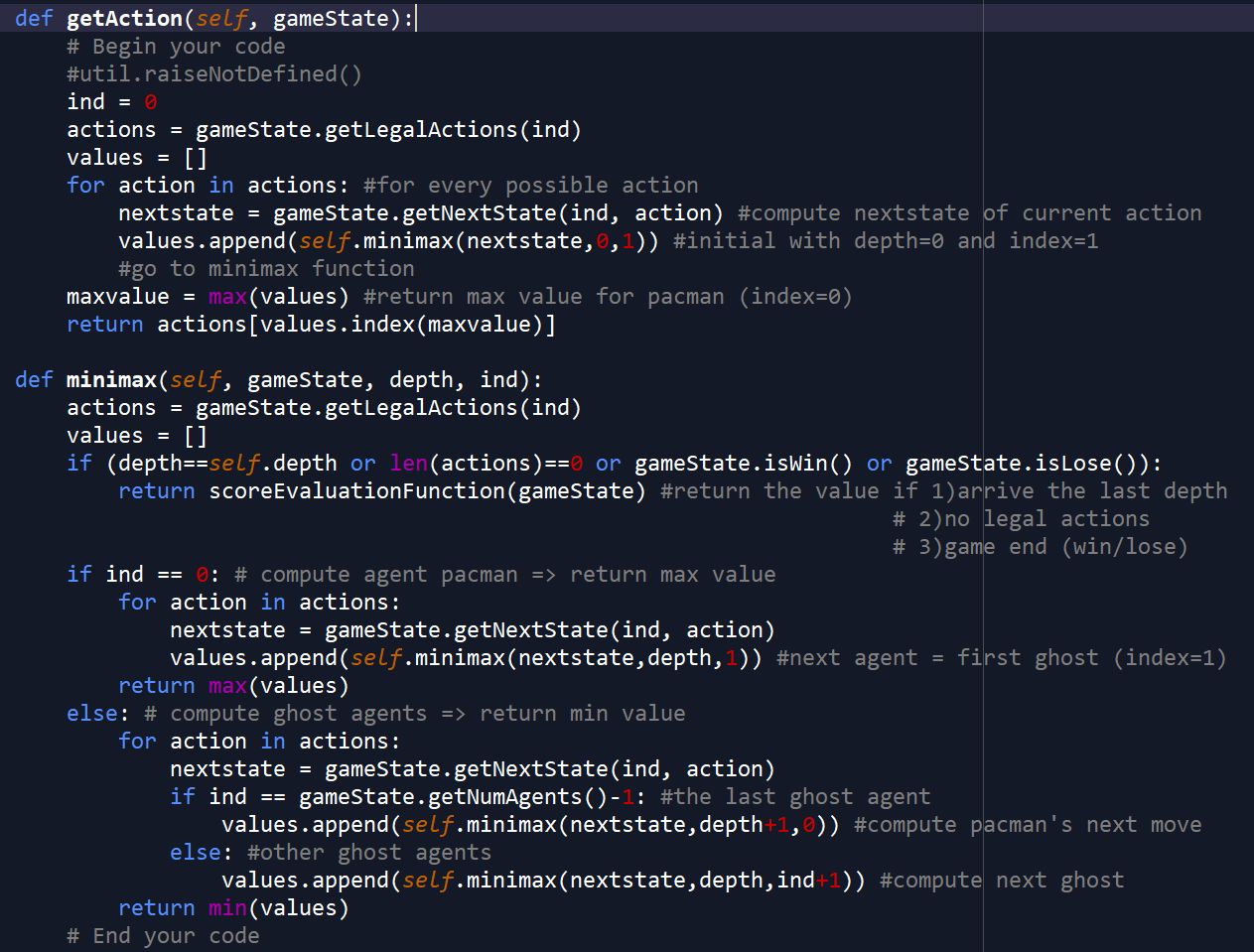
report of AI\_HW3 109550027紀竺均

1. implementation and explanation

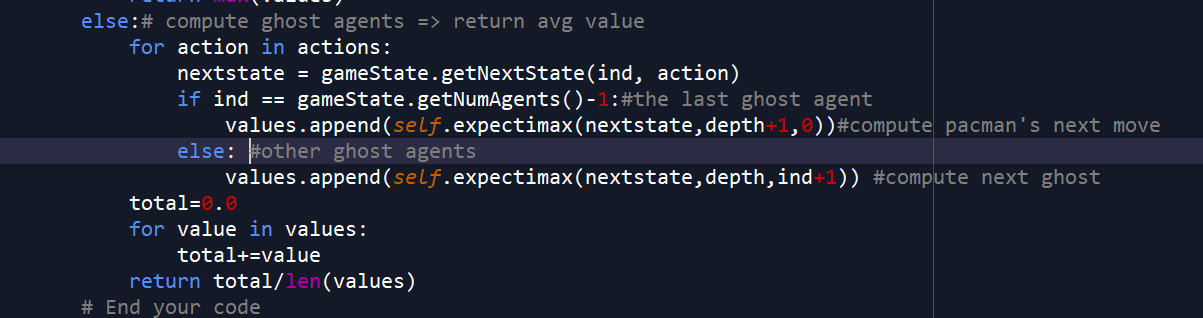
Part 1 : Adversarial search

1-1: Minimax Search

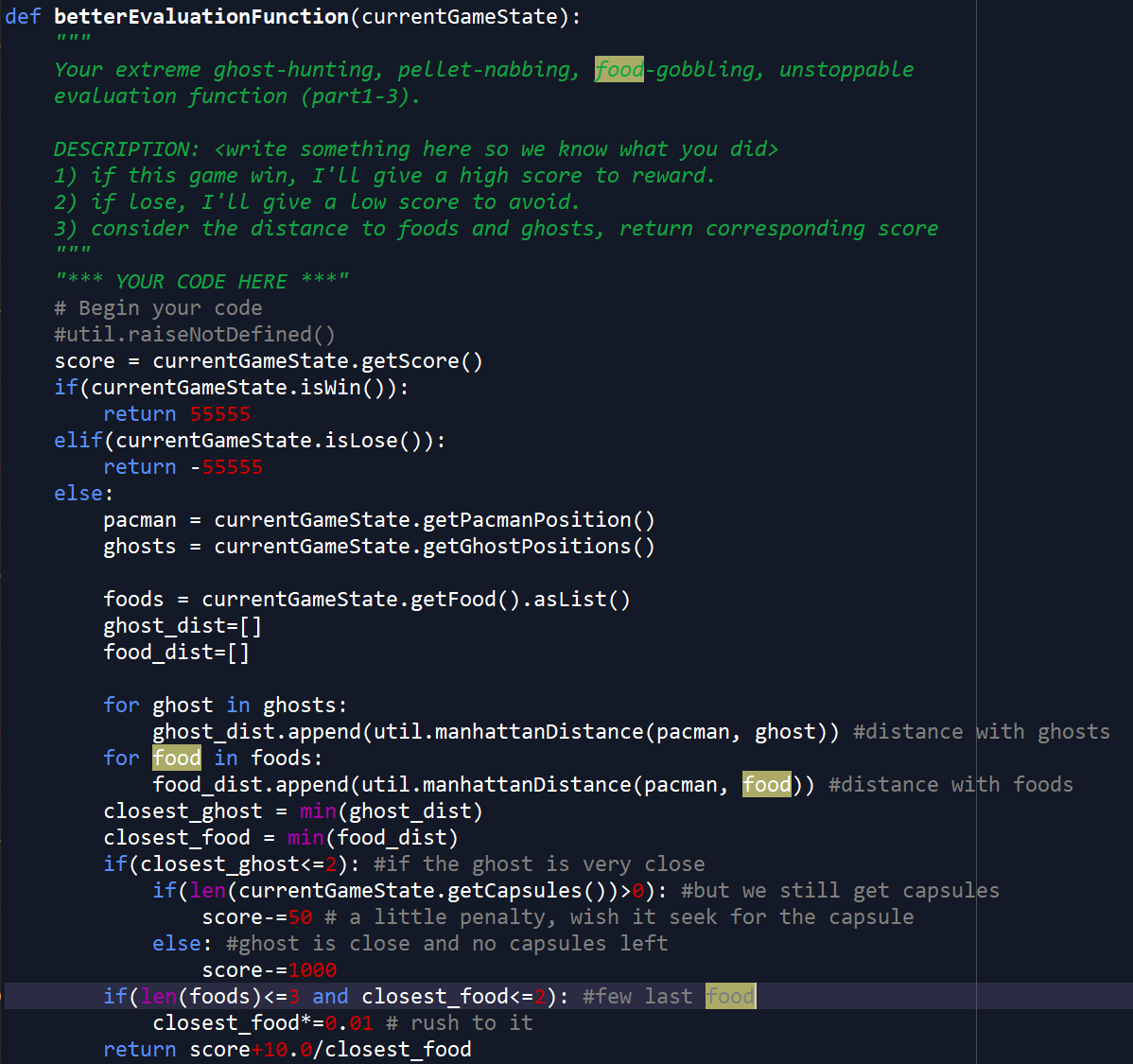


1-2: Expectimax Search

Almost same as minimax search, but instead of return min value for ghost agents, return the average values. This is