# ARTIFICIAL INTELLIGENCE PROJECT 1 REPORT

N-Queens Problem Solver using Differential Evolution.

**GITHUB LINK:**
https://github.com/AbdELrahmanYasser1/N-Queens-problem-solver

**Team ID: 12**

**Team Members:**

- Abdullah Shaaban Abd El-Razeq.
- AbdulRahman Hamdy Abd Al-Aaty.
- Abdulrahman Yasser.
- Ayat Ayman Hafez El-Shaal.
- Moaaz Soliman Sayed.
- Mohammed Ali Anwr.
- Taha Saeed Shaaban.

# Table of contents:

# Project 1 Documentation

**N-Queens Problem Solver** (for different sizes – n should be selected by the user) using Differential Evolution.

| القسم / الشعبة | المستوى الدراسي | ID | اسم الطالب |
|---|---|---|---|
| CS | الثالث | 202000171 | آيات أيمن حافظ الشال |
| CS | الثالث | 202000479 | طه سعيد شعبان |
| CS | الثالث | 202000548 | عبد الله شعبان عبد الرازق |
| CS | الثالث | 202000510 | عبد الرحمن حمدى عبد العاطى |
| CS | الثانى | 202000542 | عبد الرحمن ياسر |
| CS | الثالث | 202000800 | محمد على أنور |
| CS | الثالث | 202000923 | معاذ سليمان سيد |

## 1. Instructions to Students:

o This is a group work project. Each group consists of **Six to Seven students** “General Program” /Four **to Five** “Med. Info. Program” *(group members must be approved by the Teaching Assistant through registration).*

o Each group must develop the idea assigned to them using Python.

o **Project Objectives:** The objectives of this project can be summarized as applying the main ideas, fundamental concepts, and key algorithms in the fields of artificial intelligence, and machine learning.

o **Submission:** Submission is done according to the following schedule:

- **Week 7:** During the Weekly Labs

 Registration: Every team leader should register his/her team according to the regulations/steps announced by the TAs during the labs.

- ***Week 9 / 10:*** **Submission and Discussion of a (1) Documentation.**

**The report should include the following: (1) Project idea in details, (2) Main functionalities, (3) Similar applications in the market, (4) An initial literature review of Academic publications (papers) relevant to the idea (at least 2 papers), (5) the Dataset employed (preferably a publicly available dataset), (6) Details of the algorithm(s)/approach(es) that will be used.**

- **Week 13:** Submission and Discussion of the (1) Project, and

(2) Documentation. The report should include the following: (1) Project idea in details, (2) Main functionalities, (3) Similar applications in the market, (4) A literature review of Academic publications (papers) relevant to the idea (at least 5 papers), (5) the Dataset employed (preferably a publicly available dataset), (6) Details of the algorithm(s)/approach(es) used and the results of the experiments, and (7) Development platform.

- **Assessment:** Assessment will be on the reports and code submitted, in addition to discussions with team members. All the team members must contribute to all the phases, and the role of each member must be clearly stated in each report.
- The Project will be assessed based on the following criteria:
   -  The complexity of the problem, & the correctness of the algorithms employed.
   -  The quality/comprehensiveness of your experiments & documentation.
   -  The correctness of your analysis and design diagrams.
   -  Implementation correctness.

# I. Introduction and Overview.

## 2. Project idea in details.

### N-Queens Problem

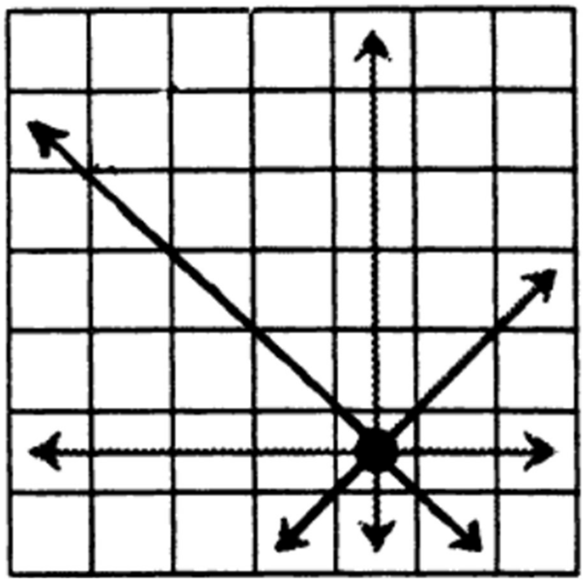


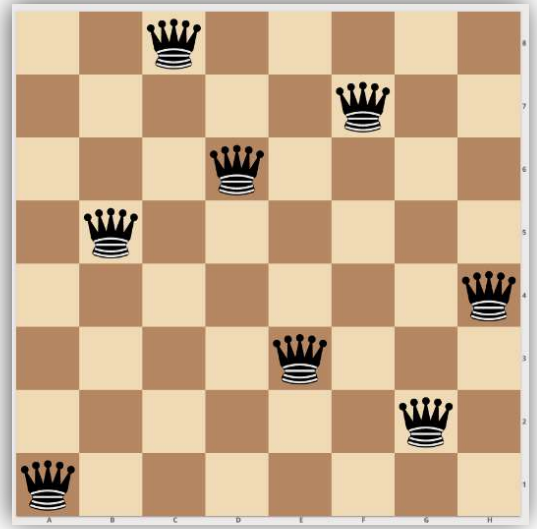
Figure. 1

The NQP is a classical artificial intelligence problem. It is a general case of the 8-Queens problem. This combinatorial optimization problem has been studied for more than a century. The 8-Queens problem was first introduced by a chess player, Max Bezzel, in 1848. Since 1850, the problem has attracted the attention of several famous mathematicians including Gauss, Polya, and Lucas.

The N-Queens problem can be defined as follows: place N queens on an N x N chessboard, each queen on a square, so that no queen could capture any of the others, that is, a configuration in which there exists at most one queen on a given row, column, or diagonal as shown is Figure 1.

A solution for the 8QP is given in Figure figure 2. A possible solution of the N-queens problem where N=8. During the last three decades, the problem has been discussed in the context of computer science and used as an example of backtrack algorithms, permutation generation, divide and conquer paradigm, program development methodology, constraint satisfaction problems, integer programming, specification, and neural networks [1, 12].

