A project report on

Intelligent MultiAgent and Expert Systems

ARTIFICIAL LIFE SIMULATOR

Submitted in partial fulfillment for the award of the degree of

B-TECH COMPUTER SCIENCE WITH AI & ML

by

AVANTHIKA RAJESH- 20BAI1144

SHREE SMITI – 20BAI1179

VINITHI K-20BAI1071

SCOPE

NOVEMBER ,2022

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

ABSTRACT

The field of AL attempts to understand life by creating artificial systems that exhibit life's properties through bottom-up mechanisms.

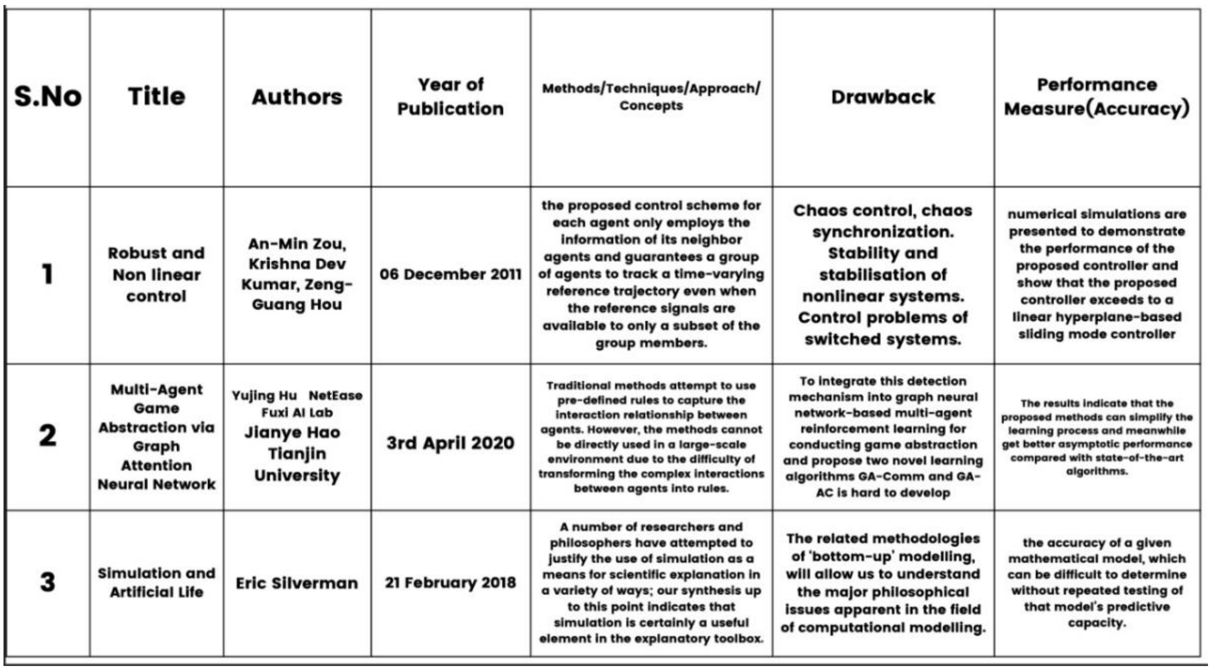
This is similar to how AI attempts to re-create and explore the properties of intelligent behavior through artificial systems. Artificial life has now become a mature inter-discipline. In this contribution, its roots are traced, its key questions are raised, its main methodological tools are discussed, and finally its applications are reviewed. As part of the growing body of knowledge at the intersection between the life sciences and computing, artificial life will continue to thrive and benefit from further scientific and technical progress on both sides, the biological and the computational. It is expected to take center stage in natural computing. The field of AL attempts to understand life by creating artificial systems that exhibit life's properties through bottom-up mechanisms. This is similar to how AI attempts to re-create and explore the properties of intelligent behavior through artificial systems. Artificial life has now become a mature inter-discipline. In this contribution, its roots are traced, its key questions are raised, its main methodological tools are discussed, and finally its applications are reviewed. As part of the growing body of knowledge at the intersection between the life sciences and computing, artificial life will continue to thrive and benefit from further scientific and technical progress on both sides, the biological and the computational. It is expected to take center stage in natural computing. Simulation-based learning can be the way to develop health professionals' knowledge, skills, and attitudes, whilst protecting patients from unnecessary risks. Simulation-based medical education can be a platform which provides a valuable tool in learning to mitigate ethical tensions and resolve practical dilemmas. Simulation-based training techniques, tools, and strategies can be applied in designing structured learning experiences, as well as be used as a measurement tool linked to targeted teamwork competencies and learning objectives. It has been widely applied in fields such aviation and the military. The field of AL attempts to understand life by creating artificial systems that exhibit life's properties through bottom-up mechanisms. This is similar to how AI attempts to re-create and explore the properties of intelligent behavior through artificial systems.

INTRODUCTION

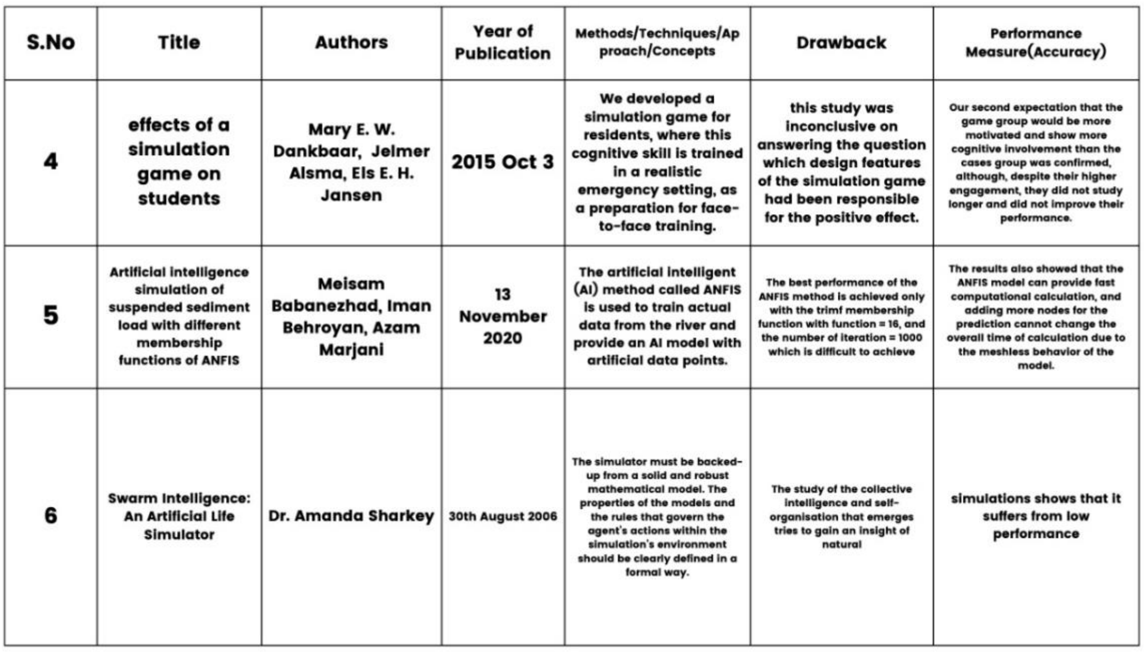
Artificial life studies the fundamental processes of living systems in artificial environments in order to gain a deeper understanding of the complex information processing that define such systems. These topics are broad, but often include evolutionary dynamics, emergent properties of collective systems, biomimicry, as well as related issues about the philosophy of the nature of life and the use of lifelike properties in artistic works. The objective of this project is to study evolution process in a computer-simulated artificial world. We hope that, like in the real world, despite randomness and aimlessness of basic evolution mechanisms, it will lead to creation of more and more efficient artificial organisms, still better and better adapted to the artificial world conditions.

For many years people used computers to simulate Nature. This kind of research also belongs to a field of science called "Artificial Life" (AL). ALife is the bottom-up scientific study of the fundamental principles of life. Just as Artificial Intelligence researchers ponder the nature of intelligence by trying to build intelligent systems from scratch, ALife researchers investigate the nature of “life” by trying to build living systems from scratch. Our academic field naturally overlaps with biology and chemistry, but also computer science, astrobiology, physics, complex systems, network sciences, geology, evolutionary science, origins of life research, and of course AI and animal behavior studies. The name is obviously connected with "Artificial Intelligence" – both fields of study partly overlap, but AL has more in common with biology and physics. It might be called a branch of biology, because we study living (or "living", depending on the definition of this word) organisms in an environment. No matter that the environment is an artificial, virtual world inside a computer: philosophers haven't yet decided whether our world is real and, nevertheless, biologists keep examining its living organisms.

Literature Review



S.No	Title	Authors	Year of Publication	Methods/Techniques/Approach/ Concepts	Drawback	Performance Measure(Accuracy)
1	Robust and Non linear control	An-Min Zou, Krishna Dev Kumar, Zeng-Guang Hou	06 December 2011	the proposed control scheme for each agent only employs the information of its neighbor agents and guarantees a group of agents to track a time-varying reference trajectory even when the reference signals are available to only a subset of the group members.	Chaos control, chaos synchronization. Stability and stabilisation of nonlinear systems. Control problems of switched systems.	numerical simulations are presented to demonstrate the performance of the proposed controller and show that the proposed controller exceeds to a linear hyperplane-based sliding mode controller
2	Multi-Agent Game Abstraction via Graph Attention Neural Network	Yujing Hu NetEase Fuxi AI Lab Jianye Hao Tianjin University	3rd April 2020	Traditional methods attempt to use pre-defined rules to capture the interaction relationship between agents. However, the methods cannot be directly used in a large-scale environment due to the difficulty of transforming the complex interactions between agents into rules.	To integrate this detection mechanism into graph neural network-based multi-agent reinforcement learning for conducting game abstraction and propose two novel learning algorithms GA-Comm and GA-AC is hard to develop	The results indicate that the proposed methods can simplify the learning process and meanwhile get better asymptotic performance compared with state-of-the-art algorithms.
3	Simulation and Artificial Life	Eric Silverman	21 February 2018	A number of researchers and philosophers have attempted to justify the use of simulation as a means for scientific explanation in a variety of ways; our synthesis up to this point indicates that simulation is certainly a useful element in the explanatory toolbox.	The related methodologies of 'bottom-up' modelling, will allow us to understand the major philosophical issues apparent in the field of computational modelling.	the accuracy of a given mathematical model, which can be difficult to determine without repeated testing of that model's predictive capacity.



S.No	Title	Authors	Year of Publication	Methods/Techniques/Approach/Concepts	Drawback	Performance Measure(Accuracy)
4	effects of a simulation game on students	Mary E. W. Dankbaar, Jelmer Alsma, Els E. H. Jansen	2015 Oct 3	We developed a simulation game for residents, where this cognitive skill is trained in a realistic emergency setting, as a preparation for face-to-face training.	this study was inconclusive on answering the question which design features of the simulation game had been responsible for the positive effect.	Our second expectation that the game group would be more motivated and show more cognitive involvement than the cases group was confirmed, although, despite their higher engagement, they did not study longer and did not improve their performance.
5	Artificial intelligence simulation of suspended sediment load with different membership functions of ANFIS	Meisam Babanezhad, Iman Behroyan, Azam Marjani	13 November 2020	The artificial intelligent (AI) method called ANFIS is used to train actual data from the river and provide an AI model with artificial data points.	The best performance of the ANFIS method is achieved only with the trimf membership function with function = 16, and the number of iteration = 1000 which is difficult to achieve	The results also showed that the ANFIS model can provide fast computational calculation, and adding more nodes for the prediction cannot change the overall time of calculation due to the meshless behavior of the model.
6	Swarm Intelligence: An Artificial Life Simulator	Dr. Amanda Sharkey	30th August 2006	The simulator must be backed-up from a solid and robust mathematical model. The properties of the models and the rules that govern the agent's actions within the simulation's environment should be clearly defined in a formal way.	The study of the collective intelligence and self-organisation that emerges tries to gain an insight of natural	simulations shows that it suffers from low performance

PROPOSED METHODOLOGY

PROBLEM STATEMENT:
One of the main goals of this project is to evolve agents that are adapted to survive in a particular environment.
We have to achieve this goal for simple environments where the project is required to find a balance between movement and resource consumption.
Preliminary results should show that the evolved agent is typically better than a randomly generated agent with statistical analysis

Methodology

User can easily modify the system through function calls to a well-documented API and can design an environment by distributing resources across a 2D grid.
User can save statistics and aspects of a simulation for further analysis.
User interface is easy to navigate and visualize the resource levels in the environment, and the distribution of creatures in the environment.

