 to L8, L8 to L5, L5 to L2, and L2 to L1 for a total of 4 actions (Figure 6, Figure 7). The code was then adjusted from the baseline to change the number of iterations in which the agent learns

(Question 7; Table 1). The agent was taught using 50, 200, and 1,000 iterations of training; the optimal path stayed the same but the Q-values changed significantly. The array of values should be read where rows are *from* and columns are *to*. For example, using the 1,000-iteration array, the 5th row represents starting *from* state 5 and moving *to* either state 2 or 8 (since those are the only possible moves). All zeroes represent impossible moves. The Q-values are 7,974 and 6,458 for state 2 and 8 respectively showing that there's a higher cumulative reward to move to state 2 from state 5. The differences between Q-values using 50 iterations versus 1,000 are on average 7,000. Using 50 iterations of training still yields the correct optimal path; there is never a wrong path which would be chosen because the Q-values for the correct path are larger. There is a large differential between Q-values and this would need to be accounted for if there was a more complicated environment like one with ambiguity. This scenario would likely necessitate more training so that the Q-values for the right path are always highest so the agent acts correctly.

The code was then run using the reverse starting and ending states, from L1 to L9 (Question 8; Table 1; Figure 9). This code failed at first. Upon debugging, it was found this was caused since the original rewards array provided for this analysis incorrectly was missing a 1 from row 2, column 5. Having the 1 there would indicate to the agent that the action moving from state 2 to state 5 was an allowable action. Once a 1 was added here, the code ran correctly showing the correct optimal route: L1, L2, L5, L8, L9 (the same as from L9 to L1 but in reverse). The while loop similarly took 4 iterations, and the Q-values were similar to the reverse route (a penultimate cumulative reward of 9,817 versus 9,863 respectively). As a final test, the environment was adjusted such that there is a new state, L10 that is only accessible by L9 (Question 9; Table *1*; Figure 10). The agent was then told to find the optimal route from L10 to L1 which yielded the expected result which took 5 iterations of the while loop since there would

be a total of 5 actions (L10 to L9, and the 4 actions from L9 to L1)(Figure 11). The Q-values were similar to the L9 to L1 values (with the penultimate cumulative reward for this being the same as L9 to L1, 9,817). This penultimate reward is observed in the Q-value array in column 4, row 4 as this indicates the agent had a starting position that was the same as the ending position (L4) which terminated the while loop.

Conclusion:

In summary, a 3x3 grid was used as a simulated environment in which an agent was taught to navigate. Six questions were addressed showing 1. Why the optimal route is L9, L8, L5, L2, L1; 2. The effects of alpha and gamma on the agent; 3. How many times the while loop iterates; 4. The effects of using more or less iterations of training; 5. The effects of using the reverse route; 6. The effects of expanding the environment and learning new routes. While the hyperparameters can be lowered in this analysis while still obtaining the correct result, these should be adjusted conservatively in more complex production environments. It is ideal to use less training iterations to use less computing power; however, in complicated environments, an agent may need more training to learn effectively. Similarly, gamma and alpha could be lowered while still obtaining the right answer for this analysis, but they should be adjusted sparingly. As noted by Berti-Équille (2019), these should be changed appropriately to each problem where some agents may thrive on lower alphas/gammas while others may need higher.

References:

Berti-Équille, L. (2019). Reinforcement learning for data preparation with active reward learning. In *International Conference on Internet Science* (pp. 121-132). Cham: Springer International Publishing.