A common way to solve this problem consists in trying to put the queens on the board squares one after the other. If one queen threatens the newly introduced queen, we withdraw the queen and search for another position. If we cannot find a solution, we choose to remove a queen already positioned, assign it another position that has not yet been used, and start the search again.



This last operation is called a back-track, and the whole strategy is called a trial-and-error algorithm. It is known that for *N=8*, there are exactly 92 solutions, or less if we consider symmetric solutions as equal. The number of solutions for n = 1, 2... 15, is 1, 0, 0, 2, 10, 4, 40, 92, 352, 724, 2680, 14200, 73712, 365596, 2279184.

## 3. Similar applications in the market.

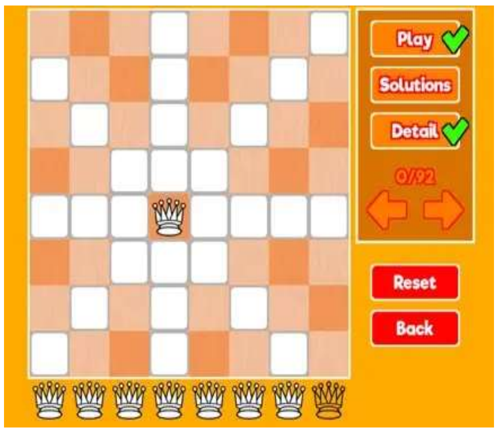
- ### Apple Eight Queens - (8 Queens)

Designed for iPad, By Michal Galusko.

The eight queens puzzle requires a player to place eight chess queens on an 8x8 chessboard so that no two queens are threatening each other. The solution requires that no two queens share the same row, column, or diagonal. The researchers are thrilled to find solution for this puzzle. However, they want the algorithm to work on a bigger square chess board.

**Features:**

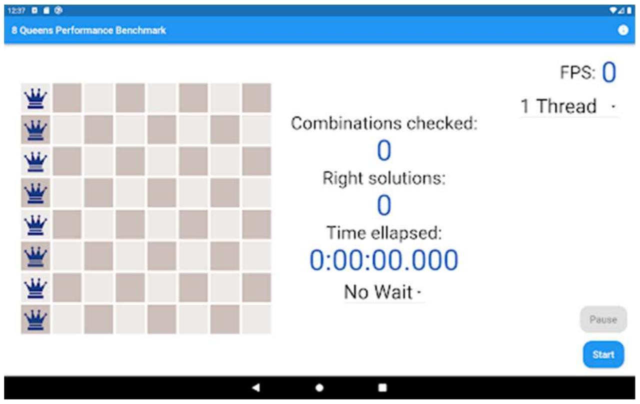
- Play mode (4x4, 5x5, 6x6, 7x7, 8x8)
- Solution mode - for analyse.



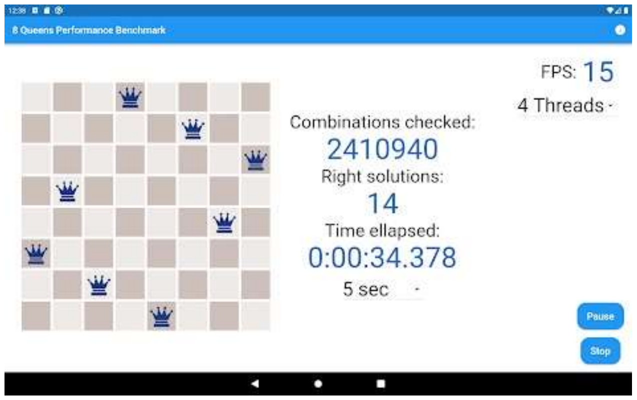
## Queens performance Benchmark.

By Jack Davis.

Eight Queens Performance Benchmark Test and Meter App is a Visual CPU Performance App. You can meter your smart mobile phone's speed with the Eight Queens Chess Problem solving.

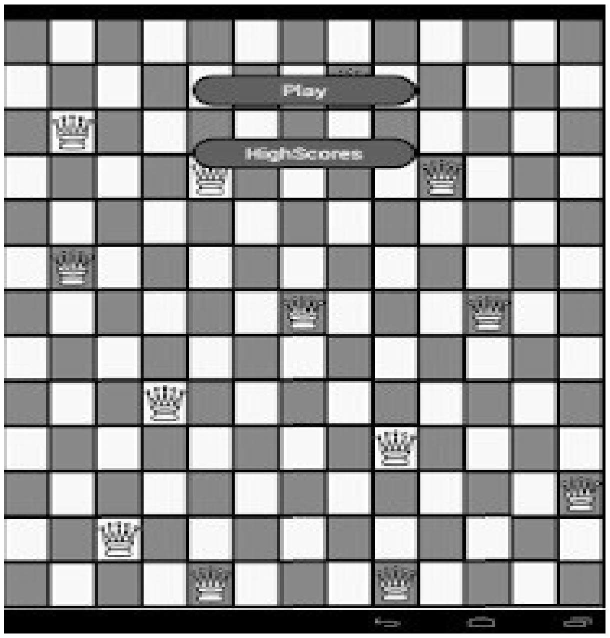


The App finds the 92 solutions from the 16 million variations where the eight queens don't attack each other on the 8x8 chess board. Finding the 92 right solutions with an up-to-date decent smart mobile phone lasts less than a half minute.



## N Queens.

Designed for Android systems, By ggwp.



Place N queens on a N x N chess board so that none of them attack the other. A generalised form of the 8 queens problem of placing 8 queens on a chess board. Can you do it in the minimum possible moves? Give it a shot!

This is a classical puzzle problem which is solved by backtracking. The algorithm used is quite easy.

