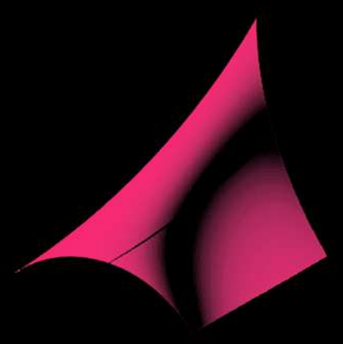
PARK
CHEOL
HEE



Computational Design

Code for architecture

Algorithmic Sketchbook

Contents

Github Link

02

Toy Projects

Toy Projects are projects that I worked on to explore what insights can be obtained when design and coding are combined.

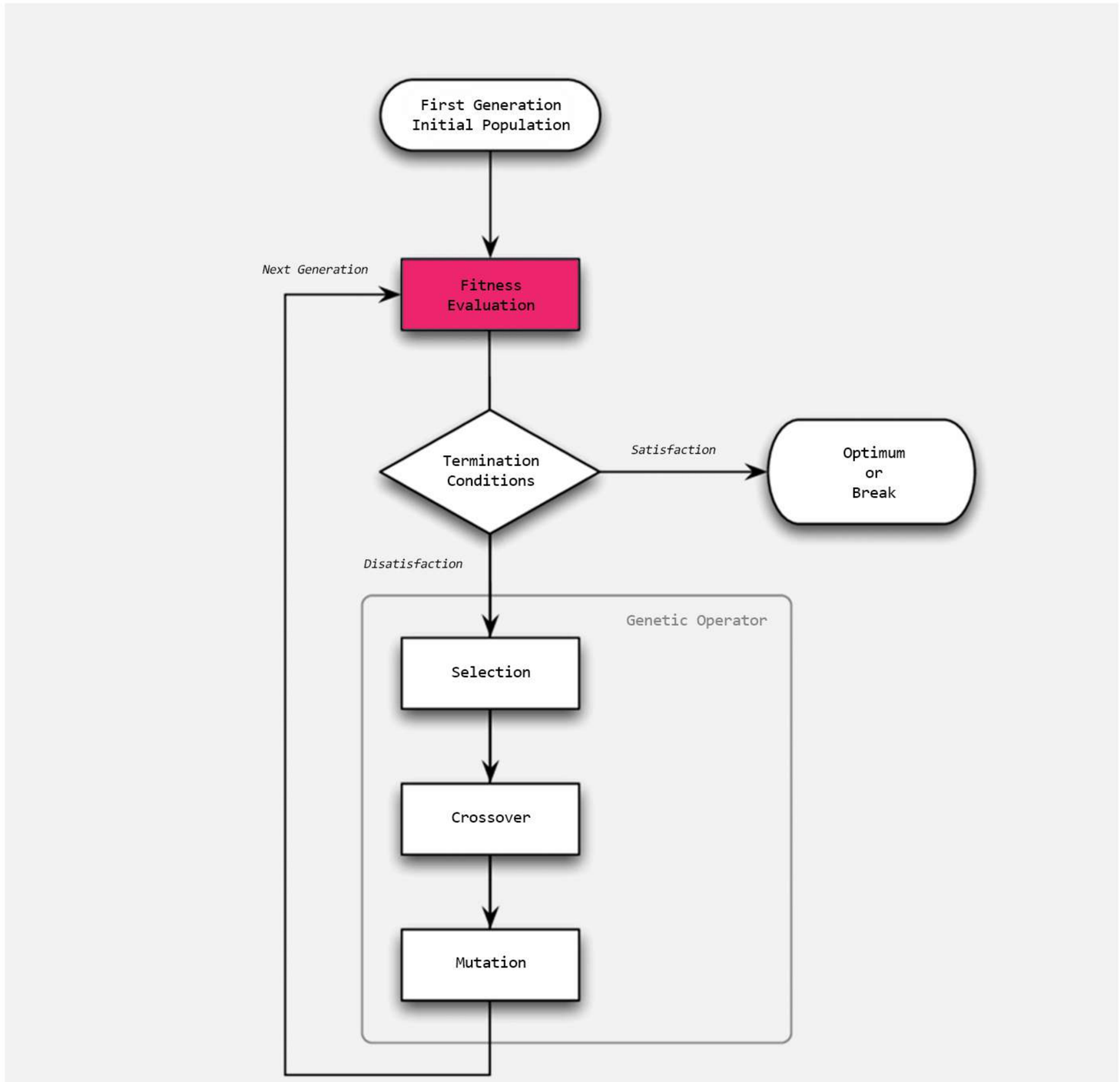
Genetic Algorithm

Lucky Number Generator

Overview

This work is a project to understand the concept of genetic algorithm and to consider simple application.

I tried to write the code in the easiest way to apply genetic algorithms. So The goal is simple. Make Chromosome, which owns seven lucky numbers, and then make Genome, which has seven Chromosome again.



" Flow Chart "

" Definition of Requriments "

```
GENE_COUNT = 7
CHROMOSOME_COUNT = 7
GENERATION_LIMIT = 777

goal_genome    =>    [[7, 7, 7, 7, 7, 7, 7],
                      [7, 7, 7, 7, 7, 7, 7],
                      [7, 7, 7, 7, 7, 7, 7],
                      [7, 7, 7, 7, 7, 7, 7],
                      [7, 7, 7, 7, 7, 7, 7],
                      [7, 7, 7, 7, 7, 7, 7],
                      [7, 7, 7, 7, 7, 7, 7]]
```

Dominant

By subtracting lucky numbers from each gene and adding all Chromosome's scores, I made 7 The dominant factor.

" Dominant Factor Evaluation "

```
curr_genome     =>    [[1, 7, 6, 5, 9, 3, 2],
                       [2, 6, 3, 1, 0, 8, 5],
                       [5, 3, 6, 8, 1, 7, 2],
                       [6, 5, 8, 7, 3, 0, 7],
                       [8, 3, 6, 5, 1, 0, 7],
                       [1, 5, 7, 8, 7, 6, 3],
                       [4, 6, 1, 0, 8, 3, 3]]

genome_score    =>    [20, 26, 19, 15, 21, 14, 26]
```


" Conception "

03

Parents Selection

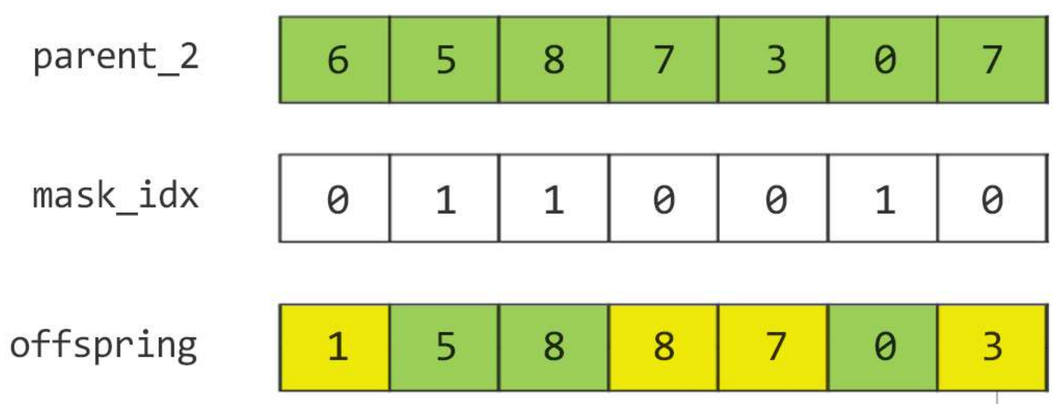
Evaluate the generated population through the 'Fitness Evaluation function' and score each Chromosome.

And I use two Chromosomes with the most suitable scores as parent chromosomes.

```
curr_genome     =>  [[1, 7, 6, 5, 9, 3, 2],
                     [2, 6, 3, 1, 0, 8, 5],
                     [5, 3, 6, 8, 1, 7, 2],
                     [6, 5, 8, 7, 3, 0, 7],
                     [8, 3, 6, 5, 1, 0, 7],
                     [1, 5, 7, 8, 7, 6, 3],
                     [4, 6, 1, 0, 8, 3, 3]]

genome_score    =>  [20, 26, 19, 15, 21, 14, 26]

best_score_index    =>  [5, 3, 2, 0, 4, 1, 6]
```

```
parent_1   1 5 7 8 7 6 3
parent_2   6 5 8 7 3 0 7
```

“ Parents Selection ”



Uniform Crossover

Reproduction is carried out through the parent chromosome derived through the previous process. In the gene manipulation stage, the uniform crossover method was used. Randomly select one of the parent chromosomes to mask the genes of the parent chromosome and assign the masked gene to the child chromosome. Repeat this process until the termination condition is satisfied.

```python
def reproduction_genome(self, parent_chromosome, offspring_chromosome, best_parents_indices):
    for i in range(self.GENOME_COUNT):
        for j in range(self.GENE_COUNT):
            if random.random() < self.MUTATION_RATE:
                mutated_gene = random.randint(0, 9)
                offspring_chromosome[i][j] = mutated_gene
            else:
                idx = random.randint(0, 1)
                selected_parent = best_parents_indices[idx]
                crossoverd_gene = parent_chromosome[selected_parent][j]
                offspring_chromosome[i][j] = crossoverd_gene

def evaluate_genome(self, genome):
    evaluation_values = [0] * len(genome)
    for i, chromosome in enumerate(genome):
        score = 0
        for gene in chromosome:
            score += abs(gene - self.DOMINANT_FACTOR)
        evaluation_values[i] = score
    return evaluation_values

def copy_genome(self, genome):
    return copy.deepcopy(genome)

def evaluate_dominant(self, evaluation_values):
    if evaluation_values.count(0) == CHROMOSOME_COUNT:
        return True

    # for i in range(GENOME_COUNT):
    #     if evaluation_values[i] == 0:
    #         return True

def best_parents(self, evaluation_values):
    PARENTS_COUNT = 2
    best_parents_indices = [0] * PARENTS_COUNT
    for i in range(PARENTS_COUNT):
        best_parent_index = np.argsort(evaluation_values)[i]
        best_parents_indices[i] = best_parent_index
    return best_parents_indices

def main(self):
    generation = 1
    while generation < self.GENERATION_LIMIT:
        if generation == 1:
            parent_genome = self.generate_genome()
        offspring_genome = self.copy_genome(parent_genome)

    score = self.evaluate_genome(parent_genome)
    best_parents_indices = self.best_parents(evaluation_values)

    self.reproduction_genome(parent_genome, offspring_genome, best_parents_indices)
    parent_genome = self.copy_genome(offspring_genome)

    if self.evaluate_dominant(evaluation_values):
        break

    score = self.evaluate_genome(parent_genome)
    best_parents_indices = self.best_parents(evaluation_values)

    print("\n")
    print("generation:", generation)
    print("genome:", parent_genome)
    print("score:", evaluation_values)
    print("parents:", best_parents_indices)

    generation += 1
```

```python
if __name__ == "__main__":
    random.seed(777)

    GENE_COUNT = 7
    CHROMOSOME_COUNT = 7
    MUTATION_RATE = 0.07
    DOMINANT_FACTOR = 7
    GENERATION_LIMIT = 777

    ga = GeneticAlgorithm(gene_count=GENE_COUNT,
                          chromosome_count=CHROMOSOME_COUNT,
                          mutation_rate=MUTATION_RATE,
                          dominant_factor=DOMINANT_FACTOR,
                          generation_limit=GENERATION_LIMIT)

    ga.main()
```

lucky_number_generator.py

Result

Through the repetition of the process described above, chromosomes with dominant factors are gradually created. Genes made up of lucky numbers appeared in the 104th generation!

```
generation: 1
genome:     [[3, 0, 7, 9, 8, 6, 5],
             [3, 7, 7, 9, 9, 4, 9],
             [8, 0, 7, 9, 9, 4, 9],
             [8, 0, 7, 9, 4, 6, 9],
             [8, 0, 7, 5, 9, 6, 5],
             [3, 0, 7, 9, 3, 4, 9],
             [8, 7, 7, 5, 8, 4, 5]]
score:      [17, 13, 17, 16, 15, 22, 9]
parents:    [6, 1]
```

```
generation: 2
genome:     [[8, 7, 7, 9, 9, 4, 5],
             [4, 7, 7, 5, 3, 4, 9],
             [7, 7, 7, 9, 9, 4, 5],
             [3, 7, 7, 5, 9, 4, 9],
             [3, 7, 7, 9, 9, 4, 2],
             [3, 7, 7, 9, 8, 4, 5],
             [8, 7, 7, 9, 8, 4, 9]]
score:      [10, 14, 9, 13, 16, 12, 9]
parents:    [2, 6]
```

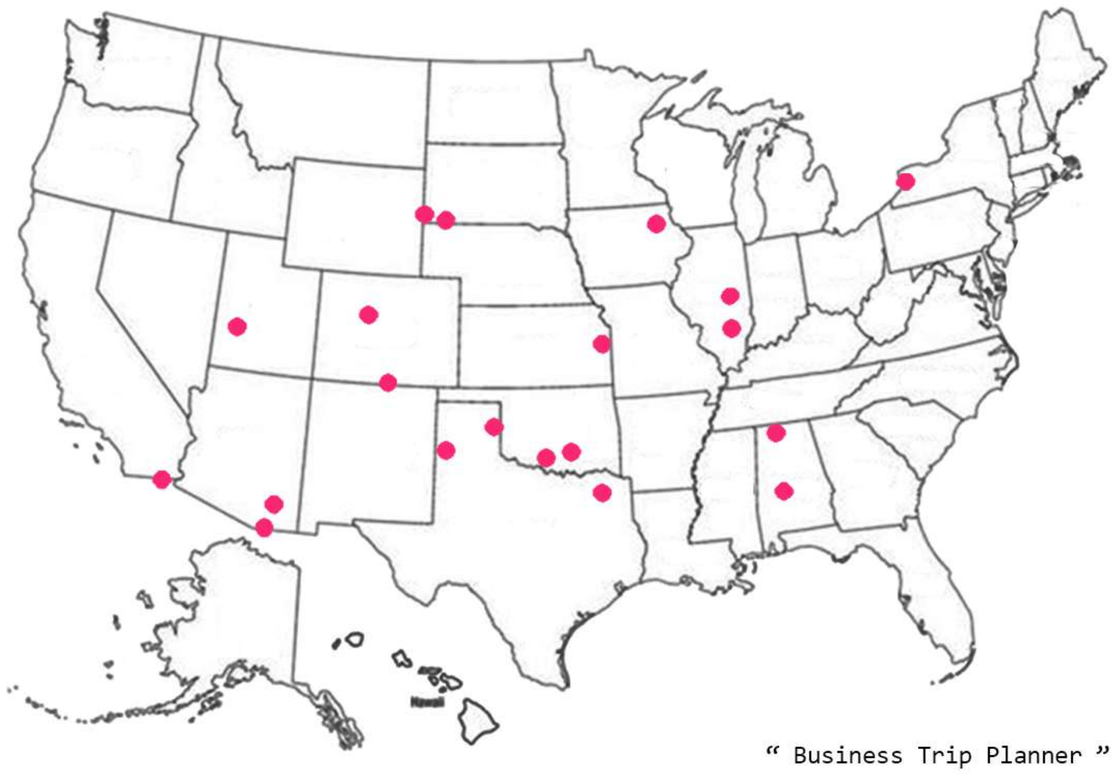
```
generation: 3
genome:     [[7, 7, 7, 9, 0, 4, 5],
             [7, 7, 7, 9, 8, 4, 5],
             [8, 7, 7, 9, 8, 4, 5],
             [7, 7, 7, 9, 9, 4, 5],
             [8, 7, 7, 9, 8, 4, 9],
             [8, 7, 7, 9, 9, 4, 9],
             [7, 0, 7, 9, 9, 4, 9]]
score:      [14, 8, 9, 9, 9, 10, 16]
parents:    [1, 2]
```

. . . . .

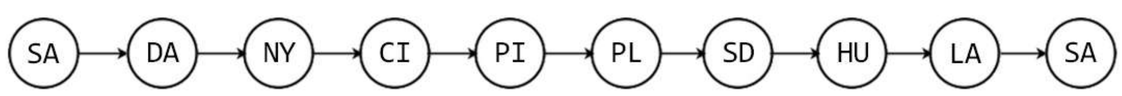
```
generation: 104
genome:     [[7, 7, 7, 7, 7, 7, 7],
             [7, 7, 7, 7, 7, 7, 7],
             [7, 7, 7, 7, 7, 7, 7],
             [7, 7, 7, 7, 7, 7, 7],
             [7, 7, 7, 7, 7, 7, 7],
             [7, 7, 7, 7, 7, 7, 7],
             [7, 7, 7, 7, 7, 7, 7]]
score:      [0, 0, 0, 0, 0, 0, 0]
parents:    [0, 1]
```

Genetic Algorithm

Travelling Salesman Problem



“ Business Trip Planner ”



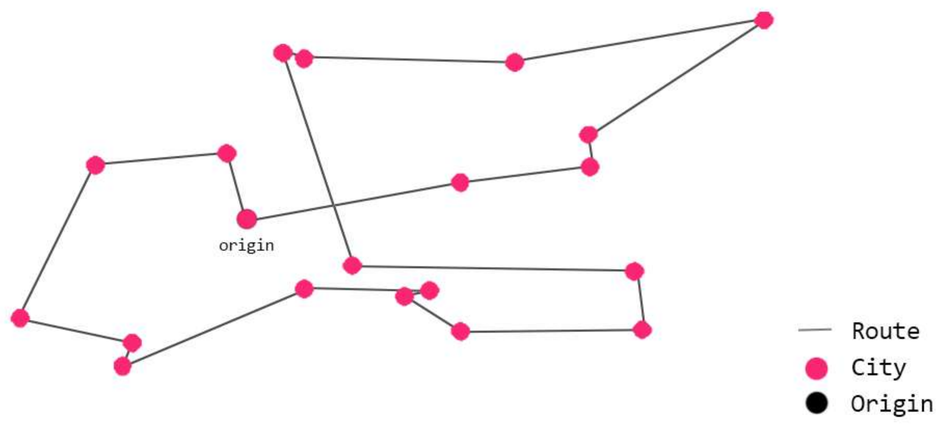

```
cities = ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Phoenix', 'Philadelphia', 'San Antonio', 'San Diego', ‘Dallas’]
```

Goal => Shortest Distance

“ Requirement ”

Problem Definition

After studying the concept of genetic algorithms earlier, I studied the TSP problem to study more applied problems.



This problem is to visit all cities only once and find the order of the minimum distance of moving back to the original starting point when there are several cities to travel and coordinates of cities to travel are given.

Population

Genome is the order of cities to visit when there are coordinates of a given city. And the beginning and last of the order were set to the origin so that the shape of the sequence became a ring shape.

Evaluate

Find the distance between the two cities based on the order of cities to visit (index) set by Genome.

Find the distance of all cities in Genome, and added all the distances calculated for each city. The calculated value was set to fitness value.

```
city_coordinates = [[217, 314], [328, 294], [394, 269], [272, 337], [485, 210],
                    [328, 371], [418, 340], [299, 353], [247, 230], [158, 377],
                    [139, 285], [153, 389], [100, 364], [395, 286], [247, 349],
                    [236, 227], [422, 370], [207, 279], [356, 232], [312, 350]]
```

```
genome = [0, 12, 10, 17, 14, 3, 11, 5, 19, 16, 8, 7, 1, 9, 15, 4, 6, 13, 2, 18, 0]
```

Genome => Order of Visit

$$d = \sqrt{\left(x_2 - x_1\right)^2 + \left(y_2 - y_1\right)^2}$$

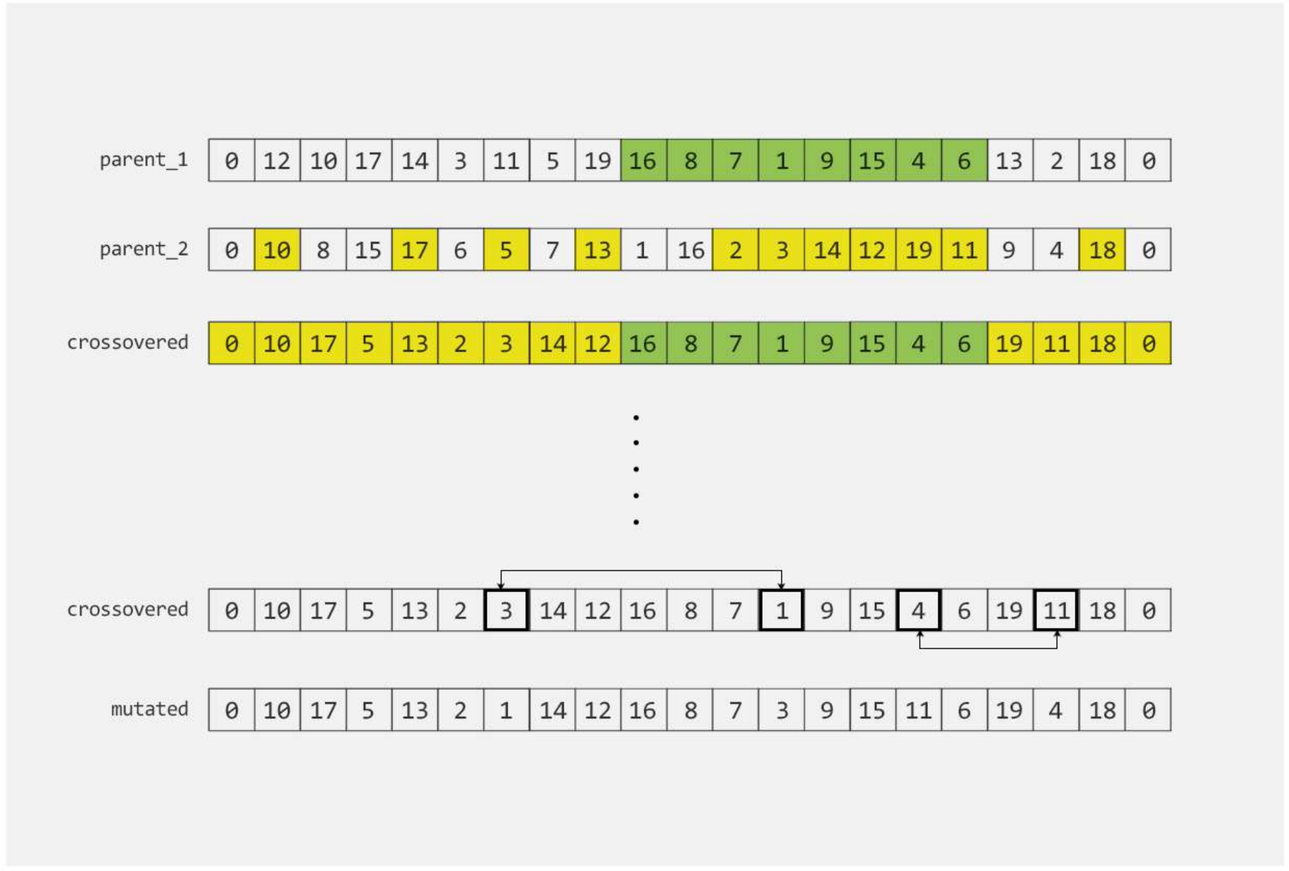
```
distance = [127.24, 88.1, 68.26, 80.62, 27.73, 129.87, 175.92, 26.4, 111.8, 224.11,
            133.54, 65.74, 189.18, 169.07, 249.58, 146.25, 58.69, 17.03, 53.04, 161.38]
```

```
fitness = sum(distance)
```

“ Evaluation Function ”

Order One Crossover

The crossover method used the order one crossover method. First, part of the parent_1 genome is randomly sliced and assigned to the offspring genome. And the rest are given genes from the parent_2 genome to create a offspring genome.



" Crossover Conception "

Mutation

In the Mutation step, a mutant dielectric is created by constructing a function that changes the order of the crossover offspring Genome.