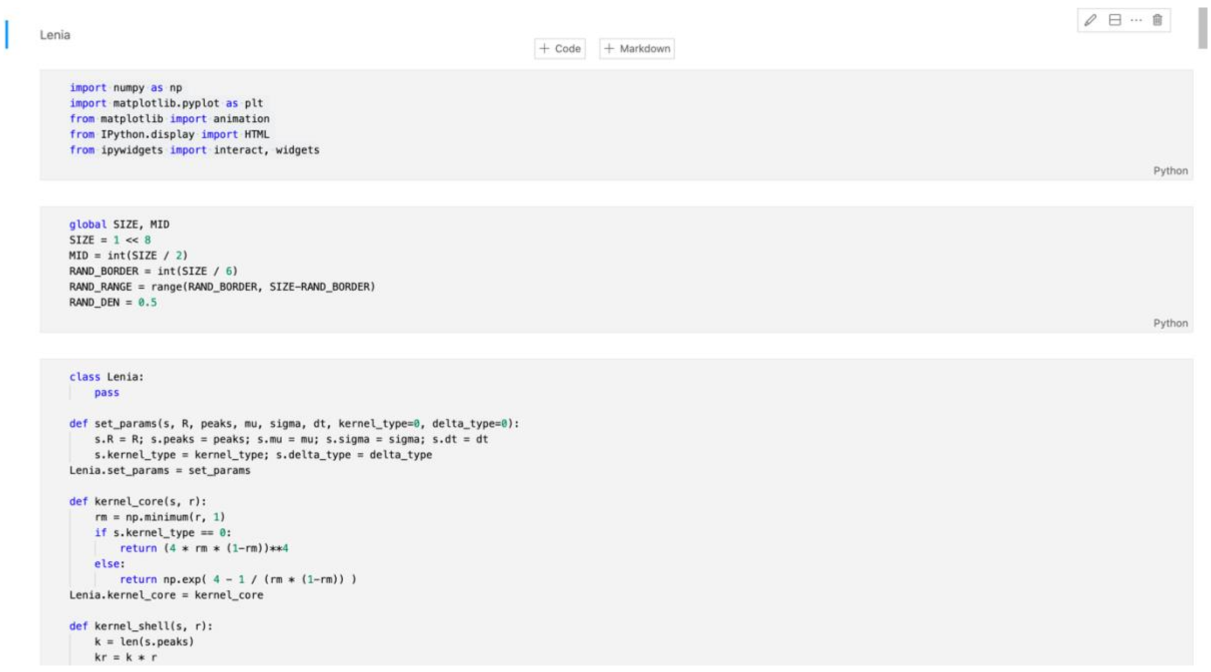
Software Requirements:
C# thread
Unity MonoBehavior script

The Algorithms we used for our Artificial Simulator are:-

LENIA:

A new system of artificial life called Lenia (from Latin lenis “smooth”), a two-dimensional cellular automaton with continuous spacetime state and generalized local rule, is reported. Computer simulations show that Lenia supports a great diversity of complex autonomous patterns or “life forms” bearing resemblance to real-world microscopic organisms. More than 400 species in 18 families have been identified, many discovered via interactive evolutionary computation. They differ from other cellular automata patterns in being geometric, metameric, fuzzy, resilient, adaptive and rule generic.

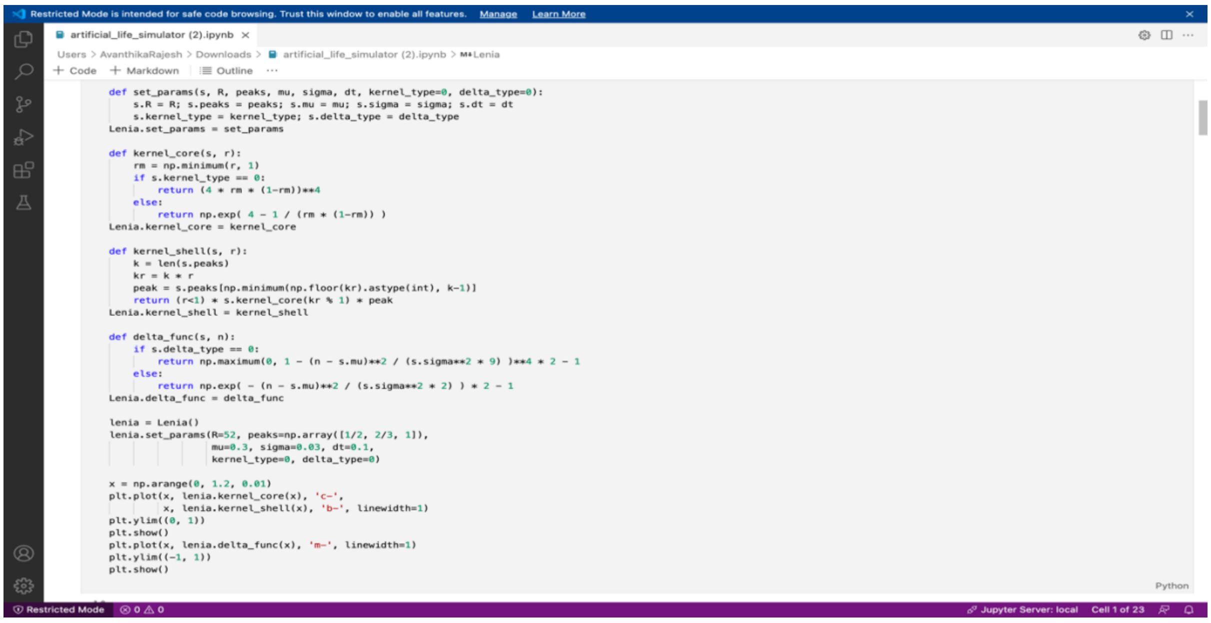
Basic observations of the system are presented regarding the properties of spacetime and basic settings. A broad survey of the life forms is provided and categorized into a hierarchical taxonomy, and their distribution is mapped in the parameter hyperspace. Their morphological structures and behavioral dynamics are described, and possible mechanisms of their self-organization, self-direction and plasticity are proposed.



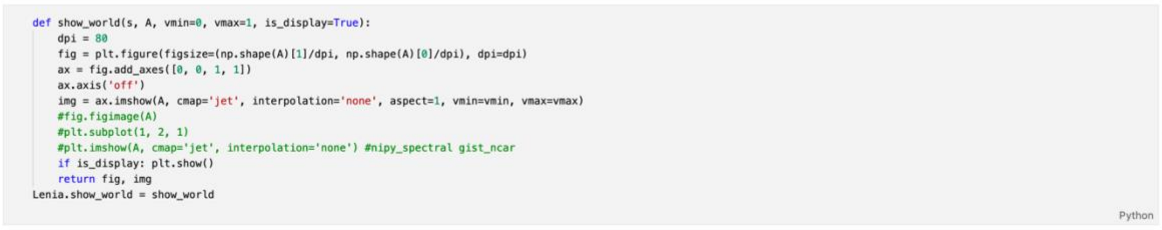
```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation
from IPython.display import HTML
from ipywidgets import interact, widgets
```

```python
global SIZE, MID
SIZE = 1 << 8
MID = int(SIZE / 2)
RAND_BORDER = int(SIZE / 6)
RAND_RANGE = range(RAND_BORDER, SIZE-RAND_BORDER)
RAND_DEN = 0.5
```

```python
class Lenia:
    pass

def set_params(s, R, peaks, mu, sigma, dt, kernel_type=0, delta_type=0):
    s.R = R; s.peaks = peaks; s.mu = mu; s.sigma = sigma; s.dt = dt
    s.kernel_type = kernel_type; s.delta_type = delta_type
Lenia.set_params = set_params

def kernel_core(s, r):
    rm = np.minimum(r, 1)
    if s.kernel_type == 0:
        return (4 * rm * (1-rm))**4
    else:
        return np.exp( 4 - 1 / (rm * (1-rm)) )
Lenia.kernel_core = kernel_core

def kernel_shell(s, r):
    k = len(s.peaks)
    kr = k * r
```


Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
artificial_life_simulator (2).ipynb
Users > AvanthikaRajesh > Downloads > artificial_life_simulator (2).ipynb > M↓Lenia
+ Code + Markdown Outline ···
Python
Restricted Mode ⊗ 0 ⚠ 0
Jupyter Server: local Cell 1 of 23

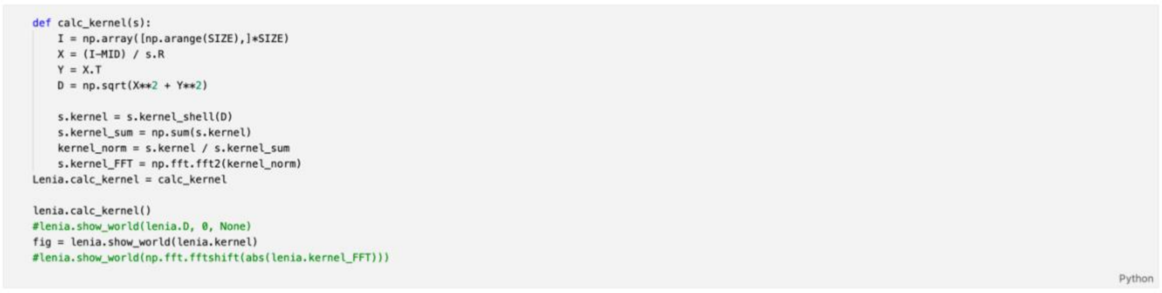
```python
def set_params(s, R, peaks, mu, sigma, dt, kernel_type=0, delta_type=0):
    s.R = R; s.peaks = peaks; s.mu = mu; s.sigma = sigma; s.dt = dt
    s.kernel_type = kernel_type; s.delta_type = delta_type
Lenia.set_params = set_params

def kernel_core(s, r):
    rm = np.minimum(r, 1)
    if s.kernel_type == 0:
        return (4 * rm * (1-rm))**4
    else:
        return np.exp( 4 - 1 / (rm * (1-rm)) )
Lenia.kernel_core = kernel_core

def kernel_shell(s, r):
    k = len(s.peaks)
    kr = k * r
    peak = s.peaks[np.minimum(np.floor(kr).astype(int), k-1)]
    return (r<1) * s.kernel_core(kr % 1) * peak
Lenia.kernel_shell = kernel_shell

def delta_func(s, n):
    if s.delta_type == 0:
        return np.maximum(0, 1 - (n - s.mu)**2 / (s.sigma**2 * 9) )**4 * 2 - 1
    else:
        return np.exp( - (n - s.mu)**2 / (s.sigma**2 * 2) ) * 2 - 1
Lenia.delta_func = delta_func

lenia = Lenia()
lenia.set_params(R=52, peaks=np.array([1/2, 2/3, 1]),
                 mu=0.3, sigma=0.03, dt=0.1,
                 kernel_type=0, delta_type=0)

x = np.arange(0, 1.2, 0.01)
plt.plot(x, lenia.kernel_core(x), 'c-',
         x, lenia.kernel_shell(x), 'b-', linewidth=1)
plt.ylim((0, 1))
plt.show()
plt.plot(x, lenia.delta_func(x), 'm-', linewidth=1)
plt.ylim((-1, 1))
plt.show()
```



```python
def show_world(s, A, vmin=0, vmax=1, is_display=True):
    dpi = 80
    fig = plt.figure(figsize=(np.shape(A)[1]/dpi, np.shape(A)[0]/dpi), dpi=dpi)
    ax = fig.add_axes([0, 0, 1, 1])
    ax.axis('off')
    img = ax.imshow(A, cmap='jet', interpolation='none', aspect=1, vmin=vmin, vmax=vmax)
    #fig.figimage(A)
    #plt.subplot(1, 2, 1)
    #plt.imshow(A, cmap='jet', interpolation='none') #nipy_spectral gist_ncar
    if is_display: plt.show()
    return fig, img
Lenia.show_world = show_world
```

