



KELOMPOK 9



Ong Ming Sen AI



Sounds like a plan

# 8-QUEEN

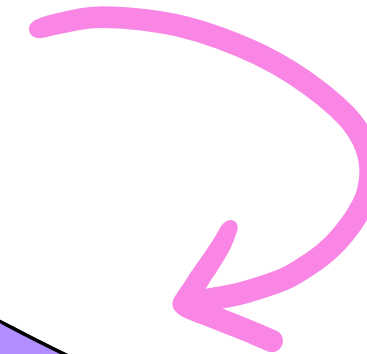
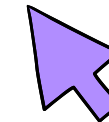
## Using

# Genetic Algorithm



Great idea!

On it!



Let's get started



This presentation is optimized for whiteboard use



**Let's get  
started!**

[Back to Agenda Page](#)



# Agenda



Introduction



8-Queen

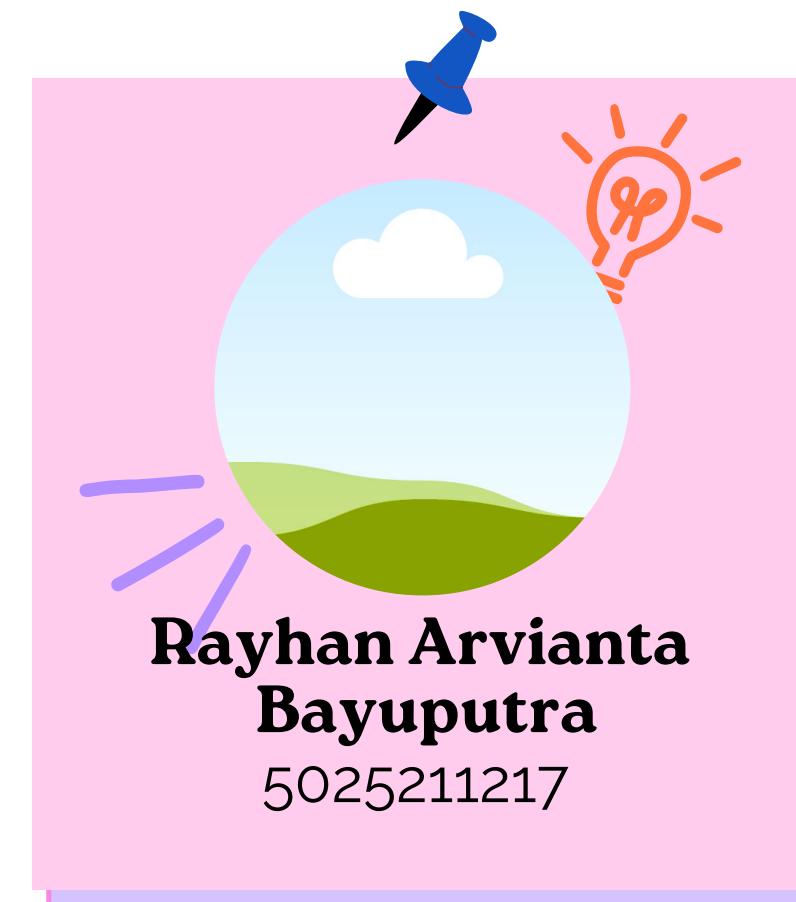


Genetic  
Algorithm

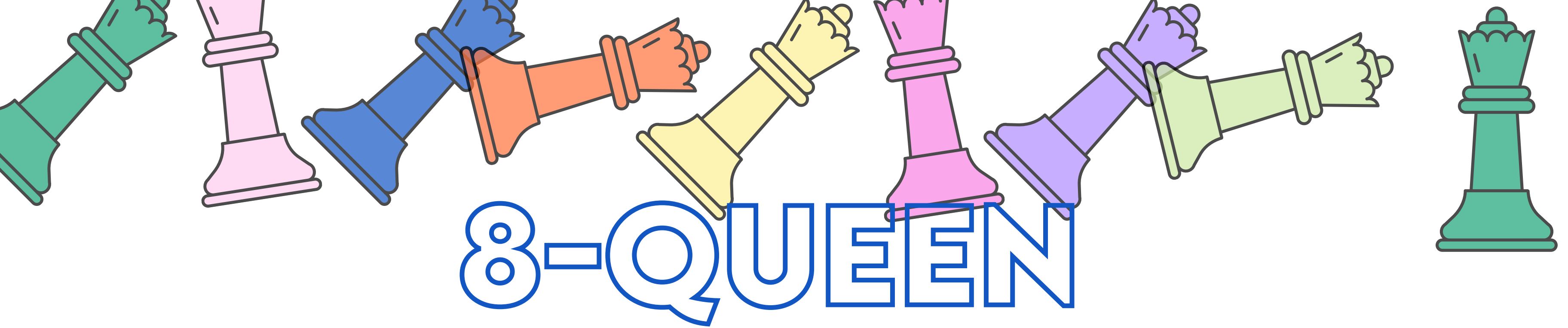


Implementation

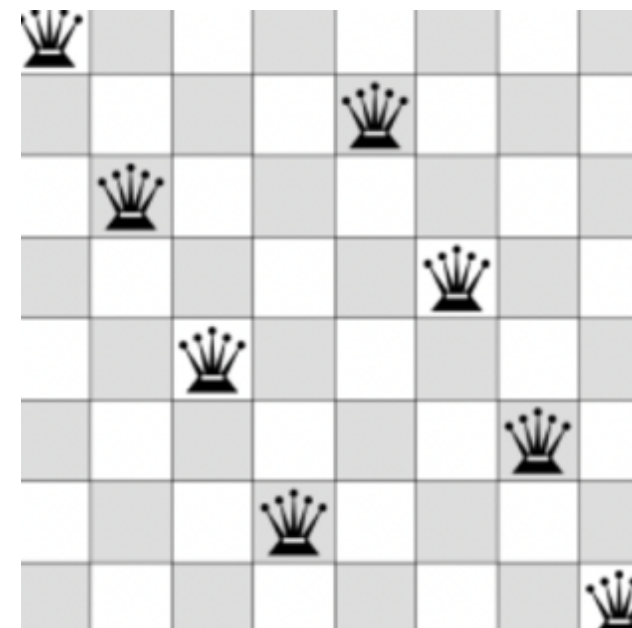
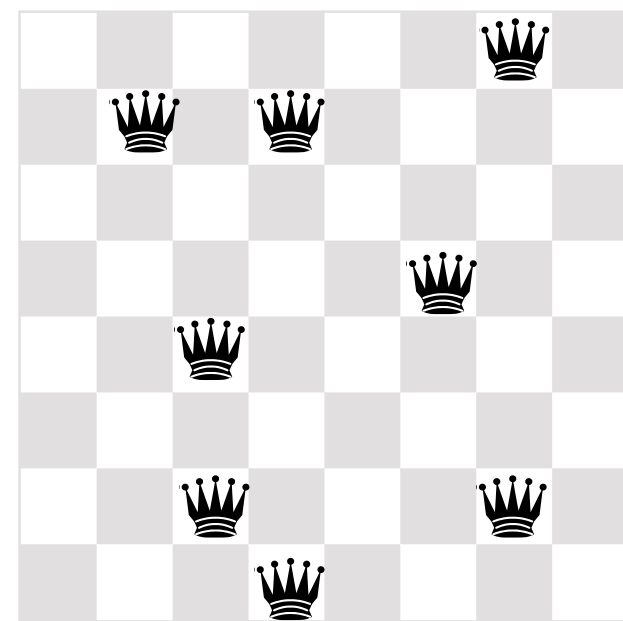
# Introduction



[Back to Agenda Page](#)