TSP.py

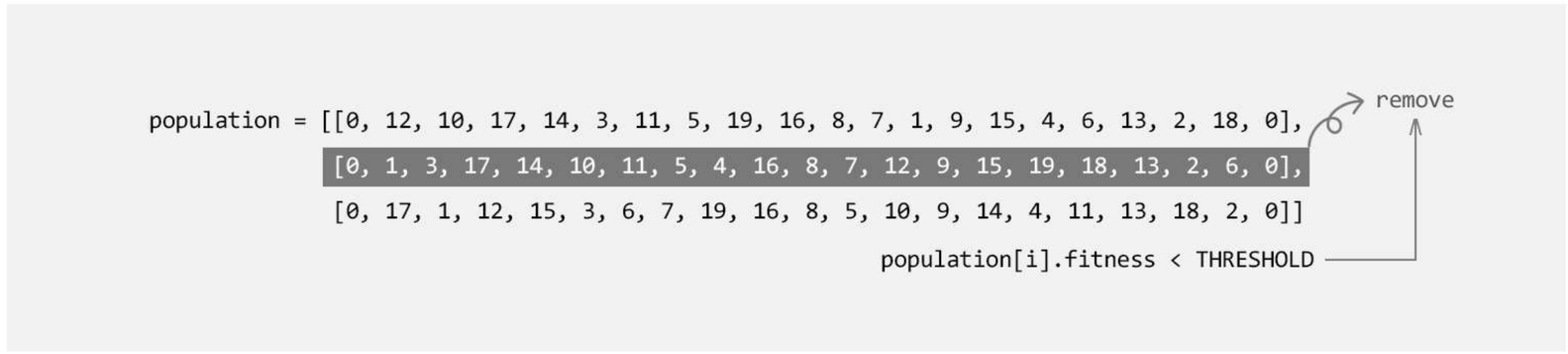
```python
class GeneticAlgorithm:
    def __init__(self, city_coordinates, population_size, generation_limit, mutation_rate, map_path, cities_count, weakness_threshold, chromosome_size):
        self.weakness_threshold = weakness_threshold
        self.city_coordinates = city_coordinates
        self.generation_limit = generation_limit
        self.chromosome_size = chromosome_size
        self.population_size = population_size
        self.mutation_rate = mutation_rate
        self.cities_count = cities_count
        self.map_path = map_path

    def evaluate_fittest(self, genome):
        fitnesses = []
        for chromosome in genome:
            fitness = chromosome.get_fitness()
            fitnesses.append(fitness)

        i = fitnesses.index(min(fitnesses))
        fittest = genome[i]
        return fittest

    def tournament_selection(self, genome, count):
        selected_genome = []
        for _ in range(count):
            i = random.randrange(len(genome))
            selected_genome.append(genome[i])

        fittest = self.evaluate_fittest(selected_genome)
        return fittest

    def reproduction_genome(self, genome):
        parent_1 = self.tournament_selection(genome, 20)
        parent_2 = self.tournament_selection(genome, 20)

        while parent_1 == parent_2:
            parent_2 = self.tournament_selection(genome, 20)

        parents = [parent_1, parent_2]
        offspring = self.crossover_genome(parents)
        return offspring

    def crossover_genome(self, parents):
        origin = 0
        blank = -1
        parent_1, parent_2 = parents
        offspring = [blank] * (len(parent_1) - parent_1.get_chromosome().count(origin))
        offspring.insert(0, origin)
        offspring.append(origin)

        start = len(parent_1) // 3
        end = len(parent_1) - 4

        start = random.randrange(start, end)

        order_genes = parent_1[start:end]
        offspring[start:end] = order_genes
        difference_genes = copy.deepcopy(parent_2)
        for gene in order_genes:
            i = difference_genes.get_chromosome().index(gene)
            difference_genes.get_chromosome()[i] = origin

        while origin in difference_genes:
            difference_genes.get_chromosome().remove(origin)

        for gene in difference_genes:
            i = offspring.index(blank)
            offspring[i] = gene

        if random.randrange(0,100) < self.mutation_rate:
            offspring = self.mutation_genome(offspring)

        offspring_genom = Genome()
        chromosome_object = Chromosome(self.city_coordinates)
        offspring_genom.set_chromosome(offspring)
        offspring_genom.set_fitness(chromosome_object.evaluate_chromosome(offspring_genom))

        return offspring_genom

    def mutation_genome(self, offspring):
        mutation_repeat_count = 2
        for _ in range(mutation_repeat_count):
            p1, p2 = [random.randrange(1, len(offspring)-1) for _ in range(2)]

            while p1 == p2:
                p2 = random.randrange(1, len(offspring)-1)

            offspring[p1], offspring[p2] = offspring[p2], offspring[p1]
        return offspring
```

Reproduction

Reproduction proceeds through the method described above. It is repeated as many times as the number of populations, and some of the results are as follows.

```
order_genes : [12, 19, 7]
parent_1 : [0, 9, 18, 6, 13, 8, 5, 17, 4, 2, 16, 3, 10, 11, 12, 19, 7, 1, 15, 14, 0]
parent_2 : [0, 12, 15, 10, 7, 13, 16, 1, 8, 4, 18, 5, 2, 6, 19, 14, 9, 11, 17, 3, 0]
offspring_genom : [0, 15, 6, 19, 16, 1, 8, 4, 18, 5, 2, 10, 14, 9, 12, 13, 7, 11, 17, 3, 0]

order_genes : [19, 14, 9]
parent_1 : [0, 12, 15, 10, 7, 13, 16, 1, 8, 4, 18, 5, 2, 6, 19, 14, 9, 11, 17, 3, 0]
parent_2 : [0, 9, 18, 6, 13, 8, 5, 17, 4, 2, 16, 3, 10, 11, 12, 19, 7, 1, 15, 14, 0]
offspring_genom : [0, 8, 6, 13, 18, 5, 17, 4, 2, 16, 12, 10, 11, 3, 19, 14, 9, 7, 1, 15, 0]

order_genes : [19, 14, 9]
parent_1 : [0, 12, 15, 10, 7, 13, 16, 1, 8, 4, 18, 5, 2, 6, 19, 14, 9, 11, 17, 3, 0]
parent_2 : [0, 12, 10, 17, 14, 3, 11, 5, 19, 16, 8, 7, 1, 9, 15, 4, 6, 13, 2, 18, 0]
offspring_genom : [0, 12, 1, 17, 3, 8, 5, 16, 11, 7, 10, 15, 4, 6, 19, 14, 9, 13, 2, 18, 0]

order_genes : [18, 17, 8, 5, 3, 14, 1, 15, 12, 10]
parent_1 : [0, 7, 2, 13, 6, 16, 19, 18, 17, 8, 5, 3, 14, 1, 15, 12, 10, 4, 9, 11, 0]
parent_2 : [0, 6, 7, 9, 11, 12, 19, 4, 2, 16, 17, 14, 10, 1, 5, 18, 13, 3, 8, 15, 0]
offspring_genom : [0, 6, 7, 9, 11, 19, 4, 18, 17, 8, 5, 3, 14, 1, 15, 12, 10, 2, 16, 13, 0]

order_genes : [3, 14, 1, 15, 12, 10]
parent_1 : [0, 7, 2, 13, 6, 16, 19, 18, 17, 8, 5, 3, 14, 1, 15, 12, 10, 4, 9, 11, 0]
parent_2 : [0, 8, 6, 13, 18, 5, 17, 4, 2, 16, 12, 10, 11, 3, 19, 14, 9, 7, 1, 15, 0]
offspring_genom : [0, 8, 6, 13, 18, 5, 17, 4, 2, 16, 11, 3, 14, 1, 15, 12, 10, 19, 9, 7, 0]

order_genes : [16, 2, 3, 14, 12, 19, 11]
parent_1 : [0, 10, 8, 15, 17, 6, 5, 7, 13, 1, 16, 2, 3, 14, 12, 19, 11, 9, 4, 18, 0]
parent_2 : [0, 12, 15, 10, 7, 13, 16, 1, 8, 4, 18, 5, 2, 6, 19, 14, 9, 11, 17, 3, 0]
offspring_genom : [0, 15, 10, 7, 13, 1, 8, 4, 18, 5, 16, 2, 3, 14, 12, 19, 11, 6, 9, 17, 0]

order_genes : [16, 8, 7, 1, 9, 15, 4, 6]
parent_1 : [0, 12, 10, 17, 14, 3, 11, 5, 19, 16, 8, 7, 1, 9, 15, 4, 6, 13, 2, 18, 0]
parent_2 : [0, 10, 8, 15, 17, 6, 5, 7, 13, 1, 16, 2, 3, 14, 12, 19, 11, 9, 4, 18, 0]
offspring_genom : [0, 10, 17, 5, 13, 2, 3, 15, 12, 16, 8, 1, 7, 9, 14, 4, 6, 19, 11, 18, 0]
```

" Reproduction "



“ Threshold Conception ”

Threshold

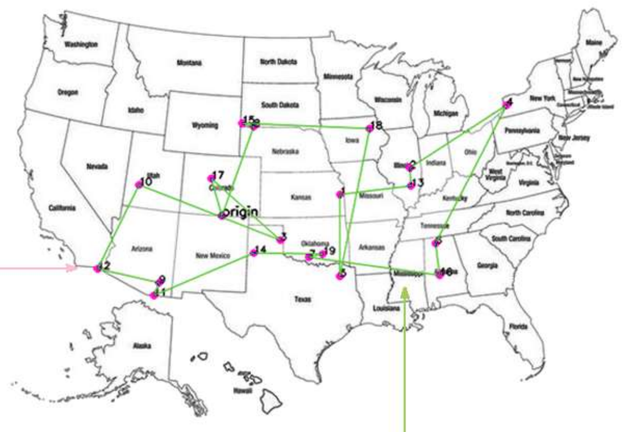
The fitness of the genome object in the population was checked and excluded from learning if it was lower than the set threshold.

TSP.py

```python
def main(self):

    RED = (0,0,255)
    BLUE = (255,0,0)
    title = 'TSP'
    text = 'Press any button to get started'

    travel_map = cv2.imread(self.map_path)
    for coord in self.city_coordinates:
        x, y = coord
        cv2.circle(travel_map, center=(x, y), radius=3, color=RED, thickness=-1, lineType=cv2.LINE_AA)

    cv2.putText(travel_map, org=(10, 35), text=text, fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.4, color=BLUE, thickness=1, lineType=cv2.LINE_AA)
    cv2.imshow(title, travel_map)
    cv2.waitKey(0)
    chromosome_object = Chromosome(self.city_coordinates)
    population = chromosome_object.configurate_chromosome(population_size=self.population_size, chromosome_size=self.cities_count)


    generation = 1
    while generation < self.generation_limit:

        for _ in range(self.population_size // 2):
            population.append(self.reproduction_genome(population))

        for genome in population:
            fitness = genome.get_fitness()

            if fitness > self.weakness_threshold:
                population.remove(genome)

        best_genome = self.evaluate_fittest(population)
        distance = best_genome.get_fitness()

        print("\n")
        print(f"Generation: {generation}\nDistance: {distance}\nBest Genome: {best_genome}")

        for i in best_genome:
            x, y = self.city_coordinates[i]
            cv2.circle(travel_map, center=(x, y), radius=3, color=RED, thickness=-1, lineType=cv2.LINE_AA)

            if i == 0:
                cv2.putText(travel_map, org=(x, y), text='origin', color=0, fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.4, lineType=cv2.LINE_AA)

            else:
                cv2.putText(travel_map, org=(x, y), text=f'{i}', color=0, fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.3, lineType=cv2.LINE_AA)

            fancy_coordinates = np.array(self.city_coordinates)[best_genome]
            for i in range(len(fancy_coordinates)-1):
                curr_point = fancy_coordinates[i]
                next_point = fancy_coordinates[i+1]
                cv2.line(travel_map, curr_point, next_point, BLUE, 1, cv2.LINE_AA)

            cv2.putText(travel_map, org=(10, 25), text=f'generation: {generation}', fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.4, color=0, thickness=1, lineType=cv2.LINE_AA)
            cv2.putText(travel_map, org=(10, 45), text=f'distance: {distance}', fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.4, color=BLUE, thickness=1, lineType=cv2.LINE_AA)
            cv2.putText(travel_map, org=(10, 65), text=f'best route: {best_genome}', fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.4, color=0, thickness=1, lineType=cv2.LINE_AA)

            cv2.imshow(title, travel_map)
            if cv2.waitKey(1) == ord('q'):
                break

            travel_map = cv2.imread(self.map_path)

            generation += 1
            if generation == 4:
                break

        cv2.waitKey(0)


if __name__ == "__main__":

    random.seed(777)

    CITIES_COUNT = 20
    POPULATION_SIZE = 100
    GENERATION_LIMIT = 2000
    MUTATION_RATE = 65
    WEAKNESS_THRESHOLD = 1400

    # CITIES_COUNT = 20
    # POPULATION_SIZE = 100
    # GENERATION_LIMIT = 2000
    # MUTATION_RATE = 65
    # WEAKNESS_THRESHOLD = 1500

    map_path = 'america.jpg'


    city_coordinates = []
    for i in range(CITIES_COUNT):
        x = random.randint(100, 500)
        y = random.randint(200, 400)
        city_coordinates.append([x, y])

    ga = GeneticAlgorithm(weakness_threshold=WEAKNESS_THRESHOLD,
                          generation_limit=GENERATION_LIMIT,
                          population_size=POPULATION_SIZE,
                          chromosome_size=CITIES_COUNT,
                          mutation_rate=MUTATION_RATE,
                          cities_count=CITIES_COUNT,
                          city_coordinates=city_coordinates,
                          map_path=map_path)

    ga.main()
```

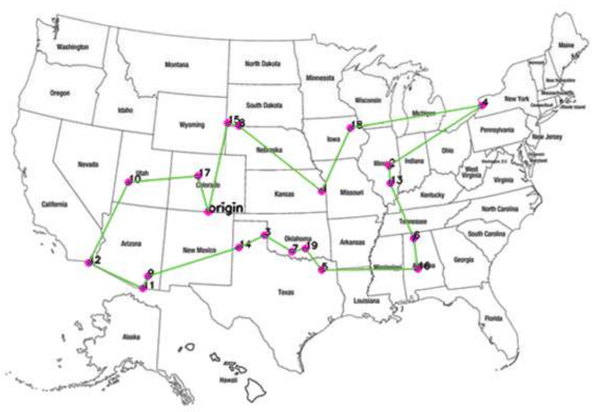
Visualization

Using openCV, the coordinates of each city are expressed in dots and the order of the cities visited is indicated by lines.



Result

Visualized results can be found in the image below. After setting the threshold, the minimum distance was found faster when the algorithm was executed.



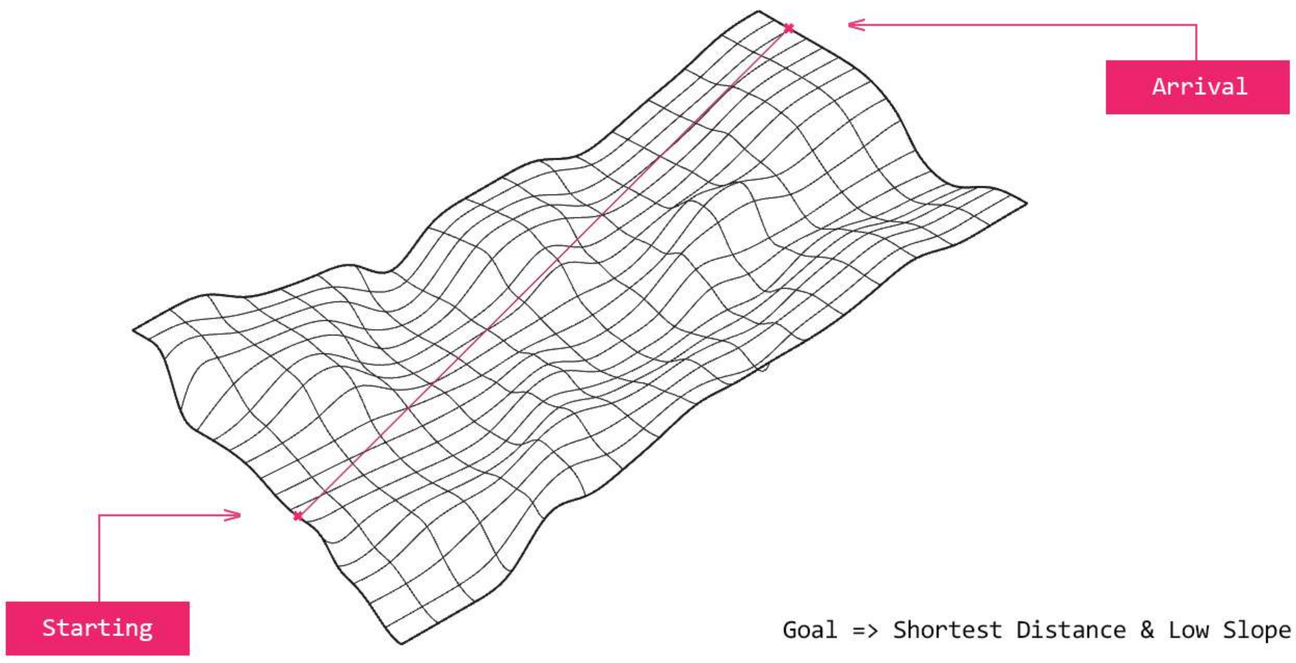
VS

“ No Threshold ”

“ With Threshold ”

Path Finder

Minimize the Slope and Distance using GALAPAGOS



Goal => Shortest Distance & Low Slope

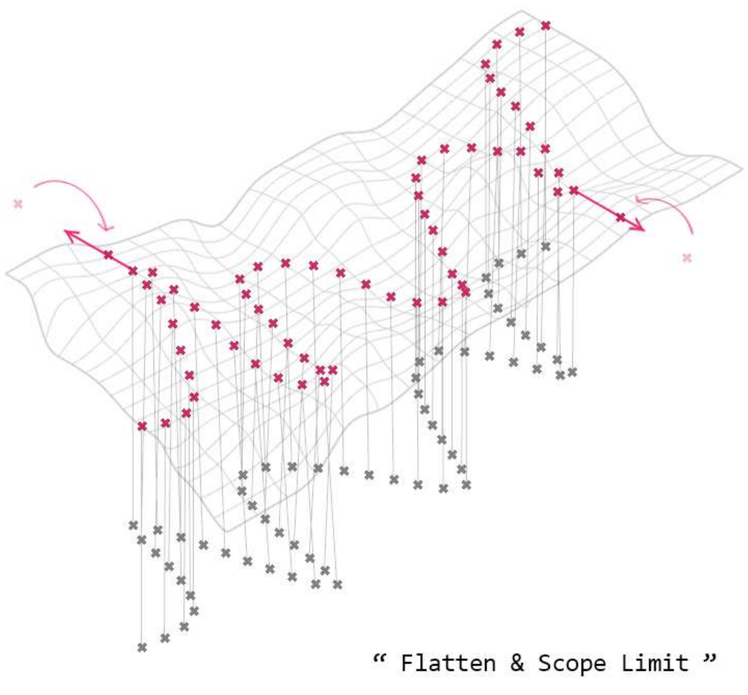
Overview

I worked in the Rhino environment to apply the 2D TSP to the 3D space.

This project uses ghPython and GALAPAGOS to explore the lowest slope and the shortest path when the slope topography, starting point, and arrival point are provided.

Preprocess

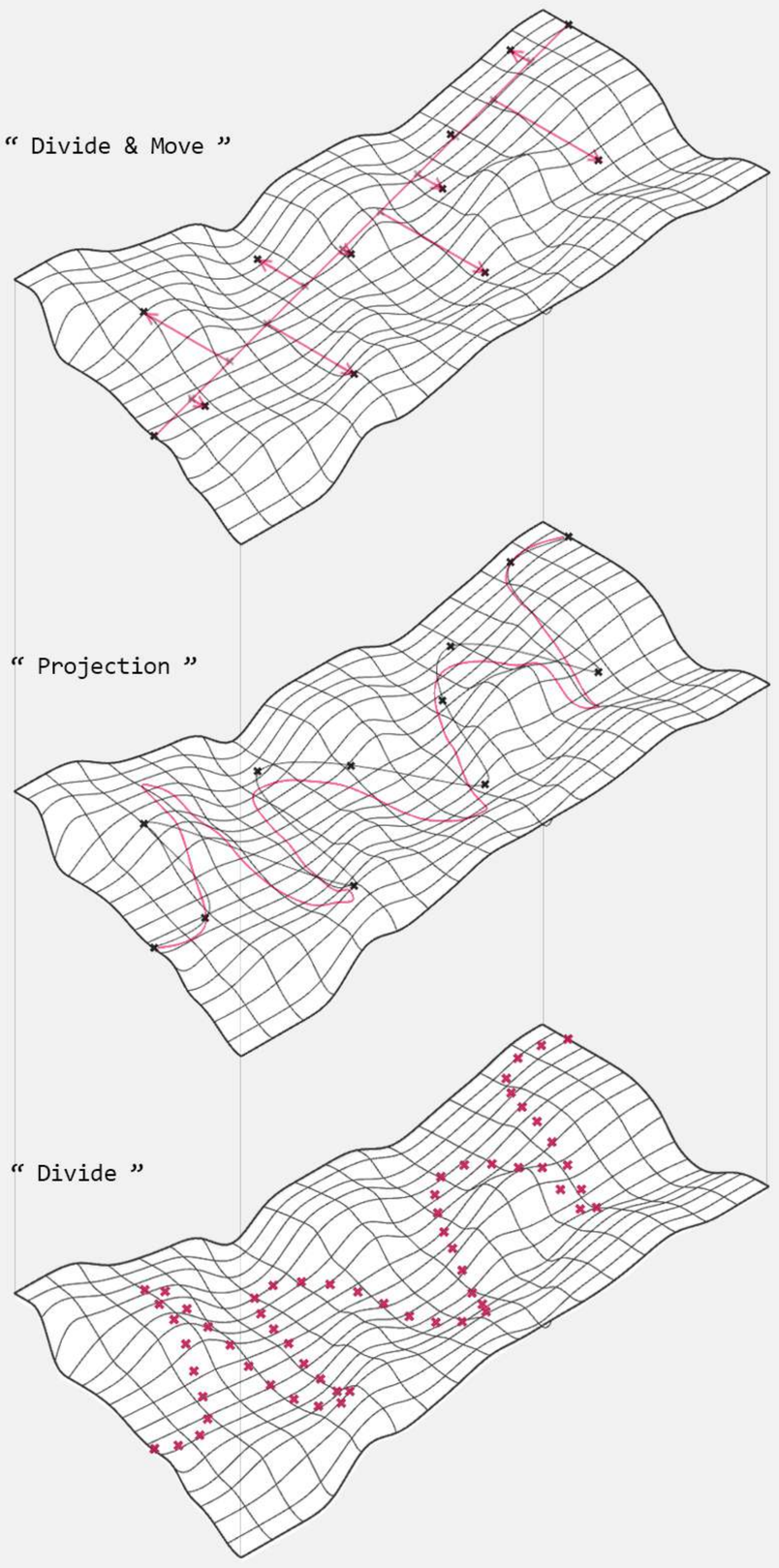
Before creating the evaluation function, the slope of each point needs to be calculated, so the preprocessing process shown in the image on the right was performed.



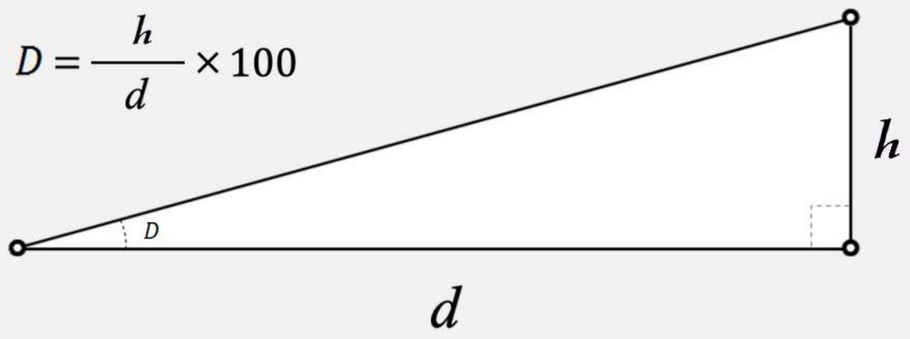
“ Flatten & Scope Limit ”

First, connect the starting point and arrival point with a straight line. After that, the straight line is divided into N lines and the obtained points are moved in the X-axis direction of the topography.

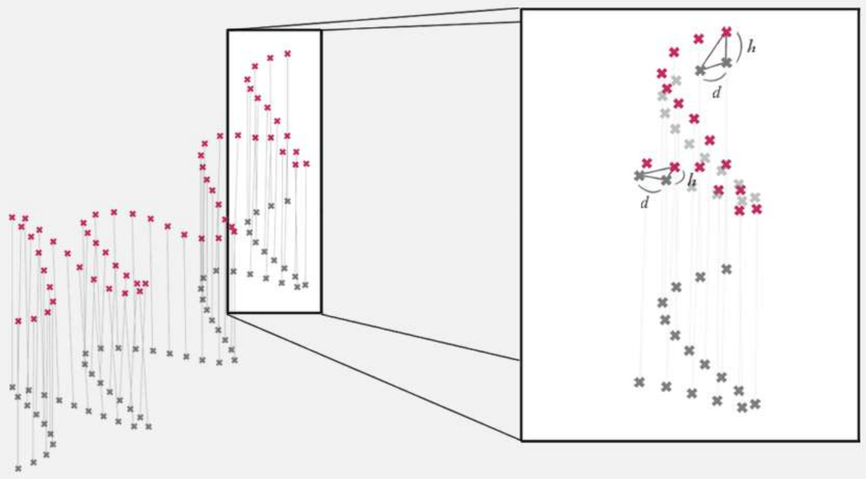
Create an interpolation curve through the moved points, divide them again, and project the points on the sloped topography.



“ Preprocess Diagram ”



“ Slope Calculation Expression ”



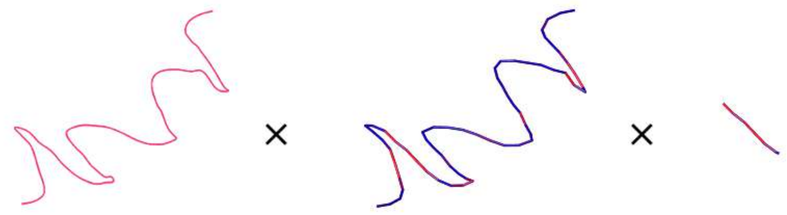
“ Calculate Degree of Slope ”

Degree of Slope

Through the points constructed in the pre-processing process, I was able to obtain the slope of each point of the sloped topography using the expression to obtain the slope.