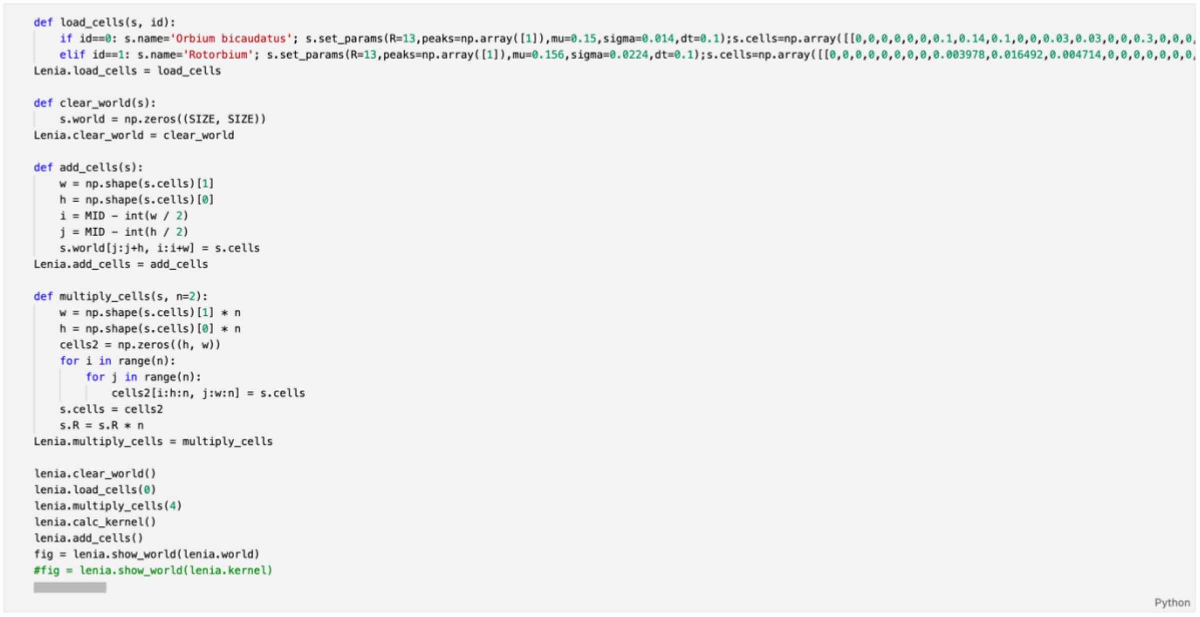
Python

Python



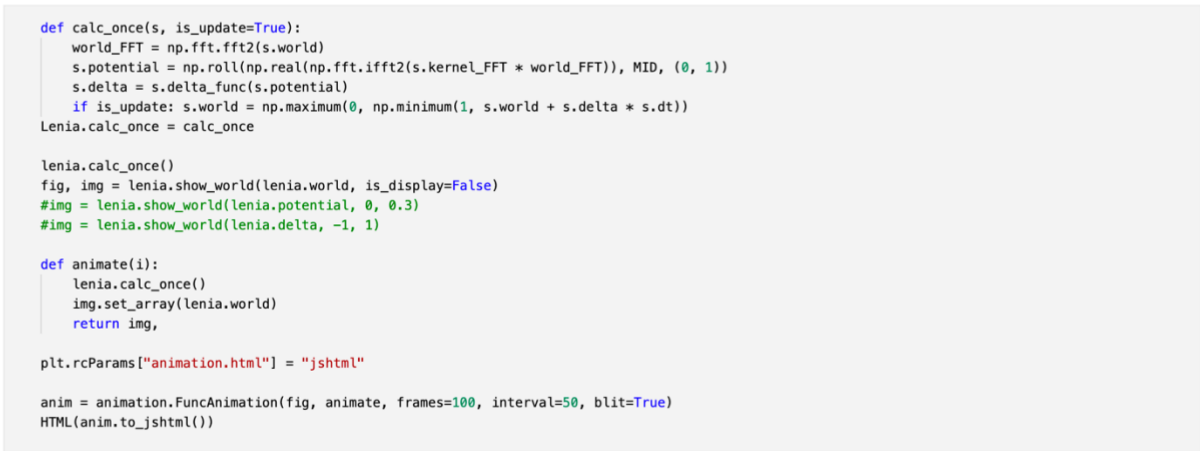
```python
def calc_kernel(s):
    I = np.array([np.arange(SIZE),]*SIZE)
    X = (I-MID) / s.R
    Y = X.T
    D = np.sqrt(X**2 + Y**2)

    s.kernel = s.kernel_shell(D)
    s.kernel_sum = np.sum(s.kernel)
    kernel_norm = s.kernel / s.kernel_sum
    s.kernel_FFT = np.fft.fft2(kernel_norm)
Lenia.calc_kernel = calc_kernel

lenia.calc_kernel()
#lenia.show_world(lenia.D, 0, None)
fig = lenia.show_world(lenia.kernel)
#lenia.show_world(np.fft.fftshift(abs(lenia.kernel_FFT)))
```

Python



```python
def load_cells(s, id):
    if id==0: s.name='Orbium bicaudatus'; s.set_params(R=13,peaks=np.array([1]),mu=0.15,sigma=0.014,dt=0.1);s.cells=np.array([[0,0,0,0,0,0,0.1,0.14,0.1,0,0,0.03,0.03,0,0,0.3,0,0,0,
    elif id==1: s.name='Rotorbium'; s.set_params(R=13,peaks=np.array([1]),mu=0.156,sigma=0.0224,dt=0.1);s.cells=np.array([[0,0,0,0,0,0,0,0.003978,0.016492,0.004714,0,0,0,0,0,0,0,
Lenia.load_cells = load_cells

def clear_world(s):
    s.world = np.zeros((SIZE, SIZE))
Lenia.clear_world = clear_world

def add_cells(s):
    w = np.shape(s.cells)[1]
    h = np.shape(s.cells)[0]
    i = MID - int(w / 2)
    j = MID - int(h / 2)
    s.world[j:j+h, i:i+w] = s.cells
Lenia.add_cells = add_cells

def multiply_cells(s, n=2):
    w = np.shape(s.cells)[1] * n
    h = np.shape(s.cells)[0] * n
    cells2 = np.zeros((h, w))
    for i in range(n):
        for j in range(n):
            cells2[i:h:n, j:w:n] = s.cells
    s.cells = cells2
    s.R = s.R * n
Lenia.multiply_cells = multiply_cells

lenia.clear_world()
lenia.load_cells(0)
lenia.multiply_cells(4)
lenia.calc_kernel()
lenia.add_cells()
fig = lenia.show_world(lenia.world)
#fig = lenia.show_world(lenia.kernel)
```

Python



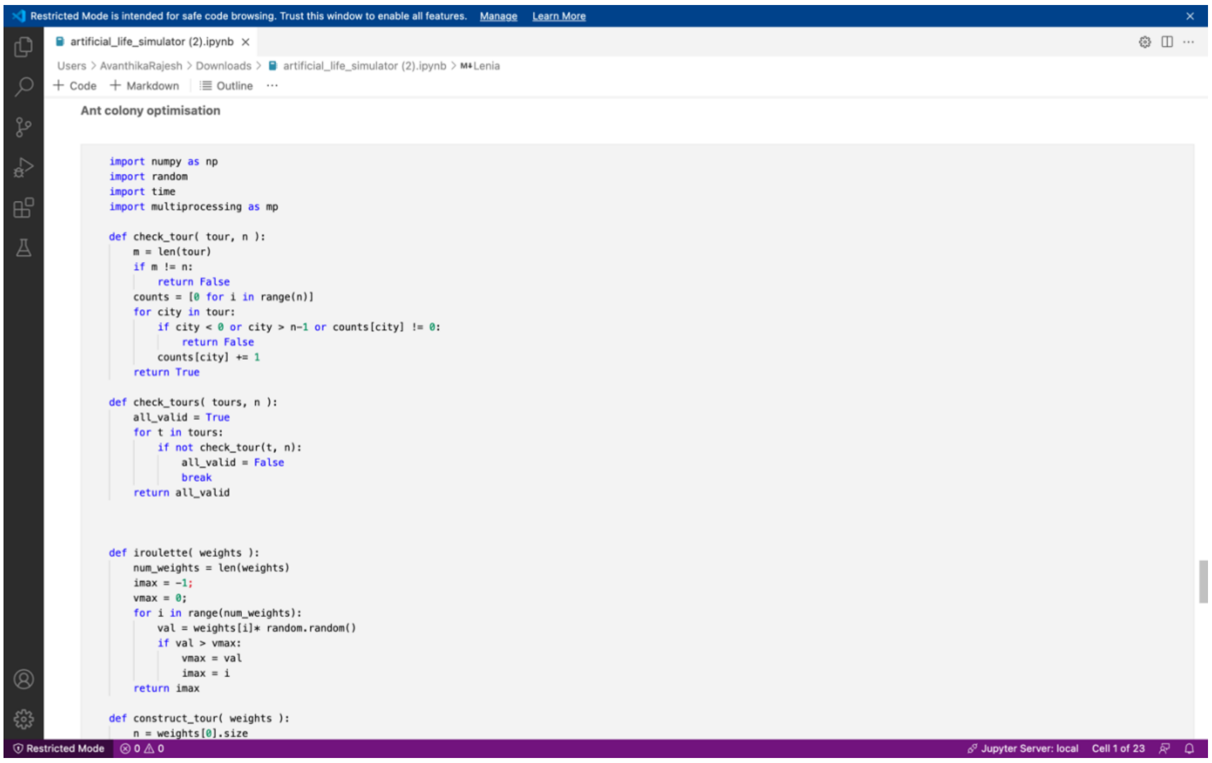
```python
def calc_once(s, is_update=True):
    world_FFT = np.fft.fft2(s.world)
    s.potential = np.roll(np.real(np.fft.ifft2(s.kernel_FFT * world_FFT)), MID, (0, 1))
    s.delta = s.delta_func(s.potential)
    if is_update: s.world = np.maximum(0, np.minimum(1, s.world + s.delta * s.dt))
Lenia.calc_once = calc_once

lenia.calc_once()
fig, img = lenia.show_world(lenia.world, is_display=False)
#img = lenia.show_world(lenia.potential, 0, 0.3)
#img = lenia.show_world(lenia.delta, -1, 1)

def animate(i):
    lenia.calc_once()
    img.set_array(lenia.world)
    return img,

plt.rcParams["animation.html"] = "jshtml"

anim = animation.FuncAnimation(fig, animate, frames=100, interval=50, blit=True)
HTML(anim.to_jshtml())
```

```python
def run_lenia(animal_id):
    def run_animate(i):
        lenia.calc_once()
        img.set_array(lenia.world)
        return img,

    lenia = Lenia()
    lenia.clear_world()
    lenia.load_cells(animal_id)
    lenia.multiply_cells(4)
    lenia.calc_kernel()
    lenia.add_cells()
    fig, img = lenia.show_world(lenia.world, is_display=False)
    return animation.FuncAnimation(fig, run_animate, frames=200, interval=50, blit=True)

i = interact(run_lenia, animal_id={
    'Orbium bicaudatus':0,
    'Rotorbium':1})
```