**Features:** Play from levels 1 to 10 placing 4 to 13 queens.

**Applications Download Links:**

https://apps.apple.com/us/app/eight-queens-8-queens/id1317679393

https://play.google.com/store/apps/details?id=com.benchmark.eightqueens

https://play.google.com/store/apps/details?id=com.ggwp.nqueens&hl=en&gl=US&pli=1

## 4. literature reviews of academic publications relevant to the idea.

### 1. Different perspectives of the N-Queens problem

The N-Queens problem is a commonly used example in computer science. There are numerous approaches proposed to solve the problem. We introduce several definitions of the problem, and review some of the algorithms. We classify the algorithms for the N-Queens problem into 3 categories. The N-Queens problem can be defined aa follows: Place N queens on a N-by-N chessboard, one queen on each square, so that no queen capture any other, that is, the board configuration in which there exists at most one queen on the same row, column and diagonals. Figure.4 illustrates a solution to the 8-Queens problem. Solutions to the N-Queens problem can be represented by a permutation. For example, the solution given in Figure.4 can be represented by the following permutation: $\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 4 & 1 & 7 & 0 & 6 & 3 & 5 \end{bmatrix}$

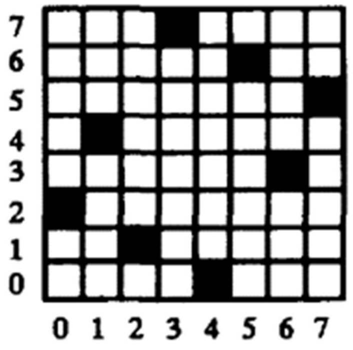


Fig.4

#### 1. ALGORITHMS GENERATING ALL SOLUTIONS

The objective of these algorithms is to find all the solutions for a specified N. Determining the exact number of solutions is a difficult problem, and so far, no formula has been found for the general case.

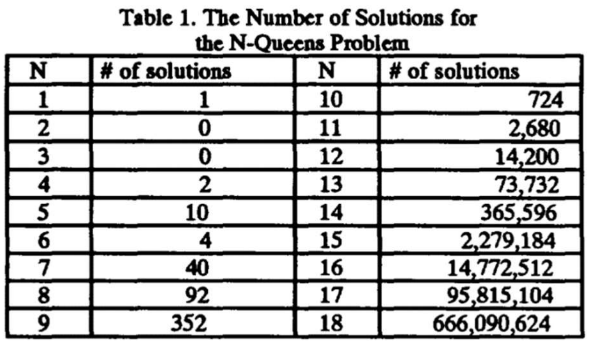


Table 1. The Number of Solutions for the N-Queens Problem

| N | # of solutions | N | # of solutions |
|---|---|---|---|
| 1 | 1 | 10 | 724 |
| 2 | 0 | 11 | 2,680 |
| 3 | 0 | 12 | 14,200 |
| 4 | 2 | 13 | 73,732 |
| 5 | 10 | 14 | 365,596 |
| 6 | 4 | 15 | 2,279,184 |
| 7 | 40 | 16 | 14,772,512 |
| 8 | 92 | 17 | 95,815,104 |
| 9 | 352 | 18 | 666,090,624 |

Table.1 gives the number of solutions vs. the number of queens, based on the empirical results.

Empirical observations indicate that the number of solutions increases exponentially.

a. Trial and Error Algorithms.

b. Backtracking Algorithms.

c. Permutation Generation Algorithms.

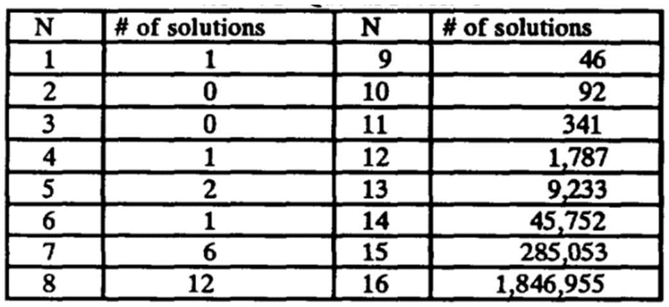
**a. The brute-force trial and error algorithms** generate all of the M possible vectors and test each of them according to the criterion function f. For the N-Queens problem, we have to place each queen on s separate row. Bute-force trial and error algorithms are among the most inefficient algorithms.

**b. Backtracking Algorithms**, the basic idea of backtracking algorithms is to build up the solution vector one component at a time and test it according to the criterion function to determine whether the vector being formed still has a chance of success. This algorithm can be considered as a tree traversal process.

**c. Permutation Generation Algorithms** are based on generating all of the permutations of N different elements. The brute-force trial and error algorithm allows a conflagration with more than one queen on each column before-testing the solution. All possible permutations of these N numbers are generated and tested-to see whether they are solutions to the problem or not. The complexity of this approach is O(N!).

## 2. ALGORITHMS GENERATING FUNDAMENTAL SOLUTIONS

The number of fundamental solutions and the number of N queens is given Table.2. As can be observed, the number of fundamental solutions increases exponentially.



| N | # of solutions | N | # of solutions |
|---|---|---|---|
| 1 | 1 | 9 | 46 |
| 2 | 0 | 10 | 92 |
| 3 | 0 | 11 | 341 |
| 4 | 1 | 12 | 1,787 |
| 5 | 2 | 13 | 9,233 |
| 6 | 1 | 14 | 45,752 |
| 7 | 6 | 15 | 285,053 |
| 8 | 12 | 16 | 1,846,955 |

Table.2

The following algorithms generate the fundamental solutions:

a. Generate and Test Algorithms

b. Partial Elimination of Symmetries (Naur's Algorithm)

c. An Approach with Group Properties ('Topor'sAlgorithm)

**a. Generate and Test Algorithms** use one of the algorithms generating all solutions, as the "generating' part of the algorithm. When a solution is generated, it is first tested to see whether it is fundamental or not. If it can be transformed to a previously found fundamental solution, it is discarded. Otherwise, it is added to the fundamental solutions list.

**b. Partial Elimination of Symmetries (Naur's Algorithm)** uses the central rows to eliminate some of the symmetries while for the other rows it still uses "generate and test" methods to discard the isomorphic solutions. This algorithm has original] y been written for 8-Queens. Although it is possible to generalize it for N-Queens, it is not significant to do so because the number of symmetries eliminated by the algorithm relatively decreases as the number of queens increases.

### c. An Approach with Group Properties (Topor's Algorithm)

This algorithm uses a direct method for generating fundamental solutions. Topor's algorithm utilizes the group property of the N-Queens problem to generate the fundamental solutions. The algorithm uses the concept of orbits. An orbit of a square under a group is defined as the set of squares to which the given square can be mapped by elements of the group.

