Ashkan Zarkhah
610399196

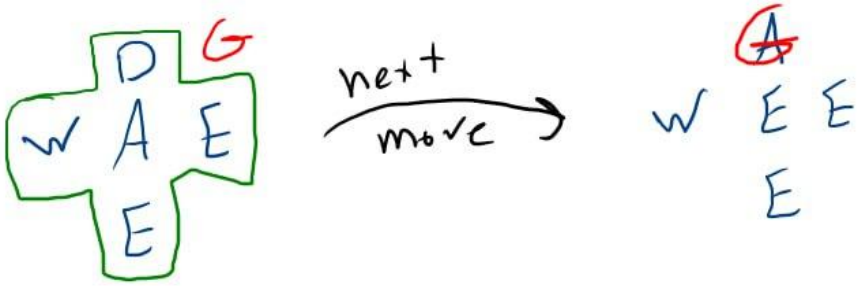
Question 2.

First, before answering questions, we describe our code and then we will answer questions about it and analyze them.

- **Code**

  First of all, we defined our plan, and because of our state compression(we would describe later) we needed to add two wall rows from up and down to our plan and also we needed to add two columns of walls from left and right.

  After that, it was time to build our states. We compressed our state to our agent's needed view and the agent needed to know every cell with at most 2 distance from it. One distance wasn't enough because the ghost could have come from its blind spot to its one distance and the agent could have gone there blindly too. We also added a state for when our agent loses.



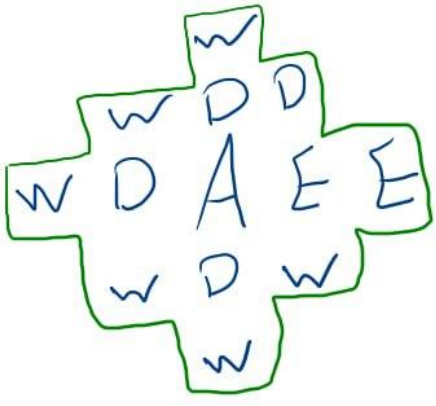
  Now for building states, we didn't just make states that were present in the plan because we wanted to run our learned model on another environment too, so we just made sure that A is in the middle and there is at most one ghost in the view.

  After building states, it was time to define two of our most important functions. The first one provides a random walk for our ghost and finds its next location and the second one finds our next state based on our current plan, our agent's and ghost's location, and our action type; first, we find our agent's next location and call to find our ghost's next location; then we check if our agent and our ghost will be in a same spot or will they pass each other, because in both cases our agent is going to loos and if it was the case, we would return our losing state with a reward of -10; but if we don't lose, we first check if agent eats any dots (we give 1 reward for every dot) and then we find our agent's 2-distance state and return it as our next state.

  Now before going to learn our Q function, we need one more function to make our learning progress faster. We need to hash our states so that we can access a unique ID for each of them to fill our Q table. We use a classic hashing method and we calculate every state's corresponding number in 256 base and module it by our number of states and put each one of them in its ID's cell in a list and if two states get equal ID, we increase second ones ID till that ID is not taken anymore.

Now that everything is prepared, it's time to learn our Q function. We define a function to do so, because later for checking different gamma and alpha values we need to have a generalized Q learner function. Our function gets all states, epsilon (our dynamic probability for not acting based on our Q table), alpha, gamma, episode number, and a flag which tells our function to print its progress or not, because for our first model's learning, we want to check if it converges or not, but for our gamma and alpha testing we don't need that much information. First, we set our Q table to be all zeros and then we start our first episode and in each episode, we first choose our agent's move (with a probability of epsilon we choose randomly, and with a probability of 1 - epsilon we choose it based on our Q table) and then we get our next state and our reward and we update our Q table's value with bellman equation, after that we check if we have not lost yet, we continue to our next move. We move 100 times in each of our episodes because even if we have eaten all the dots, we want our agent to learn how to react in states with a ghost and no dot (to just run away).
After learning it with variables of epsilon = 1, alpha = 0.5, gamma = 0.6, and number_of_episodes = 30000, we achieved a 41.2 average reward for our last 1000 episodes and it's a very assuring result. From now on our code blocks are specifically for one of the next 7 questions and we describe them there.

