Modular Reinforcement Learning for a Multitasking Walking Agent

Nov. 2020, Machine Learning

Chenkuan Liu

1 Introduction

In this project, we apply the idea of modular reinforcement learning to a walking agent that needs to perform multiple tasks at the same time. We construct our environment as a 2D gridworld. The agent starts somewhere on the left side and aims to reach a specified destination on the right side. During the process, the agent needs to try to stay on the sidewalk, which is consisted of the middle rows of the gridworld. It also needs to pick up the litters on the sidewalk and try to avoid the obstacles at the same time. Modular reinforcement learning is particularly useful for the task, as it allows us to decompose the objective into several smaller modules. We will learn four modules: the module keeping the agent on the sidewalk, the module guiding the agent to reach the destination, the module guiding the agent picking up litters during walking, and the module helping the agent avoiding obstacles. The action space for the agent is consisted of four elements: moving up, right, down, or left. It is the same for all four modules. For state space, we need to create appropriate representations that are suitable for different modules respectively. For each module, We will use Q-learning and ϵ-greedy algorithm to learn the values inside the associated Q-table. After that, we will tune the weights and compute the weighted sum of Q-table entries for the combined module, and use that to guide the agent to move through the gridworld that will be generated at random.

2 Procedure

2.1 Initialization

To standardize our procedure, we fix the size of our world to be 8×16, and use '\$' to denote litter, 'x' to denote obstacle, 'a' to denote agent, and '0' to denote empty grids. The sidewalk is from the 2nd row to the 5th row. The agent will start at position $[2, 0]$ and try to reach the goal at position $[5, 15]$. To agree with the index notation of `numpy`, the rows and columns at here all start from 0.

```
[['0' '0' '0' '0' '$' '0' '0' '0' '0' '0' 'x' 'x' 'x' '0' 'x' '0']
 ['0' 'x' '0' '0' '0' '0' 'x' '0' '0' 'x' '0' '0' '0' '$' '0' '0']
 ['a' '0' '0' '0' '0' '0' 'x' '$' '0' '0' '$' '0' '0' 'x' '0' '0']
 ['0' 'x' '0' '0' '0' '0' 'x' '0' '$' '0' '0' '0' '0' '$' '0' '0']
 ['0' '0' '0' '0' 'x' '0' '0' 'x' 'x' '0' '0' '0' '0' '0' '$' '0']
 ['0' '$' '0' '0' '0' '0' '0' '0' '$' '0' '0' 'x' '0' '0' '0' 'g']
 ['0' '$' 'x' '0' '0' '0' '0' '$' '0' '0' '0' '0' '0' '0' 'x' '0']
 ['0' '0' 'x' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0']]
```

Figure 1: A Sample Gridworld. The agent starts at $[0, 2]$ and tries to reach the goal at $[5, 15]$ denoted by 'g'. The middle four rows are the sidewalk the agent needs to try to stay on while picking up litters '\$' and avoiding obstacles 'x'.[1]

2.2 Decomposition

As mentioned earlier, we will decompose the whole task into four modules: staying on sidewalk, reaching goal, picking up litters, and avoiding obstacles. In general, the modules for sidewalk and goal are more straightforward, while the modules for obstacle and litter are more complicated. We will start from the two easier ones, and then move on to the two more complex ones.

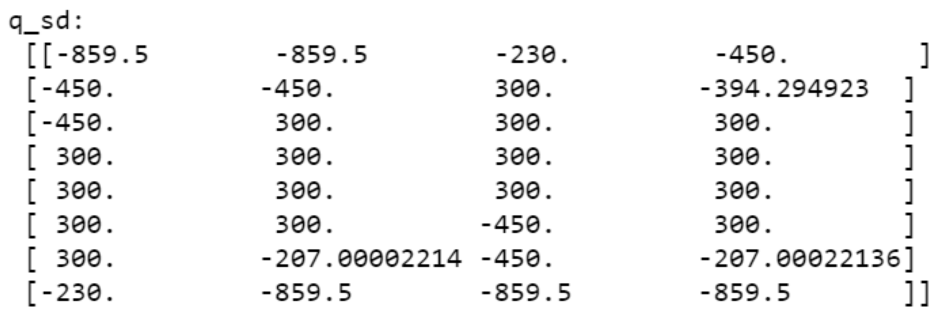
[1]The agent can only move in one of the four cardinal directions. It will remain on the same grid if it tries to move out of boundary. In addition, if the agent steps onto a grid with litter, that grid will be turned into an empty grid. The agent is allowed to step on the obstacle but will receive large penalty. The walking terminates once the agent reaches the goal or exceeds the assigned step limit.

2.2.1 Sidewalk Module

The sidewalk module is the easiest one of the four. Since we define the middle four rows (row 2 to row 5) as our sidewalk, we only need to care about the row number. Hence, we only need 8 states in our state space, namely the row the agent is standing on. Therefore, the ith state denotes the ith row. Combining with 4 possible actions inside each row, we get a 8×4 Q-table. For the reward, we reward the agent when it takes a move that will take it to or keep it on the sidewalk, and punish it (i.e. give negative reward) when it takes a move that will take or keep it outside the sidewalk.

There are infinitely combination of values that can be assigned to the rewards. At here, I set the positive reward as 30 for being on the sidewalk and negative reward as -500 for being off the sidewalk. The other parameters are: $num\_episodes = 4000$, $max\_step = 200$, $\mu = 0.9$ (learning rate), $\gamma = 0.9$ (discount factor), $\epsilon = 0.5$ (for ϵ-greedy algorithm).

After learning, we can get the following Q-table:

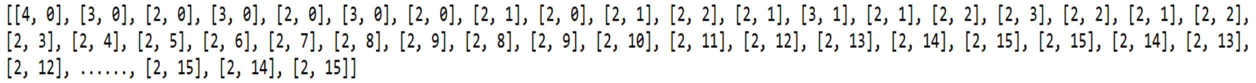


```
q_sd:
 [[-859.5        -859.5        -230.          -450.        ]
 [-450.         -450.          300.         -394.294923  ]
 [-450.          300.          300.          300.        ]
 [ 300.          300.          300.          300.        ]
 [ 300.          300.          300.          300.        ]
 [ 300.          300.         -450.          300.        ]
 [ 300.         -207.00002214 -450.         -207.00022136]
 [-230.         -859.5        -859.5        -859.5       ]]
```

Figure 2: Q-table for Sidewalk Module. The columns represent moving up, right, down, and left from the leftmost column to rightmost one. The rows represent the row agent is currently on.

At here, and for all other modules, 0th, 1st, 2nd, 3rd columns represent moving up, right, down, left respectively. So, entry $[0, 2]$ of the Q-table above tells us the Q-value (i.e. expected cumulative reward) for the agent moving down from 0th row is -230. It is the largest value compared to all other actions, which are all -859.5. It makes sense, since if the agent is on the 0th row, the only way for it to make to the sidewalk is moving down. The same holds if the agent is on the second row. Similarly, the last two rows of Q-table tell us that if the agent is on the 6th or 7th row, going up leads to the highest Q-value. If the agent is on the 2nd row, moving up has the lowest Q-value (-450), since such action will make him leave the sidewalk. The opposite holds true when the agent is on the 5th row. For the 3rd and 4th row, all Q-values are 300, since taking any direction will still keep the agent on the sidewalk.

