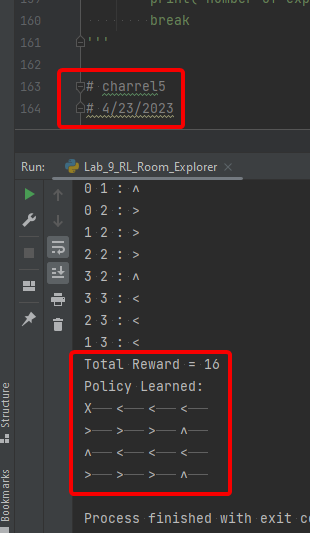
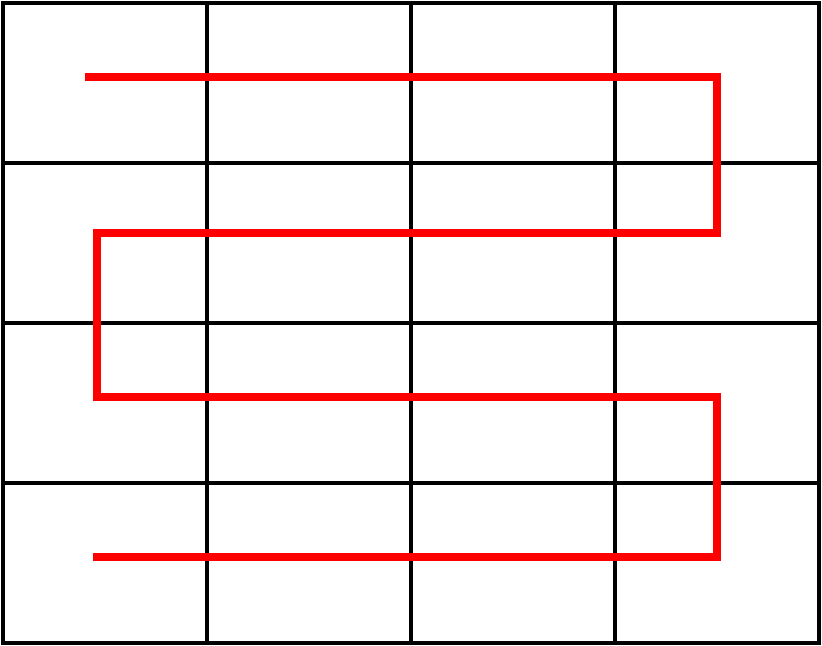
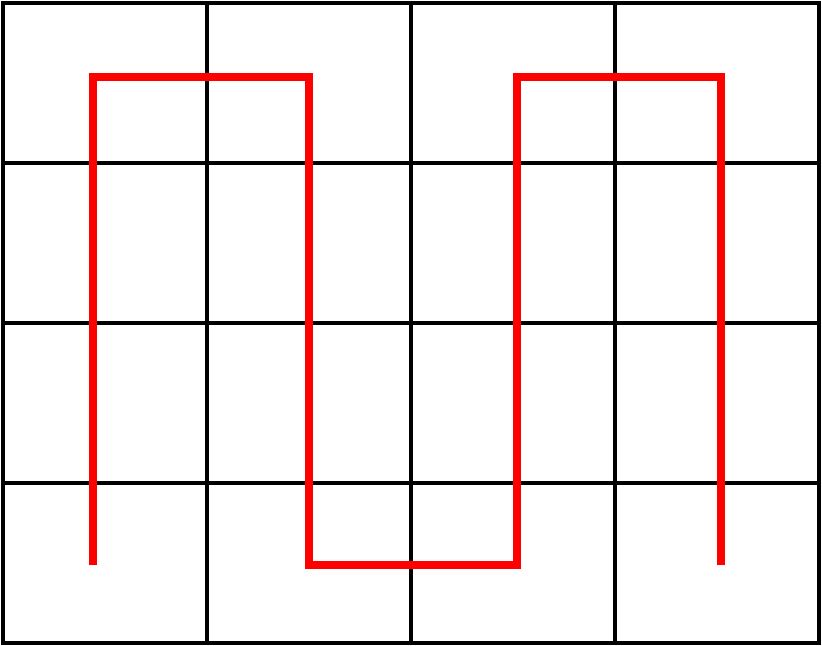
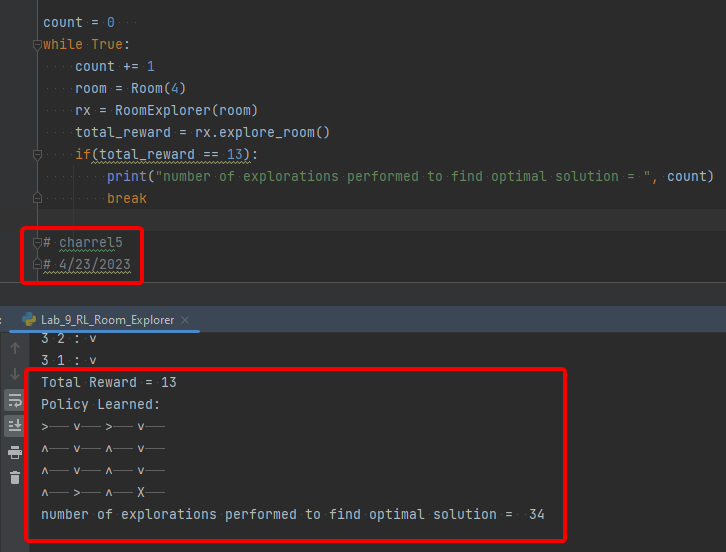
Lab 9

