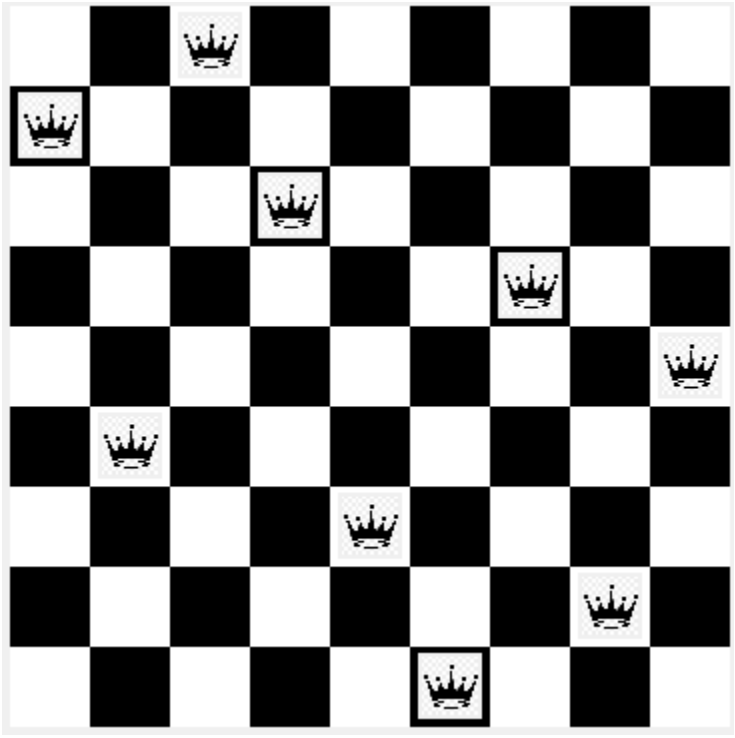
name	ID	specialization	level
على محمد محمود عثمان الديب	20210583	Computer Science	3-passed
عمر أحمد صبري محمد	20210592	Computer Science	3-passed
عبدالرحمن صلاح محمد فكري عليوه	20210512	Computer Science	3-passed
عبدالرحمن صفوت عبدالقادر محمد	20210510	Computer Science	3-passed
عبدالرحمن توفيق عبدالمهيمن	20210495	Computer Science	3-passed
محمد أحمد عبدالباسط مصطفى	20210731	Computer Science	3-passed
محمد حمدي ابراهيم داود	20210766	General	2-continue

Code&Report link on github:

https://github.com/Eldeep1/N-Queens

N-Queens



1.Project overview:

The N-Queens Problem Solver project aims to develop an

intelligent system capable of solving the N-Queens problem for various board sizes. The N-Queens problem is a

classic chessboard puzzle where the objective is to place N queens on an N×N chessboard in such a way that no

two queens threaten each other. Threatening means no two queens share the same row, column, or diagonal.

2. similar applications:

University of San Fransisco N-queen solver:

https://www.cs.usfca.edu/~galles/visualization/RecQueens.html

The site mainly uses recursive algorithm to check for the next place of the queen and it stops when the board is full or there's no place has been checked.

Here's a breakthrough the functions of the Algorithm used on the site

1.CalcQueens

It's the first function used on the application.

It takes the size of the board (we refer to it as n) as its parameters.

It's used to create an array of n elements, with each element holding -1 as a value.

And then, calling the most important method on the program 'queens'.

2.queens

Queens is the most important function of that site.

It takes size, board and current as its parameters.

Size refers to the size of the board, board to the last state of the board and current for the place in which we will try to insert the next queen.

The function will stop it's iterations when the size is equal to current.

The function loops through every element of the board using two variables current and I, current for the column and I for the place on the row.

So, when the function iterates again the value of, I is restarted so that we check the new column from it's first, but current is saved to keep track of the current row!

Every time I increases, the function checks if there's any conflicts or no using noConflicts function.

If there's none, then call queens function again with the value of the new board and current+1.

If we looped through all the values of I and didn't find any place on the column, then that mean that there's no available place to put the queen, and the function returns false

3.noConflicts

That function takes 2 parameters: board and current.

It first checks if there's a queen already on the same row as the board is passed to it, then it checks if there's a queen on the same diagonal or not, if there's any then it returns false, else it returns true.

Algorithm on the site overview:

That site shows animation with the ability to pause on any specific moment, but in different of most solutions, it solves the problem from column by column not row by row.

