**Model:**

import random

import matplotlib.pyplot as plt

import operator

import time

import my\_modules.agentframework1 as af

import my\_modules.io1 as io

import geometry1

environment, n\_rows, n\_cols = io.read\_data()

# Set the pseudo-random seed for reproducibility

random.seed(0)

# A variable to store the number of agents

n\_agents = 10

n\_iterations = 100

# Create a list to store agents and initialise agents

agents = []

store\_share = 0

for i in range(n\_agents):

# Create an agent

agents.append(af.Agent(agents, i, environment, n\_cols, n\_rows))

print(agents[i])

print(agents)

# Variables for constraining movement

# The minimum x coordinate

x\_min = 0

# The minimum y coordinate

y\_min = 0

# The maximum an agents x coordinate is allowed to be

x\_max = n\_cols - 1

# The maximum an agents y coordinate is allowed to be

y\_max = n\_rows - 1

# Calculate the Euclidean distance between (x0, y0) and (x1, y1)

# Set x0 and y0 to equal 0, x1 to equal 3, and y1 to equal 4

x0 = 0

y0 = 0

x1 = 3

y1 = 4

# Calculate the difference in the x coordinates

a = x1 - x0

# Claculate the difference in the y coordinates

b = y1 - y0

# Square the differences and add the squares

c = a\*a + b\*b

# Calculate the square root

distance = c \*\* 0.5

print("distance", distance)

start = time.perf\_counter()

max\_distance = 0 # Initialise max\_distance

for a in agents:

for b in agents:

distance = geometry1.get\_distance(a.x, a.y, b.x, b.y)

print("distance between", a, b, distance)

max\_distance = max(max\_distance, distance)

print("max\_distance", max\_distance)

def get\_max\_distance():

max\_distance = 0

for i in range(len(agents)):

a = agents[i]

for j in range(len(agents)):

b = agents[i]

distance = geometry1.get\_distance(a.x, b.x, a.y, b.y)

print("distance between", a, b, distance)

max\_distance = max(max\_distance, distance)

print("max\_distance", max\_distance)

end = time.perf\_counter()

print("Time taken to calculate maximum distance", end - start, "seconds")

def sum\_environment():

sum\_env = 0

for i in range(len(environment)):

for j in range(len(environment[i])):

sum\_env += environment[i][j]

return sum\_env

def sum\_agent\_stores():

sum\_as = 0

for i in range (len(agents)):

sum\_as = agents[i].store

return sum\_as

# Move agents and model loop

for ite in range(n\_iterations):

print("Iteration", ite)

# Move agents

print("Move")

for i in range(n\_agents):

# Change agents[i] coordinate randomly

agents[i].move(x\_min, y\_min, x\_max, y\_max)

agents[i].eat()

# print(agents[i])

# Share store

# Distribute shares

for i in range(n\_agents):

print(agents[i])

agents[i].store = agents[i].store\_shares

agents[i].store\_shares = 0

print(agents)

# Print the maximum distance between all the agents

print("Maximum distance between all the agents", get\_max\_distance())

# Print the total amount of resource

sum\_as = sum\_agent\_stores()

print("sum\_agent\_stores", sum\_as)

sum\_e = sum\_environment()

print("sum\_environment", sum\_e)

print("total resource", (sum\_as + sum\_e))

# Plot the agents

plt.imshow(environment)

plt.ylim(y\_max / 3, y\_max \* 2 / 3)

plt.xlim(x\_max / 3, x\_max \* 2 / 3)

for i in range(n\_agents):

plt.scatter(agents[i].x, agents[i].y, color='black')

# Plot largest x red

lx = max(agents, key=operator.attrgetter('x'))

plt.scatter(lx.x, lx.y, color='red')

# Plot smallest x blue

sx = min(agents, key=operator.attrgetter('x'))

plt.scatter(sx.x, sx.y, color='blue')

# Plot largest y yellow

ly = max(agents, key=operator.attrgetter('y'))

plt.scatter(ly.x, ly.y, color='yellow')

# Plot smallest y green

sy = min(agents, key=operator.attrgetter('y'))

plt.scatter(sy.x, sy.y, color='green')

plt.show()

**agentframework:**

import random

import geometry1

class Agent():

pass

def \_\_init\_\_(self, agents, i, environment, n\_rows, n\_cols):

"""

The constructor method.

Parameters

----------

agents: List

A list of Agent instances.

i : Integer

To be unique to each instance.

environment : List

A reference to a shared environment

n\_rows : Integer

The number of rows in environment.

n\_cols : Integer

The number of columns in environment.

Returns

-------

None.

"""

self.agents = agents

self.i = i

self.environment = environment

tnc = int(n\_cols / 3)

self.x = random.randint(tnc - 1, (2 \* tnc) - 1)

tnr = int(n\_rows / 3)

self.y = random.randint(tnc - 1, (2 \* tnr) - 1)

self.store = 0

self.store\_shares = 0

def \_\_str\_\_(self):

return self.\_\_class\_\_.\_\_name\_\_ + "(x=" + str(self.x) \

+ ", y=" + str(self.y) + ", i=" + str(self.i) + ")"

def \_\_repr\_\_(self):

return str(self)

def move(self, x\_min, y\_min, x\_max, y\_max):

rn = random.randint(0, 99)

if rn < 55:

self.x = self.x + 2

else:

self.x = self.x - 2

# Change y-coordinate

rn = random.randint(0, 99)

if rn > 55:

self.y = self.y - 2

else:

self.y = self.y + 2

# Apply movement constraints

if self.x < x\_min:

self.x = x\_min

if self.y < y\_min:

self.y = y\_min

if self.x > x\_max:

self.x = x\_max

if self.y > y\_max:

self.y = y\_max

def eat(self):

if self.environment[self.y][self.x] >= 10:

self.environment[self.y][self.x] -= 10

self.store += 10

def share(self, neighbourhood):

# Create a list of agents in neighbourhood

neighbours = []

# print(self.agents[self.i])

for a in self.agents:

distance = geometry1.get\_distance(a.x, a.y, self.x, self.y)

if distance < neighbourhood:

neighbours.append(a.i)

# Calculate amount to share

n\_neighbours = len(neighbours)

# print("n\_neighbours", n\_neighbours)

shares = self.store / n\_neighbours

# print("shares", shares)

# Add shares to store\_shares

for i in neighbours:

self.agents[i].store\_shares += shares

**io:**

import csv

def read\_data():

# Read input data

f = open('C:/Users/xiaoyu/programming/data/input/in (2).txt', newline='')

data = []

n\_rows = 0

n\_cols = None

for line in csv.reader(f, quoting=csv.QUOTE\_NONNUMERIC):

row = []

for value in line:

row.append(value)

# Print(value)

if n\_cols is None:

n\_cols = len(row)

assert(n\_cols == len(row))

data.append(row)

n\_rows += 1

f.close()

return data, n\_rows, n\_cols

**geometry:**

def get\_distance(x0, y0, x1, y1):

x = x1 - x0

y = y1 - y0

return (x\*x + y\*y) \*\* 0.5