Marta Balairón García 100451724
Gracia Estrán Buyo 100452014

Bachelor in Data Science and Engineering

Departamento de Informática
Universidad Carlos III de Madrid

Machine Learning
Course 2021-2022

Assignment 2: Reinforcement learning

May 13, 2022

Marta Balairón García
100451724

Gracia Estrán Buyo
100452014

Marta Balairón García 100451724
Gracia Estrán Buyo 100452014

INTRODUCTION

In this last assignment, we are going to apply the Q-learning algorithm to the Pac-Man game we have worked with in all practices, to build an agent which plays automatically doing the best performance in every labyrinth. We will use the Q-learning algorithm to maximize the score obtained, using reinforcement learning techniques.

PHASE 1:

In this initial phase, we must decide which are going to be the attributes which form each state, and which is going to be the reward function.

On the one hand, we are going to select the attributes. Since we want that Pacman eats the ghost, we may select attributes that relate them. Firstly, we are interested in store the relative position of Pacman and the closest ghost. Therefore, as a first approach, we may describe the position as north (N), south (S), east (E), west (W), north-east (NE) and so on with the 8 possible combinations. What is more, we must have in mind a state which describe the situation when Pacman eats the ghost. Hence, the relative position of Pacman and the ghost are the same.

Moreover, another important attribute that relates Pacman and the ghosts is the distance between them. This distance is calculated as a Manhattan distance which makes it discrete. What is more, the maximum value for the distance is 17. Consequently, we will have 137 states of the form (pos, dist), being pos all the possibilities of relative positions and dist all the possible distances discarding 0 since it is the same state.

This first approach produced us some problems since computing all those states in the q-table was too time consuming. For solving this, we discretized the distance dividing it in being close or far away. This reduced our states to 17, being much more efficient. Moreover, as we needed to consider Pac-man's distance from the ghosts considering the walls in between, we took advantage of the method Distancer, which calculates the Manhattan distance having in mind walls.

On the other hand, we have designed a reward function which works by giving a penalization or a reward regarding if Pac-man gets closer or further from the ghosts and considering if the score gets better. This new reward function, also considers the possibility of getting stick in a corner, therefore, it also prices getting out of that loop.

This approach performs relatively good, but when appearing food, the agent ignored it. Therefore, we had to make some changes for obtaining an agent that also has as an objective eating the Pacdots.

PHASE 2:

The first part of this phase is to code the methods update(), getReward(), and computePosition().

- computePosition() :
  For obtaining the position of the row associated to each of the states, first, we created a method named closestGhost, which calculated which was the ghost with the minimum Manhattan distance to Pac-man, considering the walls. What is more, we implement the method getClosestFood(), which returned the position and the distance to the closest Pacdot. As well, we implemented the the method getNumFood() which returns a number representing the number of pacdots in the game.
  To continue with, we select the objective that Pacman wants to eat. For obtaining that, we find out if the ghost or if the food is closest. The one which is closest to Pacman is the objective now.
  Then we compared each of the positions of the closest objective and the Pacman obtaining this way the relative position between them. Parallel to that, we put the terminal state in the row 0, being that state the correspondent to being distance 0 and SAME position. Finally, we calculate the row of the rest of the states by considering at each case if the distance is considered close or far.
- getReward():
  For obtaining the reward we obtained the score in the current state and in the next state. Then, computing the difference between them we obtain the score obtained in the tick. Then, we must add a penalization or an extra reward depending on which was de objective. On the one hand, if the objective was a ghost, we must add an extra penalization if it gets farther away and an extra reward if it gets closer. On the other hand, if the objective is a Pacdot, we add a biggest extra penalization or reward, following the same criterium that for the ghost in order of

priorate eating the dost before the game ends. What is more, by observing the legal actions at each tick, we give a reward for the cases in which there was a corner blocking Pacman and it was able to sort it and getting out of there.

- update():
  To update the q-table, firstly we computed the associated row to the state using the computePosition() function and the column number associated to the action done. Then, we performed the appropriate formula depending on if the state is terminal (row = 0) or if not.

After this first approach, our q-table has 17 rows and 4 columns.

