ROBOT PATH PLANNER

Genetic Algorithm Project

Youssef Maher Elsebaey	*20201701747*
Yassin Wail Masry	*20201701746*
Youssef Mohamed Rady	*20201701748*

First class: Chromosome

Initializes a chromosome object with a path attribute and a fitness attribute.

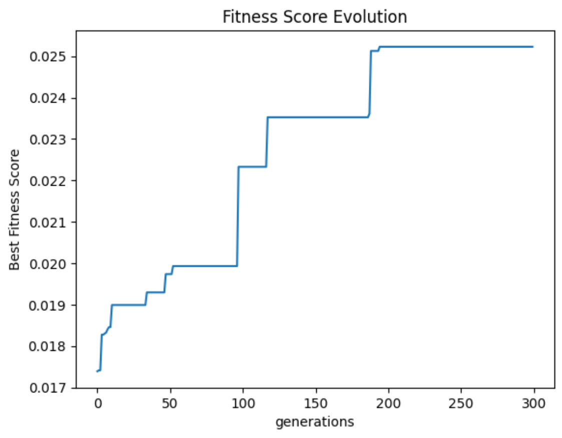
- **Class method: random_chromosome(chromo, points, start, finish):** creates a new chromosome object with a random path. x_interval is calculated by dividing the difference between the x-coordinates of the starting and finishing points by the number of points plus 1. This ensures that the points are evenly spaced between the start and finish. The method then checks if the y-coordinates of the start and finish points are the same. If they are, it creates a path with points at the same y-coordinate. Otherwise, it creates a path with random y-coordinates within the range of the start and finish y-coordinates. It returns a chromosome object with the generated path.
- **calculate_fitness(self, obstacles):** calculates the fitness of the chromosome based on its total distance and the number of collisions with obstacles. It takes a list of obstacles as a parameter. It gets total distance by summing the total distance between each point in the path. It iterates over the range from 0 to the path's length – 1 to not count in an additional point after the final point. It returns total_distance.
- **get_distance(self, point1, point2):** Calculates Euclidean distance between point1 and point2 and returns the distance calculated.
- **count_collisions(self, obstacles):** Counts the number of collisions between the chromosome's path and a list of obstacles. It takes the obstacles list as a parameter. It iterates over each obstacle in the obstacles list calls collides_with_obstacle method to check if the chromosome's path collides with each obstacle. The sum function is then used to sum up the true values, counting the number of collisions. The method returns the total count of collisions.
- **collides_with_obstacle(self, obstacle):** The obstacle values are obtained from the obstacle object passed. It checks if any point in the chromosome's path satisfies the condition with respect to the obstacle's x-coordinate, y-coordinate, and size. The function returns True if there is a collision with the obstacle and False otherwise.

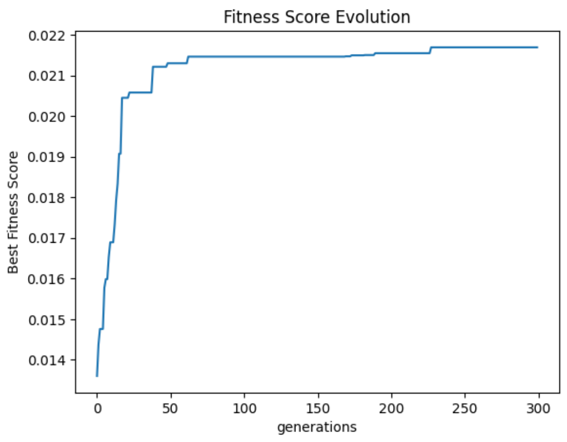
Second class: GeneticAlgorithm

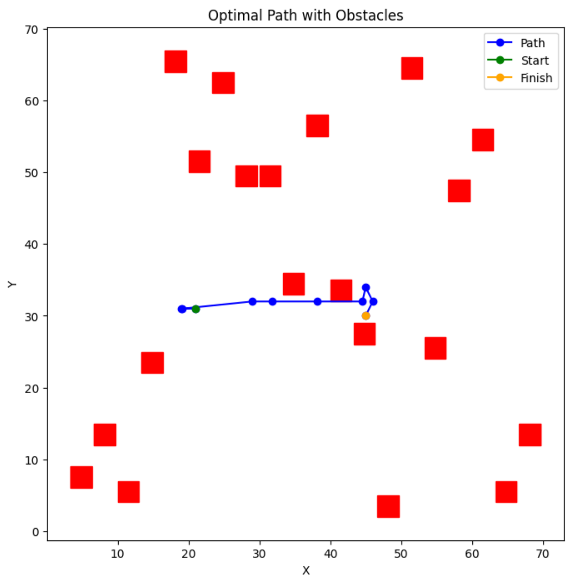
Represents a genetic algorithm used for optimizing path.

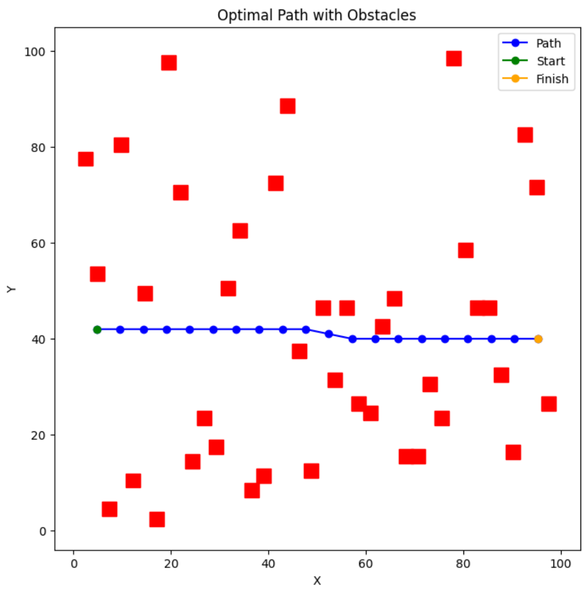
- Constructor: population_size: The size of the population (number of chromosomes).
- chromosome_length: The length of each chromosome.
- mutation_rate: The probability of mutation occurring during reproduction.
- crossover_rate: The probability of crossover occurring during reproduction.
- elitism_rate: The proportion of the top-performing individuals to be preserved unchanged in the next generation.
- **generate_initial_population(self):** Initializes the population with random chromosomes and creates a random path from a start point to a finish point and a list of obstacles. iterates over the number of obstacles to create obstacles with random positions within the plot boundaries. Then, for each individual in the population, a random chromosome is created using the random_chromosome method.
- **select_parents(self):** randomly selects two parents and returns the one with the higher fitness value.
- **crossover(self, parent1, parent2):** randomly selects a crossover point and combines the path segments of the two parents to create a child chromosome.
- **mutate(self, child):** applies a mutation to a child chromosome by iterating through each point in the child's path and replacing it with a new random point based on the mutation rate.
- **evolve(self):** Initialize the population and obstacle list by calling the generate_initial_population function. For each individual in the population, calculate their fitness by calling the calculate_fitness method on each individual and passing the obstacle_list. Get the best individual based on fitness using max(). Create a new population by starting with the best individual and then repeatedly selecting parents, performing crossover, and mutation until the new population reaches the desired size. Then update the population. Get the new best individual in the new population. Extract the best individual’s path, obstacle list, start point, and finish point to visualize. Return the best individual.

Sample Outputs


Fitness Score Evolution
Best Fitness Score
generations


Fitness Score Evolution
Best Fitness Score
generations


Optimal Path with Obstacles
Path
Start
Finish
X
Y


Optimal Path with Obstacles
Path
Start
Finish
X
Y