Below is the trajectory of the agent in sidewalk module during the last episode. It is denoted by the coordinate location in the gridworld. It contains 200 steps, we omit the middle part for displaying purpose. We can observe that the agent moves across the gridworld from $[4, 0]$ on the right to $[2, 15]$ on the right, but it always remains on the side walk.



```
[[4, 0], [3, 0], [2, 0], [3, 0], [2, 0], [3, 0], [2, 0], [2, 1], [2, 0], [2, 1], [2, 2], [2, 1], [3, 1], [2, 1], [2, 2], [2, 3], [2, 2], [2, 1], [2, 2],
[2, 3], [2, 4], [2, 5], [2, 6], [2, 7], [2, 8], [2, 9], [2, 8], [2, 9], [2, 10], [2, 11], [2, 12], [2, 13], [2, 14], [2, 15], [2, 15], [2, 14], [2, 13],
[2, 12], ......, [2, 15], [2, 14], [2, 15]]
```

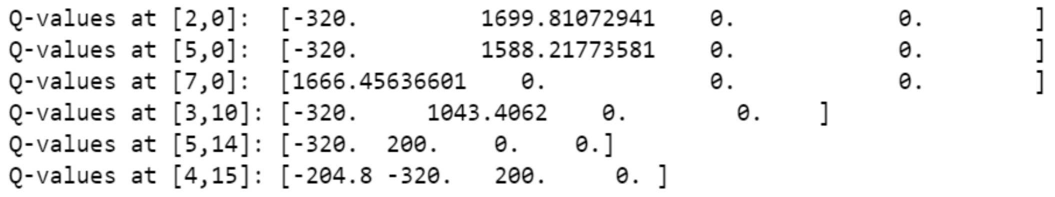
Figure 3: A Sidewalk Module Trajectory.

2.2.2 Goal Module

The module for the agent reaching the goal at $[5, 15]$ is slightly more complicated than the sidewalk module above. At here, to understand how far the agent is from the goal, we need to consider both the row number and column number the agent is standing on. And so, we create a state space with $8 \times 16 = 128$ states for the goal module. Each state corresponds to a specific location of the agent. To make the correspondence clearer, we define a function such that location $[r, c]$ is mapped to state number $16r + c$. Thus, given a state number s, it means that the agent is standing on the $(s//16)^{th}$ row and $(s\%16)^{th}$ column ("//" stands for integer division and "%" stands for modulo operation). It is easy to verify that this is a one-to-one correspondence.

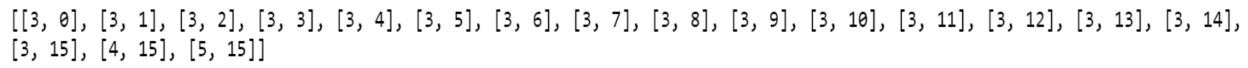
Then, for the rewards, I set them based on the sum of the row difference and column difference between the agent and the goal state $[5, 10]$. If the agent is on the goal state (namely zero sum), the reward is 100000. Otherwise, we compare the distance sum between the agent's original location and its new location. If the agent gets closer to the goal, the reward is 200; if it gets farther away from the goal, the reward is -400. The other parameters are: $num\_episodes = 4000$, $max\_step = 50$, $\mu = 0.8$, $\gamma = 0.9$, $\epsilon = 0.01$.

Since the goal module Q-table is of size 128×4, we do not display all of them here. Instead, we select certain locations of interest to observe the behavior. From Figure 4, we see that if the agent is above or on the 5th row (location $[2, 0]$, $[5, 0]$, and $[3, 10]$), moving right has the greatest cumulative reward; if the agent is at bottom-left corner (location $[7, 0]$), moving up has the highest Q-value; if the agent is one step away from the goal (location $[5, 14]$ and $[4, 15]$), the action that takes it to the goal (moving right from $[5, 14]$ or moving down from $[4, 15]$) has the highest Q-value. These Q-values agree with our environment. Figure 5 depicts the trajectory of the agent starting from $[3, 0]$, we can see that it prioritizes moving rightward and then moves down to the goal after reaching the 15th column.



```
Q-values at [2,0]:  [-320.         1699.81072941    0.            0.        ]
Q-values at [5,0]:  [-320.         1588.21773581    0.            0.        ]
Q-values at [7,0]:  [1666.45636601    0.            0.            0.        ]
Q-values at [3,10]: [-320.     1043.4062    0.        0.    ]
Q-values at [5,14]: [-320.  200.    0.    0.]
Q-values at [4,15]: [-204.8 -320.   200.     0. ]
```

Figure 4: Q-values for Certain States in Goal Module.



```
[[3, 0], [3, 1], [3, 2], [3, 3], [3, 4], [3, 5], [3, 6], [3, 7], [3, 8], [3, 9], [3, 10], [3, 11], [3, 12], [3, 13], [3, 14],
[3, 15], [4, 15], [5, 15]]
```

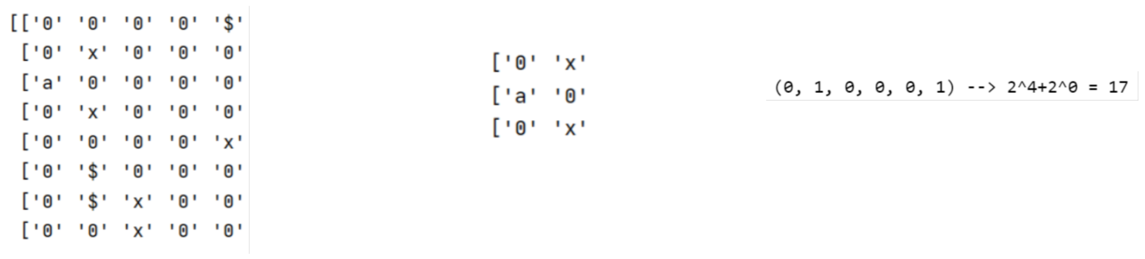
Figure 5: A Goal Module Trajectory.

2.2.3 Obstacle Module

The module for avoiding obstacle become more complicated than the two modules above. In sidewalk and goal modules, we only need to care about the agent's location. If we take obstacles into consideration, the computational cost will be too high to allow us to represent each gridworld's situation, which has 2^{128} possible scenarios. In addition, the obstacles too far away will have little significance to the agent's current action. Therefore, instead, we will only look at the grids that are adjacent to the agent.

In my setup, I only look at six locations: the agent itself, grid above the agent, grid below the agent, grid right of the agent, grid upper-right of the agent, and grid lower-right of the agent. In this way, the state number decreases from 2^{128} to 2^6. To map these locations into state number, I borrow the idea of binary representation such that the empty grid is marked as 0 and the grid with obstacle is marked as 1. Denote the marked value at grid $[i, j]$ as v_{ij} (we only look at the six locations, so $0 \leq i \leq 2$, $0 \leq j \leq 1$), the state number s corresponding to the local environment is defined as $s = 2^5 v_{00} + 2^4 v_{01} + 2^3 v_{10} + 2^2 v_{11} + 2^1 v_{20} + 2^0 v_{21}$.