because we assume the ghosts will choose their action randomly, every action will have the same probability.



1-3 better evaluation function

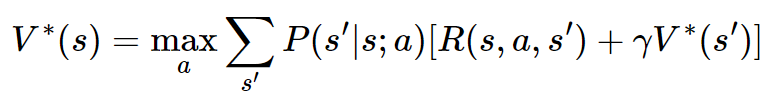


Part 2 : Q-learning

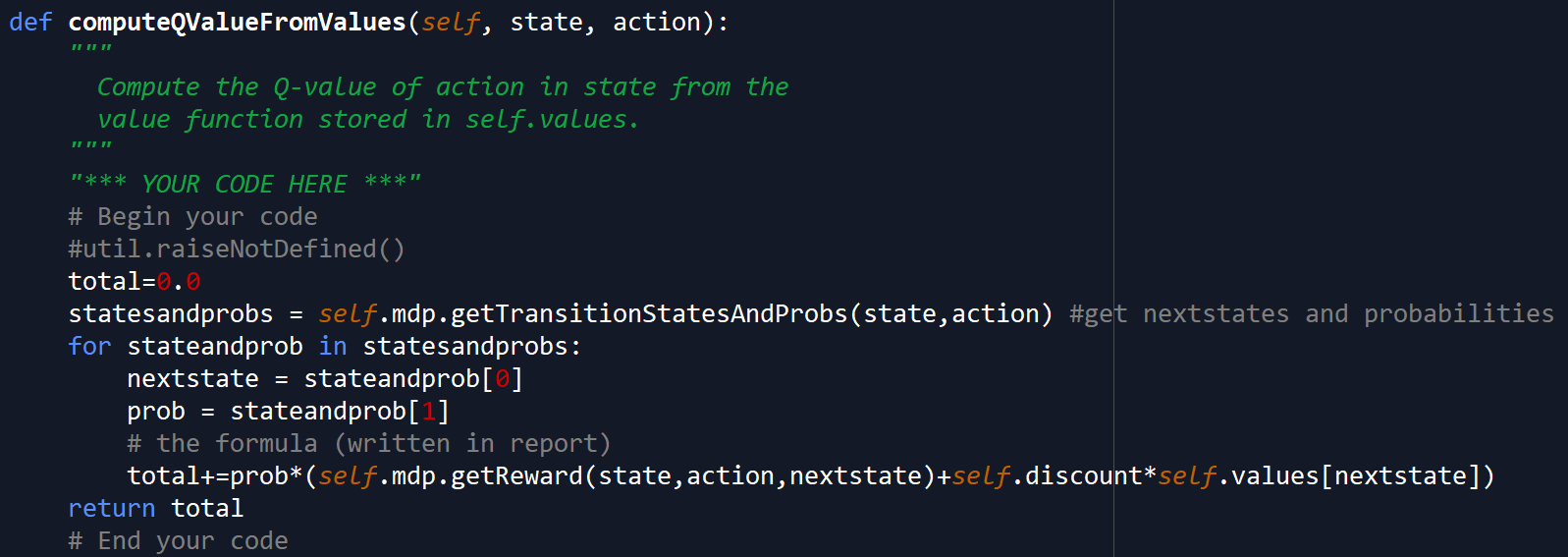
2-1: Value Iteration

The value iteration agent construct the value dictionary of every state base on Markov Decision Processes(MDP).

