Assignment 2 : Reinforcement Learning

Martín Ranea Marina and Alejandro Taboada Esteban
100474046-100472778

1. Explanation of each decision you made when defining the states and the reward function.

To construct the Q-table, we opted to utilize only two attributes in order to avoid high dimensionality. We selected these attributes based on their perceived significance in predicting the optimal direction for Pac-Man: the relative position of the nearest ghost (north, south, east, or west) in relation to Pac-Man, and the distance between them. Our matrix comprises 68 rows, as the maximum distance is 17 and there are four potential relative positions for the ghost.

Each row of the Q-table represents a combination of the distance, accounting for walls, separating Pac-Man and the ghost, along with the relative position of the ghost with respect to Pac-Man. As an illustration, the final row of the Q-table indicates the reward obtained by Pac-Man when the distance to the ghost is 17 (the maximum distance), and the ghost is positioned to the west of Pac-Man. The following table provides a visual representation to aid in understanding our Q-table structure:

	North	South	East	West
1. North (distance1, ghost in the north)				
…				
…				
68. West (distance8, ghost in the west)				

To determine the row number, we have implemented the computePosition() method. Allow us to provide a brief explanation of the process.

Initially, we establish a list to store the distances of all the active ghosts. This is accomplished by employing a loop: if a ghost is alive, we add its distance from Pac-Man (factoring in the presence of walls) to the list; otherwise, we append a substantial value to signify a significant distance. The purpose behind creating this list is to identify the closest ghost in proximity to Pac-Man.

Next, we proceed to determine the minimum distance from the list and utilize a loop to locate the index corresponding to this distance, which represents the closest ghost. This allows us to obtain the ghost's position using the .getGhostPositions() function. With the positions obtained, we employ specific conditions to ascertain the relative position of Pac-Man in relation to the ghost. For instance, if the x-coordinate of the ghost position (x1) is smaller than the x-coordinate of Pac-Man's position (x2), it implies that the ghost is positioned west of Pac-Man.

Based on the relative positions of the agents, we partition the Q-table as follows: each row or state encompasses a combination of a position and a distance. The initial 17 rows represent the combinations for the north direction, followed by 17 rows for the east, another 17 rows for the south, and the final 17 rows for the west. Consequently, our Q-table consists of 4 columns and 68 rows. After experimenting with various combinations, we determined that this approach strikes a balance between manageable matrix dimensions and the significance of the selected attributes for predicting Pac-Man's actions.

Moving on to constructing the reward function, our focus lies in the distance between the ghost and Pac-Man. When transitioning to the next state, if the distance has decreased compared to the current state, the reward is computed as the difference between the scores of the two states, plus one. Conversely, if the distance has increased, the reward becomes the difference between the score in the next state and the score in the current state, minus one. To determine the reward value, we have implemented the getReward() method.

Within this function, we once again compute the position of the closest ghost in both the current state and the next state. Subsequently, we apply the aforementioned conditions to determine the appropriate reward:

- If the distance has decreased in the next state compared to the current state, the reward is calculated as the difference between the score in the next state and the score in the current state, plus one.
- If the distance has increased, the reward is computed as the difference between the score in the next state and the score in the current state, minus one.

2. The experimentation that was carried out at each phase.

In the first phase, which involved selecting the state information and designing the reward function, our primary objective was to identify a relevant combination of attributes that would result in an effective Q-table and yield improved results. This

phase consumed a significant amount of time due to its crucial influence on subsequent phases. In our perspective, we found that focusing on attributes related to distances between the agents (Pac-Man and ghosts) and the actions performed yielded promising outcomes. This decision was influenced by our prior experience where we observed that the action attribute consistently held considerable relevance in score prediction.

Initially, we experimented with a wider range of actions, including southeast, northwest, southwest, northeast, north, south, east, and west. However, upon considering the final dimensions of the Q-matrix, we realized that it would result in an excessive number of rows. Consequently, we decided to limit the actions to just north, south, east, and west, combined with the distance to the closest ghost.

