**SE 4050 – Deep Learning**

4th Year second semester

A picture containing text, clipart, vector graphics

Description automatically generated

Lab Assignment 3

Lasal Sandeepa Hettiarachchi

IT19132310

B.Sc. (Hons) in Information Technology Specializing in Software Engineering

Submitted to

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

## Table of Contents

[Table of Contents ii](#_Toc118084038)

[1. Model-based algorithms (Value iteration approach) 3](#_Toc118084039)

[1.1 valueIterationAgents.py 3](#_Toc118084040)

[1.2 analysis.py 4](#_Toc118084041)

[2. Model-Free algorithms (Q-learning) 5](#_Toc118084042)

[2.1 qlearningAgents.py 5](#_Toc118084043)

[3. Autograder tool results 10](#_Toc118084044)

[3.1 Question 1 10](#_Toc118084045)

[3.2 Question 2 10](#_Toc118084046)

[3.3 Question 3 11](#_Toc118084047)

[3.4 Question 4 11](#_Toc118084048)

[3.5 Question 5 11](#_Toc118084049)

[3.6 Question 6 12](#_Toc118084050)

[3.7 Question 7 12](#_Toc118084051)

[4. Appendix 13](#_Toc118084052)

## Model-based algorithms (Value iteration approach)

Model-based algorithms require a model of the environment in that it incoperates the knowledge of the transition process from one state to another.

### valueIterationAgents.py

1. # Write value iteration code here
2. "\*\*\* YOUR CODE HERE \*\*\*"
3. for \_ in range(iterations):
4. current\_values = self.values.copy() #Getting a copy of the
5. states = self.mdp.getStates()
6. for state in states: # iterating through each state
7. if self.mdp.isTerminal(state):
8. continue
9. # get value for best possible action for changing state
10. actions = self.mdp.getPossibleActions(state)
11. val\_arr = []
12. for a in actions:
13. val\_arr.append(self.getQValue(state, a))
14. best\_value = max(val\_arr)
15. current\_values[state] = best\_value
16. self.values = current\_values

def computeQValueFromValues(self, state, action):

"""

Compute the Q-value of action in state from the

value function stored in self.values.

"""

"\*\*\* YOUR CODE HERE \*\*\*"

qValue = 0

# go through every possible outcome of the action

t\_state\_probs = self.mdp.getTransitionStatesAndProbs(state, action)

for nextState, probability in t\_state\_probs:

# add reward & future reward (=V) \* probability of the outcome

reward = self.mdp.getReward(state, action, nextState)

discount = self.discount

val = self.values[nextState]

qValue = qValue + probability \* (reward + discount \* val)

return qValue

util.raiseNotDefined()

def computeActionFromValues(self, state):

"""

The policy is the best action in the given state

according to the values currently stored in self.values.

You may break ties any way you see fit. Note that if

there are no legal actions, which is the case at the

terminal state, you should return None.

"""

"\*\*\* YOUR CODE HERE \*\*\*"

# retreive the best possible action for the state

policies = util.Counter()

pos\_actions = self.mdp.getPossibleActions(state)

for action in pos\_actions:

# how good is an action = q-value (which considers all possible outcomes)

policies[action] = self.getQValue(state, action)

# return the best action, e.g. 'north'

return policies.argMax()

util.raiseNotDefined()

### analysis.py

1. def question2():
2. answerDiscount = 0.9
3. answerNoise = 0
4. return answerDiscount, answerNoise
5. def question3a():
6. answerDiscount = 0.1
7. answerNoise = 0
8. answerLivingReward = 0.7 # give a negative penalty for moving
9. return answerDiscount, answerNoise, answerLivingReward
10. def question3b():
11. answerDiscount = 0.1
12. answerNoise = 0.1
13. answerLivingReward = 0.7
14. return answerDiscount, answerNoise, answerLivingReward
15. def question3c():
16. answerDiscount = 0.5
17. answerNoise = 0
18. answerLivingReward = 0.5
19. return answerDiscount, answerNoise, answerLivingReward
20. def question3d():
21. answerDiscount = 0.9
22. answerNoise = 0.1
23. answerLivingReward = 0.2
24. return answerDiscount, answerNoise, answerLivingReward
25. def question3e():
26. answerDiscount = 0
27. answerNoise = 0
28. answerLivingReward = 0
29. return answerDiscount, answerNoise, answerLivingReward
30. def question6():
31. answerEpsilon = None
32. answerLearningRate = None
33. # return answerEpsilon, answerLearningRate
34. # If not possible, return 'NOT POSSIBLE'
35. return 'NOT POSSIBLE'
36. if \_\_name\_\_ == '\_\_main\_\_':
37. print 'Answers to analysis questions:'
38. import analysis
39. for q in [q for q in dir(analysis) if q.startswith('question')]:
40. response = getattr(analysis, q)()
41. print ' Question %s:\t%s' % (q, str(response))

