Implementing adversarial search using alpha-beta pruning on top of a PAC-Man game. I used an implementation of Pac-Man projects developed at UC-Berkeley for the introductory AI course. In addition, I used implemented codes in some parts of my project, I referenced them inside the code.

Note: Python 2.7.2(Jun 12 2011) has been used to implement new parts.

First Task:

Implement alpha-beta pruning (slide 52 in game.pdf lecture) in the class provided in multiagents.py. Assume that there is only 1 ghost. Therefore, it becomes a two player game with pacman as the max node and the ghost as the min node. Use the existing evaluation function. Here depth will refer to the number of plys. For example, depth of 2 means Pacman and ghost each make two moves before returning the score.

Based on the game lecture that we had in the class: Alpha –Beta implementation in python would be:

Alpha-Beta Implementation

- α: MAX's best option on path to root
- β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        # top min-value not care what remains, if v > β
        if v > β return v  # must not prune on equality
        # update global max value
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        # top max-value not care what remains, if v < α
        if v < α return v # must not prune on equality
        # update global min value
        β = min(β, v)
    return v
```

Since we had to add this part to the implemented code, it changed to this:

```
class AlphaBetaAgent(MultiAgentSearchAgent):
    """
      Your minimax agent with alpha-beta pruning (question 3)
    """

    def getAction(self, gameState):
        """
          Returns the minimax action using self.depth and self.evaluationFunction
        """
        PACMAN = 0
        "+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
        '''
        We are using implemented parts in previous question to find out
        value of max and min here also
        '''
        def Information_About_Agent_MaxValue(state, depth, alpha, beta):
            if state.isWin() or state.isLose():
                return state.getScore()
            "Number of actions would be recorded in the action variable"
            actions = state.getLegalActions(PACMAN)
            "Best score would be initialized by the infinite value at first"
            Recorded_Best_Scored = float("-inf")
            score = Recorded_Best_Scored
            Happend_Action_Best_One = Directions.STOP
            for action in actions:
                score =
Information_About_Agent_MinValue(state.generateSuccessor(PACMAN, action), depth, 1,
alpha, beta)
                if score > Recorded_Best_Scored:
                    Recorded_Best_Scored = score
                    Happend_Action_Best_One = action
                    '''
                    Finding value of apha
                    Which is max value between alpha and recorded best scores
                    '''
                alpha = max(alpha, Recorded_Best_Scored)
                if Recorded_Best_Scored > beta:
                    return Recorded_Best_Scored
                "We would find best score in all depth until we reach to the first
depth"

            if depth == 0:
                return Happend_Action_Best_One
            else:
                return Recorded_Best_Scored
```

```
def Information_About_Agent_MinValue(state, depth, ghost, alpha, beta):
            if state.isLose() or state.isWin():
                return state.getScore()
            Available_Ghost_Next_One = ghost + 1
            '''
            Based on the state of the ghost we are trying to find out
            the number of available ghosts
            then we can find out which ghost would be closest one
            '''
            if ghost == state.getNumAgents() - 1:
                Available_Ghost_Next_One = PACMAN
            actions = state.getLegalActions(ghost)
            Recorded_Best_Scored = float("inf")
            score = Recorded_Best_Scored
            '''
            Based on the action that we recorded, we wil call evaluate function
            to evaluate and give us a score
            '''
            for action in actions:
                if Available_Ghost_Next_One == PACMAN:
                    if depth == self.depth - 1:
                        score =
self.evaluationFunction(state.generateSuccessor(ghost, action))
                    else:
                        score =
Information_About_Agent_MaxValue(state.generateSuccessor(ghost, action), depth + 1,
alpha, beta)
                else:
                    score =
Information_About_Agent_MinValue(state.generateSuccessor(ghost, action), depth,
Available_Ghost_Next_One, alpha, beta)
                if score < Recorded_Best_Scored:
                    Recorded_Best_Scored = score
                    '''
                    Same as finding value of Alpha
                    Finding value of Betta
                    Which is max value between Betta and recorded best scores
                    '''
                beta = min(beta, Recorded_Best_Scored)
                if Recorded_Best_Scored < alpha:
                    return Recorded_Best_Scored
            return Recorded_Best_Scored
        return Information_About_Agent_MaxValue(gameState, 0, float("-inf"),
float("inf"))
```

Based on the implemented code for Alpha Beta Pruning, I ran this comment:

```
python pacman.py -p AlphaBetaAgent -a depth=2 -l smallClassic –k 1
```

-p defines the agent type in the "pacmanAgents" module.

-a defines arguments that needs to send to the agent

Depth defines the number of players, here the number of players are two(1 ghost and one pacman)

-l defines layout of the game, here is "smallClassic"

-k defines the number of ghosts, here we have only one ghost

Pacman could win after getting -35 score and win rate 1/1

Second Task:

Score the leaves of your tree with the supplied self.evaluationFunction, which defaults to scoreEvaluationFunction. AlphaBetaAgent extends MultiAgentSearchAgent, which gives access to self.depth and self.evaluationFunction. Make sure your code makes reference to these two variables where appropriate as these variables are populated in response to command line options

```
class MinimaxAgent(MultiAgentSearchAgent):.
    def getAction(self, gameState):
PACMAN = 0
        "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
        def Information_About_Agent_MaxValue(state, depth):
            if state.isWin() or state.isLose():
                return state.getScore()
            "Number of actions would be recorded in the action variable"
            actions = state.getLegalActions(PACMAN)
            "Best score would be initialized by the infinite value at first"
            Recorded_Best_Scored = float("-inf")
            score = Recorded_Best_Scored
            Happend_Action_Best_One = Directions.STOP
            for action in actions:
                score = Useful_Information_Agent(state.generateSuccessor(PACMAN, action),
depth, 1)
                "We will consider Score to find out the best recorded score"
                if score > Recorded_Best_Scored:
                    Recorded_Best_Scored = score
                    Happend_Action_Best_One = action
                    "We would find best score in all depth until we reach to the first depth"
            if depth == 0:
                return Happend_Action_Best_One
            else:
                return Recorded_Best_Scored
        "+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
        "We are considering Information of the agent such as depth , ghost and 
state"
        def Useful_Information_Agent(state, depth, ghost):
            "based on the status we will return score"
            if state.isLose() or state.isWin():
                return state.getScore()
            Available_Ghost_Next_One = ghost + 1
            '''
            Based on the state of the ghost we are trying to find out
            the number of available ghosts
            then we can find out which ghost would be closest one
            '''
            if ghost == state.getNumAgents() - 1:
                Available_Ghost_Next_One = PACMAN
            actions = state.getLegalActions(ghost)
            Recorded_Best_Scored = float("inf")
            score = Recorded_Best_Scored
            '''
            Based on the action that we recorded, we wil call evaluate function
            to evaluate and give us a score
            '''
            for action in actions:
                if Available_Ghost_Next_One == PACMAN:
                    if depth == self.depth - 1:
                        score = 
self.evaluationFunction(state.generateSuccessor(ghost, action))
                    else:
                        score = 
Information_About_Agent_MaxValue(state.generateSuccessor(ghost, action), depth + 1)
                else:
                    score = Useful_Information_Agent(state.generateSuccessor(ghost, 
action), depth, Available_Ghost_Next_One)
                if score < Recorded_Best_Scored:
                    Recorded_Best_Scored = score
            return Recorded_Best_Scored
        return Information_About_Agent_MaxValue(gameState, 0)
```

Third Task:

Test your code against at least five layouts specified in the layouts/ folder. Experiment with depths 1,2 and 3, and report in each case, the pacman score and whether pacman won or lost.

I ran all these comments to collect result of the table:

Default Evaluation function

```
python pacman.py -p AlphaBetaAgent -a depth=1 -l capsuleClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=2 -l capsuleClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=3 -l capsuleClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=1 -l contestClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=3 -l contestClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=1 -l minimaxClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=2 -l minimaxClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=3 -l minimaxClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=1 -l trappedClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=2 -l trappedClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=3 -l trappedClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=1 -l smallClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=2 -l smallClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic -k 1
```

Layout	Depth=1		Depth=2		Depth=3	
	Score	Won/Lost	Score	Won/Lost	Score	Won/Lost
capsuleClassic	-408	Lost	-1383	Lost	-629	Lost
contestClassic	-223	Lost	-326	Lost	-67	Lost
minmaxClassic	492	Won	486	Won	516	Won
trappedClassic	-506	Lost	532	Won	532	Won
smallClassic	178	Won	-282	Lost	1012	Won

Fourth Task:

Implement a new evaluation function under betterEvaluationFunction in multiagents.py. Example of a new evaluation function might be reciprocal of distance to the closest food. Re-run your experiments. Did the results improve? (Can you think of anything else which works even better?). Describe the evaluation function you implemented and the experimental results. I ran all these comments to collect result of the table:For this task, the evaluation function