Terdapat sebuah papan catur yang masih kosong, problem yang akan kita selesaikan adalah meletakkan pion queen sebanyak 8 buah dengan posisi tertentu agar pion queen tidak bisa saling menyerang satu sama lain. Ketentuan dalam permainan catur adalah queen bisa bergerak secara diagonal atau tegak lurus



# Genetic Algorithm

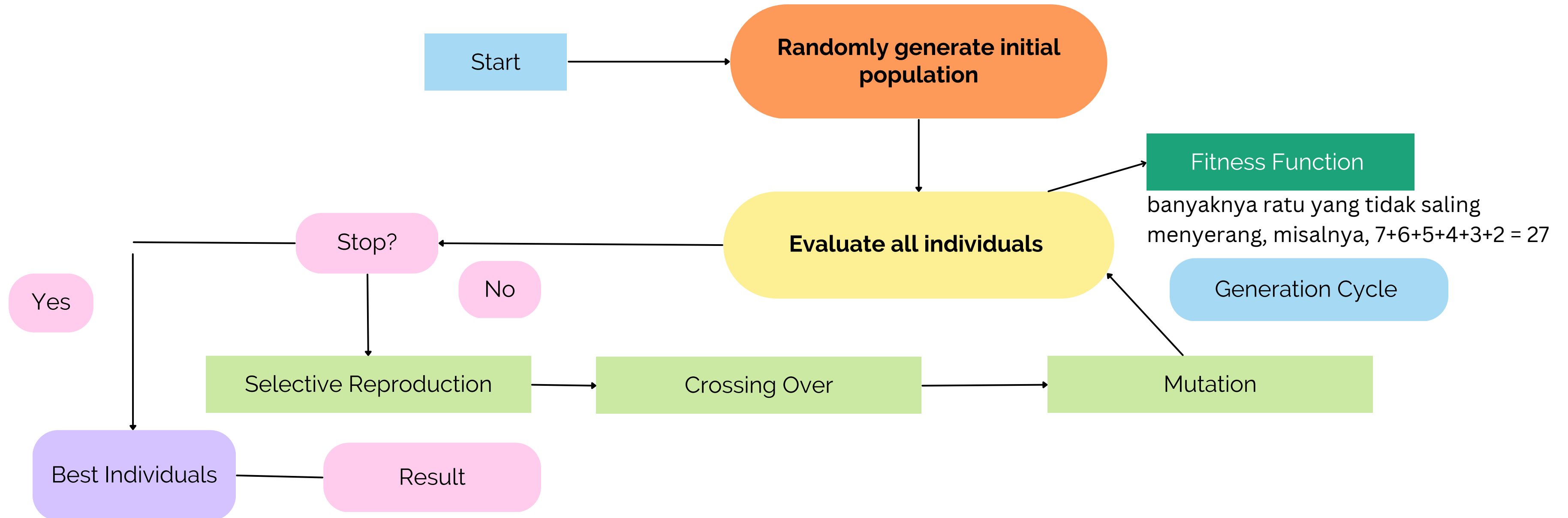
GA beroperasi pada populasi solusi, yang dimanipulasi selama beberapa iterasi yang disebut generasi

