**Virtual Pi2Go Programming: WS24 & Ex24 Sample Answers**

**WS24**

**Exercise 1:** As with the previous worksheet, the lower epsilon\_reduce and learning\_rate are, the more likely the program is to learn the correct algorithm but the longer it will take. These values seem to work most of the time and its relatively quick.

import simclient.simrobot as pi2go

import time, random

pi2go.init()

actions = ['forward','backward','left','right']

def action\_reward(action\_list, default):

 action\_rewards = {}

 for i in range(0, 2):

 for j in range(0, 2):

 for k in (action\_list):

 action\_rewards[((i, j), k)] = default

 return action\_rewards

def execute\_action(action):

 if (action == "forward"):

 pi2go.forward(10)

 elif (action == "backward"):

 pi2go.reverse(10)

 elif (action == "left"):

 pi2go.spinLeft(10)

 elif (action == "right"):

 pi2go.spinRight(10)

 else:

 pi2go.stop()

 time.sleep(3)

def best\_action(state):

 max\_reward = 0

 for act in actions:

 if (reward\_dictionary[(state, act)] > max\_reward):

 action = act

 max\_reward = reward\_dictionary[(state, act)]

 return action

reward\_dictionary = action\_reward(actions, 1)

rewards = {(1, 1):1, (1, 0):2, (0, 1):0, (0, 0):1}

epsilon = 1

epsilon\_reduce = 0.05

learning\_rate = 0.5

while (epsilon > 0):

 explore = random.random()

 state = (pi2go.irLeftLine(), pi2go.irRightLine())

 if (explore < epsilon):

 action = random.choice(actions)

 print("Random Action: " + action)

 else:

 action = best\_action(state)

 print("Best Action: " + action)

 execute\_action(action)

 reward = rewards[(pi2go.irLeftLine(), pi2go.irRightLine())]

 reward\_dictionary[(state, action)] = reward\_dictionary[(state, action)] + (reward - reward\_dictionary[(state, action)])\*learning\_rate

 if (reward == 2):

 epsilon = epsilon - epsilon\_reduce

 print("New epsilon: " + str(epsilon))

pi2go.stop()

print(reward\_dictionary)

for key in reward\_dictionary:

 print("Average reward for " + str(key) + " is " + str(reward\_dictionary[key]))

**Ex24**

**Exercise 1:** I did some tweaking of the speed epsilon reduced to get this working. Students may also want to experiment with the learning rate and the rewards.

import simclient.simrobot as pi2go

import time, random

pi2go.init()

actions = ['forward','left','right']

def action\_reward(action\_list, default):

 action\_rewards = {}

 for i in range(0, 2):

 for j in range(0, 2):

 for k in range(0, 2):

 for a in (action\_list):

 action\_rewards[((i, j, k), a)] = default

 return action\_rewards

def execute\_action(action):

 if (action == "forward"):

 pi2go.forward(10)

 elif (action == "left"):

 pi2go.spinLeft(10)

 elif (action == "right"):

 pi2go.spinRight(10)

 else:

 pi2go.stop()

 time.sleep(3)

def best\_action(state):

 max\_reward = 0

 for act in actions:

 if (reward\_dictionary[(state, act)] > max\_reward):

 action = act

 max\_reward = reward\_dictionary[(state, act)]

 return action

reward\_dictionary = action\_reward(actions, 1)

rewards = {(0, 0, 0):2, (0, 0, 1):1, (0, 1, 0):1, (0, 1, 1):1, (1, 0, 0):0, (1, 0, 1):0, (1, 1, 0):0, (1, 1, 1):0}

epsilon = 1

epsilon\_reduce = 0.02

learning\_rate = 0.5

while (epsilon > 0):

 explore = random.random()

 state = (pi2go.irCentre(), pi2go.irLeft(), pi2go.irRight())

 if (explore < epsilon):

 action = random.choice(actions)

 print("Random Action: " + action)

 else:

 action = best\_action(state)

 print("Best Action: " + action)

 execute\_action(action)

 reward = rewards[(pi2go.irCentre(), pi2go.irLeft(), pi2go.irRight())]

 reward\_dictionary[(state, action)] = reward\_dictionary[(state, action)] + (reward - reward\_dictionary[(state, action)])\*learning\_rate

 if (reward == 2):

 epsilon = epsilon - epsilon\_reduce

 print("New epsilon: " + str(epsilon))

pi2go.stop()

print(reward\_dictionary)

for key in reward\_dictionary:

 print("Average reward for " + str(key) + " is " + str(reward\_dictionary[key]))

