Question 1

Introduction:

The problem statement asked for the improvement of a reflex agent Pacman and hints on using food and ghost locations to evaluate the action depending on the current state the agent is in. Template code provided gives a number of information that can be extracted from currentGameState object.

Analysis & Explanation:

There were a few factors considered when evaluating an action based on the current state - the distance of Pacman to the ghost, whether the ghost is scared, distance of the Pacman to the nearest food and the number of food pellets. The issue was to assign appropriate weight to these factors in order for the Pacman to avoid the ghost and eat the food pellets appropriately to reach end game. The weight assignment was performed using trial and error method to ensure Pacman behaves as intended. After attempting various weight values, I found the values that fulfills the criteria stated by the question.

The score value is initiated as that of the current state so as to judge what would the action producing highest score from that state would be. Since getting close to food pellets is equally as important as staying away from the ghosts, the factors were given a weight of 2. In case of food pellets, the distance of the closest food to the Pacman's successor state (the state Pacman will be in after the action taken) was calculated. and subtrated from the score. The subtraction ensured that Pacman would try to minimize the distance to the nearest food pellet in order to maximize the score. On the other hand, the opposite was done for the distance of the closest ghost to the successor state. If the nearest ghost was found to be scared, the distance multiplied by 2 was subtracted from the score so Pacman would try to get close to the ghost. Otherwise, the distance multiplied by 2 was added to the score so Pacman would try to maximize distance to the closest ghost.

However, just rewarding Pacman based on the distance to the food pellets and ghosts does not produce the desired behavior thus another factor has to be introduced - the number of food pellets. This ensures that Pacman does not just stay close to the food pellet to maximize reward but eats the food pellet as well. Thus Pacman is rewarded extra 5 points if

number of food pellets have decreased from one state to another. This ensures Pacman selects the action that decreases the number of food pellets. Another factor that is considered is the action itself. If the action is "STOP" i.e. the Pacman does not make a move, typically due to receiving high rewards for being close to the food pellet and far from the ghost, the Pacman is penalized by 10 points and thus have to make a move.

Interesting Findings:

The first problem and interesting observation was when different weights are assigned to Pacman regarding distance to closest ghost and closest food pellet, one of the two happens:

- Pacman either tries to run from the ghost and not eat the food pellets because more rewards are associated with distancing from the ghost
- or Pacman tries to eat the food pellets at the risk of being eaten which leads to its defeat when the ghost eats it

Another observation was including the living penalty. It was fruitless to include the living penalty in the evaluation process because when the actions are ultimately compared the living penalty becomes a constant in all the values and thus end up not affecting the Pacman's behavior for choosing between the actions. Thus it was excluded.

Pacman was also noticed to eat food pellets more when the ghost was nearer than further. This could be related to the decrease in the influence of rewards when avoiding the ghost being higher compared to the reward of eating the food pellets. As an example, when the ghost is far away, the Pacman has more incentive to avoid the ghost since that gives immediate higher reward, compared to the lower reward that eating food will provide.

Challenges:

One of the challenges faced was to ensure that Pacman actually eats the food pellet rather than roaming around it or staying in one place, which incurs the living penalty. This was solved by rewarding the Pacman if number of food pellets decreased from current to successor state and penalizing if Pacman stays still.

Another challenge was balancing the weights, reward and penalty to ensure Pacman behaves as required. This was solved using countless trial and error sessions by adjusting

weights, adding more factors and observing the change in Pacman due to the changes made. Although the current approach passes the tests by Autograder, Pacman still exhibits the behavior of not eating the food even when the pellets are nearby in states where the ghosts are far away.

Behavior of code for different hyper parameters:

During the trial, some of the factors considering alternatively were - number of capsules (the number of power pellets) and decrease in number of agents. However, neither of these factors made a difference in the score or the behavior of Pacman. This was done initially to encourage the Pacman to eat the capsule and the ghost. However, since the number of capsule and agent is always 1, the overall impact on the score might not have been significant enough compared to other factors. It could also be that we are comparing between two states here and the number of power pellet and ghost are likely to stay the same so the reward/penalty ends up being a constant value. Reflex agents also prioritize eating the capsules if they are nearby but does not take into consideration whether the capsules exist or not. The same also applies to number of ghosts.