Multiple Fitness

The purpose of this project is to explore the shortest of the gentle slopes. For this reason, we derived several types of Fitness as shown below and evaluated each condition.



```
fitness = path_length * average_slope * max_slope
```

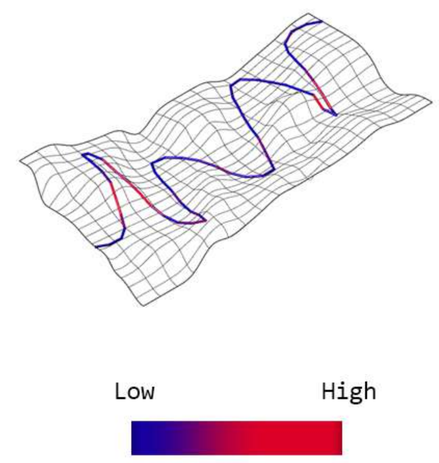
“ Evaluation Conception ”

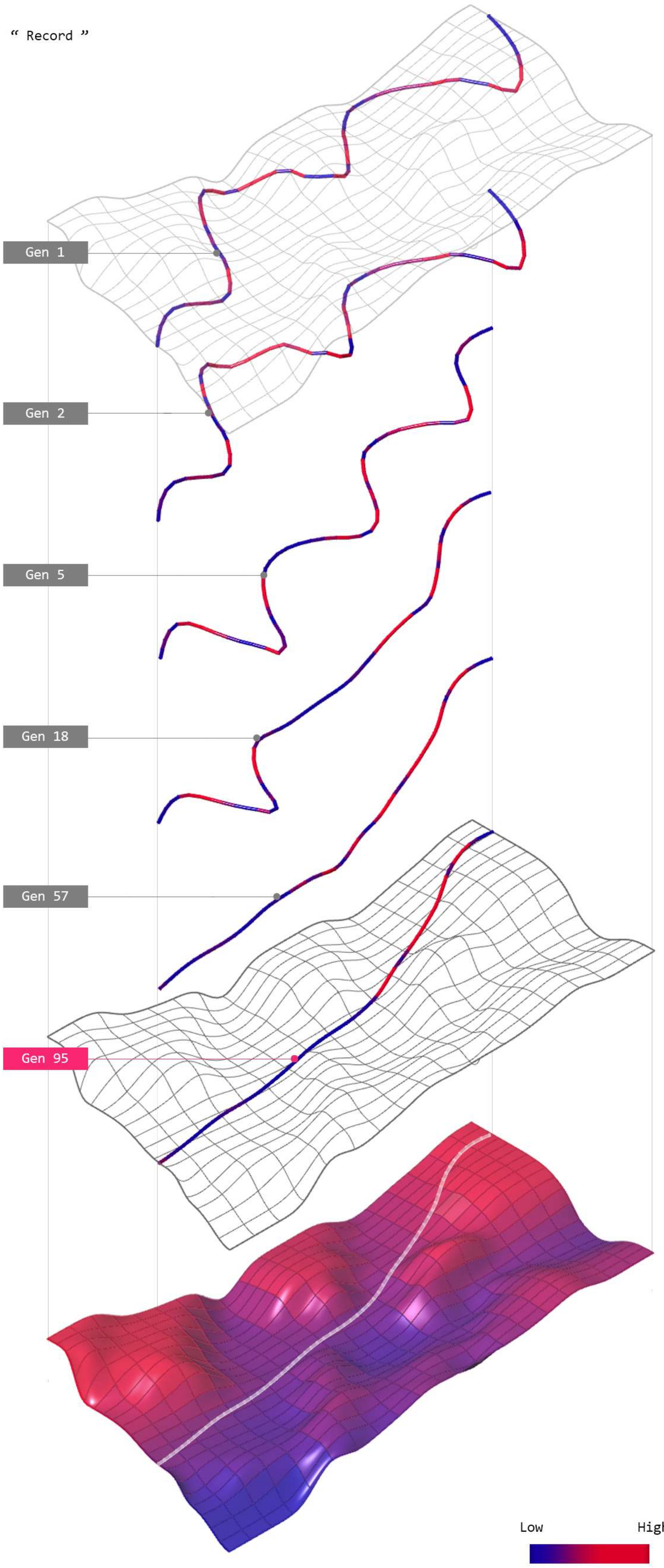
path_finder.py

```python
class PathFinder:
    def __init__(self, mountain, start_point, end_point, gene):
        self.mountain = mountain
        self.start_point = start_point
        self.end_point = end_point
        self.gene = gene

        self.utils = Utils()

    def get_mountain(self):
        return self.mountain

    def get_start_point(self):
        return self.start_point

    def get_end_point(self):
        return self.end_point

    def get_gene(self):
        return self.gene

    def calculate_slope(self, pt_1, pt_2):
        x1, y1, z1 = pt_1
        x2, y2, z2 = pt_2
        distance = ((x2-x1)**2 + (y2-y1)**2) ** 0.5
        slope_degree = abs((z2-z1)/distance * 100)
        return slope_degree

    def calculate_min_max(self, degree_of_slope):
        return min(degree_of_slope), max(degree_of_slope)

    def calculate_average(self, degree_of_slope):
        return sum(degree_of_slope) / len(degree_of_slope)

    def limit_point(self):
        mountain_points = self.utils.surface_points(self.mountain)
        x_coordinates = []

        for point in mountain_points:
            x = point[0]
            if x not in x_coordinates:
                x_coordinates.append(x)

        min_x = min(x_coordinates)
        max_x = max(x_coordinates)
        return min_x, max_x

    def evaluate_fitness(self, degree_of_slope, length_of_path):
        average_degree = self.calculate_average(degree_of_slope)
        _, max_degree = self.calculate_min_max(degree_of_slope)
        return average_degree * max_degree * length_of_path

    def main(self):
        STRAIGHT_SEG_COUNT = 11
        PROJECTION_SEG_COUNT = 60
        MARGIN = 20
        RADIUS = 3

        min_x, max_x = self.limit_point()

        straight = self.utils.generate_line(self.start_point, self.end_point)
        divide_straight = self.utils.divide_curve(straight, STRAIGHT_SEG_COUNT)
        for i, point in enumerate(divide_straight[1:-1]):
            gene = self.get_gene()[i]

            translation = [gene, 0, 0]
            self.utils.move_object(point, translation)
            if point[0]-MARGIN < min_x:
                point[0] = min_x+MARGIN
            elif point[0]+MARGIN > max_x:
                point[0] = max_x-MARGIN

        interpolate_curve = self.utils.interpolate_curve(divide_straight)
        projection_curve = self.utils.projection_curve(interpolate_curve, self.get_mountain())

        divide_projection = self.utils.divide_curve(projection_curve, PROJECTION_SEG_COUNT)
        length_of_path = 0
        degree_of_slope = []
        path = []
        for i in range(1, len(divide_projection)):
            pt_1 = divide_projection[i-1]
            pt_2 = divide_projection[i]

            line = self.utils.generate_line(pt_1, pt_2)
            pipe = self.utils.curve_pipe(line, RADIUS)[0]
            path.append(pipe)

            line_length = self.utils.length_curve(line)
            length_of_path += line_length

            degree = self.calculate_slope(pt_1, pt_2)
            degree_of_slope.append(degree)

        fitness = self.evaluate_fitness(degree_of_slope, length_of_path)
        average_degree = self.calculate_average(degree_of_slope)
        min_degree, max_degree = self.calculate_min_max(degree_of_slope)
        annotation = "fitness: {}\naverage degree: {}\nmax degree: {}\nlength of path: {}" \
                     .format(fitness, average_degree, max_degree, length_of_path)

        return path, fitness, degree_of_slope, average_degree, min_degree, max_degree, length_of_path, annotation

if __name__ == "__main__":
    utils = Utils()

    translation = [0,0,0]                                                    # convert surface to guid
    mountain = utils.move_object(mountain, translation)                      # convert surface to guid

    pf = PathFinder(mountain, start_point, end_point, gene)

    path, fitness, degree_of_slope, average_degree, min_degree, max_degree, length_of_path, annotation = pf.main()
```

Visualization

By dividing the path into N equal parts, each point was connected with a straight line and the slope of each section was visualized.





Record

Genomes were generated by generation through GALAPAGOS and each generation was recorded. Haven't found a better year since 'Gen 95' and Fitness is higher than before

```
generation : 1
average degree : 22.97
max degree : 63.99
length of path : 1483.2
fitness : 2180259.73
genome : [27, -14, 87, 167, 87, -12, 59, 179, 0, -139]

generation : 2
average degree : 23.59
max degree : 63.29
length of path : 1505.52
fitness : 2248712.86
genome : [28, -12, 88, 168, 88, -20, 60, 180, 0, -144]

generation : 5
average degree : 15.55
max degree : 45.833
length of path : 1388.69
fitness : 990027.09
genome : [20, -144, -32, 4, -40, -80, 20, 16, -88, 12]

generation : 18
average degree : 12.32
max degree : 37.86
length of path : 1206.54
fitness : 563099.46
genome : [4, -132, -12, -20, -32, -40, -36, -52, -20, 24]

generation : 57
average degree : 11.02
max degree : 29.78
length of path : 1053.96
fitness : 345989.26
genome : [0, -12, -16, -24, -48, -40, -36, -24, -12, 20]

generation : 95
average degree : 22.97
max degree : 9.07
length of path : 1047.1
fitness : 280731.97
genome : [-4, -8, -20, -24, -28, -40, -36, -24, -12, 16]
```

Result

To evaluate the optimal solution derived through the genetic algorithm, the slope of the topography was visualized and overlapped. It's not a quantitative evaluation, but it appears to have yielded an appropriate solution.

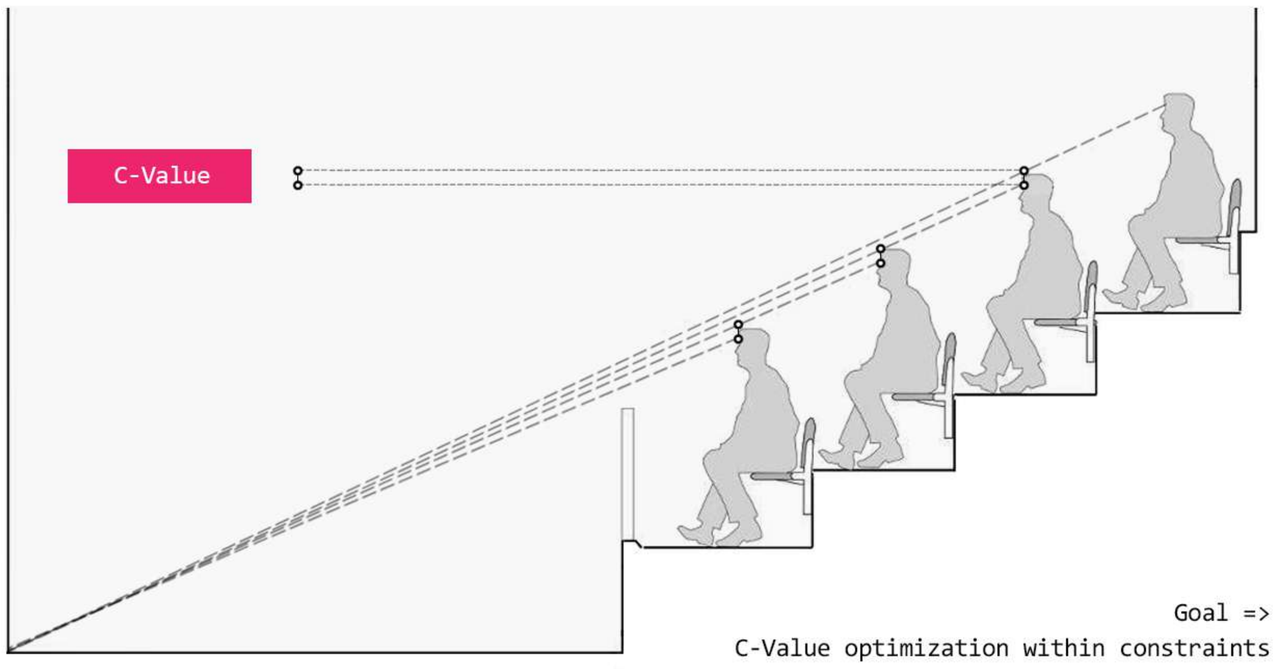
Theater Sightlines

C - Value Optimization Using GALAPAGOS

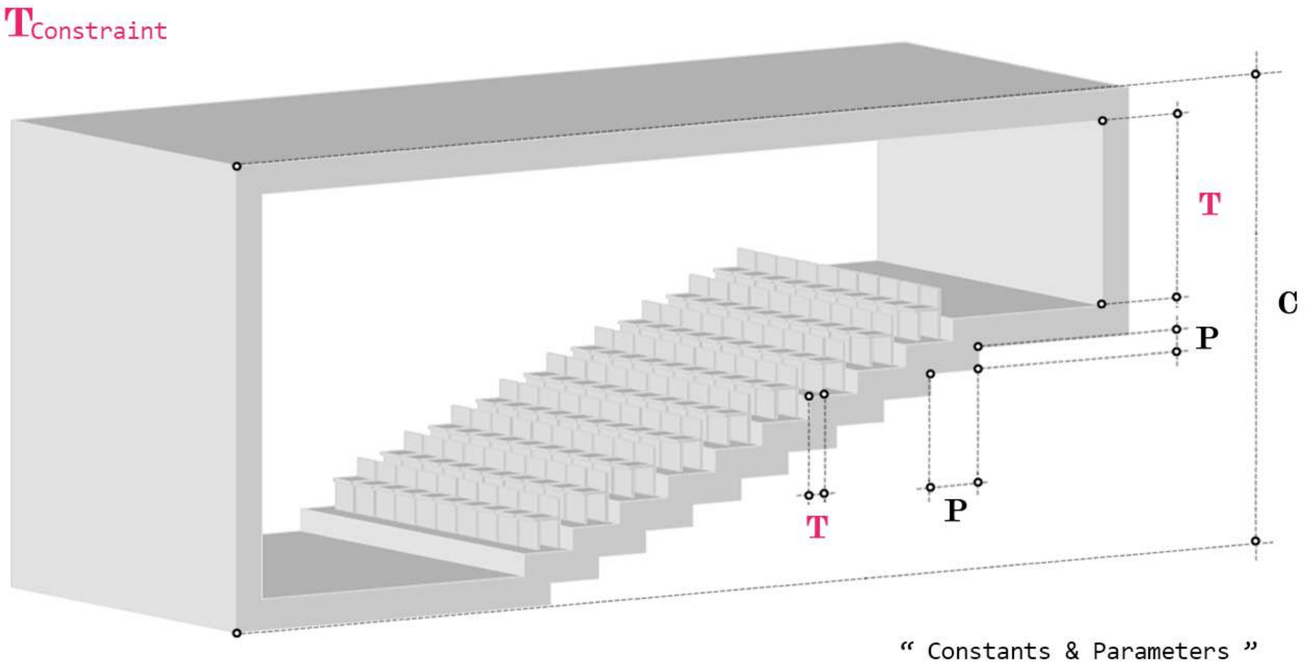
C - Value

When you go to the theater, have you ever had any inconvenience in viewing because of the back of the person in front of you? C - Value is a measure of the quality of viewing in a theater.

This project aims to quantitatively evaluate and optimize viewing quality in theaters using Galapagos and ghPython.



C Constant
P Parameter
T Constraint



" Constants & Parameters "

State Space Design

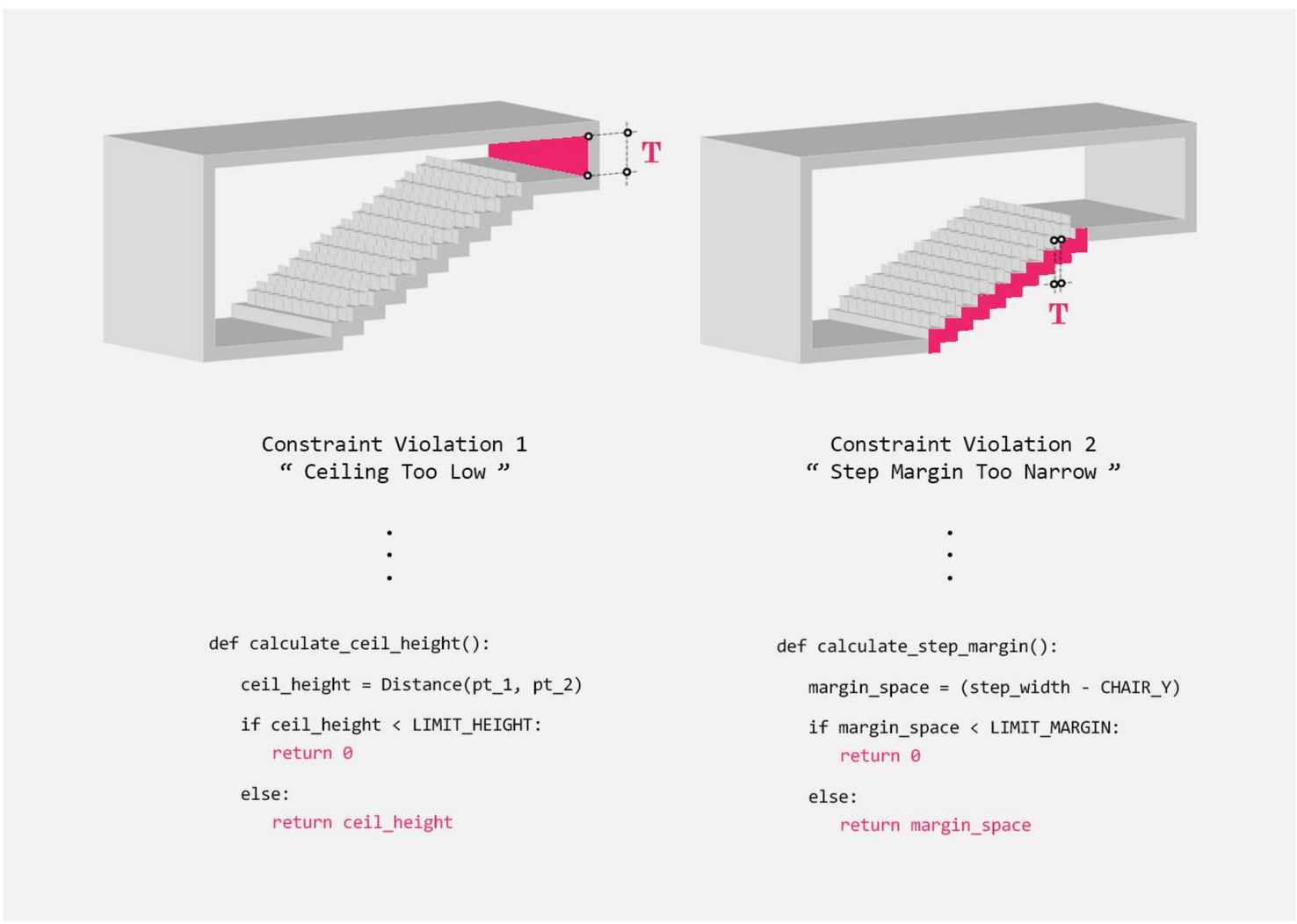
To evaluate Fitness, I designed State Space based on predefined parameters and constants.

And, I set constraints to specify the scope of the learning. As you can see in the image on the left, I wanted to derive the optimal C-Value within the constraints by putting the ceiling height of the theater and the width of the stairs as constraints.

Constraints

The C-Value is determined by the width and height of the stairs. If there is no constraint set above, a high and narrow stairs will be created unconditionally by the genetic algorithm.

Therefore, I wrote an exception handling conditional statement in the code of ghPython. Since the purpose of this project is to maximize C-Value, I set it to return a very low value if a constraint is violated.




```
def calculate_ceil_height():
    ceil_height = Distance(pt_1, pt_2)
    if ceil_height < LIMIT_HEIGHT:
        return 0
    else:
        return ceil_height
```

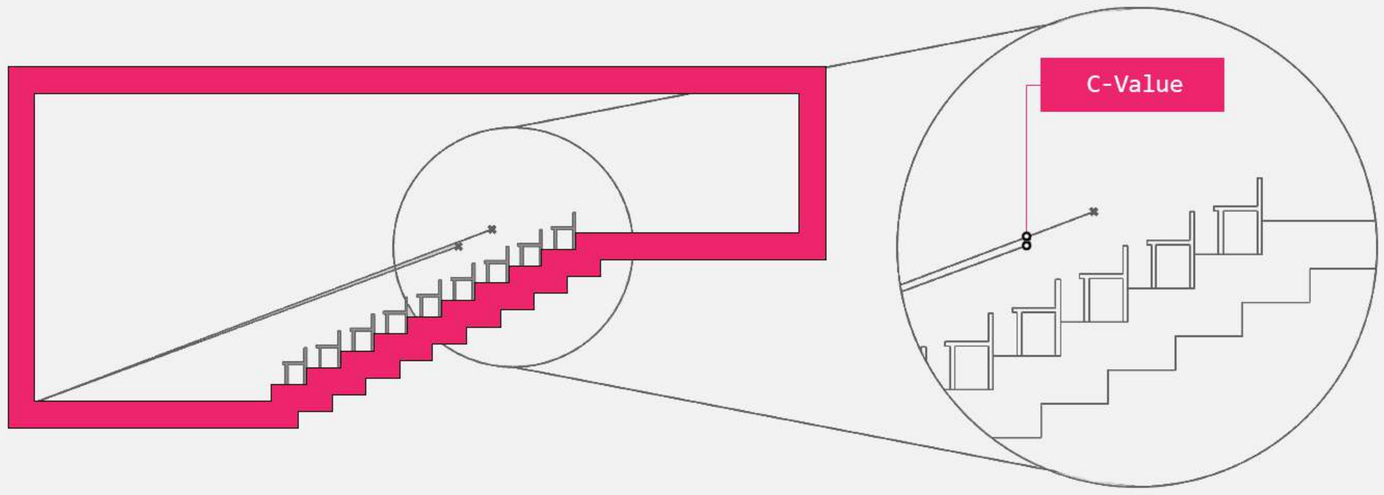
```
def calculate_step_margin():
    margin_space = (step_width - CHAIR_Y)
    if margin_space < LIMIT_MARGIN:
        return 0
    else:
        return margin_space
```

" Constraint Violation "

Evaluate

Based on the previously defined constraints and the return value when the constraints are violated, the C-Value, the fitness of the project, was evaluated for each State Space.

C-Value can be evaluated by finding the distance between two sight points, and GALAPAGOS is used to replace the genetic algorithm code.




```
def calculate_cvalue():

    curr_ceil_height = calculate_ceil_height()
curr_step_width = calculate_step_margin()

                .
                .
                .

if curr_ceil_height == 0 or curr_step_width == 0:
    cvalue = 0
else:
    cvalue = rs.Distance(cal_pt_1, cal_pt_2
return round(cvalue, 1)
```

“ Sectional State ”

theater_sightlines.py

```
140     def generate_section(self):
141         cutter_crv = Curve(self.CUTTER_PTS).generate_curve()
142         cutter = Surface(cutter_crv).generate_surface()
143         cutting_idx = [i for i in range(5, 95, 10)]
144         section_lines = []
145         for i in cutting_idx:
146             split_chair = rs.SplitBrep(self.chairs[i], cutter)[1];\
147                           rs.CapPlanarHoles(split_chair)
148             section_chair = self.elements_auditorium(split_chair)[0][0]
149             section_lines.append(section_chair)
150
151         split_auditorium = rs.SplitBrep(self.auditorium, cutter)[1];\
152                            rs.CapPlanarHoles(split_auditorium);
153         section_auditorium = self.elements_auditorium(split_auditorium)[0][0]
154         section_lines.append(section_auditorium)
155         rs.MoveObjects(section_lines, self.SECTION_M)
156         return section_lines
157
158     def generate_sight(self):
159         section_lines = self.generate_section()[-1]
160         sitting_lines = self.elements_auditorium(section_lines)[1]
161         base_line = self.elements_auditorium(section_lines)[1][24]
162         sight_points = gh.CurveMiddle([sitting_lines[37], sitting_lines[39]]);\
163                        rs.MoveObjects(sight_points, [0,0,self.SIGHT_LEVEL]);
164         focus_point = rs.CurvePoints(base_line)[0]
165
166         sight_curve = Curve([sight_points[0], focus_point, sight_points[1]]).generate_curve()
167         return [sight_curve, sight_points[0], sight_points[1]]
168
169     def calculate_ceil_height(self):
170         ceil_height = rs.Distance(self.base_pts[-3], self.base_pts[-4]) * self.SCALE
171
172         if ceil_height < self.LIMIT_HEIGHT * self.SCALE:
173             return 0
174         else:
175             return ceil_height
176
177     def calculate_step_margin(self):
178         margin_space = (self.step_width - self.CHAIR_Y) * self.SCALE
179
180         if margin_space < self.LIMIT_MARGIN * self.SCALE:
181             return 0
182         else:
183             return margin_space
184
185     def calculate_cvalue(self):
186         curr_ceil_height = self.calculate_ceil_height()
187         curr_step_width = self.calculate_step_margin()
188
189         sight_elements = self.generate_sight()
190         cal_crv = sight_elements[0]
191         cal_pt_1 = sight_elements[1]
192         cal_vect = Point(0,0,1).generate_point()
193         cal_line = gh.LineSDL(cal_pt_1, cal_vect, 30)
194         cal_pt_2 = gh.CurveXLine(rs.coercecurve(cal_crv), cal_line)[0][1]
195
196         if curr_ceil_height == 0 or curr_step_width == 0:
197             cvalue = 0
198         else:
199             cvalue = rs.Distance(cal_pt_1, cal_pt_2) * self.SCALE
200         return round(cvalue, 1)
201
202 >   def visualization_circle(self): ···
216
217     def visualization_zoom(self):
218         circle_list = self.visualization_circle()
219         scaled_origin = rs.coerce3dpoint(gh.Area(rs.coercegeometry(circle_list[1]))[1])
220         merge_curve = gh.Merge(gh.DeconstructBrep(self.generate_section())[1], self.generate_sight()[0])
221         merge_curve[-1] = rs.coercecurve(merge_curve[-1])
222
223         trimed_region = circle_list[0]
224         trimed_curves = gh.TrimwithRegion(merge_curve, trimed_region, rg.Plane.WorldZX)[0];\
225                         rs.MoveObjects(trimed_curves, [130,0,0])
226         trimed_curves = gh.Scale(trimed_curves, scaled_origin, 2)[0]
227         sight_curve = rs.coercecurve(self.generate_sight()[0]);\
228                       rs.MoveObject(sight_curve, [130,0,0])
229         sight_curve = gh.Scale(sight_curve, scaled_origin, 2)[0]
230         sight_points = gh.EndPoints(sight_curve)
231         cal_vect = Point(0,0,1).generate_point()
232         cal_line = gh.LineSDL(sight_points[0], cal_vect, 30)
233         cvalue_point = gh.CurveXCurve(sight_curve, cal_line)[0]
234         cvalue_line = Curve([cvalue_point[0], cvalue_point[1]]).generate_curve()
235
236         trimed_curves.extend([cvalue_line, cvalue_point[0], cvalue_point[1], sight_points[1]])
237
238         return trimed_curves
239
```

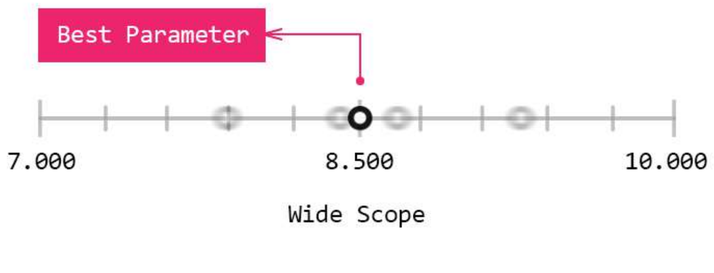
Record

Through the GALAPAGOS Record written below, you can see the C-Value that increases with each generation.

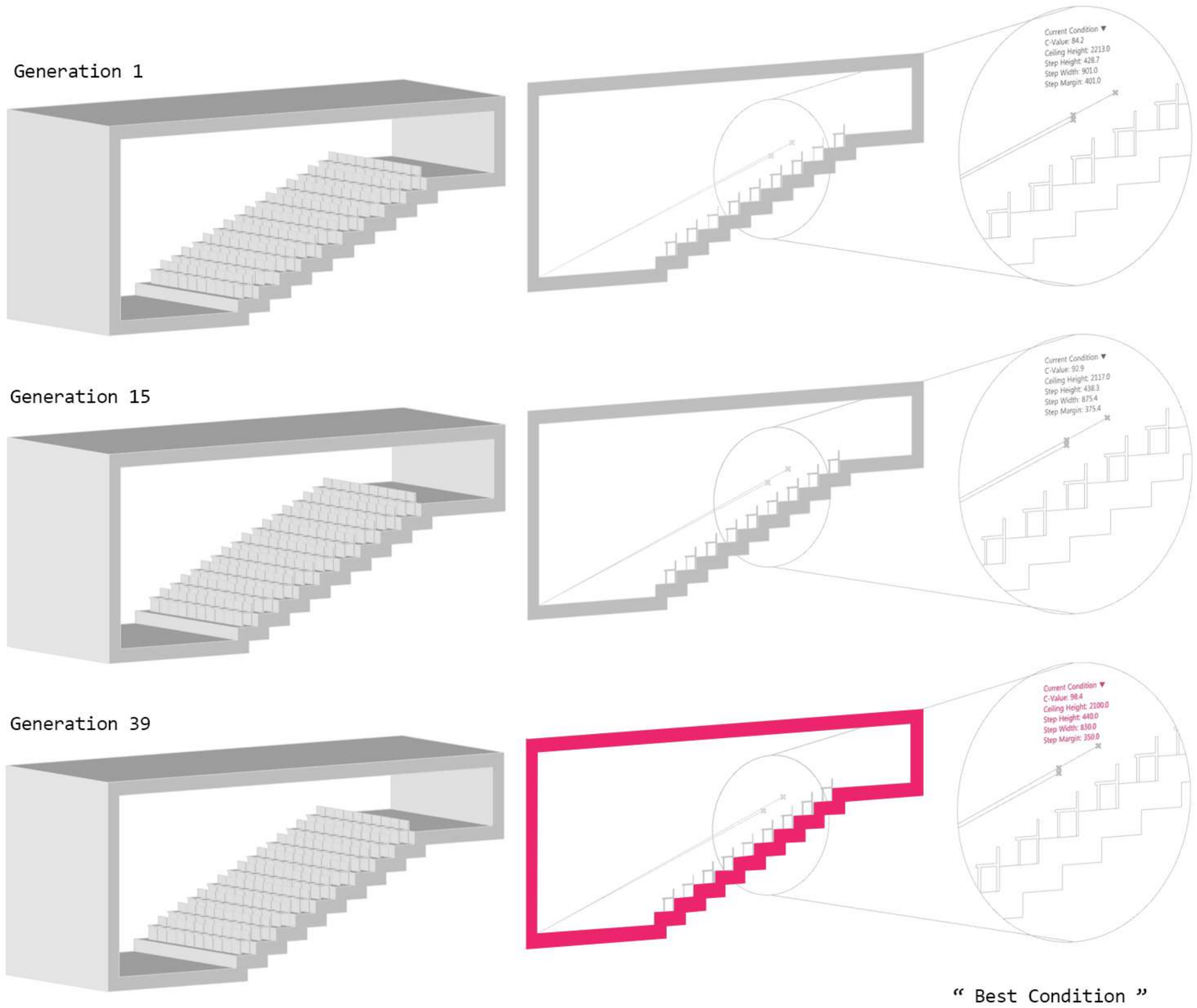
```
generation : 1
C-Value : 84.2
Ceiling Height : 2213.0
Step Height : 428.7
Step Width : 901.0
Step Margin : 401.0
Genome : [9.010, 4.287]

generation : 7
C-Value : 88.5
Ceiling Height : 2213.0
Step Height : 428.7
Step Width : 877.0
Step Margin : 377.0
Genome : [8.770, 4.287]

generation : 15
C-Value : 92.9
Ceiling Height : 2117.0
Step Height : 438.3
Step Width : 875.4
Step Margin : 375.4
Genome : [8.754, 4.383]

generation : 24
C-Value : 97.4
Ceiling Height : 2111.0
Step Height : 438.9
Step Width : 852.6
Step Margin : 352.6
Genome : [8.526, 4.389]

generation : 39
C-Value : 98.4
Ceiling Height : 2100.0
Step Height : 440.0
Step Width : 850.0
Step Margin : 350.0
Genome : [8.500, 4.400]
```

Result

This project was a problem that could be answered intuitively. What I wanted to check through this work was whether it was possible to set the desired constraints and learn in the desired direction.



