Nanyang Technological University (NTU)

CZ3005 Artificial Intelligence

Lab Assignment 2

Adrian Goh Jun Wei
U1721134D

Description of Algorithm

Initialization

In this project, I have decided to use Q-learning as the reinforcement learning algorithm.

The RandomAgent will be initialized with the discount, learning rate and exploration rate given.

In addition, the Q value for a state-action (*s*,*a*) pair is updated to be -99 if taking that specific action (*a*) is not possible for the state (*s*) due to the constraints of the environment. This is because the environment produces negative reward. This is important to ensure that only valid actions are taken during the action decision process, which will be explained later.

```
def __init__(self, dimension):
        self.action_space = ['left','right','forward','backward','up','down']
        self.Q = defaultdict(lambda: np.zeros(len(self.action_space)))
        self.discount = 0.99
        self.learning_rate = 0.5
        self.exploration_rate = 0.01
        self.dimension = dimension

        for x in range(self.dimension):
            for y in range(self.dimension):
                for z in range(self.dimension):
                    state = str(x) + str(y) + str(z)
                    if str(x) == '0':
                        self.Q[state][self.action_space.index("backward")] = -99
                    if str(x) == '3':
                        self.Q[state][self.action_space.index("forward")] = -99
                    if str(y) == '0':
                        self.Q[state][self.action_space.index("left")] = -99
                    if str(y) == '3':
                        self.Q[state][self.action_space.index("right")] = -99
                    if str(z) == '0':
                        self.Q[state][self.action_space.index("down")] = -99
                    if str(z) == '3':
                        self.Q[state][self.action_space.index("up")] = -99
```

Exploration v.s. Exploitation

In Q-learning, one of the problems to solve is to balance between exploration and exploitation. In this case, we use a simplified non-decaying formula to determine if the agent should explore or exploit the existing values. The given method **take_action** was updated to include this decision making process. A random float was used to compare to the exploration rate initialized, thus giving it a possibility of exploiting 99% of the time, and exploring 1% of the time.

In the case of exploiting, we choose to move the direction based on the action with the highest value in the Q table for that particular state. In the case of exploring, a random action will be taken based on the possible action spaces.

In both exploitation and exploration, the algorithm will only consider available actions for the specific state. For example, if the agent is in the state (1, 1, 3), it cannot move upwards. Thus, this action is omitted from the consideration.

```python
# added exploration or exploitation
def take_action(self, state):
    if random.random() > self.exploration_rate:
        return self.__exploit(state)
    else:
        return self.__explore(state)

def __exploit(self, state):
    state = str(state)

    action_index_with_largest_q_value = np.argmax(self.Q[state])
    return self.action_space[action_index_with_largest_q_value]

def __explore(self, state):
    state = str(state)

    available_actions = [
        self.action_space[index] for index in range(len(self.Q[state])) \
        if self.Q[state][index] != -99
    ]
    return random.choice(available_actions)
```

Training the agent

To train the agent, we will be updating the Q-value. The formula for calculating the new Q-value for state-action pair *(s, a)* at time *t* is as followed:

$$q^{new}(s,a) = (1 - \alpha)\ q(s,\ a) + \alpha\,(R_{t+1} + \gamma\ max(q(s',a'))\ )$$

Learning rate (α) can be defined as how much you accept the learned value vs the old value. In our case, a learning rate of 0.5 is being used, which means the learned and old value has equal weightage in the calculation of the new value.

The old value is represented by $q(s,\ a)$.

- This is based on the current Q value for state (s) and action (a)

The learned value is represented by $R_{t+1} + \gamma\ max(q(s',a'))$.

