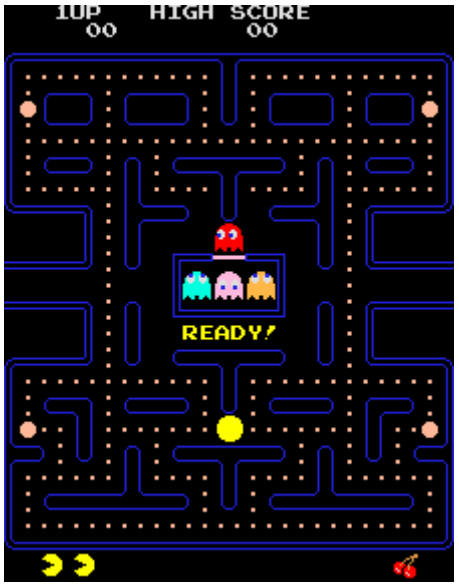
Assignment 2
Machine Learning



Paula Serna del Amo 100429945
Sofía Pérez Pérez 100429985

1. Introduction

One of the most popular games is without any doubt pacman. It is a maze action game developed and released by Namco for arcades in 1980. Our goal in this assignment is to transform this classic game by using reinforcement learning in such a way that it learns the optimal path to learn the game while maximizing the reward.
We will use the techniques seen in class, in particular the Q Learning algorithm, which updates the Q table until obtaining the optimal policy..

2. Feature selection. Different ideas.

In order to end up with an agent that is able to learn with reinforcement learning, the feature selection process plays an important role in terms of the success of the results. Therefore, we implement three different models, or state functions, and then we analyse how pacman is able to learn with those states, and the possible improvements.

Remark: note that for all the distances, we have computed the **relative** distance from pacman to the ghosts/ pac dots, with the **Euclidean** one.

Idea 1:

- **Feature description**

For this first idea, two of the attributes we have considered are the distances from pacman to the closest ghost and pac dot. Since the maximum distance in every labyrinth is 17,it would result in a huge number of states. Therefore, we discretize these distances. In this case, we create 3 ranges:

Ghosts:

Coincident: distance between 0 and 1.
Close: distance between 2 and 8.
Far: distance larger than 9.

Pac dots:

Coincident: distance between 0 and 1.
Close: distance between 2 and 8.
Far: distance larger than 9.

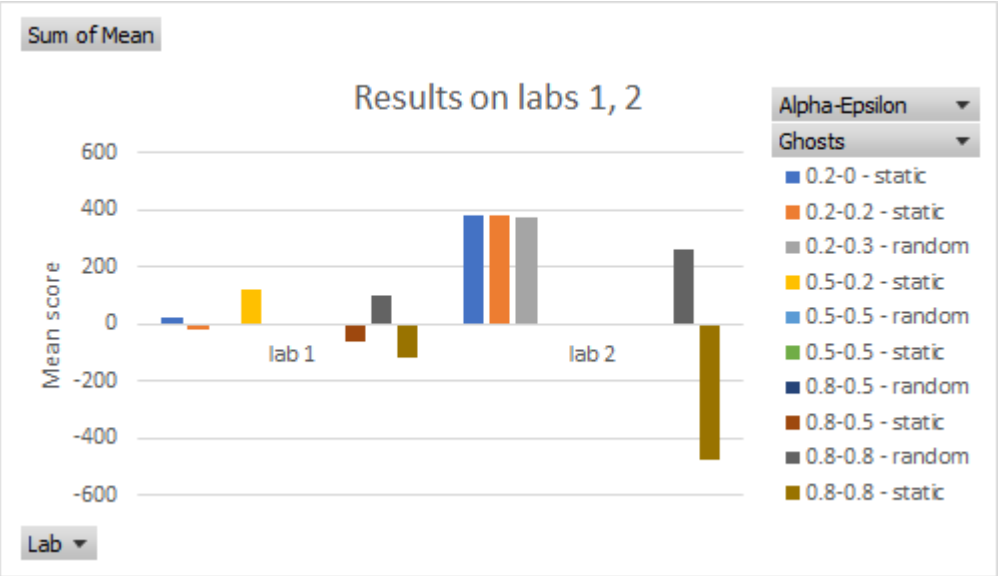
For pac dots, we have realized that in some labyrinths, there may not appear. This is why we have included the possibility that there are no pac dots in the *far* option, so that it will consider value 2 if it is far or non-existent, so that pacman does not focus on it. Other attributes we have included here are the *horizontal* and *vertical* ones. They include 3 values each, which are 0,1 or 2. The former consists of comparing the absolute value of the difference in positions in the x-axis with the nearest ghost, to see if it is closer to it in the west, east or none (this situation can occur if it is closer in the y-axis than in the x-axis, it is more worthy to go in the other two possible directions, north or south, than going to the east or west). Therefore, in this case, east: 0, west: 1