Rewarding based on the difference in number of food pellets compared to rewarding a fixed amount if the food pellet count decreased turned out to be the same. This is because only food pellet count can only decrease by 1 from one state to another. Thus both the lines end up having the same result.

```
if(successorGameState.getFood().count() < currentGameState.getFood().count()):
    score += 5
if(successorGameState.getFood().count() < currentGameState.getFood().count()):
    score += 5 * (currentGameState.getFood().count() - successorGameState.
    getFood().count()) # 5 * 1 = 5
```

When the penalty for stopping is not applied to Pacman, the average score decreases. When reward is not provided for decreasing the number of food pellets, the Pacman resorts to stopping in one position that is near to the food and far from the ghost in order to maximize reward. However when this reward is applied, Pacman stops when the food pellet is far away as no immediate reward for consuming food is available in the next states.

When the scared time for the ghost is not considered, the result is divided. For some games,

the score sees an increase while for other games, the score sees a decrease. However, the average score worsens if scared time is not considered. This could be due to the fact that Pacman tries to run away from the ghost even when it is scared, which results in moving away from the food pellets sometimes and thus incurring living penalty when it could have eaten the ghost and the food pellets to earn more points. Furthermore, this might not be a concern to Pacman if it does not come into contact with the power pellet, which give the same result as not considering the scared time.

Question 2

Introduction:

The task required designing an adversarial agent that could work with any number of adversaries. The agent would have to maximize its own utilities while minimizing other agents' utilities.

Analysis & Explanation:

This solution was provided by our lab instructor that was general for different number of agents. It calls a function for obtaining an action leading to the state with the highest score. This in turns calls either of the two function - minValue or maxValue, depending on whether the agent is Pacman or a ghost. For Pacman, maxValue is called since we aim to maximize the value achieved by the Pacman and for ghosts, minValue is called to minimize the value achieved by ghosts, which will in turn maximize Pacman's score.

Both of the above mentioned functions are similar in their functionalities in that they explore all possible moves and compares the score obtained by each of the states they end up in. For minValue, the minimum of the scores obtained is returned while for maxValue, the maximum of the scores are returned. Along with the score, the corresponding action is also taken. This ensures that Pacman takes the action that maximizes the score obtained.

Instead of exploring all possible states until terminal states/end of game, which would take a very long time, a depth of 2 is used to limit the search. This implementation assumes that the opponent, the ghost, is playing optimally and there is no probability associated with

each move i.e. if a move is supposed to land in a state A, it will always land in A.

Interesting Findings:

One of the observations that was apparent when running Autograder with depth 2 on smallClasssic layout was Pacman staying stationary when there is no ghost or food pellets nearby. This behavior is similar to the problem faced in the first question where Pacman did not move due to lower number of food pellets being further away and ghosts being far away as well. This is mostly due to the fact that Pacman runs a depth 2 search and cannot find any improvement in score which is why it does not make a move.

Another observation was Pacman choosing to die when cornered. This ensured instead of being penalized while trying to avoid the ghosts in vain, Pacman chooses to die faster.

Behavior of code for different hyper parameters:

I explored some of the layouts in order to get a better understanding of how the code behaves:

trappedClassic (2 ghosts):

Pacman's behavior of staying stationary is observed clearly at a depth of 2. At a certain point in the game, Pacman keeps moving away when the ghost comes near but as the ghost moves away again, Pacman moves back into the previous position and repeats this. Pacman also loses the game, possibly due to limited depth and staying motionary instead of ending the game earlier by eating food.