Fitch, T. (2024a). *RL-Tic-Tac-Toe*. GitHub. https://github.com/Capadetated/RL-Tic-Tac-Toe

Fitch, T. (2024b). *RL-Environment-Navigation*. GitHub. https://github.com/Capadetated/RL-Environment-Navigation

FreeCodeCamp. (2018). *An introduction to Q-learning: Reinforcement learning*. FreeCodeCamp. https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/

Sayak, P. (2019). FloydHub-Q-Learning-Blog. GitHub. https://github.com/sayakpaul/FloydHub-Q-Learning-Blog

Sutton, R. & Barto, A. (2017). Reinforcement Learning: An Introduction. The MIT Press. http://incompleteideas.net/book/bookdraft2017nov5.pdf

Appendix:

Index	Question #	Question
1	**Question 3**	If you are successful running the code, the result of print(get_optimal_route('L9', 'L1')) will be ['L9', 'L8', 'L5', 'L2', 'L1']. Explain why this is the case.
2	**Question 5**	Run with different levels of hyperparameters gamma and alpha **Discuss the key results**
3	**Question 6**	When you run **print(get_optimal_route('L9', 'L1'))** how many times does the while loop in the **get_optimal_route()** get executed? Why it is this number?
4	**Question 7**	The Q-learning process for loop in get_optimal_route() performs 1000 iterations. Do you think these many iterations are required? **Try with 50. Try with 200.** Explain what happens and why you think it is happening.
5	**Question 8**	Now try to run the reverse path print(get_optimal_route('L1', 'L9')) ○ The code will most likely keep iterating without returning. Why do you think this is? Hint: Look at the rewards for L2. Can it get to L5 from L2? ○ Make the change so that this case works; see the previous hint. Show the results you obtain.
6	**Question 9**	The existing code has 9 states L1-L9. Add a tenth state L10, where it is only possible to go from L10 to L9 and vice versa as depicted in Figure 1. ○ Make all of the required changes to the code. ○ Run **print(get_optimal_route('L10', 'L1'))** and show your results and explain if they are as expected or not. ○ Run **print(get_optimal_route('L10', 'L4'))** and show your results and explain if they are as expected or not.

Table 1. Overview of the 6 questions needed to be answered via this report. Each question is referenced in the text of the discussion.

Term	Definition
Agent	The learner or decision-maker interacting with the environment.
Environment	The external system with which the agent interacts.
Actions	The set of possible decisions or moves that the agent can take.
State	The current situation or configuration of the environment.
Rewards	Numeric feedback from the environment indicating the desirability of the agent's action.
Policy	The strategy or mapping from states to actions that the agent follows to maximize cumulative reward.
Value Function	An estimate of the expected cumulative reward that an agent can obtain from a given state or state-action pair.
Q-Learning	A model-free RL algorithm that learns a Q-value function representing the expected future rewards of taking a particular action in a given state.
Q-Value	A value of expected future rewards for the agent of taking a particular action in a given state.
Exploration vs. Exploitation	The trade-off between trying new actions to discover better strategies (Exploration) and exploiting known actions for immediate reward (Exploitation).
Alpha ($\boldsymbol{\alpha}$)	The learning rate (how much new information overwrites old information).
Gamma ($\boldsymbol{\gamma}$)	Discount factor (determines the importance of future rewards).
Bellman Equation	A recursive relationship that expresses the value of a decision problem as the sum of its immediate reward and the value of the best possible successor state: $V(s)=\max_a(R(s,a)+\gamma V(s'))V(s)=\max_a(R(s,a)+\gamma V(s'))$
Markov Decision Processes (MDP)	A Markov Decision Process (MDP) is a discrete time stochastic control process. It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker.
Temporal Difference (TD)	A method to update the estimated value of a state or action based on the difference between the observed and predicted rewards experienced at successive time steps.

Table 2. Definitions for common reinforcement learning terms needed to understand this analysis (Sutton and Barto, 2017)(Sayak, 2019).