## 3. ALGORITHMS GENERATING SOME SOLUTIONS

The algorithms in this category aim to find only one or several solutions to the problem.

a. Nondeterministic Algorithms,

b. Backtracking Algorithms,

c. Probabilistic Algorithms,

d. Divide-and-Conquer Algorithm.

**a. Nondeterministic algorithms** differ born their deterministic counterparts because of a special primitive called “choice”. This primitive is used to make a selection among a number of choices. It is possible to translate a nondeterministic algorithm to a backtracking algorithm. The specification of the nondeterministic algorithm can be stated as: Pick one square on each column, being careful not to pick two on the same row or diagonal.

**b. Backtracking Algorithms,** the general characteristics of backtracking algorithms have been discussed earlier. These algorithms can be modified to provide only one solution. These algorithms find the lexicographically first solution to the N-Queens problem. Empirical results show that the run-times for the backtracking algorithms which find only the first solution, increase exponentially.

**c. Probabilistic Algorithms**

This algorithm is able to generate a solution for extremely large values of N. In this algorithm, a random permutation generator is used to place the queens on the board. A permutation guarantees that no two queens are either on the same row or on the same column. This generally produces collisions along the diagonals. The gradient-based heuristic is applied to all possible pairs of queens until there is no collisions along the diagonals.

**d. Divide-and-Conquer Algorithms**

The general solution to the N-Queens problem is defined in terms of the concept of knight-walk and solution. If a solution to the N-SuperQueens problem starts with a queen in the upper left-hand comer and proceeds with a queen in the third column of the second row, and with a queen in the fifth column of the third row, and so on, this solution is called knight-walk.

## 4. CONSTRUCTIVE APPROACHES

there are several other attempts intended to solve the problem using constructive methods. The difference between the algorithmic and the constructive approaches is that the latter does not rely on search techniques but generate solutions using performed patterns. [2,18,11,23,8] are the examples in the literature to the constructive methods.

## 2. A Quantum-Inspired Differential Evolution Algorithm for Solving the N-Queens Problem Research paper.

The N-queens problem aims at placing N queens on an N x N chessboard, in such a way that no queen could capture any of the others. As the Classical methods are very limited in solving such problems because a large set of data is to be processed and a large solution space must be explored. So, other methods, known as approximate methods, are used to solve such problems

The paper presents a new approach for solving the NQP. A quantum-inspired differential evolution algorithm has been employed. The proposed algorithm is a novel hybridization between differential evolution algorithms and quantum computing principles approach to have a remarkable efficiency and good results. The first is the DEA that uses the differential evolution operator as alternative of classical GAs' mutation. The second is the QGA, proposed for the first time in the works of Han and which was spread in an exponential manner to touch many fields. A QGA has been used for solving the N-Queens problem and has given promising results. Compared with CGA and QGA, that hybrid algorithm has given better results either in term of computation time or in term of fitness evolution. The quantum-inspired differential evolution algorithm is well situated to be among the good alternatives to solve combinatorial optimization problems, especially in term of efficiency and algorithmic complexity. Conventional DEAs operate on a set of individuals (chromosomes) forming a population. To be more representative this population must contain a fit number of chromosomes. This makes the solution space very large. So, the classical DEAs are usually very expensive in terms of processing time and memory size. For reducing the number of chromosomes and consequently reducing the heavy computation time, an alternative is proposed: it is the quantum-inspired differential evolution algorithm.

During the whole process we keep in memory the global best solution. The algorithm consists of applying cyclically the following quantum inspired and differential evolution operations:

- The first operation is a quantum interference which allows a shift of each qubit in all the direction of the corresponding bit value in the best solution.

- The second operation consists of a quantum differential mutation which will perturb values of quantum chromosomes using a difference between two other quantum chromosomes. This difference is multiplied by another factor *F* before being added to the perturbed chromosome. In this work we have chosen *F=1/1000*. This operator replaces the quantum-inspired crossover and the quantum-inspired mutations because it has been proved that only the use of differential mutation in a conventional DEA do produce the diversity and the convergence at the same time. So, in this work we will only use the quantum differential mutation.

Finally, we perform a selection of $m$ chromosomes among the $n$ existing in the current generation. For this, we apply first a measurement on each chromosome to have from it one solution among all those present in superposition. But unlike pure quantum systems, the measurement here does not destroy the states' superposition. Since our algorithm operates on conventional computer and does not require the presence of a quantum machine, it is possible and, in our interest, to keep all the possible solutions in the superposition for the next iterations. For each measurement result, we extract a distribution of N-queens. To evaluate the quality of a solution, we compute its fitness. The best solution is that having the minimal fitness (0 in the optimum).

## 3. Comparative Study of Different Algorithms to Solve N-Queens Problem.

This Paper provides a brief description of the Genetic Algorithm (GA), the Simulated Annealing (SA) Algorithm, the Backtracking (BT) Algorithm and the Brute Force (BF) Search Algorithm and attempts to explain the way as how each of them can be employed in finding the best solution of N Queens Problem and makes a comparison between these four algorithms.

**Genetic Algorithm** is an optimization algorithm follows the concept of natural selection. It is a probabilistic search method, based on the ideas of evolution. It follows the Darwinian's Survival of the Fittest Principle. Unlike the traditional techniques, it is suitable for many real-world problems, in which the goal is to find optimal solution. It searches among a group of points, rather than a single point; and operates with a coding of parameter set, not the parameters themselves. Genetic Algorithm can reach to a solution, close to the global optima. It has advantages over traditional algorithms, which cannot always achieve a global or close to global optima. But it does not always guarantee optimal solution, because of its randomness.

**Applying the Genetic Algorithm on N-Queens problem:**

**Input:** A Population (Group) of Solutions (Chromosomes), each representing the placement of N number of Queens in (N*N) Chessboard.
**Output:** The Optimal Solution (Fitness), representing the placement of N number of Queens in the (N*N) Chessboard according to the rule of the N Queens Problem.

```
Step 1: Randomly generate the Initial Population of Queens (Parent Pool).
Step 2: Evaluate the chromosomes of the Parent Pool as:
2.1: Fitness function is the no. of non-attacking queen pairs=nc2 (maximum value).
2.2: Calculate the fitness value of each of the chromosomes as: ((nc2) - the no. of diagonal conflicts).
Step 3: Find the best fitness' chromosome of the Parent Pool.
 Loop (generation)
Step 4: Perform Tournament Selection as the selection procedure.
```

Step 5: Now, generate offspring (Child Pool) by performing Two Point Crossover Operation.
Step 6: Mutate the offsprings by replacing the duplicate bits of the Child Pool chromosomes by
the unused ones.

