Amirabbas Rezasoltani HW1 610399205

11, 13, 2023 AI

1) a) To find optimal policy in MDP, Q-learning is used. Suppose in one of the episods, agent is in a particular state S. So based on updating function of q-learning, $Q(s,a) = (1-\alpha) Q(s,a) + \alpha (r_{s'} + \lambda \max_a Q(s',a))$, which s' is next state after doing action a. $r_{s'}$ is changed to $r_{s'} + c$. Assume all cells of q-table are updated sufficient times. To update $Q(s,a)$, $Q(s,a) = (1-\alpha) Q(s,a) + \alpha (r_{s'} + c + \lambda \max_{a'} Q(s',a'))$.
assume $Q(s',a)$ is changed by $n_{s',a}$. So $Q(s,a)$ is changed $\alpha c + \alpha \lambda n_{s',a'}$. Also $\max_{a'} Q(s',a')$ is changed by $\alpha c + \alpha \lambda n_{s'',a''}$ where s'' is next state of $\max Q(s',a)$ and a'' is $\max_{a''} Q(s'',a'')$. So $Q(s,a)$ is changed

$$\alpha c + \alpha \lambda (\alpha c + \alpha \lambda n_{s'',a''}) = \alpha c + \alpha^2 \lambda c + \alpha^2 \lambda^2 n_{s'',a''}.$$ Continueing computing $n_{s'',a''}$, we can show update for $Q(s,a)$ is

$$\alpha c + \alpha^2 \lambda c + \alpha^3 \lambda^2 c + \dots = \alpha c (1 + \alpha \lambda + \alpha^2 \lambda^2 + \dots) = \boxed{\alpha c \times \frac{1}{1-\alpha\lambda}}$$

since $Q(s,a)$ was arbitrary, all cells of table are added with $\frac{\alpha c}{1-\alpha\lambda}$. If update continues, this added value would be cumalative and will be a constant cofficient of $\frac{\alpha c}{1-\alpha\lambda}$. so optimal policy doesn't change.

b) Assume all cells of Q-table are updated after multiplying all rewards by c. So to update $Q(s,a)$, we have,

$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r_{s'} \times c + \lambda \max_{a'} Q(s',a'))$.

assume $Q(s',a')$ is added by $x_{s',a'}$. So $Q(s,a)$ is changed by $(\alpha r_{s'}c - \alpha r_{s'}) + \lambda\alpha x_{s',a'}$. So $Q(s,a)$ is changed by $\alpha r_{s'}(c-1) + \lambda\alpha x_{s',a'}$. Also $x_{s',a'}$ is $\alpha r_{s''}(c-1) + \lambda\alpha x_{s'',a''}$ (s'',a'' is introduced in previous part). So $Q(s,a)$ is updated by

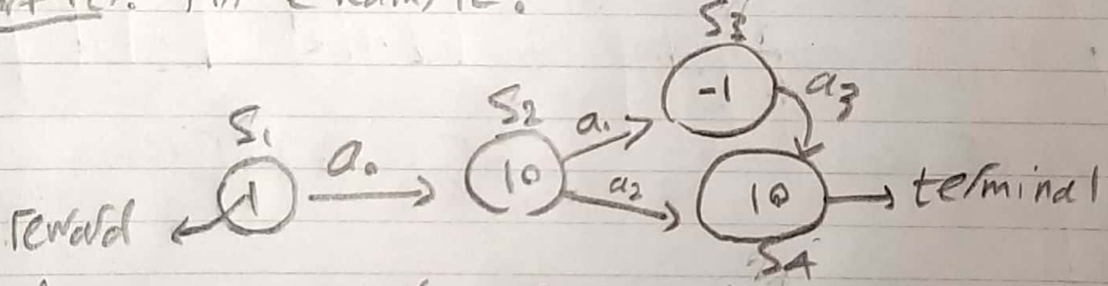
$\alpha r_{s'}(c-1) + \lambda\alpha(\alpha r_{s''}(c-1) + \lambda\alpha x_{s'',a''}) = \alpha r_{s'}(c-1) + \lambda\alpha^2 r_{s''}(c-1) +$

$\lambda^2\alpha^2 x_{s'',a''}$. By continueing computations, $Q(s,a)$ is added by $\alpha(c-1)(r_{s'} + \lambda\alpha r_{s''} + \dots)$. It depends on value of r_s that a state can be more valueable or less valueable. If $c=1$, optimal π and values of $Q(s,a)$s don't differ. If $c \neq 1$, values and optimal policy π may differ.

If $c=0$, all rewards become 0 and so all cells of Q-table remain 0. so all policies are optimal.

clips

c) Assume terminal state S_{end}. There is no valid action for that so Qs related to that won't update and its value is 0. So like part a, value change for $Q(s,a)$ is $\alpha c(1+\alpha\lambda+\alpha^2\lambda^2+\dots+\alpha^k\lambda^k)$. Because steps to terminal state for each state differs, amount of update for $Q(s,a)$ differs. I mean if there are k steps from state s to terminal, update is $\alpha c(1+\alpha\lambda+\dots+\alpha^k\lambda^k)$. So optimal π may differ. for example:



for diagram above, optimal policy π is:

$s_1 \to a_0$

$s_2 \to a_2$

$s_3 \to a_3$

But for $c=2$, optimal policy π will be $s_1 \to a_0$

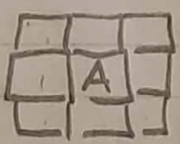
$s_2 \to a_1$

$s_3 \to a_3$

2) a) To obtain what state we are in, check 8 cells alound agent. In each cell, either there is wall, or food, or is empty or is invalid move. So number of states is maximum 4^8. But some states are never happen. For example when all cells are invalid (out of bound), or all states are wall. So these states number can be reduced.

b) Actions: {up, right, bottom, left}. But in states which doing action causes going to invalid (out of bound) cell, this action is banned.