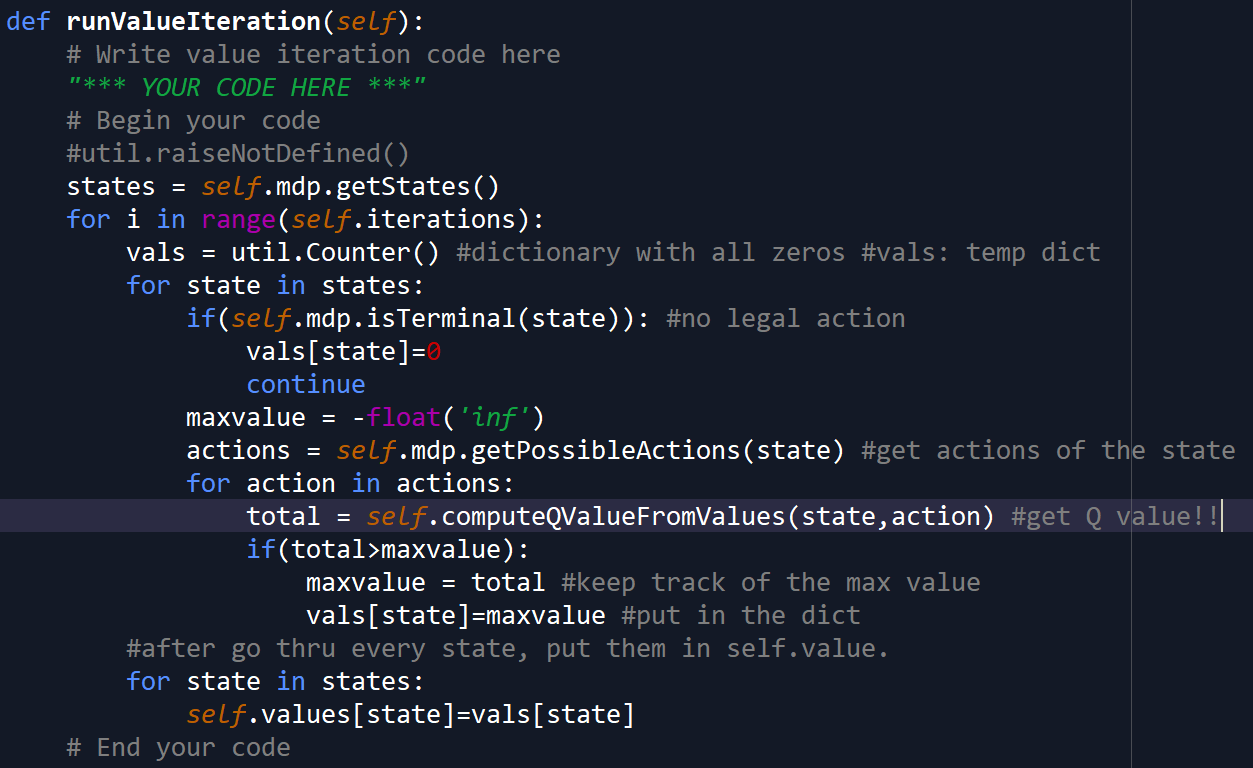
The formula can be written as:



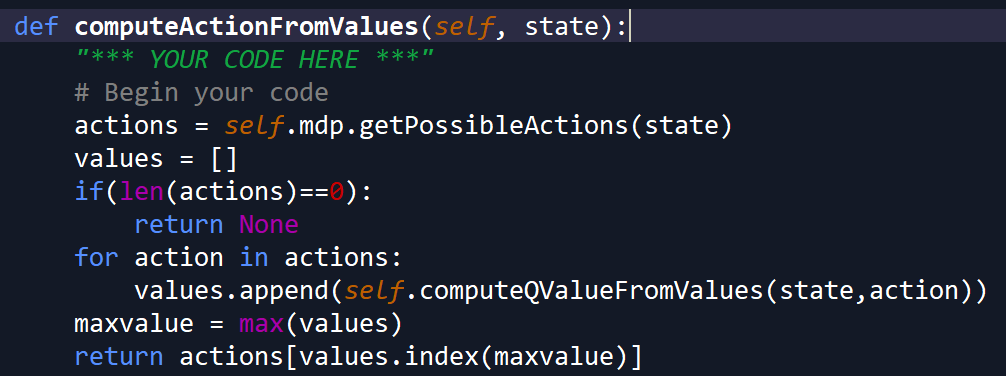
where P(s’|s;a) is the probs, R(s,a,s’) is the rewards, andγstands for discount.



Then, value iteration function. For #iterations, we should sum up all the states’ values and return self.values for every state.