ANT-COLONY METHOD:

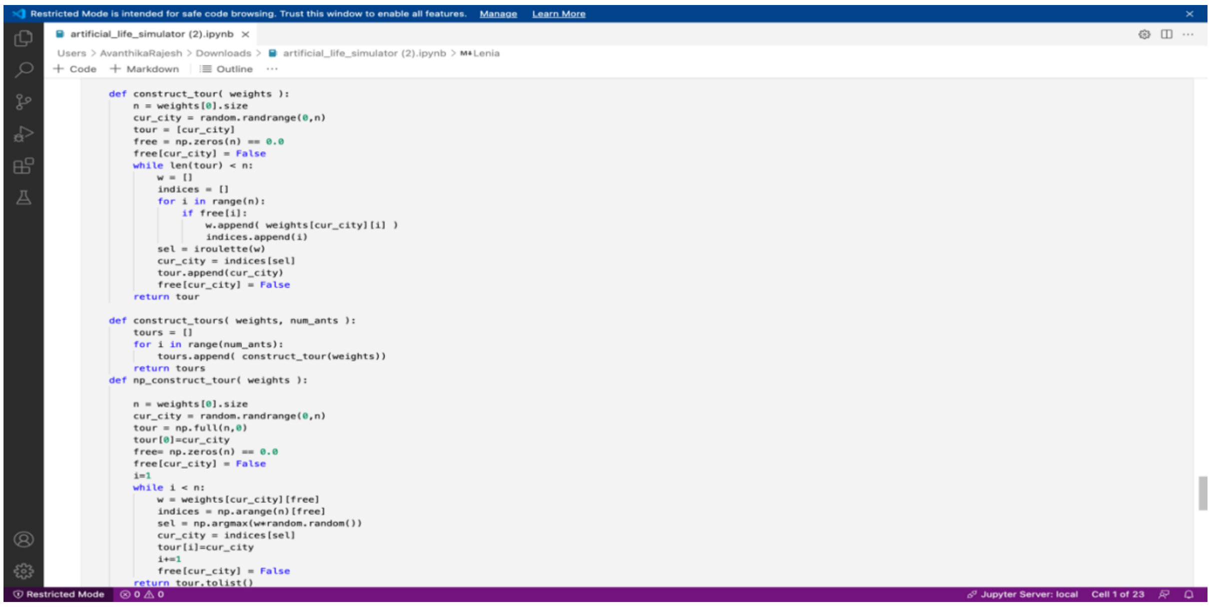
Ant colony optimization is a probabilistic technique for finding optimal paths. In computer science and researches, the ant colony optimization algorithm is used for solving different computational problems. This algorithm is introduced based on the foraging behavior of an ant for seeking a path between their colony and source food. Initially, it was used to solve the well-known traveling salesman problem. Later, it is used for solving different hard optimization problems. Ant Colony Optimization technique is purely inspired from the foraging behaviour of ant colonies. Ants are eusocial insects that prefer community survival and sustaining rather than as individual species. They communicate with each other using sound, touch and pheromone. Pheromones are organic chemical compounds secreted by the ants that trigger a social response in members of same species. These are chemicals capable of acting like hormones outside the body of the secreting individual, to impact the behaviour of the receiving individuals. Since most ants live on the ground, they use the soil surface to leave pheromone trails that may be followed (smelled) by other ants.

Ants live in community nests and the underlying principle of ACO is to observe the movement of the ants from their nests in order to search for food in the shortest possible path. Initially, ants start to move randomly in search of food around their nests. This randomized search opens up multiple routes from the nest to the food source. Now, based on the quality and quantity of the food, ants carry a portion of the food back with necessary pheromone concentration on its return path. Depending on these pheromone trials, the probability of selection of a specific path by the following ants would be a guiding factor to the food source. Evidently, this probability is based on the concentration as well as the rate of evaporation of pheromone. It can also be observed that since the evaporation rate of pheromone is also a deciding factor, the length of each path can easily be accounted for.

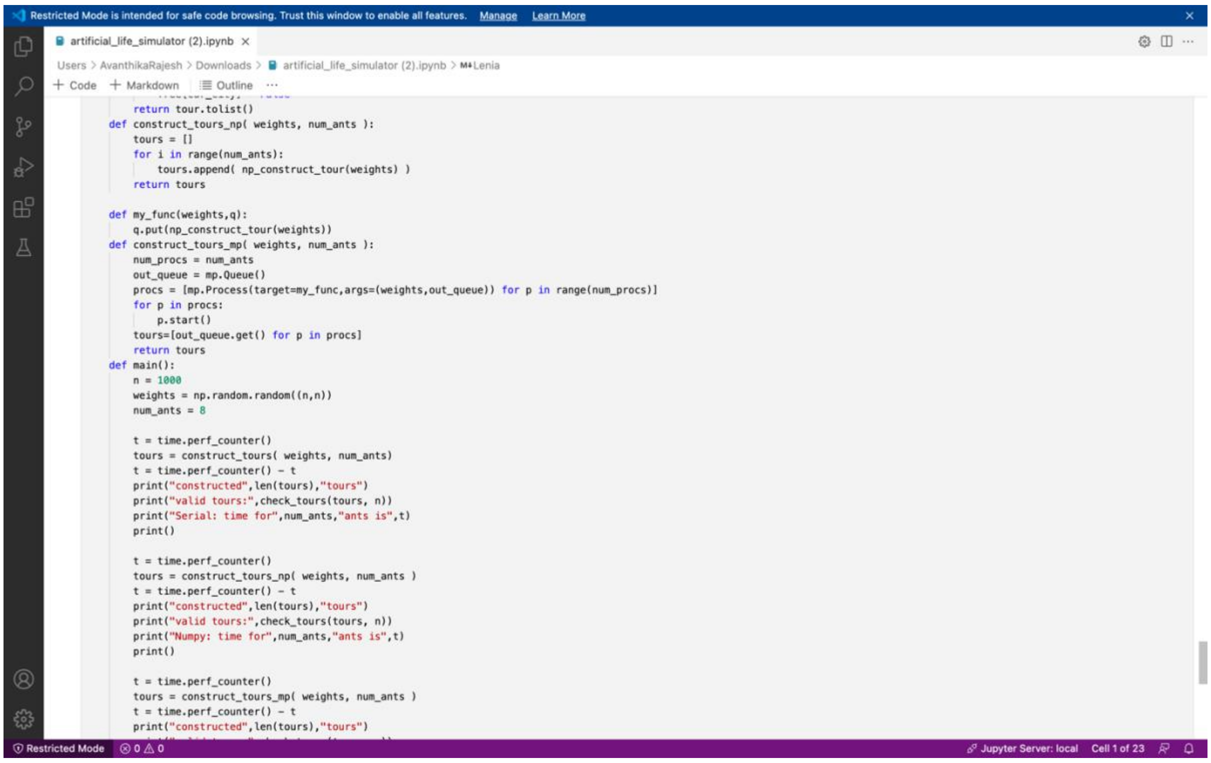

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
artificial_life_simulator (2).ipynb
Users > AvanthikaRajesh > Downloads > artificial_life_simulator (2).ipynb > M↓Lenia
+ Code + Markdown Outline ···
Restricted Mode ⊗ 0 ⚠ 0
Jupyter Server: local Cell 1 of 23

Ant colony optimisation

```python
import numpy as np
import random
import time
import multiprocessing as mp

def check_tour( tour, n ):
    m = len(tour)
    if m != n:
        return False
    counts = [0 for i in range(n)]
    for city in tour:
        if city < 0 or city > n-1 or counts[city] != 0:
            return False
        counts[city] += 1
    return True

def check_tours( tours, n ):
    all_valid = True
    for t in tours:
        if not check_tour(t, n):
            all_valid = False
            break
    return all_valid



def iroulette( weights ):
    num_weights = len(weights)
    imax = -1;
    vmax = 0;
    for i in range(num_weights):
        val = weights[i]* random.random()
        if val > vmax:
            vmax = val
            imax = i
    return imax

def construct_tour( weights ):
    n = weights[0].size
```


Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
artificial_life_simulator (2).ipynb
Users > AvanthikaRajesh > Downloads > artificial_life_simulator (2).ipynb > M↓Lenia
+ Code + Markdown Outline ···
Restricted Mode ⊗ 0 ⚠ 0
Jupyter Server: local Cell 1 of 23

```python
def construct_tour( weights ):
    n = weights[0].size
    cur_city = random.randrange(0,n)
    tour = [cur_city]
    free = np.zeros(n) == 0.0
    free[cur_city] = False
    while len(tour) < n:
        w = []
        indices = []
        for i in range(n):
            if free[i]:
                w.append( weights[cur_city][i] )
                indices.append(i)
        sel = iroulette(w)
        cur_city = indices[sel]
        tour.append(cur_city)
        free[cur_city] = False
    return tour

def construct_tours( weights, num_ants ):
    tours = []
    for i in range(num_ants):
        tours.append( construct_tour(weights))
    return tours
def np_construct_tour( weights ):

    n = weights[0].size
    cur_city = random.randrange(0,n)
    tour = np.full(n,0)
    tour[0]=cur_city
    free= np.zeros(n) == 0.0
    free[cur_city] = False
    i=1
    while i < n:
        w = weights[cur_city][free]
        indices = np.arange(n)[free]
        sel = np.argmax(w*random.random())
        cur_city = indices[sel]
        tour[i]=cur_city
        i+=1
        free[cur_city] = False
    return tour.tolist()
```

