```

import random
import math

class SensorNode:
    def __init__(self, x, y, energy):
        self.x = x
        self.y = y
        self.energy = energy

def calculate_distance(node1, node2):
    return math.sqrt((node1.x - node2.x)**2 + (node1.y - node2.y)**2)

def calculate_coverage(nodes, coverage_radius, area_width, area_height):
    covered_area = 0
    grid_size = 1  # Adjust grid size for accuracy vs. computation time
    for x in range(0, area_width, grid_size):
        for y in range(0, area_height, grid_size):
            for node in nodes:
                if calculate_distance(node, SensorNode(x, y, 0)) <= coverage_radius:
                    covered_area += grid_size**2
                    break
    return covered_area

def gwo(num_nodes, area_width, area_height, max_energy, coverage_radius, max_iterations):


    nodes = [SensorNode(random.uniform(0, area_width), random.uniform(0, area_height), max_energy) for _ in rang


    alpha_pos = nodes[0]
    alpha_score = calculate_coverage(nodes, coverage_radius, area_width, area_height)
    beta_pos = nodes[1]
    beta_score = calculate_coverage(nodes, coverage_radius, area_width, area_height)
    delta_pos = nodes[2]
    delta_score = calculate_coverage(nodes, coverage_radius, area_width, area_height)


    for i in range(3, num_nodes):
        current_score = calculate_coverage([nodes[i]], coverage_radius, area_width, area_height)
        if current_score > alpha_score:
            alpha_score = current_score
            alpha_pos = nodes[i]
        elif current_score > beta_score:
            beta_score = current_score
            beta_pos = nodes[i]
        elif current_score > delta_score:
            delta_score = current_score
            delta_pos = nodes[i]


    for iteration in range(max_iterations):
        a = 2 - 2 * iteration / max_iterations # linearly decrease from 2 to 0
        for i in range(num_nodes):
            # Update position of each wolf
            r1 = random.random()
            r2 = random.random()
            A1 = 2 * a * r1 - a
            C1 = 2 * r2
            D_alpha = abs(C1 * alpha_pos.x - nodes[i].x)
            X1 = alpha_pos.x - A1 * D_alpha

            r1 = random.random()
            r2 = random.random()
            A2 = 2 * a * r1 - a
            C2 = 2 * r2
            C2 = 2 * r2
            D_beta = abs(C2 * beta_pos.x - nodes[i].x)
            X2 = beta_pos.x - A2 * D_beta

            r1 = random.random()
            r2 = random.random()
            A3 = 2 * a * r1 - a
            C3 = 2 * r2
            D_delta = abs(C3 * delta_pos.x - nodes[i].x)
            X3 = delta_pos.x - A3 * D_delta

            nodes[i].x = (X1 + X2 + X3) / 3

            nodes[i].y = (abs(C1*alpha_pos.y - nodes[i].y) + abs(C2 * beta_pos.y - nodes[i].y) + abs(C3 * delta_

            nodes[i].x = max(0, min(nodes[i].x, area_width))
            nodes[i].y = max(0, min(nodes[i].y, area_height))


            current_score = calculate_coverage([nodes[i]], coverage_radius, area_width, area_height)
            if current_score > alpha_score:
              alpha_score = current_score; alpha_pos = nodes[i]
            elif current_score > beta_score:
                beta_score = current_score; beta_pos = nodes[i]
            elif current_score > delta_score:
                delta_score = current_score; delta_pos = nodes[i]

    return nodes, alpha_score


num_nodes = 20
area_width = 100
area_height = 100
max_energy = 100
coverage_radius = 10
max_iterations = 100

optimized_nodes, best_coverage = gwo(num_nodes, area_width, area_height, max_energy, coverage_radius, max_iterat

print("Optimized Node Positions:", [(node.x,node.y) for node in optimized_nodes])
print("Best Coverage:", best_coverage)
```

```
Optimized Node Positions: [(51.88343249692067, 5.734988552282058e-24), (51.54688733247386, 5.528569390969574e-24), (51.71523287762446, 7
Best Coverage: 4218
```