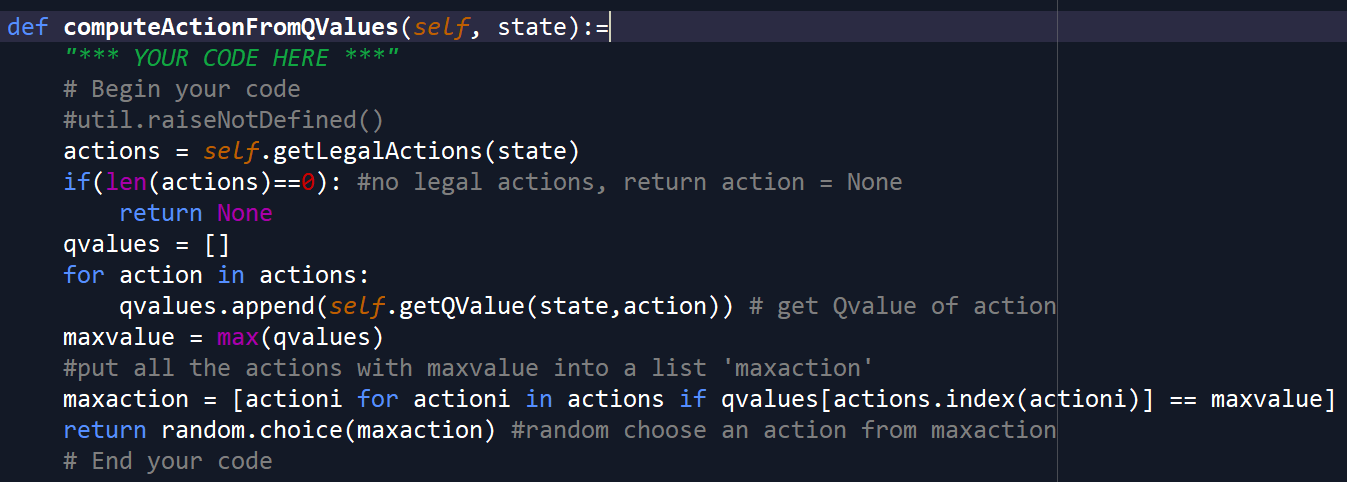


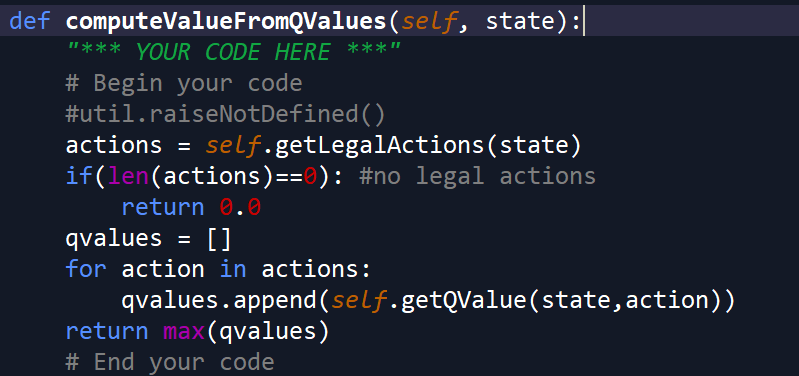
Finally, compute\_action\_from\_values function will compute actions corresponding to the max Q-value.



Part 2-2: Q-learning (10%)

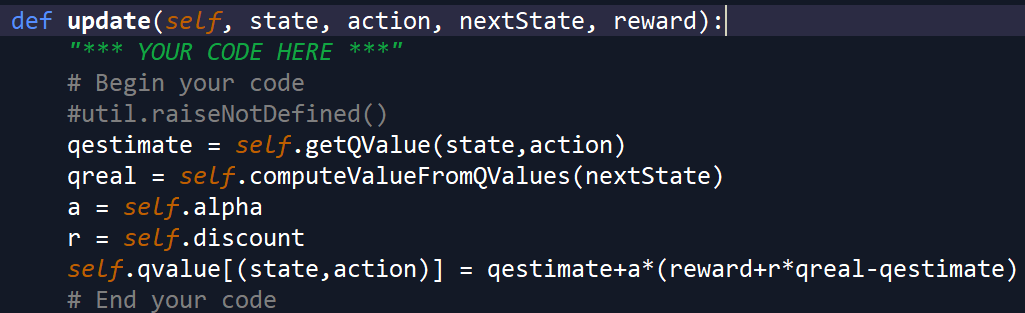
Q-learning updates Q-value base on Q-real and Q-estimate.



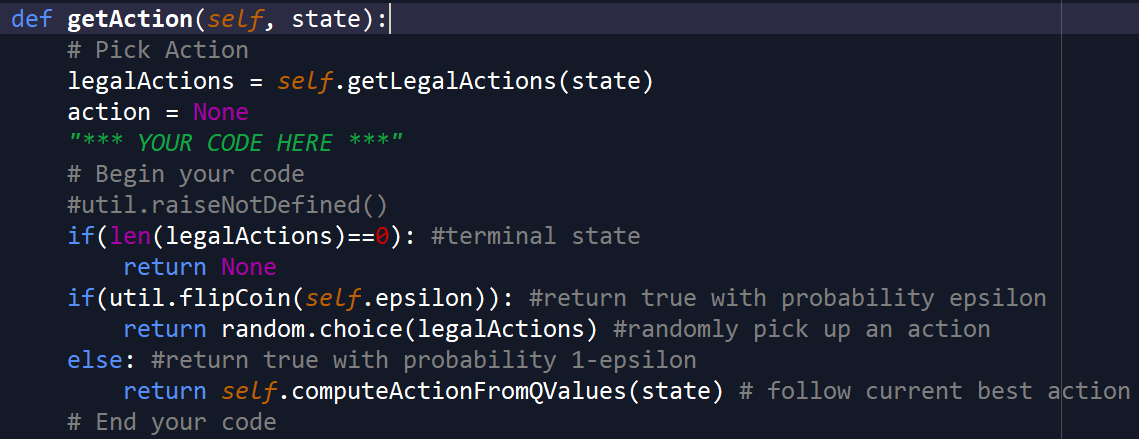


update Q-value’s formula:



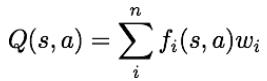


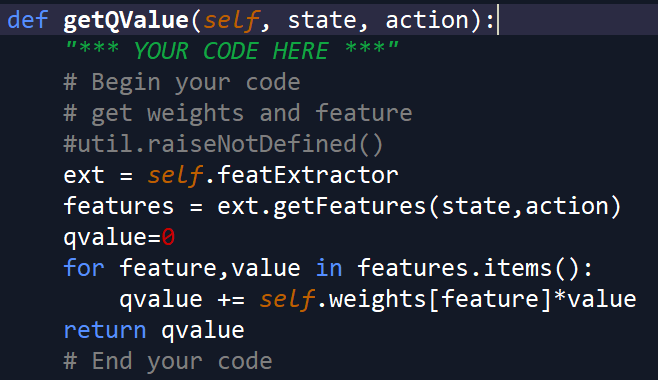
Part 2-3: epsilon-greedy action selection (5%)



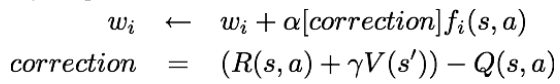
Part 2-4: Approximate Q-learning (10%)