We can use the sample gridworld in Figure 1 to demonstrate the transition. Figure 6 displays the transformation process from the entire gridworld to the local environment we concern and the corresponding state number:



```
[['0' '0' '0' '0' '$'
 ['0' 'x' '0' '0' '0'
 ['a' '0' '0' '0' '0'
 ['0' 'x' '0' '0' '0'
 ['0' '0' '0' '0' 'x'
 ['0' '$' '0' '0' '0'
 ['0' '$' 'x' '0' '0'
 ['0' '0' 'x' '0' '0'
```

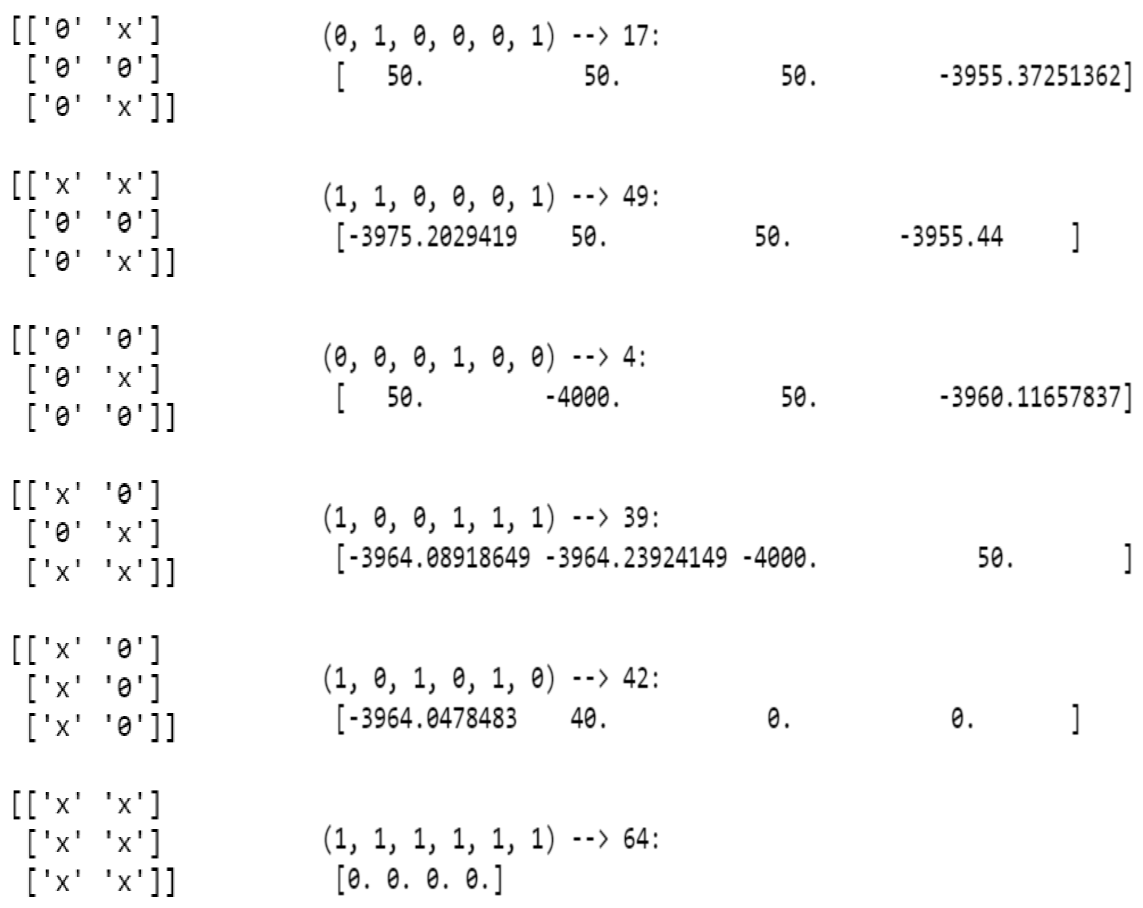
```
['0' 'x'
['a' '0'
['0' 'x'
```

```
(0, 1, 0, 0, 0, 1) --> 2^4+2^0 = 17
```

Figure 6: State Representation Demonstration. The left figure is directly cropped from the first five columns of Figure 1. The middle figure shows the 6-grid local environment surrounding the agent that we concern here. The right figure shows the encoding based on whether the grid is an obstacle or not and transformation from it to state number.

Note that for any 6-grid local environment, the agent is always at the middle-right position. For convenience, if the agent is on the edges of the gridworld, the entries in the 6-grid environment that are out of boundary are marked as 0 instead.

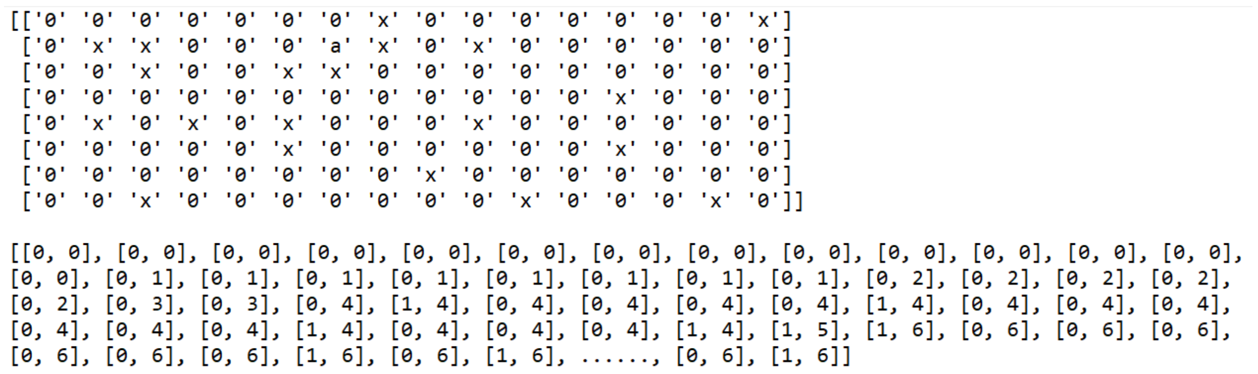
For the setup, I set the reward to be 5 when the agent does not hit obstacle during its movement, and -5000 when it hits one. The other parameters are: $num\_episodes = 4000$, $max\_step = 200$, $\mu = 0.8$, $\gamma = 0.9$, $\epsilon = 0.33$. Again, since there are too many states, we only select a few of them to demonstrate the potential action of the agent after learning. Figure 7 below demonstrate six examples from top to bottom. We neglect the agent representation since it is always at the middle-right grid (in practice, this can be done by creating another gridworld that only tracks the movement of the agent, and combine it with the obstacle-world for learning process). The first example is exactly the local environment in Figure 6. Since there is no adjacent obstacle, moving up, right, or down all has reward 50. In the second example, there is one obstacle on the top, so moving up has large penalty, while moving right or down has reward 50. Similarly, in the third example, an obstacle is on the right, so moving right gives -4000 while moving up or down gives 50. In all these three examples, moving left has large penalty, since it is out of the local environment and we do not know what it is. In the fourth example, however, since moving up, right, or down will all encounter obstacle, the three actions receive large penalty, while moving left receives reward 50.



```
[['0' 'x']          (0, 1, 0, 0, 0, 1) --> 17:
 ['0' '0']           [   50.           50.           50.        -3955.37251362]
 ['0' 'x']]

[['x' 'x']          (1, 1, 0, 0, 0, 1) --> 49:
 ['0' '0']           [-3975.2029419    50.          50.       -3955.44     ]
 ['0' 'x']]

[['0' '0']          (0, 0, 0, 1, 0, 0) --> 4:
 ['0' 'x']           [   50.        -4000.           50.        -3960.11657837]
 ['0' '0']]

[['x' '0']          (1, 0, 0, 1, 1, 1) --> 39:
 ['0' 'x']           [-3964.08918649 -3964.23924149 -4000.           50.       ]
 ['x' 'x']]

[['x' '0']          (1, 0, 1, 0, 1, 0) --> 42:
 ['x' '0']           [-3964.0478483    40.          0.           0.      ]
 ['x' '0']]

[['x' 'x']
 ['x' 'x']          (1, 1, 1, 1, 1, 1) --> 64:
 ['x' 'x']]          [0. 0. 0. 0.]
```

Figure 7: Selected States and Q-table Entries. We choose 6 possible scenarios of the local environment, print the encoding and state number, and look at the relevant Q-values.