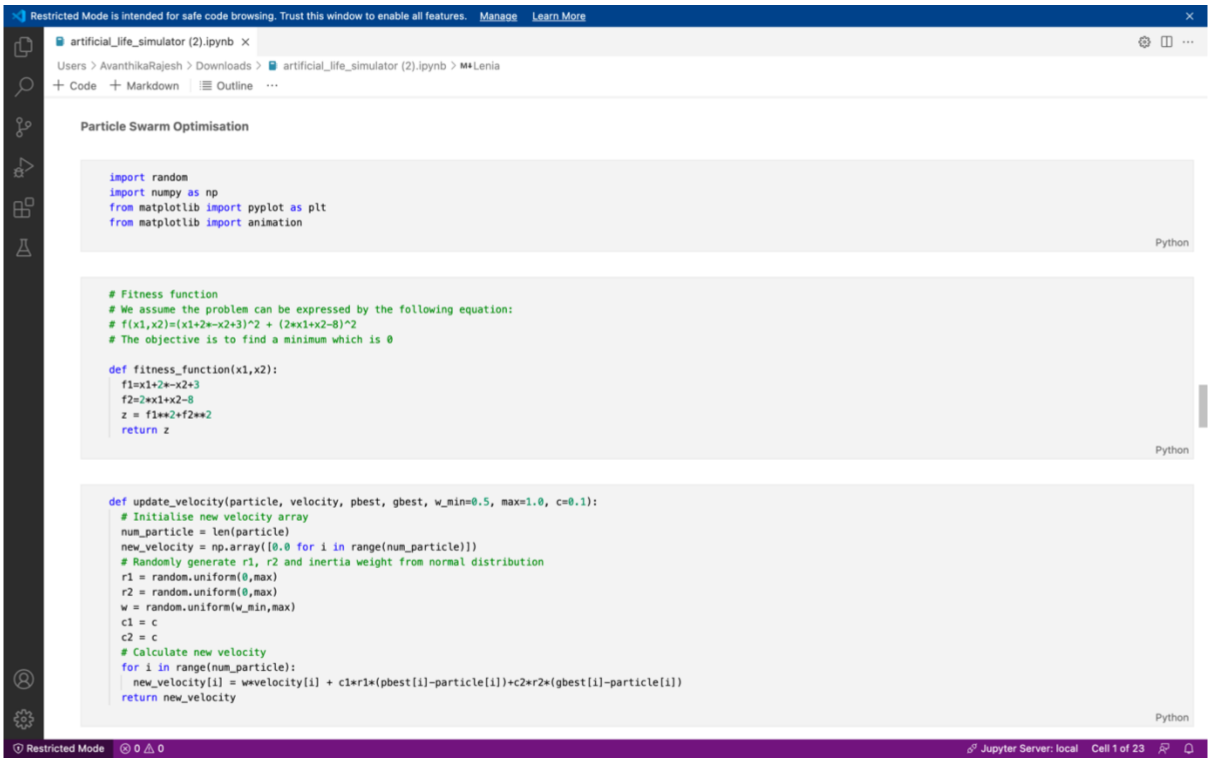

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
artificial_life_simulator (2).ipynb
Users > AvanthikaRajesh > Downloads > artificial_life_simulator (2).ipynb > M↓Lenia
+ Code + Markdown ☰ Outline ···
Restricted Mode ⊗ 0 ⚠ 0
Jupyter Server: local Cell 1 of 23

```python
        return tour.tolist()
    def construct_tours_np( weights, num_ants ):
        tours = []
        for i in range(num_ants):
            tours.append( np_construct_tour(weights) )
        return tours

    def my_func(weights,q):
        q.put(np_construct_tour(weights))
    def construct_tours_mp( weights, num_ants ):
        num_procs = num_ants
        out_queue = mp.Queue()
        procs = [mp.Process(target=my_func,args=(weights,out_queue)) for p in range(num_procs)]
        for p in procs:
            p.start()
        tours=[out_queue.get() for p in procs]
        return tours
    def main():
        n = 1000
        weights = np.random.random((n,n))
        num_ants = 8

        t = time.perf_counter()
        tours = construct_tours( weights, num_ants)
        t = time.perf_counter() - t
        print("constructed",len(tours),"tours")
        print("valid tours:",check_tours(tours, n))
        print("Serial: time for",num_ants,"ants is",t)
        print()

        t = time.perf_counter()
        tours = construct_tours_np( weights, num_ants )
        t = time.perf_counter() - t
        print("constructed",len(tours),"tours")
        print("valid tours:",check_tours(tours, n))
        print("Numpy: time for",num_ants,"ants is",t)
        print()

        t = time.perf_counter()
        tours = construct_tours_mp( weights, num_ants )
        t = time.perf_counter() - t
        print("constructed",len(tours),"tours")
```

PARTICLE SWARM OPTIMIZATION:

Particle swarm optimization is a population-based stochastic optimization algorithm motivated by intelligent collective behavior of some animals such as flocks of birds or schools of fish. Since presented in 1995, it has experienced a multitude of enhancements. As researchers have learned about the technique, they derived new versions aiming to different demands, developed new applications in a host of areas, published theoretical studies of the effects of the various parameters and proposed many variants of the algorithm. Then, we analyze its present situation of research and application in algorithm structure, parameter selection, topology structure, discrete PSO algorithm and parallel PSO algorithm, multi-objective optimization PSO and its engineering applications. Finally, the existing problems are analyzed and future research directions are presented. PSO algorithm simulates animal's social behavior, including insects, herds, birds and fishes. These swarms conform a cooperative way to find food, and each member in the swarms keeps changing the search pattern according to the learning experiences of its own and other members.

Main design idea of the PSO algorithm is closely related to two researches: One is evolutionary algorithm, just like evolutionary algorithm; PSO also uses a swarm mode which makes it to simultaneously search large region in the solution space of the optimized objective function.


Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

artificial_life_simulator (2).ipynb

Users > AvanthikaRajesh > Downloads > artificial_life_simulator (2).ipynb > M↓Lenia

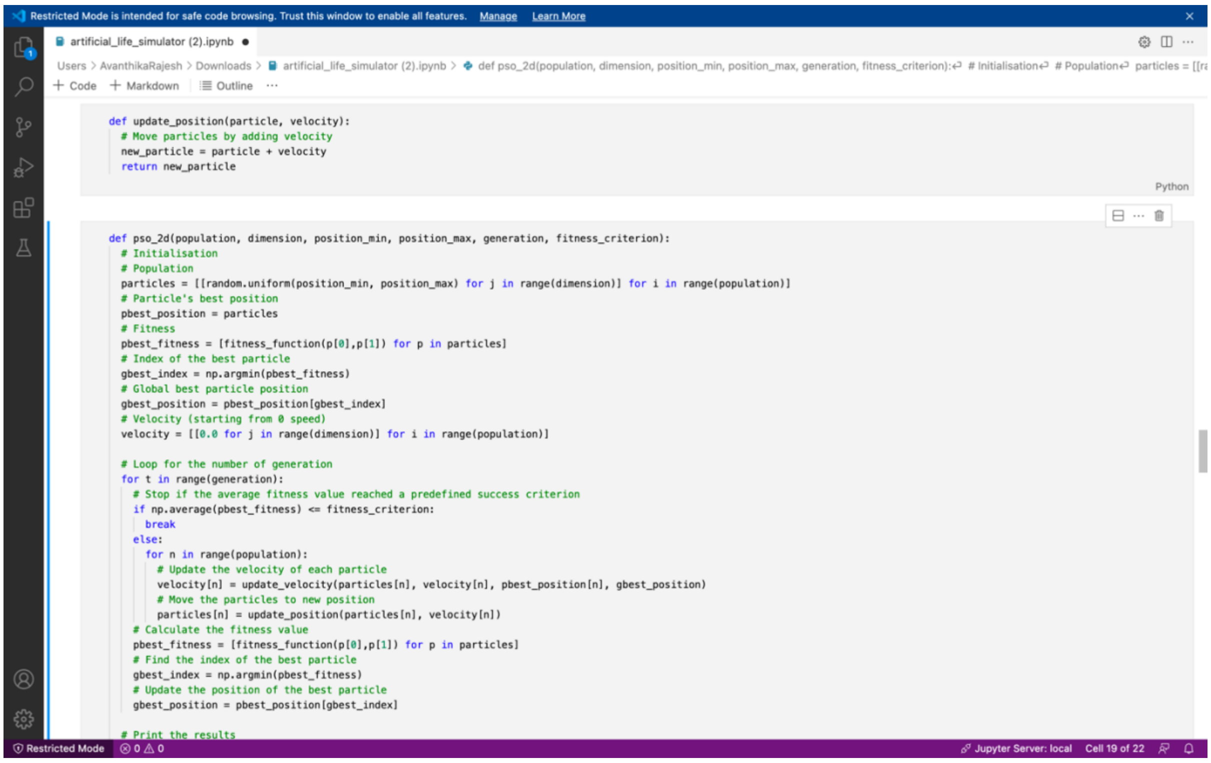
Particle Swarm Optimisation

```python
import random
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import animation
```

```python
# Fitness function
# We assume the problem can be expressed by the following equation:
# f(x1,x2)=(x1+2*-x2+3)^2 + (2*x1+x2-8)^2
# The objective is to find a minimum which is 0

def fitness_function(x1,x2):
  f1=x1+2*-x2+3
  f2=2*x1+x2-8
  z = f1**2+f2**2
  return z
```

```python
def update_velocity(particle, velocity, pbest, gbest, w_min=0.5, max=1.0, c=0.1):
  # Initialise new velocity array
  num_particle = len(particle)
  new_velocity = np.array([0.0 for i in range(num_particle)])
  # Randomly generate r1, r2 and inertia weight from normal distribution
  r1 = random.uniform(0,max)
  r2 = random.uniform(0,max)
  w = random.uniform(w_min,max)
  c1 = c
  c2 = c
  # Calculate new velocity
  for i in range(num_particle):
    new_velocity[i] = w*velocity[i] + c1*r1*(pbest[i]-particle[i])+c2*r2*(gbest[i]-particle[i])
  return new_velocity
```

Restricted Mode ⊗ 0 ⚠ 0 Jupyter Server: local Cell 1 of 23


Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

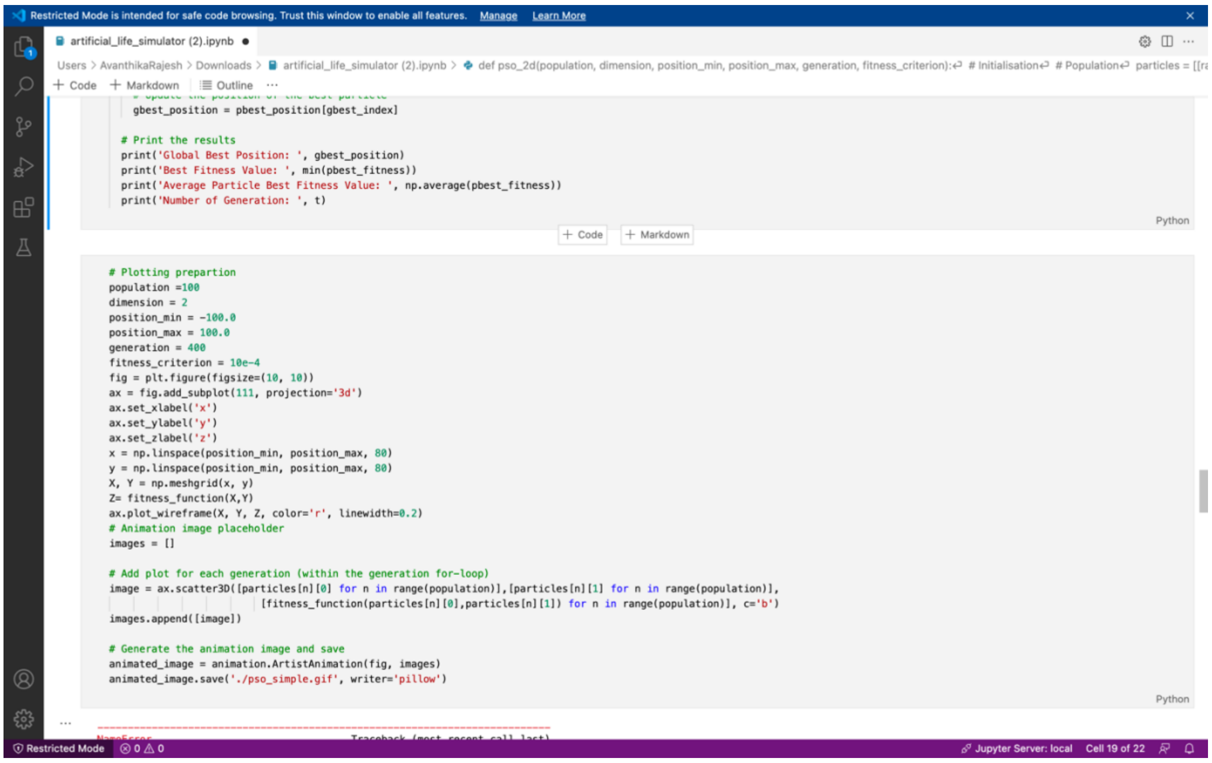
artificial_life_simulator (2).ipynb

Users > AvanthikaRajesh > Downloads > artificial_life_simulator (2).ipynb > def pso_2d(population, dimension, position_min, position_max, generation, fitness_criterion):↵ # Initialisation↵ # Population↵ particles = [[ra

```python
def update_position(particle, velocity):
  # Move particles by adding velocity
  new_particle = particle + velocity
  return new_particle
```

```python
def pso_2d(population, dimension, position_min, position_max, generation, fitness_criterion):
  # Initialisation
  # Population
  particles = [[random.uniform(position_min, position_max) for j in range(dimension)] for i in range(population)]
  # Particle's best position
  pbest_position = particles
  # Fitness
  pbest_fitness = [fitness_function(p[0],p[1]) for p in particles]
  # Index of the best particle
  gbest_index = np.argmin(pbest_fitness)
  # Global best particle position
  gbest_position = pbest_position[gbest_index]
  # Velocity (starting from 0 speed)
  velocity = [[0.0 for j in range(dimension)] for i in range(population)]

  # Loop for the number of generation
  for t in range(generation):
    # Stop if the average fitness value reached a predefined success criterion
    if np.average(pbest_fitness) <= fitness_criterion:
      break
    else:
      for n in range(population):
        # Update the velocity of each particle
        velocity[n] = update_velocity(particles[n], velocity[n], pbest_position[n], gbest_position)
        # Move the particles to new position
        particles[n] = update_position(particles[n], velocity[n])
    # Calculate the fitness value
    pbest_fitness = [fitness_function(p[0],p[1]) for p in particles]
    # Find the index of the best particle
    gbest_index = np.argmin(pbest_fitness)
    # Update the position of the best particle
    gbest_position = pbest_position[gbest_index]

  # Print the results
```