and none: 2. Regarding the latter, *vertical ,* we have the same idea. However, instead of the x-axis,it consists of comparing differences of positions with the closest ghost in the y-axis so that we can know if it is closer in the north (0), south (1),or none, 2 (pacman is at the same level than the closest ghost).
Finally, we obtain a tuple of the form: **(dist_ghost, dist_dot, horiz,vert).**
Each of the attributes included in this idea contain 3 different values, so the total number of states that our Q-table has is **81**. It is a large number for states; this is the reason why we have just taken into account 3 ranges for the distances, knowing in advance that it would have been better to include more.

- **Reward function**

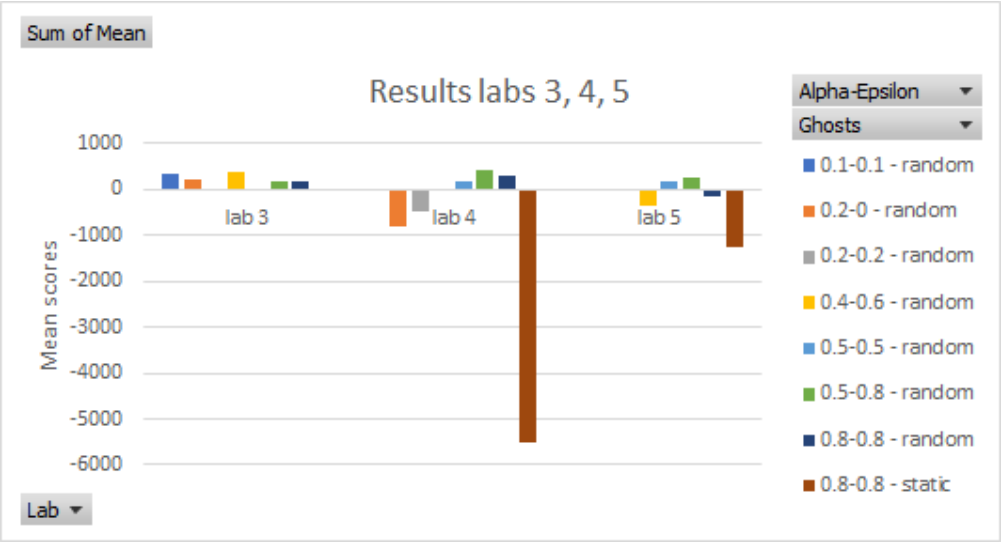
For the reward function we give pacman a reward of 1 if at the current state it is at the same position with the ghost or with the pac dot, so that he knows he is doing well. And we have also given a reward of 0.5 when in the nextState, its position is also coincident (as before), so this way, he will know that he is on the good way.

- **Results**



Since our objective is that at the end of the training process the agent has learnt which is the best path to reach the ghosts and obtain the maximum reward, we have changed the values of epsilon and alpha, until they are both 0, which means the agent has learnt.
We have started initializing values of alpha and epsilon to 0.8, which implies that the agent moves almost completely random (the higher epsilon is, the more random it moves); and setting the ghosts as static. Progressively, we have reduced the parameters, and have changed the type of ghosts to see the difference in the behaviour of the agent.
We must note that the results for static ghosts in both labyrinths is worse than with random ghosts and It is important to highlight that in this idea we have not included a recommended action, so it has not behaved as well as the other two ideas, but at the end of the procedure, we have observed an improvement when reducing the

parameters,especially in labyrinth 2 with values 0.2 for alpha and 0 for epsilon, which is what we were looking for.



Regarding labyrinth 3,4 and 5, we have different results than before. We have tried to run it with static ghosts, but it has not worked; each round lasted a very long time obtaining negative scores; that's why we then executed it establishing random ghosts. The reason why our idea has not been successful with static ghosts is that it does not take into account the existence of walls, so it gets stuck in the same place when eating a ghost.

