CSEN 166 Artificial Intelligence

Lab Assignment #2: Vacuum Cleaner Agent

Name: Joshua Colleran ID: W1637923

Explanation of the defined functions:

def goal\_test(self, state):

#checks if both locations are clean

def actions(self, state):

#Determines the possible actions given a current state

#Takes current state as input and returns possible actions

#if the state is dirty suck is a possible function

#if state is on the right go left is an action

#if state is on the left go right is an action

def update(self, state, action):

#updates the state based on the current action

#takes the current state and action as an input and returns the new state after the action

#makes a copy of the current state

#if action is suck then the new state is clean

#updates states location

def run(self, initial\_state):

#excutes the action of the agent until goal state

#It repeatedly selects actions, updates the state, and

#increments the total cost until the goal state is reached.

#returns the actions and total cose

def test\_vacuum\_cleaner\_agent():

#test case

# This function tests the vacuum cleaner agent on different initial states and compares the actions taken with the expected actions and total cost.

# it iterates through each test case, if the actual and expected values match, it prints "Correct Outputs!" Otherwise, it prints "Wrong Outputs!".

class Agent:

#checks if both locations are clean

def goal\_test(self, state):

if state[0] == "Clean" and state[1] == "Clean":

return True

else:

return False

#Determines the possible actions given a current state

#Takes current state as input and returns possible actions

def actions(self, state):

possible\_actions = []

#if the state is dirty suck is a possible function

if state[state[2]] == "Dirty":

possible\_actions.append("Suck")

#if state is on the right go left is an action

elif state[2] == 1:

possible\_actions.append("Left")

#if state is on the left go right is an action

else:

possible\_actions.append("Right")

return possible\_actions

#updates the state based on the current action

#takes the current state and action as an input and returns the new state after the action

def update(self, state, action):

#makes a copy of the current state

new\_state = state.copy()

#if action is suck then the new state is clean

if action == "Suck":

new\_state[new\_state[2]] = "Clean"

#updates states location

elif action == "Left":

new\_state[2] = 0

elif action == "Right":

new\_state[2] = 1

return new\_state

#excutes the action of the agent until goal state

def run(self, initial\_state):

state = initial\_state

action\_seq = []

total\_cost = 0

#It repeatedly selects actions, updates the state, and

#increments the total cost until the goal state is reached.

while not self.goal\_test(state):

possible\_actions = self.actions(state)

action = possible\_actions[0]

action\_seq.append(action)

total\_cost += 1

state = self.update(state, action)

#returns the actions and total cose

return action\_seq, total\_cost

# Test case

def test\_vacuum\_cleaner\_agent():

test\_cases = [

(["Dirty", "Dirty", 0], ["Suck", "Right", "Suck"], 3),

(["Clean", "Dirty", 0], ["Right", "Suck"], 2),

(["Dirty", "Clean", 1], ["Left", "Suck"], 2),

(["Clean", "Clean", 0], [], 0),

(["Dirty", "Dirty", 1], ["Suck", "Left", "Suck"], 3),

(["Clean", "Dirty", 1], ["Suck"], 1),

(["Dirty", "Clean", 0], ["Suck"], 1),

(["Clean", "Clean", 1], [], 0),

]

for initial\_state, expected\_actions, expected\_cost in test\_cases:

agent = Agent()

actions\_taken, total\_cost = agent.run(initial\_state)

if actions\_taken == expected\_actions and total\_cost == expected\_cost:

print("Correct Outputs!")

else:

print("Wrong Outputs!")

test\_vacuum\_cleaner\_agent()