## Model-Free algorithms (Q-learning)

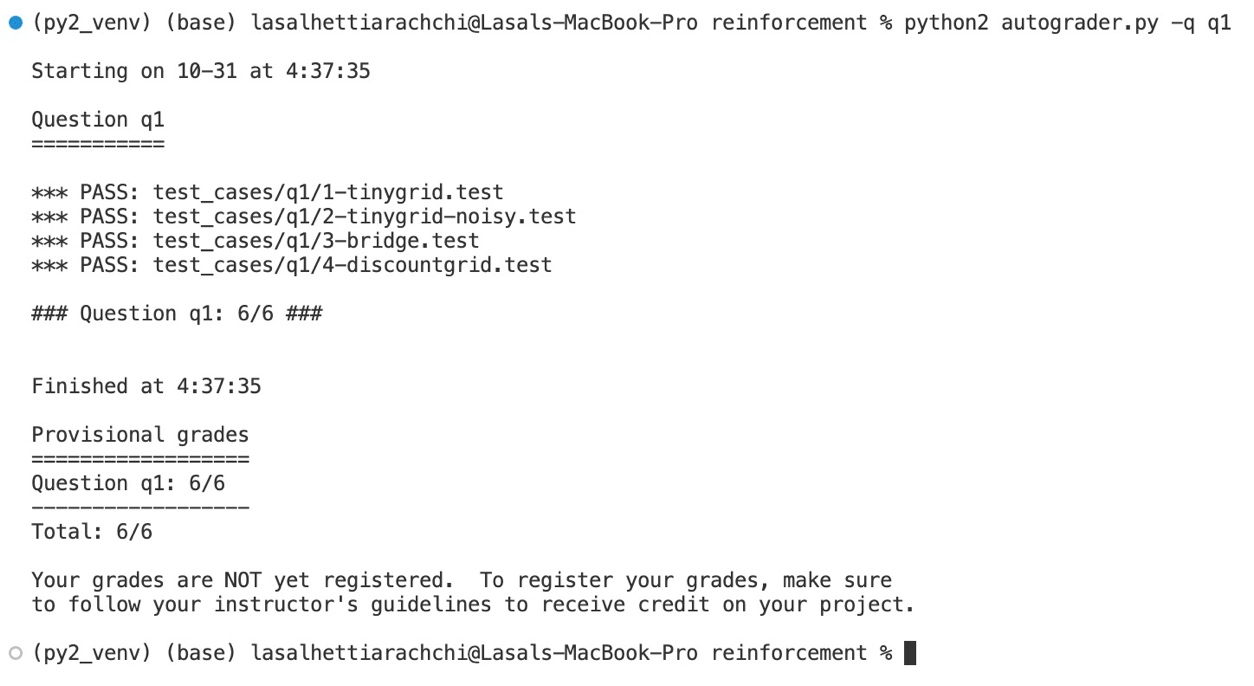
In reinforcement learning, a model-free algorithm is an algorithm which does not use the transition probability distribution associated