- **Explain what the number of states depends on. Can the number of states be reduced?**
  Number of the states depends on the location of our agent, the location of our ghost, and whether other non-wall cells are dots or they are empty. If we assume first we don't have an empty cell, we have 46(where the agent is to be) * 45(where the ghost is to be) * 2^{44}(others be dots or empty) + 1(our losing state). But here we put our states to be our agent's two distance cells and we reduce it to
  (12(where the ghost to be) * 3^{11}(other cells be dots, walls or empty)) + 3^{12}(just having dots, walls or empty cells) + 1(our losing state)) = 2657206



We could have compressed our states even more but as said before, it would have made a blind spot for the agent to see the ghost.

- **Define Action, State, Rewards, and Goal State for the problem and provide your explanation.**
  Our actions are going left, right, up, or down.
  Our states are 2-distance view of our agent. (like the picture above)
  We get a -10 reward if we touch the ghost and we get a +1 reward for eating each dot, we have not put any negative rewards for going to walls or running freely, because in the main game, these are not any negative things to do and I believe our agent needs to know how to run away from ghost too and just doesn't focus on eating dots.
  Our goal state in learning is to live for 100 steps but in checking, it's to eat all dots again because we wanted our agent to learn how to escape too.
  And we have a losing state and it is a terminal state so that if our agent loses it can't go any further.
- **First, consider the environment as shown above. Analyze the effect of γ for at least 3 gamma values equal to 0.25, 0.5, and 1. Also, for at least 3 α values, analyze the results and determine the impact of α.**
  Exactly in the first unexplained block, we do so. We set two lists of values for our gamma and alpha.

```
#here we check which combination of
alpha_list = [0.1, 0.5, 0.8]
gamma_list = [0.25, 0.5, 0.9, 1]
```

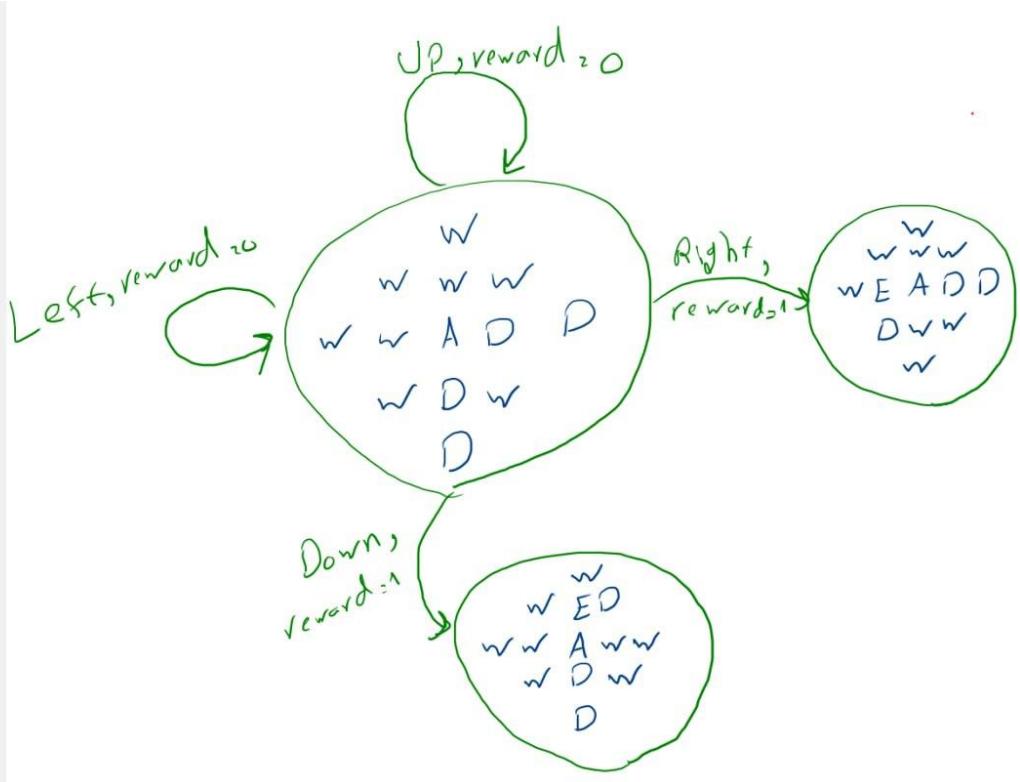
  Our gamma_list has just one value of 0.9(for being common in gamma values) but our alpha values are assigned by us and we have chosen these values because alpha is our learning rate and having both small and large learning rates can help us to find the best way to learn our Q function.

```
0.1 0.25 42.252 7
0.1 0.5 41.972 8
0.1 0.9 40.882 4
0.1 1 6.307 1
0.5 0.25 40.005 3
0.5 0.5 41.647 2
0.5 0.9 40.979 6
0.5 1 5.902 0
0.8 0.25 38.521 2
0.8 0.5 38.293 3
0.8 0.9 38.717 10
0.8 1 4.379 5
0.1 0.25 42.252
```

