**Question 1 (5 points)** Improve the ReflexAgent in multiAgents.py to play respectably. The provided reflex agent code provides some helpful examples of methods that query the GameState for information. A capable reflex agent will have to consider both food locations and ghost locations to p0erform well. Your agent should easily and reliably clear the testClassic layout:

**python pacman.py -p ReflexAgent -l testClassic**

**Pacman emerges victorious! Score: 562**

**Average Score: 562.0**

**Scores: 562.0**

**Win Rate: 1/1 (1.00)**

**Record: Win**

**python pacman.py --frameTime 0 -p ReflexAgent -k 1**

**Pacman emerges victorious! Score: 1049**

**Average Score: 1049.0**

**Scores: 1049.0**

**Win Rate: 1/1 (1.00)**

**Record: Win**

**python pacman.py --frameTime 0 -p ReflexAgent -k 2**

**Pacman died! Score: -187**

**Average Score: -187.0**

**Scores: -187.0**

**Win Rate: 0/1 (0.00)**

**Record: Loss**

**Pacman emerges victorious! Score: 1083**

**Average Score: 1083.0**

**Scores: 1083.0**

**Win Rate: 1/1 (1.00)**

**Record: Win**

**How does your agent fare? The agent fares pretty good, few losses amidst many wins**

**python pacman.py -p ReflexAgent -l openClassic -n 10 –q**

**Pacman emerges victorious! Score: 1259**

**Pacman emerges victorious! Score: 1260**

**Pacman emerges victorious! Score: 1196**

**Pacman emerges victorious! Score: 1257**

**Pacman emerges victorious! Score: 1120**

**Pacman died! Score: 47**

**Pacman emerges victorious! Score: 1260**

**Pacman emerges victorious! Score: 1246**

**Pacman emerges victorious! Score: 1174**

**Pacman emerges victorious! Score: 1256**

**Average Score: 1107.5**

**Scores: 1259.0, 1260.0, 1196.0, 1257.0, 1120.0, 47.0, 1260.0, 1246.0, 1174.0, 1256.0**

**Win Rate: 9/10 (0.90)**

**Record: Win, Win, Win, Win, Win, Loss, Win, Win, Win, Win**

**Question 2 (20 points) Now you will write an adversarial search agent in the provided MinimaxAgent class stub in multiAgents.py. Your minimax agent should work with any number of ghosts, so you'll have to write an algorithm that is slightly more general than what appears in the textbook. In particular, your minimax tree will have multiple min layers (one for each ghost) for every max layer.**

**python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4**

**minimax values of the initial state : -492**

**Pacman emerges victorious! Score: 516**

**Average Score: 516.0**

**Scores: 516.0**

**Win Rate: 1/1 (1.00)**

**Record: Win**

**python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=3**

**minimax values of the initial state : 7**

**Pacman emerges victorious! Score: 513**

**Average Score: 513.0**

**Scores: 513.0**

**Win Rate: 1/1 (1.00)**

**Record: Win**

**python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=2**

**minimax values of the initial state : 8**

**Pacman emerges victorious! Score: 516**

**Average Score: 516.0**

**Scores: 516.0**

**Win Rate: 1/1 (1.00)**

**Record: Win**

**python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=1**

**minimax values of the initial state : 9**

**Pacman emerges victorious! Score: 516**

**Average Score: 516.0**

**Scores: 516.0**

**Win Rate: 1/1 (1.00)**

**Record: Win**

**python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4 --numGames 1000 --frameTime 0 --fixRandomSeed –textGraphics**

**Win Rate: 684/1000 (0.68)**

**python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3**

**Pacman died! Score: -501**

**Average Score: -501.0**

**Scores: -501.0**

**Win Rate: 0/1 (0.00)**

**Record: Loss**

**Question 3 (20 points) Make a new agent that uses alpha-beta pruning to more efficiently explore the minimax tree, in AlphaBetaAgent. Again, your algorithm will be slightly more general than the pseudo-code in the textbook, so part of the challenge is to extend the alpha-beta pruning logic appropriately to multiple minimizer agents.**

**python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic**

**Average Score: 954.0**

**Scores: 954.0**

**Win Rate: 1/1 (1.00)**

**Record: Win**

**python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=4 --numGames 1000 --frameTime 0 --fixRandomSeed –textGraphics**

**Win Rate: 687/1000 (0.69)**

**python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=4**

**minimax values of the initial state : -492**

**Pacman emerges victorious! Score: 516**

**Average Score: 516.0**

**Scores: 516.0**

**Win Rate: 1/1 (1.00)**

**Record: Win**

**python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=3**

**minimax values of the initial state : 7**

**Pacman emerges victorious! Score: 515**

**Average Score: 513.0**

**Scores: 513.0**

**Win Rate: 1/1 (1.00)**

**Record: Win**

**python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=2**

**minimax values of the initial state : 8**

**Pacman emerges victorious! Score: 516**

**Average Score: 513.0**

**Scores: 513.0**

**Win Rate: 1/1 (1.00)**

**Record: Win**

**python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=1**

**minimax values of the initial state : 9**

**Pacman emerges victorious! Score: 516**

**Average Score: 513.0**

**Scores: 513.0**

**Win Rate: 1/1 (1.00)**

**Record: Win**