6CCS3AIN Coursework

CHAOYU DENG

December 12, 2020

1 Introduction and Overview

In this coursework, I combined a modified wavefront pathfinder and MDP to create a Pacman solver and got a remarkable result in both small Grid and medium Classic.

2 MDP Solver

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s') \tag{1}$$

According to the Bellman equation (1), every position on the Pacman map is a state(s) with five different actions (a | a ∈ {up, down, left, right, stop }) and for each state, they have different rewards according to different environments.

2.1 Discount

The discount factor in this equation is critical that it decide the ratio of present reward to future reward. If discount close to 0, the Pacman will tend to get immediate benefits, otherwise the long-term benefit is as significant as the present reward. In Pacman, reducing discount can avoid the incorrect future prediction by MDP, while increasing discounts can increase the appeal of food or capsules but sometimes it is not the best choice to aim for food first.

2.2 Optimal policies

In order to find the optimal policies, I use value iteration with a tolerant, which can reduce the number of iterations rather than compare with 17 decimal places of float in python.

2.3 Terminal States and Reward

In terms of terminal states, as the map changes as we move, no state can be regarded as a terminal state although I found that setting food as a terminal state not only gave a more reasonable value (easy to debug and understand) but also reduced the number of iterations (from 92 to 50 in the medium map) with similar results.

In term of reward, In order to make Pacman trend to escape when it gets closer to the ghost, but return to relatively normal behaviour when it is far away from the ghost, I choose to use the inverse proportional function as value function. For all states except ghosts, I use a similar reward function (2) {1} and a negative reward for the ghost. After the MDP solver, Pacman will find a balance between eating the food closest to him and the food remotest from the ghost according to different parameters.

$$(\text{the initial reward of this state}) - (\frac{\text{max penalty constant}}{\text{distance to ghost}} - 1) \tag{2}$$

2.3.1 Distance to ghost

After testing, I found that the actual distance is more useful than the Manhattan distance because ghosts only go back when they encounter dead ends, which means it is safe to follow ghosts sometimes and we can eat more foods even though the ghost is close to us.

- Modified Wavefront {2}

  Wavefront is a pathfinding algorithm like BFS; it starts from the origin and increases the distance of neighbouring position until the target point is found. In this task, except for the first action, I store the facing direction of the ghost, and then only expand the neighbouring points that ghost can walk each step, and when the ghost encounter the end, I will start from there and take the shortest distance until each point is traversed twice. Because I use a list as a queue and a dictionary, so the time complexity is O(2n) = O(n), n is the total number of points in the map. Moreover, If there are two or more ghosts, I will take the minimum distance ,and if they are scared, I will add the time of scare to each distance value.

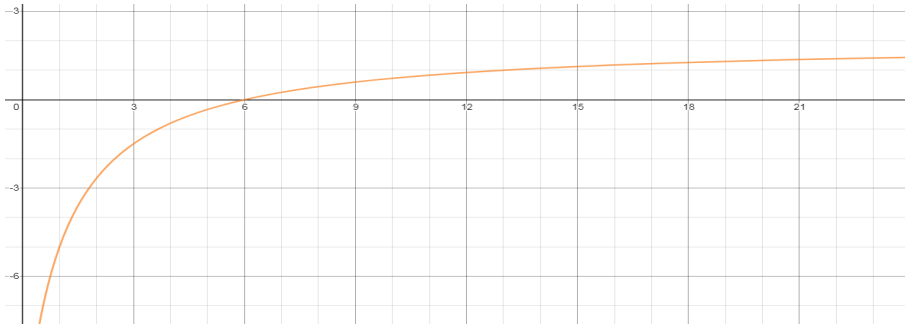


Figure 1: function: 2 - $(\frac{14}{x+1})$

- Map basic strategy

  Common strategy: In the initial utility map {3}, except the wall is '%', and everything else is 0. Each state has an initial reward based on the reward function, and the rewards of food and capsules will change base on this value according to reward function too, which makes it easier for Pacman to judge whether the food is worth eating.

  Small Grid: Unless Pacman is only two grid away from the ghost, Pacman will eat the food in the bottom left corner first.