```
def betterEvaluationFunction(currentGameState):
    "+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
    def Available_Nearest_Small_Food(cur_pos, food_pos):
        Distance_of_Food = []
        for food in food_pos:
            Distance_of_Food.append(util.manhattanDistance(food, cur_pos))
        return min(Distance_of_Food) if len(Distance_of_Food) > 0 else 1
    "+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
    def closest_ghost(cur_pos, ghosts):
        Distance_of_Food = []
        for food in ghosts:
            Distance_of_Food.append(util.manhattanDistance(food.getPosition(), cur_pos))
        return min(Distance_of_Food) if len(Distance_of_Food) > 0 else 1
        "+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
    def Useful_Information_Ghost(cur_pos, ghost_states, radius, scores):
        Ghost_Number_Variable = 0
        for ghost in ghost_states:
            if util.manhattanDistance(ghost.getPosition(), cur_pos) <= radius:
                scores -= 30
                Ghost_Number_Variable += 1
        return scores
        "+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
   def Useful_Information_Food(cur_pos, food_positions):
        Distance_of_Food = []
        for food in food_positions:
            Distance_of_Food.append(util.manhattanDistance(food, cur_pos))
        return sum(Distance_of_Food)
        "+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
    def Variable_Of_Food_Number(cur_pos, food):
        return len(food)
    "+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
    "+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
    Position_of_Ghost = currentGameState.getPacmanPosition()
    score = currentGameState.getScore()
    food = currentGameState.getFood().asList()
    ghosts = currentGameState.getGhostStates()
    "Based on the available nearest small foods or dots"
    "we calculates current score"
    score = score * 2 if Available_Nearest_Small_Food(Position_of_Ghost, food) <
closest_ghost(Position_of_Ghost, ghosts) + 3 else score
    "We call position of ghost and food to count new score"
    score -= .35 * Useful_Information_Food(Position_of_Ghost, food)
    return score

# Abbreviation
better = betterEvaluationFunction
"+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
"+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
class ContestAgent(MultiAgentSearchAgent):
```

evaluates states, rather than actions. Based on this evaluation function we tried to use distance of the food instead of the values themselves.

I ran the code in smallClassic layout several times:

```
python pacman.py -p AlphaBetaAgent -a depth=1 -l smallClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=2 -l smallClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=1 -l smallClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=2 -l smallClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=1 -l smallClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=2 -l smallClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=1 -l smallClassic -k 1
```

-429.0, 1356.0, -429.0, 898.0, -345.0, -429.0, 836.0, -247.0, 881.0, 858.0

The average score in 10 times was 295 in 5 times winning rate. Hence, compare to the default evaluation function, it works better.

Layout = `smallClassic`
Score= -429
Score= 1356
Score= -429
Score= 898
Score= -345
Score= -429
Score= 836
Score= -247
Score= 881
Score= 858

Extra Credit:

Change your implementation of alpha-beta to work with arbitrary number of ghosts:

To test the implementation I ran algorithm with these comments, it shows that by increasing number of ghosts the score is decreasing:

```
python pacman.py -p AlphaBetaAgent -a depth=1 -l minimaxClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=1 -l minimaxClassic -k 2
python pacman.py -p AlphaBetaAgent -a depth=1 -l minimaxClassic -k 3

python pacman.py -p AlphaBetaAgent -a depth=1 -l trappedClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=1 -l trappedClassic -k 2
python pacman.py -p AlphaBetaAgent -a depth=1 -l trappedClassic -k 3

python pacman.py -p AlphaBetaAgent -a depth=1 -l smallClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=1 -l smallClassic -k 2
python pacman.py -p AlphaBetaAgent -a depth=1 -l smallClassic -k 3

python pacman.py -p AlphaBetaAgent -a depth=1 -l contestClassic -k 1
python pacman.py -p AlphaBetaAgent -a depth=1 -l contestClassic -k 2
python pacman.py -p AlphaBetaAgent -a depth=1 -l contestClassic -k 3
```

Layout	Depth=1, #Ghost=1		Depth=1,#Ghost=2		Depth=1,#Ghost=3	
	Score	Won/Lost	Score	Won/Lost	Score	Won/Lost
contestClassic	-356	Lost	72	Lost	87	Lost
minmaxClassic	511	Won	516	Won	-497	Won
trappedClassic	518	Lost	-503	Lost	-503	Lost
smallClassic	-339	Lost	-934	Lost	-221	Lost