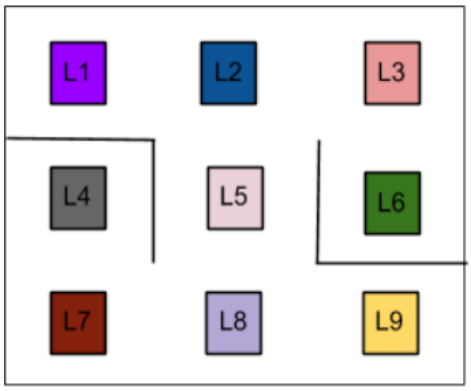

Figure 1. The defined environment of the guitar manufacturing facility. The lines represent walls such that L4 is only accessible by L7, and L7 is only accessible by L4 and L8. Question 3.

```
    while(next_location != end_location):
        # Fetch the starting state
        starting_state = location_to_state[start_location]
        # Fetch the highest Q-value pertaining to starting state
        next_state = np.argmax(Q[starting_state,])
        # We got the index of the next state. But we need the corresponding letter.
        next_location = state_to_location[next_state]
        route.append(next_location)
        # Update the starting location for the next iteration
        start_location = next_location

    return route

print(get_optimal_route('L9', 'L1'))
```

```
['L9', 'L8', 'L5', 'L2', 'L1']
```

Figure 2. Final piece of Q-Learning code and printout of get_optimal_route function using a starting location of L9 and ending location of L1. Question 3.

```
1,000 Iterations
[[830.   2.   0.   0.   0.   0.   0.   0.   0.]
 [ 35.   0.   1.   0.   0.   0.   0.   0.   0.]
 [  0.   2.   0.   0.   0.   1.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   1.   0.   0.]
 [  0.   2.   0.   0.   0.   0.   0.   1.   0.]
 [  0.   0.   1.   0.   0.   0.   0.   0.   0.]
 [  0.   0.   0.   1.   0.   0.   0.   1.   0.]
 [  0.   0.   0.   0.   1.   0.   1.   0.   1.]
 [  0.   0.   0.   0.   0.   0.   0.   1.   0.]]
(['L9', 'L8', 'L5', 'L2', 'L1'], 4)
```

Figure 3. The output Q-value array, while loop iteration number, and optimal route from L9 to L1 using 1,000 iterations of training and an α = 0.05 and $\boldsymbol{\gamma}$ = 0.05. The Q-values are much lower than α = 0.75 and $\boldsymbol{\gamma}$ = 0.9. Question 5.

```
1,000 Iterations
[[2331. 1029.    0.    0.    0.    0.    0.    0.    0.]
 [1592.    0.  413.    0.    0.    0.    0.    0.    0.]
 [   0.  894.    0.    0.    0.  291.    0.    0.    0.]
 [   0.    0.    0.    0.    0.    0.   48.    0.    0.]
 [   0.  922.    0.    0.    0.    0.    0.  108.    0.]
 [   0.    0.  628.    0.    0.    0.    0.    0.    0.]
 [   0.    0.    0.   16.    0.    0.    0.   87.    0.]
 [   0.    0.    0.    0.  257.    0.   33.    0.   28.]
 [   0.    0.    0.    0.    0.    0.    0.  131.    0.]]
(['L9', 'L8', 'L5', 'L2', 'L1'], 4)
```

Figure 4. The output Q-value array, while loop iteration number, and optimal route from L9 to L1 using 1,000 iterations of training and an α = 0.05 and $\boldsymbol{\gamma}$ = 0.9. The Q-values are much higher than α = 0.05 and $\boldsymbol{\gamma}$ = 0.05. Question 5.

```
1,000 Iterations
[[1052    4    0    0    0    0    0    0    0]
 [  54    0    1    0    0    0    0    0    0]
 [   0    4    0    0    0    1    0    0    0]
 [   0    0    0    0    0    0    1    0    0]
 [   0    4    0    0    0    0    0    1    0]
 [   0    0    1    0    0    0    0    0    0]
 [   0    0    0    1    0    0    0    1    0]
 [   0    0    0    0    1    0    1    0    1]
 [   0    0    0    0    0    0    0    1    0]]
(['L9', 'L8', 'L5', 'L2', 'L1'], 4)
```

Figure 5. The output Q-value array, while loop iteration number, and optimal route from L9 to L1 using 1,000 iterations of training and an α = 0.75 and γ = 0.05. The Q-values are much higher than α = 0.05 and γ = 0.05. Question 5.