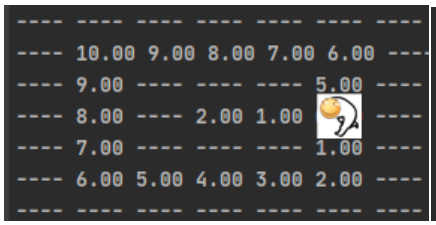


Figure 2: walk distance of ghost

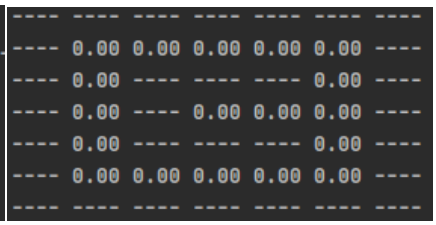


Figure 3: initial utility map

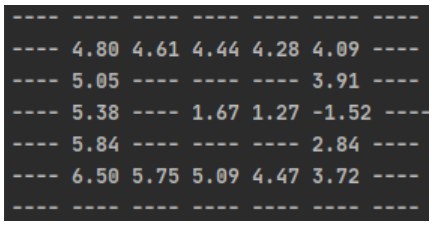
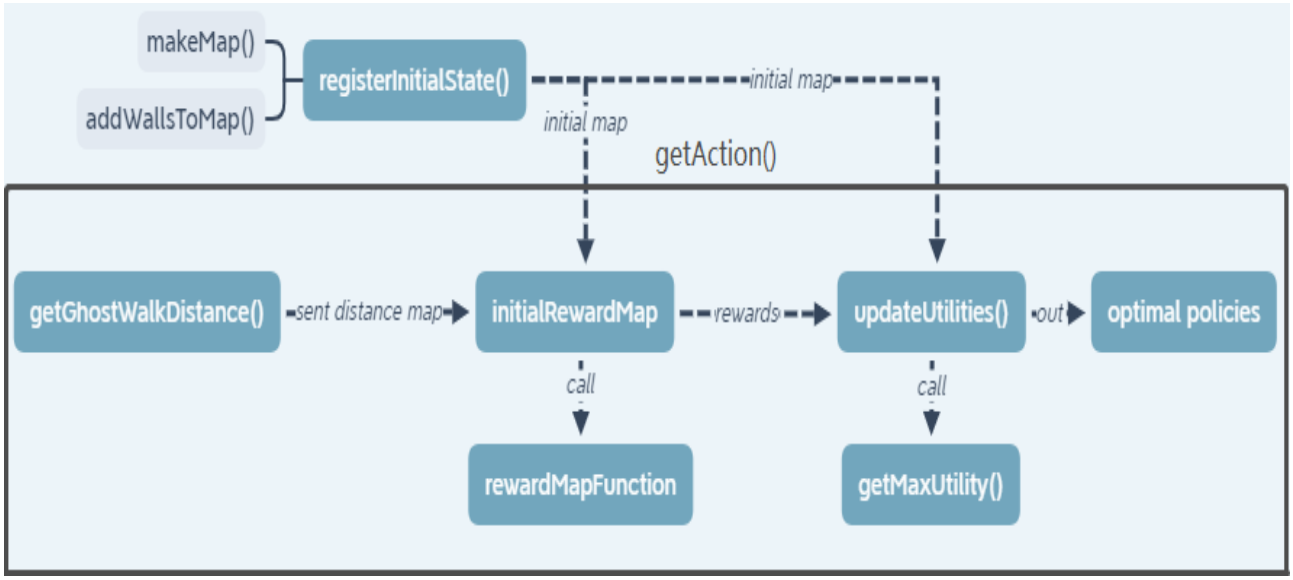


Figure 4: optimize utility map

3 Flow Chart of functions



4 Experiments and Results

After lots of tests, I found over 10000 rounds in the small grid and 500 rounds in medium Classic can get a more accurate win-rate in two decimal places, so in most of the test below I use over 10000 rounds in the small grid, and over 500 in the medium map and the empty block is meaningless test data

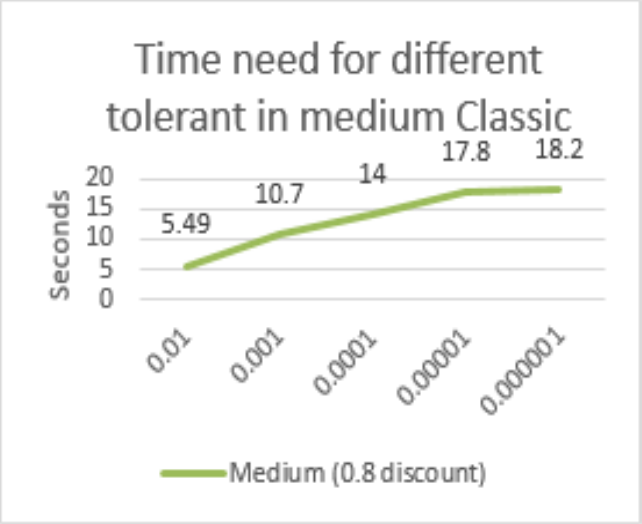
4.1 Average Time consuming (Seconds) and number of iterations