Step 7: Perform Elitism operation as:
7.1: Copy the Child Pool to the Parent Pool.
7.2: Find the best fitness' chromosome of the New Parent Pool.
7.3: Compare the fitnesses of the best fitness' chromosomes of the Parent Pool and the
New Parent Pool.
7.4: The Better Fitness' Chromosome is copied in the 0th location of the New Parent
Pool.
7.5: If their fitnesses are equal, then the best fitness' chromosome of the New Parent
Pool is copied in the 0th location of the New Parent Pool.
End Loop (generation).

**BACKTRACKING**
It is a very general algorithm that uses the concept of partial candidate solution and does a quick test to check if it can be ended to a proper solution or not. When it is applicable, backtracking is very much quicker than brute force search, as it can eliminate large number of candidates with just a single test. It is a very much important tool for solving problems like Crosswords, Sudoku, and many kinds of puzzles etc. It is not a specific algorithm – although, unlike many other non-specific algorithms, it is guaranteed to find all possible solutions of a problem in a comparatively less amount of time.

**Applying BACKTRACKING Algorithm on N-Queens problem:**

**Input:** The number of Queens (N).
**Output:** The Number of Solutions (Placements) of that very number of Queens' Problem,
according to the rule of the problem.

Step 1: Firstly, place a queen in the top row and take a note of the column and diagonals it occupies.
Step 2: Now, place another queen in the next row, such that, it is not in the same column
or diagonal with the first one. Keep note of the occupied columns and diagonals and
proceed to the next row to place the next queen.
Step 3: If no position is available open in the next row, get back to the previous row and place that queen to the next available position in its row and the process starts over again, until finding the correct arrangements for all the queens

**Simulated Annealing**

(SA) is a probabilistic search and optimization method. It is very much used when the search space is discrete. It follows “Annealing”, which is a process, in which metals are cooled gradually to make them reach a state of less energy, where they are very much strong. Simulated Annealing technique is a meta-heuristic optimization algorithm.

**Brute Force Search**, which is also known as Generate and Test, examines all possible candidates for the solution and checks, whether each candidate fulfils the problem's criteria or not. The basic idea of the brute force search algorithm for N-Queen’s problem is to place each of the n queens on all possible positions and check regularly, whether the queens threaten each other or not. If this does not occur, then it has reached a proper solution. Its complexity grows very quickly, as increment occurs in the problem size. So, it is mostly used when the problem size is relatively small.

From the Results, the Proposed Genetic Algorithm (GA) performed better than the Proposed Simulated Annealing (SA) Algorithm using GA, the Backtracking (BT) Algorithm and the Brute Force (BF) Search Algorithm and it also provided better fitness value (solution) than the Proposed Simulated Annealing Algorithm (SA) using GA, the Backtracking (BT) Algorithm and the Brute Force (BF) Search Algorithm, for different N values. Also, it was noticed that the Proposed GA took more time to provide result than the Proposed SA using GA.

## *4.* An Adaptive Genetic Algorithm for Solving N-Queens Problem.

*N* Queens Problem is such an optimization problem attributed to the class of NP-Complete Problems. The goal of *N* Queens Problem is to suitably place *N* number of Queens on an *N* x *N* chessboard in a way that there is no conflict between them due to the arrangement, that is, there is no intersection between them vertically or horizontally or diagonally. Eight Queens Problem is a just a special case of *N* Queens Problem, where *N* is equal to 8. Jordan B. and Brett S. [12] considered extensions of the *N* Queens Problem including various board topologies and dimensions at the same time surveying already known solutions for the *N* Queens Problem of placing *N* Queens on an *N* x *N* chessboard. Time complexity of an *N*-Queen problem is O(*n*!). Here, we are proposing a heuristic approach to obtain the best solutions for this problem.

Heuristic approaches are required to solve *N* Queens Problem in real time with optimal solutions. Genetic algorithms (GA) are one such powerful heuristic method which is capable of efficiently solve the problem in real time by virtue of its extensively developed exploration and exploitation properties. In this paper we propose a modified and state-of-the-art Genetic Algorithm with a novel, efficient and robust Fitness Function for generating the best solution for *N*-Queens Problem with different random initial candidate solutions and calculating and comparing the fitness values for each of the trial solutions.

Genetic Algorithm is a widely accepted popular Evolutionary Optimization technique that imitates the process of natural selection. Firstly, in Genetic Algorithm a population of individual chromosomes is selected randomly. These chromosomes are better known as phenotypes in terms of genetics each of which has its individual set of characteristics or traits widely known as genotypes. These genotypes can be altered by adopting various strategies including mutation and crossover. These chromosomes are evolved towards better (eventually optimal) solutions for a combinatorial optimization problem. This optimal solution is dependent on the termination criterion. Genetic Algorithm gives the optimal solution depending on the nature of the fitness function and on the structure of the algorithm used. Here, Genetic Algorithm is used to solve the *N* Queens Problem. A general outline how Genetic Algorithm (GA) works is given below:

1. A random population of candidate solutions is created and the fitness scores of the individuals are calculated, and the chromosomes are sorted in the population and ranked according to the fitness values.

2. Certain number of chromosomes will pass onto the next generation depending on a selection operator, favouring the better individuals based on their ranking in the population.
3. Selected chromosomes acting as parents will take part in crossover operation to create children whose fitness values are to be calculated simultaneously. Crossover probability is generally kept high, because it is seen to give better children in terms of fitness values.
4. Then based on a mutation probability, a mutation operator is applied on new individuals which randomly changes few chromosomes. Mutation probability is generally kept low.
5. Evaluated off-springs, together with their parents form the population for the next generation.

Steps 2-5 are repeated until a given number of iterations, that is, generations have been evaluated or solution improvement rate falls below some threshold, that is, the difference between two best solutions from consecutive generations is below a certain tolerance limit.

Genetic Algorithms were framed as heuristics for solving problems with good and bad solutions depending on the fitness values of the solutions, yet here they proved their ability to solve combinatorial optimization problems with simple binary (yes and no) answers. In the previous literatures, attempts were made to solve *N*-Queens Problems with Genetic Algorithms. In this paper with modifications in the fitness functions, hereby giving rise to a novel modified GA, it is seen that this approach yields promising and satisfactory results in less time compared to that obtained from the previous approaches.

## 5. A New Solution for N-Queens Problem using Blind approaches.

In the body of literature, various research is done to overcome n-queens problem. For example, Sosik and Gu proposed a novel local search algorithm with conflict minimization to solve the n-queens problem. This algorithm is significantly faster than any backtracking search algorithm and can solve the n-queens problem in linear time.

Backtracking algorithms offers a very limited class of resolutions for great size boards. Because it is difficult for a backtracking search to detect resolutions that are separate in the solution space considerably. Reference [4] presents probabilistic algorithm utilizes gradient-based heuristic search. This algorithm is able to generate a solution for extremely large values of n. In this algorithm, a random permutation generator is used to place the queens on the board and reduce the number of collisions by swapping a pair of queens.

The results indicate that the number of permutations necessary to build a resolution is usually very small. Also, reference defines perspective of n-queens problem and presents the classification of applied algorithms for the n-queens problem into three categorizations. Manzuik implemented a binary Hopfield neural network (HNN) in the asynchronous mode as well as in the synchronous to solving n-queens problem. The HNN is a simple artificial neural (ANN) network. It can save specified patterns in a manner similar to the brain in that the complete pattern for a given problem can be recovered if the network is indicated with only regional information. The results show that convergence rate is up to 100% and the experimental predicate of the average computational complexity is polynomial. Various models of ANN such as HNN are able to adapt and learn from input information in optimization problems such the n-queens problem and many more. Also, Shagrir proposed a new algorithm by means of AI techniques based on HNN. In this HNN, every unit (neurons) is permissive to prevent itself. The results indicate that the HNN never continually oscillates but relaxes into a resolution in polynomial time and it solves the n-queens problem regardless of the dimension n or the initialized values. In addition, in reference [8], has been presented a new algorithm by global parallel genetic algorithm to solve n-queens problem. Also, reference [9] presents a new probabilistic algorithm that it is based on gradient based heuristic. It is capable to find a resolution for millions of queens and runs in polynomial time with a local search.

## ▪ Research publications and resources Citations.

1. Cengiz Erbas, Seyed Sarkeshik, and Murat M. Tanik. 1992. **Different perspectives of the N-Queens problem**. In Proceedings of the 1992 ACM annual conference on Communications (CSC '92). Association for Computing Machinery, New York, NY, USA, 99–108. https://doi.org/10.1145/131214.131227

2. Draa, Amer & Meshoul, Souham & Talbi, Hichem & Batouche, M.. (2010). **A Quantum-Inspired Differential Evolution Algorithm for Solving the N-Queens Problem**. Int. Arab J. Inf. Technol.. 7. 21-27. https://www.researchgate.net/publication/220413939_A_Quantum-Inspired_Differential_Evolution_Algorithm_for_Solving_the_N-Queens_Problem

3. Mukherjee, S., Datta, S., Chanda, P. B., & Pathak, P. (2015). **Comparative Study of Different Algorithms to Solve N Queens Problem**. International Journal in Foundations of Computer Science & Technology, 5(2), 15–27. https://doi.org/10.5121/IJFCST.2015.5202 https://www.academia.edu/11844318/COMPARATIVE_STUDY_OF_DIFFERENT_ALGORITHMS_TO_SOLVE_N_QUEENS_PROBLEM

4. Nag, Sayan & Sarkar, Uddalok. (2017). **An Adaptive Genetic Algorithm for Solving N-Queens Problem**. 10.17605/OSF.IO/ZP7ND. https://arxiv.org/pdf/1802.02006

5. SoleimanianGharehchopogh, F., Seyyedi, B.M., & Feyzipour, G. (2012). **A New Solution for N-Queens Problem using Blind Approaches: DFS and BFS Algorithms**. *International Journal of Computer Applications, 53*, 45-48. https://www.semanticscholar.org/paper/A-New-Solution-for-N-Queens-Problem-using-Blind-DFS-SoleimanianGharehchopogh-Seyyedi/1b2fe8de4fb733b236783537650c9adf252143eb

6. **A survey of known results and research areas for n-queens**. Jordan Bell∗ , Brett Stevens School of Mathematics and Statistics, Carleton University, 1125 Colonel By Drive, Ottawa, Ontario, K1S 5B6, Canada Received 14 September 2005; received in revised form 28 November 2007; accepted 13 December 2007 Available online 4 March 2008 https://www.sciencedirect.com/science/article/pii/S0012365X07010394?via%3Dihub

7. San Segundo, Pablo. (2011). **New decision rules for exact search in N-Queens**. Journal of Global Optimization. 51. 497-514. 10.1007/s10898-011-9653-x. https://www.researchgate.net/publication/220249601_New_decision_rules_for_exact_search_in_N-Queens

8. Rivin, I., Vardi, I., & Zimmerman, P. (1994**). The n-Queens Problem**. The American Mathematical Monthly, 101(7), 629–639. https://doi.org/10.2307/2974691 http://www.jstor.org/stable/2974691

9. H. S. Stone and J. M. Stone, "**Efficient search techniques—An empirical study of the N-Queens Problem,**" in IBM Journal of Research and Development, vol. 31, no. 4, pp. 464-474, July 1987, doi: 10.1147/rd.314.0464. https://ieeexplore.ieee.org/document/5390100

## II. Proposed Solution.

### 5. Main Functionalities and Use Case Description.

- **Main functionalities:**

**[User]**

1. The user should select the number of Queens (N).
2. The user could show the solution on the board.

**[System]**

1. The system should check on the number entered whether it's valid or not
2. The system should Make the board based on the entered number of Queens (N X N) and solves the problem.