With increasing depth, the game plays out much slower because of the increased amount of time needed to calculate the next move. It takes Pacman about 2-3 seconds per move at a depth of 4. Pacman shows the behavior of stopping when far from ghosts and food pellets but the number of such occurrences decreases and Pacman ultimately won the game.

minimaxClassic (3 ghosts):

As the layout is smaller and number of ghosts is higher, with higher depth Pacman seems to have a higher winning rate. At a depth of 4, Pacman won 4/6 of the games which was a slight improvement of 3/6 winning rate at a depth of 2. Pacman still displayed the behavior of remaining stationary but this occurred much less as the layout was small and ghosts were

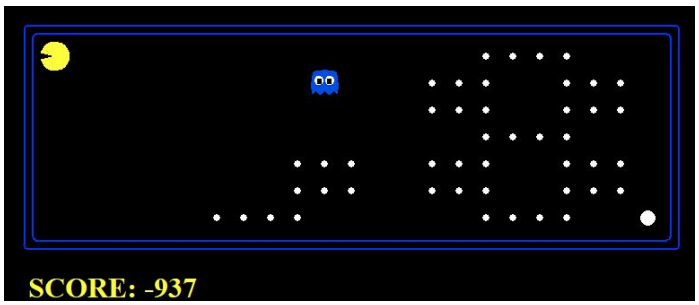
thus most of the time near to Pacman. It is observed with increasing depth, the winning rate does not always improve due to the randomness of the ghost. However, Pacman wins or dies fast as depth increases because it is able to predict its loss or win more accurately.

mediumClassic (2 ghosts):

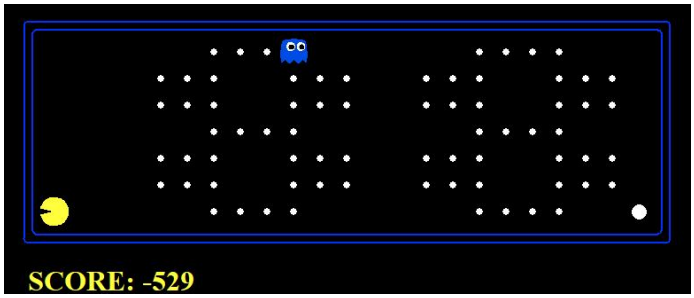
At this layout, Pacman remaining stationary is observed much more than the previous layouts. When depth value is set to 2, agent remains stationary after sometime. At a point where there is very little food available and when ghosts come near, Pacman makes some moves to avoid the ghost and continues to remain stationary. This continues until Pacman loses. When depth value is set higher, Pacman also behaves the same but wins the game at the end.

openClassic (1 ghost):

At a depth of 2, after eating the nearby food pellets, Pacman gets stuck in a corner with no food pellet nearby and no ghost. This continues even though Pacman gets a high negative score. This can be due to the limited depth, where Pacman does not see any change in the score no matter which action is taken.



However, the opposite occurs for higher depth. Pacman is stuck near the food pellet but does not eat it even as the score becomes lower and lower.



Overall it was observed that on smaller layouts, Pacman can win regardless of the depth used. With increasing depth, the rate of winning increases but as does the time and

computational power needed to calculate each move. Smaller layouts also ensure that ghosts are usually never too far away from Pacman so Pacman do not remain stationary for too long. On larger layouts, Pacman eats all the nearby food and stays in one place until the ghost comes near. The larger layouts also make it impossible to explore high depth values since each move takes much more time. Only a maximum depth of 4 was explored and due to computational resource constraint, higher values of depth could not be explored.

Question 3

Introduction:

The question asked to design an algorithm that can be used apply alpha-beta pruning to as many minimax agents as required.

Analysis & Explanation:

The only difference between AlphaBetaAgent and MinimaxAgent is that there is pruning involved to ensure better perfomance. Instead of traversing through all possible branches, two variables are introduced: alpha and beta. Alpha keeps track of the highest score found so far for Pacman and beta keeps track off the lowest score found for the ghosts. This leads to less nodes being explored by avoiding the nodes that might give less desired result.

Interesting Findings:

Surprisingly, when run by Autograder, AlphaBetaAgent and MinimaxAgent performs almost the same. Both of these result in the same score and outcome - Pacman losing with a score of 84. Pacman does exhibit the behavior of not making a move at times but for a very short period of time as ghost comes after Pacman, causing Pacman to move to escape.

Similar to the previous question, when cornerered, Pacman chooses to die faster since AlphaBetaAgent is also a Minimax agent with the objective of maximizing score. The difference achieved by pruning in different layouts and number of agents is explored further in the section below. However, in general, the game runs faster and smoother with

AlphaBetaAgent compared to MinimaxAgent.