Because of the random environment setup during learning, we cannot guarantee that the agent will learn all of the possible states. In the fifth example, the agent is located on the obstacle in middle-right, moving right has the highest reward 40, since it will let the agent leave the obstacle. What is worthy to notice at here is that, after a certain amount of learning, it will become increasingly rare to encounter the situation where the agent is on the obstacle (since our primary objective is to let the agent avoid the obstacle rather than how to act when it hits one). So, though example five predicts a reasonable action to move right, the last two entries of moving down or left are all zero despite that moving down will also hit an obstacle, which is probably because this situation is rare to encounter in the learning process. The sixth example further demonstrates this point. It is very rare when all six grids are obstacles and the agent has hit on the middle-right one, so the Q-values are all zero, which is the default value during initialization.



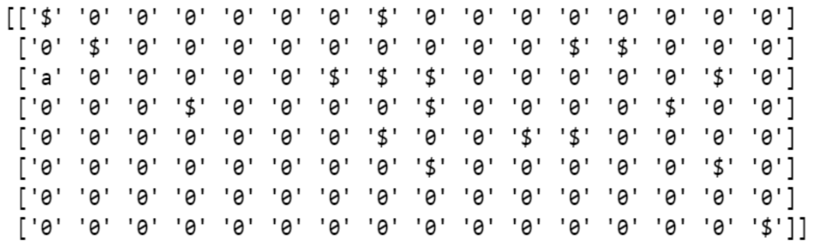
```
[['0' '0' '0' '0' '0' '0' '0' 'x' '0' '0' '0' '0' '0' '0' '0' 'x']
 ['0' 'x' 'x' '0' '0' '0' 'a' 'x' '0' 'x' '0' '0' '0' '0' '0' '0']
 ['0' '0' 'x' '0' '0' 'x' 'x' '0' '0' '0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' 'x' '0' '0' '0']
 ['0' 'x' '0' 'x' '0' 'x' '0' '0' '0' 'x' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' 'x' '0' '0' '0' '0' '0' '0' 'x' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' '0' 'x' '0' '0' '0' '0' '0' '0' '0']
 ['0' '0' 'x' '0' '0' '0' '0' '0' '0' '0' 'x' '0' '0' '0' 'x' '0']]

[[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0],
[0, 0], [0, 1], [0, 1], [0, 1], [0, 1], [0, 1], [0, 1], [0, 1], [0, 1], [0, 2], [0, 2], [0, 2], [0, 2],
[0, 2], [0, 3], [0, 3], [0, 4], [1, 4], [0, 4], [0, 4], [0, 4], [0, 4], [1, 4], [0, 4], [0, 4], [0, 4],
[0, 4], [0, 4], [0, 4], [1, 4], [0, 4], [0, 4], [0, 4], [1, 4], [1, 5], [1, 6], [0, 6], [0, 6], [0, 6],
[0, 6], [0, 6], [0, 6], [1, 6], [0, 6], [1, 6], ......, [0, 6], [1, 6]]
```

Figure 8: An Obstacle Module Gridworld and Trajectory.

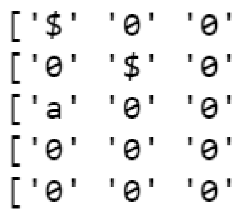
Figure 8 shows the gridworld in the last learning episode in obstacle module and the agent's trajectory. The agent starts at $[0, 0]$, standing at there for some time because it tries to go out of boundary by moving up or left, and then moves around the top left area of the gridworld without hitting the obstacles at $[1, 1]$ and $[1, 2]$. After it reaches the 6^{th} column, it keeps moving back and forth between $[0, 6]$ and $[1, 6]$ (we omit the steps here in the display above; there are 200 steps in total) and stops when the step limit is reached. Note that the reason it moves back and forth between $[0, 6]$ and $[1, 6]$ is that the adjacent grids $[0, 7]$, $[1, 7]$ and $[2, 6]$ are all obstacles. As a result, we can see that the agent does a fair job to avoid the obstacles.

2.2.4 Litter Module

The ideas behind litter module are similar to those in obstacle module. One of the difference is that, in obstacle module, it is better to avoid the obstacles only when they get close to the agent (otherwise, the agent will stay back when it perceives an obstacle very far away, which will hinder its learning), but it is better to let the agent perceive the litters beyond the adjacent grids so that the agent can go for them instead of wandering locally. Therefore, in my setup, I set the local environment for the litter module to be composed of 15 grids, which ranges from two rows above the agent to two rows below the agent, and ranges from the current column to the two columns right of the agent. So, the size of state space is 2^{15} for litter module. We perform the same binary encoding process at here for the transformation from 15-grid local environment to the corresponding state number.



```
[['$' '0' '0' '0' '0' '0' '0' '$' '0' '0' '0' '0' '0' '0' '0' '0']
 ['0' '$' '0' '0' '0' '0' '0' '0' '0' '0' '0' '$' '$' '0' '0' '0']
 ['a' '0' '0' '0' '0' '0' '$' '$' '$' '0' '0' '0' '0' '0' '$' '0']
 ['0' '0' '0' '$' '0' '0' '0' '0' '$' '0' '0' '0' '0' '$' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' '$' '0' '0' '$' '$' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' '0' '$' '0' '0' '0' '0' '0' '$' '0']
 ['0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '$']]
```



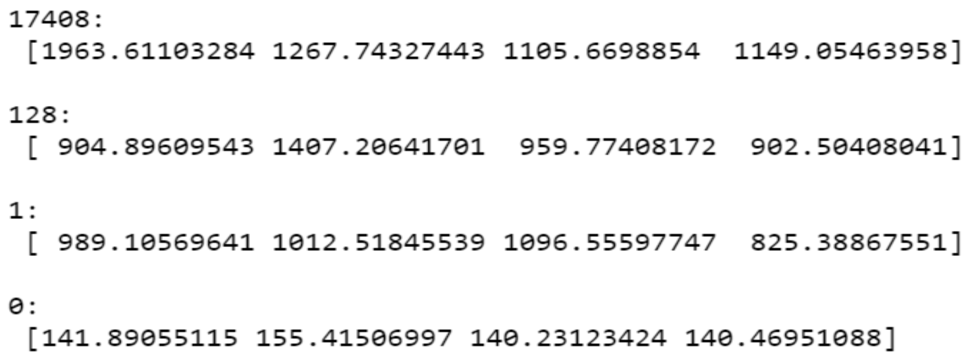
```
['$' '0' '0'
['0' '$' '0'
['a' '0' '0'
['0' '0' '0'
['0' '0' '0'
```

Figure 9: A Sample Little Module Demonstration. The left figure is an extracted gridworld that only contains litters and the agent. The right figure is the local 15-gridworld surrounding the agent cropped from the left figure that are needed for our encoding process.

Figure 9 is a sample gridworld that demonstrates the process. We crop the 15 local grids surrounding the agent that we concern here. We count from top to down in rows while count from left to right within each row. Hence, in this local gridworld, only the first one ($[0, 0]$) and the fifth one ($[1, 1]$) are litters, so the encoded list is $[1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$, and the corresponding state number is $2^{14} + 2^{10} = 17408$.

In general, the process is similar to that in the obstacle module, despite that we have more states to consider. Therefore, I set the training episode number *num_episodes* to be 6000, which will help the agent cover more situations and have better learning. The reward is 500 when the agent encounters a litter, and -1 otherwise. All other parameters remain the same. Since the 15-gridworld is much larger than 6-gridworld, we will not print out the gridworld sample during our following demonstration. We will look at specific state numbers, describe the pattern inside that number, and observe the corresponding Q-table entries.