3..An initial literature review of Academic publications (papers)

Research Paper Number 1:

Research Paper Link : https://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-article-BAT2-0001-1001

Neural networks can be successfully applied to solving certain types of combinatorial optimization problems. In this paper several neural approaches to solving constrained optimization problems are presented and their properties discussed. The main goal of the paper is to present various improvements to the well-known Hopfield models which are intensively used in combinatorial optimization domain. These improvements include deterministic modifications (binary Hopfield model with negative self-feedback connections and Maximum Neural Network model), stochastic modifications (Gaussian Machine), chaotic Hopfield-based models (Chaotic Neural Network and Transiently Chaotic Neural Network), hybrid approaches (Dual-mode Dynamic Neural Network and Harmony Theory approach) and finally modifications motivated by digital implementation feasibility (Strictly Digital Neural Network). All these models are compared based on a commonly used benchmark problem - the N-Queens Problem (NQP). Numerical results indicate that each of modified Hopfield models can be effectively used to solving the NQP. Convergence to solutions rate of these methods is very high - usually close to 100%. Experimental time requirements are generally low - polynomial in most Casos. Some discussion of non-neural, heuristic approaches to solving the NQP is also presented in the paper.

Research Paper Number 2:

Research Paper Link: https://www.researchgate.net/profile/Sathya-Ss/publication/333661204_Survey_on_NQueen_Problem_with_Genetic_Algorithm/links/5cfb7e6e92851c874c56c1d2/Survey-on-N-Queen-Problem-with-Genetic-Algorithm.pdf

The combinatorial optimization problem is a collection of problems which need a sample amount of time and effort

to be solved. Vast difficulties have been occurring to solving these types of problem that there is no exact formula to solve the

problem. Each feasible solution works on some order and the size of the probability increases algorithmically as the number of

the problem also increases dynamically. This paper discusses about N–Queen problem, it is also a type of NP – hard problem.

Many researchers have proposed various methods and algorithms for this problem. Henceforth, Genetic Algorithm is one kind

of famous algorithm for solving NP hard problems. This paper mainly focuses on the review work of genetic algorithm to solve

the N -Queen Problems (NPQ).

Research Paper Number 3:

Research Paper Link:

https://link.springer.com/article/10.1007/s10589-013-9578-z

The n-queens problem is a classical combinatorial optimization problem which has been proved to be NP-hard. The goal is to place n non-attacking queens on an n×n chessboard. In this paper, two single-solution-based (Local Search (LS) and Tuned Simulated Annealing (SA)) and two population-based metaheuristics (two versions of Scatter Search (SS)) are presented for solving the problem. Since parameters of heuristic and metaheuristic algorithms have great influence on their performance, a TOPSIS-Taguchi based parameter tuning method is proposed, which not only considers the number of fitness function evaluations, but also aims to minimize the runtime of the presented metaheuristics. The performance of the suggested approaches was investigated through computational analyses, which showed that the Local Search method outperformed the other two algorithms in terms of average runtimes and average number of fitness function evaluations. The LS was also compared to the Cooperative PSO (CPSO) and SA algorithms, which are currently the best algorithms in the literature for finding the first solution to the n-queens problem, and the results showed that the average fitness function evaluation of the LS is approximately 21 and 8 times less than that of SA and CPSO, respectively. Also, a fitness analysis of landscape for the n-queens problem was conducted which indicated that the distribution of local optima is uniformly random and scattered over the search space. The landscape is rugged and there is no significant correlation between fitness and distance of solutions, and so a local search heuristic can search a rugged plain landscape effectively and find a solution quickly. As a result, it was statistically and analytically proved that single-solution-based metaheuristics outperform population-based metaheuristics in finding the first solution of the n-queens problem.

Research Paper Number 4:

Research Paper Link:

https://iopscience.iop.org/article/10.1088/1757-899X/1116/1/012195/meta

N-Queen is a well-known NP-Hard problem. In N-Queen problem, the challenge is to place n queens on n*n chess board such that no two queens can attack each other. The problem can-not be solved using traditional algorithms. Genetic algorithm is a well-known optimization technique. The problem has to be represented in genetic form to solve it using genetic algorithm. In this paper a new mutation operator is

proposed to solve N-Queen problem. The proposed algorithm is applied on some instances of N-Queen problem and it is observed that the proposed performs better as compared to other existing algorithms.

