APPENDIX

```
import random
import numpy as np

class Makanan:
    def __init__(self, name, calories, protein, fat, carb, fiber):
        self.name = name
        self.calories = calories
        self.protein = protein
        self.fat = fat
        self.carbohydrate = carbohydrate
        self.serat = serat

class Individu:
    def __init__(self, makanan_list):
        self.makanan_list = makanan_list  # List of Makanan objects
        self.fitness = 0.0

class ARTIFICIAL IMMUNE SYSTEM ALGORITHMMenuDiabetes:
    def __init__(self):
        # Healthy food database for diabetes
        self.daftar_makanan = [
            # Source of Carbohydrates
            Food("Brown Rice (100g)", 110, 2.5, 0.5, 23, 1.0),
            Food("Oatmeal (100g)", 68, 2.4, 1.4, 12, 1.7),
            Food("Whole Wheat Bread (2 slices)", 138, 6, 1.8, 26, 4.0),
            Food("Boiled Sweet Potatoes (100g)", 86, 1.6, 0.1, 20, 3.0),
            Food("Corn Boiled (100g)", 96, 3.4, 1.5, 21, 2.4),

            # Protein Source
            Food("Boiled Chicken (100g)", 165, 31, 3.6, 0, 0),
            Food("Grilled Fish (100g)", 129, 22, 4, 0, 0),
            Food("Tofu (100g)", 76, 8, 4.8, 1.9, 0.3),
            Food("Tempeh (100g)", 193, 20, 11, 8, 1.4),
            Food("Hard-boiled eggs (2 eggs)", 155, 13, 11, 1.1, 0),
            Food("Lean Beef (100g)", 250, 26, 15, 0, 0),

            # Vegetables
            Food("Boiled Broccoli (100g)", 35, 2.4, 0.4, 7, 2.6),
            Food("Boiled Spinach (100g)", 23, 2.9, 0.4, 3.6, 2.2),
            Food("Boiled Carrots (100g)", 35, 0.8, 0.2, 8, 2.8),
            Food("Boiled Chickpeas (100g)", 35, 2.4, 0.2, 8, 3.4),
            Food("Boiled Kale (100g)", 28, 2.6, 0.4, 5, 2.0),
```

```
# Fruits
            Food("Apples (1 piece)", 95, 0.5, 0.3, 25, 4.4),
            Food("Pears (1 piece)", 102, 0.6, 0.2, 27, 5.5),
            Food("Oranges (1 piece)", 62, 1.2, 0.2, 15, 3.1),
            Food("Papaya (100g)", 43, 0.5, 0.3, 11, 1.7),

            # Others
            Makanan("Yogurt Plain (100g)", 59, 10, 0.4, 3.6, 0),
            Food("Almond Milk (200ml)", 60, 1, 2.5, 8, 1),
            Foods ("Almonds (30g)", 174, 6, 15, 6, 3.5)
        ]

        # Nutrition target (to be input by the user)
        self.target_nutrisi = {}

        self.population = []
        self.memory_cells = []
        self.generation = 0

    def input_target_nutrisi(self):
        print("=== INPUT DAILY NUTRITION TARGET ===")
        print("Enter nutrition targets for diabetics:\n")

        self.target_nutrisi = {}

        Try:
            self.target_nutrisi['kalori'] = float(input("Target Energy/Kalori: "))
            self.target_nutrisi['protein'] = float(input("Target Protein (g): "))
            self.target_nutrisi['fat'] = float(input("Target Fat(g)"))
            self.target_nutrisi['carbo'] = float(input("Target Carbohydrate(g): "))
            self.target_nutrisi['serat'] = float(input("Target Serat (g): "))
            print("\n✓ Nutrition target successfully saved!")
        except ValueError:
print(" ❌ Input must be a number! Using diabetes default target.")
            self.target_nutrisi = {
                'calories': 1500,
                'protein': 60,
                'fat': 40,
                'Carbohydrates': 150,
                'fiber': 25
            }

    def hitung_total_nutrisi(self, individual):
        total = {'calories': 0, 'protein': 0, 'fat': 0, 'carbohydrates': 0, 'fiber': 0}
        for makanan in individu.makanan_list:
            total['calories'] += food.calories
            total['protein'] += makanan.protein
            total['fat'] += food.fat
            total['carbohydrates'] += food.carbohydrates
```

```
def hitung_total_nutrisi(self, individual):
total = {'calories': 0, 'protein': 0, 'fat': 0, 'carbohydrates': 0, 'fiber': 0}
for makanan in individu.makanan_list:
total['calories'] += food.calories
total['protein'] += makanan.protein
total['fat'] += food.fat
total['carbohydrates'] += food.carbohydrates
total['fiber'] += food.fiber
return total

def evaluasi_fitness(self, individual):
total_nutrisi = self.hitung_total_nutrisi(individual)

# Calculate the difference with the target
selisih_total = 0
for nutrisi in self.target_nutrisi:
target = self.target_nutrisi[nutrisi]
current = total_nutrisi[nutrisi]

# Normalization of the difference by target
if target > 0:
Difference = abs(actual - target) / target
selisih_total += difference

# Fitness = 1 / (1 + total difference)
fitness = 1 / (1 + selisih_total)

# Fitness bonus if the number of meals is reasonable (3-6 meals)
jumlah_makanan = len(individu.makanan_list)
if 3 <= jumlah_makanan <= 6:
fitness *= 1.1# 10% Bonus

return min(fitness, 1.0) # Maximum 1.0

def buat_individu_random(self):