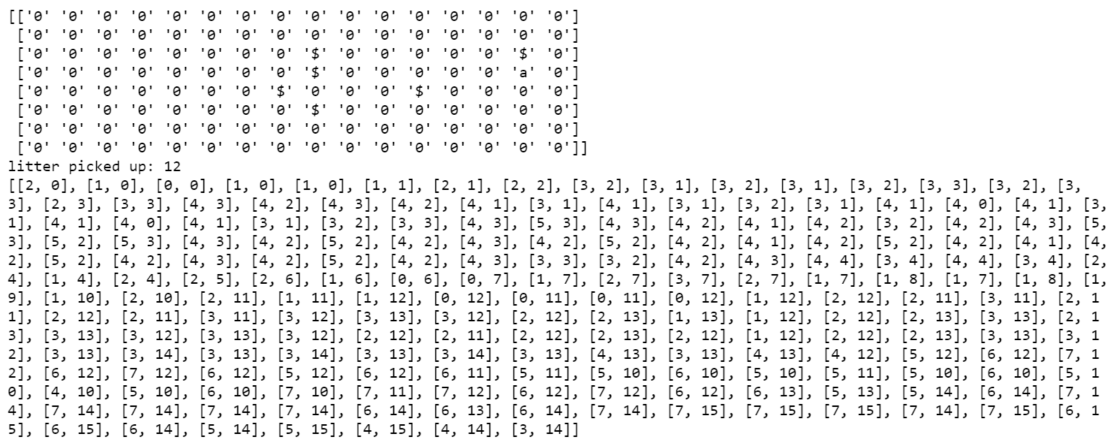
Figure 10 shows 4 representative examples with their state numbers and associated Q-table entries. The first example is state 17408, which is exactly the state we showed in Figure 9. In that state, one litter is 2 grid above the agent, another is located up-right of the agent. To get close to both of the litters, it is best for the agent to go up. We see that the Q-entry agree with our understanding, as the 0th entry (i.e. moving up) has the highest Q-value (1963.6). The second example is state 128. Note that $128 = 2^7$, it corresponds to the state where there is only one litter right of the agent. The Q-entry shows that moving right has the highest Q-value (1407.2), which agrees with our understanding. The third example is state



```
17408:
 [1963.61103284 1267.74327443 1105.6698854  1149.05463958]

128:
 [ 904.89609543 1407.20641701  959.77408172  902.50408041]

1:
 [ 989.10569641 1012.51845539 1096.55597747  825.38867551]

0:
 [141.89055115 155.41506997 140.23123424 140.46951088]
```

Figure 10: Sample States and Q-table Entries.[2]

1, in which there is only one litter at the bottom-right location of the local gridworld. The Q-entry shows that moving down has the highest Q-value (1096.6) and moving down has the second highest (1012.5), as both of these two actions will get the agent closer to the litter. The fourth example is state 0, which means that there is no litter in the local gridworld. We see that there is no significant value difference between different actions. Moving right is slightly higher (155.4), while moving up, down, and left are all around 140.



```
[['0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' '0' '$' '0' '0' '0' '0' '0' '$' '0']
 ['0' '0' '0' '0' '0' '0' '0' '0' '$' '0' '0' '0' '0' '0' 'a' '0']
 ['0' '0' '0' '0' '0' '0' '0' '$' '0' '0' '0' '$' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' '0' '$' '0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0']]
litter picked up: 12
[[2, 0], [1, 0], [0, 0], [1, 0], [1, 0], [1, 1], [2, 1], [2, 2], [3, 2], [3, 1], [3, 2], [3, 1], [3, 2], [3, 3], [3, 2], [3,
3], [2, 3], [3, 3], [4, 3], [4, 2], [4, 3], [4, 2], [4, 1], [3, 1], [4, 1], [3, 1], [3, 2], [3, 1], [4, 1], [4, 0], [4, 1], [3,
1], [4, 1], [4, 0], [4, 1], [3, 1], [3, 2], [3, 3], [4, 3], [5, 3], [4, 3], [4, 2], [4, 1], [4, 2], [3, 2], [4, 2], [4, 3], [5,
3], [5, 2], [5, 3], [4, 3], [4, 2], [5, 2], [4, 2], [4, 3], [4, 2], [5, 2], [4, 2], [4, 1], [4, 2], [5, 2], [4, 2], [4, 1], [4,
2], [5, 2], [4, 2], [4, 3], [4, 2], [5, 2], [4, 2], [4, 3], [3, 3], [3, 2], [4, 2], [4, 3], [4, 4], [3, 4], [4, 4], [3, 4], [2,
4], [1, 4], [2, 4], [2, 5], [2, 6], [1, 6], [0, 6], [0, 7], [1, 7], [2, 7], [3, 7], [2, 7], [1, 7], [1, 8], [1, 7], [1, 8], [1,
9], [1, 10], [2, 10], [2, 11], [1, 11], [1, 12], [0, 12], [0, 11], [0, 11], [0, 12], [1, 12], [2, 12], [2, 11], [3, 11], [2, 1
1], [2, 12], [2, 11], [3, 11], [3, 12], [3, 13], [3, 12], [2, 12], [2, 13], [1, 13], [1, 12], [2, 12], [2, 13], [3, 13], [2, 1
3], [3, 13], [3, 12], [3, 13], [3, 12], [2, 12], [2, 11], [2, 12], [2, 13], [2, 12], [1, 12], [2, 12], [2, 13], [3, 13], [3, 1
2], [3, 13], [3, 14], [3, 13], [3, 14], [3, 13], [3, 14], [3, 13], [4, 13], [3, 13], [4, 13], [4, 12], [5, 12], [6, 12], [7, 1
2], [6, 12], [7, 12], [6, 12], [5, 12], [6, 12], [6, 11], [5, 11], [5, 10], [6, 10], [5, 10], [5, 11], [5, 10], [6, 10], [5, 1
0], [4, 10], [5, 10], [6, 10], [7, 10], [7, 11], [7, 12], [6, 12], [7, 12], [6, 12], [6, 13], [5, 13], [5, 14], [6, 14], [7, 1
4], [7, 14], [7, 14], [7, 14], [7, 14], [6, 14], [6, 13], [6, 14], [7, 14], [7, 15], [7, 15], [7, 15], [7, 14], [7, 15], [6, 1
5], [6, 15], [6, 14], [5, 14], [5, 15], [4, 15], [4, 14], [3, 14]]
```

Figure 11: Agent Action in Fig. 9's Gridworld.

Figure 11 depicts the action of the agent if we set the sample gridworld in Fig. 9 as the last

[2]Since our gridworlds are generated randomly, with each re-run of the Q-learning algorithm, we will get slightly different Q-values for each state. However, the general pattern should stay the same. For instance, for state 128, moving right should always have the highest Q-value.

environment in agent's learning. In Fig. 9, agent starts from $[2, 0]$. We can also count that there are 18 litters in total. The upper gridworld in Figure 11 is the outcome after the agent walked 200 steps. During learning, I set a counter to record the number of litters the agent picked up. We can see that the agent picked up 12 litters, which are two-thirds of the total litters in the environment. In general, it is a decent result.

2.2.5 Final Combined Module

Now we can combine all the four modules together to form our final module. We use weighted-sum to compute the Q-values in the final module. The pseudo-algorithm is as follows:

- Given a randomly generated gridworld environment
- While within the step number limit or not reached the goal:
    1. extract the relevant environment information and get the corresponding state numbers in the 4 modules;
    2. get the 4 Q-table entries related to the 4 states of each modules, each entry should contain 4 values for moving up, right, down, or left in the current location;
    3. use weighted-sum to combine the 4 Q-table entries into one final entry, choose the action (i.e. direction) with the highest Q-value inside that final entry;
    4. update agent location and environment (set litter grid to empty if picked up).