  The first number is alpha, the second number is gamma, the third number is our last 1000 episodes reward average and the last number is the number of losses in the last 1000 episodes. By watching the results, we see that by increasing our alpha our reward average decreases because we have enough episodes to learn completely, and with a big alpha we jump over the best answer. Also, we can see that the gamma value doesn't have that much of an effect, only when it is 1, and that is because with gamma = 1, we have that it doesn't matter how far, each dot

we eat has the same value, but with having a ghost and a limit on our actions in an episode, time matters and we can't assume that we have all the time in the word.
And at last, we can see that our best choice of alpha and gamma is:
Alpha = 0.1, Gamma = 0.25, Best average reward = 42.25
This is a very good reward because we have a cell in which the ghost can trap us in.

- **Display the environment as a weighted graph. The vertices of different modes and their weights are the amount of Reward after applying the corresponding Action. You can mention the corresponding Action as a Label on each edge.**
  We have over 2.5 million states and each one of them has 4 edges (4 different actions) and this makes it impossible for us to display the graph, but we build it and save it in the adjancy_list. For building it, finding each state's next state is dependent on our agent's location, so we try to learn our Q function one more time and this time by watching our transitions, we fill our adjancy_list with each state's next state and its reward. But for displaying a small part of our graph, we display our start state and its adjacency.



- **Draw the Q-Table for this problem.**
  Our Q-Table's shape is (2657206, 4), and again it's not possible to display it, so we saved it in a file named "Q_Table.txt" and you can see it there. Also for it to be easier to read we saved it in the format of states * actions, and because of that each row is only 4 numbers length and it can be read much easier.

With drawing our Q-table I found out that many of its cells are zero and have not been updated but this is all because we have computation limits and we are learning our Q function only on a single plan and because of this too many of states are not possible to be found and with learning our function on other plans too, we can fill our Q table more accurately.

- **Test your code on another arbitrary environment and report the result.**
  First, we draw a new plan.

```
['W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W'],
['W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W'],
['W', 'W', 'E', 'D', 'D', 'D', 'D', 'D', 'D', 'D', 'D', 'W', 'W'],
['W', 'W', 'D', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'D', 'W', 'W'],
['W', 'W', 'D', 'D', 'D', 'D', 'D', 'D', 'D', 'D', 'D', 'W', 'W'],
['W', 'W', 'D', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'D', 'W', 'W'],
['W', 'W', 'D', 'D', 'D', 'D', 'E', 'D', 'D', 'D', 'D', 'W', 'W'],
['W', 'W', 'D', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'D', 'W', 'W'],
['W', 'W', 'D', 'D', 'D', 'D', 'D', 'D', 'D', 'D', 'D', 'W', 'W'],
['W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W'],
['W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W']]
```

  After that, we ran our agent on it 10 times and for making its decisions we used a probability distribution again(because some states might be new to our function) with a probability of 0.3 to go randomly and with a probability of 0.7 to go base on the Q function and we didn't change our Q_function base on achieved reward, but we saved its sum and we saw that our function is doing good but it always gets stuck somewhere(because some states are new for it) and meanwhile, the ghost catches it and it lose. Our achieved rewards are:

```
28
29
6
15
5
28
27
12
13
9
Average =  17.2
```

- **In the original part of the Pacman assignment, you don't have to add a ghost in the environment, but if you do so, it is an extra point and optional. (You can use random walk for the ghost.)**
  As described before, we have made our states and our movements coordinated with our ghost too and it always starts from its cell and walks randomly in the plan. And also whenever our agent and our ghost go to the same cell or they pass through each other our agent loses and gets a negative reward of -10.