Research Paper Number 5:

Research Paper Link:

https://beei.org/index.php/EEI/article/view/1351

In this paper, a hybrid of Bat-Inspired Algorithm (BA) and Genetic Algorithm (GA) is proposed to solve N-queens problem. The proposed algorithm executes the behavior of microbats with changing pulse rates of emissions and loudness to final all the possible solutions in the initialization and moving phases. This dataset applied two metaheuristic algorithms (BA and GA) and the hybrid to solve N-queens problem by finding all the possible solutions in the instance with the input sizes of area 8*8, 20*20, 50*50, 100*100 and 500*500 on a chessboard. To find the optimal solution, consistently, ten run have been set with 100 iterations for all the input sizes. The hybrid algorithm obtained substantially better results than BA and GA because both algorithms were inferior in discovering the optimal solutions than the proposed randomization method. It also has been discovered that BA outperformed GA because it requires a reduced amount of steps in determining the solutions.

2.Proposed Solution for the N-Queen problem:

On our application the user can choose a specific board size and then he submits it, so he navigates to another screen.

The second screen contains 4 options, showing the solution using backtracking, using hill climbing, using best first search, using genetic Algorithms, and when the user clicks on any of that button, he can see the animation and the steps that the algorithm had followed to reach to the final solution.

Use case of the system:

3. Applied Algorithms

A)Backtracking:

Backtracking is a systematic algorithmic technique for finding all (or some) solutions to a computational problem that incrementally builds candidates for the solution, and abandons a candidate ("backtracks") as soon as it determines that the candidate cannot possibly be completed to a valid solution. This method is particularly useful for solving problems with a combinatorial nature, such as the N-Queens problem, Sudoku, and the Knight's Tour problem.

In the context of the N-Queens problem, backtracking involves placing queens on the chessboard one by one, and if the current placement violates any constraints, the algorithm backtracks and explores other possibilities.

In our code, we implemented backtracking using four functions:

1. solveNQ

Purpose:

- Initiates the solution search for the N-Queens problem.

Details:

- Clears previous queen placements on the chessboard.
- Randomly selects a starting column for the first queen.
- Calls the solver function to find a solution.
- Updates GUI to show or clear queens based on the solution.
- Displays the number of backtracking trials and whether a solution was found.

2. solver:

Purpose:

- Implements backtracking to find a solution to the N-Queens problem.

Parameters:

- board: 2D array representing the chessboard.
- row: Row index for the current queen placement.
- col: Column index for the current queen placement.
- first: Boolean flag indicating whether it's the first queen placement.
- lim: Limit to control the column movement during backtracking.

Details:

- Recursive function implementing backtracking logic.

- Places queens based on specified conditions and updates the solution sequence.

3. isSafe

Purpose:

- Checks if it's safe to place a queen at a given position on the chessboard.

Parameters:

- board: 2D array representing the chessboard.
- row: Row index of the candidate queen.
- col: Column index of the candidate queen.

Details:

- Checks for conflicts in the same row, the same column, and both diagonals.

4. print_solution

Purpose:

- Visualizes the N-Queens problem solution on a chessboard GUI.

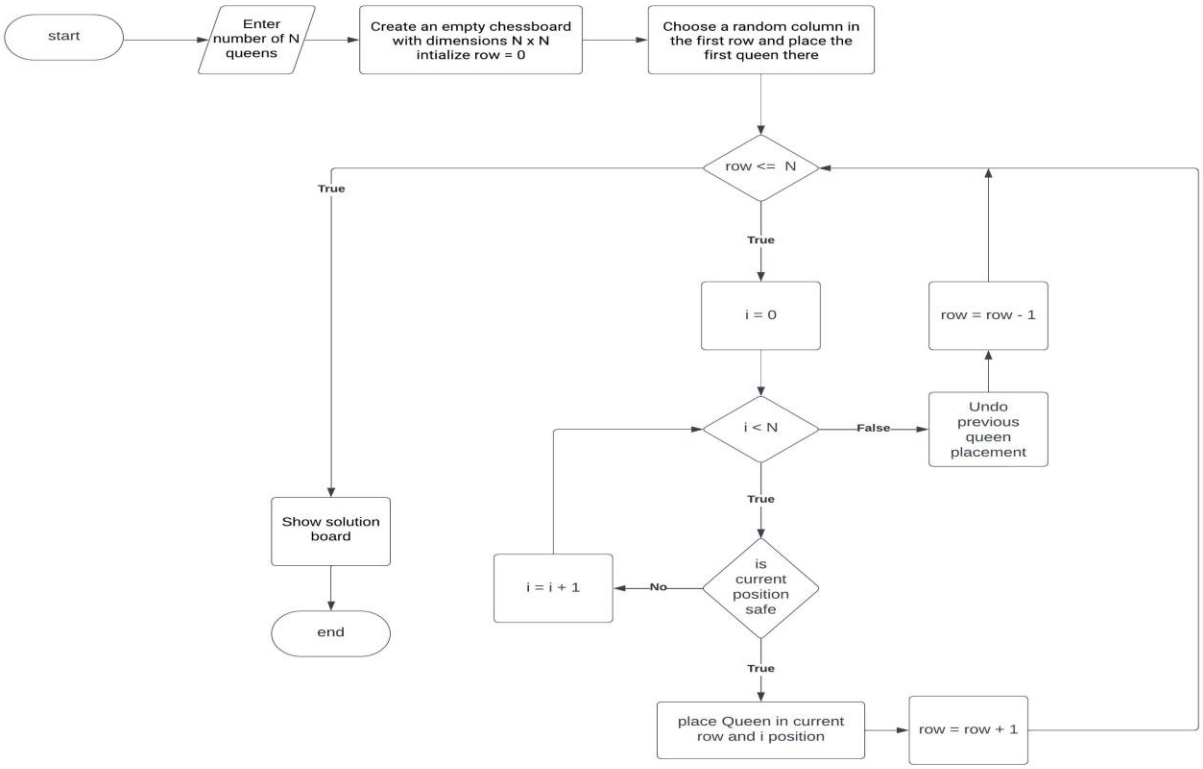
Parameters:

- queue_index: An index to keep track of the current queen placement in the solution sequence.

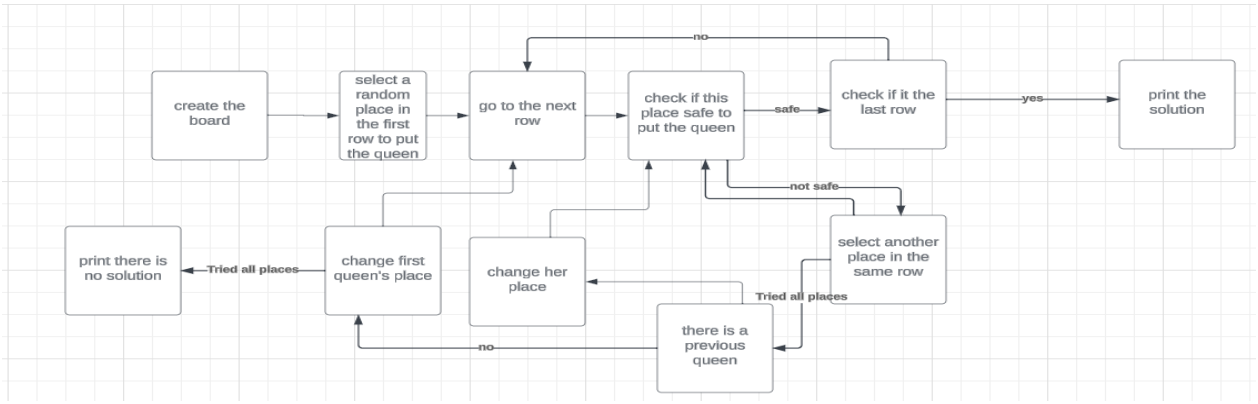
Details:

- Clears the chessboard if it's not the first backtracking attempt.
- Iterates through queen positions and updates the GUI to show or remove queens accordingly.
- Uses after method to schedule periodic updates.

Flowchart:



Block Diagram:



B)Hill Climbing:

Hill climbing is a local search algorithm used for mathematical optimization problems. The goal is to find the peak of the mountain (maximum or minimum) by making incremental steps toward the direction of increasing elevation (for maximization) or decreasing elevation (for minimization).

In our code, we implemented hill climbing using three functions:

1. hill_climbing

Purpose:

- Implements the hill-climbing algorithm to find a solution for the N-Queens problem.
- Parameters:
- n: Size of the chessboard.

Details:

- Initializes the solution with random queen placements.
- Iteratively explores neighboring solutions, choosing the one with fewer attacks.
- Updates the GUI to visualize each step of the algorithm.
- Returns the final solution state.

2. calculate_attacks

Purpose:

- Calculates the number of queen attacks (conflicts) in a given state.

Parameters:

- board: List representing the current queen placements.
- n: Size of the chessboard.

Details:

- Checks for horizontal, vertical, and diagonal attacks between queens.
- Returns the total number of attacks.

3. print_hill_climbing

Purpose:

- Visualizes the N-Queens problem solution on a chessboard GUI.