Moving on to the second phase, which involved agent generation, we implemented the three essential methods required for the completion of the Q-learning algorithm:

- *computePosition()*: This method computes the row/state index in the Q-table and is utilized within the update method.
- *update()*: The update method calculates the new values for each cell in the Q-table using specific formulas. We have distinct formulas for terminal states and non-terminal states.
- *getReward()*: This method assigns a higher value to the table when Pac-Man is approaching the ghost, thus providing a reward mechanism.

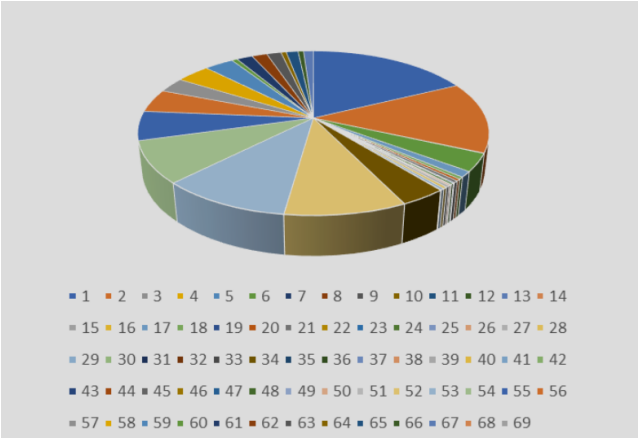
Once these three functions were correctly implemented, we began testing the game's functionality. To achieve this, we thoroughly examined the code and identified three important parameters that significantly impact gameplay: alpha, gamma, and epsilon. Alpha represents the learning rate or step size, gamma is the discount factor that is multiplied by the estimation of the optimal future value, and epsilon is related to the epsilon-greedy action selection procedure in the Q-learning algorithm.

Based on our understanding of these parameters, we concluded that increasing the value of epsilon, which ranges between 0 and 1, would result in a greater number of random decisions being made. Therefore, our initial approach involved running the busters.py file in the terminal with a higher epsilon value than 0. As the Pac-Man agent seemed to exhibit learning behavior with the current epsilon value, we gradually decreased its value. We could assess the learning progress by observing the updated Q-table, which was continuously modified by the implemented functions, along with the rewards assigned to each state.

Ultimately, the optimal value for the epsilon parameter is 0, indicating that the movements of the Pac-Man agent are no longer random.

The subsequent step involves testing the algorithm in the game environment. To accomplish this, execute the code using the command "*python busters.py -p QLearningAgent -l labAA1 -k 1 -t 0.01*". Here, the "-p" flag specifies the utilization of the QLearning class, "-l" is used to select the layout, "-t" is employed to enhance game speed, and "-k" determines the number of ghosts. We iterate through layouts 1 to 5, playing each layout until achieving the desired results. Notably, a challenging aspect was ensuring the Q-table's compatibility with all layouts. Therefore, we created two separate tables: one for layouts 1, 2, and 3, and another for layouts 4 and 5.

In the following graph, we present an example of a column from the Q-table. Each cell within the column can contain various values, up to the total number of rows in the table.



We have reached different conclusions regarding this assignment. Our main finding is that several factors need to be considered to develop a Q-learning algorithm that can adapt to various layouts. One crucial aspect is the impact of adjusting the three parameters of Q-learning on Pac-Man's decisions and selected actions. Additionally, it is important to avoid overtraining Pac-Man, as it could result in a model that performs exceptionally well in some maps but poorly in others. Therefore, it is preferable to have a model that performs effectively across all possible layouts. Instead of relying on an abundance of similar data, it is advisable to avoid creating a complex model by incorporating numerous attributes.

Furthermore, we have realized that reinforcement learning is a highly effective approach for video games. This is because it offers a straightforward means of providing greater rewards for playing the game well, while also imposing negative consequences for poor performance.