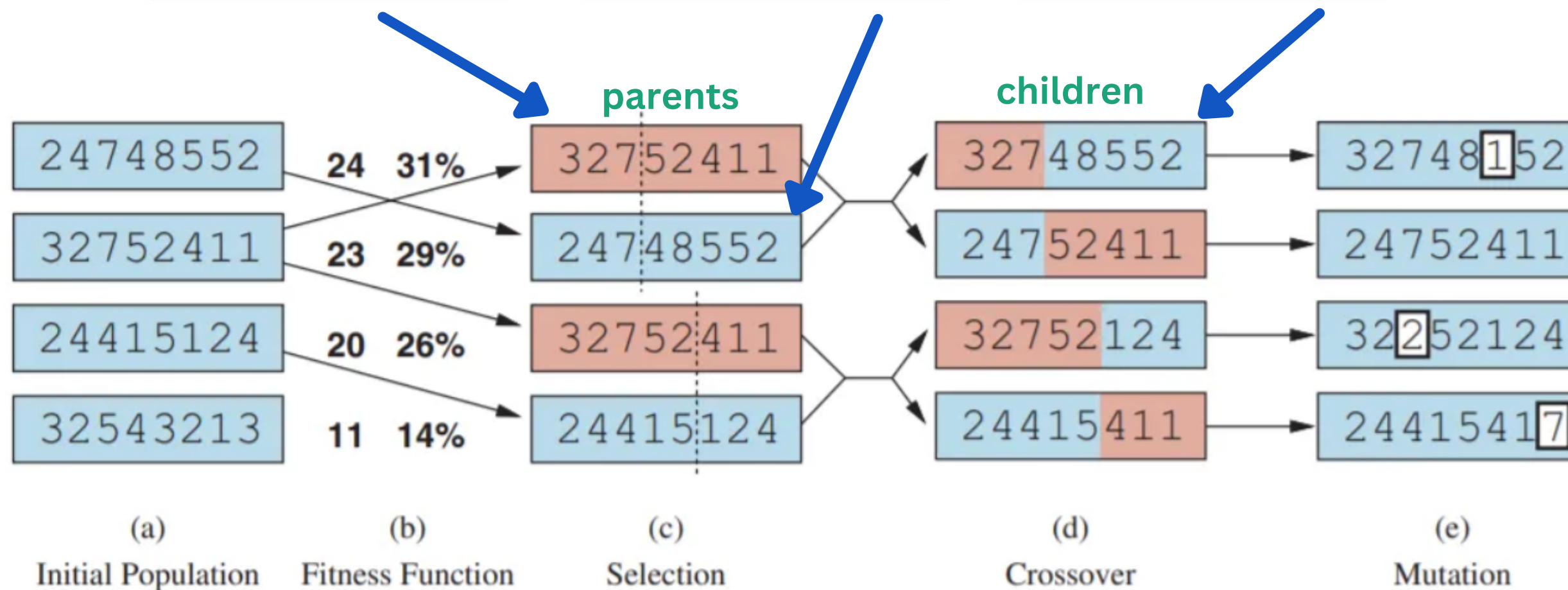
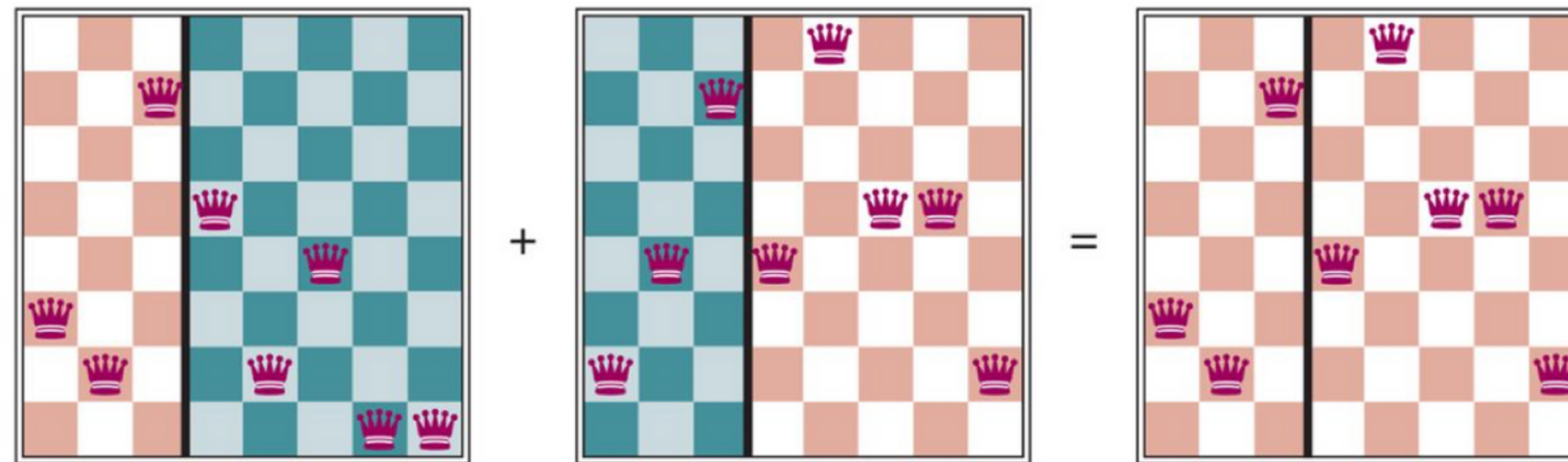