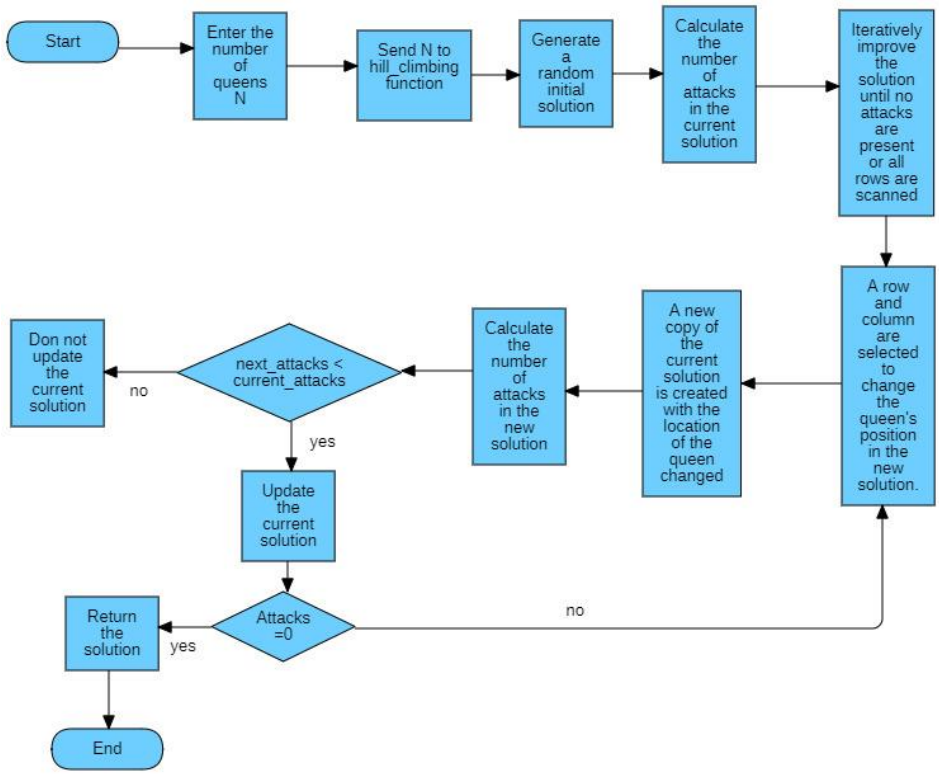
Parameters:

- queue_index: An index to keep track of the current queen placement in the solution sequence.

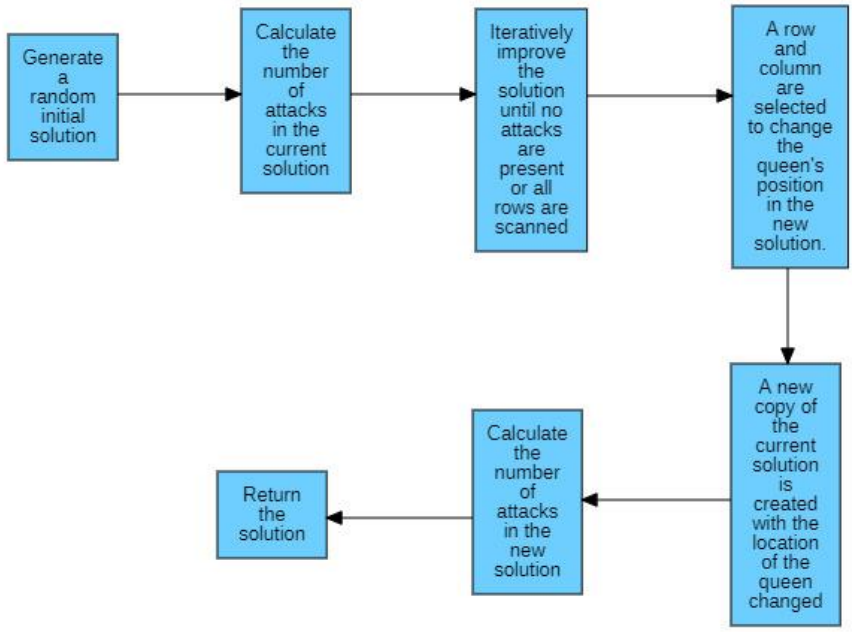
Details:

- Prints the current state during each step of the algorithm.
- Updates the GUI to show changes in queen positions on the chessboard.
- Utilizes a delay of 1000 milliseconds (1 second) between updates.

Flowchart:



Block Diagram:



C)Best First Search:

Best-First Search is an informed search algorithm that explores a graph or tree by selecting the most promising node based on a heuristic function. The goal is to reach the solution by prioritizing nodes that are estimated to be closer to the goal.

In our code, we implemented best first search using six functions:

1. get_solution_and_drow:

Purpose:

- Orchestrates the execution of the Best-First Search algorithm and visualizes the solution.