Therefore, we have trained it with random ghosts, and again, we have been reducing the parameters. We can see that the results have not been great, even using these. In conclusion, we can say that this idea has not been a good election, so after this, we have created two more ideas including the attributes that we think this idea needs.

- **Problems in the implementation/ Possible improvements**

This idea is the one with more drawbacks. We should take into account that it was the first one, we have improved it in the rest of ideas with the introduction of new attributes. The one we think is the most significant is that it does not take into account the walls. This fact may not be very significant for the first two labyrinths, but it does for the rest. When pacman finds a wall it begins to act in an uncontrolled way since he does not know what he is facing.

Another drawback is that we created only 3 regions for each of the attributes including distances (coincident, near and far), which we consider is a very general discretization.

Idea 2:

- **Feature description**

In this case, we also take into consideration the distance from pacman to both the closest pac dot and ghost. In order not to end up with a lot of states that would make the learning process much harder or even impossible, we discretize the values by creating 4 (including one more range than in Idea 1) and 3 ranges for the distance to the ghost and pac dot, respectively.

The maximum distance from pacman to any agent is 17, if they are in opposite corners. Therefore, the discretization is:

Ghosts:

Coincident: distance between 0 and 1.
Very close: distance between 2 and 5.
Close: distance between 5 and 11.
Far: distance larger than 11.

Pac dots:

Coincident: distance between 0 and 1.
Close: distance between 2 and 8.
Far: distance larger than 8.

Why do we decide to include one more range to the ghost's distance?

Since by eating a ghost pacman gains more points than with the pac dots, we believe that we should emphasize the attributes regarding the latter.

We then decide to include an attribute that recommends pacman the best action to do at any tick, taking into consideration distances to the closest ghost, which was created in tutorial 1. However, this method is mainly focused on checking the best direction to reach a ghost, and if it is not possible to execute it, do a random movement. Again, we have decided to take into account the latter because we want pacman to focus on where the ghosts are, since we are including information about the food in attribute 2.

Finally, we obtain a state tuple of the form: **state (dist_ghost, dist_food, best_action).**
The number of combinations is **48** (4 * 3 * 4), and as we can see, in this idea we will study the case when having less number of states (48), and we analyse later the results.

- ## Reward function

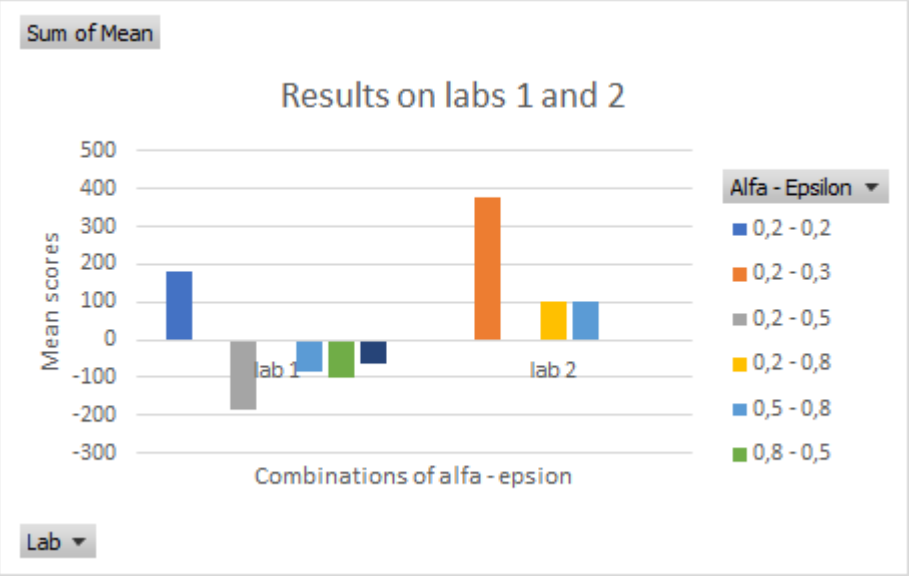
For this implementation pacman will obtain 0.75 of reward if the distance to the closest ghost/ pac dot in the state or in the next state is "coincident. This way, we make sure that once pacman reaches the agent, it stays close to it. The second reward will help to once reached the area, being able to eat it. That is, if pacman and the agent are coincident, (distance <= 1), and it executes the recommended action, the reward is 1.

