Tvrtko Zadro, NIA

State:

Q2 and Q3 working, but Q1 not working well.

Q1:

For evaluation function first I check if the potential next state is winning state, if so, I return big value because we should instantly prefer that state.

After that, I find manhattan distance to closest ghost, manhattan distance to closest food point and number of food points left. I use this to calculate score. I multiply by *ghostDist* because I prefer being further to the ghost and divide by *foodDist* and *numFood* because I prefer being closer to food and having less food left. I feel this could be a bit better if I put some coefficients, but I haven't spent too much time on this question, and by the time I got back to it I didn't have time to finish it. Current solution performing poorly.

```
if successorGameState.isWin():  # if next state is win return +inf
    return 999999

# find manhattan distance to closest ghost
ghostDist = min([util.manhattanDistance(newPos, ghostPos) for ghostPos in successorGameState.getGhostPositions()])

# find manhattan distance to closest food
foodDist = min([util.manhattanDistance(newPos, foodPos) for foodPos in successorGameState.getFood().asList()])
numFood = len(successorGameState.getFood().asList())

return (successorGameState.getScore() * ghostDist) / (foodDist * numFood)  # prefer being further to ghost and closer to food
```

Q2:

This tasking working well. I made a recursive function *minimax* consisting of two main parts: player's turn (maximization) and ghost's turn (minimization). This function is called once for every agent at single depth.

```
if agent == 0:  # if pacman then maximize
    values = []

    for action in gameState.getLegalActions(agent):  # for every action pacman can take
        nextState = gameState.generateSuccessor(agent, action)  # translate it to state it would lead to

        value = self.minimax(nextState, depth, agent + 1)  # and calculate recursively minimax for next agent

        values.append((value, action))  # add value and action leading to that value so we can use it later

    return max(values)  # since it's player's turn find maximum value
```

Pic 1. Maximizing

```
else:  # else it's one of the ghost and minimize
    values = []

    for action in gameState.getLegalActions(agent):  # for every action current ghost can take
        nextState = gameState.generateSuccessor(agent, action)  # translate it to a state it would lead to

        if agent == gameState.getNumAgents() - 1:  # if we are on the last ghost
            value = self.minimax(nextState, depth - 1, 0)  # call again for pacman but deduct depth
        else:
            value = self.minimax(nextState, depth, agent + 1)  # else we find min for next ghost

        values.append(value)

    return min(values)  # since it's opponent's turn find minimum value
```

Pic 2. Minimizing

In case of trapped Pacman (*python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3*), he rushes the closes ghost because he realizes that in no case he could ever get good score, by get good score meaning get food and by no case meaning exploring to depth 3. So agent decides it is better to finish the game quicker, since there is a penalty for staying alive.

Q3:

This task is also working well. I used pseudocode from Berkeley's task explanation. Code is similar to regular minimax, but main difference is in Q2 we first expand all children and then look for max / min. But here we go step by step, and for every action we compare value to previous best value. This way if we decide in the middle of process we want to stop expanding we can.

```
for action in gameState.getLegalActions(0):  # for every action pacman can take
    nextState = gameState.generateSuccessor(agent, action)  # translate it to action it leads to

    value = max(value, self.alphabeta(nextState, depth, alpha, beta, agent + 1))

    if value > beta:  # if current best value is bigger than what is worst value for minimizing player
        return value  # then further expanding won't make difference and beta prune

    alpha = max(alpha, value)  # update alpha

return value
```

Pic 3. Example of maximizing player