Restricted Mode ⊗ 0 ⚠ 0 Jupyter Server: local Cell 19 of 22



Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

artificial_life_simulator (2).ipynb

Users > AvanthikaRajesh > Downloads > artificial_life_simulator (2).ipynb > def pso_2d(population, dimension, position_min, position_max, generation, fitness_criterion):↩ # Initialisation↩ # Population↩ particles = [[r

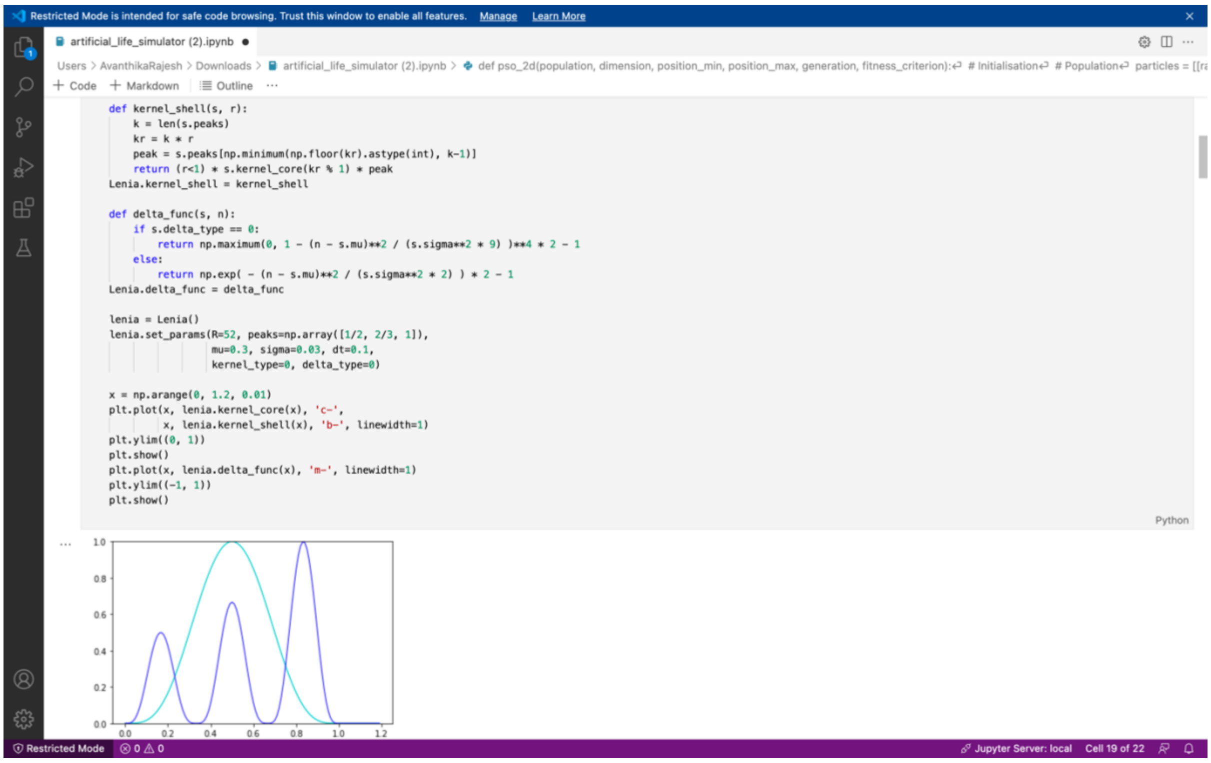
```python
        gbest_position = pbest_position[gbest_index]

    # Print the results
    print('Global Best Position: ', gbest_position)
    print('Best Fitness Value: ', min(pbest_fitness))
    print('Average Particle Best Fitness Value: ', np.average(pbest_fitness))
    print('Number of Generation: ', t)
```

```python
# Plotting prepartion
population =100
dimension = 2
position_min = -100.0
position_max = 100.0
generation = 400
fitness_criterion = 10e-4
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
x = np.linspace(position_min, position_max, 80)
y = np.linspace(position_min, position_max, 80)
X, Y = np.meshgrid(x, y)
Z= fitness_function(X,Y)
ax.plot_wireframe(X, Y, Z, color='r', linewidth=0.2)
# Animation image placeholder
images = []

# Add plot for each generation (within the generation for-loop)
image = ax.scatter3D([particles[n][0] for n in range(population)],[particles[n][1] for n in range(population)],
                     [fitness_function(particles[n][0],particles[n][1]) for n in range(population)], c='b')
images.append([image])

# Generate the animation image and save
animated_image = animation.ArtistAnimation(fig, images)
animated_image.save('./pso_simple.gif', writer='pillow')
```

Restricted Mode ⊗ 0 ⚠ 0 Jupyter Server: local Cell 19 of 22

OUTPUT(SCREENSHOT):

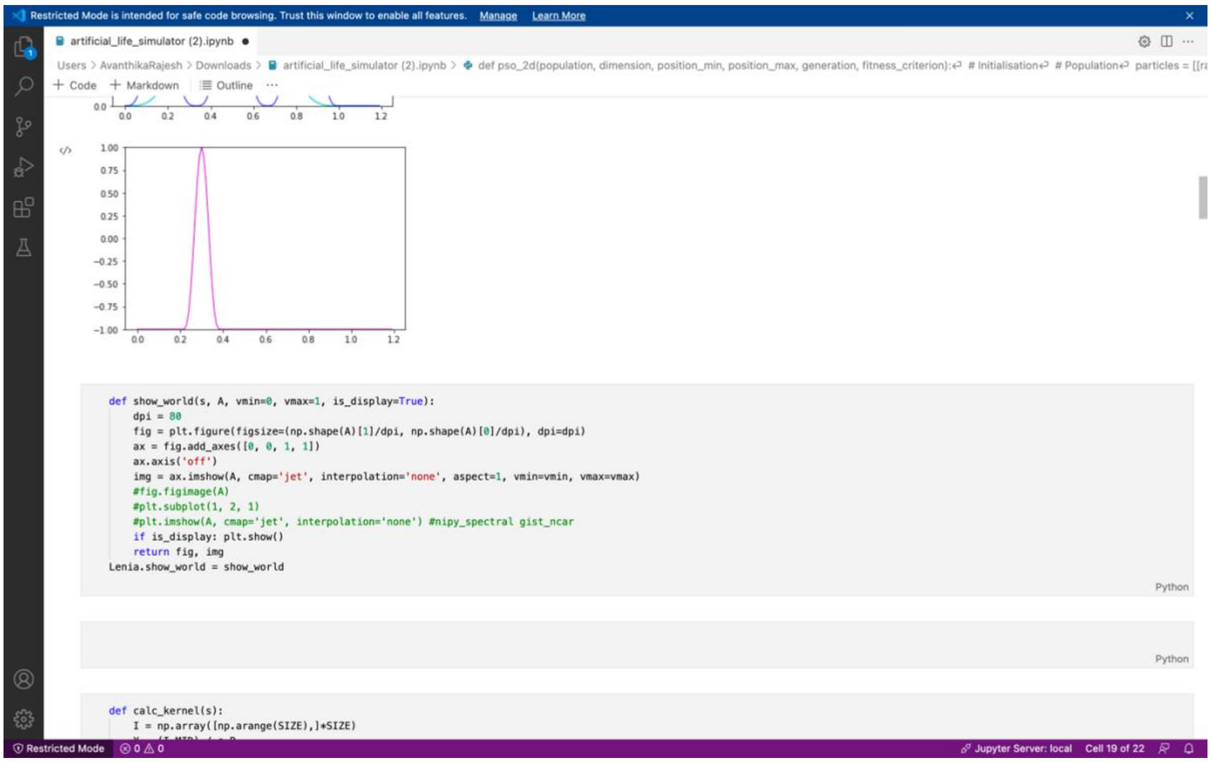
LENIA:



Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
artificial_life_simulator (2).ipynb
Users > AvanthikaRajesh > Downloads > artificial_life_simulator (2).ipynb > def pso_2d(population, dimension, position_min, position_max, generation, fitness_criterion): # Initialisation # Population particles = [[ra
+ Code + Markdown Outline
Python
Restricted Mode 0 0
Jupyter Server: local Cell 19 of 22

```python
def kernel_shell(s, r):
    k = len(s.peaks)
    kr = k * r
    peak = s.peaks[np.minimum(np.floor(kr).astype(int), k-1)]
    return (r<1) * s.kernel_core(kr % 1) * peak
Lenia.kernel_shell = kernel_shell

def delta_func(s, n):
    if s.delta_type == 0:
        return np.maximum(0, 1 - (n - s.mu)**2 / (s.sigma**2 * 9) )**4 * 2 - 1
    else:
        return np.exp( - (n - s.mu)**2 / (s.sigma**2 * 2) ) * 2 - 1
Lenia.delta_func = delta_func

lenia = Lenia()
lenia.set_params(R=52, peaks=np.array([1/2, 2/3, 1]),
                 mu=0.3, sigma=0.03, dt=0.1,
                 kernel_type=0, delta_type=0)

x = np.arange(0, 1.2, 0.01)
plt.plot(x, lenia.kernel_core(x), 'c-',
        x, lenia.kernel_shell(x), 'b-', linewidth=1)
plt.ylim((0, 1))
plt.show()
plt.plot(x, lenia.delta_func(x), 'm-', linewidth=1)
plt.ylim((-1, 1))
plt.show()
```

