**Programming in Python with the Pi2Go Simulator: Part 3 Answer Booklet**

**Chapter 1**

**Question:** The program moves forward for 3 seconds, then backwards for 3 seconds, then turns left for 3 seconds, then turns right for 3 seconds and then stops.

**Sample Answer Exercise 1:**

import simclient.simrobot as pi2go

import time

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

pi2go.init()

for action in actions:

if (action == 'forward'):

pi2go.forward(10)

time.sleep(3)

elif (action == 'backward'):

pi2go.reverse(10)

time.sleep(3)

elif (action == 'left'):

pi2go.spinLeft(10)

time.sleep(3)

elif (action == 'right'):

pi2go.spinRight(10)

time.sleep(3)

else:

pi2go.stop()

**Sample Answer Exercise 2:**

import simclient.simrobot as pi2go

import time

actions = ['forward', 3, 'backward', 2, 'left', 1, 'right', 5, 'stop']

pi2go.init()

act = 1

for el in actions:

if (act == 1):

if (el == 'forward'):

pi2go.forward(10)

elif (el == 'backward'):

pi2go.reverse(10)

elif (el == 'left'):

pi2go.spinLeft(10)

elif (el == 'right'):

pi2go.spinRight(10)

else:

pi2go.stop()

act = 0

else:

time.sleep(el)

act = 1

**Sample Answer Exercise 3:**

import simclient.simrobot as pi2go

import time

pi2go.init()

pi2go.forward(10)

readings = []

while (len(readings) < 30):

reading = pi2go.getDistance()

readings.append(reading)

time.sleep(3)

pi2go.stop()

for reading in readings:

print(str(reading))

**Sample Answer Exercise 4:**

import simclient.simrobot as pi2go

import time

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

pi2go.init()

pi2go.forward(10)

while (len(actions) > 0):

for action in actions:

if (action == 'forward'):

pi2go.forward(10)

time.sleep(3)

elif (action == 'backward'):

pi2go.reverse(10)

time.sleep(3)

elif (action == 'left'):

pi2go.spinLeft(10)

time.sleep(3)

elif (action == 'right'):

pi2go.spinRight(10)

time.sleep(3)

else:

pi2go.stop()

time.sleep(3)

if (pi2go.getDistance() < 50):

actions.remove(action)

print("removing " + action)

pi2go.stop()

**Sample Answer Exercise 5:**

import simclient.simrobot as pi2go

import time

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

permitted = ['backward', 'left', 'stop']

pi2go.init()

for action in actions:

if (action in permitted):

if (action == 'forward'):

pi2go.forward(10)

time.sleep(3)

elif (action == 'backward'):

pi2go.reverse(10)

time.sleep(3)

elif (action == 'left'):

pi2go.spinLeft(10)

time.sleep(3)

elif (action == 'right'):

pi2go.spinRight(10)

time.sleep(3)

else:

pi2go.stop()

else:

print(action + " is not permitted")

**Sample Answer Exercise 6:**

import simclient.simrobot as pi2go

import time

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

data = []

pi2go.init()

for action in actions:

before = pi2go.getDistance()

if (action == 'forward'):

pi2go.forward(10)

time.sleep(3)

elif (action == 'backward'):

pi2go.reverse(10)

time.sleep(3)

elif (action == 'left'):

pi2go.spinLeft(10)

time.sleep(3)

elif (action == 'right'):

pi2go.spinRight(10)

time.sleep(3)

else:

pi2go.stop()

time.sleep(3)

after = pi2go.getDistance();

data.append([action, before, after])

print(data)

**Chapter 2**

**Exercise:**

import simclient.simrobot as pi2go

import time

import random

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

pi2go.init()

action = random.choice(actions)

duration = random.randint(1, 5)

if (action == 'forward'):

pi2go.forward(10)

time.sleep(duration)

elif (action == 'backward'):

pi2go.reverse(10)

time.sleep(duration)

elif (action == 'left'):

pi2go.spinLeft(10)

time.sleep(duration)

elif (action == 'right'):

pi2go.spinRight(10)

time.sleep(duration)

else:

pi2go.stop()

pi2go.stop()

**Chapter 3**

**Question 1:** The keys are forward, backward, left, right and stop

**Question 2:** The values are 3, 2, 5, 3, 2

**Question 3:** The program picks an action at random and executes for a specific duration name 3 seconds for forward, 2 seconds for backward, 5 seconds for left, 3 seconds for right and 2 seconds for stop.