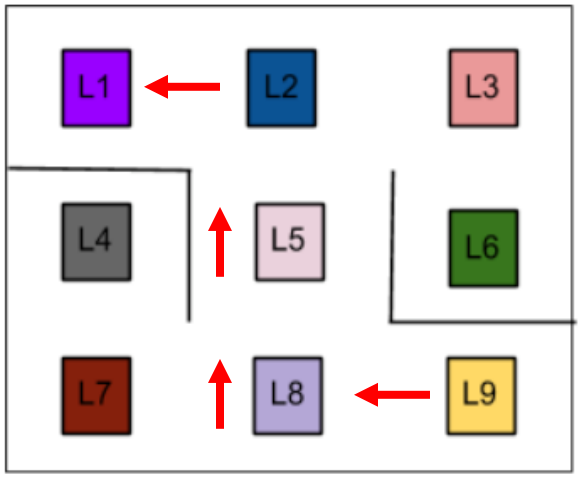

Figure 6. The most efficient route in the environment is from L9 to L1 as it only takes 4 actions to arrive (L9 to L8, L8 to L5, L5 to L2, and L2 to L1). Question 6.

```
    # Editing while loop to show iterations
    steps=0
    # We don't know about the exact number of iterations needed to reach to the
    # final location hence while loop will be a good choice for iteratiing
    while(next_location != end_location):
        # Fetch the starting state
        starting_state = location_to_state[start_location]
        # Fetch the highest Q-value pertaining to starting state
        next_state = np.argmax(Q[starting_state,])
        # We got the index of the next state. But we need the corresponding letter.
        next_location = state_to_location[next_state]
        route.append(next_location)
        # Update the starting location for the next iteration
        start_location = next_location
        # Upversion steps
        steps=steps+1

    return route,steps

print(get_optimal_route('L9', 'L1'))
```

```
(['L9', 'L8', 'L5', 'L2', 'L1'], 4)
```

Figure 7. Question 6 answer showing it took 4 iterations of the while loop in order to find the optimal route. This is because it takes 4 actions to move from state L9 to L1. Question 6.

```
50 Iterations
[[1442.  449.    0.    0.    0.    0.    0.    0.    0.]
 [ 665.    0.  233.    0.    0.    0.    0.    0.    0.]
 [   0.  343.    0.    0.    0.  271.    0.    0.    0.]
 [   0.    0.    0.    0.    0.    0.    2.    0.    0.]
 [   0.  598.    0.    0.    0.    0.    0.  158.    0.]
 [   0.    0.  308.    0.    0.    0.    0.    0.    0.]
 [   0.    0.    0.    0.    0.    0.    0.  158.    0.]
 [   0.    0.    0.    0.  232.    0.  108.    0.  134.]
 [   0.    0.    0.    0.    0.    0.    0.  197.    0.]]
['L9', 'L8', 'L5', 'L2', 'L1']
200 Iterations
[[7719. 6221.    0.    0.    0.    0.    0.    0.    0.]
 [6930.    0. 5061.    0.    0.    0.    0.    0.    0.]
 [   0. 5944.    0.    0.    0. 4054.    0.    0.    0.]
 [   0.    0.    0.    0.    0.    0.  889.    0.    0.]
 [   0. 5622.    0.    0.    0.    0.    0. 2866.    0.]
 [   0.    0. 4557.    0.    0.    0.    0.    0.    0.]
 [   0.    0.    0.  732.    0.    0.    0. 4310.    0.]
 [   0.    0.    0.    0. 5034.    0.  890.    0. 3119.]
 [   0.    0.    0.    0.    0.    0.    0. 3871.    0.]]
['L9', 'L8', 'L5', 'L2', 'L1']

 1,000 Iterations
 [[9863. 7960.    0.    0.    0.    0.    0.    0.    0.]
  [8873.    0. 7125.    0.    0.    0.    0.    0.    0.]
  [   0. 7915.    0.    0.    0. 6413.    0.    0.    0.]
  [   0.    0.    0.    0.    0.    0. 5770.    0.    0.]
  [   0. 7974.    0.    0.    0.    0.    0. 6458.    0.]
  [   0.    0. 7125.    0.    0.    0.    0.    0.    0.]
  [   0.    0.    0. 5194.    0.    0.    0. 6410.    0.]
  [   0.    0.    0.    0. 7176.    0. 5756.    0. 5766.]
  [   0.    0.    0.    0.    0.    0.    0. 6458.    0.]]
 ['L9', 'L8', 'L5', 'L2', 'L1']
```