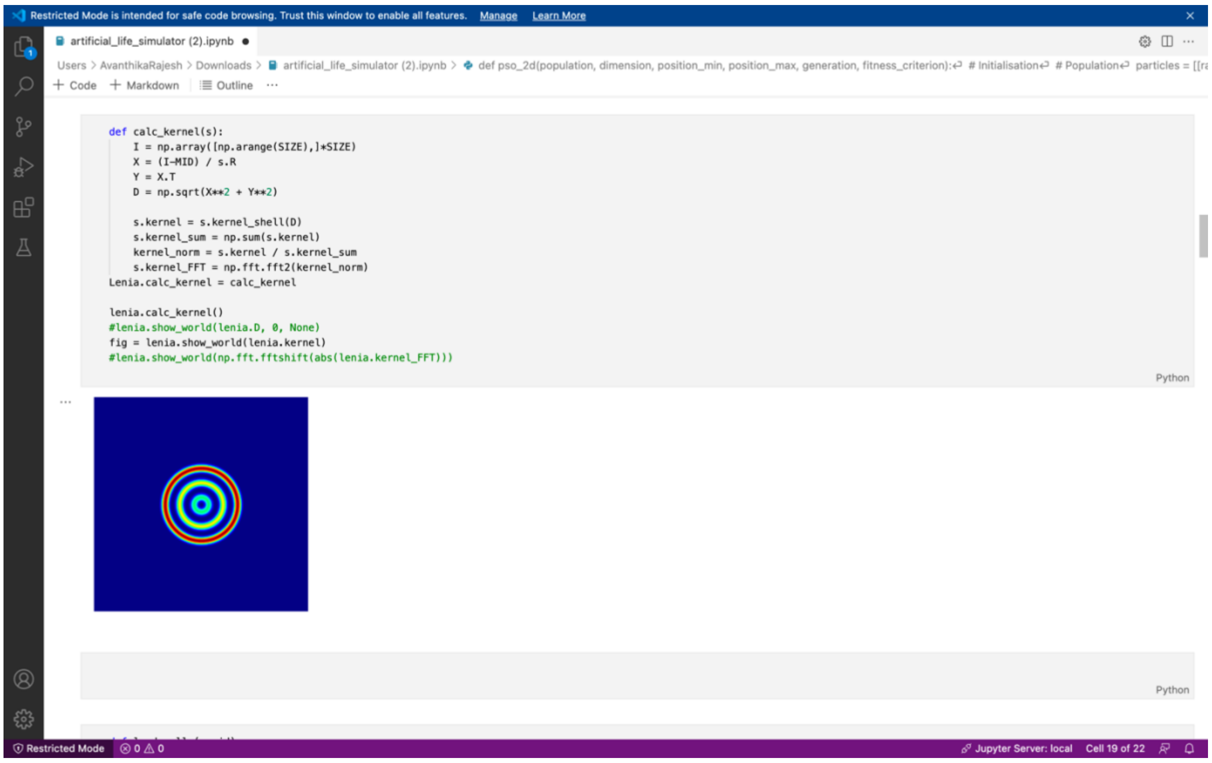
1.0
0.8
0.6
0.4
0.2
0.0
0.0 0.2 0.4 0.6 0.8 1.0 1.2



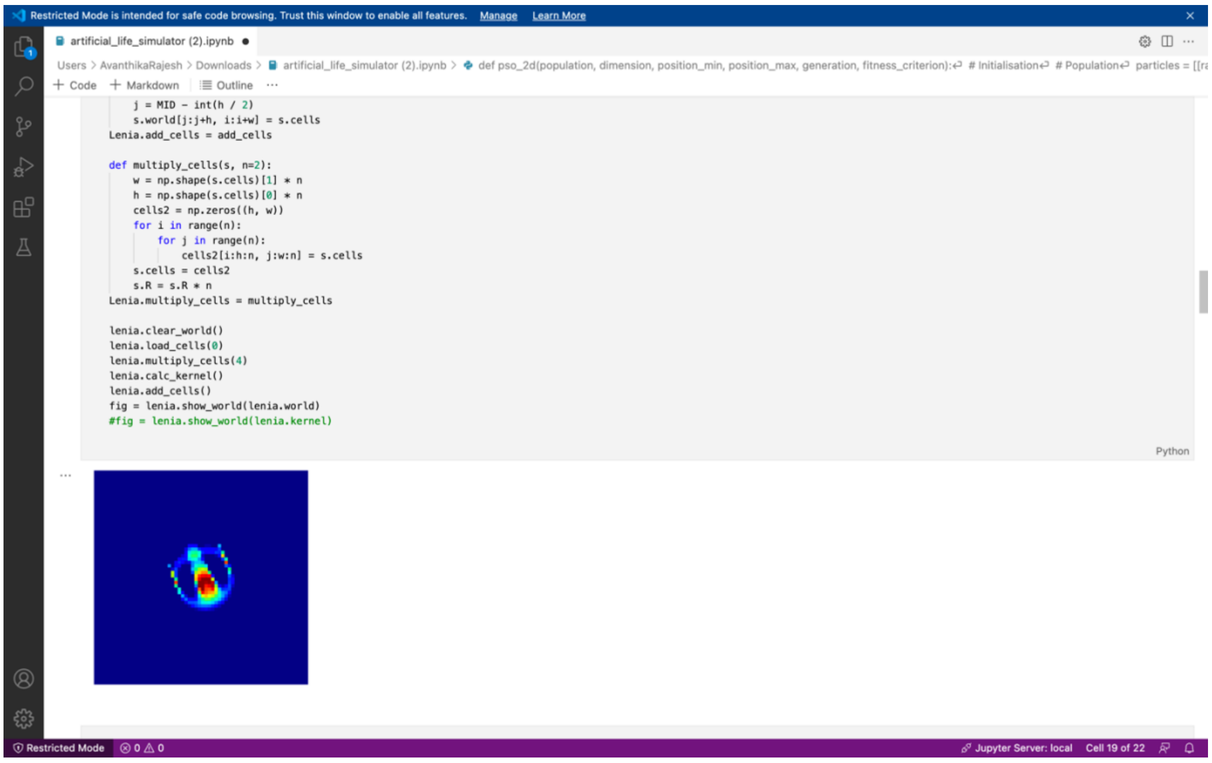
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
artificial_life_simulator (2).ipynb
Users > AvanthikaRajesh > Downloads > artificial_life_simulator (2).ipynb > def pso_2d(population, dimension, position_min, position_max, generation, fitness_criterion): # Initialisation # Population particles = [[ra
+ Code + Markdown Outline
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0 0.2 0.4 0.6 0.8 1.0 1.2
Python
Python
Restricted Mode 0 0
Jupyter Server: local Cell 19 of 22

```python
def show_world(s, A, vmin=0, vmax=1, is_display=True):
    dpi = 80
    fig = plt.figure(figsize=(np.shape(A)[1]/dpi, np.shape(A)[0]/dpi), dpi=dpi)
    ax = fig.add_axes([0, 0, 1, 1])
    ax.axis('off')
    img = ax.imshow(A, cmap='jet', interpolation='none', aspect=1, vmin=vmin, vmax=vmax)
    #fig.figimage(A)
    #plt.subplot(1, 2, 1)
    #plt.imshow(A, cmap='jet', interpolation='none') #nipy_spectral gist_ncar
    if is_display: plt.show()
    return fig, img
Lenia.show_world = show_world
```

```python
def calc_kernel(s):
    I = np.array([np.arange(SIZE),]*SIZE)
```

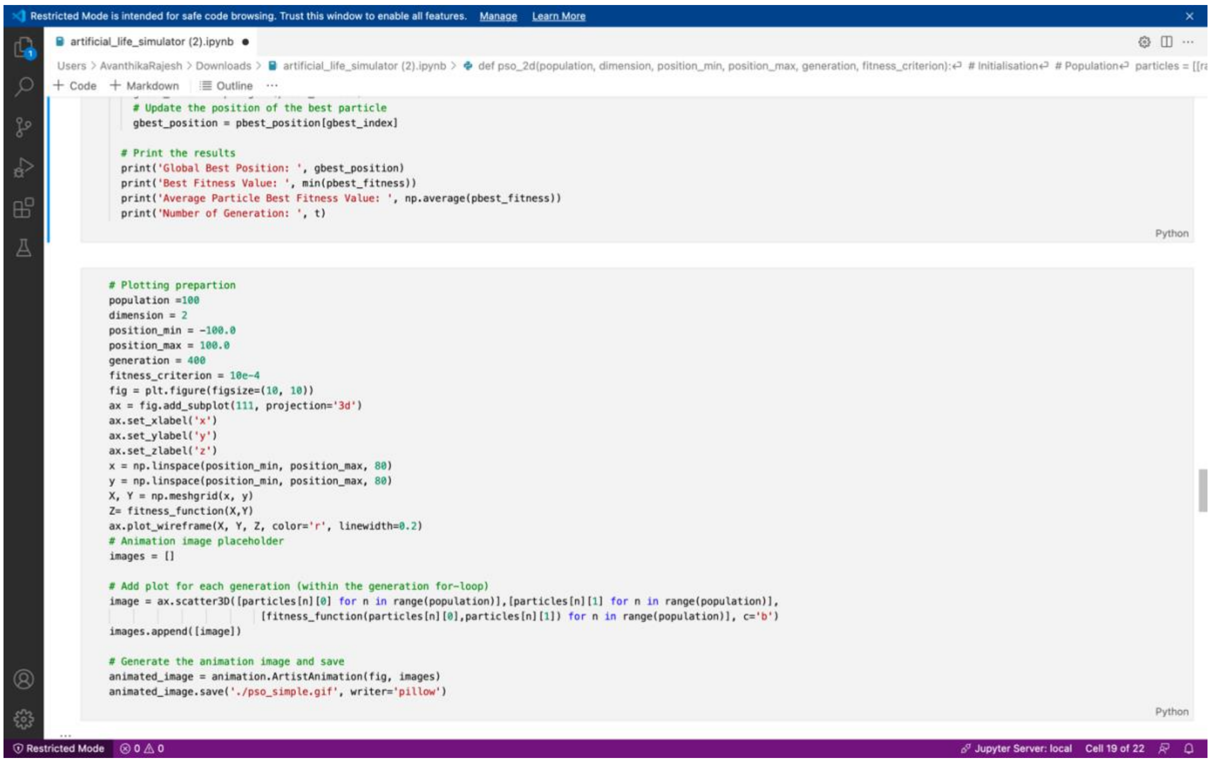

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
artificial_life_simulator (2).ipynb
Users > AvanthikaRajesh > Downloads > artificial_life_simulator (2).ipynb > def pso_2d(population, dimension, position_min, position_max, generation, fitness_criterion):↩ # Initialisation↩ # Population↩ particles = [[ra
+ Code + Markdown Outline
Python
Python
Restricted Mode 0 0
Jupyter Server: local Cell 19 of 22

```python
def calc_kernel(s):
    I = np.array([np.arange(SIZE),]*SIZE)
    X = (I-MID) / s.R
    Y = X.T
    D = np.sqrt(X**2 + Y**2)

    s.kernel = s.kernel_shell(D)
    s.kernel_sum = np.sum(s.kernel)
    kernel_norm = s.kernel / s.kernel_sum
    s.kernel_FFT = np.fft.fft2(kernel_norm)
Lenia.calc_kernel = calc_kernel

lenia.calc_kernel()
#lenia.show_world(lenia.D, 0, None)
fig = lenia.show_world(lenia.kernel)
#lenia.show_world(np.fft.fftshift(abs(lenia.kernel_FFT)))
```


Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
artificial_life_simulator (2).ipynb
Users > AvanthikaRajesh > Downloads > artificial_life_simulator (2).ipynb > def pso_2d(population, dimension, position_min, position_max, generation, fitness_criterion):↩ # Initialisation↩ # Population↩ particles = [[ra
+ Code + Markdown Outline
Python
Restricted Mode 0 0
Jupyter Server: local Cell 19 of 22

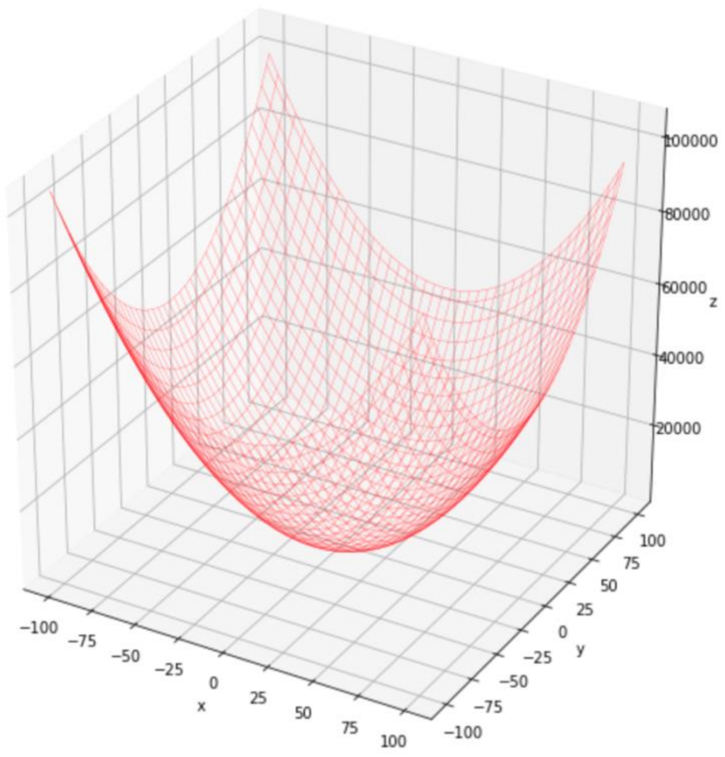
```python
    j = MID - int(h / 2)
    s.world[j:j+h, i:i+w] = s.cells
Lenia.add_cells = add_cells

def multiply_cells(s, n=2):
    w = np.shape(s.cells)[1] * n
    h = np.shape(s.cells)[0] * n
    cells2 = np.zeros((h, w))
    for i in range(n):
        for j in range(n):
            cells2[i:h:n, j:w:n] = s.cells
    s.cells = cells2
    s.R = s.R * n
Lenia.multiply_cells = multiply_cells

lenia.clear_world()
lenia.load_cells(0)
lenia.multiply_cells(4)
lenia.calc_kernel()
lenia.add_cells()
fig = lenia.show_world(lenia.world)
#fig = lenia.show_world(lenia.kernel)
```