State: check 8 cells alound agent like this.

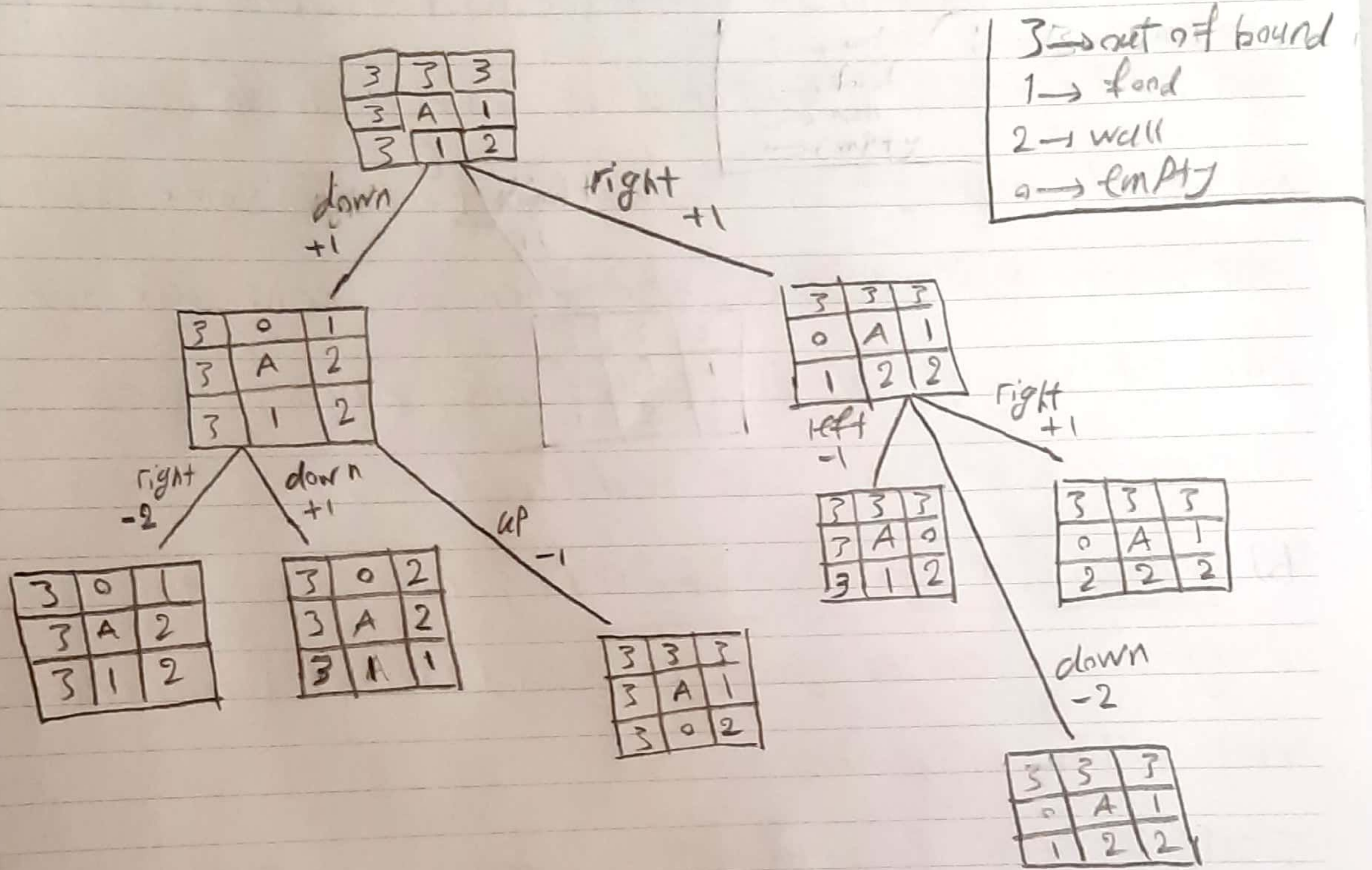


where agent is in cell A. If cell is empty, its value is 0, if food is on that, value is 1, if wall is on that, its value is 2 and if it's out of bound, its value is 3. Using this, we can code all states.

Reward: Based on value of each cell, reward function is as this: R(wall)=-2, R(food)=1, R(empty)=-1. The goal is find if all foods are eaten.

3) Is answered in code.

4) in episod 1, state of agent is like below for given map.



5) As mentioned, to define state, 8 cells around agent is checked and based on them state is specified. so there are 4^8 states, since in each cell around, either there is wall, or food, or is out of bound or is empty. Also for each state, there are 4 moves (right, up, left, down). But moving out of boundary is invalid so in q-table, $-\infty$ is placed for action which cause invalid moves as symbol. After updating q-table, for first state (root in tree in question 4):

down, right, up, left have values $1.108, 2.667, -\infty, -\infty$ in order. For right node of root, values are $-0.33, 2.67, -\infty, -0.31$ in order. For left node of root, values are $2.67, -0.6, -0.47, -\infty$ in order. These results is gathered from game with $\lambda = 0.25$ and $\alpha = 0.2$, for map A.

6) In code, there are 2 version of maps. map A is given in question and map B is arbitrary.

7) Graphic part is added to code.

8) A version of pacman with ghost is implemented. what differs from simple pacman is, in Q-table, there can be 8^8 states. Becaus in every cell, there can be ghost or not. But it can be reduced. Because ghost is unique. so there are 2 total case. Either ghost is in 8 cells or not. if ghost is, so there are 8 cells which ghost can be. so there are only 9×4^8 states.