Behavior of code for different hyper parameters:

For this question, I again explored different layouts and depth to test how it performs:

trappedClassic (2 ghosts):

At a depth of 1, it is noticed that the Pacman stops at a certain time when the blue ghost is two steps away and the orange ghost and food pellets are far away. This is because due to depth being 1, Pacman cannot see the ghost. Compared to this, at a depth of 2, Pacman does not stop because it detects the blue ghost 2 moves away from it and moves away. In one of the two observed cases, blue and orange ghost spawns in front and behind Pacman and Pacman becomes cornered. In both depth values 1 and 2, Pacman kills itself immediately in order to avoid living penalty and ultimately loses.

An interesting observation about higher values of depth for this layout is, Pacman moves towards the ghost to kill itself immediately. This is because the layout is smaller and there are two ghosts. Pacman senses ghosts 3 moves away each time and assumes the worst i.e. it will be eaten by the ghost. It loses each time.

minimaxClassic (3 ghosts):

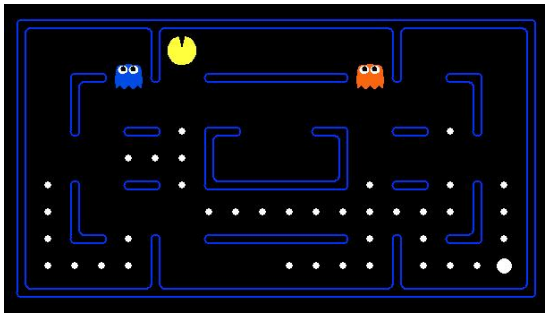
In this layout, at a lower depth values of 7 and below, Pacman senses being cornered and immediately moves to the ghost to die. In some cases, Pacman does not sense food or ghosts and does not move. When a ghost is near, Pacman senses the ghost and moves away to eat food and avoid ghost, resulting in win but sometimes Pacman instead move towards another ghost that it could not sense due to depth limitation and gets eaten.

At a lower depth of 3 and below, the score of games where Pacman loses is lower compared to those played with a higher value of depth. This is because Pacman senses the ghost faster at higher depth and dies faster in order to avoid dragging out a losing game.

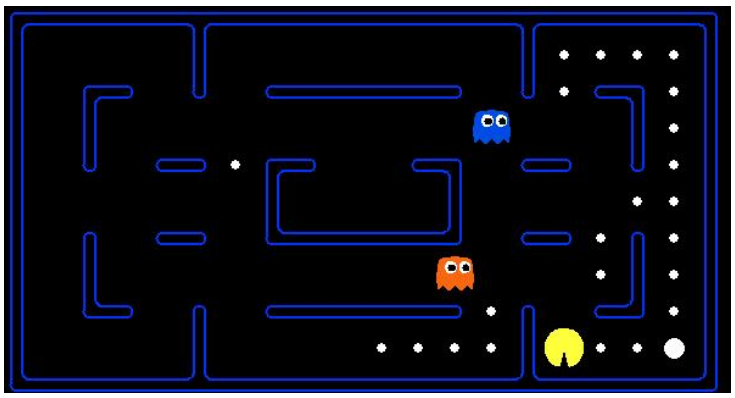
mediumClassic (2 ghosts):

At a depth of 2 and below, Pacman displays thrashing behavior as it stays stationary (as shown in the figure below) because in some cases, there are no ghosts or food pellets within the limited number of moves that it can see. However, this leads to death in some cases because Pacman only senses ghost when it gets too near and then it chooses to die to

maximize the reward.

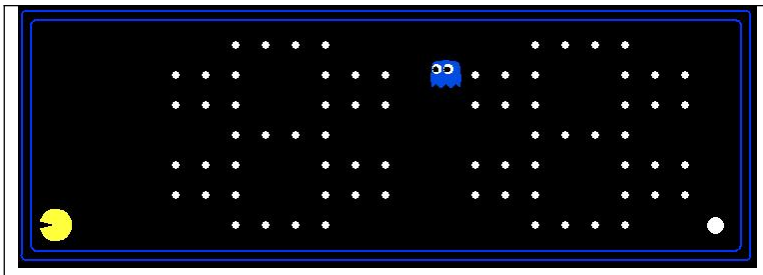


However, at a depth of 3 and above, Pacman displays the behavior of staying stationary near the food (as shown below) even though the calculations should have informed Pacman there is a reward for that move. A noticeable difference is when there is a ghost 1 move away, Pacman makes its move quicker in order to move away. It is also noticed that if Pacman is cornered, it makes a move to die but in some cases, since ghost is a sub optimal opponent, the ghost moves away. Thus Pacman again moves away.