### qlearningAgents.py

1. # qlearningAgents.py
2. # ------------------
3. # Licensing Information: You are free to use or extend these projects for
4. # educational purposes provided that (1) you do not distribute or publish
5. # solutions, (2) you retain this notice, and (3) you provide clear
6. # attribution to UC Berkeley, including a link to http://ai.berkeley.edu.
7. #
8. # Attribution Information: The Pacman AI projects were developed at UC Berkeley.
9. # The core projects and autograders were primarily created by John DeNero
10. # (denero@cs.berkeley.edu) and Dan Klein (klein@cs.berkeley.edu).
11. # Student side autograding was added by Brad Miller, Nick Hay, and
12. # Pieter Abbeel (pabbeel@cs.berkeley.edu).
13. from game import \*
14. from learningAgents import ReinforcementAgent
15. from featureExtractors import \*
16. import random,util,math
17. class QLearningAgent(ReinforcementAgent):
18. """
19. Q-Learning Agent
20. Functions you should fill in:
21. - computeValueFromQValues
22. - computeActionFromQValues
23. - getQValue
24. - getAction
25. - update
26. Instance variables you have access to
27. - self.epsilon (exploration prob)
28. - self.alpha (learning rate)
29. - self.discount (discount rate)
30. Functions you should use
31. - self.getLegalActions(state)
32. which returns legal actions for a state
33. """
34. def \_\_init\_\_(self, \*\*args):
35. "You can initialize Q-values here..."
36. ReinforcementAgent.\_\_init\_\_(self, \*\*args)
37. "\*\*\* YOUR CODE HERE \*\*\*"
38. self.values = util.Counter()
39. def getQValue(self, state, action):
40. """
41. Returns Q(state,action)
42. Should return 0.0 if we have never seen a state
43. or the Q node value otherwise
44. """
45. "\*\*\* YOUR CODE HERE \*\*\*"
46. return self.values[(state, action)]
47. util.raiseNotDefined()
48. def computeValueFromQValues(self, state):
49. """
50. Returns max\_action Q(state,action)
51. where the max is over legal actions. Note that if
52. there are no legal actions, which is the case at the
53. terminal state, you should return a value of 0.0.
54. """
55. "\*\*\* YOUR CODE HERE \*\*\*"
56. maxQ = float('-inf')
57. for action in self.getLegalActions(state):
58. maxQ = max(maxQ, self.getQValue(state, action))
59. return maxQ if maxQ != float('-inf') else 0.0
60. util.raiseNotDefined()
61. def computeActionFromQValues(self, state):
62. """
63. Compute the best action to take in a state. Note that if there
64. are no legal actions, which is the case at the terminal state,
65. you should return None.
66. """
67. "\*\*\* YOUR CODE HERE \*\*\*"
68. if len(self.getLegalActions(state)) == 0:
69. return None
70. bestQ = self.computeValueFromQValues(state)
71. bestActions = []
72. for action in self.getLegalActions(state):
73. if bestQ == self.getQValue(state, action):
74. bestActions.append(action)
75. return random.choice(bestActions)
76. util.raiseNotDefined()
77. def getAction(self, state):
78. """
79. Compute the action to take in the current state. With
80. probability self.epsilon, we should take a random action and
81. take the best policy action otherwise. Note that if there are
82. no legal actions, which is the case at the terminal state, you
83. should choose None as the action.
84. HINT: You might want to use util.flipCoin(prob)
85. HINT: To pick randomly from a list, use random.choice(list)
86. """
87. # Pick Action
88. legalActions = self.getLegalActions(state)
89. action = None
90. "\*\*\* YOUR CODE HERE \*\*\*"
91. if util.flipCoin(self.epsilon):
92. action = random.choice(legalActions)
93. else:
94. action = self.computeActionFromQValues(state)
95. return action
96. util.raiseNotDefined()
97. def update(self, state, action, nextState, reward):
98. """
99. The parent class calls this to observe a
100. state = action => nextState and reward transition.
101. You should do your Q-Value update here
102. NOTE: You should never call this function,
103. it will be called on your behalf
104. """
105. "\*\*\* YOUR CODE HERE \*\*\*"
106. oldValue = self.values[(state, action)]
107. newValue = reward + (self.discount \* self.computeValueFromQValues(nextState))
108. self.values[(state, action)] = (1 - self.alpha) \* oldValue + self.alpha \* newValue
109. #util.raiseNotDefined()
110. def getPolicy(self, state):
111. return self.computeActionFromQValues(state)
112. def getValue(self, state):
113. return self.computeValueFromQValues(state)
114. class PacmanQAgent(QLearningAgent):
115. "Exactly the same as QLearningAgent, but with different default parameters"
116. def \_\_init\_\_(self, epsilon=0.05,gamma=0.8,alpha=0.2, numTraining=0, \*\*args):
117. """
118. These default parameters can be changed from the pacman.py command line.
119. For example, to change the exploration rate, try:
120. python pacman.py -p PacmanQLearningAgent -a epsilon=0.1
121. alpha - learning rate
122. epsilon - exploration rate
123. gamma - discount factor
124. numTraining - number of training episodes, i.e. no learning after these many episodes
125. """
126. args['epsilon'] = epsilon
127. args['gamma'] = gamma
128. args['alpha'] = alpha
129. args['numTraining'] = numTraining
130. self.index = 0 # This is always Pacman
131. QLearningAgent.\_\_init\_\_(self, \*\*args)
132. def getAction(self, state):
133. """
134. Simply calls the getAction method of QLearningAgent and then
135. informs parent of action for Pacman. Do not change or remove this
136. method.
137. """
138. action = QLearningAgent.getAction(self,state)
139. self.doAction(state,action)
140. return action
141. class ApproximateQAgent(PacmanQAgent):
142. """
143. ApproximateQLearningAgent
144. You should only have to overwrite getQValue
145. and update. All other QLearningAgent functions
146. should work as is.
147. """
148. def \_\_init\_\_(self, extractor='IdentityExtractor', \*\*args):
149. self.featExtractor = util.lookup(extractor, globals())()
150. PacmanQAgent.\_\_init\_\_(self, \*\*args)
151. self.weights = util.Counter()
152. def getWeights(self):
153. return self.weights
154. def getQValue(self, state, action):
155. """
156. Should return Q(state,action) = w \* featureVector
157. where \* is the dotProduct operator
158. """
159. "\*\*\* YOUR CODE HERE \*\*\*"
160. features = self.featExtractor.getFeatures(state, action)
161. sum = 0
162. for feature, value in features.iteritems():
163. sum += self.weights[feature] \* value
164. return
165. util.raiseNotDefined()
166. def update(self, state, action, nextState, reward):
167. """
168. Should update your weights based on transition
169. """
170. "\*\*\* YOUR CODE HERE \*\*\*"
171. newValue = reward + self.discount \* self.computeValueFromQValues(nextState)
172. oldValue = self.getQValue(state, action)
173. difference = newValue - oldValue
174. features = self.featExtractor.getFeatures(state, action)
175. for feature, value in features.iteritems():
176. self.weights[feature] += self.alpha \* difference \* features[feature]
177. #util.raiseNotDefined()
178. def final(self, state):
179. "Called at the end of each game."
180. # call the super-class final method
181. PacmanQAgent.final(self, state)
182. # did we finish training?
183. if self.episodesSoFar == self.numTraining:
184. # you might want to print your weights here for debugging
185. "\*\*\* YOUR CODE HERE \*\*\*"
186. pass