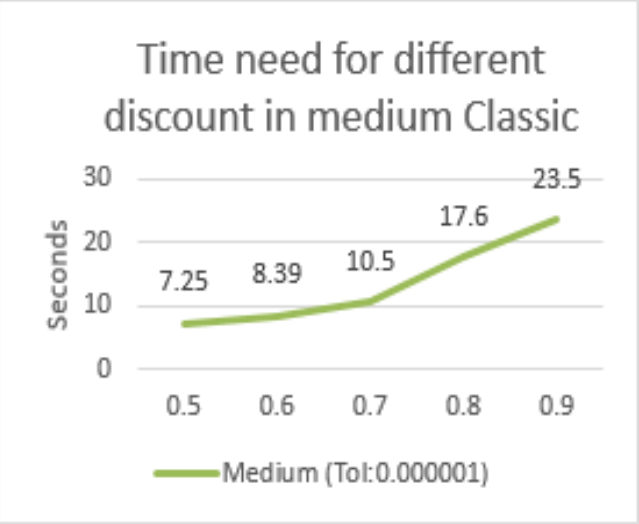
Layout \ Tolerant	0.01	0.001	0.0001	0.00001	0.000001
Small (0.8 discount)	0.120 (24)	0.161 (34)	0.205 (44)	0.23 (55)	0.276 (65)
Medium (0.8 discount)	5.49 (22.4)	10.7(32.7)	14.0 (43)	17.8 (53)	18.2 (64)

Table 1: preferment given layout and tolerant

Layout \ Discount	0.5	0.6	0.7	0.8	0.9
Small (Tol: 0.0001)		0.116 (20)	0.146 (28)	0.205 (44)	0.47 (92)
Medium (Tol: 0.000001)	7.25 (21)	8.39 (29)	10.5 (40)	17.6 (64)	23.5 (133)

Table 2: preferment given layout and discount





According to the figures above, we can quickly know that the decrease of tolerant and the increase in discount will increase the average number of iterations and the time of the win, especially in the bigger map. When tolerant is less than 0.00001, the growth rate of time becomes gentle. Conversely, the smaller the discount, the faster the time growth rate.

4.2 Wining rate and standard deviation in 25 runs

According to the above table [3], table [4] and figure {5} , a high tolerant will reduce the winning rate. I think this is because the Bellman equation has not converged to the optimal strategy, So the impact on the small Grid is relatively small. In term of discount, Because the value of the ghost will be covered by the food, a too low discount will cause Pac-Man to walk towards the food closing to the ghost and die, while a too high discount will cause Pac-Man to give up the food far away from it and choose to escape and unable to end the game, which will increase the chance of Pacman being eaten by ghosts. Also, the win-rate on the middle map has a high standard deviation, I cannot get an accurate win-rate with only 500 rounds.

4.3 Effect by Reward function

Different parameters can determine how far away from the ghost, the food will be attractive and how attractive it is. In small Grid, the best eatable distance between ghost and food is around 2, and the best initial reward is 1. In medium Classic, the best distance is around 6, and the best initial reward is 2.

In the end, I chose to use a discount of 0.8 and a tolerant of 0.0001 on the small Grid, a discount of 0.85 and a tolerant of 0.000001 on the meddle Classic. (The result is shown in figures {6})

5 Conclusion

The MDP solver can get an excellent result in Pac-Man. For the small Grid, I think that the highest win-rate is around 78% to 79% in over 10000 rounds, although it may be higher than 80% in 1000 rounds. For medium Classic, there is still room for optimization, such as deciding whether to eat the scared ghost or not according to the distance from the ghost's birth point and avoiding entering too long pipes to avoid ghosts' double trapping.

Layout \ Tolerant	0.01	0.001	0.0001	0.00001	0.000001
Small (discount: 0.8)	0.773 (2.16)	0.771 (2.11)	0.780 (2.06)	0.776 (2.03)	0.778 (1.99)
Medium (discount: 0.8)	0.694 (2.52)	0.712 (1.98)	0.718 (1.98)	0.714 (2.12)	0.734 (2.15)

Table 3: win-rate and std given layout and tolerant

Discount	0.6	0.7	0.8	0.9	
Small (Tol: 0.0001)	0.738 (2.23)	0.768 (2.08)	0.780 (2.06)	0.761 (2.27)	
Discount	0.75	0.8	0.85	0.9	0.95
Medium (Tol: 0.000001)	0.704 (2.19)	0.741 (2.09)	0.746 (2.07)	0.696 (2.09)	0.630 (2.15)