Details:

- Generates a random initial state.
- Calls best_first_search to find a solution.
- Visualizes the solution path on the chessboard.
- Prints the number of trials and updates GUI variable

2. heuristic_function:

Purpose:

- Computes a heuristic value for a given state, indicating the number of conflicts.

Details:

- Evaluates conflicts in the same row, column, and diagonals.
- Returns the total number of conflicts.

3. make_random_itial_state:

Purpose:

- Generates a random initial state for the N-Queens problem.

Details:

- Shuffles the positions of queens on the chessboard.
- Returns the randomly generated initial state.

4. best_first_search:

Purpose:

- Executes the Best-First Search algorithm to find a solution for the N-Queens problem.

Details:

- Uses a priority queue to explore nodes based on heuristic values.
- Terminates when the goal state (heuristic = 0) is reached or the entire search space is explored.

5. get_successor_states:

Purpose:

- Returns all child states (successors) from a given state.

Details:

- Creates successor states by moving queens vertically.
- Handles edge cases to ensure valid successor states.
- Returns a list of successor states.

6. plot_queens_board:

Purpose:

- Visualizes the N-Queens problem solution on a chessboard GUI.

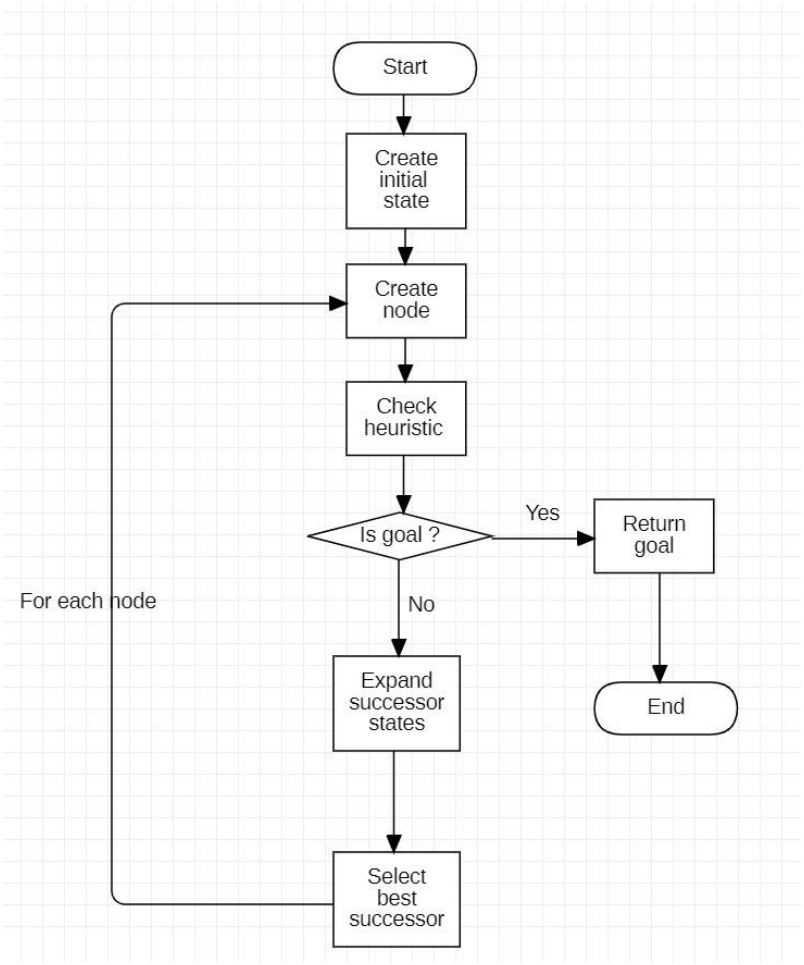
Parameters:

- queue_index: An index to keep track of the current queen placement in the solution sequence.