PARTICLE SWARM OPTIMIZATION

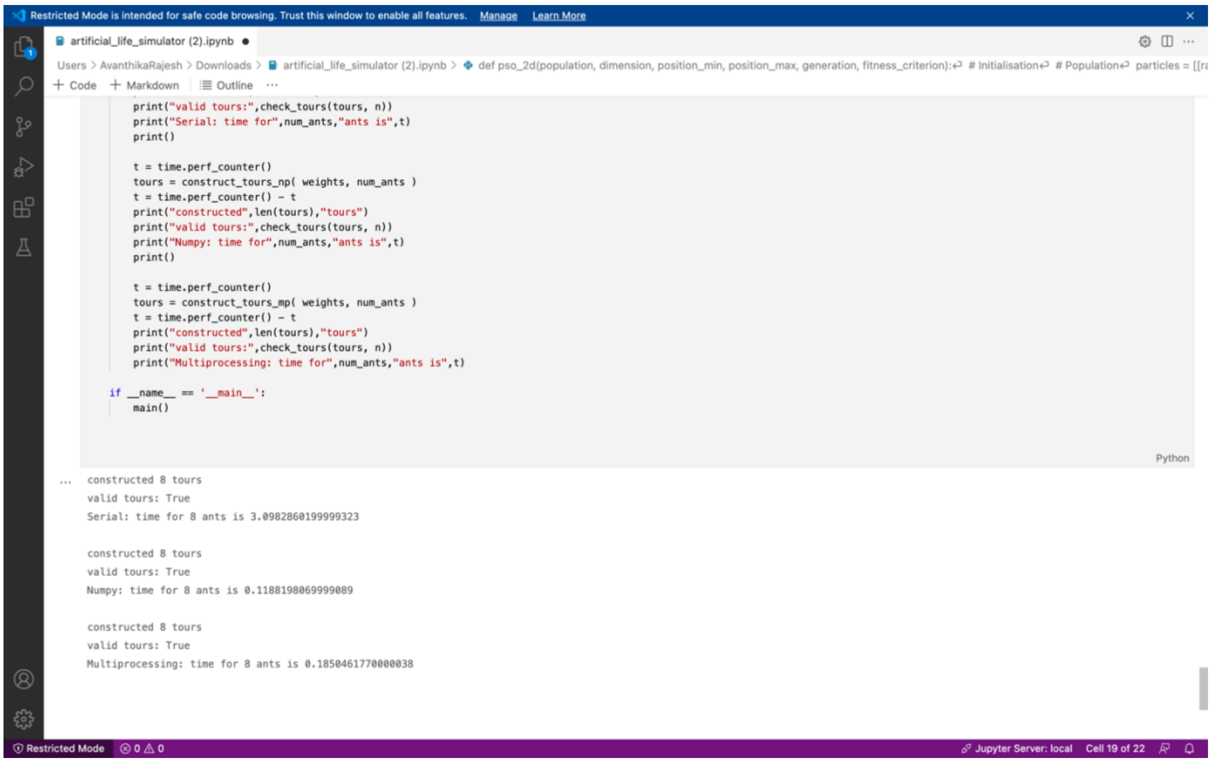

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
artificial_life_simulator (2).ipynb
Users > AvanthikaRajesh > Downloads > artificial_life_simulator (2).ipynb > def pso_2d(population, dimension, position_min, position_max, generation, fitness_criterion): # Initialisation # Population particles = [[ra
+ Code + Markdown Outline ···

```
        # Update the position of the best particle
        gbest_position = pbest_position[gbest_index]

    # Print the results
    print('Global Best Position: ', gbest_position)
    print('Best Fitness Value: ', min(pbest_fitness))
    print('Average Particle Best Fitness Value: ', np.average(pbest_fitness))
    print('Number of Generation: ', t)
```

```
# Plotting prepartion
population =100
dimension = 2
position_min = -100.0
position_max = 100.0
generation = 400
fitness_criterion = 10e-4
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
x = np.linspace(position_min, position_max, 80)
y = np.linspace(position_min, position_max, 80)
X, Y = np.meshgrid(x, y)
Z= fitness_function(X,Y)
ax.plot_wireframe(X, Y, Z, color='r', linewidth=0.2)
# Animation image placeholder
images = []

# Add plot for each generation (within the generation for-loop)
image = ax.scatter3D([particles[n][0] for n in range(population)],[particles[n][1] for n in range(population)],
                     [fitness_function(particles[n][0],particles[n][1]) for n in range(population)], c='b')
images.append([image])

# Generate the animation image and save
animated_image = animation.ArtistAnimation(fig, images)
animated_image.save('./pso_simple.gif', writer='pillow')
```



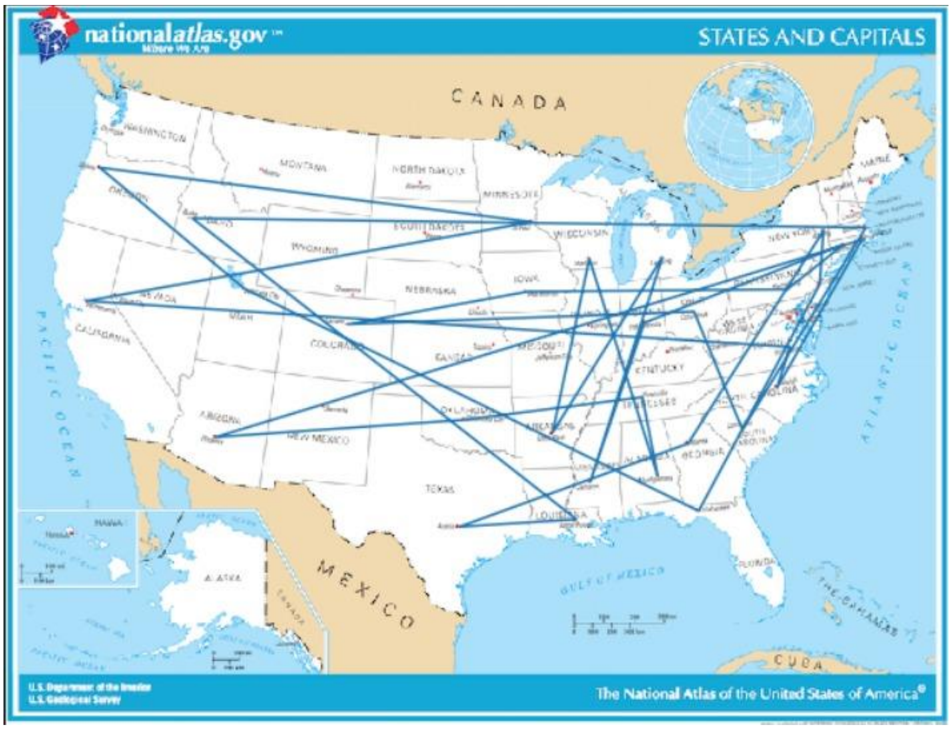
ANT-COLONY METHOD



```
        print("valid tours:",check_tours(tours, n))
        print("Serial: time for",num_ants,"ants is",t)
        print()

        t = time.perf_counter()
        tours = construct_tours_np( weights, num_ants )
        t = time.perf_counter() - t
        print("constructed",len(tours),"tours")
        print("valid tours:",check_tours(tours, n))
        print("Numpy: time for",num_ants,"ants is",t)
        print()

        t = time.perf_counter()
        tours = construct_tours_mp( weights, num_ants )
        t = time.perf_counter() - t
        print("constructed",len(tours),"tours")
        print("valid tours:",check_tours(tours, n))
        print("Multiprocessing: time for",num_ants,"ants is",t)

    if __name__ == '__main__':
        main()
```

```
constructed 8 tours
valid tours: True
Serial: time for 8 ants is 3.0982860199999323

constructed 8 tours
valid tours: True
Numpy: time for 8 ants is 0.1188198069999089

constructed 8 tours
valid tours: True
Multiprocessing: time for 8 ants is 0.1850461770000038
```

Restricted Mode is intended for safe code browsing. Trust this window to enable all features.
Manage
Learn More
artificial_life_simulator (2).ipynb
Users > AvanthikaRajesh > Downloads > artificial_life_simulator (2).ipynb > def pso_2d(population, dimension, position_min, position_max, generation, fitness_criterion): # Initialisation # Population particles = [[ra
Code
Markdown
Outline
Python
Restricted Mode
Jupyter Server: local
Cell 19 of 22


nationalatlas.gov
STATES AND CAPITALS
CANADA
MEXICO
PACIFIC OCEAN
ATLANTIC OCEAN
GULF OF MEXICO
CUBA
THE BAHAMAS
U.S. Department of the Interior
U.S. Geological Survey
The National Atlas of the United States of America

nationalatlas.gov
Where We Are
STATES AND CAPITALS
CANADA
MEXICO
PACIFIC OCEAN
ATLANTIC OCEAN
GULF OF MEXICO
CUBA
THE BAHAMAS
ALASKA
HAWAII
U.S. Department of the Interior
U.S. Geological Survey
The National Atlas of the United States of America®

CONCLUSION

Evolution famously exploits random mutations to find acceptable optimisations to a problem. Evolutionary algorithms can be used to design both the body and the controllers of robots, but that usually happens in simulation before a solution is implemented in the real world. This last step involves a lot of human labor, and often many failures, as the real world is always harsher than simulations. This paper proposes a system that builds and evolves small robots entirely in the real world, autonomously or with minimum human intervention. With this system, you would have no “reality gap” between the world the robot evolved in and the real world

One of the main goals of this project is to evolve agents that are adapted to survive in a particular environment.

We have to achieve this goal for simple environments where the project is required to find a balance between movement and resource consumption. Preliminary results should show that the evolved agent is typically better than a randomly generated agent with statistical analysis

REFERENCES:

https://github.com/BLayman/Artificial-LifE/
Simulator/blob/master/Additional_Resources/Project%20Report.pdf
https://onlinelibrary.wiley.com/doi/abs/10.1002/rnc.1829
https://alien-project.org/
https://kpfu.ru/staff_files/F_1407356997/overview.pdf
https://link.springer.com/article/10.1007/s00521-020-05458-6
https://citeseerx.ist.psu.edu/viewdoc/download?
https://www.opensourceagenda.com/tags/artificial-life
https://www.researchgate.net/publication/232672850_An_Artificial_Life_Sim
ulation_Library_Based_on_Genetic_Algorithm_3-
_Character_Genetic_Code_and_Biological_Hierarchy
https://onlinelibrary.wiley.com/doi/abs/10.1002/rnc.1829
https://doi.org/10.1002/rnc.1829Citations