## Autograder tool results

### Question 1



### Question 2

Graphical user interface, text, application, letter, email

Description automatically generated

### Question 3

Text, letter

Description automatically generated

### Question 4

Graphical user interface, text, application, letter, email

Description automatically generated

### Question 5

Graphical user interface, text, application, letter, email

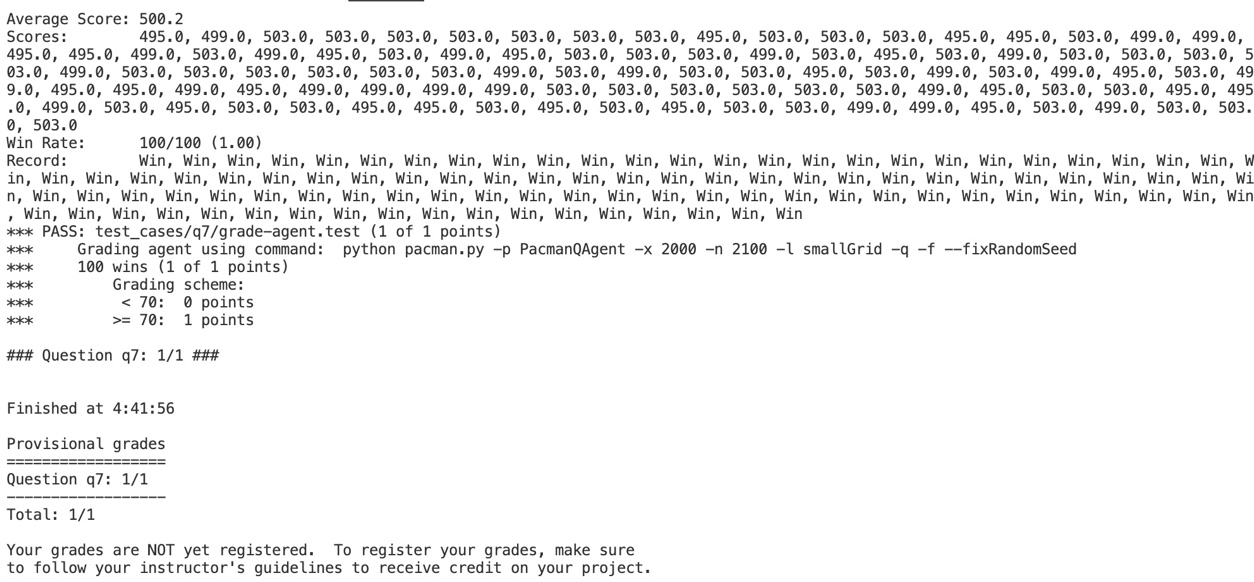
Description automatically generated

### Question 6

Graphical user interface, text, application, letter, email

Description automatically generated

### Question 7



## Appendix

A picture containing graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated