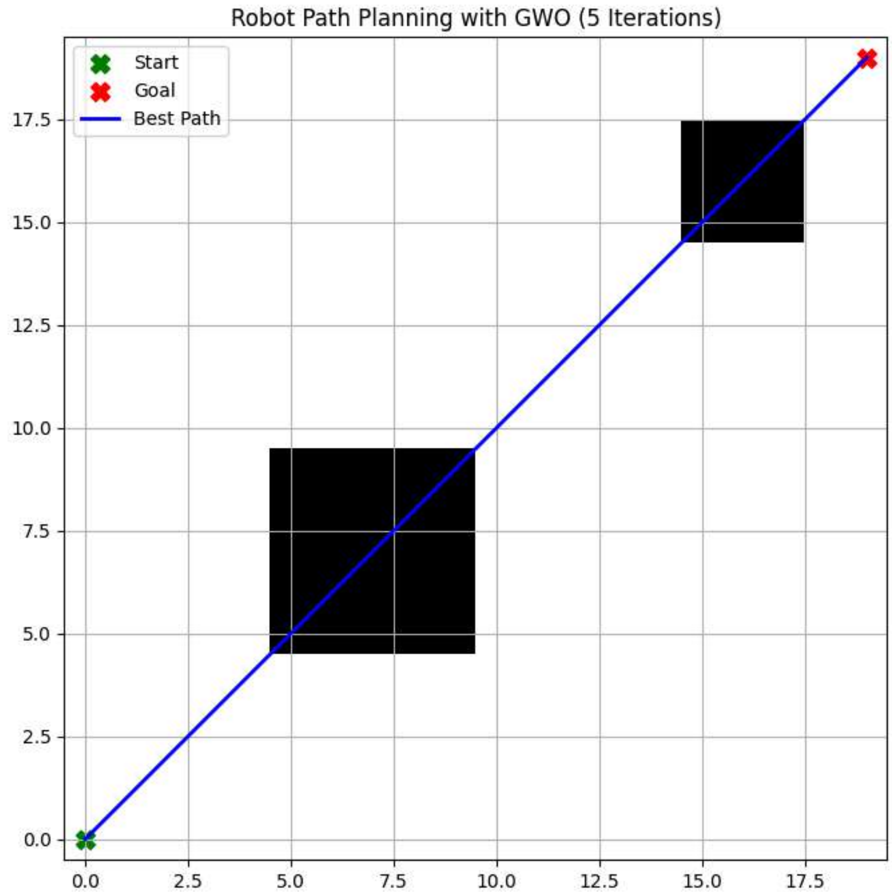
```
import numpy as np
import matplotlib.pyplot as plt
import random

# Define Euclidean distance
def euclidean_distance(x1, y1, x2, y2):
    return np.sqrt((x2 - x1)**2 + (y2 - y1)**2)

# Fitness function: Minimize path length, avoid obstacles and move towards goal
def fitness_function(path, grid, goal):
    total_distance = 0
    obstacle_penalty = 0

    # Calculate total distance of the path
    for i in range(len(path) - 1):
        x1, y1 = path[i]
        x2, y2 = path[i + 1]
        total_distance += euclidean_distance(x1, y1, x2, y2)

    # Penalize if the path intersects with any obstacles
    for (x, y) in path:
        if grid[int(y), int(x)] == 1:  # Check if the position is an obstacle
            obstacle_penalty += 1000  # Large penalty for obstacle collision

    # Add penalty for being far from the goal
    final_x, final_y = path[-1]
    goal_distance = euclidean_distance(final_x, final_y, goal[0], goal[1])

    return total_distance + obstacle_penalty + goal_distance

# Grey Wolf Optimizer (GWO) algorithm for optimizing robot path
def gwo_optimization(start, goal, grid, num_wolves, max_iter, num_waypoints):
    # Initialize wolves (each wolf represents a potential path)
    wolves = np.random.rand(num_wolves, num_waypoints, 2)  # Random initial positions of wolves

    # Scale wolves positions to fit within the grid
    wolves[:, :, 0] = wolves[:, :, 0] * (goal[0] - start[0]) + start[0]
    wolves[:, :, 1] = wolves[:, :, 1] * (goal[1] - start[1]) + start[1]

    fitness_values = np.zeros(num_wolves)

    # Initialize alpha, beta, and delta wolves
    alpha_position = np.zeros((num_waypoints, 2))
    beta_position = np.zeros((num_waypoints, 2))
    delta_position = np.zeros((num_waypoints, 2))

    alpha_fitness = float("inf")
    beta_fitness = float("inf")
    delta_fitness = float("inf")

    # Main loop (5 iterations)
    for iteration in range(max_iter):
        for i in range(num_wolves):
            # Flatten the path into a list of points
            path = [start] + list(wolves[i]) + [goal]  # Start, wolves' waypoints, goal

            # Evaluate fitness
            fitness_values[i] = fitness_function(path, grid, goal)

            # Update alpha, beta, delta wolves
            if fitness_values[i] < alpha_fitness:
                alpha_fitness = fitness_values[i]
                alpha_position = wolves[i]

            elif fitness_values[i] < beta_fitness:
                beta_fitness = fitness_values[i]
                beta_position = wolves[i]

            elif fitness_values[i] < delta_fitness:
                delta_fitness = fitness_values[i]
                delta_position = wolves[i]

        # Update the position of each wolf
