שם: [illegible] קיבנר
ת.ז: 208271775

שאלה 1.

הגישה שנבחרה ReflexAgent:

הגישה שבה בחרתי:

```
# ghost cost :
nearestGhost = sys.maxsize  # Initialized to maximum value to perform "reflexive alpha beta"
for gState in newGhostStates:
    ghostX, ghostY = gState.getPosition()
    # I have no idea why pulling a "location" of a
    #       "ghost" returns a tuple rather than a state
    ghostX = int(ghostX)
    ghostY = int(ghostY)
    if gState.scaredTimer == 0:  # Can kill
        nearestGhost = min(nearestGhost, manhattanDistance((ghostX, ghostY), newPos))
ghostCost = -(1.0 / (nearestGhost + 0.1))  # Balance to achieve a worthy result
# 0.1 Is for not dividing by 0
```

הגישה שלי היא:

שם: [illegible] קיימר
ת.ז: 208271775

תרגיל בית 1.

השאלה הראשונה:

```
# Food cost:
    foodAsList = newFood.asList()
    # Using Queue from util.py
    distToFoodQueue = PriorityQueueWithFunction(lambda x: manhattanDistance(x, newPos))
    for food in foodAsList:
        distToFoodQueue.push(food)
    if distToFoodQueue.isEmpty():  # That means no more food
        nearestFood = 0
    else:
        # The distance to the closest because it is a priority queue with a function
        nearestFood = manhattanDistance(distToFoodQueue.pop(), newPos)
    foodCost = (1.0 / (nearestFood + 0.1))  # Balance to achieve a worthy result
    # 0.1 Is for not dividing by 0
```

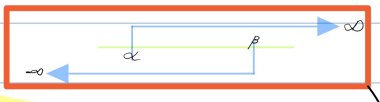
```
return successorGameState.getScore() + foodCost + ghostCost
```

```
# Start for the maximum agent
return self.maxStep(gameState, 1)
```

```
# Checks if the situation in our agent is inactive
def isInactive(self, gameState):
    return gameState.isWin() or gameState.isLose() or self.depth == 0

def maxStep(self, gameState, depth):
    if self.isInactive(gameState):  # Checks if the situation in our agent is inactive
        return self.evaluationFunction(gameState)
    maxVal = -sys.maxsize  # Initializes the maximum value (so-called beta)
    bestStep = Directions.STOP  # Initialized to stop by class directions in game.py
    for action in gameState.getLegalActions(0):
        # Performs on each successor a minimum step according to the algorithm
        successor = gameState.generateSuccessor(0, action)
        actionVal = self.minStep(successor, depth, 1)
        if actionVal > maxVal:  # He's better
            maxVal = actionVal
            bestStep = action
    if depth > 1:
        return maxVal  # A value will be returned in favor of a success calculation
    return bestStep  # An action will be returned, since this is the stage for execution
```

שם: יעל קינר
ת.ז: 208271778

חלק ב.



```
def minStep(self, gameState, depth, agentIndex):
    if self.isInactive(gameState):  # Checks if the situation in our agent is inactive
        return self.evaluationFunction(gameState)
    minVal = sys.maxsize  # Initializes the minimum value (so-called alpha)
    legalActions = gameState.getLegalActions(agentIndex)
    successors = [gameState.generateSuccessor(agentIndex, action) for action in legalActions]
    if agentIndex == gameState.getNumAgents() - 1:
        if depth < self.depth:  # It will be the turn of the maximum agent
            for successor in successors:
                minVal = min(minVal, self.maxStep(successor, depth + 1))
        else: # It will be the turn of the minimum agent
            for successor in successors:
                minVal = min(minVal, self.evaluationFunction(successor))
    else: # Last agent
        for successor in successors:
            minVal = min(minVal, self.minStep(successor, depth, agentIndex + 1))
    return minVal # A value will be returned in favor of a success calculation
```

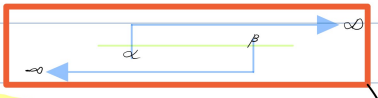
שלב ג'

סיכום ביצוע אלגוריתם "alphabeta":