Therefore, the exploration scope of the parameters was extended beyond the limit and the work was carried out. I got the results I wanted with the GALAPAGOS



Criterion

The evaluation of the C-Value that can be obtained within the constraint is shown in the table on the right.

C-Value	*Annotation*
60	*Need to look between heads in front*
90	*Can see well with head tilted backwards*
120	*Optimal viewing standard*
150	*Can see well even if over spectators with hats*

generation : 39

C-Value : 98.4

Ceiling Height : 2100.0

Step Height : 440.0

Step Width : 850.0

Step Margin : 350.0

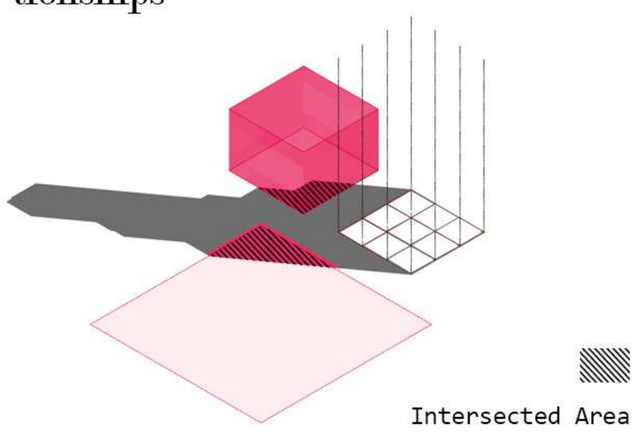
Genome : [8.500, 4.400]

Shadow of Neighbor

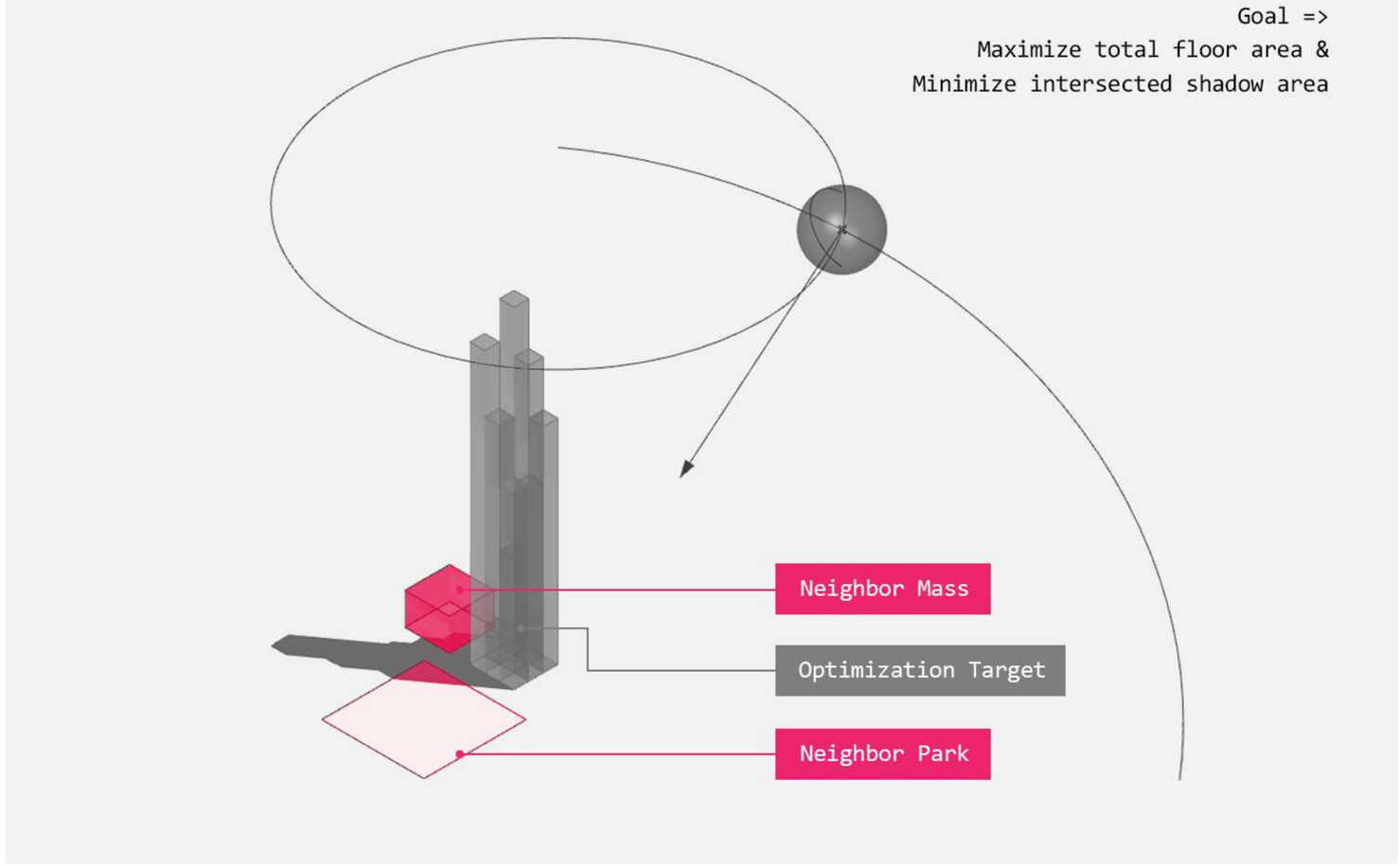
Shadow Area of Building Minimize Using GALAPAGOS

Subject

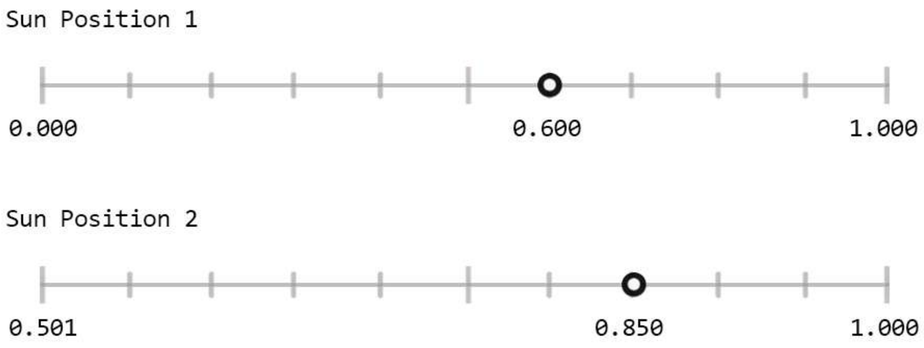
“ Architecture is the product of relationships ”



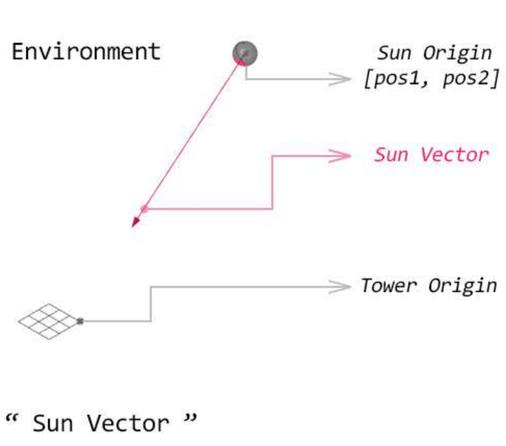
How should the shape of the building be changed to minimize shadows from neighbors and maximize the total floor area?



“ Rhino Viewport”



“ Fix The Sun Position ”



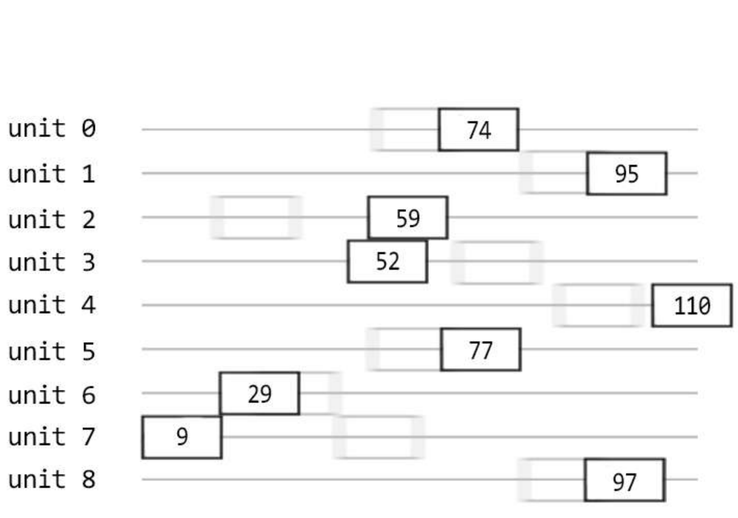
“ Sun Vector ”

Environment

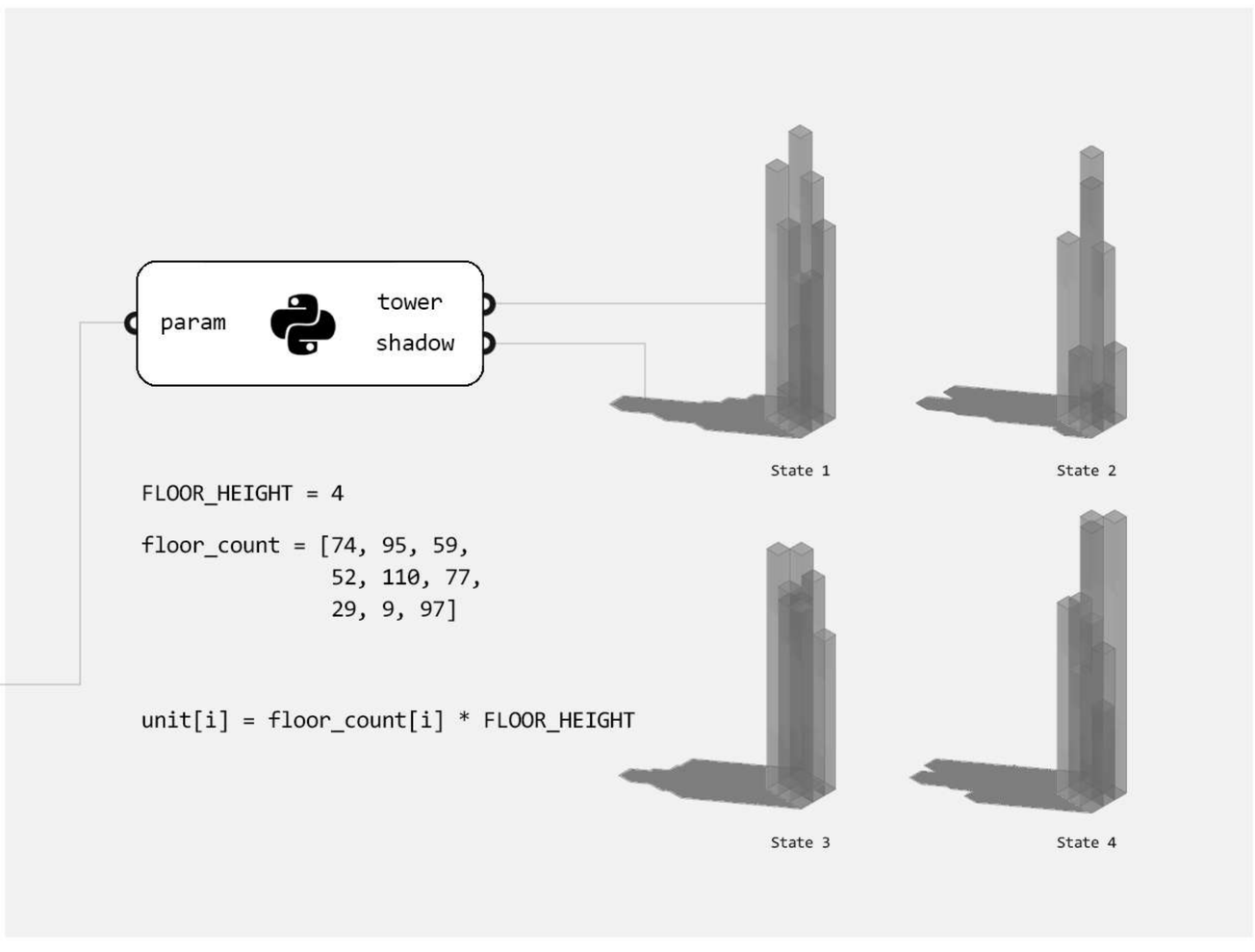
To compose the State Space, I created an environment that can create shadows. And I made a vector with the origin of the building and the origin of the sun.

Parameter

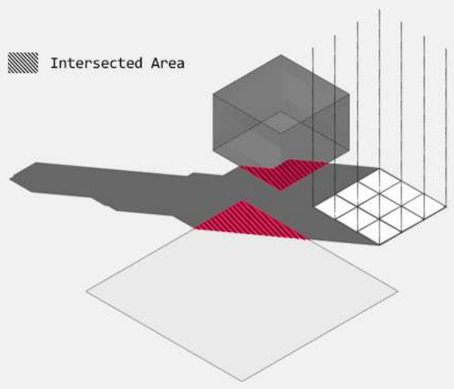
The parameter was set to the number of floors in each unit of the building. When the Python code is executed by inputting parameters, the shape of the building is created by the FLOOR_HEIGHT set as a variable.



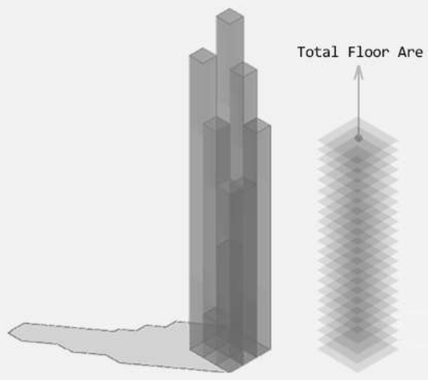
“ Parameter ”



“ State Configuration ”



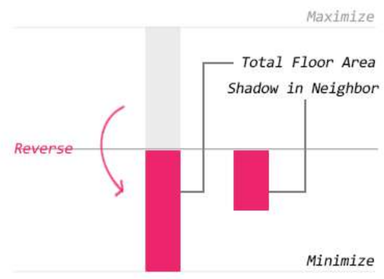
Minimize Traget



Maximize Target

Negative Fitness

Because there are things that need to be maximized and things that need to be minimized at the same time, we have unified the direction of fitness.



“ Optimization Target ”

```
fitness  = -(total_floor_area * shadow_in_neighbor)
```

building_shadow_in_park.py

```python
class Region:
    def __init__(self, curves): ...

    def evaluate_type(self, curves): ...

    def generate_union(self):
        return gh.RegionUnion(self.curves, self.plane)

    def generate_intersection(self):
        return gh.RegionIntersection(self.curves[0], self.curves[1], self.plane)

    def area_region(self):
        return round(gh.Area(self.curves)[0], 0)

    def area_intersection(self):
        try:
            intsc_area = gh.Area(self.intsc)[0]
            if type(intsc_area) == list:
                intsc_area = sum(intsc_area)
            return round(intsc_area, 0)
        except:
            return 0


class Sun(Sphere):
    def __init__(self, origin, radius, position): ...

    def split_sphere(self): ...

    def generate_path(self):
        if type(self.split_sphere) == type(rs.AddPoint(0,0,0)):
            self.split_sphere = rs.coercebrep(self.split_sphere)
        uv = rs.SurfaceParameter(self.split_sphere, [self.position[0], self.position[1], 0])
        path = gh.IsoCurve(self.split_sphere, gh.ConstructPoint(uv[0], uv[1], 0))
        return path

    def generate_sun(self):
        sun_base = rs.SurfaceParameter(self.split_sphere, self.position)
        sun_origin = rs.EvaluateSurface(self.split_sphere, sun_base[0], sun_base[1])
        sun = Sphere(sun_origin, self.SUN_RADIUS).generate_sphere()
        return sun, sun_origin

class Tower(Rectangle):
    def __init__(self, origin, size, count, heights):
        Rectangle.__init__(self, origin, size)
        self.FLOOR_HEIGHT = 4.5
        self.count = count
        self.heights = heights
        self.rectangle = self.generate_rectangle()
        self.grid = self.generate_grid()
        self.tower = self.extrude_grid()

    def generate_grid(self): ...

    def extrude_grid(self):
        tower = []
        for i in range(len(self.grid)):
            if i == 4:
                self.heights[i] = max(self.heights)
            direction = rs.AddLine(rs.AddPoint(0,0,0), rs.AddPoint(0,0,self.heights[i]*self.FLOOR_HEIGHT))
            tower.append(rs.ExtrudeCurve(self.grid[i], direction)); rs.CapPlanarHoles(tower[i])
        return tower
    def generate_shadow(self, vector):
        mesh_setting = rg.MeshingParameters()
        tower_shadow = []
        for tower in self.tower:
            mesh_tower = rg.Mesh.CreateFromBrep(rs.coercebrep(tower), mesh_setting)
            tower_shadow.extend(gh.MeshShadow(mesh_tower, vector, rg.Plane.WorldXY))
        shadow = Region(tower_shadow).generate_union()
        return gh.BoundarySurfaces(shadow)

    def area_shadow(self, shadow):
        return gh.Area(shadow)[0]

    def calculate_gfa(self):
        return self.size * self.size * sum(self.heights)
```

```python
class Genetic:
    def __init__(self, gross_floor_area, shadow_park_intsc_area, shadow_mass_intsc_area):

    def evaluate_fitness(self, park_shadow_threshold, mass_shadow_threshold):
        expression = (self.gross_floor_area
                      * (self.shadow_park_intsc_area+1)
                      * (self.shadow_mass_intsc_area+1))

        fitness = -expression

        if self.shadow_park_intsc_area > park_shadow_threshold:
            fitness = expression

        if self.shadow_mass_intsc_area > mass_shadow_threshold:
            fitness = expression

        return fitness
```

Threshold

In the process of creating the fitness function, if a threshold is not set for the area of the shadow, it is optimized in the direction of increasing the total area only.

No Threshold ▼

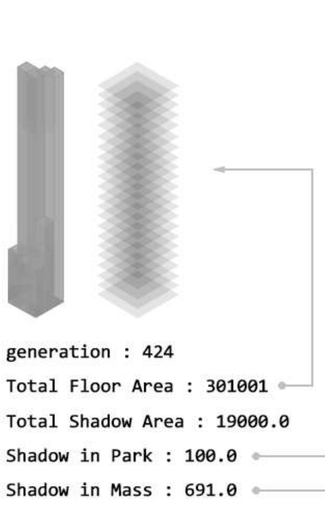
```
generation : 1
Total Floor Area : 201549
Total Shadow Area : 16513.0
Shadow in Park : 714.7
Shadow in Mass : 42.0
Fitness : 6196624005.00
Genome : [13, 86, 9, 23, 4, 97, 16, 11, 29]

generation : 25
Total Floor Area : 216361
Total Shadow Area : 2213.0
Shadow in Park : 428.7
Shadow in Mass : 901.0
Fitness : -14247804572.00
Genome : [48, 41, 8, 82, 108, 20, 2, 75, 25]

generation : 78
Total Floor Area : 228528
Total Shadow Area : 16903.0
Shadow in Park : 100.0
Shadow in Mass : 651.0
Fitness : -15049025856.00
Genome : [45, 41, 7, 95, 109, 21, 13, 76, 25]

generation : 157
Total Floor Area : 281957
Total Shadow Area : 18275.0
Shadow in Park : 100.0
Shadow in Mass : 651.0
Fitness : -18567432364.00
Genome : [108, 41, 10, 95, 108, 20, 14, 110, 25]
```

With Threshold ▼

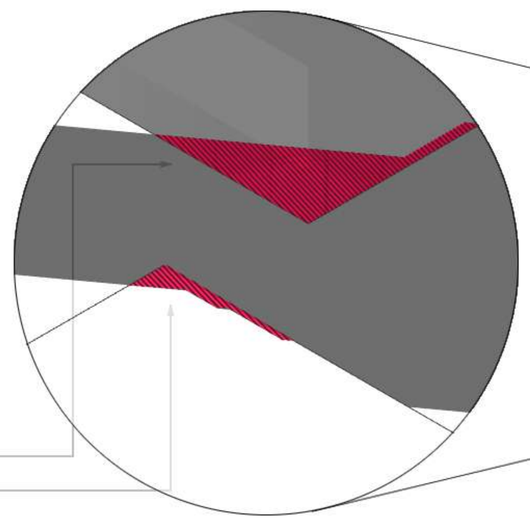
```
generation : 1
Total Floor Area : 201549
Total Shadow Area : 16513.0
Shadow in Park : 714.7
Shadow in Mass : 42.0
Fitness : 6196624005.00
Genome : [13, 86, 9, 23, 4, 97, 16, 11, 29]

generation : 25
Total Floor Area : 216361
Total Shadow Area : 2213.0
Shadow in Park : 428.7
Shadow in Mass : 901.0
Fitness : -14247804572.00
Genome : [48, 41, 8, 82, 108, 20, 2, 75, 25]

generation : 350
Total Floor Area : 300472
Total Shadow Area : 18988.0
Shadow in Park : 100.0
Shadow in Mass : 691.0
Fitness : -21000589024.00
Genome : [110, 41, 23, 108, 105, 23, 18, 110, 25]

generation : 424
Total Floor Area : 301001
Total Shadow Area : 19000.0
Shadow in Park : 100.0
Shadow in Mass : 691.0
Fitness : -21037561892.00
Genome : [110, 41, 23, 109, 108, 23, 18, 110, 25]
```

Result

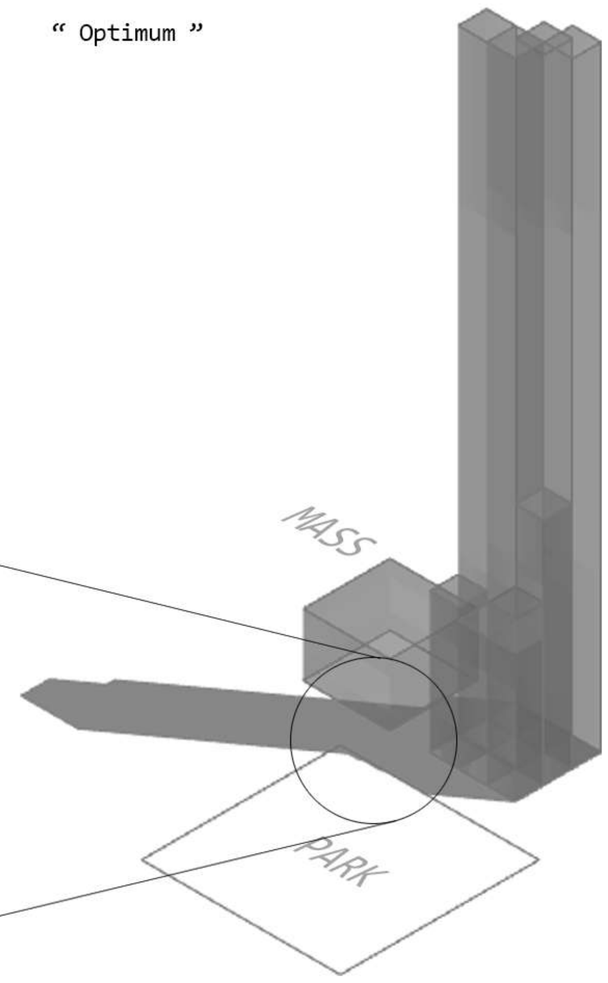
The optimal solution was found in the 424th generation and GALAPAGOS was terminated. The result can be seen through the image below.