**Exercise 2:** I spent some time experimenting with the rewards before I got this working.

import simclient.simrobot as pi2go

import time, random

pi2go.init()

actions = ['forward','left','right']

def action\_reward(action\_list, default):

 action\_rewards = {}

 for i in range(0, 2):

 for j in range(0, 2):

 for k in range(0, 2):

 for a in (action\_list):

 action\_rewards[((i, j, k), a)] = default

 return action\_rewards

def execute\_action(action):

 if (action == "forward"):

 pi2go.forward(10)

 elif (action == "left"):

 pi2go.spinLeft(10)

 elif (action == "right"):

 pi2go.spinRight(10)

 else:

 pi2go.stop()

 time.sleep(3)

def best\_action(state):

 max\_reward = 0

 for act in actions:

 if (reward\_dictionary[(state, act)] > max\_reward):

 action = act

 max\_reward = reward\_dictionary[(state, act)]

 return action

reward\_dictionary = action\_reward(actions, 1)

rewards = {(0, 0, 0):3, (0, 0, 1):4, (0, 1, 0):0, (0, 1, 1):4, (1, 0, 0):3, (1, 0, 1):3, (1, 1, 0):2, (1, 1, 1):2}

epsilon = 1

epsilon\_reduce = 0.02

learning\_rate = 0.5

while (epsilon > 0):

 explore = random.random()

 state = (pi2go.irCentre(), pi2go.irLeft(), pi2go.irRight())

 if (explore < epsilon):

 action = random.choice(actions)

 print("Random Action: " + action)

 else:

 action = best\_action(state)

 print("Best Action: " + action)

 execute\_action(action)

 reward = rewards[(pi2go.irCentre(), pi2go.irLeft(), pi2go.irRight())]

 print(str(state))

 print(str(reward))

 reward\_dictionary[(state, action)] = reward\_dictionary[(state, action)] + (reward - reward\_dictionary[(state, action)])\*learning\_rate

 if (reward == 4):

 epsilon = epsilon - epsilon\_reduce

 print("New epsilon: " + str(epsilon))

pi2go.stop()

print(reward\_dictionary)

for key in reward\_dictionary:

 print("Average reward for " + str(key) + " is " + str(reward\_dictionary[key]))

**Exercise 3:** There are other state and reward functions, as well as the one’s suggested in the execise, that could be tried. For instance the state could include information about how close in value the front light sensors were and the reward could also increase as the brightness of the light increased. This algorithm will mostly end up with the robot spinning left and right in front of the light.

import simclient.simrobot as pi2go

import time, random

pi2go.init()

actions = ['forward','left','right']

def action\_reward(action\_list, default):

 action\_rewards = {}

 for i in range(-1, 2):

 for j in range(-1, 2):

 for k in range(-1, 2):

 for a in (action\_list):

 action\_rewards[((i, j, k), a)] = default

 return action\_rewards

def execute\_action(action):

 if (action == "forward"):

 pi2go.forward(10)

 elif (action == "left"):

 pi2go.spinLeft(10)

 elif (action == "right"):

 pi2go.spinRight(10)

 else:

 pi2go.stop()

 time.sleep(3)

def best\_action(state):

 max\_reward = 0

 for act in actions:

 if (reward\_dictionary[(state, act)] > max\_reward):

 action = act

 max\_reward = reward\_dictionary[(state, act)]

 return action

reward\_dictionary = action\_reward(actions, 0.5)

epsilon = 1

epsilon\_reduce = 0.02

learning\_rate = 0.5

def get\_state():

 fl = pi2go.getLightFL()

 fr = pi2go.getLightFR()

 bl = pi2go.getLightBL()

 br = pi2go.getLightBR()

 comp\_f = 0

 if (fl > fr):

 comp\_f = 1

 elif (fr > fl):

 comp\_f = -1

 comp\_b = 0

 if (bl > br):

 comp\_b = 1

 elif (br > bl):

 comp\_b = -1

 comp = 0

 if (fl + fr > bl + br):

 comp = 1

 elif (bl + br > fl + fr):

 comp = -1

 return (comp\_f, comp\_b, comp)

def calculate\_reward():

 fl = pi2go.getLightFL()

 fr = pi2go.getLightFR()

 bl = pi2go.getLightBL()

 br = pi2go.getLightBR()

 if (bl + br > fl + fr):

 return 0

 f\_diff = fl - fr

 f\_diff\_mod = f\_diff/1023

 if (f\_diff\_mod > 0):

 return (1 - f\_diff\_mod)

 else:

 return (1 + f\_diff\_mod)

while (epsilon > 0):

 explore = random.random()

 state = get\_state()

 if (explore < epsilon):

 action = random.choice(actions)

 print("Random Action: " + action)

 else:

 action = best\_action(state)

 print("Best Action: " + action)

 execute\_action(action)

 reward = calculate\_reward()

 print(str(state))

 print(str(reward))

 reward\_dictionary[(state, action)] = reward\_dictionary[(state, action)] + (reward - reward\_dictionary[(state, action)])\*learning\_rate

 if (reward > 0.8):

 epsilon = epsilon - epsilon\_reduce

 print("New epsilon: " + str(epsilon))

pi2go.stop()

print(reward\_dictionary)

for key in reward\_dictionary:

 print("Average reward for " + str(key) + " is " + str(reward\_dictionary[key]))



 University of Liverpool, 2020

This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).