- **Use Case Description:**

***1. Name*:** select number of Queens (N).

***Initiator*:** User.

***Goal*:** Select the number of queens based on which the board is made.

***Description*:** The user enters a valid number of Queens he wants to play with, and this number must be an integer bigger than or equal 4.

***Pre-Condition*:** The user open the application and the system loads the interface of the game.

**Main Success Scenario:**

The user enters the number of Queens (N) correctly and the system shows the board and the solution.

**Alternative Scenarios:**

1. The user entered an integer number less than 4.
2. The user entered a non-inter, stung value in the field.
3. The user didn't select any numbers or entered space" ".

***2. Name*****:** Make the board

***Initiator*****:** System

***Goal*****:** Making the board based on the number of Queens entered.

***Description*****:** The system initializes and make the board of the solution o that no two queens attack each other.

***Pre-Condition*****:** The user should have entered a valid integer number bigger than or equal 4.

**Main Success Scenario:** The system should make the board and solve the problem.

---------------------------------------------

***3. Name*****:** Show the solution on board

***Initiator*****:** User

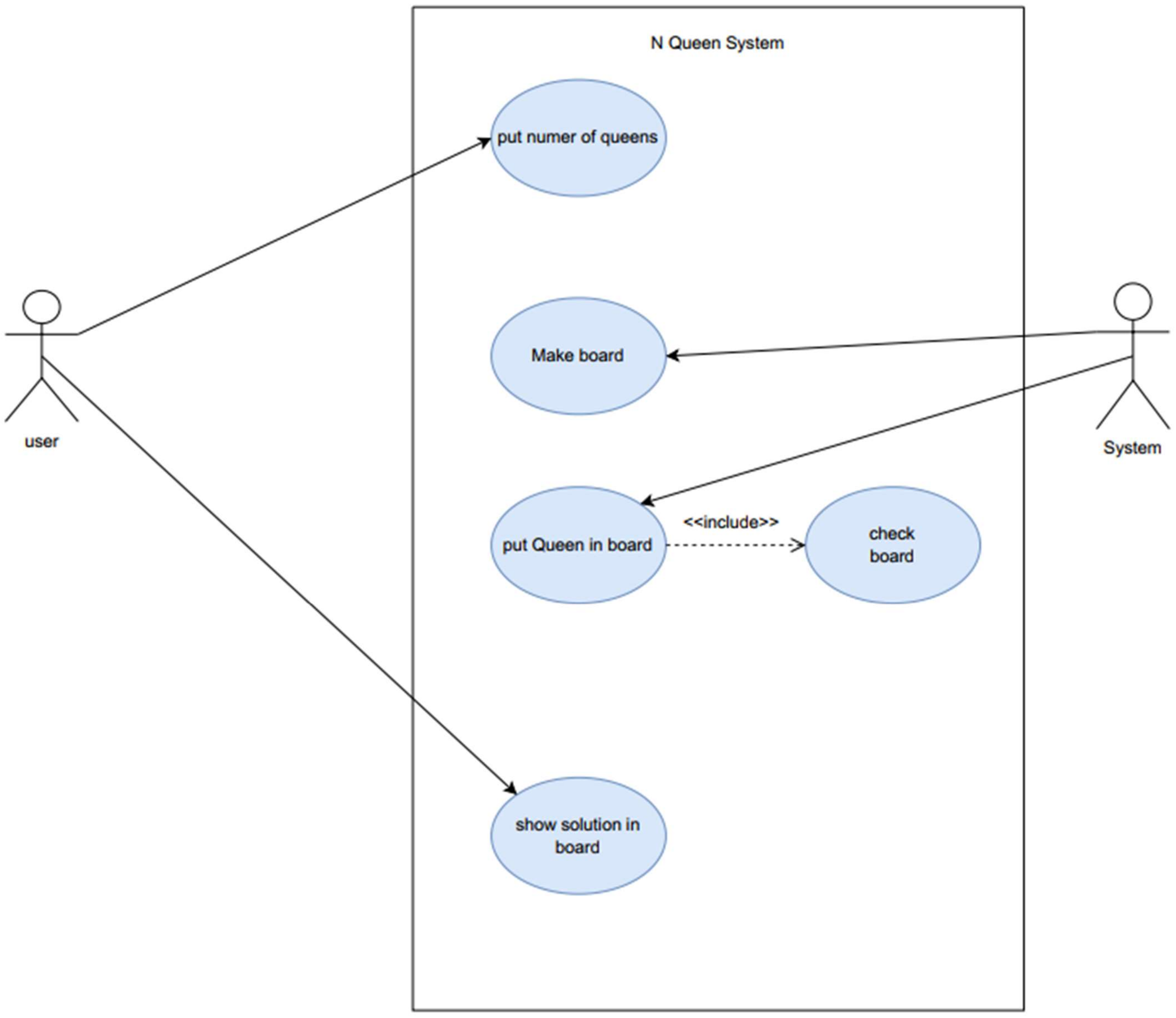
***Goal*****:** Showing the solution of the N-Queens on the board.

***Description*****:** The user should ask to represent the solution of the problem on the created board.

***Pre-Condition*****:** The system should have solved the problem.

**Main Success Scenario:** The solution of the N-Queens problem is shown on the board.

## 6. Use Case Diagram Representation



## III. Applied Algorithms.

### 7. Details of the used algorithm (Differential Evolution)

Differential evolution algorithms are population-based algorithms like genetic algorithms using similar operators, crossover, mutation and selection. The main difference is that genetic algorithms give more importance to the crossover operation while DEAs consider the mutation operation as the most important one. The DEA's main operation is based on the differences of randomly sampled pairs of solutions in the population

The main evolutionary operator in DE is meaningfully different from other evolutionary algorithms since mutation is neither based on the alteration of genes by using a mutation probability nor rest on the use of a defined probability distribution function. The general structure of a differential evolution algorithm is shown in Figure 5.

```
Generate random solutions that cover the given space.
Evaluate each solution.

g=1
While (convergence is not reached)
   For i=1 to Population size
         Apply differential mutation.
         Execute differential crossover
         Clip the solutions if necessary
         Evaluate the new solution.
         Apply differential selection.
   End

   g=g+1;
End
```

