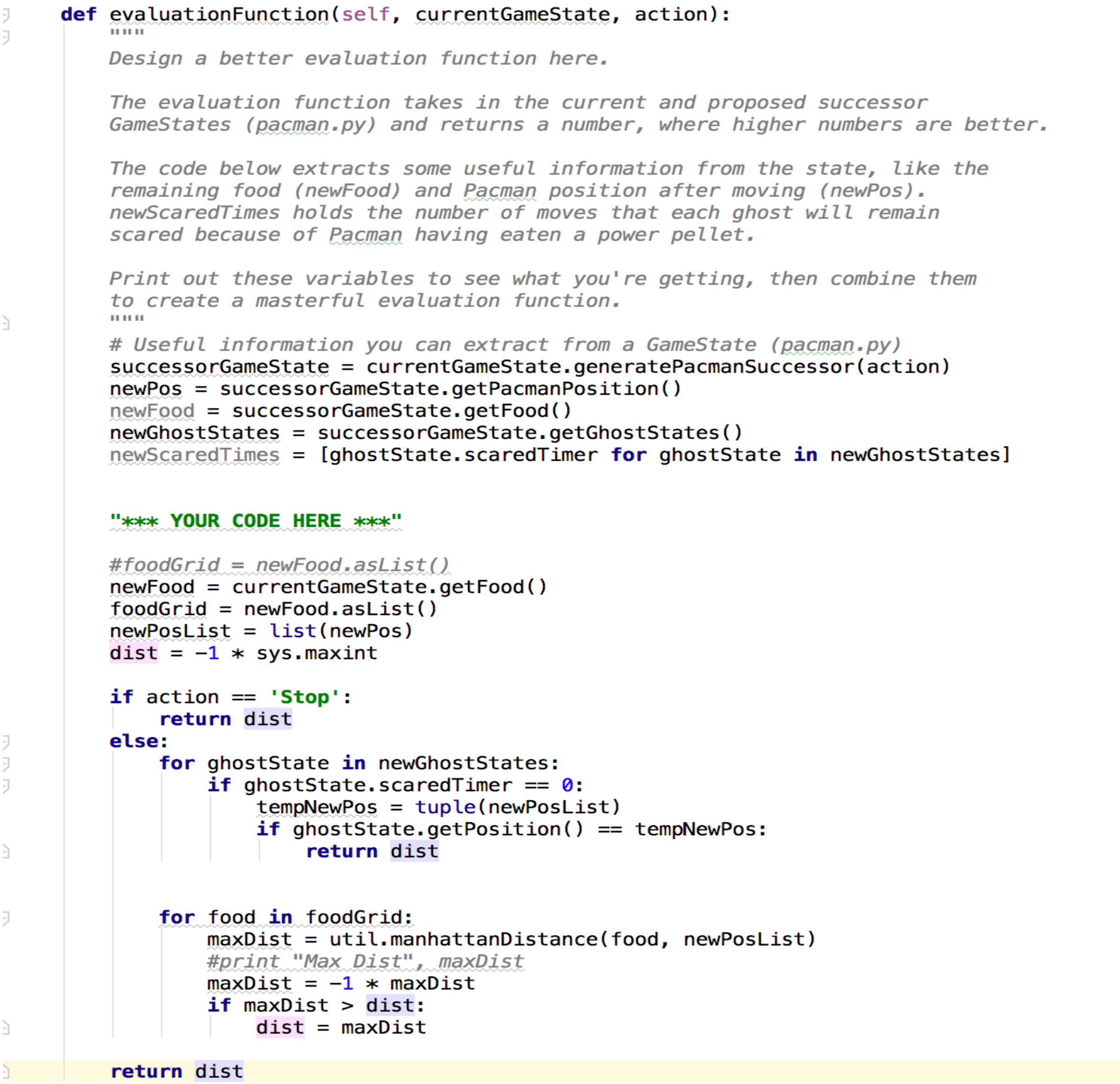
HW2 Project Details Document

1. ***Question 1*** Reflex Agent:

- **Code Details:**



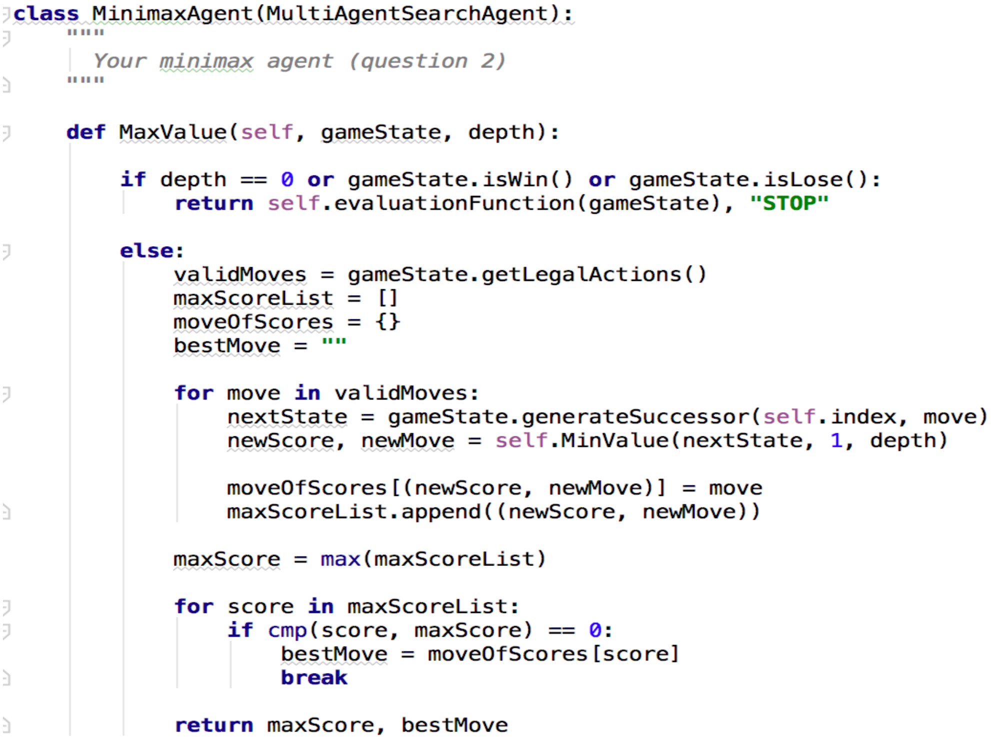
```
def evaluationFunction(self, currentGameState, action):
    """
    Design a better evaluation function here.

    The evaluation function takes in the current and proposed successor
    GameStates (pacman.py) and returns a number, where higher numbers are better.

    The code below extracts some useful information from the state, like the
    remaining food (newFood) and Pacman position after moving (newPos).
    newScaredTimes holds the number of moves that each ghost will remain
    scared because of Pacman having eaten a power pellet.

    Print out these variables to see what you're getting, then combine them
    to create a masterful evaluation function.
    """
    # Useful information you can extract from a GameState (pacman.py)
    successorGameState = currentGameState.generatePacmanSuccessor(action)
    newPos = successorGameState.getPacmanPosition()
    newFood = successorGameState.getFood()
    newGhostStates = successorGameState.getGhostStates()
    newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]


    "*** YOUR CODE HERE ***"

    #foodGrid = newFood.asList()
    newFood = currentGameState.getFood()
    foodGrid = newFood.asList()
    newPosList = list(newPos)
    dist = -1 * sys.maxint

    if action == 'Stop':
        return dist
    else:
        for ghostState in newGhostStates:
            if ghostState.scaredTimer == 0:
                tempNewPos = tuple(newPosList)
                if ghostState.getPosition() == tempNewPos:
                    return dist


        for food in foodGrid:
            maxDist = util.manhattanDistance(food, newPosList)
            #print "Max Dist", maxDist
            maxDist = -1 * maxDist
            if maxDist > dist:
                dist = maxDist

    return dist
```

- I have updated the code inside evaluationFunction() method in ReflexAgent class.
- We will generate the successor nodes/states using the current game state of Pacman.
- Find all the positions of Pacman
- For the successor nodes get all the ghost states and find the scared timer for all the ghost states.
- Using the game state gate all the food positions of Pacman and along its movement path.
- Convert the food as a food grid list.

