// 1. Setup Constants and Initial Grid

Define LETTERS, GRID\_SIZE, SUBGRID\_SIZE, POPULATION\_SIZE, GENERATIONS, MUTATION\_RATE

Define INITIAL\_GRID with fixed letters and EMPTY cells

// 2. Generate a Candidate Grid

Function InitializeGrid(initialGrid):

For each cell in initialGrid:

If cell is EMPTY:

Fill with a random letter from LETTERS

Return complete grid

// 3. Create a Population

Function CreatePopulation(initialGrid, size):

For i = 1 to size:

candidate ← InitializeGrid(initialGrid)

Add candidate to population

Return population

// 4. Evaluate Fitness

Function Fitness(grid):

Count and return number of conflicts in rows, columns, and subgrids

// 5. Select Two Parents (Tournament Selection)

Function SelectParents(population):

Randomly pick a small subset from population

Sort subset by Fitness (lowest first)

Return the two best candidates

// 6. Crossover to Create a Child

Function Crossover(parent1, parent2):

Choose a random row index as crossover point

Child ← rows from parent1 up to the crossover point + rows from parent2 after the crossover point

Return child

// 7. Mutate the Child

Function Mutate(child, initialGrid):

For each cell in child:

If corresponding cell in initialGrid is EMPTY and random chance < MUTATION\_RATE:

Change cell to a random letter from LETTERS

Return mutated child

// 8. Main Genetic Algorithm Loop

Function GeneticAlgorithm(initialGrid):

population ← CreatePopulation(initialGrid, POPULATION\_SIZE)

For generation = 1 to GENERATIONS:

Sort population by Fitness

If best candidate has 0 conflicts:

Break loop

newPopulation ← empty list

While newPopulation size < POPULATION\_SIZE:

(parent1, parent2) ← SelectParents(population)

child ← Crossover(parent1, parent2)

child ← Mutate(child, initialGrid)

Add child to newPopulation

population ← newPopulation

Return best candidate and its Fitness

// 9. Execute the Program

Call GeneticAlgorithm(INITIAL\_GRID)

Print the best solution and its fitness score