- ## Results

In this second idea we also follow the procedure of trial and error process. We start by training our agent in labs 1 and 2, decreasing the values of epsilon and alpha

progressively. The main goal is to obtain an agent that has enough knowledge to be able to play in every lab, and win. If this is achieved using low values of alpha, which determines the importance of the previous knowledge versus the current reward, and epsilon that corresponds to the number of random actions without paying attention to the Q table.

Regarding the results on labyrinths 1 and 2, the agent, after some trials, shows that is able to learn in a layout free of walls. As it can be seen in the plot, the mean score increases when the parameters are decreased. This is a good sign of learning. In fact, when epsilon = alpha = 0.2, the scores are maximized. These values were used after having tried the rest, so we conclude that in these first two labs the agent is capable of learning.



Nonetheless, when pac man plays in labs 3, 4 and 5, the results are much worse. Furthermore, we have noticed, as expected, that when playing with random ghosts the results are considerably better than if they were static. For instance, in lab three, right at the beginning of the training, with parameters epsilon = alpha = 0.8, in which pacman is mostly going to explore and do random movements, if the ghosts are random, the mean score is 213 and if they are static -1407. The bad behaviour in these labs is obviously due to the existence of walls. In addition, the state tuples do not take into consideration any information about walls, since when the recommended action is not possible, it is programmed to avoid the wall by doing random movements.

- **Problems in the implementation/ Possible improvements**

The main issue we have observed in this idea is that the walls are not well considered. Pacman behaves well in labyrinths 1 and 2, but once we introduce the walls, it gets stuck in walls where on the other side there is an agent to eat, and this is exaggerated when the ghosts are not dynamic.

Idea 3:

- **Feature Description**

This last last idea was thought after having seen the results in proposals 1 and 2, and with the expectations of being the best one. Therefore, we include all the attributes that we reckon are a good mix, and that have been a suitable option in past examples.

Firstly, we consider that including both distance to the closest ghost and pac dots are the most important and indispensable attributes, and that without them the learning process is not possible. Therefore, differently than in ideas 1 and 2, in this case we have not selected two attributes for the distances, but only one. In this feature we include only the distance to the closest agent, regardless of its nature (ghost or pac dot). Then, as in the other ideas, we have discretized by creating 4 ranges.

However, some information regarding the type of agent should be included. Hence, we create a binary attribute which indicates whether the closest agent is a ghost or a pac dot. In this way, we include information about the distances, but in less attribute-value combinations.

1 - When there is a ghost close: Pacman is focused on eating ghosts**. If distance of pacman to closest ghost <= dist pacman to closest dot.** (We include '=' in this case since it gains more points by eating a ghost than a dot)

0 - When there is a dot close: Pacman is focused on eating dots. **If distance of pacman to closest ghost > distance of pacman to closest dot.**

In addition, we also consider the recommended action as in idea 2, but adding one improvement which is another binary attribute indicating the presence of a wall. This feature checks the recommended action of tutorial 1, and if by doing that action pacman encounters a wall, the value of the attribute is 0 and 1 otherwise.

Finally, the state tuple that we obtain is: **state(distance, binary(ghost / pac dot), action, wall).**

We end up with **64** possible states. (4 * 2 * 4 * 2)

- **Reward function**

In this case, pacman obtains a reward of 1 when there is no wall, it executes the recommended action, and its distance to the closest agent is in the range "Coincident", that is, smaller or equal than 1. In this way, if it executes the recommended action which indicates that the optimal movement is in the direction of the agent to eat, and since there is no wall, we make sure that pacman eats it. Same procedure as idea 2 but including the wall.