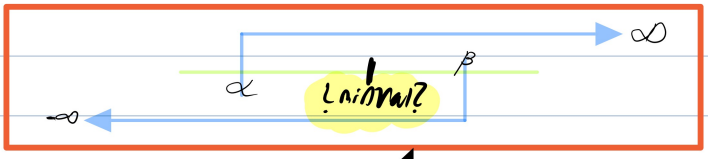
```
# Start for the maximum agent
return self.maxStep(gameState, 1, -sys.maxsize, sys.maxsize)

# Checks if the situation in our agent is inactive
def isInactive(self, gameState):
    return gameState.isWin() or gameState.isLose() or self.depth == 0

def maxStep(self, gameState, depth, alpha, beta):
    if self.isInactive(gameState): # Checks if the situation in our agent is inactive
        return self.evaluationFunction(gameState)
    maxVal = -sys.maxsize # Initializes the maximum value (so-called beta)
    bestStep = Directions.STOP # Initialized to stop by class directions in game.py
    for action in gameState.getLegalActions(0):
        # Performs on each successor a minimum step according to the algorithm
        successor = gameState.generateSuccessor(0, action)
        pruningValue = self.minStep(successor, depth, 1, alpha, beta)
        if pruningValue > maxVal: # He's better
            maxVal = pruningValue
            bestStep = action
        if maxVal > beta: # Puring will act
            return maxVal
        alpha = max(alpha, maxVal)
    if depth > 1:
        return maxVal # A value will be returned in favor of a success calculation
    return bestStep # An action will be returned, since this is the stage for execution
```

שם: יעל קיינר
ת.ז: 327178778

משימה מספר 3



```
def minStep(self, gameState, depth, agentIndex, alpha, beta):
    if self.isInactive(gameState):  # Checks if the situation in our agent is inactive
        return self.evaluationFunction(gameState)
    minVal = sys.maxsize  # Initializes the minimum value (so-called alpha)
    for action in gameState.getLegalActions(agentIndex):
        successor = gameState.generateSuccessor(agentIndex, action)
        if agentIndex == gameState.getNumAgents() - 1:
            if depth < self.depth: # It will be the turn of the maximum agent
                pruningValue = self.maxStep(successor, depth + 1, alpha, beta)
            else:  # It will be the turn of the minimum agent
                pruningValue = self.evaluationFunction(successor)
        else:  # Last agent
            pruningValue = self.minStep(successor, depth, agentIndex + 1, alpha, beta)
        # puring check
        if pruningValue < minVal:
            minVal = pruningValue
        if minVal < alpha:
            return minVal
        beta = min(beta, minVal)
    return minVal # A value will be returned in favor of a success calculation
```



שם: דניאל קייבנר
ת.ז: 208271775

שאלה 4.

סיכום ב-"exeptimax":

ב- getAction שבה ישר לסוכן מקסימום

```
return self.maxStep(gameState, 1)

# Checks if the situation in our agent is inactive
def isInactive(self, gameState):
    return gameState.isWin() or gameState.isLose() or self.depth == 0
```

העומק בגישה

פקמן חי/מת

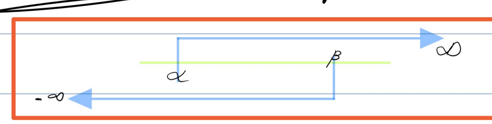
```
def maxStep(self, gameState, depth):
    if self.isInactive(gameState):  # Checks if the situation in our agent is inactive
        return self.evaluationFunction(gameState)
    maxVal = -sys.maxsize  # Initializes the maximum value (so-called alpha)
    bestStep = Directions.STOP  # Initialized to stop by class directions in game.py
    for action in gameState.getLegalActions(0):
        # Performs on each successor a probabilistic minimum step according to the algorithm
        successor = gameState.generateSuccessor(0, action)
        probabilisticPuringValue = self.minStep(successor, depth, 1)
        # Pruning test in relation to the probabilistic value
        if probabilisticPuringValue > maxVal:
            maxVal = probabilisticPuringValue
            bestStep = action
    if depth > 1:
        return maxVal  # A value will be returned in favor of a success calculation
    return bestStep  # An action will be returned, since this is the stage for execution
```

