Cera Oh
CS541 AI
Program 3
Fall 2021
11/22/2021

Program 3 Report

Introduction and instruction how to run the program

I am turning in one C code, Q_Learning.c. Variables *N, M, alpha, gamma, and epsilon* are hardcoded and can be changed in *Q_Learning*() and *Q_Test*() functions. From the formula given in the assignment sheet, ɳ is *alpha*, ɣ is *gamma*, ε is *epsilon*. Variable *N* for both the learning and test were set to 5000. Variable *M* for both the learning and test were set to 200. Variable *alpha* is 0.2, *gamma* is 0.9, and *epsilon* is 0.1, which is represented as 100 (range 0 to 99) out of 1000 numbers picked by *rand*(). Variable *epsilon* drops 0.05 per 50 epochs for *Q_Learning*() until 0 is reached. It remains at 0.1 for *Q_Test*(). Once run, the program should automatically give you an output of episode and turn numbers, and what Robby did per turn. At the end, the program should output the *Test-Average* and *Test-Standard-Deviation*. Q_Learning() function is run first to build the Q-Table and train Robby. Once the learning is complete, *Q_Test*() is called to run the test. In *main*() function, I have commented off *print_Q*() after Q_Learning() is called. You can turn this line "on" to see the Q-Table after Robby has finished learning if you wish. You can turn on the commented out *print_Q*() after the *Q_Test*() to see the updated Q-Table after the test run as well. I have *print_TRewards*() function to print out a chart of data for Training Rewards Plot from the learning session and *print_Reward_SD*() functions to print out a chart of all total rewards and standard deviation per run of the test. Those are also commented out but feel free to use these functions.

Q_Learning() and *Q_Test*() contain *printf*() statements that allows you to see what is happening visually if you wish. You can "turn on" these commented out *printf*() statements if you wish to view 10x10 board with Robby moving every turn or if you wish to view Robby's sensor states per turn. For each step, the code prints out a 10x10 board representing the current state of the board, with letter 'c' representing cans and letter 'R' representing Robby. Robby's current state is printed below the board. I

have those commented out for now because they make the code take hours to finish running on my laptop.

Description of the Experiment

Robby is represented as a struct *agent*, which contains integer variables *Current*, *North*, *South*, *West*, and *East* to hold Robby's sensory input per turn. Integer 1 represents presence of a can, 0 represents no presence of a can or a wall, and -1 represents presence of a wall. Robby also has float variable *reward* to hold total rewards he has won. A float variable *current_r* is used to store the current reward points Robby gained after his move for Q-Table calculation purposes. Robby also stores integer variables *row* and *col* to store integers in the range of 0 to 9, representing row and column of Robby's location. This information is used to place Robby initially onto the board and to keep track of Robby's movements. Robby does not use row and column information for his sensory inputs. Robby is placed on a random spot on the board each episode. For each episode, a can is placed with 0.5 probability for each square of the 10x10 board. Robby's five movements are controlled by functions *move_N*(), *move_S*(), *move_W*(), *move_E*() and *pick_up*(), which also update the total rewards and current reward earned for Robby. Robby's sensors are updated initially and after each move by *Update_Sensors*() function.

The Q-Table is represented as 6-dimensional array of integers. First five dimensions of the array represent the sensor in the order of Current, North, South, West and East. The values that can be stored are 1 for can, 0 for no can and no wall, and -1 for wall. The five sensors together represent the state Robby is in. Adding one to a value stored gives it the index number in each dimension of the array (except for the last dimension). For example, if Robby is in a square where current sensor senses a can, north sensor senses a wall, south sensor senses nothing, west sensor senses a wall, and east sensor senses a can, these values will be stored in the first five dimensions of the array by index numbers [2][0][1][0][2]. The stored values would be 1, -1, 0, -1, 1, respectively. It is represented as [1][-1][0][-1][1]

on the Q-Table. Q-Table can be printed out using *print_Q*() function. The last dimension represents the five actions available to Robby, Move North, Move South, Move West, Move East, and Pick Up. Move North is represented as index 0, Move South is represented as index 1, Move West is represented as index 2, Move East is represented as index 3, and Pick Up is represented as index 4 in the last dimension of the array. Robby selects his action according to ε-greedy action selection. $Q(s_t, a_t)$ is updated following the formula given in our assignment sheet. The best action for current state s_t and the best action for new state s_{t+1} are chosen by looping through the all available actions index for the current state and looking for the highest Q-value. If Q-Table is empty (Q-values all 0) for the current state, Robby acts randomly. If the state actions' Q-values are not all 0 but there are multiple best actions (i.e. same Q values) to take, Robby randomly chooses between the available best actions.