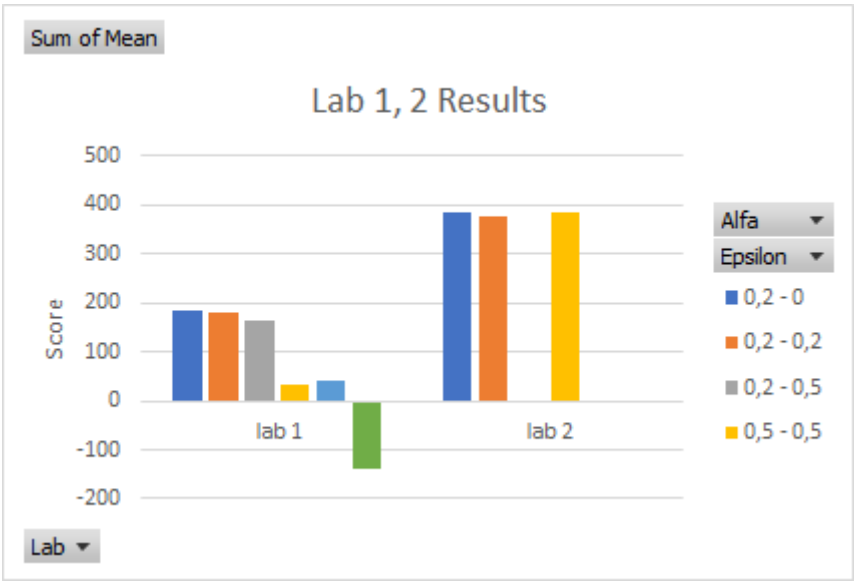
In addition, we include situations where the current state or the next state indicates that pacman is "very close" to the agent and executes the recommended action, obtaining

0.75 points. By doing this we make sure that pacman gets a reward by being in the same area as the agent.

- **Results**

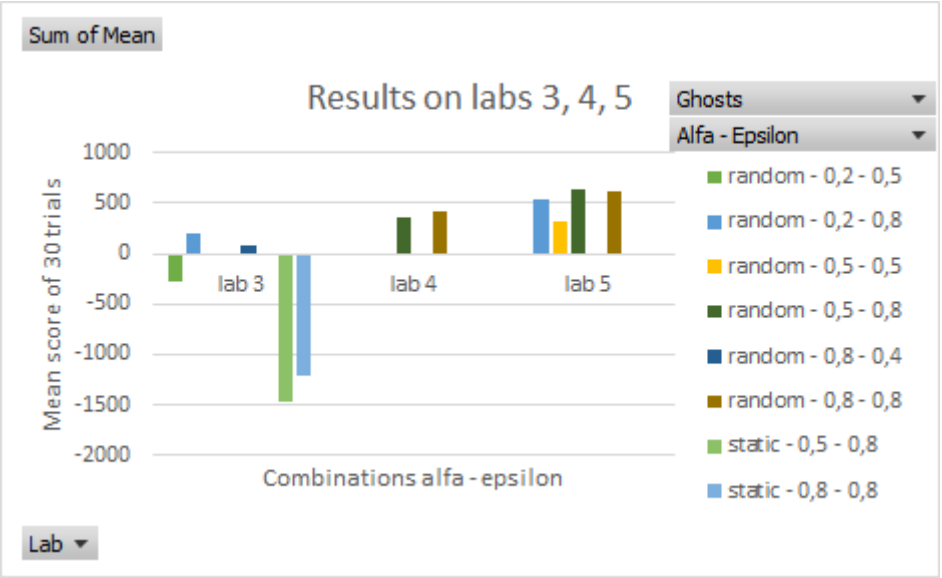
For this idea, we have followed the same procedure as previous proposals. Firstly, we train our agent with large epsilon and alpha values, and we lower them progressively. Furthermore, we do not appreciate signs of overfitting, because when starting one new lab, despite having trained the agent in the previous one, it needed some trials to relocate and start working properly.

The figure shows the mean scores when using a combination of alpha and epsilon. As we can see in the plot, the results obtained after training the agent in labs 1 and 2 show that pac man is able to learn in these layouts almost perfectly. The learning process is based on trial and error, by decreasing the value of epsilon and alpha. Finally, as the blue columns show, when epsilon is 0, that is, no random actions are executed, and alpha is 0.2, which implies that the agent is using mostly its knowledge and not the rewards, high scores are obtained.
Also we note that results in lab 2 are better than in the previous labyrinth with values that incur high randomness since it has already done lab 1 and learnt it, so for pac man it is easier to use its memory to complete lab 2. For instance, for combinations alpha = epsilon = 0.5, in the first lab it obtained a much lower mean score than in lab 2.



Regarding labs 3, 4, and 5, we included random ghosts which provide better results than when using static ones. For instance, in lab 3, if we run the game with and without static ghosts, the mean score is -1479 and -281, respectively. We are aware that the latter is not a good result either, however, it can be seen the improvement based on adding some randomness to the ghosts.