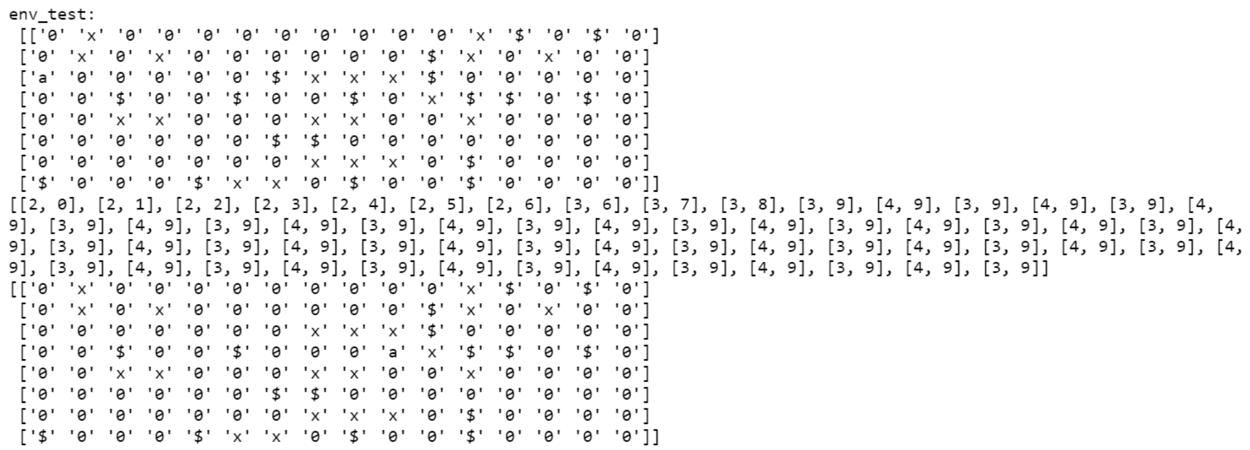
Note that different choices of weights will lead to potentially drastically different behavior. They need to be carefully tuned in order to achieve decent performance. More will be discussed in the next Performance section.

3 Performance

In our evaluation, we will set the agent's starting location at $[2, 0]$. We will also set the step limit to be 60. Therefore, the agent will either reach the goal at $[5, 15]$ and terminate the program or stand somewhere inside the gridworld after 60 steps. We will choose different weight choices and observe the behavior.

3.1 Default Equal Weights

By default, I set each weight to be 1. Therefore, after storing the 4 relevant Q-table entries from each module as `q_sd`, `q_goal`, `q_obs` and `q_lit`, the combined final entry is: `q_combined = q_sd + q_obs + q_lit + q_goal`.



```
env_test:
 [['0' 'x' '0' '0' '0' '0' '0' '0' '0' '0' '0' 'x' '$' '0' '$' '0']
 ['0' 'x' '0' 'x' '0' '0' '0' '0' '0' '0' '$' 'x' '0' 'x' '0' '0']
 ['a' '0' '0' '0' '0' '0' '$' 'x' 'x' 'x' '$' '0' '0' '0' '0' '0']
 ['0' '0' '$' '0' '0' '$' '0' '0' '$' '0' 'x' '$' '$' '0' '$' '0']
 ['0' '0' 'x' 'x' '0' '0' '0' 'x' 'x' '0' '0' 'x' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '$' '$' '0' '0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' 'x' 'x' 'x' '0' '$' '0' '0' '0' '0']
 ['$' '0' '0' '0' '$' 'x' 'x' '0' '$' '0' '0' '$' '0' '0' '0' '0']]
[[2, 0], [2, 1], [2, 2], [2, 3], [2, 4], [2, 5], [2, 6], [3, 6], [3, 7], [3, 8], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9], [4,
9], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9], [4,
9], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9], [4,
9], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9], [4, 9], [3, 9]]
[['0' 'x' '0' '0' '0' '0' '0' '0' '0' '0' '0' 'x' '$' '0' '$' '0']
 ['0' 'x' '0' 'x' '0' '0' '0' '0' '0' '0' '$' 'x' '0' 'x' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' 'x' 'x' 'x' '$' '0' '0' '0' '0' '0']
 ['0' '0' '$' '0' '0' '$' '0' '0' '0' 'a' 'x' '$' '$' '0' '$' '0']
 ['0' '0' 'x' 'x' '0' '0' '0' 'x' 'x' '0' '0' 'x' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '$' '$' '0' '0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' 'x' 'x' 'x' '0' '$' '0' '0' '0' '0']
 ['$' '0' '0' '0' '$' 'x' 'x' '0' '$' '0' '0' '$' '0' '0' '0' '0']]
```

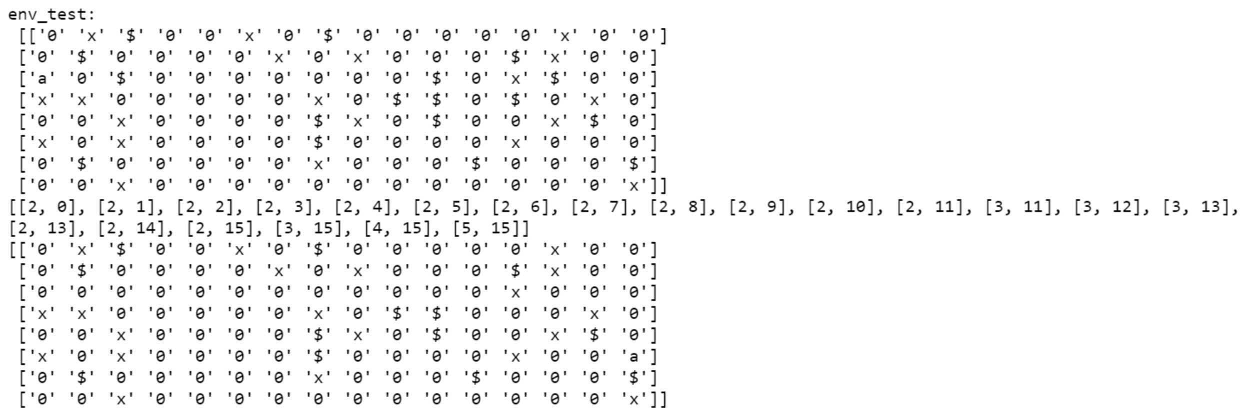
Figure 12: Equal Weights Behavior.

Figure 12 shows the behavior in a randomly generated gridworld with equal weights setup. The top gridworld is the initial setup, the list in the middle shows the agent's path, and the bottom gridworld is the result after the run. At here, we can observe that, starting from $[2, 0]$, the agent goes right, collect the litter at $[2, 6]$, goes down to third row to avoid the obstacle at $[2, 7]$, goes right, collect the litter at $[3, 8]$, but then steps back and forth between $[3, 9]$ and $[4, 9]$ until the step limit is reached.

The agent performed decently at start, but then stuck in the middle. It is probably because there are too many obstacles around $[3,9]$ and $[4,9]$, and the weight assigned to `q_goal` (the Q-values guide the agent to reach the goal) is not enough.

