**Programming Assignment 4**

1. An advantage to bootstrapping in Easy 21 is that we can update in any step of the algorithm, which allows us to adapt to the values given to us. Monte Carlo, instead updates at the end of each episode and does not allow for updating after each step.
2. I would expect bootstrapping to help more in blackjack than easy 21. In the implementation of easy 21 that we have, the value of the card is generated randomly, whereas with blackjack it is much more predictable since we are using a deck. This, along with the fact that we can update in between each step, makes it advantageous for blackjack.
3. With function approximation we could have a guess at what the outcome of the algorithm will be and use this to make our decision to maximize reward.
4. We could learn an exploration and step-size to maximize our potential reward.

CODE:

1. from \_\_future\_\_ import print\_function
2. #!/usr/bin/env python2
3. # -\*- coding: utf-8 -\*-
4. """
5. Created on Tue Nov 14 10:07:17 2017
7. @author: rditljtd
8. """
10. from random import randint
11. import operator
12. import sys
13. import random
14. import time
16. hit = False
17. currentPlayerScore = 0
18. currentDealerScore = 0
19. initialDealerScore = 0
20. bust = False
21. playGame = True
22. win = False
23. possibleDealerInitialScores = [1,2,3,4,5,6,7,8,9,10]
24. possiblePlayerScores = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21]
25. possibleHitValues = [True, False]
26. states = {}
27. **for** dis in possibleDealerInitialScores:
28. **for** ps in possiblePlayerScores:
29. **for** hv in possibleHitValues:
30. states[dis, ps, hv] = 0
31. #states = [possibleDealerInitialScores, possiblePlayerScores, possibleHitValues]
32. #[DealerInitialScore, PlayerCurrentScore, hit]
33. reward = states
34. stepsTaken = []
35. gamesPlayed = 0

38. def getCard(isFirstCard = False):
39. **if** (isFirstCard):
40. color = 2
41. **else**:
42. color = randint(1, 3)
43. number = randint(1,10)
44. **if** color == 1:
45. #print "red " + str(number)
46. number = -number
47. #if color == 2 or color == 3:
48. #print "black " + str(number)
49. **return** number


53. #Draw first card for both dealer and player
54. def initializeGame():
55. #print
56. #print ("Dealer first card: ")
57. global currentDealerScore
58. global initialDealerScore
59. currentDealerScore = getCard(True)
60. initialDealerScore = currentDealerScore
61. #print ("Score: " + str(currentDealerScore))
62. #print
63. #print "--------------PLAYER---------------------"
64. #print
65. #print ("Player first card: ")
66. global currentPlayerScore
67. currentPlayerScore = getCard(True)
68. #print ("Score: " + str(currentPlayerScore))
70. def endGame():
71. firstTimeDealer = True
72. firstTimePlayer = True
73. hit = "Y"
74. global currentPlayerScore
75. currentPlayerScore = 0
76. global initialDealerScore
77. initialDealerScore = 0
78. global currentDealerScore
79. currentDealerScore = 0
80. global win
81. win = False
82. global stepsTaken
83. stepsTaken = []
85. #For every action taken during a game, if the game ends in a win add 1 to that state.
87. **while** playGame == True:
88. initializeGame()
89. #hit = raw\_input("Hit: Y/N\t")[0].upper()
90. hit = random.choice(["Y", "N"])
91. stepsTaken.append([initialDealerScore, currentPlayerScore, hit=="Y"])
92. **while** hit == "Y":
93. global currentPlayerScore
94. global currentDealerScore
95. global initialDealerScore
96. currentPlayerScore = sum([currentPlayerScore, getCard()])
97. #print "Score: " + str(currentPlayerScore)
98. **if** currentPlayerScore > 21 or currentPlayerScore < 1:
99. #print "BUST"
100. bust = True
101. hit = "N"
102. win = False
103. **continue**
104. #hit = raw\_input("Hit: Y/N\t")[0].upper()
105. hit = random.choice(["Y", "N"])
106. stepsTaken.append([initialDealerScore, currentPlayerScore, hit=="Y"])

109. #print "-----------------------DEALER-------------------------"
110. #print "Score: " + str(currentDealerScore)
111. **while** currentDealerScore < 17 and currentPlayerScore in range (1, 22) and currentDealerScore in range(1,22):
112. currentDealerScore = sum([currentDealerScore, getCard()])
113. #print "Score: " + str(currentDealerScore)
114. **if** currentDealerScore > 21 or currentDealerScore < 1:
115. #print "BUST"
116. win = True
117. **continue**
118. #time.sleep(2)

121. **if** win == False and currentPlayerScore in range (currentDealerScore, 22):
122. win = True
123. **if** (win == True):
124. #print stepsTaken
125. **for** step in stepsTaken:
126. #print step
127. reward[step[0], step[1], step[2]] += 1
128. #print "YOU WON!"
129. **if** (win == False):
130. **for** step in stepsTaken:
131. #print step
132. reward[step[0], step[1], step[2]] += -1
133. #print "YOU LOST!"
134. endGame()
136. **if** (gamesPlayed == 1000000):
137. playGame = False
138. **if** (gamesPlayed < 1000000):
139. playGame = True#input("Play Again: Y/N\t")
140. **if** (gamesPlayed % 10000 == 0):
141. #print (str(gamesPlayed / 10000) + " percent complete ", end='\r')
142. sys.stdout.write("\r" + str(gamesPlayed/10000) + " percent complete")
143. sys.stdout.flush()
144. gamesPlayed += 1
146. print ("Thanks for Playing!")
147. sorted\_reward = sorted(reward.items(), key=operator.itemgetter(1))
148. **for** indReward in sorted\_reward:
149. print (str(indReward[0]) + " = " + str(reward[indReward[0]]))

152. #Gt = x + x2(v) + x3(v^2) + x4(v^3) ...
153. #Ss = Ss + Gt
154. #Ns = Ns + 1
155. #Vs = Ss/Ns
156. ##I'm confused as to how to implement the time-step to give correct reward values.
157. #I was not able to generate a graph or visualization