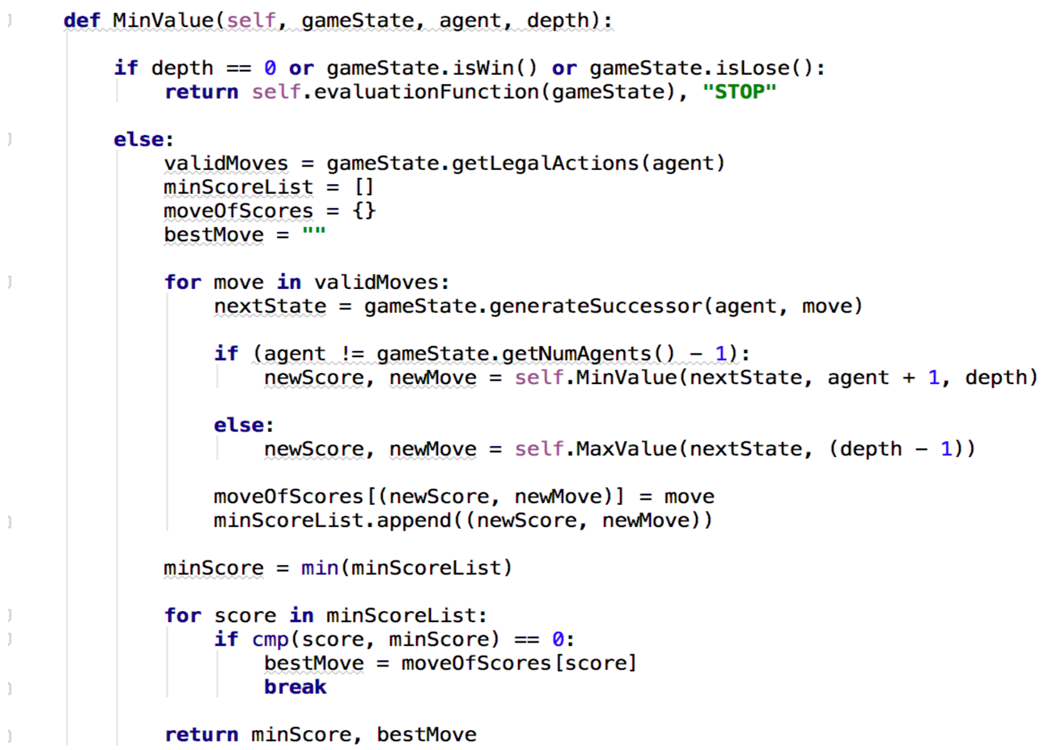
- Convert the Pacman’s positions as a list.
- Initialize the distance with minimum value
- If input action is ‘Stop’ the distance value
- Else loop over all the ghost states:
    - If scared timer of ghost is 0 and position of ghost state is same with Pacman position then in that case return the minimum distance.
    - Loop over all the foods in the food grid:
        - Calculate manhattan distance between food location and Pacman’s position
        - Make this new distance value to negative to compare this with previous distance value. So that we can get the minimum distance.
        - Compare new distance and previous distance and set the new distance accordingly
- Return the calculated distance which will be used by the game to move the Pacman accordingly.

2. Question 2 Minimax:

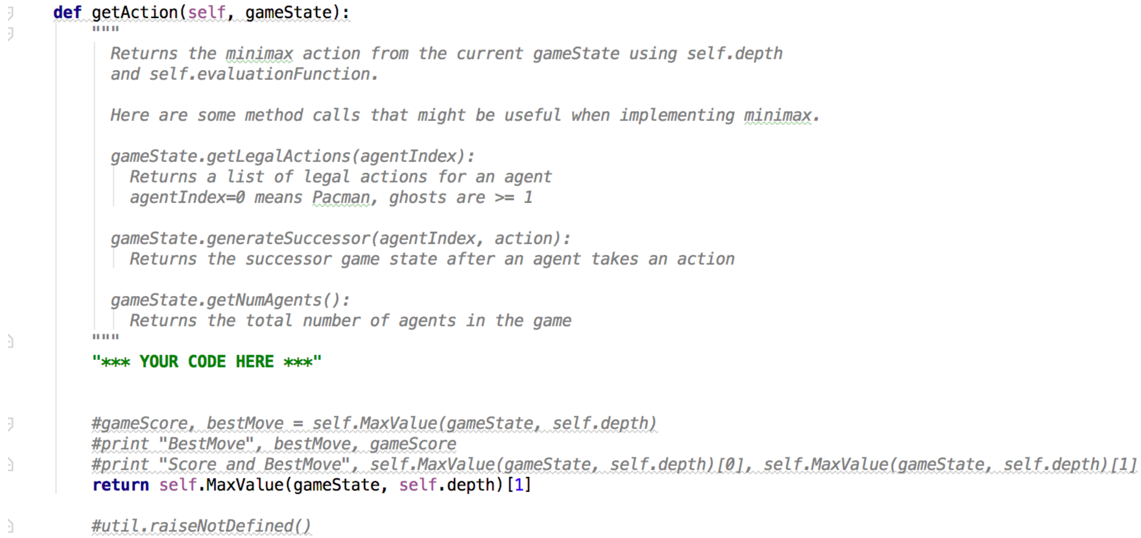
- **Code Details:**



```
class MinimaxAgent(MultiAgentSearchAgent):
    """
      Your minimax agent (question 2)
    """

    def MaxValue(self, gameState, depth):

        if depth == 0 or gameState.isWin() or gameState.isLose():
            return self.evaluationFunction(gameState), "STOP"

        else:
            validMoves = gameState.getLegalActions()
            maxScoreList = []
            moveOfScores = {}
            bestMove = ""

            for move in validMoves:
                nextState = gameState.generateSuccessor(self.index, move)
                newScore, newMove = self.MinValue(nextState, 1, depth)

                moveOfScores[(newScore, newMove)] = move
                maxScoreList.append((newScore, newMove))

            maxScore = max(maxScoreList)

            for score in maxScoreList:
                if cmp(score, maxScore) == 0:
                    bestMove = moveOfScores[score]
                    break

            return maxScore, bestMove
```



```
def MinValue(self, gameState, agent, depth):

    if depth == 0 or gameState.isWin() or gameState.isLose():
        return self.evaluationFunction(gameState), "STOP"

    else:
        validMoves = gameState.getLegalActions(agent)
        minScoreList = []
        moveOfScores = {}
        bestMove = ""

        for move in validMoves:
            nextState = gameState.generateSuccessor(agent, move)

            if (agent != gameState.getNumAgents() - 1):
                newScore, newMove = self.MinValue(nextState, agent + 1, depth)

            else:
                newScore, newMove = self.MaxValue(nextState, (depth - 1))

            moveOfScores[(newScore, newMove)] = move
            minScoreList.append((newScore, newMove))

        minScore = min(minScoreList)

        for score in minScoreList:
            if cmp(score, minScore) == 0:
                bestMove = moveOfScores[score]
                break

        return minScore, bestMove
```



```
def getAction(self, gameState):
    """
      Returns the minimax action from the current gameState using self.depth
      and self.evaluationFunction.

      Here are some method calls that might be useful when implementing minimax.

      gameState.getLegalActions(agentIndex):
        Returns a list of legal actions for an agent
        agentIndex=0 means Pacman, ghosts are >= 1

      gameState.generateSuccessor(agentIndex, action):
        Returns the successor game state after an agent takes an action

      gameState.getNumAgents():
        Returns the total number of agents in the game
    """
    "*** YOUR CODE HERE ***"


    #gameScore, bestMove = self.MaxValue(gameState, self.depth)
    #print "BestMove", bestMove, gameScore
    #print "Score and BestMove", self.MaxValue(gameState, self.depth)[0], self.MaxValue(gameState, self.depth)[1]
    return self.MaxValue(gameState, self.depth)[1]

    #util.raiseNotDefined()
```