יעריך לפי הstate עכשיו

ערך הכי [illegible] לעבור אותו שאלה בכך

probabilistic

עדכון הsuccessor



עם הערך

ישנם מצבים להערכה

צריך להחזיר מה הפעולה לביצוע

המשך בעמ' הבא

שם: רעות קיינר
ת.ז: 207271775

משימה 4.

```
def minStep(self, gameState, depth, agentIndex):
    if self.isInactive(gameState):  # Checks if the situation in our agent is inactive
        return self.evaluationFunction(gameState)
    legalActions = gameState.getLegalActions(agentIndex)
    successors = [gameState.generateSuccessor(agentIndex, action) for action in legalActions]
    probabilityPuringValue = 0
    probabilityForMove = 1.0 / len(legalActions)
    if agentIndex == gameState.getNumAgents() - 1:  # Last Agent
        if depth < self.depth:
            for successor in successors: # check Using the max step
                probabilityPuringValue += probabilityForMove * self.maxStep(successor, depth + 1)
        else:  # good time to evaluation Function
            for successor in successors:  # check Using the evaluationFunction
                probabilityPuringValue += probabilityForMove * self.evaluationFunction(successor)
    else:  # not the last agent
        for successor in successors: # check Using the min step
            probabilityPuringValue += probabilityForMove * self.minStep(successor, depth, agentIndex + 1
    return probabilityPuringValue
```

שם: [illegible] קיינר
ת.ז: 208271775

שלב 5

betterEvaluationFunction

```
# State score
currentPosition = currentGameState.getPacmanPosition()
# we will change the power of the state score power with a value of 1 to a power with a value of 1.5
stateScore = 37 * currentGameState.getScore()

# Capsules score
capsQueue = PriorityQueueWithFunction(lambda x: manhattanDistance(currentPosition, x))
capsCounter = Counter()
for cap in currentGameState.data.capsules:
    capsCounter[str(cap)] = manhattanDistance(currentPosition, cap)
    capsQueue.push(cap)
if capsQueue.isEmpty():
    contraPositiveNearestCapsDist = 100
else:
    # For giving a positive result
    contraPositiveNearestCapsDist = 100 - manhattanDistance(currentPosition, capsQueue.pop())
# using counter from util
capsTotalDistant = capsCounter.totalCount()
# For the State evaluate function to be good A change was made in the "factor" of the parameters
capsScore = -8 * capsTotalDistant + 10 * contraPositiveNearestCapsDist
```

```
    # Ghost score
    ghostStates = currentGameState.getGhostStates()
ghostQueue = PriorityQueueWithFunction(lambda x: manhattanDistance(currentPosition, x.getPosition()))
    ghostScaredCounter = Counter()
    for ghost in ghostStates:
        ghostQueue.push(ghost)
        ghostScaredCounter[str(ghost)] = ghost.scaredTimer
    nearestGhostDist = manhattanDistance(currentPosition, ghostQueue.pop().getPosition())
    # For the State evaluate function to be good A change was made in the "factor" of the parameters,
    # and factor multiplying between number of ghosts and number of capsules
    ghostScore = 3 * nearestGhostDist + 30 * ghostScaredCounter.totalCount() * capsQueue.count

    # Food score
    foods = currentGameState.getFood()
    foodsPos = foods.asList()
    numFood = currentGameState.getNumFood()
    foodQueue = PriorityQueueWithFunction(lambda x: manhattanDistance(currentPosition, x))
    for food in foodsPos:
        foodQueue.push(food)
    if foodQueue.isEmpty():
        distClosestFood = 0
    else:
        nearestFood = foodQueue.pop()
        distClosestFood = manhattanDistance(currentPosition, nearestFood)
    if numFood < nearestGhostDist:
        numFood = nearestGhostDist
    # For the State evaluate function to be good A change was made in the "factor" of the parameters
    foodScore = - 9 * distClosestFood - 10 * numFood

    # Total score
    return ghostScore + capsScore + stateScore + foodScore
```