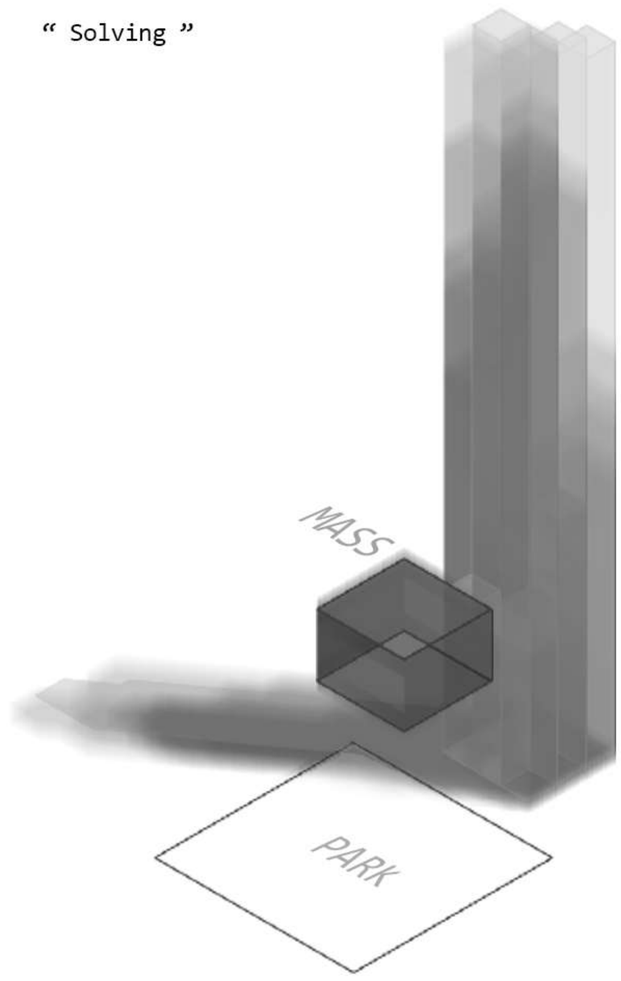
generation : 424
Total Floor Area : 301001
Total Shadow Area : 19000.0
Shadow in Park : 100.0
Shadow in Mass : 691.0



“ Optimum ”

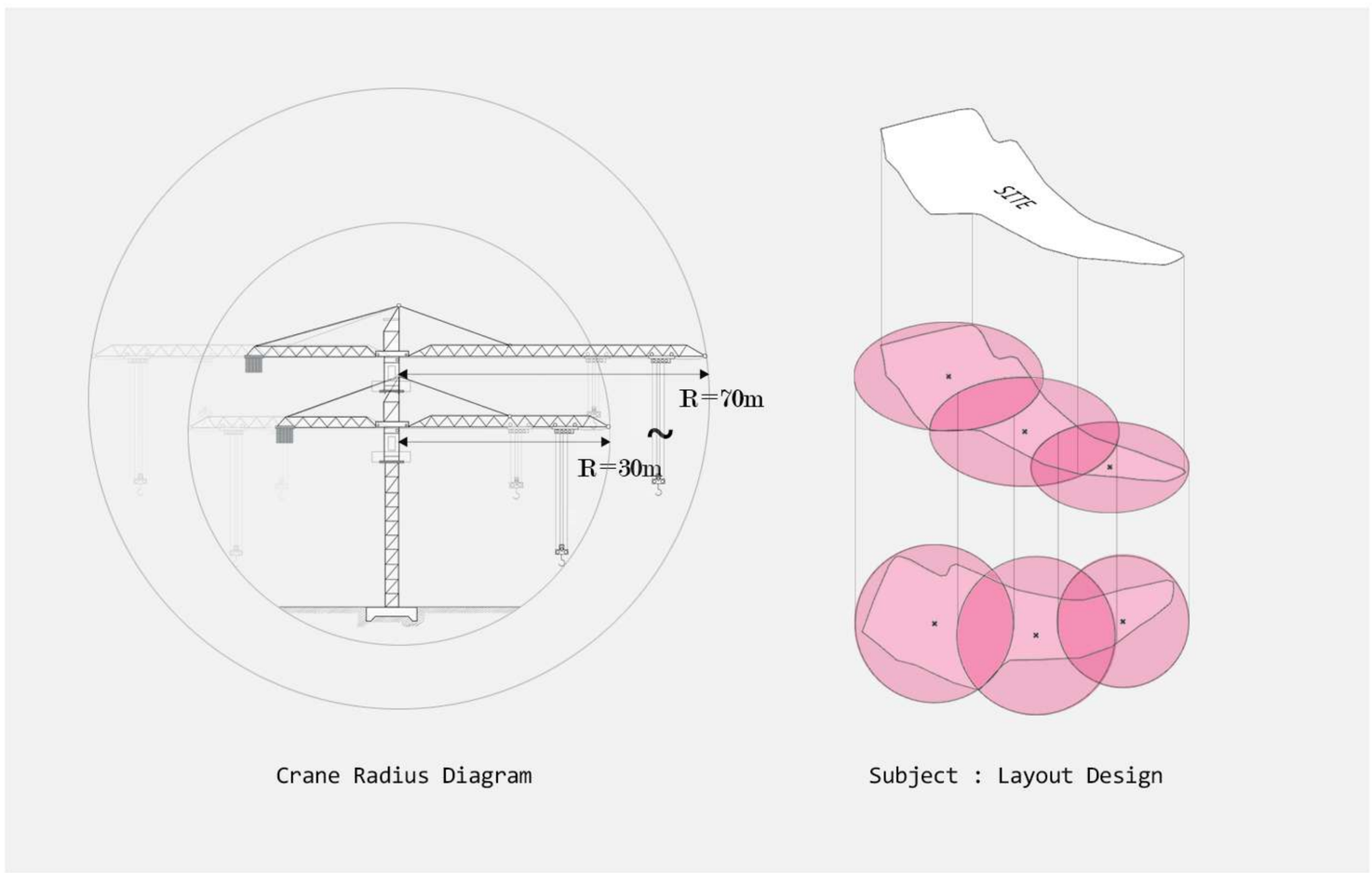


“ Solving ”



Tower Crane Arrangement

Tower Crane Layout Design Using GALAPAGOS



Crane Radius Diagram

Subject : Layout Design

“ Requirement ”

Requirement

This project is to optimize the layout of the tower crane given the required number of cranes.

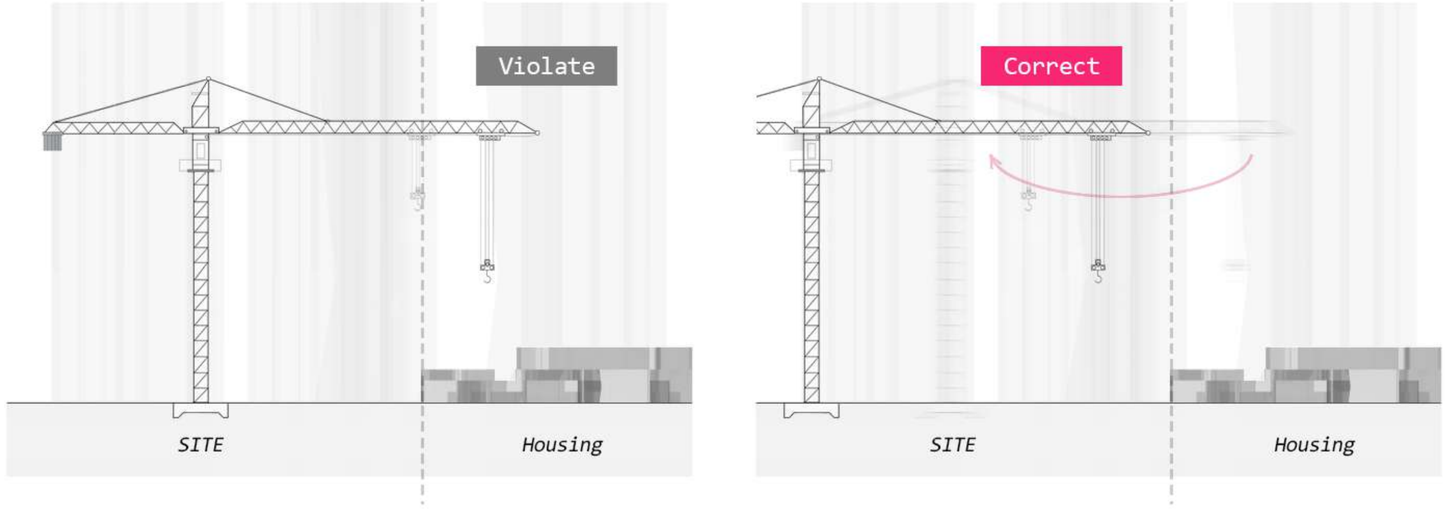

```
crane_radius = crane[i] * 10
```

The size of the crane is as shown in the diagram above, and when arranging three cranes, I thought about which size crane to choose and what criteria to place it.

Constraints

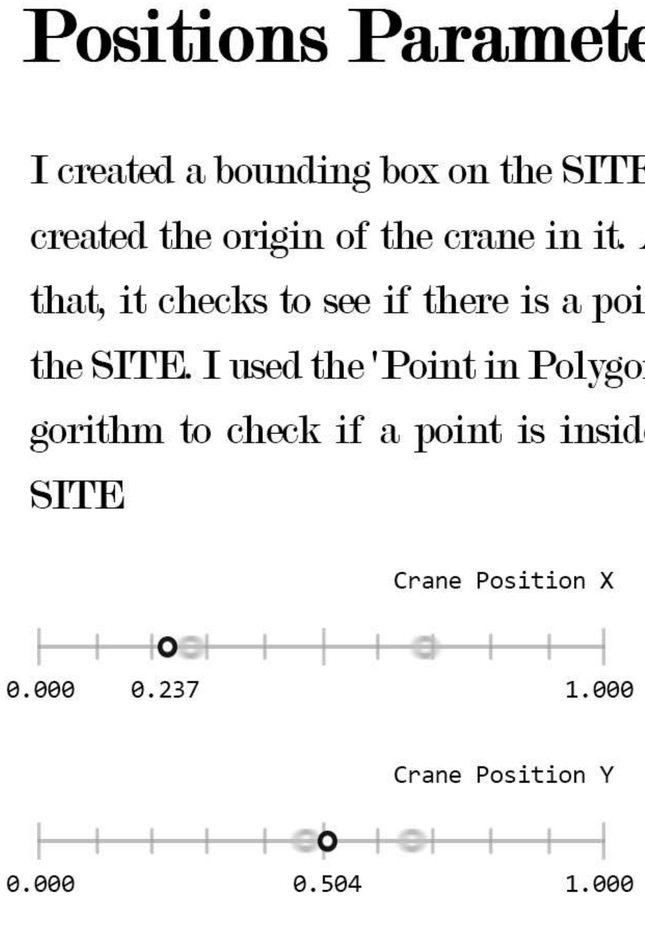
A constraint in this project is to minimize the intrusion of tower cranes into the housing areas around the site.

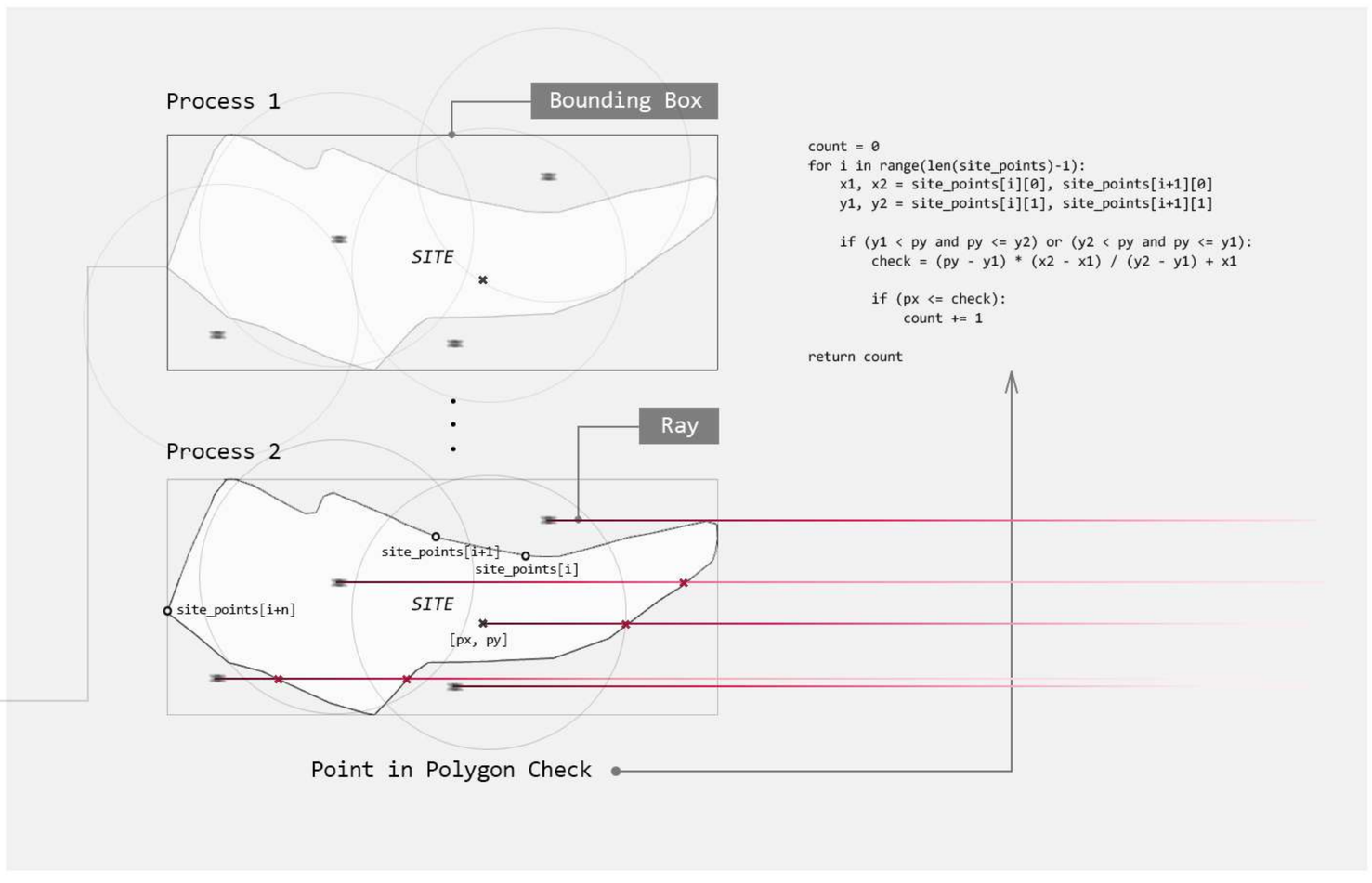
When a tower crane invades a housing area, the location where the tower crane is created is adjusted again.



Positions Parameter

I created a bounding box on the SITE and created the origin of the crane in it. After that, it checks to see if there is a point in the SITE. I used the 'Point in Polygon' algorithm to check if a point is inside the SITE





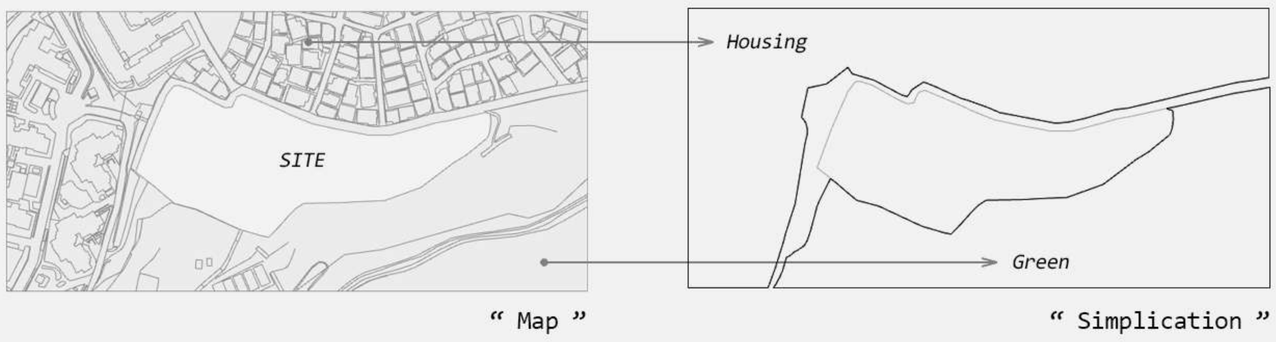

```
count = 0
for i in range(len(site_points)-1):
    x1, x2 = site_points[i][0], site_points[i+1][0]
    y1, y2 = site_points[i][1], site_points[i+1][1]

    if (y1 < py and py <= y2) or (y2 < py and py <= y1):
        check = (py - y1) * (x2 - x1) / (y2 - y1) + x1

        if (px <= check):
            count += 1

return count
```

“ Point Creation ”

Fitness

Fitness is set as the total area of the SITE minus the area occupied by the tower crane.

If you train this Fitness without any Threshold, the results will be calculated in the direction of increasing the size of the tower crane only. So, I thought that I had to set an additional threshold to the pre-set constraint, so I created an Evaluation Function.



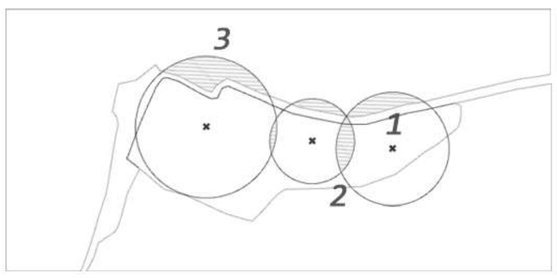
" Evaluate Fitness "

tower_crane_layout.py

```python
class Region:
    def __init__(self, crvs): ···

    def evaluate_type(self, crvs): ···

    def union_region(self):
        return gh.RegionUnion(self.curves)

    def intersection_region(self):
        return gh.RegionIntersection(self.curves[0], self.curves[1])

    def difference_region(self):
        return gh.RegionDifference(self.curves[0], self.curves[1])

    def origin_in_region(self):
        origin = rs.CurveAreaCentroid(self.curves[0])[0]
        site_points = rs.CurvePoints(self.curves[1])

        count = 0
        px, py, _ = origin
        for i in range(len(site_points)-1):
            x1, x2 = site_points[i][0], site_points[i+1][0]
            y1, y2 = site_points[i][1], site_points[i+1][1]
            if (y1 < py and py <= y2) or (y2 < py and py <= y1):
                check = (py - y1) * (x2 - x1) / (y2 - y1) + x1
                if (px <= check):
                    count += 1
        return count

    def area_region(self, region): ···


class Surface: ···


if __name__ == "__main__":
    site_object = Surface(site)
    site_surface = site_object.generate_surface()
    site_area = site_object.area_surface()

    crane_positions = [crane_position_1, crane_position_2, crane_position_3]
    crane_radius = [crane_radius_1, crane_radius_2, crane_radius_3]

    crane_origins = []
    crane_towers = []
    crane_site_intersecs = []

    for i, position in enumerate(crane_positions):
        crane_origin = site_object.evaluate_surface(position)
        crane_tower = Circle(crane_origin, crane_radius[i]).generate_circle()
        crane_site_intersec = Region([crane_tower, site]).intersection_region()

        crane_origins.append(crane_origin)
        crane_towers.append(crane_tower)
        crane_site_intersecs.append(crane_site_intersec)

    crane_site_union = Region(crane_site_intersecs).union
    crane_site_union_area = Region(crane_site_intersecs).union_area
    empty_site_area = site_area - crane_site_union_area

    fitness = empty_site_area

    exception_score = 500000
    housing_threshold = 200
    crane_threshold = 1000
    crane_housing_intersec_area =  Region([Region(crane_towers).union, housing_area]).intersec_area

    if crane_housing_intersec_area > housing_threshold:
        fitness = exception_score

    for crane in crane_towers:
        origin_check = Region([crane, site]).origin_in_region()
        if origin_check % 2 == 0:
            fitness = exception_score

    total_crane_crane_intersec_area = 0
    for crane_combi in combinations(crane_towers, 2):
        crane_crane_intersec_area = Region(crane_combi).intersec_area
        total_crane_crane_intersec_area = total_crane_crane_intersec_area + crane_crane_intersec_area

    if total_crane_crane_intersec_area >= crane_threshold = 1000:
        fitness = exception_score
```

Evaluate

More details can be found in the code on the left. I have set three thresholds. 1. Whether the point is inside the SITE, 2. How much is the intersected area of the tower crane, 3. How much the crane has invaded the housing area.




```
Threshold :

1. origin_in_region % 2 != 0

2. crane_intersected_area >= 1000

3. housing_intersected_area >= 200
```

Record

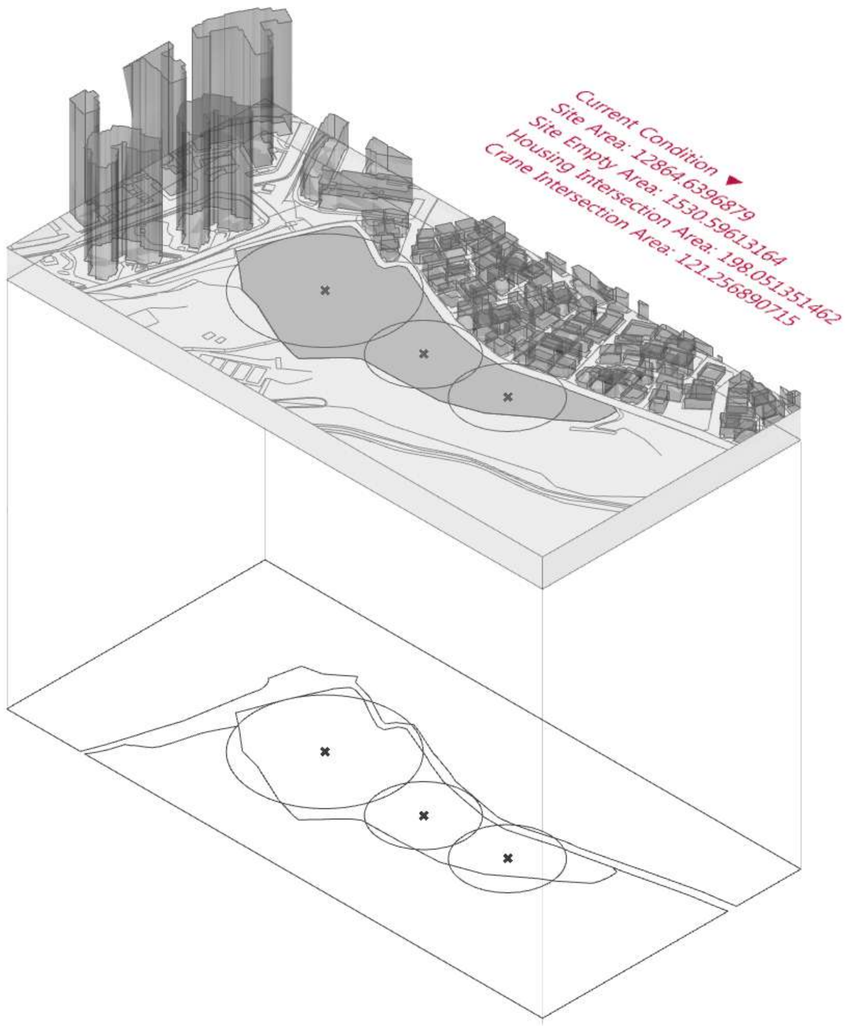
Each generation was recorded as follows. After 153rd generation, Fitness (Site Empty Area) is no longer lower.

```
generation : 1
Housing Intersection Area : 47.12
Crane Intersection Area : 81.04
Site Empty Area (Fitness) : 3434.98
Crane Size : [3, 4, 3]
Crane Position [[0.529, 0.514],
                [0.252, 0.480],
                [0.777, 0.448]]

generation : 27
Housing Intersection Area : 134.72
Crane Intersection Area : 320.59
Site Empty Area (Fitness) : 1639.10
Crane Size : [3, 5, 3]
Crane Position [[0.558, 0.483],
                [0.252, 0.480],
                [0.780, 0.493]]

generation : 41
Housing Intersection Area : 194.54
Crane Intersection Area : 220.78
Site Empty Area (Fitness) : 1566.51
Crane Size : [3, 5, 3]
Crane Position [[0.555, 0.481],
                [0.240, 0.504],
                [0.782, 0.497]]

generation : 74
Housing Intersection Area : 195.33
Crane Intersection Area : 155.33
Site Empty Area (Fitness) : 1537.88
Crane Size : [3, 5, 3]
Crane Position [[0.550, 0.473],
                [0.237, 0.506],
                [0.791, 0.503]]

generation : 105
Housing Intersection Area : 195.33
Crane Intersection Area : 155.61
Site Empty Area (Fitness) : 1537.88
Crane Size : [3, 5, 3]
Crane Position [[0.550, 0.473],
                [0.237, 0.506],
                [0.791, 0.503]]

generation : 153
Housing Intersection Area : 198.05
Crane Intersection Area : 121.25
Site Empty Area (Fitness) : 1530.60
Crane Size : [3, 5, 3]
Crane Position [[0.556, 0.469],
                [0.238, 0.511],
                [0.797, 0.502]]
```

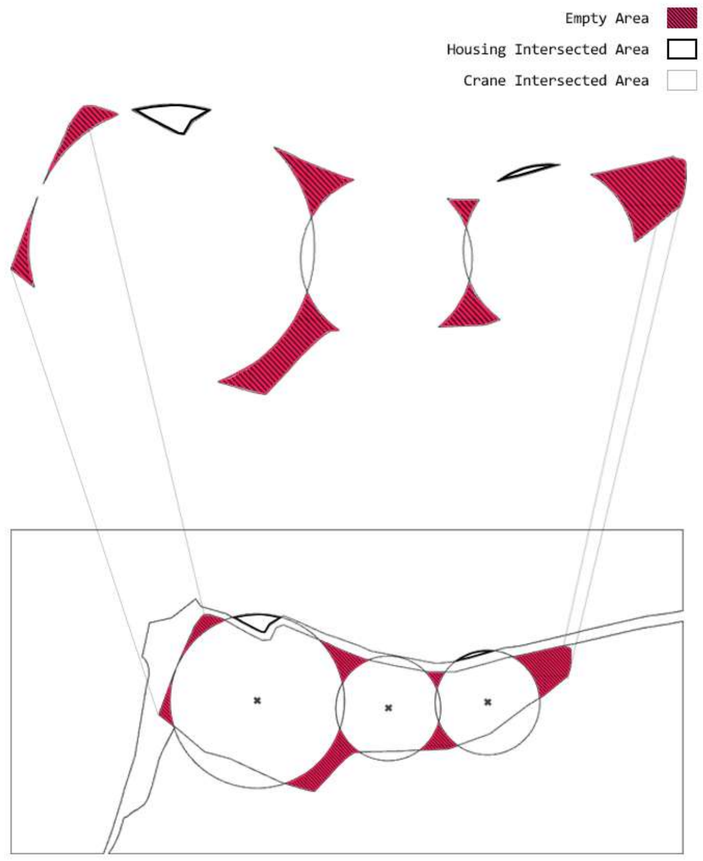
Result

The code I wrote and the optimized result within the set constraints are as follows. The area of the elements used in the fitness evaluation can be checked through the visualized images on the right.

generation : 153
Housing Intersection Area : 198.05
Crane Intersection Area : 121.25
Site Empty Area (Fitness) : 1530.60
Crane Size : [3, 5, 3]
Crane Position [[0.556, 0.469],
[0.238, 0.511],
[0.797, 0.502]]



Parallel View

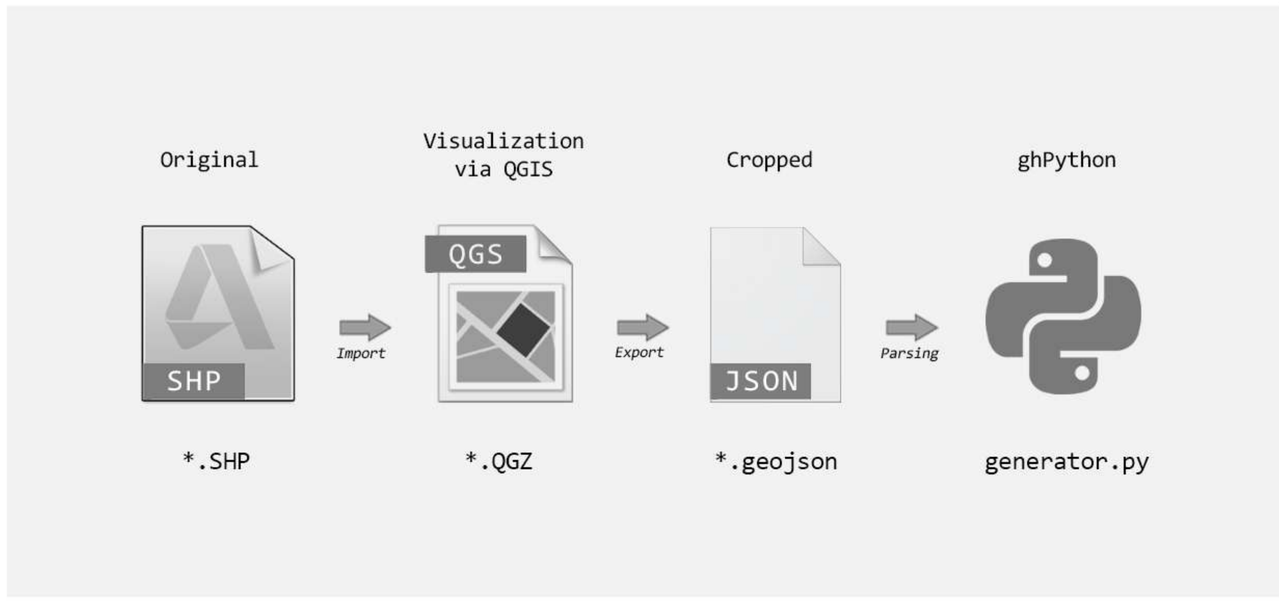


Top View

“ Rhino Viewport ”

Mass Generator via GeoJSON

Parsing GeoJSON File Format & Generating Geometry



“ Get GeoJSON ”

Get GeoJSON

After reading the shapefile from QGIS, cut out the necessary parts and convert it to the GeoJSON file format.

