1. Pacman doesn't perform as well in small classic likely because the map is more narrow and Pacman gets cornered easily since its strategy is wait for the ghost to come and run the other way. With not much space to turn and two ghosts, Pacman has no where to go.

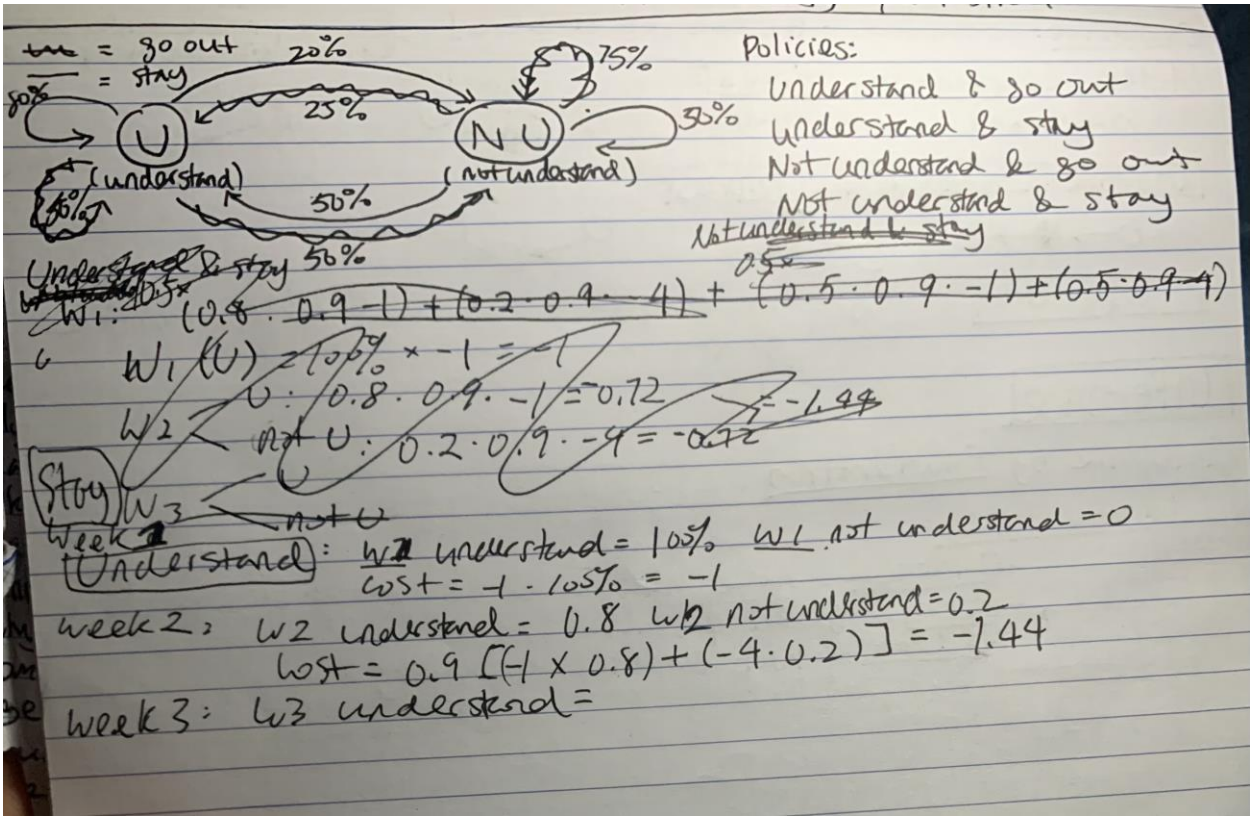
```
D:\USC - CSCI\Project 2\Multiagent\multiagent>python pacman.py -p MinimaxAgent -l smallClassic -a depth=2 -n 10 -
-frameTime 0
Pacman died! Score: -378
Pacman died! Score: 184
Pacman died! Score: -650
Pacman emerges victorious! Score: 1191
Pacman emerges victorious! Score: 1183
Pacman died! Score: -256
Pacman died! Score: -386
Pacman died! Score: -270
Pacman died! Score: -487
Pacman died! Score: -187
Average Score: -5.6
Scores:        -378.0, 184.0, -650.0, 1191.0, 1183.0, -256.0, -386.0, -270.0, -487.0, -187.0
Win Rate:      2/10 (0.20)
Record:        Loss, Loss, Loss, Win, Win, Loss, Loss, Loss, Loss, Loss
```

```
D:\USC - CSCI\Project 2\Multiagent\multiagent>python pacman.py -p AlphaBetaAgent -l smallClassic -a depth=2 -n 10
 --frameTime 0
Pacman died! Score: -303
Pacman died! Score: -158
Pacman died! Score: -160
Pacman died! Score: -298
Pacman died! Score: -206
Pacman died! Score: -335
Pacman died! Score: -191
Pacman died! Score: -202
Pacman died! Score: -350
Pacman died! Score: -311
Average Score: -251.4
Scores:        -303.0, -158.0, -160.0, -298.0, -206.0, -335.0, -191.0, -202.0, -350.0, -311.0
Win Rate:      0/10 (0.00)
Record:        Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss
```

```
D:\USC - CSCI\Project 2\Multiagent\multiagent>python pacman.py -p ExpectimaxAgent -l smallClassic -a depth=2 -n 1
0 --frameTime 0
Pacman died! Score: -501
Pacman died! Score: -323
Pacman died! Score: -238
Pacman emerges victorious! Score: 1159
Pacman died! Score: 8
Pacman emerges victorious! Score: 718
Pacman died! Score: -185
Pacman died! Score: 35
Pacman died! Score: -267
Pacman emerges victorious! Score: 1139
Average Score: 154.5
Scores:        -501.0, -323.0, -238.0, 1159.0, 8.0, 718.0, -185.0, 35.0, -267.0, 1139.0
Win Rate:      3/10 (0.30)
Record:        Loss, Loss, Loss, Win, Loss, Win, Loss, Loss, Loss, Win
```