Solusi yang lebih baik secara progresif diidentifikasi, saat algoritma memasangkan kromosom induk untuk menghasilkan keturunan, atau menerapkan mutasi acak pada kromosom yang sudah dihasilkan sebelumnya

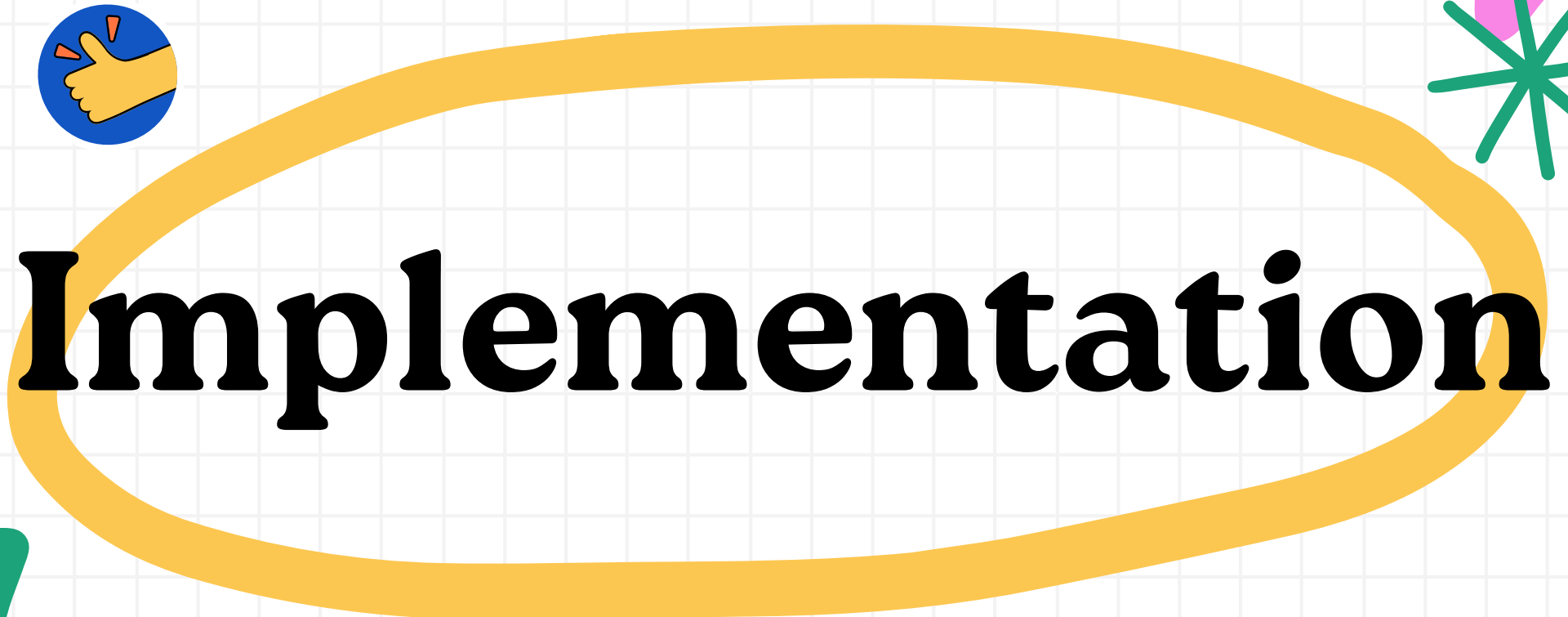
[Back to Agenda Page](#)