Furthermore, we notice that the main issue in these labs are the walls. In spite of having included some attributes that make reference to the walls, we realise that they might not be explicit enough for the learning process. However, in past ideas the results in these labs, even with random ghosts, were not as good as in this one, so we consider some improvements have been made.
After having trained our agent in labs 1 and 2, we follow the same procedure in next labs. We have collected the mean for each 30 attempts, in which we use the same values of alpha and epsilon.
We can observe in the figure that the results when using the static ghosts are much lower than the ones with dynamic. In addition, we noticed that using high values of epsilon provide better scores, which show that there is still a lot to improve since it is not able to do a good performance with values lower than 0.5. Regarding alfa, when using low values, that is, using the learning acquired from previously labyrinths, we notice that it is not sufficient since in those there were not labs, so pac man did not learn from those situations. Therefore, we had to start again our learning process with high values of alpha. As we can see, with high combinations of them, considerable large scores are obtained. Nonetheless, if we tried to run the game with epsilon lower than 0.4, the agent got stuck in the walls.



One important remark regarding the mean scores, is that the variance between samples is huge. For example, let's consider the following example that we obtain in lab 4 when using alpha = 0.2 and epsilon = 0.5. The scores we obtained are: -56, 197, -932, 146, 425, -489, -246, 300, -717, 116, -3997, 417, 279, -58, 393. As we can observe, there is a larger value that makes the mean to decrease a lot. This result was that bad because the pac man got stuck in a wall. Therefore, we should always be conscient that the mean is not sufficient, as we could include other measures to compare trials, such as the variance or the standard deviation.

In addition, regarding the Q table obtained after the training process, we observe that the elements of the table that should have large probabilities correspond to the states in which pacman is close to eating a ghost. We illustrate this by an example:

State = (0, 0, action, 1), which means that it is coincident to a ghost, it is focused on eating ghosts, a certain recommended action, and there is no wall.
If action == “north” (0), it corresponds to row 4, and in the text file line 5, the probabilities are around 0.75, which correspond to the higher values of the table. The same happens with the rest of the action. Therefore, this means that the agent is learning for the right states.

- **Problems in the implementation/ Possible improvements**

  Since we have chosen this model as the definitive one, we expose the possible improvements /problems of it in chapter 3.

3. Final model

To conclude, we briefly expose our final model, and the advantages and drawbacks of it. Firstly, the chosen model is the one of the idea 3, since we reckon is the one that has provided better results at low values of alpha and epsilon, that is, the model in which pacman is able to learn, and to put it into practice.
Furthermore, after having tried the learning process with different sizes of the Q matrix (81 ,64 and 48 rows), we reckon that in all the ideas the size has not been a problem, although maybe if we had reduced the one of 81 rows, we could have obtained better results.
Regarding the idea 3, and taking into consideration that we had pinned our hopes on this last idea, we expose the main drawbacks and advantages of it. Firstly, concerning the attributes, after the creation of implicit attributes that reveal the presence of a wall, we reckon that we have not been completely able to satisfy our goal that pac man learns in all the labyrinths. It does learn in labyrinths 1 and 2, but in the rest of labs, we had expected a better behaviour, specially when playing with static ghosts. However, it is not that we have not taken the walls into account at all, because there is an attribute dedicated to them. Maybe what fails is the type of approach that we have implemented to recognize walls and avoid them. Therefore, we think that including some other attributes that may be useful for improving our model, could be an interesting approach. For instance, making an improvement on the function coded in tutorial 1 such that it avoids the walls better.
Nonetheless, the results in labs with this idea are better than the ones obtained in the rest, since if we introduce random ghosts, it is more or less capable of eating all of the ghosts.

Another improvement that we could take into consideration for future experiments is that instead of considering the mean as the only metric, since the maximum points that you can obtain per lab increases with the difficulty of them, we can take into consideration the difference between the scores obtained and the one with the agent playing good.

4. Code modifications

In this section we briefly expose the main changes we have done to the original code in order to have a clearer understanding of how it works.

First of all, in order to obtain the state at any given tick, we have created the method “GetState”, which returns a list containing each of the attributes, which vary depending on the idea. This function is implemented in BustersAgents, and it is called in the main loop of the game to build the tuple needed for the update function. Regarding the latter, it also needs the information of the next state. This is computed the same way as we did in Assignment 1, storing the value of the current state, and using it in the next tick.