- I have developed 3 methods MaxValue(), MinValue() and getAction() method as part of MiniMaxAgent.
- MaxValue() implementation:
    - If depth is 0 or node is terminal node then return the score value of that state and move("STOP") from that step
    - If not terminal state then get all the valid actions/moves for the current state.
    - Initialize maxScoreList list, moveOfScore dict and bestMove variables.
    - For each move in valid moves:

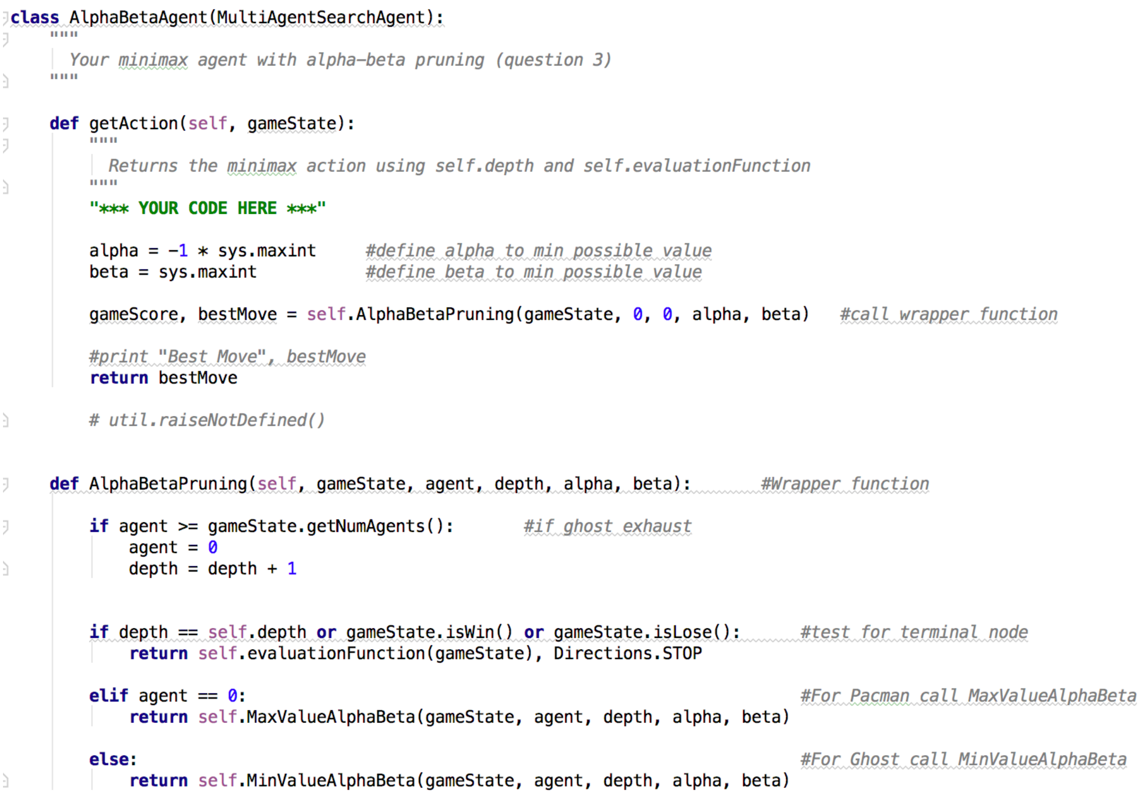
        - Generate the successor of the current state
        - Call MinValue() function with successor state, agent value and game depth value. This will return state score and best move
        - Append the retrieved values to moveOfScore dictionary as key and set the current move as the value.
        - Append the retrieved values to maxScoreList list.
    - Retrieve the max score from maxScoreList list.
    - Loop over each score of maxScoreList list:
        - Get the first repetition of max score in the list.
        - For the current score retrieve the move from the moveOfScore dictionary.
        - Assign this value to bestMove.
    - Return the max score and bestMove from this method call.

- MinValue() implementation:
    - If depth is 0 or node is terminal node then return the score value of that state and move(“STOP”) from that step
    - If not terminal state then get all the valid actions/moves for the current state.
    - Initialize minScoreList list, moveOfScore dict and bestMove variables.
    - For each move in valid moves:
        - Generate the successor of the current state
        - If more ghosts are present then call the MinValue() method recursively and pass the successor state, agent and depth value.
        - Else all the ghosts are evaluated then call the MaxValue() method and pass successor state and depth value.
        - Capture the retrieved values(score and move) in the new variables.
        - Append the retrieved values to moveOfScore dictionary as key and set the current move as the value.
        - Append the retrieved values to minScoreList list.
    - Retrieve the min score from minScoreList list.
    - Loop over each score of minScoreList list:
        - Get the first repetition of min score in the list.
        - For the current score retrieve the move from the moveOfScore dictionary.
        - Assign this value to bestMove.
    - Return the min score and bestMove from this method call.

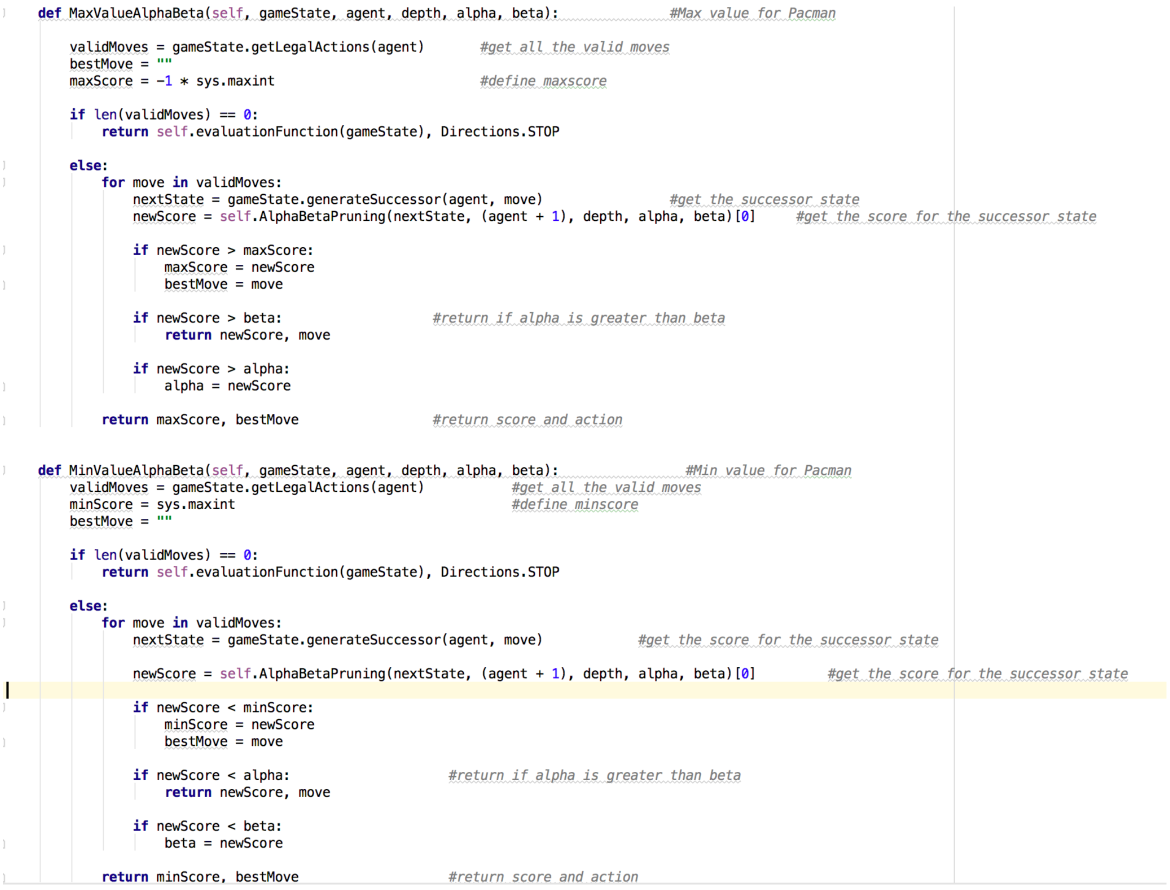
- getAction() implementation:
    - Call the MaxValue() with current game state and depth value
    - Return the best move returned by the method.

3. **Question 3** Alpha-Beta Pruning:

    - **Code Details:**



```
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
        """
        "*** YOUR CODE HERE ***"

        alpha = -1 * sys.maxint      #define alpha to min possible value
        beta = sys.maxint            #define beta to min possible value

        gameScore, bestMove = self.AlphaBetaPruning(gameState, 0, 0, alpha, beta)   #call wrapper function

        #print "Best Move", bestMove
        return bestMove

        # util.raiseNotDefined()


    def AlphaBetaPruning(self, gameState, agent, depth, alpha, beta):       #Wrapper function

        if agent >= gameState.getNumAgents():       #if ghost exhaust
            agent = 0
            depth = depth + 1


        if depth == self.depth or gameState.isWin() or gameState.isLose():      #test for terminal node
            return self.evaluationFunction(gameState), Directions.STOP

        elif agent == 0:                                                        #For Pacman call MaxValueAlphaBeta
            return self.MaxValueAlphaBeta(gameState, agent, depth, alpha, beta)

        else:                                                                   #For Ghost call MinValueAlphaBeta
            return self.MinValueAlphaBeta(gameState, agent, depth, alpha, beta)
```



```
    def MaxValueAlphaBeta(self, gameState, agent, depth, alpha, beta):              #Max value for Pacman

        validMoves = gameState.getLegalActions(agent)       #get all the valid moves
        bestMove = ""
        maxScore = -1 * sys.maxint                          #define maxscore

        if len(validMoves) == 0:
            return self.evaluationFunction(gameState), Directions.STOP

        else:
            for move in validMoves:
                nextState = gameState.generateSuccessor(agent, move)                #get the successor state
                newScore = self.AlphaBetaPruning(nextState, (agent + 1), depth, alpha, beta)[0]     #get the score for the successor state

                if newScore > maxScore:
                    maxScore = newScore
                    bestMove = move

                if newScore > beta:                     #return if alpha is greater than beta
                    return newScore, move

                if newScore > alpha:
                    alpha = newScore

            return maxScore, bestMove                   #return score and action

    def MinValueAlphaBeta(self, gameState, agent, depth, alpha, beta):                #Min value for Pacman
        validMoves = gameState.getLegalActions(agent)           #get all the valid moves
        minScore = sys.maxint                                   #define minscore
        bestMove = ""

        if len(validMoves) == 0:
            return self.evaluationFunction(gameState), Directions.STOP

        else:
            for move in validMoves:
                nextState = gameState.generateSuccessor(agent, move)            #get the score for the successor state

                newScore = self.AlphaBetaPruning(nextState, (agent + 1), depth, alpha, beta)[0]         #get the score for the successor state

                if newScore < minScore:
                    minScore = newScore
                    bestMove = move

                if newScore < alpha:                    #return if alpha is greater than beta
                    return newScore, move

                if newScore < beta:
                    beta = newScore

            return minScore, bestMove                   #return score and action
```

- I have developed 4 methods AlphaBetaPruning(), MaxValueAlphaBeta(), MinValueAlphaBeta() and getAction() method as part of AlphaBetaAgent.

- AlphaBetaPruning() implementation:
    - Check for agent value is greater than the gamestate agents numbers. This will check if the ghosts are exhausted or not.
    - If ghost exhausted then set agent value to 0
    - Increase depth by 1
    - If depth is the gamestate depth or state is won or lost then this is the terminal state.
    - If no ghost is present then pacman moves and find the max value
    - Else the ghosts moves and find the min values