Table 4: win-rate and std given layout and discount

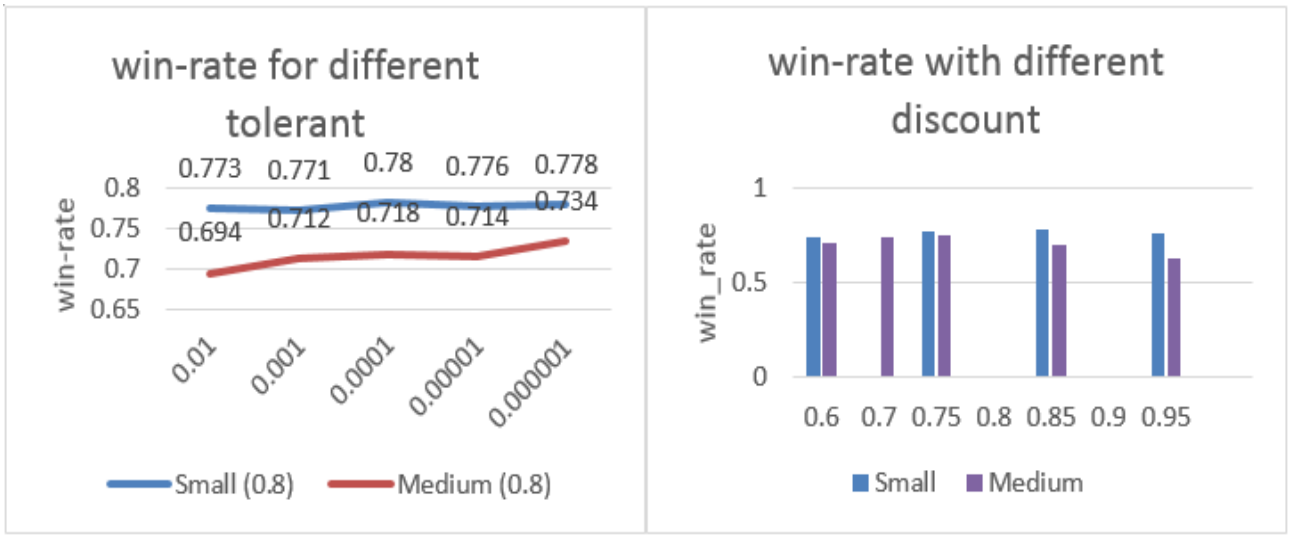


Figure 5: win-rate given layout, tolerant and discount

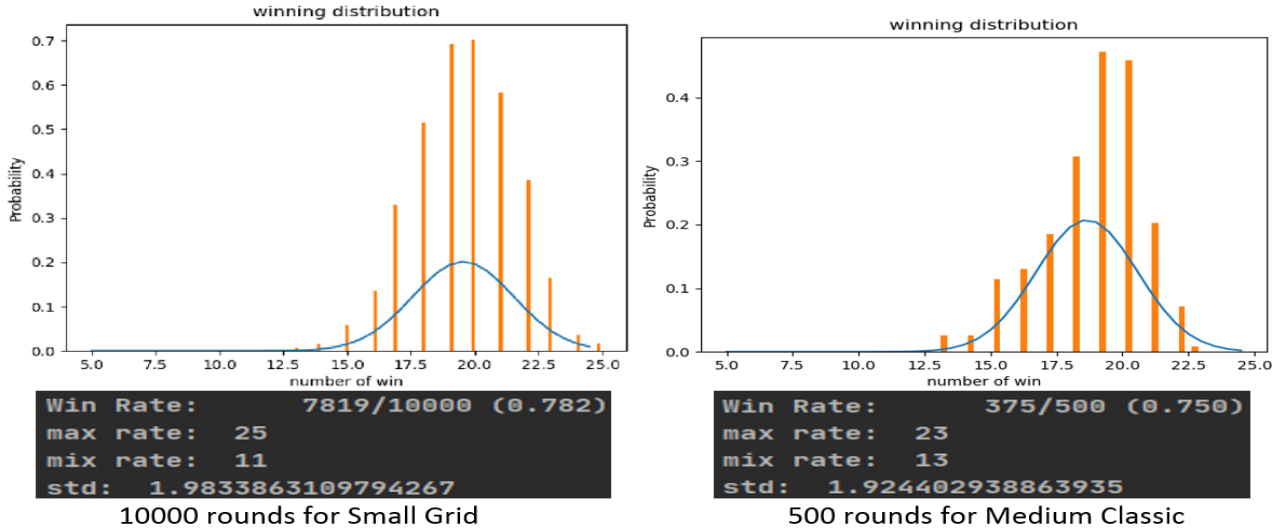


Figure 6: Distribution of win-rate