Figure 8. Q-values array displayed for using 50, 200, and 1,000 training iterations for the Q-Learning process in get_optimal_route. Question 7.

	L9 to L8	**L8 to L5**	**L5 to L2**	**L2 to L1**	**L1 final**
50	197	232	598	655	1,442
200	3,871	5,034	5,622	6,930	7,719
1,000	6,458	7,176	7,974	8,873	9,863

Table 3. The Q-values displayed from Figure 8 for each of the above actions and for each repetition of training using 50, 200, and 1,000 iterations. Question 7.

```
    # Rounding off the Q-values
    Q = np.round(Q)
    print("1,000 Iterations")
    print(Q)
    return route,steps

print(get_optimal_route('L1', 'L9'))
```

```
1,000 Iterations
[[   0. 6412.    0.    0.    0.    0.    0.    0.    0.]
 [5659.    0. 5751.    0. 7123.    0.    0.    0.    0.]
 [   0. 6407.    0.    0.    0. 5179.    0.    0.    0.]
 [   0.    0.    0.    0.    0.    0. 7106.    0.    0.]
 [   0. 6408.    0.    0.    0.    0.    0. 7920.    0.]
 [   0.    0. 5767.    0.    0.    0.    0.    0.    0.]
 [   0.    0.    0. 6396.    0.    0.    0. 7894.    0.]
 [   0.    0.    0.    0. 7104.    0. 7105.    0. 8834.]
 [   0.    0.    0.    0.    0.    0.    0. 7919. 9817.]]
(['L1', 'L2', 'L5', 'L8', 'L9'], 4)
```

Figure 9. The output Q-value array, while loop iteration number, and optimal route from L1 to L9 using 1,000 iterations of training. The Q-values are similar to route L9 to L1. Question 8.