**Exercise 1:**

import simclient.simrobot as pi2go

import time, random

pi2go.init()

action\_dictionary = {"forward":3, "backward":2, "left":5, "right":3, "stop":2}

for action in action\_dictionary:

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(action\_dictionary[action])

**Question 4:** The program selects an action at random. If the action results in either of the line sensors detecting black then it adds one to the value stored for the action in the dictionary. When any action gets a score of more than 3 then it stops.

**Question 5:** If the program starts on the square it terminates really quickly whereas if it starts off the square it runs for a long time until the robot by chance moves onto the square.

**Exercise 2:**

import simclient.simrobot as pi2go

import time, random

pi2go.init()

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

action\_dictionary = {"forward":0, "backward":0, "left":0, "right":0, "stop":0}

count = 0

while (count < 20):

action = random.choice(actions)

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)

if (pi2go.irLeftLine()):

action\_dictionary[action] = action\_dictionary[action] + 1

if (pi2go.irRightLine()):

action\_dictionary[action] = action\_dictionary[action] + 1

count = count + 1

pi2go.stop()

score = 0

winner = "no winner"

for key in action\_dictionary:

if (action\_dictionary[key] > score):

score = action\_dictionary[key]

winner = key

print("And the winner is: " + winner)

**Question 6:** The rewards dictionary provides a score which depends upon the values of the two line sensors: 2 if both sensors detect black, 1 if only one sensor detects black and 0 if neither sensor detects black. These scores are then added to the action scores each time an action is selected at random.

**Exercise 3:**

import simclient.simrobot as pi2go

import time, random

pi2go.init()

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

dictionary = {"forward":0, "backward":0, "left":0, "right":0, "stop":0}

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

while True:

action = random.choice(actions)

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)

dictionary[action] = dictionary[action] + rewards[(pi2go.irLeftLine(), pi2go.irRightLine())]

if (dictionary[action] > 3):

break

pi2go.stop()

print(dictionary)

**Chapter 4**

**Exercise 1:**

import simclient.simrobot as pi2go

import time, random

pi2go.init()

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

rewards = {1:1, 0:0}

action = random.choice(actions)

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)

pi2go.stop()

reward = rewards[pi2go.irLeftLine()]

print("The reward is: " + str(reward))

**Exercise 2:**

import simclient.simrobot as pi2go

import time, random

pi2go.init()

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

scores = {'forward':0,'backward':0,'left':0,'right':0,'stop':0}

rewards = {1:1, 0:0}

count = 0

while count < 20:

action = random.choice(actions)

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)

reward = rewards[pi2go.irLeftLine()]

scores[action] = scores[action] + reward

count = count + 1

pi2go.stop()

print("The rewards are: " + str(scores))

**Exercise 3:**

import simclient.simrobot as pi2go

import time, random

pi2go.init()

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

scores = {'forward':0,'backward':0,'left':0,'right':0,'stop':0}

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

count = 0

while count < 20:

action = random.choice(actions)

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)

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

scores[action] = scores[action] + reward

count = count + 1

pi2go.stop()

print("The rewards are: " + str(scores))

**Exercise 4:**

import simclient.simrobot as pi2go

import time, random

pi2go.init()

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

scores = {'forward':0,'backward':0,'left':0,'right':0,'stop':0}

attempts = {'forward':0,'backward':0,'left':0,'right':0,'stop':0}

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

count = 0

while count < 20:

action = random.choice(actions)

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)

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

scores[action] = scores[action] + reward

attempts[action] = attempts[action] + 1

count = count + 1

pi2go.stop()

for action in actions:

if (attempts[action] != 0):

average\_reward = scores[action]/attempts[action]

else:

average\_reward = 0

print("The reward for " + action + " is " + str(average\_reward))

**Chapter 5**

**Question 1:** No, just knowing the best action overall isn’t enough. We want the robot to different things depending upon whether it is accurately placed at the edge of the oval or is in the middle of the black and so on.

