```python
# Name: Dragon hunting algorithm
# Author: Team 1911426 at MCM contest
# Time: 2019.1.28

### Important ###
# Units for functions parameters and global variables standard:
# kg for weight, calories for energy, km for area,  days for time

import numpy as np
import matplotlib.pyplot as plt

##### Begin Global variables #####
# Experiment number, it determines where to write the output
EXP_NUM  = 'distribution'

# Species names
SPECIES_NAME = [
        ['Cattle', 'Sheep', 'Hare'],
        ['D', 'E', 'F'],
        ['G', 'H', 'J'],
]
# MASS of species in kg
MASS = [
        [753, 87.5, 3.94625],
        [0, 0, 0],
        [0, 0, 0]
]
# DENSITY of species in /km^2
DENSITY = [
        [3.4130, 9.4514, 0],
        [0, 0, 0],
        [0, 0, 0]
]
# ENERGY of species per MASS in calorie/kg
ENERGY_PER_MASS = [
        [1250000, 1180000, 1020000],
        [0, 0, 0],
        [0, 0, 0]
]
# Heat capacity of the meat of the species in calories/(kg * Celsius)
HEAT_CAPACITY = [
        [351.33843212, 358.50860421, 389.5793499],
        [0, 0, 0],
        [0, 0, 0]
```

```python
]

# The side length of the square area
SIDE_LENGTH = 10.0
# The area of the square area
AREA = SIDE_LENGTH ** 2

# Times of hunting
HUNTIMG_TIMES = 0

# Number of species
NUM_OF_SPECIES_PER_SPECIES = np.array(AREA * np.array(DENSITY), dtype=int)
NUM_OF_SPECIES = np.sum(NUM_OF_SPECIES_PER_SPECIES)

# Species used in our
COW = SPECIES_NAME[0][0]
SHEEP = SPECIES_NAME[0][1]
## Comment it out if you want to add more species
# HARE = SPECIES_NAME[0][2]

# Time period of dragon hunting
DAYS = 2

# The dragon's initial position in the area
DRAGON_POS = np.array([
        [0],
        [0]
])
# Not reachable area's position
NOT_REACHABLE = np.inf

# Net energy percentage
NET_ENERGY_PERCENTAGE = 0.57
# eta, see the paper
ETA = 0.7
##### End Global variables #####


##### Begin Helper functions #####
def get_mu_and_weight_at(age):
        """
        Get mu and weight at `age` using calculated S curve function.
        """
        mu_m = 2523.8311119211758
        lam = 19.76261573148329
        v = - 1/3
        A = 281.6 * 1000

```

```
    from sympy import symbols, exp, solve
    from sympy import Symbol

    t = symbols('t')

    temp1 = (mu_m / A) * ((1+v) ** (1 + 1/v)) * (lam - t)
    temp2 = exp(temp1)
    temp3 = v * exp(1 + v) * temp2
    temp4 = A * (1 + temp3)**(-1/v)

    y = temp4
    y_derivative = y.diff(t)

    return (y_derivative.subs(t, age), y.subs(t, age))

def find_nearest(array, value):
    """
    Find the nearest element to `value`, return its index in `array`.
    """
    array = np.asarray(array)
    new_array = array - value
    norm_array = np.empty(len(new_array[0]))
    for i in range(len(new_array[0])):
        norm_array[i] = new_array[0][i] ** 2 + new_array[1][i] ** 2
    idx = norm_array.argmin()
    return idx

def get_basic_metabolish_energy(weight):
    """
    Calculate E_m.
    """
    m_d = weight
    V_E = 2.25
    period = DAYS * 24
    V_O2 = m_d * V_E * period
    density_o2 = 1.429
    m_O2 = V_O2 * density_o2
    M_O2 = 32
    n_O2 = m_O2 / M_O2
    n_glucuse = n_O2 / 6
    energy = 277485.66 * n_glucuse
    return energy
def get_growth_energy(mu):
    """
    Calculate E_g.
    """
    period = DAYS * 24
```

```python
    dmd = mu / 365 / 24 * period
    rho_m = 1.12
    rho_b = 1.23
    r_b = 4
    r_m = 5
    coefficient_1 = rho_m + rho_b * (r_b ** 2) / (r_m ** 2)
    coefficient_2 = (r_b ** 2) / (r_b ** 2 + r_m ** 2)
    dS = dmd / coefficient_1 / coefficient_2

    E_p = 17130 * 1000 / 4.184
    E_b = 0.1 * E_p
    coefficient_3 = rho_m * (r_m ** 2) / (r_b ** 2 + r_m ** 2) * E_p
    coefficient_4 = rho_b * coefficient_2 * E_b
    E_g = (coefficient_3 + coefficient_4) * dS
    return E_g
def get_fly_energy(weight, distance):
    """
    Calculate E_f.
    """
    m_d = weight
    # v_d is in m/s
    v_d = 5.70 * (m_d ** 0.16)
    # convert to m
    L_d = distance * 1000
    # convert to hours
    temp_time = L_d / v_d / 60 / 60
    E_v = 300 / 4.184
    E_f = m_d * E_v * temp_time
    return E_f
def get_fire_energy(weight, x, y):
    """
    Calculate E_b.