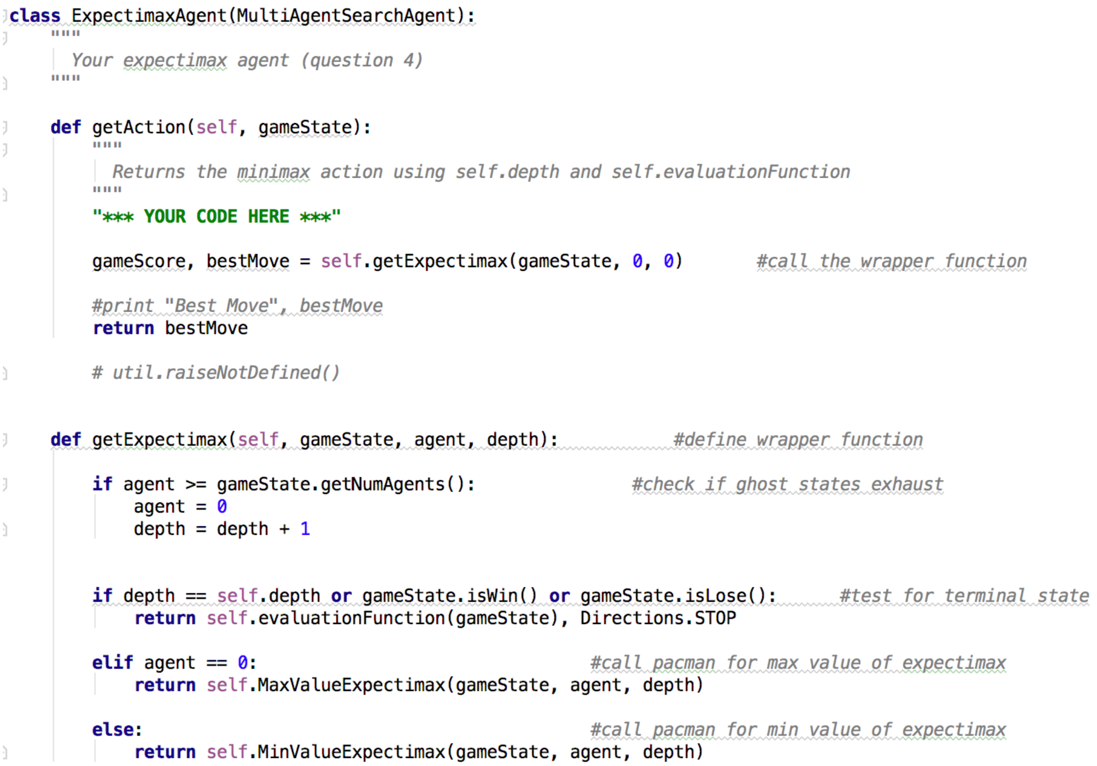
- MaxValueAlphaBeta() implementation:
    - Get all the valid actions/moves for the current state.
    - Initialize variables bestMove and maxScore which is highest minimum value.
    - If no valid moves are present then return the value of current state
    - For each move in valid moves:
        - Generate the successor of the current state
        - Call AlphaBetaPruning() function with successor state, agent value game depth value, alpha and beta value. This will return state score
        - If new score is greater than maxscore then:
            - Set the new score value as maxscore.
            - Get the best move from the valid moves.
        - If alpha > beta then return alpha and best move from this method call.
        - Set the newscore to alpha value
    - Return the max score and bestMove from this method call.

- MinValueAlphaBeta() implementation:
    - Get all the valid actions/moves for the current state.
    - Initialize variables minScore which is highest maximum value and bestmove.
    - If no valid moves are present then return the value of current state
    - For each move in valid moves:
        - Generate the successor of the current state
        - Call AlphaBetaPruning() function with successor state, agent value game depth value, alpha and beta value. This will return state score.
        - Capture the retrieved values(score) in the new variable.
        - If new score is less than minscore then set this to new minscore and set the best move to current move
        - If new score is less than alpha then return alpha and move
        - If new score is less than beta then set new score to beta
    - Return the min score and best move from this method call.

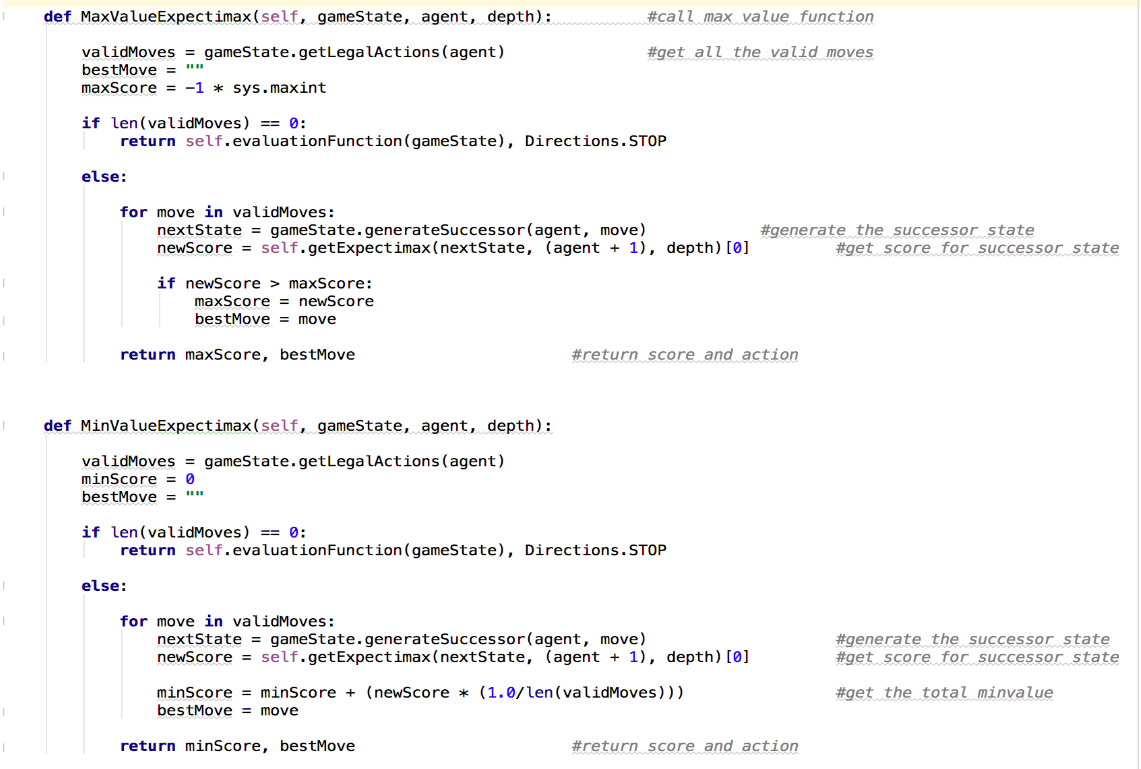
- getAction() implementation:
    - initialize alpha as the maximum negative number and beta as the maximum positive number
    - Call the AlphaBetaPruning() with current game state, depth value, alpha and beta values.
    - Return the best move returned by the method call.

