Artificial Intelligence Project Report

Mohamed Abd El Rahman

201700030

Contents

1 Question 1

In this question, we are required to create an evaluation function that takes the current state and a legal action to compute the new state and return a number that represents the evaluation of this action.
The evaluation function I used is given by the following equation:

$$\begin{aligned} Evaluation =& score - \frac{2.5}{min\_active\_ghost} + \frac{4}{min\_scared\_ghost} + \frac{2}{min\_food} \\ &+ \frac{1}{num\_food} + \frac{4}{min\_capsule} - 5 * num\_capsule + wait\_penalty \end{aligned} \tag{1}$$

Where:

1. score = Pacman score in this particular state(score shown in the GUI).
2. min_active_ghost = distance between Pacman and the nearest active ghost.
3. min_scared_ghost = distance between Pacman and the nearest scared ghost(if any).
4. min_food = distance between Pacman and the nearest food piece.
5. num_food = Number of food pieces in the new state.
6. min_capsule = distance between Pacman and the nearest capsule(if any).
7. num_capsule = number of capsules in the new state.
8. wait_penalty = -50 if the action produced this state was "STOP" and 0 otherwise.

The rationale behind each item in the evaluation function is given by the following:

1. score: the score increases if Pacman eats a food piece and decreases if it gets eaten by ghost, so including the score in the function encourages Pacman to eat more food and avoid getting eaten by a ghost.
2. min_active_ghost: subtracting the reciprocal of the distance to the nearest ghost means that the closer the ghost the smaller the evaluation function will be. Therefore, Pacman will tend to avoid getting closer to ghosts, and thus avoid losing. The weight of 2.5 is intentionally greater than the weight of getting closer to food, to prevent pacman from chasing food very near to ghosts and dying.
3. min_scared_ghost: adding the reciprocal of the distance to the nearest ghost means that the closer the scared ghost the larger the evaluation function will be. Therefore, Pacman will tend to get closer to scared ghosts and eat them, and thus increasing its score.

4. min_food: adding the reciprocal of the distance to the nearest food piece means that the closer the food the larger the evaluation function will be. Therefore, Pacman will tend to get closer to food which is needed to win.

5. num_food: adding the reciprocal of the number of food pieces means that the lesser the food pieces the larger the evaluation function will be. Therefore, Pacman will be encouraged to eat nearby food.

6. min_capsule: same rationale used with min_min food.

7. num_capsule: subtracting the number of capsules encourages Pacman to eat them. In this case, reciprocal is not used, because number of capsules may be zero.

8. wait_penalty: this penalty reduces the evaluation score of any state resulting from the action "Stop" to discourage Pacman from stopping which decreases its score as the time passes.

2 Question 2

To implement minimax algorithm we need to decide who is max and who is min. In our case, Pacman is max and we can have N number of ghosts, so we have N min agents. To account for the fact that we have several min agents we made use of the depth value as an extra parameter added to the min function, it can identify which min agent is currently making a move. Knowing the total number of agents and the depth, which begins from zero in the first call and gets incremented by 1 every recursive call, we can get the index of the agent using *depth mod M*, where M is the total number of agents.

In each turn, max should decide the best action to make based on the actions of all the other min agents. This is achieved through the following steps:

1. max function is called with the initial state and depth zero.

2. max finds all the legal actions and calls min function several times each time giving it a new state after applying one of the legal actions and depth+1. max function returns the maximum score among all the scores returned by the different calls to the min function.

3. min function will calculate the agent index as described before, and find all the legal actions for this agent given the current state. If the current agent is the last min agent to play before max then max function is called several times each time giving it a new state after applying one of the legal actions and depth+1, but if the current was not the last min agent then min function is called instead of max function. min function returns the minimum score among all the scores returned by the different calls to the min/max function.

4. The above process keeps recursing until terminal state is reached where the function just returns the current score. This score is propagated upwards until eventually all recursive calls are over, and the initial max function call gets to choose the best action corresponding to the maximum score.

The above algorithm is extremely slow to the extent of being unusable, since the game tree gets very large with few turns. Therefore, in this project the game tree depth was limited to 2/3 which means that in every action Pacman will only consider the consequences of this action on the next 2/3 moves only.

2.1 Comments

Pacman is good at not dying, it is not good at winning. This is because, the current evaluation function used is just the score, shown in the GUI, of each state. This score penalizes Pacman if it gets eaten by a ghost and awards it if it eats ajacent food. Therefore, Pacman has no mean of getting close to far away food, and it can only eat adjacent food or avoid getting eaten by a ghost.

3 Question 3

As discussed in the previous section, minimax algorithm becomes unusable with large game trees due to its computational complexity. In this part, we attempt to reduce the complexity of minimax by preventing the agents from discovering useless branches which will never be chosen by implementing alpha beta pruning. To implement alpha beta pruning I used the same exact algorithm as the one in question 2, but I added two more parameters to the min and max functions which are alpha beta. Alpha and beta are initialized in the first call to $-\infty$, ∞ respectively. Alpha corresponds to the maximum score found by a maximizer in this path while beta corresponds to the minimum score found by a minimuzer in this path.
I amended the algorithm in the previous question by some extra new lines to update alpha in the max function and beta in the min function, and to make the max function check if the score of the current action is greater than beta then we simply discard this branch since it will never be taken by min. On the other hand, min function will check if the scre of the current action is less than the alpha then we simply discard this branch since it will never be taken by min.

3.1 Comments

The performance of Pacman does not change at all after implementing alpha beta pruning, however the game now runs smoother with less computation required.

4 Question 4

In questions 2 and 3 we modelled the ghosts as optimal agents, which in fact is not true. The ghosts are random agents. Therefore, modelling them as optimal prevents Pacman from potentially achieving a better score as it will always be cautious from the ghosts that it thinks are optimal. In this question, we provide a more realistic modelling of the ghosts using the expectimax algorithm. In expectimax algorithm, we change the min function to return an expected score based on the score of each action and the probability of choosing this action instead of just returning the minimum score found among all actions. In our case, all actions are equiprobable.

4.1 Comments

The main difference between Pacman's performance with expectimax vs its performace with minimax or alpha beta pruning is the fact that it's less likely to commit suicide if it feels that it is getting trapped. In minimax or alpha beta pruning, sometimes pacman will predict that it will get trapped by the two ghosts, since it thinks that they are optimal and will choose to trap and eat him. Pacman's reaction in this case is to commit suicide by moving straight towards the nearest ghost to die quickly, since this is the best available action as dying later will produce less score due to time penalty. However, with expectimax Pacman does not always predict that the ghosts will always be optimal, so he might think that he can escape a trap if a ghost does not chose the optimal move to trap it. Therefore, Pacman is now less likely to commit suicide.

5 Question 5

In this question, I used the same exact evaluation function used in question 1, since it already had everything needed, in my opinion. The only change is that now, the evaluation function has no wait penalty, because we do not not the action that produces this state.