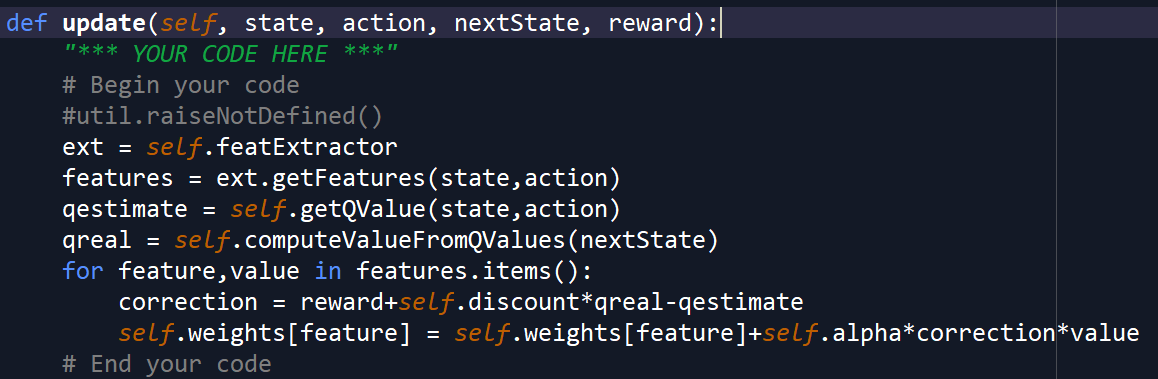
Q-value formula:





update formula:





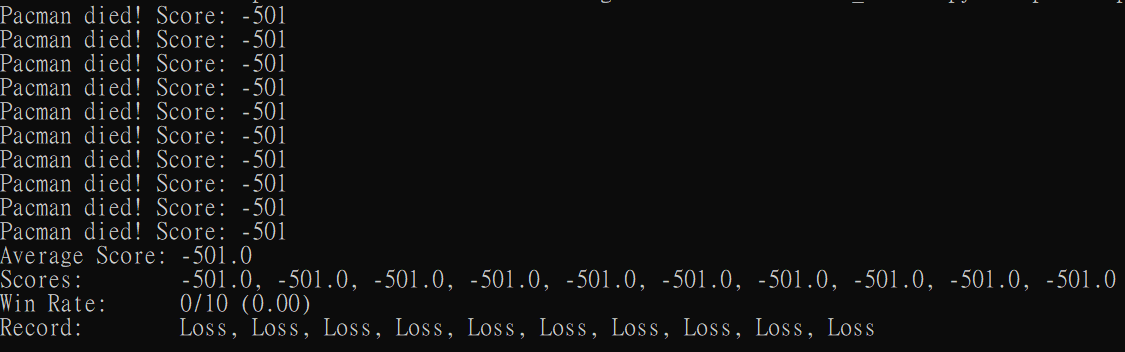
1. Observations

Part1: Adversarial Search

1-1: Minimax Search

python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3 -n 10

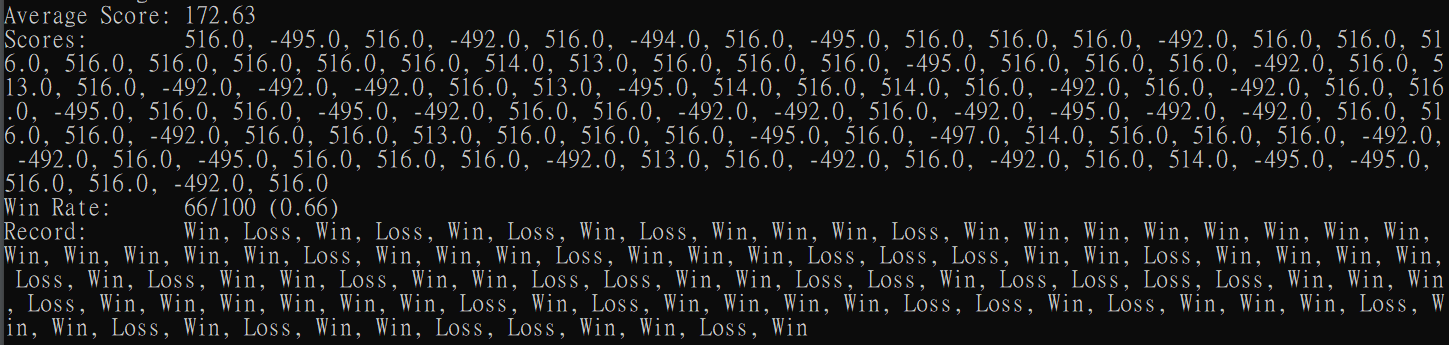
result:



Pacman think that it’s impossible to win this game (to escape), so it rushes to the closest ghost because the longer it stays alive, the lower score it will get.

python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4 –n 100

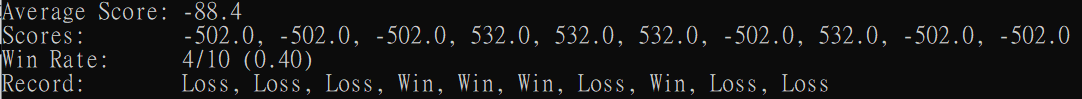
result:



1-2: Expectimax Search

python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -n 10

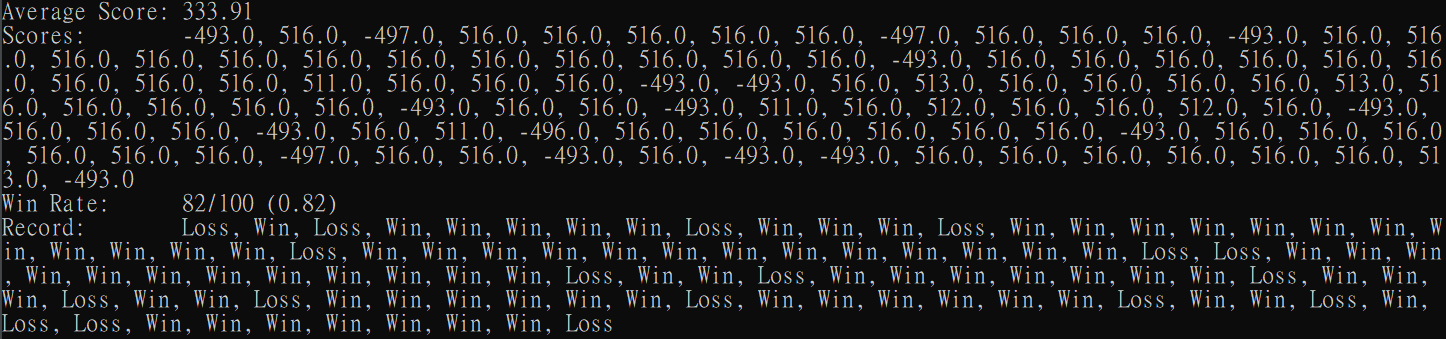
result:



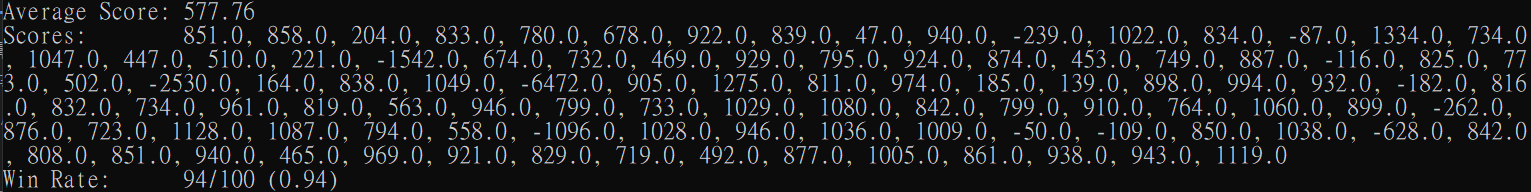
Here, unlike minimax, pacman won’t suicide in the beginning because it thinks there might be some chances to escape. Thus get a better result.

python pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=4 –n 100

result:



1-3 better evaluation function:



Conclusion for part1:

|  |  |  |
| --- | --- | --- |
| layout = minimax classic, n=100 | Avg score | Win rate |
| Minimax | 172.63 | 0.66 |
| Expectimax | 333.91 | 0.82 |
| Expectimax using betterEvalFn | 577.76 | 0.94 |

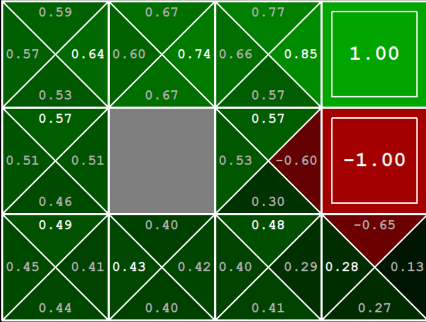
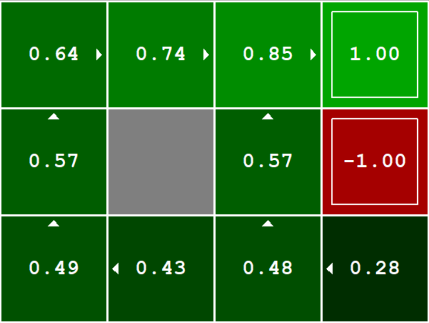
1. Minimax is a more pessimistic algorithm compared to Expectimax. It goes extreme sometimes. Because the ghost agents in this game isn’t as clever as minimax thought it would be, so minimax might not be a great algorithm in this game. On the other hand, expectimax assume the ghosts act randomly. As the result, Expectimax performs better in this case.
2. Better evaluation function helps expectimax to have a higher score and win rate.

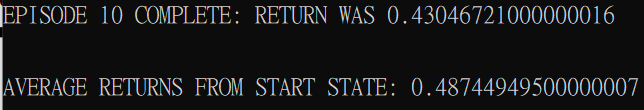
Part2:

2-1: Value Iteration

python gridworld.py -a value -i 100 -k 10

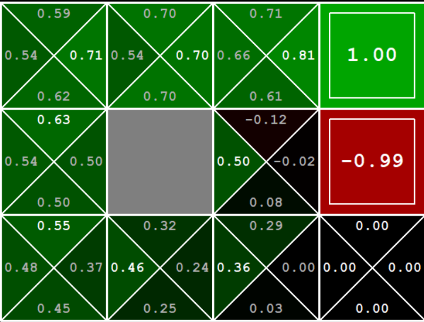
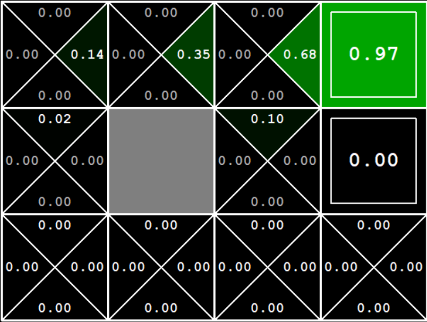
result: V(start) is close to avg reward!

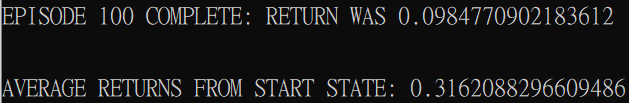




2-2: Q-learning、2-3: epsilon-greedy action selection

python gridworld.py -a q -k 5 –m / python gridworld.py -a q -k 10

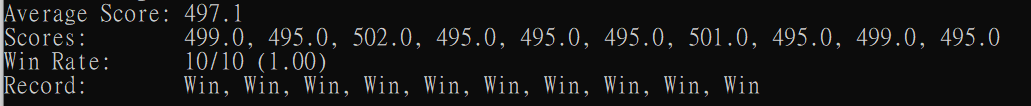




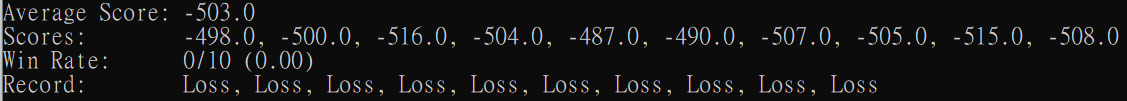
* average return is lower than the Q-values predicted due to random actions and initial learning phase.