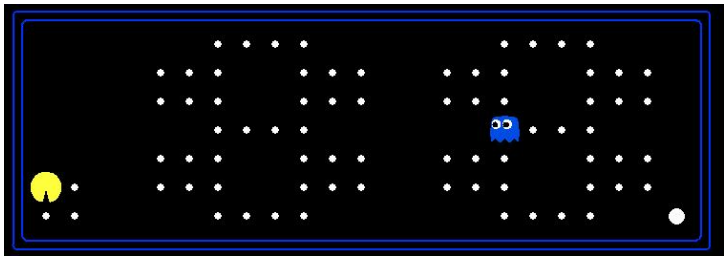


openClassic (1 ghost):

As shown below, at a depth of 2 and below, Pacman eats the first group of food pellets near its spawning position and then stays stationary (shown below) as it cannot see any improvement in score due to the distance to the food pellet. The ghost also stays far away so Pacman stays in that position since it cannot eat any food pellet and cannot kill itself either due to the distance to the ghost.



At a depth of above 2, for some reason, Pacman stays stationary near the food pellet and does not move even though eating the food pellet would reward Pacman. Similar to the dilemma at lower depths, Pacman does not move even though the score becomes a large negative number.



In conclusion, AlphaBetaAgent produces the same result as MinimaxAgent but with better performance. It still exhibits some of the same issues as the previous solution does, which is a plus point since it means pruning does not affect the final outcome very much.

Question 4

Introduction:

This question requires modifying the code of the minimax agent to adjust to that of a sub-optimal adversary i.e. an expectimax agent.

Analysis & Explanation:

Expectimax takes into account that the opponent is not playing optimally but instead each of its action is associated with a probability i.e. it can take any of the actions randomly. The probabilities are calculated using 1/number of moves and multiplied with the score. The product of that is summed and the result is considered the possible score for that state. For

calculation of Pacman's possible action is the same as minimax agent with the maxValue function. This makes sure that Pacman tries to maximize its result while considering that the opponent is sub optimal.

Behavior of code for different hyper parameters:

trappedClassic (2 ghosts):

This layout does not show much differences compared to the previous solution. As mentioned before, Pacman either kills itself and lose or successfully avoids the ghost and win. This is because Pacman is still maximizing its score and ghosts usually have two possible actions in any state thus probability of it being eaten is still high. However, the program runs more smoothly at high depth values.

minimaxClassic (3 ghosts):

One of the differences noticed is Pacman no longer killing itself unless absolutely cornered. It considers that ghosts might move randomly and thus sometimes takes chances.

mediumClassic (2 ghosts) and openClassic (1 ghost):

In both these layouts, while Pacman no longer takes extreme actions to end game early when ghost is near, all other behavior remains the same. In case of mediumClassic, Pacman chooses to avoid ghost and eat pellets at times but for openClassic, the ghost is too far away so Pacman's behavior for being near the ghost could not be observed properly.

It was my observation that expectimax performs a little worse than AlphaBetaAgent but better than MinimaxAgent.

Question 5

Introduction:

The question asked for a function that could evaluate states than actions. The function receives the current game state object with information associated with it which must be used to come up with a score for the state.

Analysis & Explanation:

There are a number of information that can be obtained from the current state that could be used to evaluate the state :

- Number of agents
- Number of capsules
- Distance to closest ghost
- Distance to closest food pellet
- Number of food pellets
- Scared time of the ghosts
- Current score of the state

Following a method of trial and error, the following approach was found to be efficient among other working solutions. Logically, a state would be better if they had less capsules and food pellets, Pacman was closer to the food pellet and further away from the ghost and the ghosts were scared. Thus to account for these, each were given a weight deemed appropriate after trial and errors and subtracted or added to the score.