3.2 Increase Weight on Goal

We increase the weight assigned to goal module to 1.5. Now, the combined final entry becomes `q_combined = q_sd + q_obs + q_lit + 1.5*q_goal`.



```
env_test:
 [['0' 'x' '$' '0' '0' 'x' '0' '$' '0' '0' '0' '0' '0' 'x' '0' '0']
 ['0' '$' '0' '0' '0' '0' 'x' '0' 'x' '0' '0' '0' '$' 'x' '0' '0']
 ['a' '0' '$' '0' '0' '0' '0' '0' '0' '0' '$' '0' 'x' '$' '0' '0']
 ['x' 'x' '0' '0' '0' '0' '0' 'x' '0' '$' '$' '0' '$' '0' 'x' '0']
 ['0' '0' 'x' '0' '0' '0' '0' '$' 'x' '0' '$' '0' '0' 'x' '$' '0']
 ['x' '0' 'x' '0' '0' '0' '0' '$' '0' '0' '0' '0' 'x' '0' '0' '0']
 ['0' '$' '0' '0' '0' '0' '0' 'x' '0' '0' '0' '$' '0' '0' '0' '$']
 ['0' '0' 'x' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' 'x']]
[[2, 0], [2, 1], [2, 2], [2, 3], [2, 4], [2, 5], [2, 6], [2, 7], [2, 8], [2, 9], [2, 10], [2, 11], [3, 11], [3, 12], [3, 13],
[2, 13], [2, 14], [2, 15], [3, 15], [4, 15], [5, 15]]
[['0' 'x' '$' '0' '0' 'x' '0' '$' '0' '0' '0' '0' '0' 'x' '0' '0']
 ['0' '$' '0' '0' '0' '0' 'x' '0' 'x' '0' '0' '0' '$' 'x' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' 'x' '0' '0' '0']
 ['x' 'x' '0' '0' '0' '0' '0' 'x' '0' '$' '$' '0' '0' '0' 'x' '0']
 ['0' '0' 'x' '0' '0' '0' '0' '$' 'x' '0' '$' '0' '0' 'x' '$' '0']
 ['x' '0' 'x' '0' '0' '0' '0' '$' '0' '0' '0' '0' 'x' '0' '0' 'a']
 ['0' '$' '0' '0' '0' '0' '0' 'x' '0' '0' '0' '$' '0' '0' '0' '$']
 ['0' '0' 'x' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' 'x']]
```

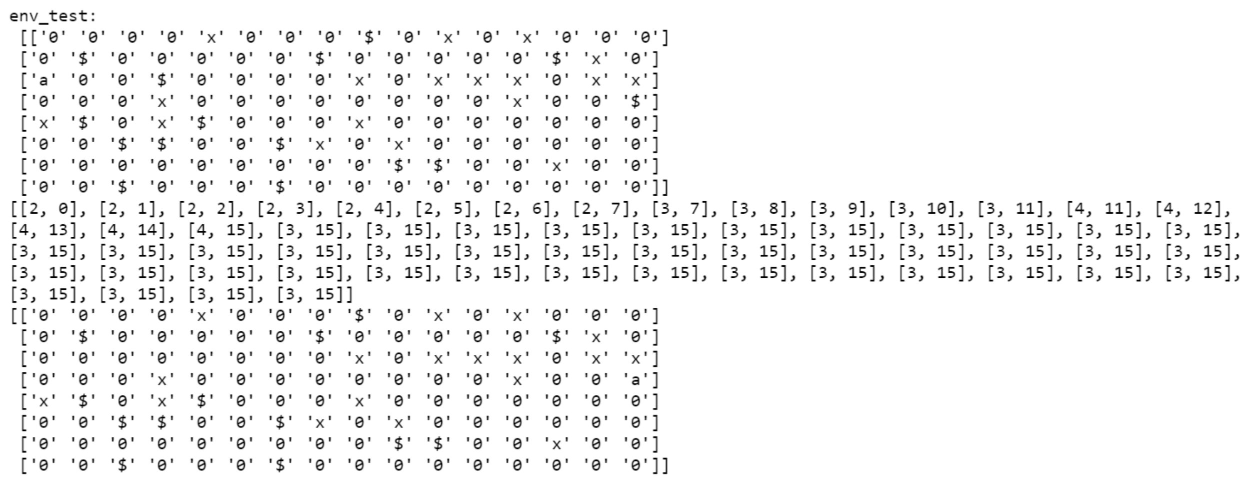
Figure 13: Increased Weight on Reaching Goal.

Figure 13 shows a sample run after we increased the weight on `q_goal`. It is a decent run. The agent successfully reaches the goal. Moreover, we can observe that the agent performs an "S-shaped" curve during $[2,11]$, $[3,11]$, $[3,12]$, $[3,13]$, $[2,13]$, $[2,14]$, $[2,15]$, $[3,15]$ when it tries to avoid the obstacles at $[2,12]$, $[4,13]$ and $[3,14]$.

However, we can also see that the agent did not pick up the litters at $[3,9]$, $[3,10]$ or $[4,10]$; instead, it chooses to go straight right in the second row and only goes downward to third row after it senses the obstacle at $[2,12]$. If we also increases the weight assigned on the litter `q_lit`, it will probably pick up more litters.

3.3 Increase Weight on Litter

Based on the previous setup, we increase the weight assigned to litter module to 2. Now, the combined final entry becomes `q_combined = q_sd + q_obs + 2*q_lit + 1.5*q_goal`.



```
env_test:
 [['0' '0' '0' '0' 'x' '0' '0' '0' '$' '0' 'x' '0' 'x' '0' '0' '0']
 ['0' '$' '0' '0' '0' '0' '0' '$' '0' '0' '0' '0' '0' '$' 'x' '0']
 ['a' '0' '0' '$' '0' '0' '0' '0' 'x' '0' 'x' 'x' 'x' '0' 'x' 'x']
 ['0' '0' '0' 'x' '0' '0' '0' '0' '0' '0' '0' '0' 'x' '0' '0' '$']
 ['x' '$' '0' 'x' '$' '0' '0' '0' 'x' '0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '$' '$' '0' '0' '$' 'x' '0' 'x' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' '0' '0' '$' '$' '0' '0' 'x' '0' '0']
 ['0' '0' '$' '0' '0' '0' '$' '0' '0' '0' '0' '0' '0' '0' '0' '0']]
[[2, 0], [2, 1], [2, 2], [2, 3], [2, 4], [2, 5], [2, 6], [2, 7], [3, 7], [3, 8], [3, 9], [3, 10], [3, 11], [4, 11], [4, 12],
[4, 13], [4, 14], [4, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15],
[3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15],
[3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15], [3, 15],
[3, 15], [3, 15], [3, 15], [3, 15]]
[['0' '0' '0' '0' 'x' '0' '0' '0' '$' '0' 'x' '0' 'x' '0' '0' '0']
 ['0' '$' '0' '0' '0' '0' '0' '$' '0' '0' '0' '0' '0' '$' 'x' '0']
 ['0' '0' '0' '0' '0' '0' '0' '0' 'x' '0' 'x' 'x' 'x' '0' 'x' 'x']
 ['0' '0' '0' 'x' '0' '0' '0' '0' '0' '0' '0' '0' 'x' '0' '0' 'a']
 ['x' '$' '0' 'x' '$' '0' '0' '0' 'x' '0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '$' '$' '0' '0' '$' 'x' '0' 'x' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' '0' '0' '$' '$' '0' '0' 'x' '0' '0']
 ['0' '0' '$' '0' '0' '0' '$' '0' '0' '0' '0' '0' '0' '0' '0' '0']]
```

Figure 14: Increased Weight on Litter.