Original

Cropped

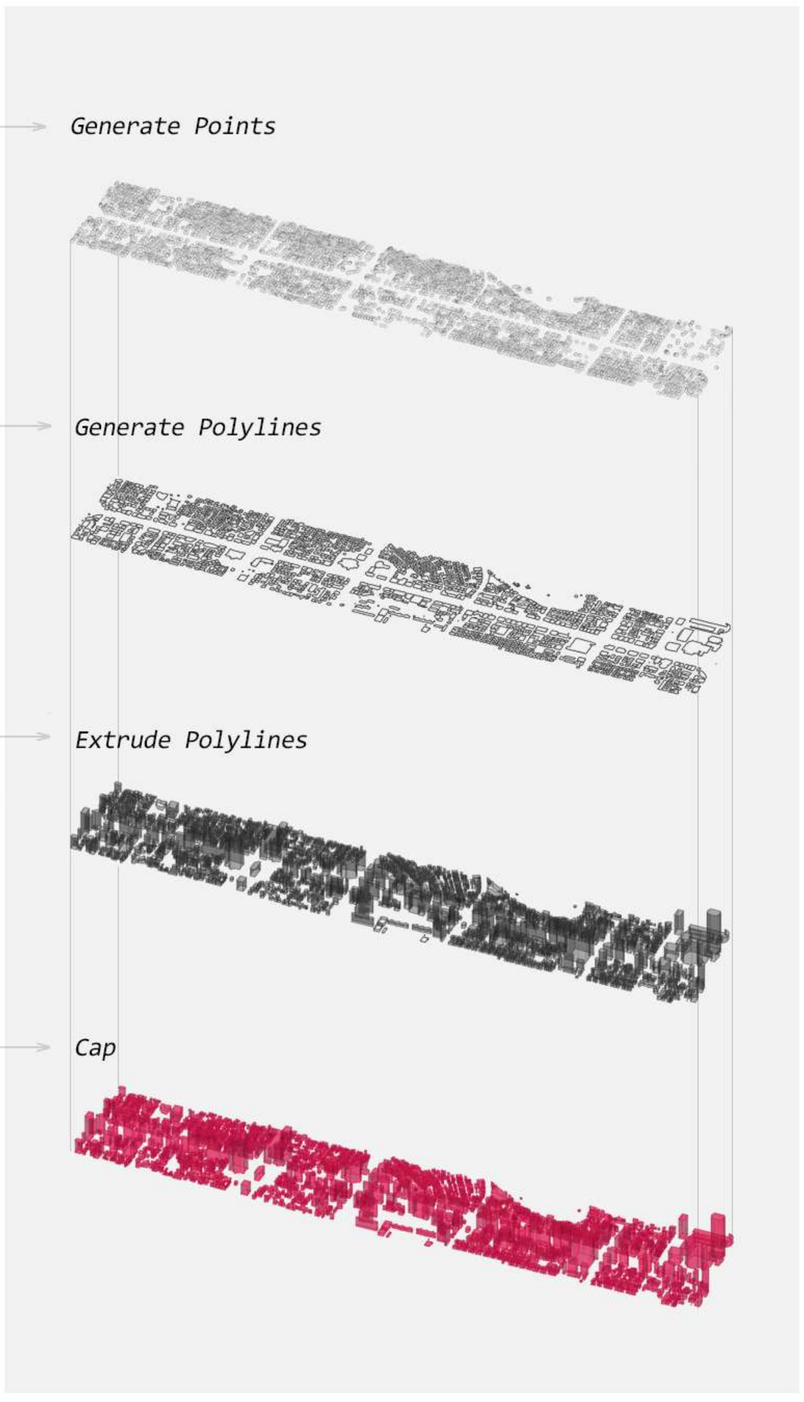
mass_generator.py

```python
class Data:
    def __init__(self, file_path):...

    def read_data(self):...

    def read_columns(self):...

    def read_geometry(self):
        geometry_data = []
        for data in self.data_features:
            geometries = data['geometry']['coordinates'][0][0]
            geometry_data.append(geometries)
        return geometry_data

    def read_information(self):
        information_data = []
        for data in self.data_features:
            properties = data['properties']
            information_data.append(properties)
        return information_data

    def calculate_height(self):
        building_information = self.read_information()
        building_heights = []
        floor_height = 4
        for information in building_information:
            height_information = information['HEIGHT']
            floor_information = information['GRND_FLR']

            building_height = max(height_information, floor_information * floor_height)
            building_heights.append(building_height)
        return building_heights

    def preprocess_information(self, keyword):
        postprocess = []
        for data in self.read_information():
            target_data = data[keyword]
            if target_data is not None:
                postprocess.append(int(target_data))
            else:
                postprocess.append(0)
        return postprocess


class Generator(Data):
    def __init__(self, path):
        Data.__init__(self, path)

    def generate_polylines(self):
        polylines = []
        coordinates = self.read_geometry()
        for coords in coordinates:
            points = []

            for c in coords:
                x, y, z = c[0], c[1], 0
                point = rs.AddPoint(x, y, z)
                points.append(point)

            polyline = rs.AddPolyline(points)
            polylines.append(polyline)
        return polylines

    def generate_centroids(self):
        polylines = self.generate_polylines()
        centroids = []
        for polyline in polylines:
            centroid = rs.CurveAreaCentroid(polyline)[0]
            centroids.append(centroid)
        return centroids

    def extrude_polylines(self):
        building_polylines = self.generate_polylines()
        building_heights = self.calculate_height()

        buildings = []
        for i, polyline in enumerate(building_polylines):
            z = building_heights[i]
            path = rs.AddLine(rs.AddPoint(0,0,0), rs.AddPoint(0,0,z))
            building = rs.ExtrudeCurve(polyline, path)

            if building is not None:
                rs.CapPlanarHoles(building)

            buildings.append(building)
        return buildings
```

Other Information Parsing Method

Generate Geometry

Read the geojson file and parse the geometry. The geojson file contains information about the coordinates of the geometry. Then, it parses the information about the height of the building or creates the geometry through the following process.



“ Process Diagram ”

mass_generator.py

```python
    def preprocess_information(self, keyword):
        postprocess = []
        for data in self.read_information():
            target_data = data[keyword]
            if target_data is not None:
                postprocess.append(int(target_data))
            else:
                postprocess.append(0)
        return postprocess


if __name__ == "__main__":
    data_object = Generator(file_path)
    buildings = data_object.extrude_polylines()
    centroids = data_object.generate_centroids()

    heights = data_object.calculate_height()
    min_height = min(heights)
    max_height = max(heights)

    regist_keyword = 'REGIST_DAY'
    regist_codes = data_object.preprocess_information(regist_keyword)
    min_regist_code = min(regist_codes)
    max_regist_code = max(regist_codes)

    structure_keyword = 'STRCT_CD'
    structure_codes = data_object.preprocess_information(structure_keyword)
    min_structure = min(structure_codes)
    max_structure = max(structure_codes)

    usability_keyword = 'USABILITY'
    usability_codes = data_object.preprocess_information(usability_keyword)
    min_usability = min(usability_codes)
    max_usability = max(usability_codes)
```

Visualization

After modeling each geometry contained in the geojson file, I proceeded with visualization with other information contained in the properties key.


```
{ "properties": { "fid": 1, "UFID": "2007203361814441481500000000", "BLD_NM": null,
"DONG_NM": null, "GRND_FLR": 2, "UGRND_FLR": 1, "PNU": "1168010100107250000", "ARCHAR-
EA": 660.15, "TOTALAREA": 1658.86, "PLATAREA": 1120.5, "HEIGHT": 11.7, "STRCT_CD":
"31", "USABILITY": "04000", "BC_RAT": 58.92, "VL_RAT": 89.13, "BLDRGST_PK": "13678",
"USEAPR_DAY": "20070919", "REGIST_DAY": "20170126", "GB_CD": "0", "VIOL_BD_YN": "0",
"GEOIDN": "B001000000012VJ72", "BLDG_PNU": null, "BLDG_PNU_Y": null, "BLD_UNLICE":
null, "BD_MGT_SN": "1168010100107250000000001", "SGG_OID": 88, "COL_ADM_SE": "11680" }
```


“ GeoJSON Properties ”

The visualized result can be seen in the image below. I visualized by height, regist date, structure, and usability.

227 0

“ Heights Gradient ”

20110207 20211015

“ Regist Day Gradient ”

NaN 11 19 21 31 32

33 39 41 42 51 99

“ Structure Codes Legend ”

NaN 1000 2000 3000 4000 5000 6000 7000

9000 10000 11000 14000 15000 16000 19000 20000

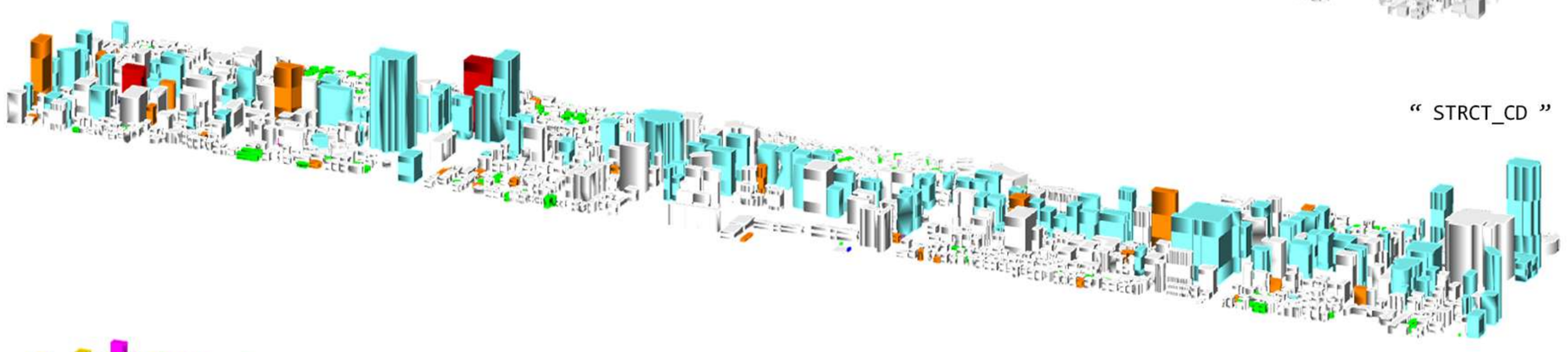
“ Usability Codes Legend ”

Rhino Viewport

“ HEIGHT & GRND_FLR ”

“ REGIST_DAY ”

“ STRCT_CD ”



“ USABILITY ”

Space Syntax

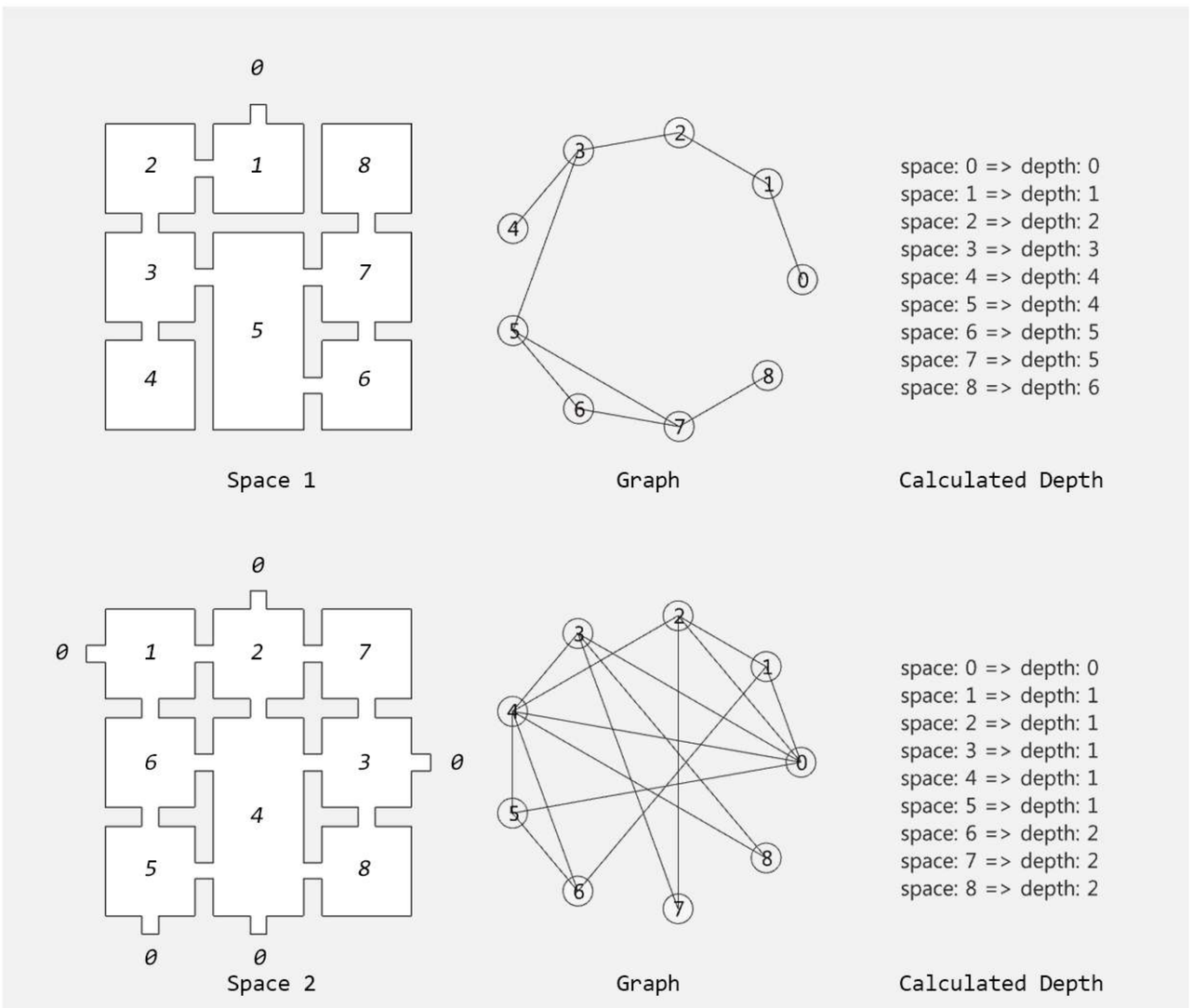
Algorithms for Quantitative Evaluation of Spatial Accessibility

Depth of Space

The depth of the space can be a criterion for evaluating the accessibility of the space. Let's take a look at the image below.

“ Same Space Different Evaluation ”

There are two spaces, and the way they are all laid out is the same. However, you can see that the accessibility to the space changes depending on the way the spaces are connected, that is, the depth.



“ Depth of Space Evaluation ”

```python
class Graph:
    def __init__(self, nodes, origin):
        self.nodes = nodes
        self.origin = origin

    def get_length(self):
        return len(self.nodes)

    def calculate_depth(self):
        start_node = 0
        graph_length = len(self.nodes)
        level = [None] * graph_length
        marked = [False] * graph_length

        level[start_node] = 0
        marked[start_node] = True
        queue = [start_node]

        while len(queue) != 0:
            curr_node = queue.pop(0)
            curr_conn = self.nodes[curr_node].get_connection()

            for i in range(len(curr_conn)):
                b = curr_conn[i]

                if marked[b] == False:
                    queue.append(b)
                    level[b] = level[curr_node] + 1
                    marked[b] = True

        level_text = ""
        for i, depth in enumerate(level):
            level_text = level_text + "space: {} => depth: {}\n".format(i, depth)

        return level_text

    def generate_polygon(self):
        size = 20
        segments = self.get_length()
        if segments < 3:
            segments = 3
        fillet = 0
        return gh.Polygon(rs.coerce3dpoint(self.origin), size, segments, fillet)[0]

    def visualization_nodes(self):
        nodes_points = gh.DeconstructBrep(self.generate_polygon())[2]
        if self.get_length() == 2:
            nodes_points = nodes_points[:2]
        return nodes_points

    def visualization_circles(self):
        radius = 2
        nodes_points = self.visualization_nodes()
        return gh.Circle(nodes_points, radius)

    def visualization_connections(self):
        nodes_points = self.visualization_nodes()

        lines = []
        for i, node in enumerate(self.nodes):
            for c in node.get_connection():
                line = rs.AddLine(nodes_points[i], nodes_points[c])
                lines.append(line)
        return lines
```

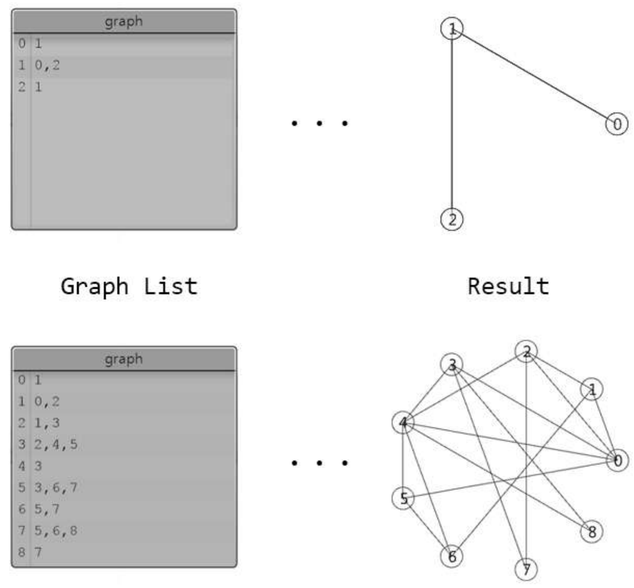
graph_node.py

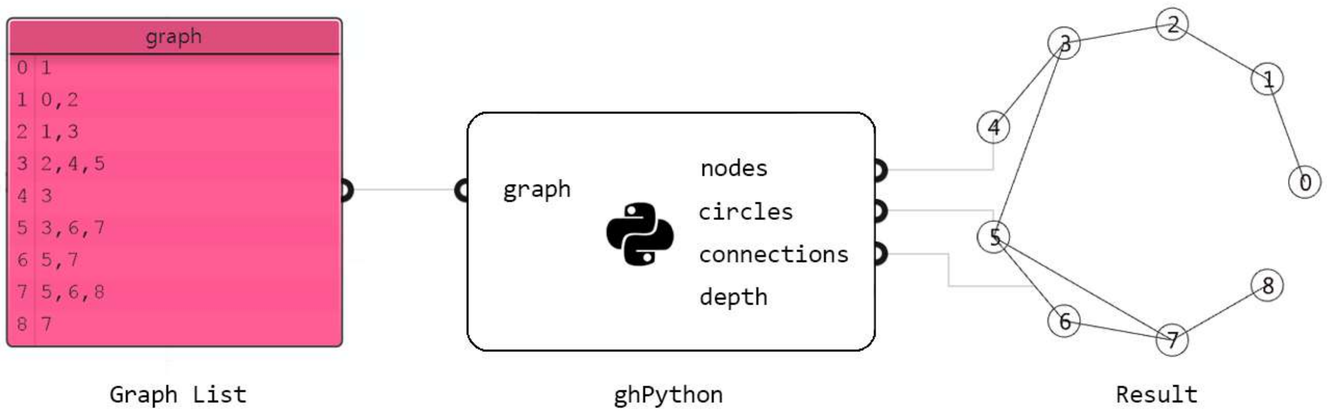
```python
if __name__ == "__main__":
    convert_data = Data(datas).preprocessing()
    node_list = []
    for i, conn in enumerate(convert_data):
        curr_node = Node(i)
        curr_node.connect_node(conn)
        node_list.append(curr_node)

    graph = Graph(node_list, origin)
    nodes = graph.visualization_nodes()
    circles = graph.visualization_circles()
    connections = graph.visualization_connections()
    depth = graph.calculate_depth()

```

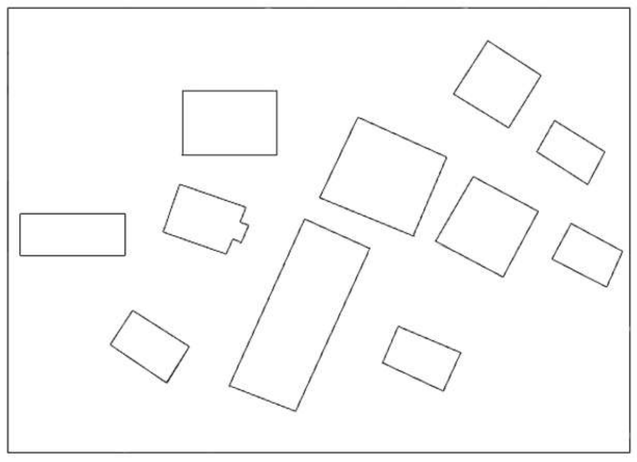
Graph

This code visualizes how each node is connected and how deep the node (space) is given its connected number.

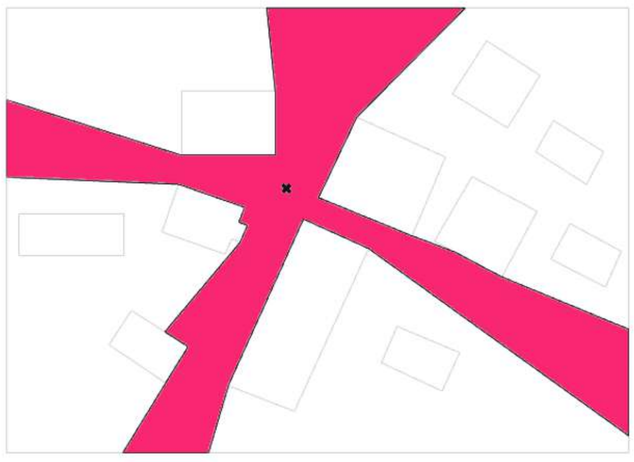
Here, the index of the vertex of the polygon becomes the number of the node.







“ Obstacles ”



“ Visibility Polygon ”

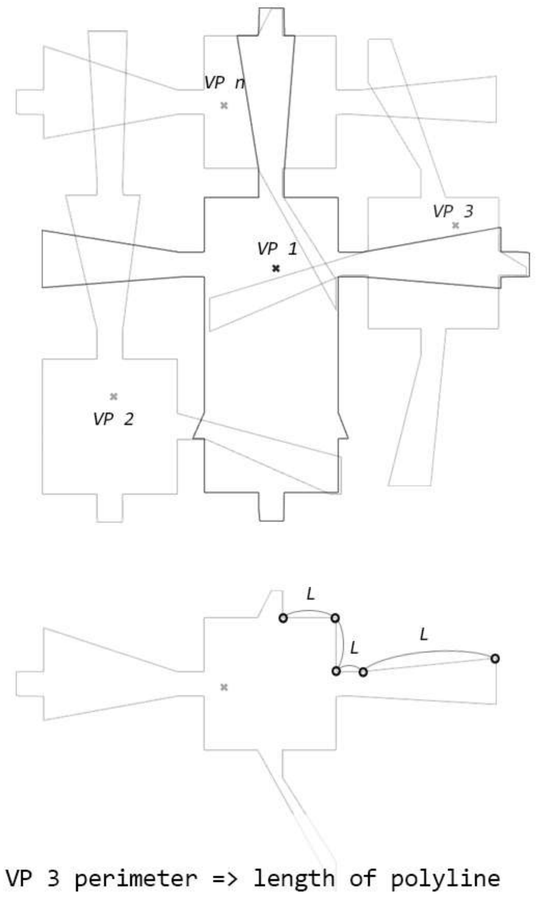
Visibility Polygon

Visibility polygons are the criterion for evaluating the visibility at a specific point. This can be used to quantitatively evaluate spatial activity or connectivity. Let's evaluate the previous two spaces through the following process.

Process

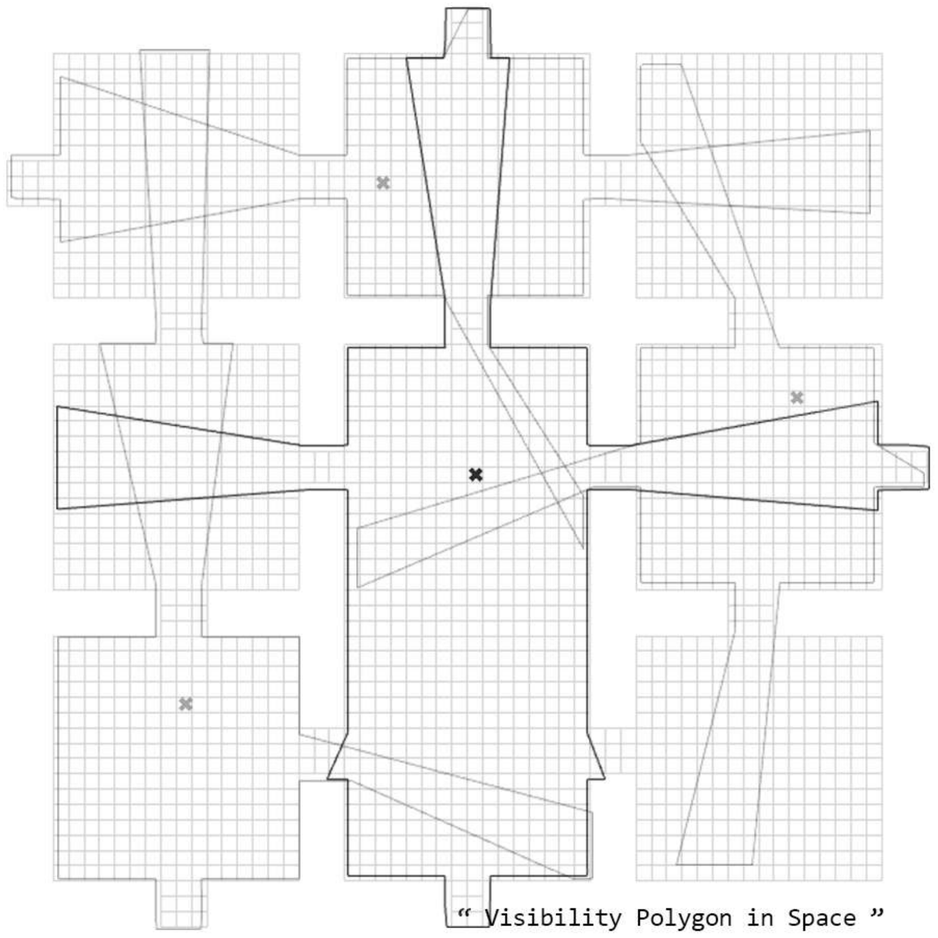
First we need to create a bounding box to wrap the all space. And then write a square grid in the bounding box.