Data

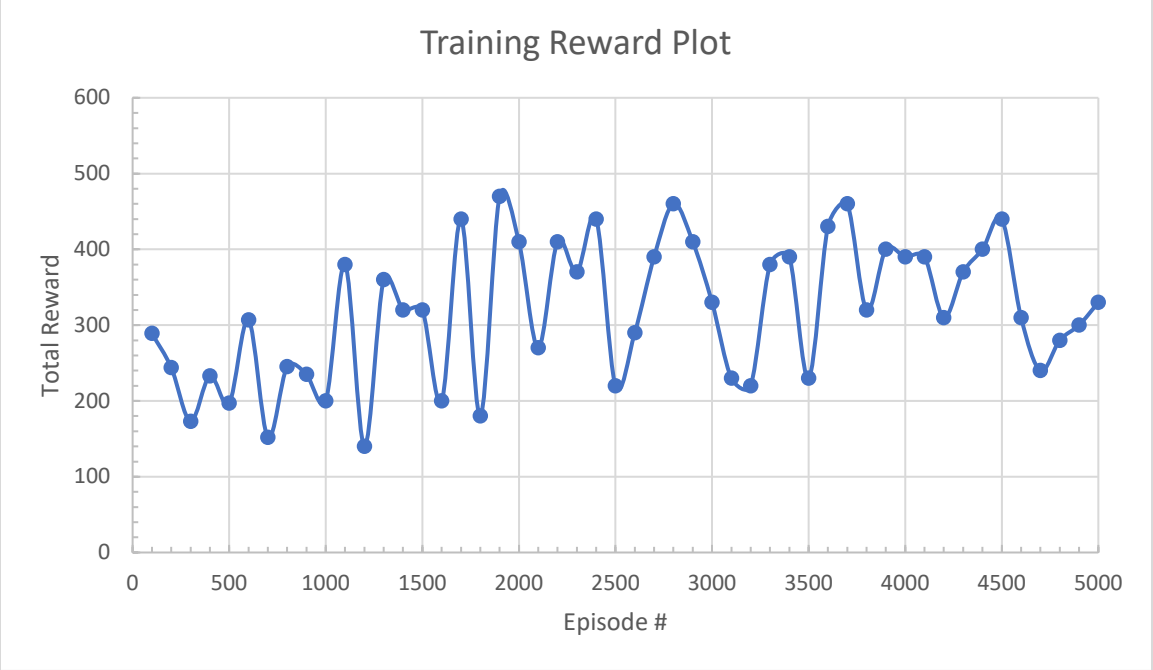
Example Test-Averages & Standard Deviations after Test Runs (5000 episodes with 200 steps):

Test Run #	Test-Average	Test-Standard-Deviation	Highest Total Reward	Lowest Total Reward
1	270.282806	69.993446	493	-44
2	272.020599	70.118515	504	-58
3	271.708008	69.254997	500	-4
4	274.450012	69.572144	483	-21
5	273.551392	69.128448	496	-2
6	269.323608	68.348366	485	-11
7	272.519012	70.093747	497	-56
8	273.566986	71.233047	480	-35
9	274.082214	69.692375	497	-35
10	272.863403	68.897949	498	-8
11	274.582397	69.917358	491	-40
12	273.214600	69.737160	492	-26
13	273.882813	70.951599	489	-17
14	274.113403	68.299622	481	-8
15	271.916992	70.417297	484	-34
16	270.706390	69.644791	490	-20
17	273.532410	68.711327	496	-17
18	277.868988	70.833870	499	-6

19	273.722992	69.981743	499	-12
20	272.898987	68.603851	481	-31

Training Reward Plot

I will use Q-Learning data from Test run #1 for the Training Reward Plot:



Episode #	Reward for Episode
100	289
200	244
300	173
400	233
500	197
600	307
700	152
800	245
900	235
1000	200
1100	380
1200	140
1300	360
1400	320
1500	320
1600	200
1700	440
1800	180

1900	470
2000	410
2100	270
2200	410
2300	370
2400	440
2500	220
2600	290
2700	390
2800	460
2900	410
3000	330
3100	230
3200	220
3300	380
3400	390
3500	230
3600	430
3700	460
3800	320
3900	400
4000	390
4100	390
4200	310
4300	370
4400	400
4500	440
4600	310
4700	240
4800	280
4900	300
5000	330

The Training Reward Plot shows that by 100 episodes, Robby has learned to perform near Test-Average already in at least one episode. There are some fluctuations throughout the plot but there seems to be a slight increase in performance as Robby went through more training.

Conclusion

The first chart in the data section shows example 20 runs of the program. The range for Test-

Average was between 269 and 278. The range for Test-Standard-Deviation was between 68 and 72. The Test-Standard-Deviation seems to indicate that there's still quite some fluctuation in Robby's ability to maximize rewards. In general, Robby seems to have behaved very well in certain episodes and he was able to maximize his rewards (490-500 range). However, in certain episodes, he still managed to get negative rewards. I believe the high rewards point results are due to the fact that Robby has robust Q-value calculations for certain states he has already encountered many times during his learning phase. Robby has learned enough for these states. His low reward results are probably due to the fact that he has not learned enough for which actions to take for certain states. This was reflected in the Q-Table, as some Q-values for certain states were pretty high while some of the states had empty and very low Q-values. I also witnessed Robby trying to pick up and fail even in episode 5000 of the test, which seems to also support that Robby's learning is not quite finished. Random actions, which occurred at 10% probability during the test, also likely contributed to some of the low reward scores. Gradually lowering the ε (*epsilon*) value like we did for the learning phase during testing phase may also help raise the Test-Average and lower the Test-Standard-Deviation since it takes out some of the randomness introduced once Robby has sufficiently learned.