# Genetic Algorithm Diagram









# Implementation

# Setting Constants and Parameters



NUM\_QUEENS = 8 menentukan ukuran papan dan jumlah ratu yang akan ditempatkan di atasnya.  
POPULATION\_SIZE = 10 menetapkan jumlah individu dalam setiap populasi.  
MIXING\_NUMBER = 2 menentukan berapa banyak orang tua yang digunakan untuk rekombinasi.  
MUTATION\_RATE = 0.05 menetapkan probabilitas masing-masing elemen urutan genetik individu mengalami mutasi acak.



```
2  NUM_QUEENS = 8
3  POPULATION_SIZE = 10
4  MIXING_NUMBER = 2
5  MUTATION_RATE = 0.05
```



# fitness score

**fitness\_score(seq)** adalah fungsi yang menghitung skor kecocokan (fitness score) suatu individu dalam populasi. Fungsi ini menerima satu parameter **seq**, yaitu sebuah list yang berisi posisi tiap ratu di papan catur. Fungsi ini mengembalikan skor kecocokan (fitness score) dari individu, di mana skor ini menunjukkan jumlah pasangan ratu yang tidak saling menyerang.



```
7 def fitness_score(seq):
8     score = 0
9
10    for row in range(NUM_QUEENS):
11        col = seq[row]
12
13        for other_row in range(NUM_QUEENS):
14
15            #queens cannot pair with itself
16            if other_row == row:
17                continue
18            if seq[other_row] == col:
19                continue
20            if other_row + seq[other_row] == row + col:
21                continue
22            if other_row - seq[other_row] == row - col:
23                continue
24            #score++ if every pair of queens are non-attacking.
25            score += 1
26
27    #divide by 2 as pairs of queens are commutative
28    return score/2
```



# Selection




```
33 def selection(population):
34     parents = []
35
36     for ind in population:
37         #select parents with probability proportional to their fitness score
38         if random.randrange(sc.comb(NUM_QUEENS, 2)*2) < fitness_score(ind):
39             parents.append(ind)
40
41
42     return parents
```

**selection(population)** adalah fungsi yang melakukan seleksi orang tua (parent selection) dari populasi saat ini. Fungsi ini menerima satu parameter **population**, yaitu list yang berisi seluruh individu dalam populasi. Fungsi ini mengembalikan list yang berisi individu yang dipilih sebagai orang tua dengan probabilitas sebanding dengan skor kecocokan mereka.

# crossover



**crossover(parents)** adalah fungsi yang melakukan rekombinasi (recombination) atau persilangan antara beberapa orang tua (parents). Fungsi ini menerima satu parameter **parents**, yaitu list yang berisi individu yang dipilih sebagai orang tua untuk melakukan rekombinasi. Fungsi ini mengembalikan list yang berisi semua anak turunan (offspring) yang dihasilkan dari rekombinasi orang tua yang diberikan



```
46 def crossover(parents):
47
48     #random indexes to to cross states with
49     cross_points = random.sample(range(NUM_QUEENS), MIXING_NUMBER - 1)
50     offsprings = []
51
52     #all permutations of parents
53     permutations = list(itertools.permutations(parents, MIXING_NUMBER))
54
55     for perm in permutations:
56         offspring = []
57
58         #track starting index of sublist
59         start_pt = 0
60
61         for parent_idx, cross_point in enumerate(cross_points): #doesn't account for last parent
62
63             #sublist of parent to be crossed
64             parent_part = perm[parent_idx][start_pt:cross_point]
65             offspring.append(parent_part)
66
67             #update index pointer
68             start_pt = cross_point
69
70         #last parent
71         last_parent = perm[-1]
72         parent_part = last_parent[cross_point:]
73         offspring.append(parent_part)
74
75         #flatten the list since append works kinda differently
76         offsprings.append(list(itertools.chain(*offspring)))
77
78     return offsprings
```