Creates the center point of all grids, and checks whether the point is located inside the space or outside. Create a visibility polygon through Point in Space. And measure the perimeter of the polygon. You can check the process through the code below.



VP 3 perimeter => length of polyline

“ Calculate VP Perimeter ”



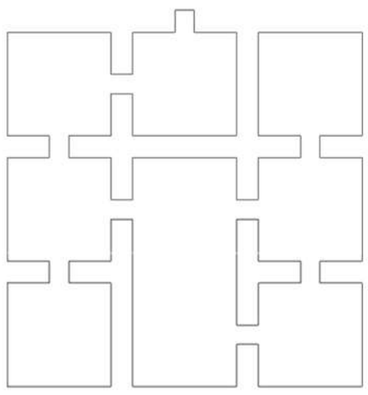
“ Visibility Polygon in Space ”

space_syntax.py

```python
class Space:
    def __init__(self, curves, resolution): ···

    def get_coordinate(self):
        points = []
        for curve in self.curves:
            points.extend(rs.CurvePoints(curve))

        min_coords = []
        max_coords = []
        for i in range(2):
            min_coord = min([coord[i] for coord in list(set(points))])
            max_coord = max([coord[i] for coord in list(set(points))])
            min_coords.append(min_coord)
            max_coords.append(max_coord)

        min_x, min_y = min_coords
        max_x, max_y = max_coords
        coordinates = [rg.Point3d(min_x, min_y, 0),
                       rg.Point3d(max_x, min_y, 0),
                       rg.Point3d(min_x, max_y, 0),
                       rg.Point3d(max_x, max_y, 0)]
        return coordinates

    def generate_brep(self): ···

    def generate_boundingbox(self): ···

    def generate_grid(self):
        bbox_width = self.coordinate[0].DistanceTo(self.coordinate[1])
        bbox_height = self.coordinate[0].DistanceTo(self.coordinate[2])
        unit_width = bbox_width / self.resolution
        unit_height= bbox_height / self.resolution
        start_point = self.coordinate[0]

        grid = []
        for i in range(resolution):
            for j in range(resolution):
                origin = [start_point[0] + i*unit_width, start_point[1] + j*unit_height, start_point[2]]
                unit = rs.AddRectangle(origin, unit_width, unit_height)
                grid.append(unit)
        return grid

    def generate_centroid(self):
        grid = self.generate_grid()
        centroids = [rs.CurveAreaCentroid(unit)[0] for unit in grid]
        return centroids

    def generate_inside_grid(self):
        brep = self.generate_brep()
        centroids = self.generate_centroid()
        inside_check = gh.PointInBrep(brep, centroids, False)
        inside_grid = []
        inside_centroid = []
        for i, check in enumerate(inside_check):
            if check == True:
                inside_grid.append(self.grid[i])
                inside_centroid.append(centroids[i])
        return inside_grid, inside_centroid

    def generate_vispolygon(self):
        inside_centroid = self.generate_inside_grid()[1]
        point_count = 100
        radius = 100
        vispolygons = []
        for centroid in inside_centroid:
            isovist = gh.IsoVist(centroid, point_count, radius, self.curves)[0]
            vispolygons.append(gh.PolyLine(isovist, True))
        return vispolygons

    def perimeter_vispolygon(self): ···
```

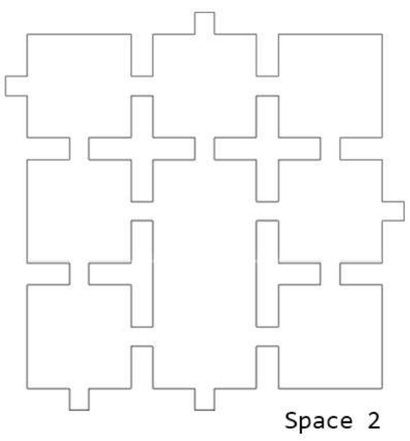
Evaluate

If you put the perimeter of the polygon obtained earlier in the list and match it with the index of the grid to brightly visualize the wide perimeter, the image below is created.

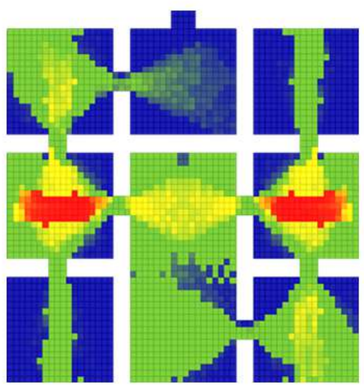


Space 1

VS

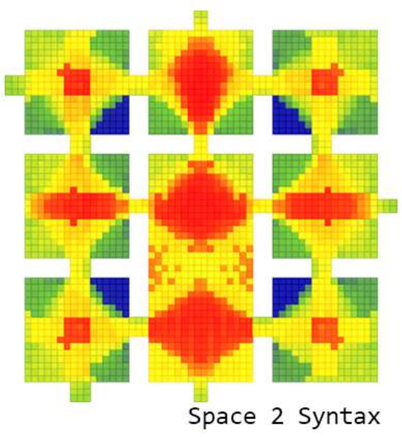


Space 2



Space 1 Syntax

VS



Space 2 Syntax

Geometric Algorithms

The Geometric Algorithms page is a small projects I've been working on to understand how geometry is created.

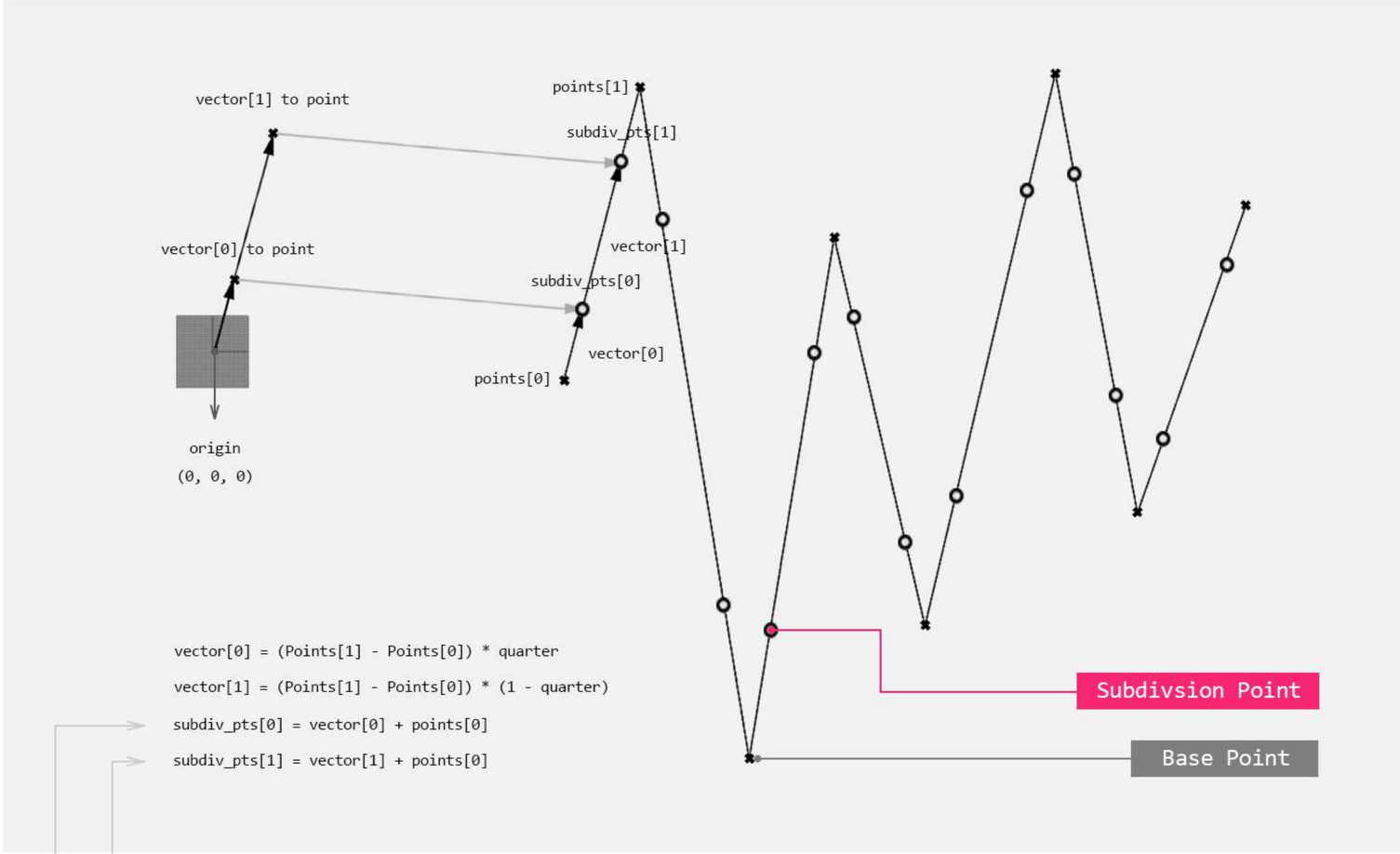
Subdivision Curve

Polyline to Subdivision Curve

Conception

Subdivision Curve is created based on the Base Point as shown in the figure on the right. Given the Base Points, find the vectors of the 25% and 75% points between the i-th point and the i+1-th point.

The generated vector can be checked through the figure on the right. When the Anchor Point of the created vector is moved from the origin to the i-th point, a Subdivision Point is created.



“ Subdivision Curve ”

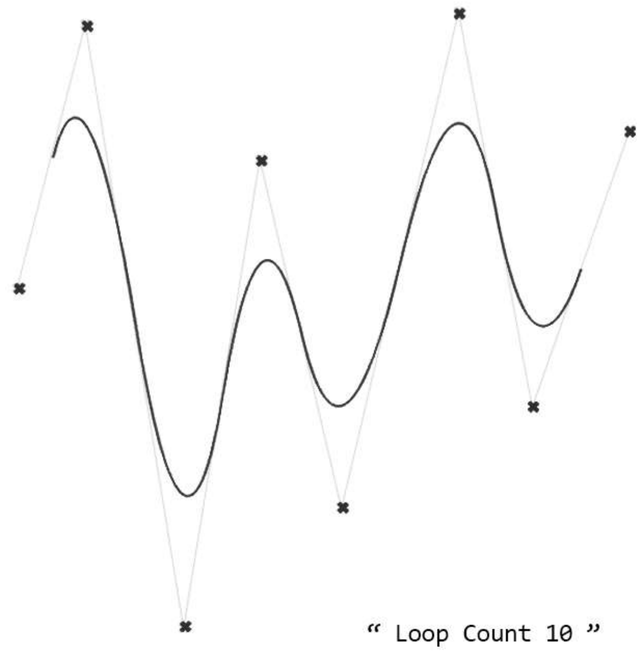
subdivsion_curve.py

```python
import Rhino.Geometry as rg
import rhinoscriptsyntax as rs

quarter = 0.25
for j in range(count):
    pts = []
    for i in range(len(points)-1):
        x1, y1, z1 = ((points[i+1] - points[i]) * quarter) + points[i]
        pt_1 = rg.Point3d(x1, y1, z1)
        pts.append(pt_1)

        x2, y2, z2 = ((points[i+1] - points[i]) * (1 - quarter)) + points[i]
        pt_2 = rg.Point3d(x2, y2, z2)
        pts.append(pt_2)
    points = pts

subdiv_points = points
subdiv_curve = rs.AddPolyline(points)
```

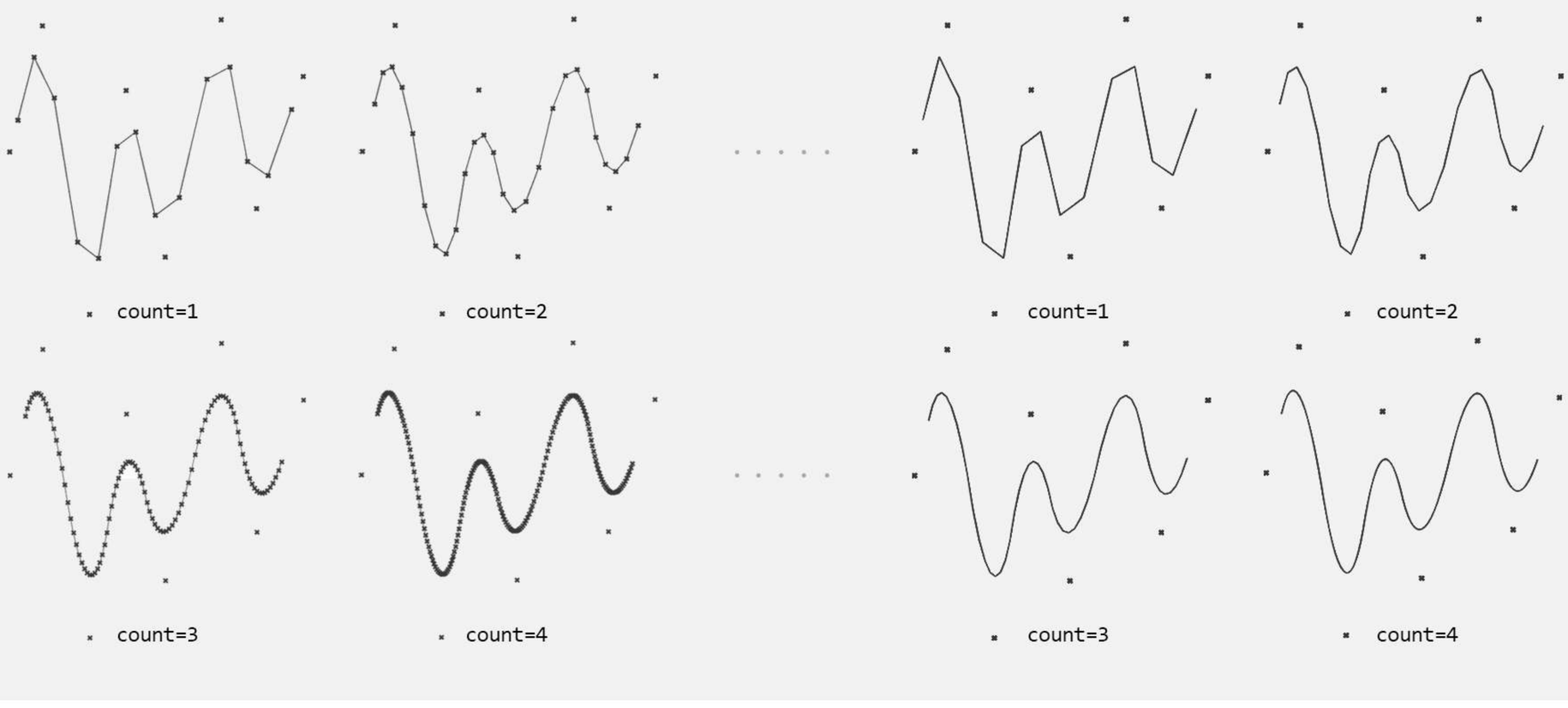
Count

As you increase the number of iterations, you get closer to the curve.

By adjusting the variable called 'quarter', you can also change the creation position of the Subdivision Point.



“ Loop Count 10 ”



“ Change by Count ”

Two Points Matrix

3D Points Grid Generator

two_points_matrix.py

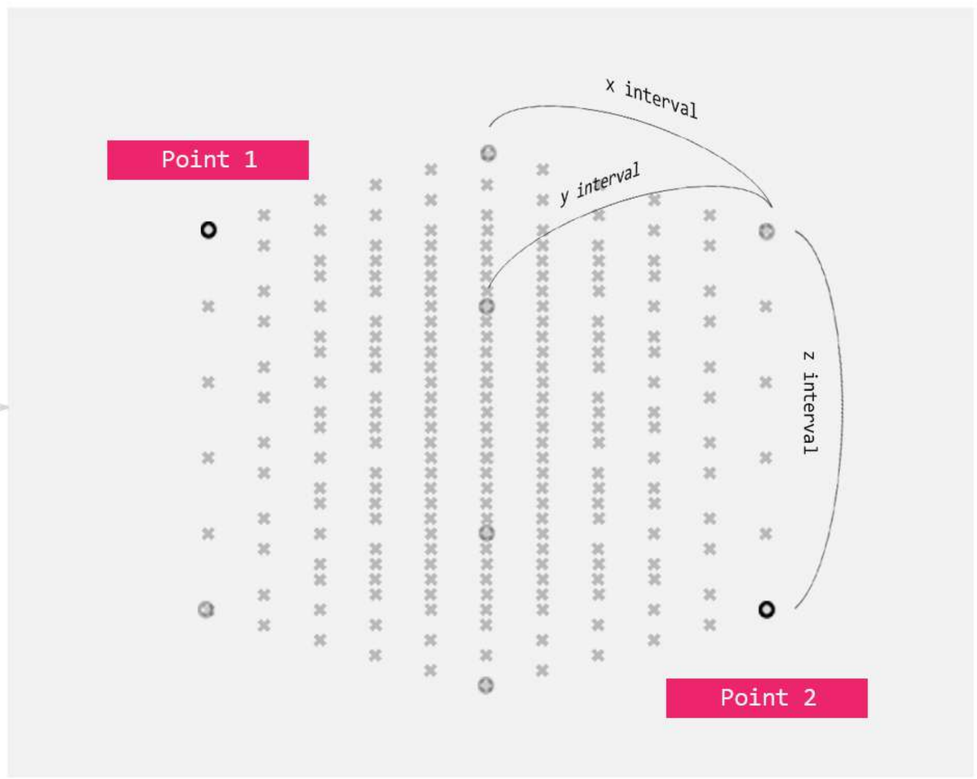
```python
class Point:
    def __init__(self, x, y, z):
        self._x = x
        self._y = y
        self._z = z

    def coordinate(self):
        coord_dict = {'x':self._x, 'y':self._y, 'z':self._z}
        return coord_dict

    def generate(self):
        x = self.coordinate()['x']
        y = self.coordinate()['y']
        z = self.coordinate()['z']
        pt = rs.AddPoint(x, y, z)
        return pt


class Matrix(Point):
    def __init__(self, p1, p2, cnt):
        self.p1 = p1
        self.p2 = p2
        self.cnt = cnt
        self.pts = []

    def points(self):
        p1_coord = self.p1.coordinate()
        p2_coord = self.p2.coordinate()
        points_dict = {'p1':p1_coord, 'p2':p2_coord}
        return points_dict

    def distance(self):
        p1_x = self.points()['p1']['x']; p1_y = self.points()['p1']['y']
        p2_x = self.points()['p2']['x']; p2_y = self.points()['p2']['y']
        x_distance = abs(p1_x - p2_x)
        y_distance = abs(p1_y - p2_y)
        z_distance = abs(p1_z - p2_z)
        distance_dict = {'x':x_distance, 'y':y_distance, 'z':z_distance}
        return distance_dict

    def interval(self):
        x_interval = self.distance()['x'] / self.cnt
        y_interval = self.distance()['y'] / self.cnt
        z_interval = self.distance()['z'] / self.cnt
        interval_dict = {'x':x_interval, 'y':y_interval, 'z':z_interval}
        return interval_dict

    def grid(self):
        def grid_gen(pos_1, pos_2, pos_3):
            for i in range(self.cnt+1):
                for j in range(self.cnt+1):
                    for k in range(self.cnt+1):
                        x = i*self.interval()['x']+pos_1
                        y = j*self.interval()['y']+pos_2
                        z = k*self.interval()['z']+pos_3
                        pt = Point(x,y,z).generate()
                        self.pts.append(pt)

        p1_x = self.points()['p1']['x']; p1_y = self.points()['p1']['y']; p1_z = self.points()['p1']['z']
        p2_x = self.points()['p2']['x']; p2_y = self.points()['p2']['y']; p2_z = self.points()['p2']['z']

        if p1_x < p2_x:
            if p1_y < p2_y:
                if p1_z <= p2_z:
                    grid_gen(p1_x, p1_y, p1_z)

                elif p1_z > p2_z:
                    grid_gen(p1_x, p1_y, p2_z)

            elif p1_y > p2_y:
                if p1_z <= p2_z:
                    grid_gen(p1_x, p2_y, p1_z)

                elif p1_z > p2_z:
                    grid_gen(p1_x, p2_y, p2_z)

        if p1_x > p2_x:
            if p1_y > p2_y:
                if p1_z > p2_z:
                    grid_gen(p2_x, p2_y, p2_z)

                elif p1_z <= p2_z:
                    grid_gen(p2_x, p2_y, p1_z)

            elif p1_y < p2_y:
                if p1_z <= p2_z:
                    grid_gen(p2_x, p1_y, p1_z)

                elif p1_z > p2_z:
                    grid_gen(p2_x, p1_y, p2_z)

        return self.pts
```

Condition 1

Condition 2

Conception

Even if we only know two points, we can get information such as the minimum coordinates, maximum coordinates, and distance of two points. So I tried to create a Point Matrix with two points.

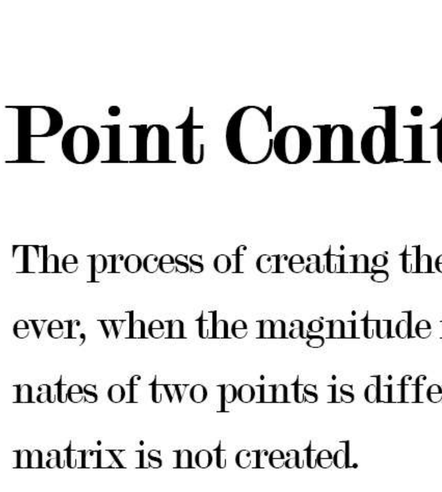


“ Two Points Matrix ”

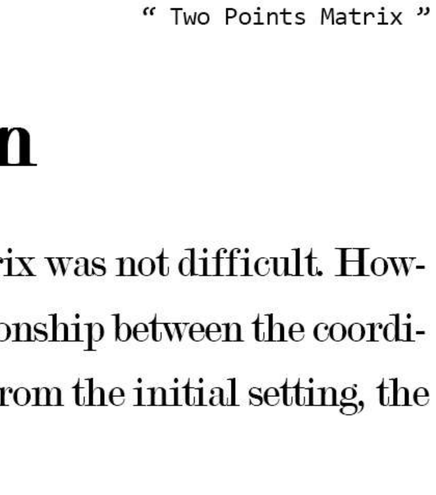
Point Condition

The process of creating the matrix was not difficult. However, when the magnitude relationship between the coordinates of two points is different from the initial setting, the matrix is not created.

So I added a number of conditions that can happen as in the code on the left. You can now move the points at will and the matrix will be created.



“ Condition 1 ”



“ Condition 2 ”

Extra Works

It is a small project that I worked on in PLAYADATA's data scientist training course. It contains the process from data collection to model learning.

Secure Room in Seoul

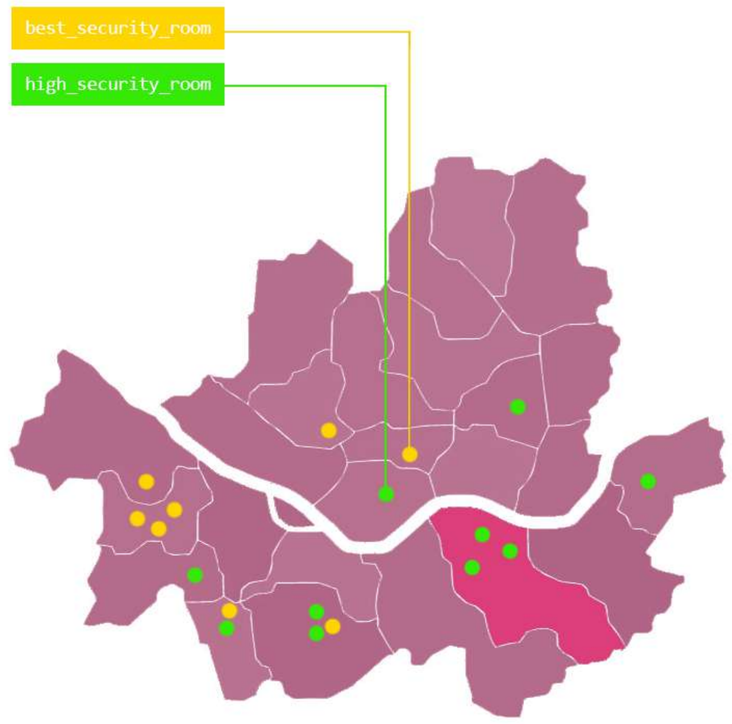
Recommend a Room with a High Security Score

	자치구	범죄발생합계 (14 ~ 19)	위도	경도
0	강동구	27292	37.549208	127.146482
1	양천구	23893	37.527062	126.856153
2	금천구	20931	37.460097	126.900155
3	동대문구	25217	37.583801	127.050700
4	은평구	26030	37.617612	126.922700
5	영등포구	37821	37.520641	126.913924
6	용산구	22540	37.531101	126.981074
7	도봉구	14584	37.665861	127.031767
8	광진구	30071	37.548144	127.085753
9	노원구	27354	37.655264	127.077120
10	구로구	30813	37.495486	126.858121

“ crime_total_coordinaes.csv ”

	보안점수	보안시설	위도	경도
0	19	현관보안 / CCTV / 인터폰 / 방범창	37.547893	126.938934992773
1	9	인터폰 / 카드키 / 방범창	37.533455	126.85559901145
2	17	현관보안 / CCTV / 인터폰 / 비디오폰	37.544646	126.87437592245
3	17	현관보안 / CCTV / 인터폰 / 비디오폰	37.536653	126.855945474322
4	10	현관보안 / 비디오폰	37.481145	126.812851806812
5	10	현관보안 / 비디오폰	37.484668	126.79309871833
6	22	현관보안 / CCTV / 인터폰 / 비디오폰 / 카드키 / 방범창	37.481785	127.03874973879
7	22	현관보안 / CCTV / 인터폰 / 비디오폰 / 카드키 / 방범창	37.509201	127.02707083768
8	22	현관보안 / CCTV / 인터폰 / 비디오폰 / 카드키 / 방범창	37.487307	126.98898870735
9	17	현관보안 / CCTV / 인터폰 / 비디오폰	37.485060	126.77737274378
10	22	현관보안 / CCTV / 인터폰 / 비디오폰 / 카드키 / 방범창	37.497369	127.00106796346

“ room_security_score.csv ”



“ Seoul Crime Choropleth Map ”

Overview

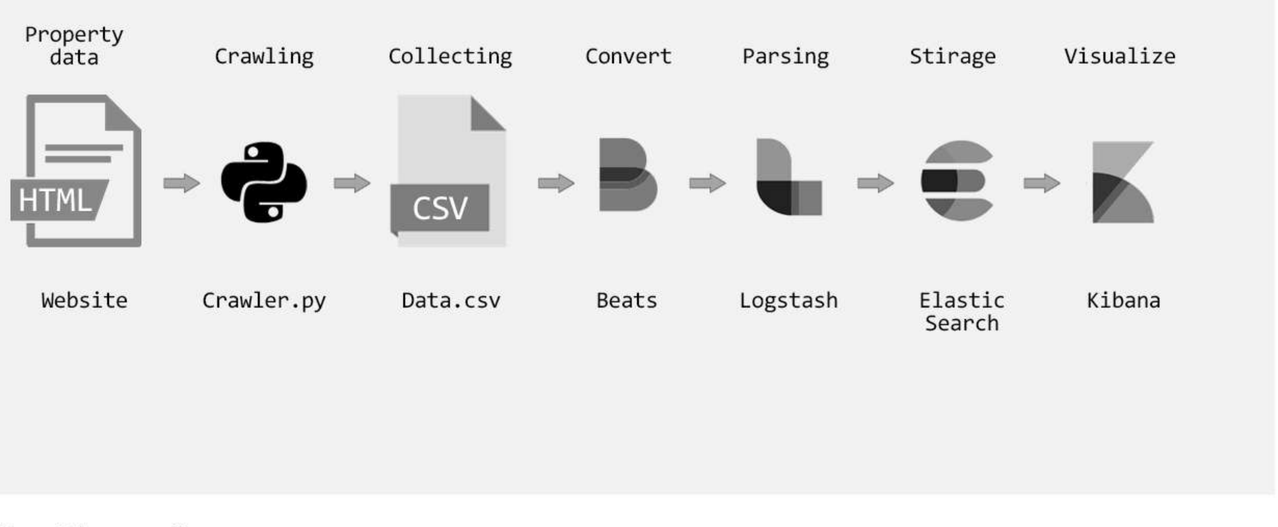
This project maps rooms with high security scores by overlapping the number of crimes and property data from 2014 to 2019.