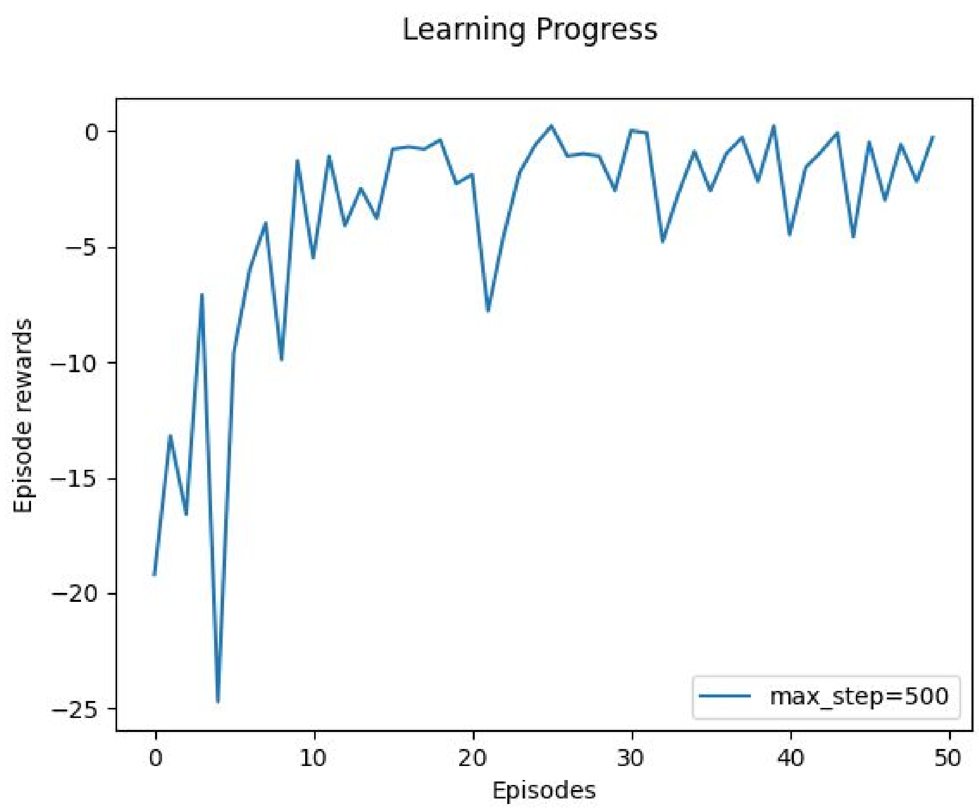
- R_{t+1} is the reward for moving to the next state
- γ is the discount factor

- Suppose, our agent was in state (*s*) and it took some action (*a*). As such, the environment might land our agent to any of the states (s') and from these states we get to maximize the action our agent will take by choosing the action with maximum Q value. This gives us the value of $max(q(s', a'))$.

```python
def train(self, state, action, next_state, reward):
    action_index = self.__get_action_index(action)
    old_value = self.Q[state][action_index]
    learned_value = reward + (self.discount * self.__get_max_Q_value(next_state))
    self.Q[state][action_index] = \
        ((1 - self.learning_rate) * old_value) + (self.learning_rate * learned_value)

def __get_action_index(self, action):
    if action not in self.action_space:
        pass
    return self.action_space.index(action)

def __get_max_Q_value(self, state):
    state = str(state)

    available_q_values = [
        self.Q[state][index] for index in range(len(self.Q[state])) \
        if self.Q[state][index] != -99
    ]
    return max(available_q_values)
```

Learning Progress

The learning progress is plotted using the Python library matplotlib. The method **plot_learning_progress** was written to generate a graph based on the episode rewards passed into it, and save the image as a file.



```
def plot_learning_progress(max_step, episode_rewards_list):
    Range = [i for i in range(len(episode_rewards_list))]
    plt.plot(Range, episode_rewards_list, label=f'max_step={max_step}')
    plt.suptitle("Learning Progress")
    plt.xlabel("Episodes")
    plt.ylabel("Episode rewards")
    ax = plt.gca()
    plt.legend()
    plt.savefig("learning_progress.png")
```

We can observe that the episode rewards are significantly more negative at the start. This is because the agent has little or no understanding of the environment and thus, approaches it with mostly exploratory behaviour. Therefore, it will require a number of steps before it reaches the goal state.

After some episodes, the agent will have a better understanding of the environment and approaches it with a more exploitative behaviour by making use of existing knowledge it has gained. As such, it will require less and less movement before it reaches the goal state, and thus, has a higher reward.

Q-table

Rounding up values in Q-table

The method **round_up_Q_values** was added to the class RandomAgent. It will iterate through the values in the agent's dictionary containing the Q-table, and round them up to a certain number of decimal points, for ease of reading. In our case, we round it up to 3 decimal points.

```
def round_up_Q_values(self, num_of_dp):
        for key, values in self.Q.items():
            for index in range(len(values)):
                self.Q[key][index] = round(values[index], num_of_dp)
```

Exporting Q-table into CSV

The Q-table is then generated using the method **export_Q_table_to_csv**. The values are exported as a CSV (comma-separated values) file. The CSV file can also be found in the same folder, titled "*Q_table.csv*".

```
def export_Q_table_to_csv(agent, csv_file_name="Q_table.csv"):
    q_table_dict = agent.Q
    q_table_list = list(dict(agent.Q).keys())
    q_table_list.sort()

    with open(csv_file_name, 'w', newline='') as file:
        file_writer = csv.writer(file, delimiter=",")
        file_writer.writerow(["state"] + agent.action_space)

        for state in q_table_list:
            row_data = [state]
            values = agent.Q[state]
            for index in range(len(values)):
                row_data.append(str(values[index]))
            file_writer.writerow(row_data)

    print("File exported!")
```

Values

state	left	right	forward	backward	up	down
000	-99.000	-0.613	-0.361	-99.000	-0.479	-99.000
001	-99.000	-0.508	-0.461	-99.000	-0.298	-0.488
002	-99.000	-0.085	-0.305	-99.000	-0.409	-0.513