    """
    c_p = HEAT_CAPACITY[x][y]
    m_p = MASS[x][y]
    constant = 5
    delta_T = 80 - 25
    return c_p * m_p * constant * delta_T

def get_reproduction_res(index, now, period):
    """
    Index: 0-cattle,1-sheep,2-hare
    period: in days
    """
    per_day_animal = np.array([0.5, 4, 6]) / 365

    return int(now + now * per_day_animal[index] * period)
```

```python

    def get_pos(idx):
        accr = np.cumsum(NUM_OF_SPECIES_PER_SPECIES)
        # Calculate species class from index in `total` (all the species)
        index1 = 0
        for i in range(len(accr)):
            if idx < accr[i]:
                index1 = i
                break
        x = index1 // 3
        y = index1 % 3
        return (x, y)
    ##### End Helper functions #####

    def hunting_at_age(age):
        """
        Main entraince of the hunting algorithm.
        """
        global HUNTIMG_TIMES
        print(f'############### Hunting times: {HUNTIMG_TIMES} at age {age} ###############')
        HUNTIMG_TIMES += 1

        (mu, weight) = get_mu_and_weight_at(age)

        # Regenerate animals
        global NUM_OF_SPECIES_PER_SPECIES
        cow = np.random.rand(2, NUM_OF_SPECIES_PER_SPECIES[0][0]) * SIDE_LENGTH
        sheep = np.random.rand(2, NUM_OF_SPECIES_PER_SPECIES[0][1]) * SIDE_LENGTH
        ## Comment it out if you want to add more species
        # hare = np.random.rand(2, NUM_OF_SPECIES_PER_SPECIES[0][2]) * SIDE_LENGTH

        # Recovery the hunting animals
        total = np.append(cow, sheep, axis=1)
        ## Comment it out if you want to add more species
        # total = np.append(
        #     np.append(cow, sheep, axis=1),
        #     hare,
        #     axis=1
        # )

        # Generate dragon
        dragon_pos = DRAGON_POS

        energy_got = 0
        base_consumption = get_basic_metabolish_energy(
```

```python
weight)
        print('Base consumption:', base_consumption)
        growth_consumption = get_growth_energy(mu)
        print('Growth consumption:', growth_consumption)
        hurt_consumption = 0.1 * base_consumption
        print('Hurt consumption:', hurt_consumption)
        fly_consumption = 0
        fire_cos = 0
        energy_consumed = base_consumption + growth_consumption + hurt_consumption

        # Begin iteration
        iter_times = 0
        hunted_number = 0
        hunted_each = np.array([
                [0, 0, 0],
                [0, 0, 0],
                [0, 0, 0]
        ])
        global NUM_OF_SPECIES
        while hunted_number < NUM_OF_SPECIES:
                iter_times += 1
                print('========================')
                print(f'Iteration: {iter_times}')

                idx = find_nearest(total, dragon_pos)
                (x, y) = get_pos(idx)

                ######### Begin hunting #########
                hunted_number += 1
                hunted_each[x][y] += 1

                energy_got += ENERGY_PER_MASS[x][y] * MASS[x][y]

                temp_fire_energy = get_fire_energy(weight, x, y)
                fire_cos += temp_fire_energy
                energy_consumed += temp_fire_energy
                temp_fly_energy = get_fly_energy(weight, np.linalg.norm(dragon_pos-np.array([total[:,idx]]).T))
                fly_consumption += temp_fly_energy
                energy_consumed += temp_fly_energy

                print(f'Delta Energy this round: {ENERGY_PER_MASS[x][y] * MASS[x][y] - temp_fire_energy - temp_fly_energy}')

                print(f'Nearest point {SPECIES_NAME[x][y]} at ({total[0][idx]}, {total[1][idx]}) hunted')
```

```
                dragon_pos = np.array([total[:, idx]]).T
                total[0][idx] = total[1][idx] = NOT_REACHABLE
                ######### End hunting ##########

                if energy_got * NET_ENERGY_PERCENTAGE * ETA >= energy_consumed:
                    print('Success got all energy')
                    print('Animals before:\n', NUM_OF_SPECIES_PER_SPECIES)
                    print('Animals hunted:\n', hunted_each)
                    NUM_OF_SPECIES_PER_SPECIES -= hunted_each
                    print('Animals left:\n', NUM_OF_SPECIES_PER_SPECIES)
                    # Breeding Animals
                    for i in range(3):
                        NUM_OF_SPECIES_PER_SPECIES[0][i] = get_reproduction_res(i, NUM_OF_SPECIES_PER_SPECIES[0][i], DAYS)
                    print('Animals after reproduction:\n', NUM_OF_SPECIES_PER_SPECIES)
                    NUM_OF_SPECIES = np.sum(NUM_OF_SPECIES_PER_SPECIES)

                else:
                    print('energy_got * NET_ENERGY_PERCENTAGE * ETA', energy_got * NET_ENERGY_PERCENTAGE * ETA)
                    print('energy_consumed:', energy_consumed)
                    print('Still need these calories of energy:', energy_consumed - energy_got * NET_ENERGY_PERCENTAGE * ETA)
                    print('Energy got:', energy_got)
                    print('Fire energy need:', temp_fire_energy)
                    print('Fly energy need:', temp_fly_energy)

        print(hunted_each)
        print('######################################################################')
        return hunted_each

if __name__ == "__main__":
    hunting_at_age(100)
```