Furthermore, regarding the code of the QLearning agent, most of it was given to us. However, the Compute Position, Reward Function (explained above in each idea), and the Update method are implemented.
For the *ComputePosition*, we have derived a formula to compute the row of each state depending on the idea. We have obtained it by first writing a great quantity of combinations of the values for the attributes in an ordered way, so that when we have lots of combinations written, we see the “frequency” of the values on an attribute, better said, how many combinations take place until the next value of an attribute appears. For example, let’s see it with the final idea, number 3:

We have 4 attributes, the first one with 4 different values (due to discretization, so 0-1-2-3), the second one is binay (0-1), the third one has 4 values, in this case, directions (0-1-2-3) and the last one is binary again,it determines if there is wall or not. So now, we create the first blocks of values in a way that each column is an attribute:

0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 0 2 0	0 0 2 1	0 0 3 0	0 0 3 1
1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 0 2 0	1 0 2 1	1 0 3 0	1 0 3 1
2 0 0 0	2 0 0 1	2 0 1 0	**2 0 1 1**	2 0 2 0	2 0 2 1	2 0 3 0	2 0 3 1
3 0 0 0	3 0 0 1	3 0 1 0	3 0 1 1	3 0 2 0	3 0 2 1	3 0 3 0	3 0 3 1

and so on...

What we do is change the first element of the row, and start changing values from the end. We start iterating with the last attribute, which has 2 possible values, and we can see that its values change after 4 combinations. Then when we have finished putting all the combinations for those, we go to the previous one, which is the attribute of directions or movements, and do the same for its 4 values… In this particular example we have seen that the last attribute changes each 4, the third attribute each 8 and the second each 32, getting to the formula: ***row= first + second*32 + third*8 + last*4.***
For the shown combinations, let’s proof that the formula works well:
Let’s take the combination of 2 0 1 1 and compute the row in which it is located: row=2 + 0*32 + 1*8 + 1*4 =14. We can check above that it is the 14th row, taking into account that the first one is 0.

For the first idea, following the same procedure we obtained:
*row=first + second*27 + third*9 +last*3*
For the second:
*row=first + second*16 + last*4*

In conclusion, we did this for all the ideas we have carried out. It may be a little bit tedious but for us it has been the best option for filling the table in the sense of avoiding a lot of code.
Regarding the *Update function*, we take into consideration whether the current state is final or not, and if not, the normal Q function is used. On the contrary, if it is a terminal state, we use the same one but with the discount as 0.

Formula of the *update Q* function is:

(1- alpha) * Q(s,a) + alpha * (r(state, action, nextState) + discount *max(Q' (s', a)), with parameters alpha, discount and epsilon varying from 0 to 1.

5. Conclusions

In this second assignment we try to train pacman by means of Reinforcement Learning so that it is able to find the path to eat all of the ghosts in such a way that the reward is maximized. We use the Q Learning algorithm, by modifying in each tick the table using the formula of the update.
Regarding the training process, we have changed the values for epsilon and alpha, starting from a full exploration rate, and we decrease it until reaching 0 or 0.2, where it mainly uses the knowledge acquired.
We have created three different ideas, taking into account different attributes for each state, and we have been improving from the first idea to the last one, achieving the best possible one. In summary, for the first idea we have taken into account discretized distances to the closest ghost and to the closest pac dot, in addition to the most worthy direction to reach the ghost (vertical axis or horizontal axis). For the second idea, we have defined the states with discretized distances to the closest ghost and to the closest pac dot too, and we have added the recommended action, implemented in tutorial 1. Lastly, for the last idea, we have tried to fix the error of taking into account the presence of walls, and we have seen that it has been the best of the three ideas. We have also included to this idea the binary attribute of being more worthy to pursue the ghost or the pac dot, depending on the distance from pacman to them. With this last idea, we have obtained the best results, specially in lab 1 and 2. For the rest of the labyrinths, it has worked well with random ghosts.
To conclude, we reckon that this project has been very fruitful to explore the advantages of reinforcement learning since we have implemented this technique from the very beginning, and we have been able to discuss, depending on the type of state and the values of the parameters of epsilon and alpha, how the agent is able to learn.