# mutate

**mutate(seq)** adalah fungsi yang memodifikasi urutan (sequence) dengan memilih secara acak satu atau lebih posisi dalam urutan dan mengubah nilainya menjadi nilai acak baru.

```
80 def mutate(seq):  
81     for row in range(len(seq)):  
82         if random.random() < MUTATION_RATE:  
83             seq[row] = random.randrange(NUM_QUEENS)  
84  
85     return seq
```



# print goal



`print_found_goal(population, to_print=True)` adalah fungsi untuk mencetak individu-individu dalam populasi yang memiliki skor kebugaran tertinggi, dan mencetak pesan 'Solution found' jika ada solusi yang ditemukan. Fungsi ini mengembalikan **True** jika ada solusi yang ditemukan dan **False** jika tidak ada solusi. Jika **to\_print** diatur menjadi **False**, maka fungsi ini tidak akan mencetak apa pun ke layar.

```
87 def print_found_goal(population, to_print=True):
88     for ind in population:
89         score = fitness_score(ind)
90         if to_print:
91             print(f'{ind}. Score: {score}')
92             if score == sc.comb(NUM_QUEENS, 2):
93                 if to_print:
94                     print('Solution found')
95                 return True
96
97     if to_print:
98         print('Solution not found')
99     return False
```





# evolution



```
101 def evolution(population):
102     #select individuals to become parents
103     parents = selection(population)
104
105     #recombination. Create new offsprings
106     offsprings = crossover(parents)
107
108     #mutation
109     offsprings = list(map(mutate, offsprings))
110
111     #introduce top-scoring individuals from previous generation and keep top fitness individuals
112     new_gen = offsprings
113
114     for ind in population:
115         new_gen.append(ind)
116
117     new_gen = sorted(new_gen, key=lambda ind: fitness_score(ind), reverse=True)[:POPULATION_SIZE]
118
119     return new_gen
```



**evolution(population)**: adalah fungsi yang mengatur algoritma evolusi dalam program. Fungsi ini memanggil fungsi **selection**, **crossover**, dan **mutate** untuk menciptakan generasi baru dari populasi saat ini.





# generate population



```
121 def generate_population():
122     population = []
123
124     for individual in range(POPULATION_SIZE):
125         new = [random.randrange(NUM_QUEENS) for idx in range(NUM_QUEENS)]
126         population.append(new)
127
128     return population
```



**generate\_population()** adalah fungsi untuk menghasilkan populasi awal yang terdiri dari beberapa urutan (sequences) acak, masing-masing merepresentasikan penempatan ratu pada papan catur yang berbeda-beda.



# menampilkan solusi



```
132 #generate random population
133 population = generate_population()
134
135 while not print_found_goal(population):
136     print(f'Generation: {generation}')
137     print_found_goal(population)
138     population = evolution(population)
139     generation += 1
```



Pertama, populasi awal dihasilkan dengan memanggil fungsi **generate\_population()**. Selanjutnya, loop **while** dijalankan selama tidak ditemukan solusi yang tepat (yaitu fungsi **print\_found\_goal()** mengembalikan **False**). Di setiap iterasi, informasi tentang generasi saat ini dicetak, kemudian fungsi **print\_found\_goal()** dipanggil untuk mencetak populasi saat ini dan mengecek apakah solusi telah ditemukan. Kemudian, populasi di-update dengan memanggil fungsi **evolution()**, dan nomor generasi ditingkatkan sebanyak 1. Loop ini terus diulangi sampai solusi ditemukan.