Introduction

To win games, pacman has to be able to eat all the food. For this coursework, "winning" just means getting the environment to report a win. Also , Pacman needs to survive from ghosts. So here I build an Agent which can eat all the food without a ghost, also when ghosts appear, it should be able to avoid colliding with a ghost.

Description

If pacman needs to eat all the food from the grid , it should find a path or make a decision to choose what is the next step for pacman to eat the food. Therefore ,we should define a function to decide what is the next step for the pacman. First, getAction() takes a gameState and returns Directions from the set {North, South, West, East, Stop}.Then it will choose one of the best actions according to the evaluation function. Actually, evaluation Function is the heart for this Agent, because it will access every input score.

```
class GreedyAgent(Agent):
    def __init__(self, evalFn="scoreEvaluation"):
        self.evaluationFunction = util.lookup(evalFn, globals())
        assert self.evaluationFunction != None
```

And this evaluationFunction referenced from GreedyAgent(pacmanAgents.py)

```
def getAction(self, gameState):
    legalMoves = gameState.getLegalActions()
    scores = [self.evaluationFunction(gameState, action) for action in legalMoves]
    bestScore = max(scores)
    bestIndices = [index for index in range(len(scores)) if scores[index] == bestScore]
    chosenIndex = random.choice(bestIndices)
    return legalMoves[chosenIndex]
```

Now we need to design the evaluation Function, The evaluation function takes in the current and proposed successor GameStates (pacman.py) and returns a number(score) , where higher numbers are better.

```
def evaluationFunction(self, currentGameState, action):
    successorGameState = currentGameState.generatePacmanSuccessor(action)
    newPos = successorGameState.getPacmanPosition()
    newFood = successorGameState.getFood()
    curFood = currentGameState.getFood()
    newGhostStates = successorGameState.getGhostStates()
    newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]

    food_left = sum(int(j) for i in newFood for j in i)
```

print food_left

It will extracts all the useful information from the state, like the remaining food (newFood) and Pacman position(newPos) after moving . And it will get the Ghosts position(newGhostStates).Also, newScaredTimes holds the number of moves that each ghost will continue scared because of Pacman having eaten the capsule. Then, I will calculate the number of left food after pacman eat food every time ,and print it out on the terminal.

```
ghost_distances = []
for gs in newGhostStates:
    ghost_distances += [util.manhattanDistance(gs.getPosition(),newPos)]
```

```
for foodpos in foodList:
    food_distences += [manhattanDistance(newPos,foodpos)]
```

After that, according to the ghosts position and food position, this function simply calculate the Manhattan distance from current pacman position to food and ghosts.
First calculate the distance between ghost and pacman , and store this distance into a list[ ](ghost_distances).Then calculate the distance from all the food to pacman position(food_distences).

```
inverse_food_distences=0;
if len(food_distences)>0 and min_val > 0:
    inverse_food_distences = 1.0 / min_val
#Considering the distance between ghosts to pacman and food to pacman,
#And the ghost score is higher.
currscore += min_val*(inverse_food_distences**4)
#Collect the current gamestate score
currscore+=successorGameState.getScore()
#The score would be higher if pacman eat food now
if newPos in curfoodList:
    currscore = currscore * 1.1
return currscore
```

The final step is comparing the score(currscore). Suppose we can collect the minimum distance between food and pacman(),and the ghosts score is higher which will be returned by the function .After that, get the current Gamestate number, what if pacman eat food now then the currscore will be higher.

manhattanDistance(util.py)

```
def manhattanDistance( xy1, xy2 ):
    "Returns the Manhattan distance between points xy1 and xy2"
    return abs( xy1[0] - xy2[0] ) + abs( xy1[1] - xy2[1] )
```

scoreEvalution(pacmanAgent.py)

```
3
4  def scoreEvaluationFunction(currentGameState):
5      return currentGameState.getScore()
6
```

Results

1.Running code with the command:

python pacman.py -q -p PartialAgent -l mediumClassicNoGhosts

We can see that when pacman in the no ghosts grid, it can win the game undoubtedly.

```
[parallels@centos-7 pacman-cw1]$ python pacman.py -q -p PartialAgent -l mediumClassicNoGhosts
```

```
Pacman emerges victorious! Score: 937
Average Score: 937.0
Scores:        937.0
Win Rate:      1/1 (1.00)
Record:        Win
[parallels@centos-7 pacman-cw1]$
```

2.Running code with the command:

python pacman.py -q -n 5 -p PartialAgent -l mediumClassic

When pacman in the grid there has ghosts, it could win the game but not every time.

```
[parallels@centos-7 pacman-cw1]$ python pacman.py -q -n 5 -p PartialAgent -l mediumClassic
```

```
Pacman emerges victorious! Score: 1005
Average Score: 135.4
Scores:        -276.0, 184.0, 85.0, -321.0, 1005.0
Win Rate:      1/5 (0.20)
Record:        Loss, Loss, Loss, Loss, Win
```

3.Running code with the command:

python pacman.py -n 50 -p PartialAgent -l mediumClassic

Get a statistically significant number of runs. We can find that the win rate is 6/50 , but this could be changed. The evaluationFunction will return the score to Agent, and this score is influenced by the distance between ghost to pacman and food to pacman, and Agent will choose the higher score to take an action.Therefore , if we set different value to currscore, the win rate will be different either.

```
Pacman emerges victorious! Score: 1435
Average Score: 196.3
Scores:        146.0, 14.0, 51.0, -392.0, -419.0, 100.0, 388.0, 179.0, -164.0, -220.0, -40
1.0, 1494.0, 199.0, -401.0, 529.0, 901.0, 370.0, -55.0, -70.0, 1397.0, 36.0, -356.0, 207.0
, -338.0, -297.0, 3.0, -271.0, 167.0, 75.0, -139.0, -2.0, -20.0, 1596.0, -356.0, 414.0, -8
.0, -123.0, 154.0, 256.0, -428.0, -351.0, -188.0, 76.0, 569.0, 965.0, 576.0, 2059.0, 687.0
, -229.0, 1435.0
Win Rate:      6/50 (0.12)
Record:        Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Win, Loss
, Loss, Loss, Loss, Loss, Loss, Loss, Win, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss,
 Loss, Loss, Loss, Loss, Win, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss,
Loss, Loss, Loss, Win, Win, Loss, Win
```

4.Running on a different grid:

Layout	Score	Record
boxSearch	-	-
bigCorners	-482	Loss
capsuleClassic	-397	Loss
contestClassic	-286	Loss
greedySearch	565	Win
mediumScaryMaze	-562	Loss
testClassic	506	Win
openClassic	681	Win
smallClassic	-91	Loss
trickyClassic	-99	Loss