Since moving towards food pellets were important, this was assigned a weight of 2. Pacman's distance to the closest food is subtracted so Pacman tries to decrease the distance.

Next, food and capsule count were considered as equally important. This is to make sure Pacman evaluates states where the capsule and the food is eaten with higher score. They were given a factor of 4, where both food and capsule count was subtracted from the score so Pacman would try to minimize these values. Additionally, since count of capsules and food pellets would be a lower value than distance, higher multiplier would ensure they are prioritized as much, if not more than minimizing the distance to the nearest food pellet.

Interesting Findings:

During the various games, it was noted that Pacman sometimes stops indefinitely and does not move even ghost approaches. This happened during one of the trials and errors and Pacman stayed stationary until it lost when the ghost ate it. Some of the possible reasons could be because at that point, the reward and penalty could have balanced out or staying in that position provided more reward compared to avoiding ghost or eating food. Pacman also

exhibits the behavior of staying stationary more frequently when the number of food pellets is very low.

When assigning the same weight assigned to number of food pellets to 'distance to the closest food', it was found that Pacman stayed near the food to maximize reward instead of eating the food pellets. This could be because while distance is usually a higher value than the number of food pellets, thus giving a higher reward compared to consuming the food pellet.

Unlike my logical assumption, Pacman performed worse when ghost distances were considered. The ghost distances' value being large could have had too big of an impact on the agent's decision which made count of capsules a great feature to control the ghost indirectly. If Pacman is motivated to eat the capsules, the ghost will not be a threat most of the time and number of capsules is also a low enough number with a lower range of 0 to 2.

Challenges:

One of the challenges is Pacman stopping near the food when it is further from the ghost to maximize the reward. In the solution approach to question 1, this behavior was penalized with a negative penalty of 10 if the action performed was "STOP". However, there was no action available to evaluate this time. This could be partially solved using capsule count with a weight of 4, as mentioned. This gives Pacman more incentive to move so as to eat the capsules as well. The problem still remains at times when food pellets and capsules are far from Pacman.

Another challenge was to perform trial and error with many features which seemed like it would work but ended up producing unexpected failure. This required multiple experiments to understand what features should be considered and what weights should be assigned.

When I attempted to apply the same solution in question 1 here, the average score was much lower and Pacman won a little more than half the games. Since the number of capsules was not previously used in that task, it took me a while to search through pacman.py and find other features that could be used and ultimately lead me to find an appropriate solution.

Behavior of code for different hyper parameters:

Not using capsule count:

As mentioned, Pacman stops at times until the ghost is very near which costs him living penalty, resulting in a lower average score and a loss in one out of 10 games. Without capsule count, Pacman may remain stationary for a longer amount of time and unmotivated to move, adding to the living penalty. The average score drops by a significant amount (by about 30%).

Using a weight of 2 for capsule count:

Pacman exhibits the behavior of staying near the capsules in order to maximize reward since the weight given to decreasing count of capsules is the same as given to avoiding ghosts and staying near food. Since the weight is multiplied by the distance which is usually more than 1, Pacman prioritizes the other factors than consuming capsules - thus the problem of Pacman remaining stationary. However, ultimately Pacman wins all the 10 games with a lower average score compared to the current solution.

Considering ghost distance with/without scared time:

While considering the scared time, the average is lower than when scared time was not considered. Pacman also wins 8/10 games when scared time is considered compared to the 9/10 win without considering scared time is considered. This could be because when ghost distance is considered, Pacman is more focused on running away from the ghost or moving towards the ghost than eating the food pellets.

Considering minimum capsule distance:

This modification produced one of the most varying results among all other experiments I conducted. Pacman was very fast to eat all the food pellets at first until only two food pellets remain. Even when the score approached a very high negative score, Pacman does not consume the food pellets. Pacman is also observed to move only to avoid the ghost and then move closer to the food pellet again. Thus it can be deduced that the reward from staying near the food and away from the ghost outweighs the reward of eating the food

pellet. Since the living penalty was not included in the evaluation, Pacman does not know that the score is decreasing in each time step. Even when I attempted to include a living penalty of -10 for every state, it did not remove this problem. For this modification, Autograder timed out.