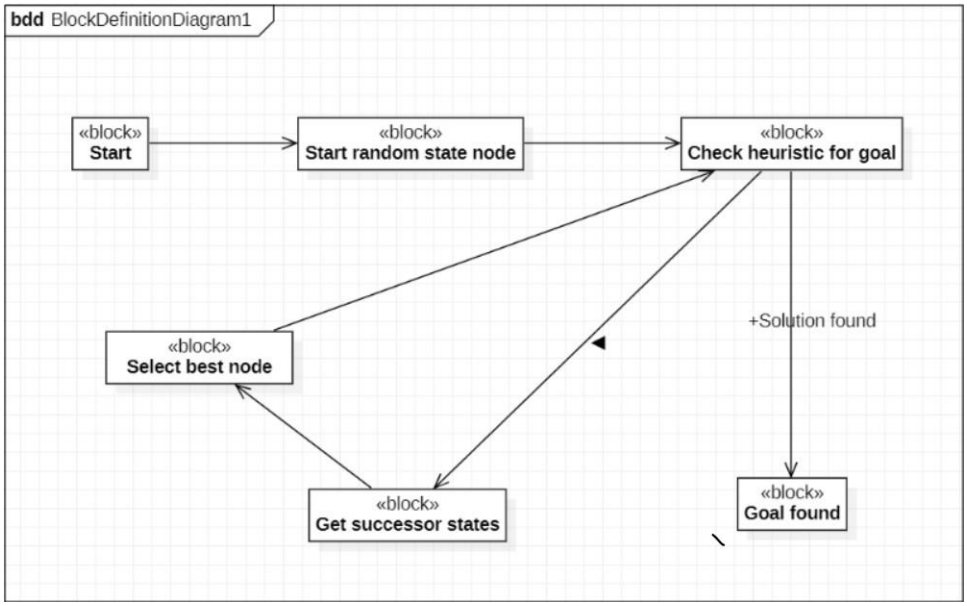
Details:

- Prints the current state during each step of the algorithm.
- Updates the GUI to show changes in queen positions on the chessboard.
- Utilizes a delay of 1000 milliseconds (1 second) between updates.

Flowchart:



Block Diagram:



D)Genetic:

Genetic Algorithms (GAs) are optimization algorithms inspired by the process of natural selection. They are designed to find approximate solutions to optimization and search problems through the principles of evolution.

In our code, we implemented genetic algorithm using eight functions:

1. n_queen_genetic:

Purpose:

- Executes the Genetic Algorithm to find a solution for the N-Queens problem.

Details:

- Initializes a random population and iteratively evolves it using genetic operators.
- Tracks the best solution and prints progress during each generation.
- Updates GUI variables with solution and trial information.

2. init_pop:

Purpose:

- Initializes a population with random queen placements.

Details:

- Generates a population matrix with dimensions (pop_size, board_size).
- Randomly assigns queen positions on the chessboard for each individual.

3. calc_fitness:

Purpose:

- Evaluates the fitness (number of conflicts) for each individual in the population.

Details:

- Computes fitness values based on the number of conflicts for each queen.
- Returns an array of negative fitness values.

4. Selection

Purpose:

- Selects individuals from the population for reproduction based on their fitness.

Details:

- Computes selection probabilities based on fitness values.
- Randomly selects individuals from the population with probabilities proportional to their fitness.
- Returns the selected population.

5. crossover_mutation:

Purpose:

- Applies crossover and mutation to pairs of selected individuals to generate a new population.

Details:

- Iterates through pairs of selected individuals and applies crossover and mutation.
- Returns a new population.

6. Crossover:

Purpose:

- Performs crossover (recombination) between two parents to create two children.

Details:

- Randomly selects a crossover point and combines genetic material from two parents.
- Returns two children.

7. Mutation:

Purpose:

- Introduces random changes (mutation) to an individual's genetic material.

Details:

- Randomly selects a gene and modifies it with a new random value.
- Returns the mutated individual.

8. print_board:

Purpose:

- Visualizes the N-Queens problem solution on a chessboard GUI.

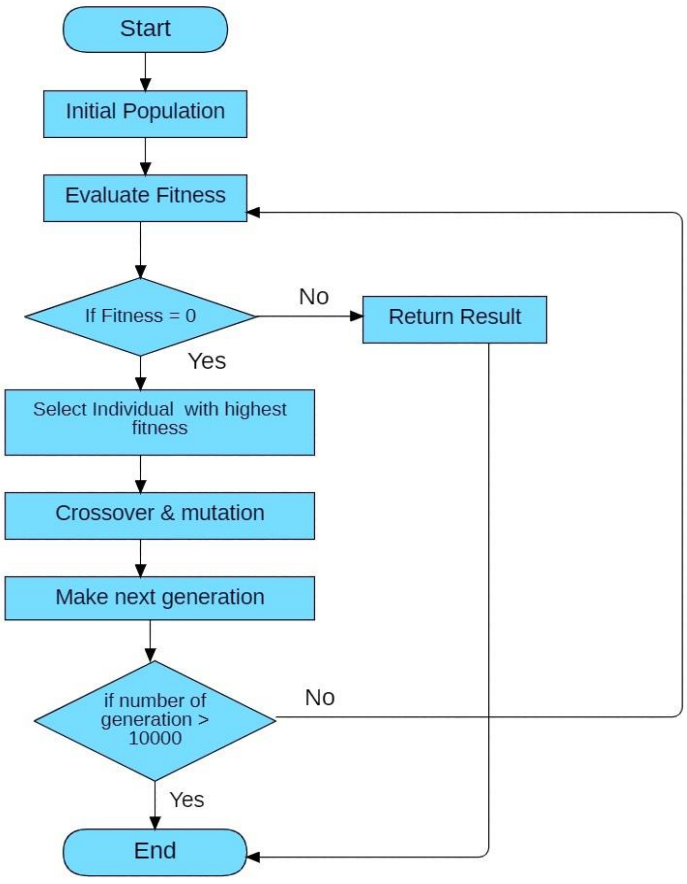
Parameters:

- queue_index: An index to keep track of the current queen placement in the solution sequence.

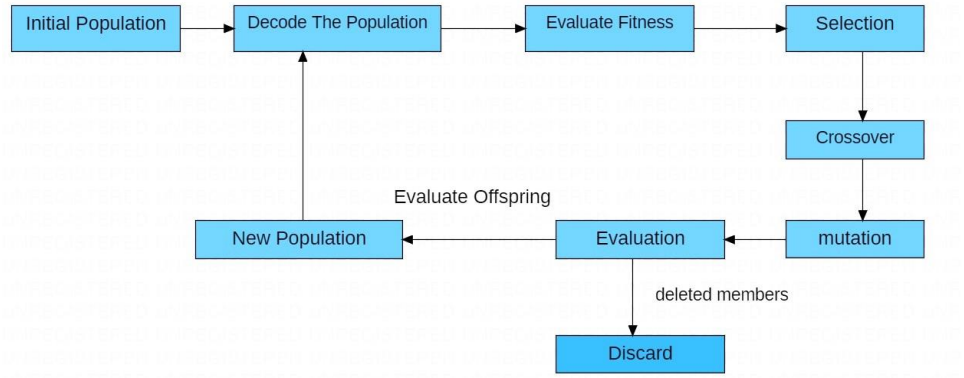
Details:

- Prints the current state during each step of the algorithm.
- Updates the GUI to show changes in queen positions on the chessboard.
- Utilizes a delay of 1000 milliseconds (1 second) between updates.

Flowchart:



Blockdiagram:



4. Experiments & Results:

We can test the results by whether the solution was found or not

- The best first search always find the solution, but may take long time with large solution spaces (large N) because it explores all the solution spaces
- The hill climbing algorithm may find the solution if the solution space is small, but it may stuck in local optima with large solution space N
- The genetic algorithm always finds a successful solution within good amount of time and time of complexity however the N is large or small comparing to other algorithms
- The back tracking algorithm also explores all possible solutions, so it may take a large amount of time with large N, but it good with small N

5. Analysis, advantages / disadvantages, and Future Work:

1- best first search

Insights: best first search is a heuristic algorithm choose the best paths first, this algorithm may use the heuristic such as the number of conflicts (attacks) of queens

Results Analysis: Best first search can find the solution by exploring the best path early , and the effectiveness depends on the choose of heuristic function

Advantages: can explore the best path first, loading to quick convergence, and

Can handle problems with large solution space

Disadvantages: highly dependent on the choosing of heuristic, and uses more power consumption for large N

2- Hill Climbing:

Insights: Hill climbing is a local search algorithm that iteratively moves toward the direction of increasing the evaluation function until the solution is reached

Results Analysis: hill climbing may stuck in local optima depends on the choice of the initial state and the local search technique, it might find the global for smaller N, it may stuck in local for large N

Advantages: ease of implementation, and works well and quickly finding the solution with small N

Disadvantages: may stuck on local optima, and sensitive to initial state

3- Genetic Algorithm:

Insights: genetic algorithms use principles by biological evaluation, such as evolution , crossover, mutation, selection, solutions are treated by the best population which generated

Results Analysis: genetic algorithms can explore different solutions (populations), And can escape the local optima for large N, the effectiveness depends on the Representaion of the last population, and they are suitable for finding the best solution in a reasonable amount of time even for large N

Advantages: very well with complex problems with large solution space

Disadvantages: computationally expensive for large populations

4 - Backtracking:

Insights: backtracking is a systematic algorithm that explore all possible solutions and ignore the paths which cannot load to valid solution

Results Analysis: this algorithm cannot load to invalid path or stuck in local optima, but it's time complexity can by high for large N, making it less efficient compared to the previous algorithms

Advantages: guarantees finding the optimal solution if it exists, and good with Problems with large solution space

Disadvantages: time complexity may be high with large problem space and it Couldn't be work