test q-learning agent on pacman

python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid –q

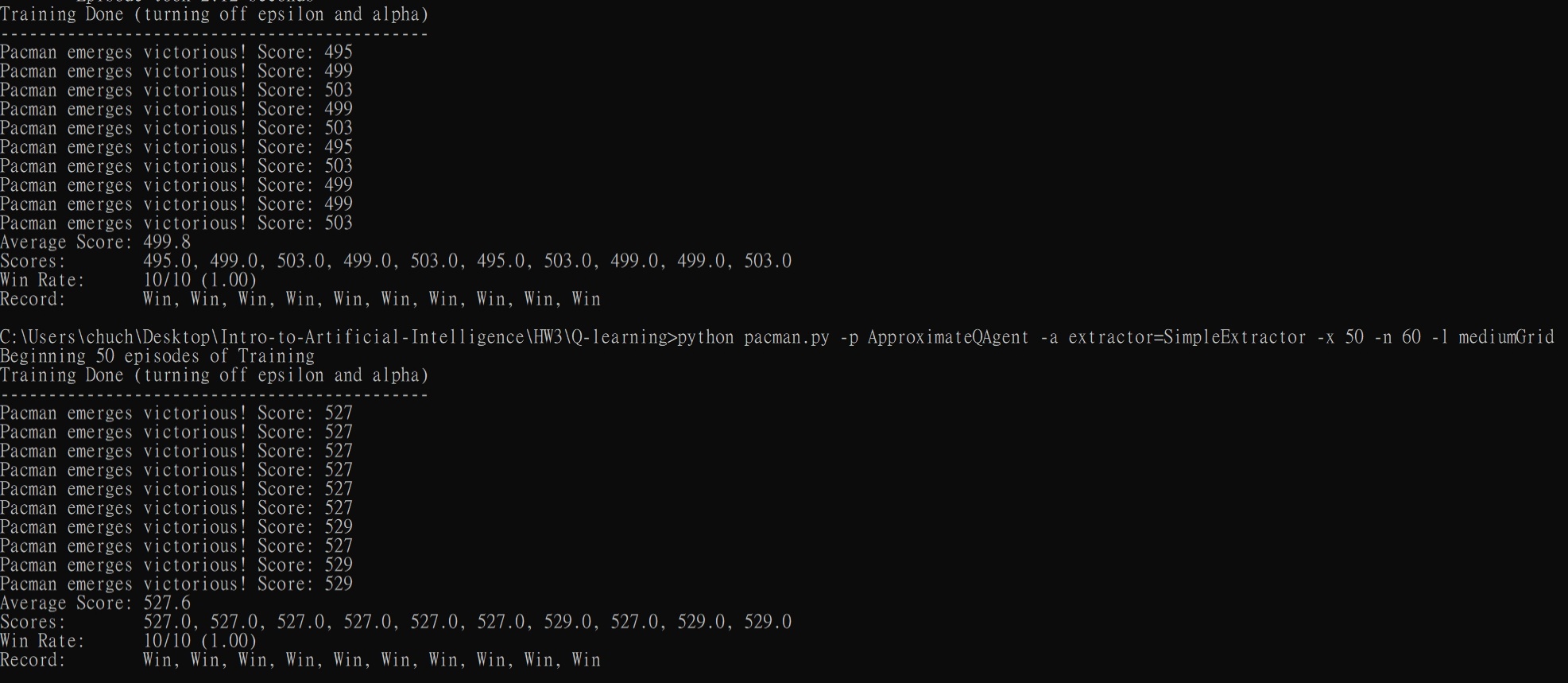


python pacman.py -p PacmanQAgent -x 50 -n 60 -l mediumGrid

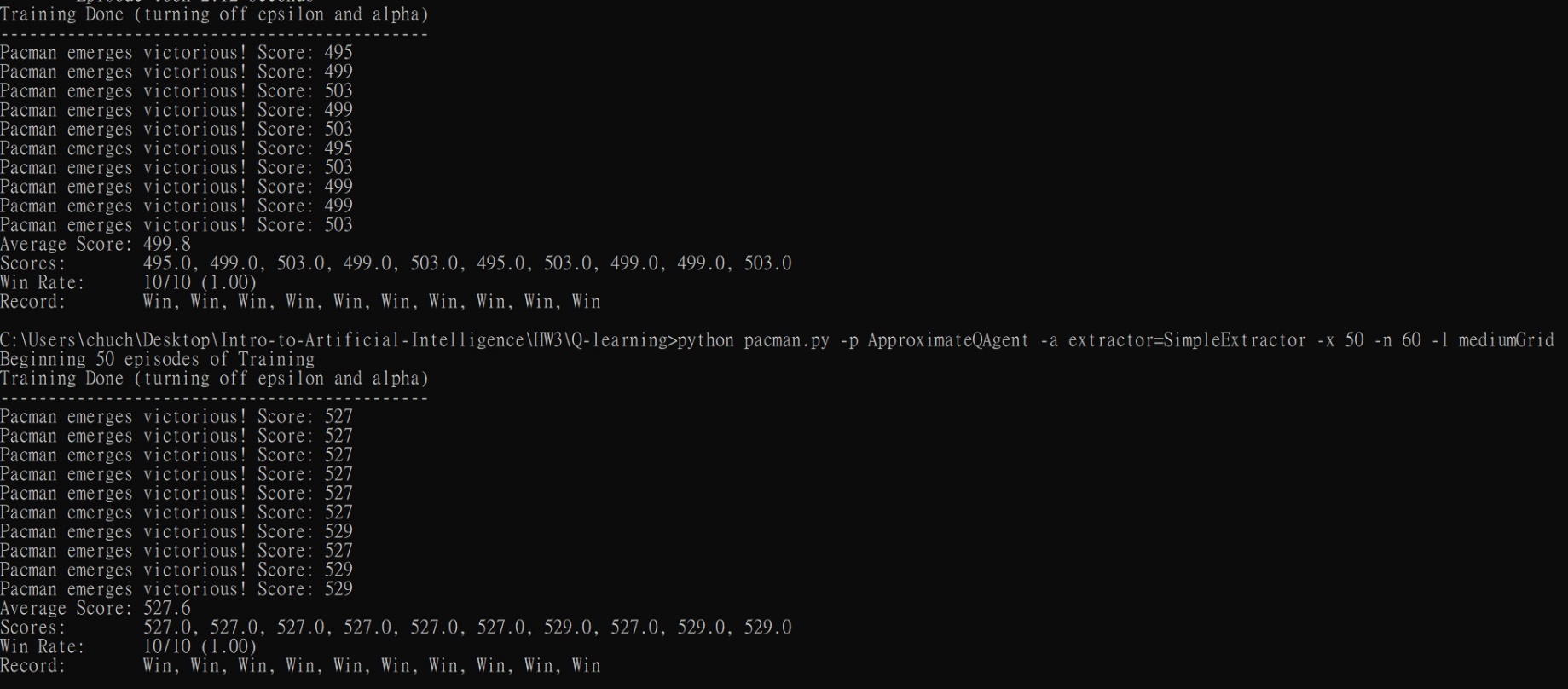


2-4 Aproximate Q-learning

python pacman.py -p ApproximateQAgent -x 2000 -n 2010 -l smallGrid



python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x 50 -n 60 -l mediumGrid



conclusion for part 2:

|  |  |  |
| --- | --- | --- |
| layout = smallGrid, train = 2000, test=10 | Avg score | Win rate |
| Q-learning | 497.1 | 1 |
| Approximate Q-learning | 499.8 | 1 |

|  |  |  |
| --- | --- | --- |
| layout = mediumGrid, train = 60, test=10 | Avg score | Win rate |
| Q-learning | -503.0 | 0 |
| Approximate Q-learning | 527.6 | 1 |

* In my opinion, Q-learning need a lot amount of training (ex: 2000) to have a good behavior, while Approximate Q-learning can learn very fast and smart given custom feature extractor.

1. comparison of different methods

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| #test episode = 100 | layout | Avg score | Win rate | #train ep |
| Minimax | MinimaxClassic | 172.63 | 0.66 | 0 |
| Expectimax | MinimaxClassic | 333.91 | 0.82 | 0 |
| Minimax | smallClassic | -122.88 | 0.1 | 0 |
| Expectimax | smallClassic | 115.23 | 0.19 | 0 |
| Q-learning | SmallClassic | -378.61 | 0 | 2000 |
| Approximate Q-learning | SmallClassic | 823.48 | 0.86 | 2000 |
| DQN | SmallClassic | 1260.23 | 0.79 | 10000 |

We can conclude several things from the result:

1. Adversarial search agent performs the worst of all three. They seems to have a good behavior in simple layout like minimaxClassic, but on smallClassic layout, the win rate is quite low.
2. Q-learning’s win rate is 0 in smallClassic. I guess this is because smallClassic is too complicated for Q-learning, so I test it on smallGrid:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| #test episode = 100 | layout | Avg score | Win rate | #train ep |