# Choose 3-6 random foods.
jumlah_makanan = random.randint(3, 6)
makanan_terpilih = random.sample(self.daftar_makanan, jumlah_makanan)
return Individu(makanan_terpilih)

def inisialisasi_populasi(self):
self.populasi = [self.buat_individu_random() for _ in range(50)]
for individu in self.populasi:
individual.fitness = self.evaluasi_fitness(individual)

def seleksi_konai(self):
# Sort by fitness
populasi_terurut = sorted(self.populasi, key=lambda x: x.fitness, reverse=True)
```

```
# Take the best 20%
jumlah_terbaik = max(1, int(0.2 * len(populasi_terurut)))
best = populasi_terurut[:jumlah_terbaik]

# Kion (child) production – more for high fitness
kion = []
For Individuals in Best:
jumlah_anak = max(1, int(individu.fitness * 8))
for _ in range(jumlah_anak):
child = self.mutation(individual)
Why?

return kion

def mutation(self, parent):
# Copy food list
makanan_baru = induk.makanan_list.copy()

# Calculate mutation rate based on fitness
mutation_rate = 1 - parent.fitness

# Apply mutation
if random.random() < mutation_rate:
# Mutation operation: add, remove, or replace food
operation = random.choice(['add', 'delete', 'remove'])

if operation == 'add' and len(makanan_baru) < 6:
makanan_tambahan = random.choice(self.daftar_makanan)
if makanan_tambahan not in makanan_baru:
makanan_baru.append(makanan_tambahan)

elif operation == 'delete' and len(makanan_baru) > 3:
makanan_baru.pop(random.randint(0, len(makanan_baru)-1))

elif operation == 'replace':
idx = random.randint(0, len(makanan_baru)-1)
makanan_pengganti = random.choice(self.daftar_makanan)
if makanan_pengganti not in makanan_baru:
makanan_baru[idx] = makanan_pengganti

individu_baru = Individual(makanan_baru)
individu_baru.fitness = self.evaluasi_fitness(individu_baru)
return individu_baru
```

```
def seleksi_memory_cells(self, kion_termutasi):
# Merge old populations and mutated kions
combined = self.population + kion_termutasi

# Sort by fitness
gabungan_terurut = sorted(gabungan, key=lambda x: x.fitness, reverse=True)

# Choose the 50 best for the new population
populasi_baru = gabungan_terurut[:50]

# Save the top 10 as memory cells
self.memory_cells = gabungan_terurut[:10]

return populasi_baru
def jalankan_algoritma(self, max_generasi=100):
print("\n" + "="*60)
print("ARTIFICIAL IMMUNE SYSTEM ALGORITHM - REKOMENDASI MENU DIABETES")
print("="*60)
# Target input from the user
self.input_target_nutrisi()
# Initialization
self.generation = 0
self.inisialisasi_populasi()
print(f"\nStarting evolution...")
print(f"Generation {self.generation}: Best fitness = {max(ind.fitness for ind in self.population):.4f}")
while self.generasi < max_generasi:
self.generation += 1
# Connie Selection
kion = self.seleksi_konai()
# Hypermuted
kion_termutasi = []
for anak in kion:
anak_mutasi = self.mutation(child)
kion_termutasi.append(anak_mutasi)

# Memory Cells Selection
self.population = self.seleksi_memory_cells(kion_termutasi)
if self.generasi % 20 == 0:
best_fitness = max(ind.fitness for ind in self.populasi)
print(f"Generation {self.generation}: Best fitness = {best_fitness:.4f}")

# Output of the final result
self.tampilkan_hasil()
```

```
def tampilkan_hasil(self):
print("\n" + "="*70)
print("FINAL RESULT - DIABETES FOOD MENU RECOMMENDATIONS")
print("="*70)

best_individu = self.memory_cells[0] if self.memory_cells else self.populasi[0]
total_nutrisi = self.hitung_total_nutrisi(best_individu)

 print(f"\n🎯 **RECOMMENDED MENU** (Fitness: {best_individu.fitness:.4f})")
print("-" * 50)

For i, foods in enumerate(best_individu.food_list, 1):
print(f"{i}. {food.name}")

      print(f"\n📊 **TOTAL NUTRISI:**")
print("-" * 60)
for nutrisi in total_nutrisi:
target = self.target_nutrisi[nutrisi]
current = total_nutrisi[nutrisi]
Difference = Actual - Target
Percentage = (difference/target) * 100 if target > 0 else 0

 status = "✓" if abs(percentage) <= 10 else "⚠" if abs(percentage) <= 20 else "✗"
print(f"{status} {nutrition.capitalize():12}: {actual:6.1f} | target: {target:5.1f} | Target: {target:6.1f}
({percentage:+.1f}%)")

 print(f"\n💡 **DESCRIPTION:**")
 print("✓ = Very good (≤ 10% difference)")
 print(" ⚠ = Quite good (≤ 20% difference)")
 print("✗ = Need improvement (> 20% difference)")

# Run the program
if __name__ == "__main__":
 print("🍽 FOOD MENU RECOMMENDATION PROGRAM FOR DIABETICS")
print(" Using the ARTIFICIAL IMMUNE SYSTEM ALGORITHM\n")

ag = ARTIFICIAL IMMUNE SYSTEM ALGORITHMMenuDiabetes()
ag.jalankan_algoritma(max_generasi=100)
```