2. I wanted the Pacman to be hungry enough to want to get close to the pellets, prioritize the capsules, be less scared of the ghosts after he ate capsules, and rewarded for winning and punished for losing.

```python
def betterEvaluationFunction(currentGameState):
    """
    Your extreme ghost-hunting, pellet-nabbing, food-gobbling, unstoppable
    evaluation function (question 4).

    DESCRIPTION: Combining win/lose/neutral, pellets left, distance to closest pellet, distance to closest ghosts (which is weighted depending on the ghost
    """
    "*** YOUR CODE HERE ***"

    currentPosition = currentGameState.getPacmanPosition()

    #Getting win lose score
    score = 0
    if currentGameState.isWin(): score = 999999
    elif currentGameState.isLose(): score = -999999

    #Getting number of food left
    foodLeft = currentGameState.getNumFood()
    foodWeight = 100000
    if foodLeft == 0:
        foodValue = 100
    else:
        foodValue = foodWeight / foodLeft

    #Getting the closest food distance
    #iterate through the grid
    grid = currentGameState.getFood()
    rowNum = grid.height
    colNum = grid.width
    foodList = [(x, y) for x in range(colNum) for y in range(rowNum) if grid[x][y] == True ]
    #Calculating shortest distance
    closestFoodDistance = 99999999
    for food in foodList:
        if manhattanDistance(food, currentPosition) < closestFoodDistance: closestFoodDistance = manhattanDistance(food, currentPosition)
    closestFoodDistanceWeight = 10
    closestFoodDistanceValue = 0
    closestFoodDistanceValue = 0
    if closestFoodDistance != 0:
        closestFoodDistanceValue = closestFoodDistanceWeight / closestFoodDistance

    #Getting distance to closest capsule
    capsules = currentGameState.getCapsules()
    remainCapsuleWeight = 10000
    remainCapsuleValue = 0
    closestCapsule = 0
    #If more capsules left
    if capsules:
        #More capsule left, lower score
        remainCapsuleValue -= remainCapsuleWeight * len(capsules)
        closestCapsule = 99999999
        for capsule in capsules:
            if (manhattanDistance(currentPosition, capsule) < closestCapsule):
                closestCapsule = manhattanDistance(currentPosition, capsule)
    #Converting capsule distance to value
    capsuleValue = 1
    capsuleWeight = 100

    if closestCapsule == 0:
        capsuleValue *= capsuleWeight
    else:
        capsuleValue = capsuleWeight / closestCapsule

    #Getting ghosts proximity
    ghostPositions = currentGameState.getGhostPositions()
    manDistances = [manhattanDistance(currentPosition, ghostPosition) for ghostPosition in ghostPositions]

    #Getting ghost scared time (controlls the weight of ghost proximity)
    ghostStates = currentGameState.getGhostStates()
    scaredTimes = [ghostState.scaredTimer for ghostState in ghostStates]

    scaredTimes = [ghostState.scaredTimer for ghostState in ghostStates]

    #Converting ghost distance to value
    scaredWeight = 0.6
    distanceValues = []
    for (manDistance, scaredTime) in zip(manDistances, scaredTimes):
        if scaredTime != 0:
            distanceValues.append(manDistance * scaredWeight / scaredTime)
        else: distanceValues.append(manDistance)
    distanceValue = sum(distanceValues)

    return score + foodValue + capsuleValue + distanceValue + remainCapsuleValue + closestFoodDistanceValue

    util.raiseNotDefined()
```



Policies:
understand & go out
understand & stay
Not understand & go out
Not understand & stay

Understand & stay
W_1: $(0.8 \cdot 0.9 \cdot -1) + (0.2 \cdot 0.9 \cdot -4) + (0.5 \cdot 0.9 \cdot -1) + (0.5 \cdot 0.9 \cdot -4)$

W_1(U) 100% × −1 = −1
W_2 U: $0.8 \cdot 0.9 \cdot -1 = 0.72$
not U: $0.2 \cdot 0.9 \cdot -4 = -0.72$

Stay
W_3
U
not U

Week 1
Understand: W1 understand = 100% W1 not understand = 0
cost = −1 · 100% = −1

week 2: W2 understand = 0.8 W2 not understand = 0.2
cost = 0.9 [(−1 × 0.8) + (−4 · 0.2)] = −1.44

week 3: W3 understand =