Training a Room Explorer using RL

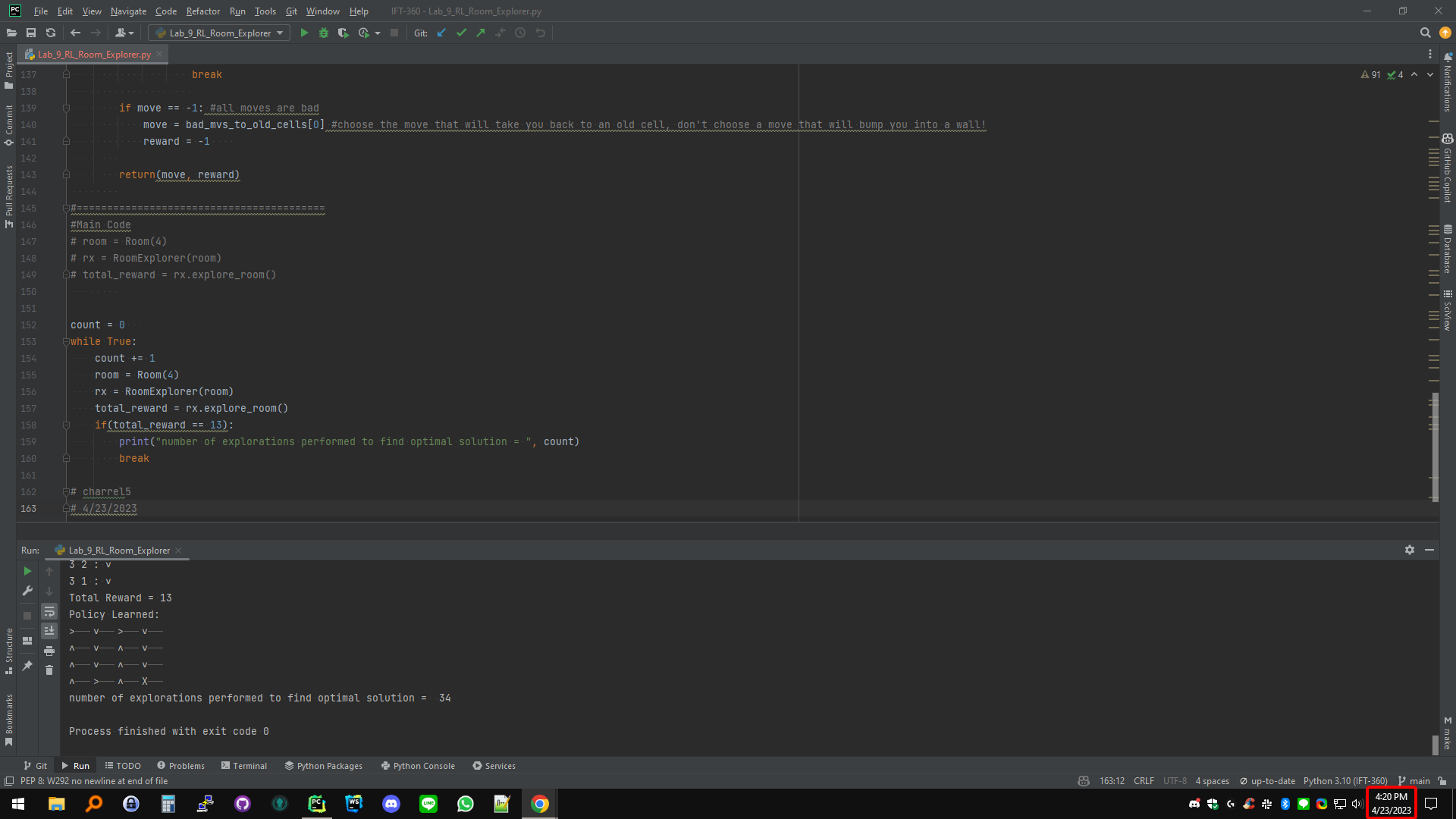




1. How does the optimal policy path look like? Draw the path on a 4 X 4 grid.
   1. 
2. Does it contain as much right or left turns as the policies generated by the old reward function?
   1. It contains the same number of turns
3. What is the number of times the exploration task was carried out until the optimal policy was found? (If you are lucky, the code will re run a couple of hundred times. If not, it can keep running for a couple of thousand times before it finds the optimal exploration policy. It should not take more than a couple of minutes maximum.)
   1. 34. See image below