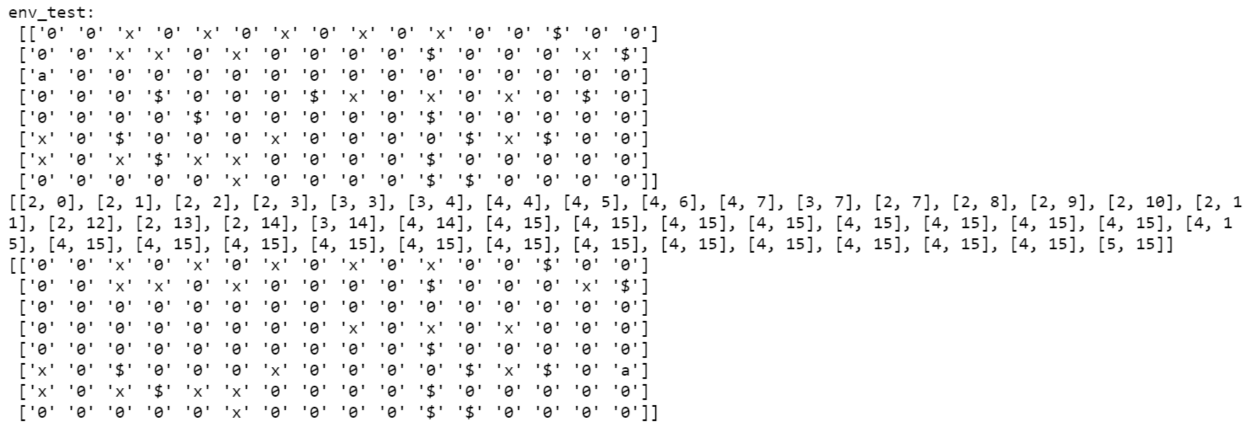
This is an interesting run. Like before, the agent successfully avoids all the obstacles and remains on the sidewalk. After it reaches $[4, 15]$, it decides to pick up the litter above at $[3, 15]$ instead of going down to $[5, 15]$ the goal. This is probably due to our increased weight assigned to the litter. However, after it picks up the litter, it stays at $[3, 15]$ until the step limit is reached. Though we cannot absolutely determine the reason for such behavior, we can speculate that there is a potential trade-off between the different weights assigned: if we increase the weights on obstacles or litters, the agent might perform better on avoiding obstacles or pick up litters, or both, but it may fail to reach the goal; if we increase the weight on goal, it may go straight into the goal and step onto the obstacles. There are many possible ways to make improvements. One way I come up with is to increase the weights assigned to sidewalk, obstacle, or litter when the number of steps taken is below a certain threshold. In this way, the agent might pick up more litters. After the threshold is reached, it means that the agent has not yet reached the goal (probably because it gets stuck between two grids or on the edge like situations in Figure 12 and Figure 14). In that way, we will reassign the weights, decreasing the weights on sidewalk, obstacle, or litter and increasing the weight assigned on the goal, so the agent will prioritize reaching the goal in the last several steps before the limit is reached.

3.4 Threshold Weight Setting

Based on the idea discussed earlier, I set the threshold in the following way:

```
if i <= 40:
    q_combined = 3*q_sd + 1.5*q_obs + 8*q_lit + q_goal
else:
    q_combined = q_sd + 0.5*q_obs + q_lit + 5*q_goal
```

Namely, if the step number is less than or equal to 40, we prioritize picking up litters, staying on sidewalk, and avoiding obstacles. If the step number surpasses 40, we decrease the weight on obstacle and greatly increase the weight on the goal so that, if there are several obstacles surrounding the agent and resulting in a deadlock, the agent will step onto them to pass the obstacles, break the deadlock, and head towards the goal.



```
env_test:
 [['0' '0' 'x' '0' 'x' '0' 'x' '0' 'x' '0' 'x' '0' '0' '$' '0' '0']
 ['0' '0' 'x' 'x' '0' 'x' '0' '0' '0' '0' '$' '0' '0' '0' 'x' '$']
 ['a' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '$' '0' '0' '0' '$' 'x' '0' 'x' '0' 'x' '0' '$' '0']
 ['0' '0' '0' '0' '$' '0' '0' '0' '0' '0' '$' '0' '0' '0' '0' '0']
 ['x' '0' '$' '0' '0' '0' 'x' '0' '0' '0' '0' '$' 'x' '$' '0' '0']
 ['x' '0' 'x' '$' 'x' 'x' '0' '0' '0' '0' '$' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' 'x' '0' '0' '0' '0' '$' '$' '0' '0' '0' '0']]
[[2, 0], [2, 1], [2, 2], [2, 3], [3, 3], [3, 4], [4, 4], [4, 5], [4, 6], [4, 7], [3, 7], [2, 7], [2, 8], [2, 9], [2, 10], [2, 1
1], [2, 12], [2, 13], [2, 14], [3, 14], [4, 14], [4, 15], [4, 15], [4, 15], [4, 15], [4, 15], [4, 15], [4, 15], [4, 15], [4, 1
5], [4, 15], [4, 15], [4, 15], [4, 15], [4, 15], [4, 15], [4, 15], [4, 15], [4, 15], [4, 15], [4, 15], [4, 15], [5, 15]]
[['0' '0' 'x' '0' 'x' '0' 'x' '0' 'x' '0' 'x' '0' '0' '$' '0' '0']
 ['0' '0' 'x' 'x' '0' 'x' '0' '0' '0' '0' '$' '0' '0' '0' 'x' '$']
 ['0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' '0' 'x' '0' 'x' '0' 'x' '0' '0' '0']
 ['0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '$' '0' '0' '0' '0' '0']
 ['x' '0' '$' '0' '0' '0' 'x' '0' '0' '0' '0' '$' 'x' '$' '0' 'a']
 ['x' '0' 'x' '$' 'x' 'x' '0' '0' '0' '0' '$' '0' '0' '0' '0' '0']
 ['0' '0' '0' '0' '0' 'x' '0' '0' '0' '0' '$' '$' '0' '0' '0' '0']]
```

Figure 15: Threshold Weight Assignment.

Figure 15 shows an interesting run. We can see that there is no obstacles on the second row, but, instead of walking straight right from $[2, 0]$ to $[2, 15]$ like what it would act before, the agent picks up many litters on the third row and fourth row, including $[3, 3]$, $[4, 4]$, $[3, 7]$ and $[3, 14]$. Then, similar to what happened in Figure 14, it gets stuck at $[4, 15]$. But now, after the step number surpasses 40, we change the weight and heavily prioritize on reaching the goal, and so the agent breaks the deadlock and moves down to reach the goal.

There are many other ways to set the threshold. Besides basing on step number, we can also set it according to specific row or column range, whether the agent is on the sidewalk or not, the number of litters remained in the environment, etc. We can also set multiple thresholds to allow more versatility of the agent's behavior. However, we should also note that setting multiple or very specific thresholds will lead to more complexity, which might in turn negatively affect the agent's behavior.

4 Summary

We developed a modular reinforcement learning process on a multitasking walking agent, and it achieved relatively decent performance. However, there are still many possible extensions, improvements and investigations on this project. The size of the gridworld, the shape of the sidewalk (we set it to be rectangular here, but it can be more complicated), the choice and representation of state space, the learning algorithm and parameters, as well as the ways to combine different modules discussed earlier and many other potential factors, can all have influence over the agent's learning process and behavior.

5 References

- `numpy` library
- Ballard, D., Robinson, A., and Sprague, N. 2007. Modeling Embodied Visual Behaviors, *ACM Transactions on Applied Perception* for problem background
- Marsland, S. 2015. *Machine Learning An Algorithmic Pespective* 2nd ed. Taylor Francis Group, LLC. Chapter 11. Specifically, the ϵ-greedy and Q-learning algorithm are modified based on the code block presented in page 243-244.