        a = 2 - iteration * (2 / max_iter)  # Decreasing factor for exploration to exploitation
        for i in range(num_wolves):
            # Coefficients for exploring and exploiting
```

```
            A1 = 2 * a * random.random() - a
            A2 = 2 * a * random.random() - a
            A3 = 2 * a * random.random() - a

            # Update wolves based on alpha, beta, delta positions
            D_alpha = np.abs(alpha_position - wolves[i])  # Distance to alpha wolf
            D_beta = np.abs(beta_position - wolves[i])    # Distance to beta wolf
            D_delta = np.abs(delta_position - wolves[i])  # Distance to delta wolf

            # Update positions
            wolves[i] = wolves[i] + A1 * D_alpha + A2 * D_beta + A3 * D_delta
            wolves[i] = np.clip(wolves[i], 0, 1)  # Keep wolves within bounds [0, 1]

        # Print best solution for every iteration (optional)
        print(f"Iteration {iteration}: Best Fitness = {alpha_fitness}")

    # Return the best path found
    best_path = [start] + list(alpha_position) + [goal]
    return best_path, alpha_fitness

# Define the grid (0 = free space, 1 = obstacle)
grid_size = 20
grid = np.zeros((grid_size, grid_size))  # Create a 20x20 grid (all free space)

# Add obstacles (1 represents an obstacle)
grid[5:10, 5:10] = 1  # An obstacle in the middle of the grid
grid[15:18, 15:18] = 1  # Another obstacle

# Define the start and goal positions
start = (0, 0)
goal = (19, 19)

# Parameters for GWO
num_wolves = 10
max_iter = 5  # Set the number of iterations to 5
num_waypoints = 5

# Run GWO to optimize robot path
best_path, best_fitness = gwo_optimization(start, goal, grid, num_wolves, max_iter, num_waypoints)

# Visualize the result
print(f"Best path (optimized): {best_path}, Fitness: {best_fitness}")

# Plot the obstacles, start, goal, and optimized path
plt.figure(figsize=(8, 8))
plt.imshow(grid, cmap="Greys", origin="lower")

# Plot the start and goal points
plt.scatter(start[0], start[1], color='green', label="Start", s=100, marker='X')
plt.scatter(goal[0], goal[1], color='red', label="Goal", s=100, marker='X')

# Plot the best path
best_path = np.array(best_path)
plt.plot(best_path[:, 0], best_path[:, 1], color='blue', linewidth=2, label="Best Path")

# Display the obstacles
plt.title("Robot Path Planning with GWO (5 Iterations)")
plt.legend()
plt.grid(True)
plt.show()
```

```
Iteration 0: Best Fitness = 45.65710553424785
Iteration 1: Best Fitness = 26.870057685088806
Iteration 2: Best Fitness = 26.870057685088806
Iteration 3: Best Fitness = 26.870057685088806
Iteration 4: Best Fitness = 26.870057685088806
Best path (optimized): [(0, 0), array([1., 1.]), array([1., 1.]), array([1., 1.]), array([1., 1.]), array([1., 1.]), (19, 19)], Fitness:
```



Start coding or generate with AI.