Code and image



# -\*- coding: utf-8 -\*-

"""

Created on Wed Sep 21 12:20:43 2022

@author: aelbadra

"""

from random import shuffle

class Room:

def \_\_init\_\_(self, x=5):

self.nrows = x

self.ncols = x

return

class RoomExplorer:

def \_\_init\_\_(self, r):

#initialize the room dimensions

self.room = r

#initialize the total cost

self.cost = 0

#initialize the number of visited cells

self.visited = 0

#initialize the policy

self.policy = []

for i in range(r.nrows):

a=[]

for j in range(r.ncols):

a.append(-1)

self.policy.append(a)

return

#explore a room and build a policy based on returned reward/punishment

#new cell reward = 1

#old cell reward = -1

#bump into wall reward = -1

def explore\_room(self):

#intial position = [0,0]

x\_p = 0

y\_p = 0

total\_reward = 1

moves = {0:"∧", 1:"∨", 2:">", 3:"<", -1:"X"}

while True:

#get next move

(move, reward) = self.get\_next\_move(x\_p, y\_p)

print(x\_p, y\_p, ":", moves[move])

#update policy and total reward

self.policy[x\_p][y\_p] = move

total\_reward += reward

#implement the move

if move == 0: # up

y\_p += 1

elif move == 1: #down

y\_p -= 1

elif move == 2: #right

x\_p += 1

elif move == 3: #left

x\_p -= 1

#check for termination condition, if true, print policy & return total reward

missing\_moves=0

for i in range(self.room.nrows):

for j in range(self.room.ncols):

if self.policy[i][j] == -1:

missing\_moves += 1

if missing\_moves == 1:

print("Total Reward =", total\_reward)

print("Policy Learned:")

for j in range(self.room.ncols-1,-1,-1):

for i in range(self.room.nrows):

print(moves[ self.policy[i][j]], end="\t")

print()

return total\_reward

#get the next move to implement based on returned reward/punishment

def get\_next\_move(self, x\_p, y\_p):

#generate random move

rand\_move = [0,1,2,3]

shuffle(rand\_move)

#print("Shuffled moves:", rand\_move)

bad\_mvs\_to\_old\_cells = []

reward = -1

move = -1

for mv in rand\_move:

#calculate reward/punishment based on the reward of the reached cell

if mv == 0: #go up

if y\_p + 1 >= self.room.nrows: #if bumping into wall

reward = -1

elif self.policy[x\_p][y\_p+1] >= 0: #if old cell

reward = -1

bad\_mvs\_to\_old\_cells.append(mv)

elif self.policy[x\_p][y\_p+1] == -1: #if new cell

reward = 1

move = mv

break

elif mv == 1: #go down

if y\_p - 1 <0: #if bumping into wall

reward = -1

elif self.policy[x\_p][y\_p-1] >= 0: #if old cell

reward = -1

bad\_mvs\_to\_old\_cells.append(mv)

elif self.policy[x\_p][y\_p-1] == -1: #if new cell

reward = 1

move = mv

break

#calculate reward/punishment based on the reward of the reached cell

elif mv == 2: #go right

if x\_p + 1 >= self.room.ncols: #if bumping into wall

reward = -1

elif self.policy[x\_p+1][y\_p] >= 0: #if old cell

reward = -1

bad\_mvs\_to\_old\_cells.append(mv)

elif self.policy[x\_p+1][y\_p] == -1: #if new cell

reward = 0

move = mv

break

elif mv == 3: #go left

if x\_p - 1 <0: #if bumping into wall

reward = -1

elif self.policy[x\_p-1][y\_p] >= 0: #if old cell

reward = -1

bad\_mvs\_to\_old\_cells.append(mv)

elif self.policy[x\_p-1][y\_p] == -1: #if new cell

reward = 0

move = mv

break

if move == -1: #all moves are bad

move = bad\_mvs\_to\_old\_cells[0] #choose the move that will take you back to an old cell, don't choose a move that will bump you into a wall!

reward = -1

return(move, reward)

#=========================================

#Main Code

# room = Room(4)

# rx = RoomExplorer(room)

# total\_reward = rx.explore\_room()

count = 0

while True:

count += 1

room = Room(4)

rx = RoomExplorer(room)

total\_reward = rx.explore\_room()

if(total\_reward == 13):

print("number of explorations performed to find optimal solution = ", count)

break

# charrel5

# 4/23/2023