003	-99.000	0.058	-0.434	-99.000	-99.000	-0.407
010	-0.530	-0.480	-0.525	-99.000	-0.170	-99.000
011	-0.321	-0.454	-0.098	-99.000	-0.469	-0.503
012	-0.207	0.037	-0.186	-99.000	-0.219	-0.326
013	-0.405	0.158	-0.377	-99.000	-99.000	-0.146
020	-0.471	-0.486	-0.130	-99.000	-0.583	-99.000
021	-0.460	-0.314	-0.045	-99.000	-0.359	-0.402
022	-0.320	-0.309	-0.302	-99.000	-0.010	-0.063
023	-0.265	0.314	-0.108	-99.000	-99.000	-0.123
030	-0.492	-99.000	-0.110	-99.000	-0.575	-99.000
031	-0.311	-99.000	-0.010	-99.000	-0.266	-0.341
032	-0.185	-99.000	0.002	-99.000	-0.171	-0.170
033	0.190	-99.000	0.577	-99.000	-99.000	-0.286
100	-99.000	-0.469	-0.317	-0.594	-0.512	-99.000
101	-99.000	-0.367	-0.106	-0.556	-0.554	-0.470
102	-99.000	-0.256	-0.067	-0.403	-0.324	-0.407
103	-99.000	-0.260	0.191	-0.267	-99.000	-0.297
110	-0.496	-0.525	-0.188	-0.550	-0.515	-99.000
111	-0.376	-0.415	0.040	-0.389	-0.369	-0.379
112	-0.138	-0.119	-0.268	-0.131	-0.045	-0.218
113	-0.225	0.195	-0.261	-0.217	-99.000	-0.214
120	-0.411	-0.323	-0.043	-0.421	-0.310	-99.000
121	-0.419	-0.190	-0.209	-0.225	0.134	-0.287
122	-0.042	-0.050	0.407	-0.074	-0.069	-0.080
123	0.069	0.011	0.603	0.027	-99.000	0.090
130	-0.429	-99.000	-0.303	-0.431	-0.017	-99.000
131	-0.113	-99.000	0.297	-0.139	-0.159	-0.140

132	-0.076	-99.000	-0.004	-0.107	0.324	-0.169
133	0.126	-99.000	0.553	0.221	-99.000	0.171
200	-99.000	-0.387	-0.569	-0.336	-0.189	-99.000
201	-99.000	-0.439	-0.383	-0.300	-0.008	-0.434
202	-99.000	0.019	-0.215	-0.194	-0.190	-0.207
203	-99.000	0.093	-0.156	-0.165	-99.000	-0.124
210	-0.356	-0.481	-0.112	-0.267	-0.247	-99.000
211	-0.249	0.242	-0.204	-0.313	-0.318	-0.352
212	-0.208	-0.039	-0.229	-0.047	0.202	-0.145
213	-0.055	0.116	0.492	-0.041	-99.000	-0.036
220	-0.375	-0.119	-0.038	-0.340	-0.325	-99.000
221	0.154	-0.125	0.209	-0.108	-0.118	-0.134
222	0.001	0.032	0.038	0.026	0.551	0.060
223	0.240	0.572	0.280	0.248	-99.000	0.416
230	-0.187	-99.000	0.318	-0.197	-0.126	-99.000
231	0.068	-99.000	0.389	-0.136	0.019	0.044
232	0.127	-99.000	0.383	0.130	0.767	0.138
233	0.331	-99.000	0.999	0.349	-99.000	0.242
300	-99.000	-0.173	-99.000	-0.339	-0.486	-99.000
301	-99.000	-0.345	-99.000	-0.151	0.144	-0.310
302	-99.000	0.061	-99.000	-0.061	-0.188	-0.099
303	-99.000	0.117	-99.000	-0.151	-99.000	-0.159
310	-0.157	0.078	-99.000	-0.219	-0.133	-99.000
311	-0.213	0.285	-99.000	-0.076	-0.225	-0.209
312	0.072	0.492	-99.000	0.117	0.168	-0.114
313	0.202	0.763	-99.000	-0.026	-99.000	0.095
320	-0.208	-0.264	-99.000	-0.031	0.268	-99.000

321	0.015	0.059	-99.000	0.083	0.379	0.053
322	0.141	0.089	-99.000	0.136	0.719	0.087
323	0.516	0.923	-99.000	0.389	-99.000	0.373
330	-0.126	-99.000	-99.000	-0.388	0.486	-99.000
331	0.157	-99.000	-99.000	0.083	0.808	0.111
332	0.252	-99.000	-99.000	0.214	0.988	0.237
333	0.000	-99.000	-99.000	0.000	-99.000	0.000