```python
import random


class QLearning:
    def __init__(self):

        self.gamma=0.9
        self.epsilon=0.9
        self.learningRate = 0.1
        self.gridWorld=[[[0.0, 0.0, 0.0, 0.0] for row in range(0,15)] for column in range(0,15)]

        self.maxEpisodes=1000

    def SelectEpsilonGreedyNeighbor(self, row, column):
        maxQ=-99999.99;
        maxQMove=[]
        possibleMoves=[]

        if row>0:
            possibleMoves.append(0)
            if maxQ<self.gridWorld[row][column][0]:
                maxQ=self.gridWorld[row][column][0]
                maxQMove=[0]
            elif maxQ==self.gridWorld[row][column][0]:
                maxQMove.append(0)

        if row<14:
            possibleMoves.append(1)
            if maxQ<self.gridWorld[row][column][1]:
                maxQ=self.gridWorld[row][column][1]
                maxQMove=[1]
            elif maxQ==self.gridWorld[row][column][1]:
                maxQMove.append(1)

        if column>0:
            possibleMoves.append(3)
            if maxQ<self.gridWorld[row][column][3]:
                maxQ=self.gridWorld[row][column][3]
                maxQMove=[3]
            elif maxQ==self.gridWorld[row][column][3]:
                maxQMove.append(3)

        if column<14:
            possibleMoves.append(2)
            if maxQ<self.gridWorld[row][column][2]:
                maxQ=self.gridWorld[row][column][2]
                maxQMove=[2]
            elif maxQ==self.gridWorld[row][column][2]:
                maxQMove.append(2)

        explorationProbability=random.randint(1,10)
        if explorationProbability/10.0 > self.epsilon:
            for move in maxQMove:
                possibleMoves.remove(move)

            if possibleMoves==[]:
                return [random.choice(maxQMove),maxQ]

            randomMove=random.choice(possibleMoves)

            if randomMove==0:
                QVal=self.gridWorld[row][column][0]
            elif randomMove==1:
                QVal=self.gridWorld[row][column][1]
            elif randomMove==2:
                QVal=self.gridWorld[row][column][2]
            else:
                QVal=self.gridWorld[row][column][3]
            return [randomMove,QVal]

        return [random.choice(maxQMove),maxQ]


    def EpsilonGreedyLearn(self):
        episode=1
        while episode<=self.maxEpisodes:
            row=1
            column=1
            steps=0
            while True:
                nextState = self.SelectEpsilonGreedyNeighbor(row,column)

                steps+=1

                if nextState[0]==0:
                    newRow= row-1
                    newColumn=column
                elif nextState[0]==1:
                    newRow=row+1
                    newColumn=column
                elif nextState[0]==2:
                    newColumn=column+1
                    newRow=row
                else:
                    newColumn=column-1
                    newRow=row


                if newRow<0:
                    self.gridWorld[row][column][nextState[0]] = self.gridWorld[row][column][nextState[0]] + self.learningRate * -2
                elif newRow>14:
                    self.gridWorld[row][column][nextState[0]] = self.gridWorld[row][column][nextState[0]] + self.learningRate * -2
                elif newColumn<0:
                    self.gridWorld[row][column][nextState[0]] = self.gridWorld[row][column][nextState[0]] + self.learningRate * -2
                elif newColumn>14:
                    self.gridWorld[row][column][nextState[0]] = self.gridWorld[row][column][nextState[0]] + self.learningRate * -2
                else:
                    if newRow == 14 and newColumn==14:
                        self.gridWorld[row][column][nextState[0]] = self.gridWorld[row][column][nextState[0]] + self.learningRate * 
(10 - self.gridWorld[row][column][nextState[0]])
                        break

                    futureMove=self.SelectEpsilonGreedyNeighbor(newRow,newColumn)

                    self.gridWorld[row][column][nextState[0]] = self.gridWorld[row][column][nextState[0]] + self.learningRate * (-1 + 
(self.gamma*futureMove[1]) - self.gridWorld[row][column][nextState[0]])
                    row=newRow
                    column=newColumn
            print steps
            episode+=1

learner=QLearning()
learner.EpsilonGreedyLearn()
print learner.gridWorld
```