Omar Al Asaad
0075155

Q1:

Firstly, I calculated the distance from Pacman to the nearest food pellet using Manhattan distance. This feature prioritizes the consumption of nearby food, ensuring Pacman's sustenance.

Secondly, I evaluated the proximity of ghosts, factoring in their scared timers. If a ghost was too close, it posed a threat, while reachable scared ghosts presented opportunities for point acquisition. Reciprocal values were used to emphasize the significance of certain features, such as close food, where higher values indicate greater importance.

Additionally, I employed thresholds for food distances, categorizing them into different ranges with corresponding scores. This approach allowed Pacman to prioritize immediate food sources while considering more distant ones when necessary.

Lastly, I incorporated a ghost strategy that distinguished between scared and active ghosts. Scared ghosts were rewarded when close, encouraging Pacman to pursue them when safe, while active ghosts imposed substantial penalties to deter proximity.

Q2:

Alpha-Beta pruning is indeed faster than Minimax due to its ability to skip unnecessary computations. Minimax evaluates all possible moves, which can be time-consuming for games with many options. In contrast, Alpha-Beta pruning cleverly eliminates branches of the game tree that won't affect the final decision. Alpha-Beta pruning maintains two values, alpha and beta, to determine which branches can be safely ignored. If it finds a branch where the alpha-beta condition is met it prunes that branch, saving valuable processing time.

Q3:

Yes, both were the same because they both use the same evaluation function and both are designed to find the optimal move.

Q4:

The code ran slower and took different actions compared to the other two. This is probably due to the fact that it has chase nodes.

Q5:

The updated evaluation function is simpler and easier to understand. The average score in the first one is 100 less than the new one. It directly calculates the score based on factors like the distance to the closest food pellet and the proximity to ghosts. This makes the code more straightforward and readable.

```python
def betterEvaluationFunction(currentGameState: GameState):
    """
    Your extreme ghost-hunting, pellet-nabbing, food-gobbling, unstoppable
    evaluation function (question 5).

    DESCRIPTION: <write something here so we know what you did>
    """
    "*** YOUR CODE HERE ***"
    newPos = currentGameState.getPacmanPosition()
    newFood = currentGameState.getFood()
    newGhostStates = currentGameState.getGhostStates()
    score = currentGameState.getScore()

    foodDistances = [util.manhattanDistance(newPos, food) for food in
newFood.asList()]
    nearestFoodDistance = min(foodDistances) if foodDistances else 0
    score += 1.0 / (nearestFoodDistance + 1)

    for i in range(len(newGhostStates)):
        ghost = newGhostStates[i]
        scaredTime = newGhostStates[i].scaredTimer
        distance = util.manhattanDistance(newPos, ghost.getPosition())

        if scaredTime > 0 and distance < scaredTime:
            score += 200 - distance
        elif distance <= 1:
            score -= 500

    return score
```

Q6:
For question 1, I focused on things like how close the nearest food was and how close ghosts were, giving more importance to these factors because they had a big impact on the agent's choices. I adjusted the importance of these factors by trying different values until the agent played well.

In question 5, I made a more advanced evaluation function. Again, I cared about food and ghosts, but I also considered the score. I tweaked the importance of these factors by testing different values to find the right balance between collecting food, avoiding ghosts, and getting a high score.