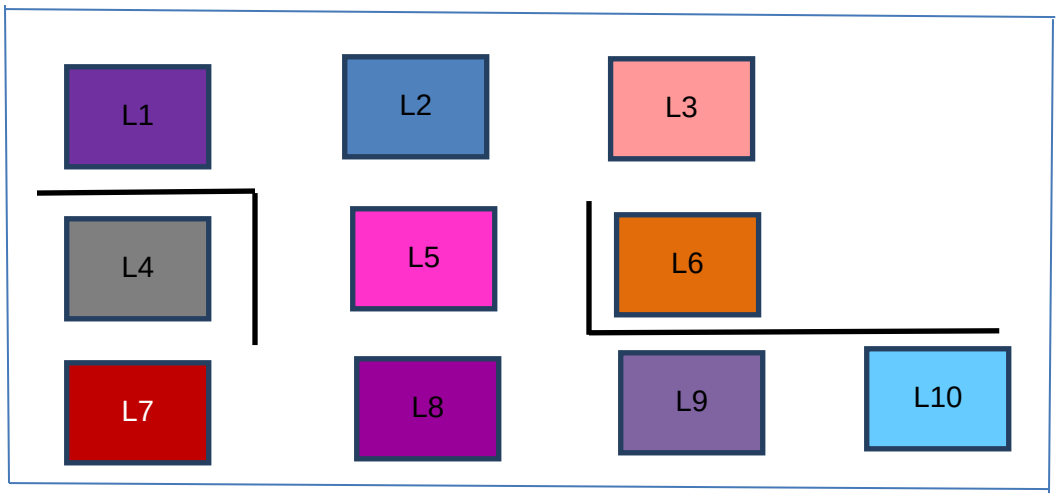

Figure 10. The adjusted environment of the guitar manufacturing facility such that there is a new state, L10, which is only accessible by L9. Question 9.

```
    # Rounding off the Q-values
    Q = np.round(Q)
    print("1,000 Iterations; L10 to L1")
    print(Q)
    return route,steps

print(get_optimal_route('L10', 'L1'))
```

```
1,000 Iterations; L10 to L1
[[9817. 7889.    0.    0.    0.    0.    0.    0.    0.]
 [8767.    0. 6940.    0. 7053.    0.    0.    0.    0.]
 [   0. 7826.    0.    0.    0. 6333.    0.    0.    0.]
 [   0.    0.    0.    0.    0.    0. 5622.    0.    0.]
 [   0. 7881.    0.    0.    0.    0.    0. 6251.    0.]
 [   0.    0. 7043.    0.    0.    0.    0.    0.    0.]
 [   0.    0.    0. 4900.    0.    0.    0. 6246.    0.]
 [   0.    0.    0.    0. 7059.    0. 5605.    0. 5602.]
 [   0.    0.    0.    0.    0.    0.    0. 6258.    0.]
 [   0.    0.    0.    0.    0.    0.    0.    0. 5632.]]
(['L10', 'L9', 'L8', 'L5', 'L2', 'L1'], 5)
```

Figure 11. The output Q-value array, while loop iteration number, and optimal route from L10 to L1 using 1,000 iterations of training. The Q-values are similar to route L9 to L1/L1 to L9. It took 5 total steps as there is an additional step from L10 to L9. Question 9.

```
    # Rounding off the Q-values
    Q = np.round(Q)
    print("1,000 Iterations; L10 to L4")
    print(Q)
    return route,steps

print(get_optimal_route('L10', 'L4'))
```

```
1,000 Iterations; L10 to L4
[[   0. 5753.    0.    0.    0.    0.    0.    0.    0.]
 [5175.    0. 5173.    0. 6392.    0.    0.    0.    0.]
 [   0. 5753.    0.    0.    0. 4651.    0.    0.    0.]
 [   0.    0.    0. 9842.    0.    0. 7928.    0.    0.]
 [   0. 5740.    0.    0.    0.    0. 7158.    0.    0.]
 [   0.    0. 5178.    0.    0.    0.    0.    0.    0.]
 [   0.    0.    0. 8857.    0.    0. 7156.    0.    0.]
 [   0.    0.    0.    0. 6413.    0. 7952.    0. 6441.    0.]
 [   0.    0.    0.    0.    0.    0.    0. 7157.    0.    0.]
 [   0.    0.    0.    0.    0.    0.    0.    0. 6442.    0.]]
(['L10', 'L9', 'L8', 'L7', 'L4'], 4)
```

Figure 12. The output Q-value array, while loop iteration number, and optimal route from L10 to L4 using 1,000 iterations of training. The Q-values are similar to route L9 to L1. It took 4 total steps as there are only 4 steps from L10 to L4 (L10 to L9, L9 to L8, L8 to L7, L7 to L4). The penultimate cumulative reward observed in column 4, row 4 of 9,842. Question 9.

Figure 13. The code used to answer question 9 is seen below between the // marks.