**Exercise 1:**

def action\_reward():

action\_rewards = {}

for i in range(0, 2):

for j in range(0, 2):

for k in ('left', 'right', 'reverse', 'forward'):

action\_rewards[((i, j), k)] = 0

return action\_rewards

**Exercise 2:**

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

**Exercise 3:** Note that this program has been tidied up a bit using functions.

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)

reward\_dictionary = action\_reward(actions, 0)

total\_attempts = action\_reward(actions, 0)

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

count = 0

while (count < 50):

action = random.choice(actions)

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

execute\_action(action)

reward\_dictionary[(state, action)] = reward\_dictionary[(state, action)] + rewards[(pi2go.irLeftLine(), pi2go.irRightLine())]

total\_attempts[(state, action)] = total\_attempts[(state, action)] + 1

count = count + 1

pi2go.stop()

print(reward\_dictionary)

for key in reward\_dictionary:

total = total\_attempts[key]

if (total == 0):

print("Never attempted " + str(key))

else:

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

**Question 2:** Probably yes, though they could get lucky.

**Question 3:** It should do. If the robot starts a long way from the oval then it is unlikely it will ever be in any of the states where a line sensor is activated.

**Chapter 6**

**Exercise 1:** Note the use of print statements. This program takes a while to run and the print statements help give the programmer a sense of progress. The program will run more quickly if epsilon\_reduce is increased, but is less likely to learn the “right” algorithm as a result. That said, the problem is simple enough that quite a high value for epsilon\_reduce can probably be used if attention span is proving a problem.

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 (total\_attempts[(state, act)] > 0):

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

action = act

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

else:

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

action = act

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

return action

reward\_dictionary = action\_reward(actions, 0)

total\_attempts = action\_reward(actions, 0)

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

epsilon = 1

epsilon\_reduce = 0.05

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

total\_attempts[(state, action)] = total\_attempts[(state, action)] + 1

if (reward == 2):

epsilon = epsilon - epsilon\_reduce

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

pi2go.stop()

print(reward\_dictionary)

for key in reward\_dictionary:

total = total\_attempts[key]

if (total == 0):

print("Never attempted " + str(key))

else:

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

**Chapter 7**

**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]))

**Chapter 8**

**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]))

**Chapter 10**

**Question 1:** The program writes the contents of the dictionary to a file called “policy\_file”. Each line in this file consists of a dictionary key, followed by “::” followed by the value for that key in the dictionary.

**Exercise 1:** The key bit is the final few lines of this program – the rest is my solution to WS24.

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()

f = open('policy\_file', 'w')

for key in reward\_dictionary:

f.write(str(key))

f.write('::')

f.write(str(reward\_dictionary[key]))

f.write('\n')

f.close()

**Exercise 2:**

f = open('policy\_file', 'r')

for line in f:

print(line)

**Question 2:**

s[0] returns ( and s[1] return 0 – these are the first and second characters in the string.

**Question 3:**

s[1:4] returns 0, 1 – these are the 2nd-5th characters in the string.

**Question 4:**

Python prints: ['(0', ' 1)'] – this is an array of two elements consisting of the part of the string before the comma and the part of the string after the comma.

**Question 5:** This reads in a file – assumed to be formatted as the output for the first example program in the worksheet. It first splits each line around :: into the key and the value from the dictionary. It then splits the key around the commas to extract strings that contain the values of the left and right line sensors and the action. It extracts the values of the left and right line sensors from these strings by knowing at which character they will occur. It then splits the action string around the ‘ to extract the string containing the action name. It then prints out the sensor values, the action name and the value string.

**Exercise 3:**

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)

f = open('policy\_file', 'r')

for line in f:

[key, value] = line.split('::')

[a, b, c] = key.split(',')

irR = a[2]

irL = b[1]

[x, action, y] = c.split('\'')

state = (int(irR), int(irL))

reward\_dictionary[(state, action)] = float(value)

f.close()

while (not pi2go.getSwitch()):

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

action = best\_action(state)

execute\_action(action)

pi2go.stop()

**Chapter 11**

**Question 1:**

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

f = open('no\_file', 'r')

FileNotFoundError: [Errno 2] No such file or directory: 'no\_file'

**Exercise 1:**

f = open('no\_file', 'r')

for line in f:

print(line)

When run the program crashes with a FileNotFoundError message.

**Exercise 2:**

while True:

number = input("Please enter a number\n")

try:

new\_number = int(number) \* 10

print(new\_number)

break

except:

print("That was not a number. Please try again")



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/).