4. Question 4 Expectimax:

- **Code Details:**



```
class ExpectimaxAgent(MultiAgentSearchAgent):
    """
      Your expectimax agent (question 4)
    """

    def getAction(self, gameState):
        """
          Returns the minimax action using self.depth and self.evaluationFunction
        """
        "*** YOUR CODE HERE ***"

        gameScore, bestMove = self.getExpectimax(gameState, 0, 0)       #call the wrapper function

        #print "Best Move", bestMove
        return bestMove

        # util.raiseNotDefined()


    def getExpectimax(self, gameState, agent, depth):           #define wrapper function

        if agent >= gameState.getNumAgents():                #check if ghost states exhaust
            agent = 0
            depth = depth + 1


        if depth == self.depth or gameState.isWin() or gameState.isLose():       #test for terminal state
            return self.evaluationFunction(gameState), Directions.STOP

        elif agent == 0:                                    #call pacman for max value of expectimax
            return self.MaxValueExpectimax(gameState, agent, depth)

        else:                                               #call pacman for min value of expectimax
            return self.MinValueExpectimax(gameState, agent, depth)
```



```
    def MaxValueExpectimax(self, gameState, agent, depth):          #call max value function

        validMoves = gameState.getLegalActions(agent)              #get all the valid moves
        bestMove = ""
        maxScore = -1 * sys.maxint

        if len(validMoves) == 0:
            return self.evaluationFunction(gameState), Directions.STOP

        else:

            for move in validMoves:
                nextState = gameState.generateSuccessor(agent, move)          #generate the successor state
                newScore = self.getExpectimax(nextState, (agent + 1), depth)[0]         #get score for successor state

                if newScore > maxScore:
                    maxScore = newScore
                    bestMove = move

            return maxScore, bestMove                          #return score and action


    def MinValueExpectimax(self, gameState, agent, depth):

        validMoves = gameState.getLegalActions(agent)
        minScore = 0
        bestMove = ""

        if len(validMoves) == 0:
            return self.evaluationFunction(gameState), Directions.STOP

        else:

            for move in validMoves:
                nextState = gameState.generateSuccessor(agent, move)                  #generate the successor state
                newScore = self.getExpectimax(nextState, (agent + 1), depth)[0]         #get score for successor state

                minScore = minScore + (newScore * (1.0/len(validMoves)))              #get the total minvalue
                bestMove = move

            return minScore, bestMove                          #return score and action
```

- I have developed 4 methods getExpectimax(),MaxValueExpectimax(),MinValueExpectimax() and getAction() method as part of AlphaBetaAgent.

- getExpectimax() implementation:
    - Check for agent value is greater than the gamestate agents numbers. This will check if the ghosts are exhausted or not.
    - If ghost exhausted then set agent value to 0
    - Increase depth by 1
    - If depth is the gamestate depth or state is won or lost then this is the terminal state.
    - If no ghost is present then pacman moves and find the max value
    - Else the ghosts move and find the min values

- MaxValueExpectimax() implementation:
    - Get all the valid actions/moves for the current state.
    - Initialize variables bestMove and maxScore which is highest minimum value.
    - If no valid moves are present then return the value of current state
    - For each move in valid moves:
        - Generate the successor of the current state
        - Call getExpectimax() function with successor state, agent value game and depth value. This will return state score
        - If new score is greater than maxscore then:
            - Set the new score value as maxscore.
            - Get the best move from the valid moves.
    - Return the max score and bestMove from this method call.

- MinValueExpectimax() implementation:
    - Get all the valid actions/moves for the current state.
    - Initialize variables minScore which is 0 value and bestmove.
    - If no valid moves are present then return the value of current state
    - For each move in valid moves:
        - Generate the successor of the current state
        - Call getExpectimax() function with successor state, agent value game and depth value. This will return state score.
        - Capture the retrieved values(score) in the new variable.
        - Calculate minscore as oldminscore value + (1.0/length of valid moves)
        - set the best move to current move
    - Return the min score and best move from this method call.

- getAction() implementation:
    - Call the getExpectimax() with current game state, agent and depth value
    - Return the best move returned by the method call.