The collected crime occurrence data and regional coordinate values of the Seoul OPEN API were mapped and visualized through the Kibana of the Elastic Stack.

Pipeline

I used the OPEN API on the "Save Peter Pan's Good Room" property sales site to crawl room information data in Seoul.

The crawled data is stored in the form of csv, inserted into the elastic search using the logstash and beats of the elastic stack, and visualized by the Kibana.



“ Pipeline Diagram ”

Idea

If you look at the HTML source of the "Save Peter Pan's Good Room" real estate sales site, there is a tag indicating the presence or absence of security facilities. I wanted to collect this and score the security score.

```
<div class="col-md-3 col-xs-4 column left-padding-20">
    보안시설
</div>
<div class="col-md-3 col-xs-8 column value">
    현관보안, CCTV, 인터폰, 비디오폰, 카드키
</div>
```

HTML

" Part of Property Site HTML "

Crawler.py

```
with open("seoul_room_list.txt", "r") as rt:
    room_info = [i.replace("\n", "") for i in rt.readlines()]
    columns = ["매물번호", "계약형태", "가격정보", "건물유형", "면적", "보안시설", "보안점수", "시구등", "위도", "경도", "링크"]
    peterpan = pd.DataFrame(columns=columns)
    loop_range = int(len(room_info))

    for i in range(11862, 12862):
        ROOM_URL = f"https://www.peterpanz.com/house/{room_info[i]}"
        req = requests.get(ROOM_URL)
        html = req.text
        soup = BeautifulSoup(html, "html.parser")

        # html parsing test ▼
        # with open("test.txt", "w", encoding="utf-8") as srl:
        #     srl.write(soup.prettify())

        try:
            r_contract = soup.select("div#contract_type")[0].string                                                                  # 계약형태
            r_price = soup.select("tr > td")[5].string.strip()                                                                       # 가격
            r_btype = soup.select("div.column.value")[12].string.strip()                                                             # 건물유형
            r_area = float(str(soup.select("div.column.value")[15]).split()[5].replace("m<sup>2</sup>", ""))                         # 면적
            r_security = soup.select(".commonHouse > .row.border-top > .col-md-3.col-xs-8.column.value")\
                                    [1].string.string.strip().replace(" ", "").replace("\n", " ").replace(",", " ").split()      # 보안시설
            r_sigudong = soup.select("div#sigudong")[0].string                                                                       # 시구등
            r_lat = float(str(soup.select("script")[44].string).split()[77].replace("'", "").replace(";", ""))                       # 위도
            r_long = float(str(soup.select("script")[44].string).split()[81].replace("'", "").replace(";", ""))                      # 경도

            btype = {"아파트": 3, "공동주택": 2, "단독주택": 1}
            rsecu = {"자체경비원": 7, "현관보안": 6, "CCTV": 5, "방범창": 4, "인터폰": 2, "비디오폰": 2, "카드키": 1, "-": 0}
            r_score = 0                                                                                                              # 보안점수

            for j in btype:
                if r_btype == j:
                    r_score = btype[j]

            for j in r_security:
                r_score += rsecu[j]

            r_security = " / ".join(r_security)

            info = {"매물번호" : room_info[i],
                    "계약형태" : r_contract,
                    "가격정보" : r_price,
                    "건물유형" : r_btype,
                    "면적"     : r_area,
                    "보안시설" : r_security,
                    "보안점수" : r_score,
                    "시구등"   : r_sigudong,
                    "위도"     : r_lat,
                    "경도"     : r_long,
                    "링크"     : ROOM_URL}

            print(f"[{loop_range}/{i}] : {room_info[i]}", info)
            # print(f"[{loop_range}/{i}] Crawling in progress")

            peterpan = peterpan.append(info, ignore_index=True)
            peterpan.to_csv("dataset/seoul_room.csv", index=False, encoding="utf-8", header=False)

        except:
            print(f"[{loop_range}/{i}] : {room_info[i]}", "crawling failed")
            # print(f"[{loop_range}/{i}] Crawling in progress")
```

" Crawling Code "

CSV

seoul_room.csv

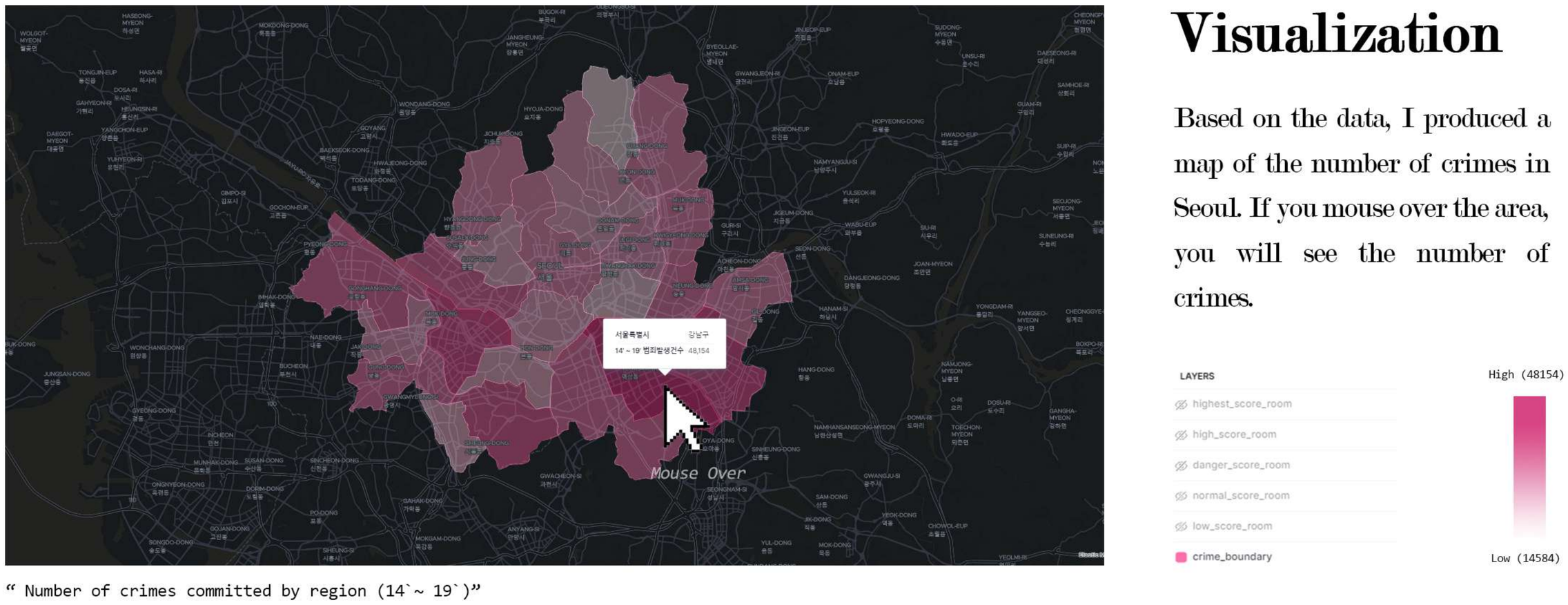
Data Collecting

Basic information such as contract type, price, and area within the property sales site was collected.

After that, tags containing security facility data were parsed and security scores were aggregated for each property.

Visualization

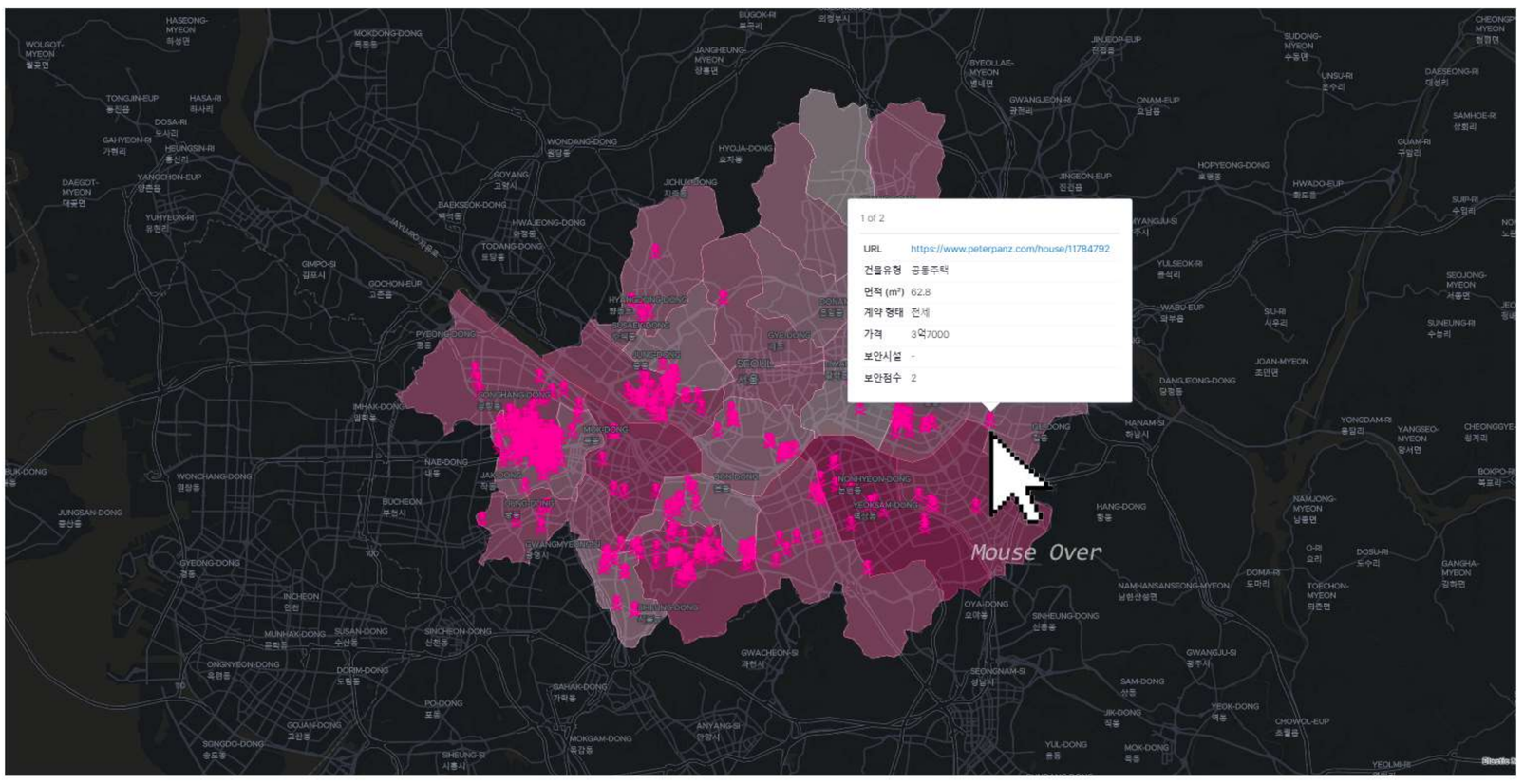
Based on the data, I produced a map of the number of crimes in Seoul. If you mouse over the area, you will see the number of crimes.



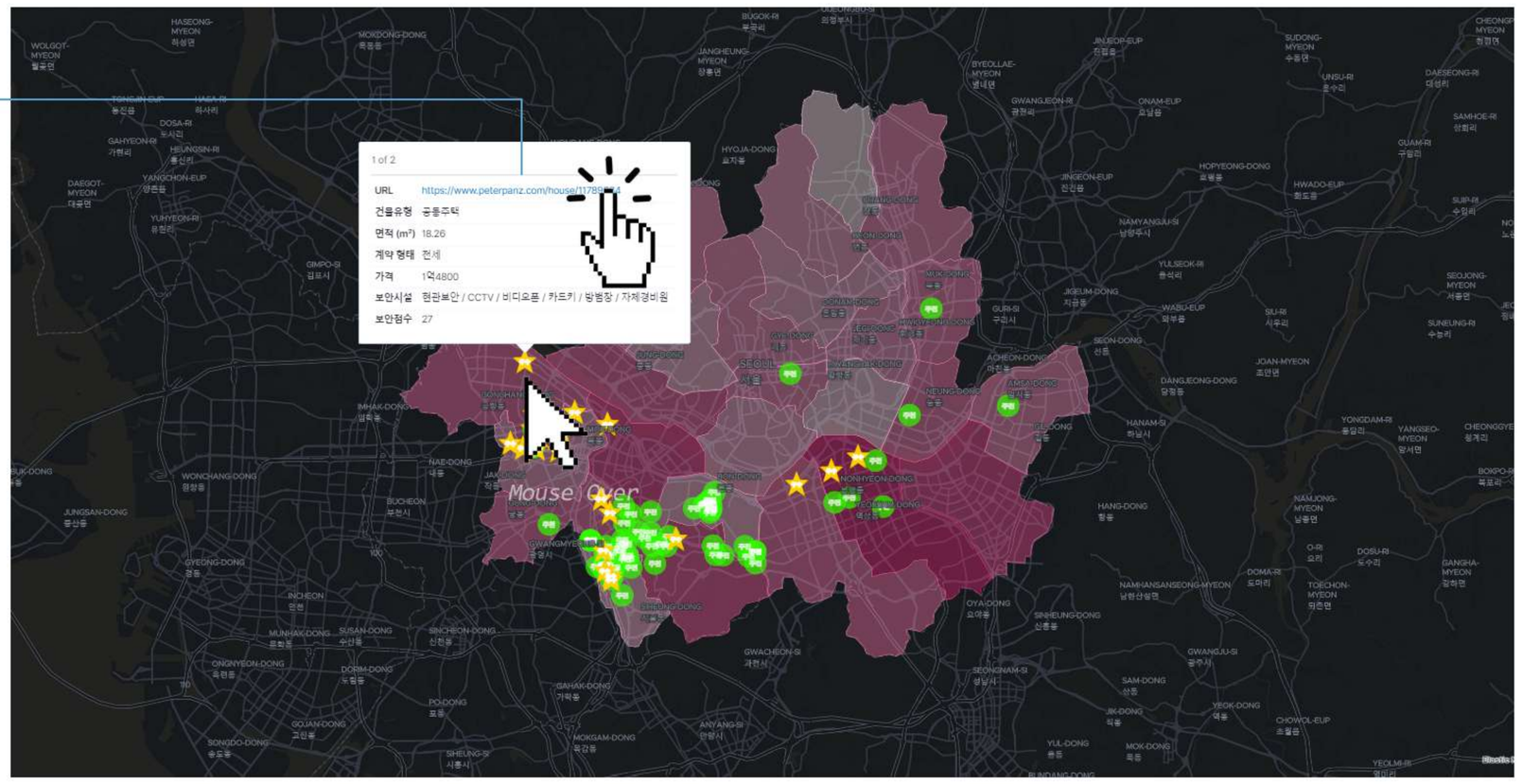
“ Number of crimes committed by region (14`~ 19`)”

Overlap

The rooms aggregated with security scores on the map above were overlapped so that users could filter rooms without security facilities.



“ Sales Site”



‘https://www.peterpanz.com/house/12190618’ => https://www.peterpanz.com/house/12190618
type(str) => type(url)

Linked

The site containing detailed information of property was switched from string type to url type so that users could move to the web page through the URL if they chose a room they liked.

Criminal's Face Recognition

Face Recognition through Deep Learning

Background

In the wanted poster, there is a person named Hwang Joo-hyun who has not disappeared for 12 years. Our team thought that the face data of criminals that had not been caught for a long time would have accumulated quite a bit.

So we thought we could design a model to identify criminals in CCTV.

Wanted　　Wanted's Face　　Wanted in CCTV

" Criminal's Facial Data "

" Fictional Wanted "

" Fictional Wanted in CCTV "

Scenario

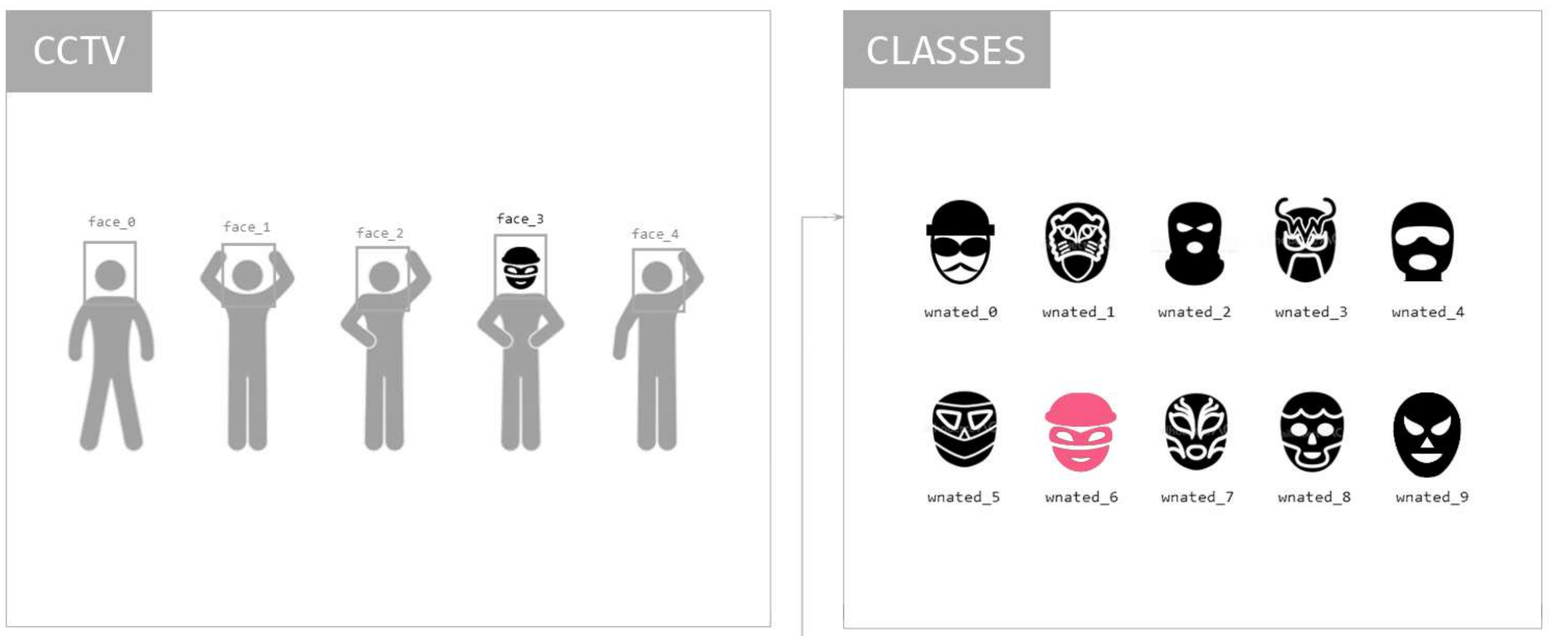
However, the criminal data could not be obtained due to the Personal Information Protection Act.

So we set celebrities who are easy to collect data as hypothetical criminals. The result we want is to recognition the criminal's face, like the picture on the left.

Process

We made the following plans: First, we collect the faces of the criminals. The collected face is trained on the model, and the probability of being a criminal is outputted with the CCTV image as input.

We used Sigmoid because we need to identify people who are not criminals.



" Face Detection "　　" Face Recognition "

“ Face Cropping ”

Data Preprocess

When preprocessing, we resize the image to a size that the model can learn. And to prevent coordinates from being generated outside the image during the cropping process, the code below is applied.

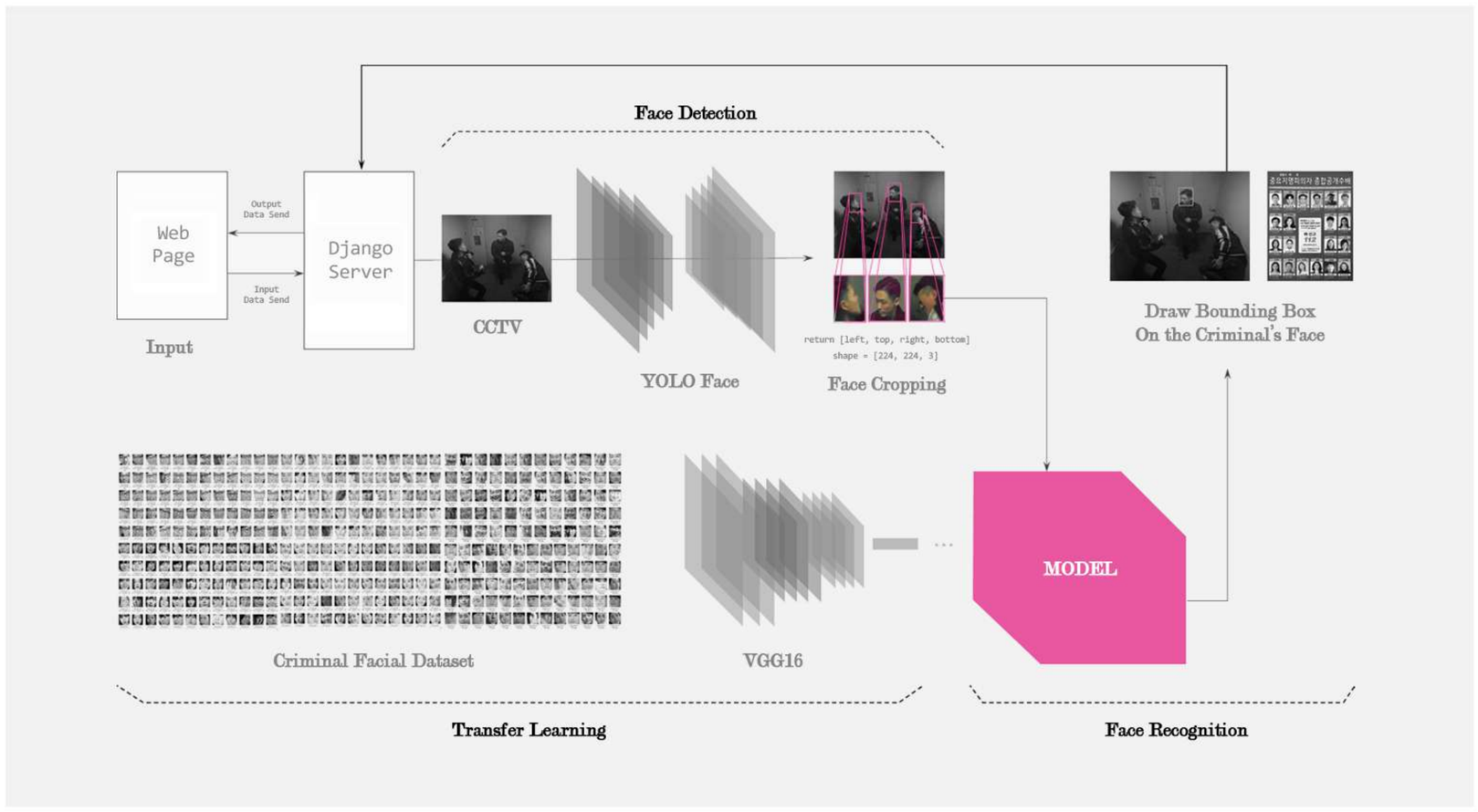
```
top = top - int(height / 4)
if top < 0:
    top = 0

bottom = top + height + int(height / 2)
if bottom >= frame_height:
    bottom = frame_height-1

left = left - int(width / 4)
if left < 0:
    left = 0

right = left + width + int(width / 2)
if right >= frame_width:
    right = frame_width-1
```

Coordinates Limit Code

Architecture

The structure of the project is as follows. Input image from web page. And it receives the data sent through the Django server. Face detection is performed on the input image and the image is pre-processed.

By inputting the pre-processed image into the pre-trained model, the probability of being a criminal and an image with a Bounding Box are returned.

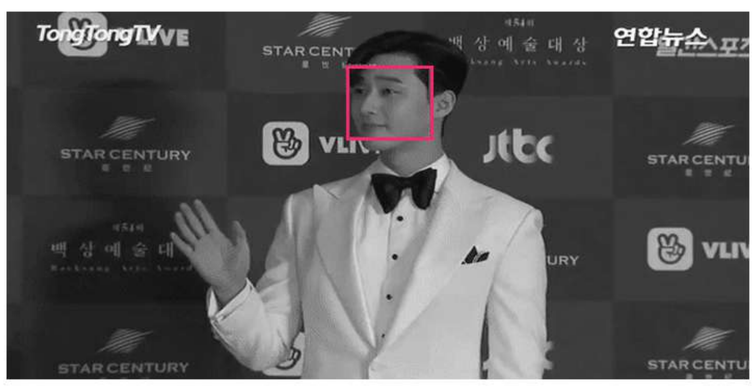


“ Architecture Diagram ”

Demo

The demo image is shown on the right. For more details, please check GitHub.

“ Web Page ”



“ Video ”

PARK
CHEOL
HEE

THANK YOU
FOR
READING