

## **ДОДАТОК А**

Вихідний код програми для імітаційного моделювання

```

import EANT
params = EANT.Parameters()

params.PopulationSize = 100

genome = EANT.Genome(0, 3, 0, 2, False,
EANT.ActivationFunction.UNSIGNED_SIGMOID,
EANT.ActivationFunction.UNSIGNED_SIGMOID, 0, params,
0)
pop = EANT.Population(genome, params, True, 1.0,
0) # the 0 is the RNG seed

def evaluate(genome):

    # this creates a neural network (phenotype)
from the genome

    net = EANT.NeuralNetwork()
genome.BuildPhenotype(net)

    # let's input just one pattern to the net,
activate it once and get the output

    net.Input( [ 1.0, 0.0, 1.0 ] )
net.Activate()
output = net.Output()

    # the output can be used as any other Python
iterable. For the purposes of the tutorial,
    # we will consider the fitness of the
individual to be the neural network that outputs
constantly
    # 0.0 from the first output (the second
output is ignored)

    fitness = 1.0 - output[0]
return fitness

for generation in range(100): # run for 100
generations

    # retrieve a list of all genomes in the
population
genome_list = EANT.GetGenomeList(pop)

```

```

        # apply the evaluation function to all
genomes
        for genome in genome_list:
            fitness = evaluate(genome)
            genome.SetFitness(fitness)

        # at this point we may output some
information regarding the progress of evolution, best
fitness, etc.
        # it's also the place to put any code that
tracks the progress and saves the best genome or the
entire
        # population. We skip all of this in the
tutorial.

        # advance to the next generation
        pop.Epoch()

import numpy as np
from time import time
from Tkinter import Tk, Canvas
from random import randint

screenSize = 700

def main():
    #loading data
    f = open("datasets/tsp0100.txt",
'r').read().splitlines()
    numPlaces = f.pop(0)
    places = np.array([ tuple( map( float,
coord.split() ) ) for coord in f ])

    #calculating path
    start = time()
    path, length = algorithm( places )
    print(path)

    tottime = time() - start
    print( "Found path of length %s in %s seconds" %
( round(length,2), round(tottime, 2) ) )

    #displaying path

```

```

drawPath( path, places, length )

def randColor():
    return "#%06x" % randint(0,0xFFFFFF)

def drawPath(path, places, length):
    places = places [ path ]

    msg = "Length*: {:.2E}".format(length)
    canvas.create_text(screenSize/2,    screenSize+50,
text = msg, fill = 'black', font = ('Helvetica', 20,
'bold'))

    addToCanvas(places)
    canvas.update()
    root.mainloop()

def addToCanvas(places):
    min_x = np.min( places[:,0] )
    min_y = np.min( places[:,1] )

    max_x = np.max( places[:,0] )
    max_y = np.max( places[:,1] )

    for i in range( len( places ) ):
        c = places[i-1]
        c_next = places[i]

        scaled_x = (c[0] - min_x) / (max_x - min_x) *
screenSize + 20
        scaled_y = (c[1] - min_y) / (max_y - min_y) *
screenSize + 20

        scaled_x_next = (c_next[0] - min_x) / (max_x
- min_x) * screenSize + 20
        scaled_y_next = (c_next[1] - min_y) / (max_y
- min_y) * screenSize + 20

        canvas.create_oval( scaled_x - 4 , scaled_y -
4 , scaled_x + 4 , scaled_y + 4 , fill = randColor()
, outline = 'black' )
        canvas.create_oval( scaled_x_next - 4 ,
scaled_y_next - 4 , scaled_x_next + 4 ,
scaled_y_next + 4 , fill = randColor() , outline =
'black' )

```

```

        canvas.create_line( scaled_x, scaled_y ,
scaled_x_next, scaled_y_next , fill = 'black' )
        root = Tk()
        canvas = Canvas( root , width = screenSize + 40,
height = screenSize + 100 , bg = 'white' )
        canvas.pack()

def evaluate_path(genome):
    net = EANT.NeuralNetwork()

    try:

        genome.BuildEANTPhenotype(net, substrate,
params)

        error = 0
        depth = 3
        correct = 0.0

        net.Flush()

        net.Input([1, 0, 1])
        [net.Activate() for _ in range(depth)]
        o = net.Output()
        error += abs(o[0] - 1)
        if o[0] > 0.75:
            correct += 1.

        net.Flush()
        net.Input([0, 1, 1])
        [net.Activate() for _ in range(depth)]
        o = net.Output()
        error += abs(o[0] - 1)
        if o[0] > 0.75:
            correct += 1.

        net.Flush()
        net.Input([1, 1, 1])
        [net.Activate() for _ in range(depth)]
        o = net.Output()
        error += abs(o[0] - 0)
        if o[0] < 0.25:
            correct += 1.

        net.Flush()

```

```

        net.Input([0, 0, 1])
        [net.Activate() for _ in range(depth)]
        o = net.Output()
        error += abs(o[0] - 0)
        if o[0] < 0.25:
            correct += 1.

    return (4 - error) ** 2

except Exception as ex:
    print('Exception:', ex)
    return 0.0

def getbest(run):
    g = EANT.Genome(0,
                   substrate.GetMinCPPNInputs(),
                   0,
                   substrate.GetMinCPPNOutputs()
                   ,
                   False,
                   EANT.ActivationFunction.TANH,
                   EANT.ActivationFunction.TANH,
                   0,
                   params, 0)

    pop = EANT.Population(g, params, True, 1.0,
run)
    for generation in range(1000):
        # Evaluate genomes
        genome_list = EANT.GetGenomeList(pop)

        fitnesses =
EvaluateGenomeList_Serial(genome_list, evaluate_xor,
display=False)
        [genome.SetFitness(fitness) for genome,
fitness in zip(genome_list, fitnesses)]

        print('Gen: %d Best: %3.5f' %
(generation, max(fitnesses)))

    # Print best fitness
    # print("-----")
    # print("Generation: ", generation)
    # print("max ",

```

```

max([x.GetLeader().GetFitness() for x in
pop.Species]))

    # Visualize best network's Genome

    net = EANT.NeuralNetwork()
    pop.Species[0].GetLeader().BuildPhenotype
(net)
        img = np.zeros((500, 500, 3),
dtype=np.uint8)
    img += 10
    EANT.DrawPhenotype(img, (0, 0, 500, 500),
net)

    cv2.imshow("RNN", img)
    # Visualize best network's Pheotype
    net = EANT.NeuralNetwork()
    pop.Species[0].GetLeader().BuildEANTPheno
type(net, substrate, params)
        img = np.zeros((500, 500, 3),
dtype=np.uint8)
    img += 10

    EANT.DrawPhenotype(img, (0, 0, 500, 500),
net, substrate=True)
    cv2.imshow("NN", img)
    cv2.waitKey(1)

    if max(fitnesses) > 15.0:
        break

    # Epoch
    generations = generation
    pop.Epoch()

return generations

```

**ДОДАТОК Б**

Відомість атестаційної роботи



