28/11/24

Grey Wolf Optimizer

CODE :

```
import numpy as np
import random

# Function to print student name and ID
def print_student_details():
    print("Chaitanya N INM22CS076\n")

# Environment: 2D grid
def get_grid_input():
    rows = int(input("Enter the number of rows in the grid: "))
    cols = int(input("Enter the number of columns in the grid: "))

    grid = []
    print("Enter the grid values (0 for free space, -1 for obstacles):")
    for i in range(rows):
        row = list(map(int, input(f"Enter row {i+1}: ").split()))
        grid.append(row)

    return grid

# Parameters
max_iterations = 100
population_size = 10

def is_valid_move(grid, x, y):
    """Check if a move is valid within the grid."""
    return 0 <= x < len(grid) and 0 <= y < len(grid[0]) and grid[x][y] != -1

def fitness(path, destination):
    """Calculate fitness of a path."""
    if not path:
        return float('inf')  # Invalid paths have infinite fitness
    distance = len(path)  # Length of the path
    end_point = path[-1]
    penalty = 0 if end_point == destination else 1000  # Penalty for not reaching the destination
    return distance + penalty

def initialize_population(grid, source, destination, population_size):
    """Randomly initialize paths."""
    population = []
```

```
    for _ in range(population_size):
        path = [source]
        current = source
        while current != destination:
            x, y = current
            # Random valid move
            possible_moves = [
                (x+1, y), (x-1, y), (x, y+1), (x, y-1)
            ]
             valid_moves = [move for move in possible_moves if is_valid_move(grid, *move) and
move not in path]
            if not valid_moves:
                break  # Dead end
            current = random.choice(valid_moves)
            path.append(current)
        population.append(path)
    return population

def update_position(alpha, beta, delta, wolf, grid):
    """Update wolf position based on alpha, beta, delta wolves."""
    new_path = []
    for i in range(len(wolf)):
        if i < len(alpha) and is_valid_move(grid, *alpha[i]):
            new_path.append(alpha[i])
        elif i < len(beta) and is_valid_move(grid, *beta[i]):
            new_path.append(beta[i])
        elif i < len(delta) and is_valid_move(grid, *delta[i]):
            new_path.append(delta[i])
        else:
            break
    return new_path

def display_grid_with_path(grid, path):
    """Display the grid with the path overlaid."""
    path_set = set(path)
    visual_grid = []
    for i in range(len(grid)):
        row = []
        for j in range(len(grid[0])):
            if (i, j) in path_set:
                row.append('*')  # Mark the path
            elif grid[i][j] == -1:
                row.append('X')  # Represent obstacles
            else:
```

```
                row.append('.')  # Represent free spaces
        visual_grid.append(row)
    return visual_grid


# Main GWO Algorithm
def gwo_path_planning():
    print_student_details()

    # Get grid input from the user
    grid = get_grid_input()

    # Get start and destination points from user
    source = tuple(map(int, input("Enter the start point (x, y): ").split()))
    destination = tuple(map(int, input("Enter the destination point (x, y): ").split()))

    population = initialize_population(grid, source, destination, population_size)

    for iteration in range(max_iterations):
        # Sort population by fitness
        population = sorted(population, key=lambda path: fitness(path, destination))
        alpha, beta, delta = population[0], population[1], population[2]

        # Update positions
        new_population = []
        for wolf in population:
            new_path = update_position(alpha, beta, delta, wolf, grid)
            new_population.append(new_path)
        population = new_population

    # Output the best path
    best_path = sorted(population, key=lambda path: fitness(path, destination))[0]
    print(f"Best Path From {source} to {destination}: ", best_path)

    # Visualize the grid with the path
    visualized_grid = display_grid_with_path(grid, best_path)
    print("\nGrid showing the Best Path with stars representing the path and X representing obstacles:")
    for row in visualized_grid:
        print(' '.join(row))

# Call the function to run the program
gwo_path_planning()
```

Output :

```
Enter the number of rows in the grid: 5
Enter the number of columns in the grid: 5
Enter the grid values (0 for free space, -1 for obstacles):
Enter row 1: 0 0 0 -1 0
Enter row 2: -1 -1 0 -1 0
Enter row 3: 0 0 0 0 0
Enter row 4: 0 -1 -1 -1 0
Enter row 5: 0 0 0 0 0
Enter the start point (x, y): 0 0
Enter the destination point (x, y): 4 4
Best Path From (0, 0) to (4, 4):  [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (2, 3), (2, 4), (3, 4), (4, 4)]

Grid showing the Best Path with stars representing the path and X representing obstacles:
* * * X .
X X * X .
. . * * *
. X X X *
. . . . *
```