For the selection of the best value for the parameters, we started with epsilon = 1 at the beginning of the agent's learning process, as the agent takes random actions for probability epsilon and greedy (qtable) actions for probability (1-epsilon). Then we followed a process of incremental development, and as the agent went deeper in its learning process, we started decreasing the epsilon value, so that it would gradually pass from choosing random actions to choosing the learnt action (from the qtable) as it learned more. Alpha is the learning rate. Setting the alpha value to 0 means that the Q-values are never updated, thereby nothing is learned. If we set the alpha to a high value such as 0.9, it means that the learning can occur quickly. Therefore, we set alpha to 0.8. Lastly, Gamma is the discount factor. This models the fact that future rewards are worth less than immediate rewards. A gamma of 0 will cause the agent to only value immediate rewards, which only works with very detailed reward functions. Therefore, we set the discount factor to 0.5.

PHASE 3:

In this part of the report, we are going to explain how we have evaluated the agent's performance at each map. In the first approach, with 137 states, we found it to be too time-consuming checking so many states. And at this point was when we decided it was necessary to reduce the number of states, as we described in phase 1. Before applying the method, we created which considered the Manhattan distance and the walls, we obtained good results for the first two labyrinths, but when the agent had to deal with walls, it did not learn as well. Therefore, we considered the Manhattan distance and had to start the learning process again with the new code. Also, to improve the Qtable, we changed the getReward method so that it did not only return the difference in the score (this is, if the score improved), but also it considers if the distance to the closest ghost got better, worse or the same, as explained in phase 2. We did the learning process starting from the simplest labyrinth to the most complex.

- MAP 1:
  In this map we just have 1 ghost and no walls, so the learning process for it was not very complicated. As we explained before and like we did in all labyrinths, we started with epsilon = 1, so that every action taken was random and performed 25 games so that the agent would learn from the actions taken. Then we did a second round of 25 games with epsilon = 0.8, this way it had a probability of 1/5 of choosing an action based on the qtable against a probability of 4/5 of choosing a random action. After this, we did a third round of 25 games with epsilon = 0.7. We did the same for epsilon = 0.5, 0.4, 0.3, 0.15 and 0.05. Lastly, our purpose was to have the agent performing perfectly with epsilon = 0. We obtain an average score of 183 and a win rate of the 100%.

- MAP 2:
  In this map we followed the same learning process of decrementing epsilon gradually and it was very similar to the first one as it did not have walls, and with the only difference of having 2 ghosts instead of one. We obtain an average score of 383 and a win rate of the 100%.

- MAP 3:
  In this map we started encountering more difficulties due to the walls. Also, the number of ghosts in the 3 labyrinths was 3. It was at this part of the assignment where we realized that we must make some changes for considering the walls somehow. This learning process took a little bit more of time because of the complexity but finally we achieve an average score of 569 and a win rate of the 100%.

- MAP 4:
  In this map we found out new difficulties. We have at the same time pacdots and walls. We needed again to make changes in the code for obtaining an agent which consider the food and priorate eating them before the game ends. Moreover, our Pacman got stacked several times, therefore we had to code some instructions at the reward function to learn it how to get out of there. This learning process was much more time consuming than the ones before. We

needed to perform 35 games for each epsilon and what is more, we needed to do a slowly decremental process of the epsilon.

- MAP 5:
  In this map we found out similar difficulties as in map 4. This learning process was worse since there are more corners difficult to sort by pacman.

Summing up, we have created the following table of the mean score after 10 games:

MAP	MAP 1	MAP 2	MAP 3	MAP 4	MAP 5
SCORE	183	383	569	721 (e = 0.1)	947 (e = 0.1)

CONCLUSIONS

In this assignment, we have learnt and proven the usefulness of reinforcement learning techniques and an agent's performance using the Q-learning algorithm. We have seen it is a very appropriate model for videogames such as Pac-man where there is an agent whose performance you want to be the best and automatically done. We have encountered some difficulties when approaching the tasks asked, but we considered we have accomplished to get through all of them and we have made a good use of reinforcement learning techniques to accomplish all tasks as it has been explained during the report.