//

```
# Code based upon work by Sayak, P. (2019). FloydHub-Q-Learning-Blog. GitHub. https://github.com/sayakpaul/FloydHub-Q-Learning-Blog
# Generated for DATA 640 for Dr. Steven Knode by Theodore Fitch
# Last updated 26MAR24

# Import only numpy
import numpy as np

# Initialize parameters
gamma = 0.9 # Discount factor. Previously adjusted to 0.05 to observe effects.
alpha = 0.75 # Learning rate. Previously adjusted to 0.05 to observe effects.

# Define the states. 10th state added
location_to_state = {
    'L1' : 0,
    'L2' : 1,
    'L3' : 2,
    'L4' : 3,
    'L5' : 4,
    'L6' : 5,
    'L7' : 6,
    'L8' : 7,
    'L9' : 8,
    'L10' : 9
```

```
}

# Define the actions. 10th action added
actions = [0,1,2,3,4,5,6,7,8,9]

# Define the rewards. A 10th column and row added. L10 is only accessible
via L9
rewards = np.array([[0,1,0,0,0,0,0,0,0,0],
              [1,0,1,0,1,0,0,0,0,0],
              [0,1,0,0,0,1,0,0,0,0],
              [0,0,0,0,0,0,1,0,0,0],
              [0,1,0,0,0,0,0,1,0,0],
              [0,0,1,0,0,0,0,0,0,0],
              [0,0,0,1,0,0,0,1,0,0],
              [0,0,0,0,1,0,1,0,1,0],
              [0,0,0,0,0,0,0,1,0,1],
              [0,0,0,0,0,0,0,0,1,0]])

# Maps indices to locations
state_to_location = dict((state,location) for location,state in
location_to_state.items())

def get_optimal_route(start_location,end_location):
    # Copy the rewards matrix to new Matrix
    rewards_new = np.copy(rewards)
    # Get the ending state corresponding to the ending location as given
    ending_state = location_to_state[end_location]
    # With the above information automatically set the priority of the
given ending state to the highest one
    rewards_new[ending_state,ending_state] = 999

    # -----------Q-Learning algorithm-----------

    # Initializing Q-Values. Array expanded to 10x10
    Q = np.array(np.zeros([10,10]))

    # Q-Learning process
    for i in range(1000):
        # Pick up a state randomly
        current_state = np.random.randint(0,10) # Python excludes the
upper bound
        # For traversing through the neighbor locations in the maze
        playable_actions = []
        # Iterate through the new rewards matrix and get the actions > 0
        for j in range(9):
```

```
            if rewards_new[current_state,j] > 0:
                playable_actions.append(j)
        # Pick an action randomly from the list of playable actions
leading us to the next state
        next_state = np.random.choice(playable_actions)
        # Compute the temporal difference
        # The action here exactly refers to going to the next state
        TD = rewards_new[current_state,next_state] + gamma * Q[next_state,
np.argmax(Q[next_state,])] - Q[current_state,next_state]
        # Update the Q-Value using the Bellman equation
        Q[current_state,next_state] += alpha * TD

    # Initialize the optimal route with the starting location
    route = [start_location]
    # We do not know about the next location yet, so initialize with the
value of starting location
    next_location = start_location
    # Editing while loop to show iterations
    steps=0
    # We don't know about the exact number of iterations needed to reach
to the final location hence while loop will be a good choice for
iteratiing
    while(next_location != end_location):
        # Fetch the starting state
        starting_state = location_to_state[start_location]
        # Fetch the highest Q-value pertaining to starting state
        next_state = np.argmax(Q[starting_state,])
        # We got the index of the next state. But we need the
corresponding letter.
        next_location = state_to_location[next_state]
        route.append(next_location)
        # Update the starting location for the next iteration
        start_location = next_location
        # Upversion steps
        steps=steps+1
     # Rounding off the Q-values
    Q = np.round(Q)
    # For question 9, alternate title L1/L4
    print("1,000 Iterations; L10 to L4")
    print(Q)
    return route,steps

# For question 9, this is alternated from L10 to L1, and L10 to L4
print(get_optimal_route('L10', 'L4'))
//
```