| Q-learning | SmallGrid | 499.8 | 1 | 2000 |
| Q-learning | SmallClassic | -378.61 | 0 | 2000 |

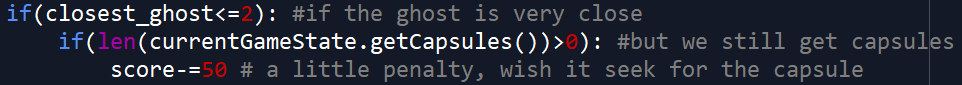
* We can observe from the result that my guess is correct! Q-learning only have a good behavior in simple layout like smallGrid, it is totally defeated in smallClassic.

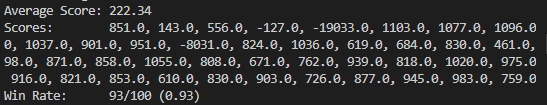
1. Normally, DQN should have a better performance than normal Q-learnings, but my result doesn’t fit (on aspect of win rate). I guess there are 2 possible reasons to explain this:
2. My computer resources. My friend has a win rate of 0.87 with the same parameters.
3. 10000 times of training is not enough for this neural network. A higher number of training episodes should gave a higher win rate.
4. Approximate Q-learning have a high win rate, I guess this is due to the well written of features.
5. problems encounter

However I write the better evaluation function, autograder for part1-3 return the same scores as normal evaluation function. Finally, I found out that I return scoreEvaluationFunction instead of self.evaluationFunction in expectimax.

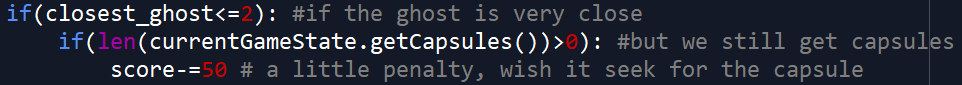
Then, it turns out I only get 5 points in autograder part1-3. Here’s why:

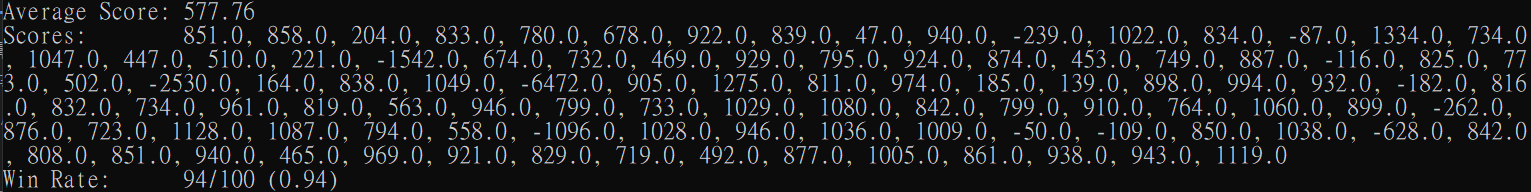
At first, I let the score-=500 in this condition. And I get the following result.





The average score is lower than expectimax using normal evaluation function. I guess this is because there are some extreme low scores in some cases (-19033 for example).

Then, I consider a scenario that pacman want to eat the capsule to kill the ghosts, I realize I’m the one who stand in its way!! So, I turn the value to -50, thus get a way better result in scores. 



And I get 7 points in autograder part1-3. (Not the best evaluation function but a better evaluation function ☺)