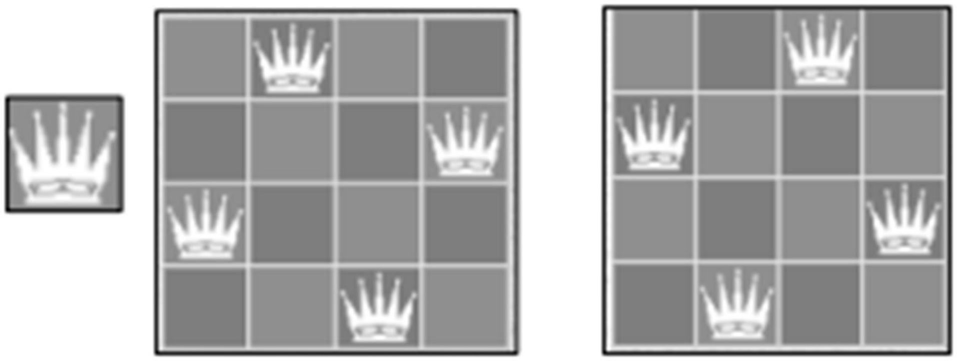
Fig.5

As in other evolutionary algorithms, the crossover operator is introduced in order to increase the diversity of the population.

#### Solution Modelling

Every queen on a checker square can reach the other squares that are located on the same horizontal, vertical, and diagonal line. So there can be at most one queen at each horizontal line, at most one queen at each vertical line, and at most one queen at each of the 4n-2 diagonal lines. Furthermore, since we want to place as many queens as possible, namely exactly n queens, there must be exactly one queen at each horizontal line and at each vertical line [3].

For a 1x1 board, there is one trivial solution: for 2x2 and 3x3 boards, there are no solutions. For a 4x4 board, there are two:



These are considered distinct solutions, even though the solutions are mirror images of each other. There is 24 The International Arab Journal of Information Technology, Vol. 7, No. 1, January 2010 no quick and easy way to calculate the total number of NQP solutions for an NxN board. We can represent the NQP solution by an NxN matrix A containing only N ones and satisfying the constraint that only one 1 can be in a raw, in a column or in a diagonal. For example the matrix bellow represents a solution of the 4-qeens problem:

$$\begin{array}{cccc} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{array}$$

Figure 6. A solution matrix of the 4-queens problem

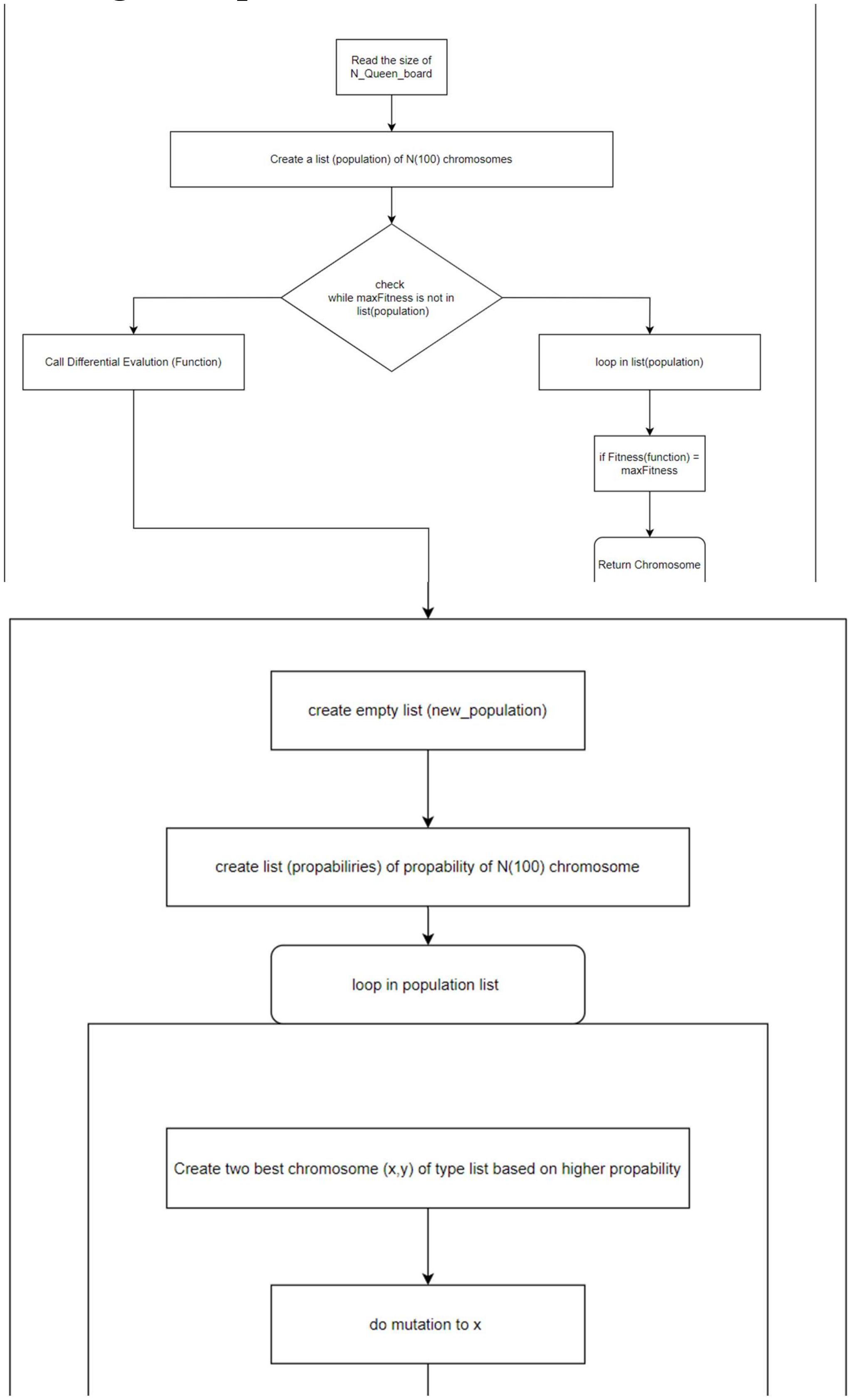
## Fitness Function

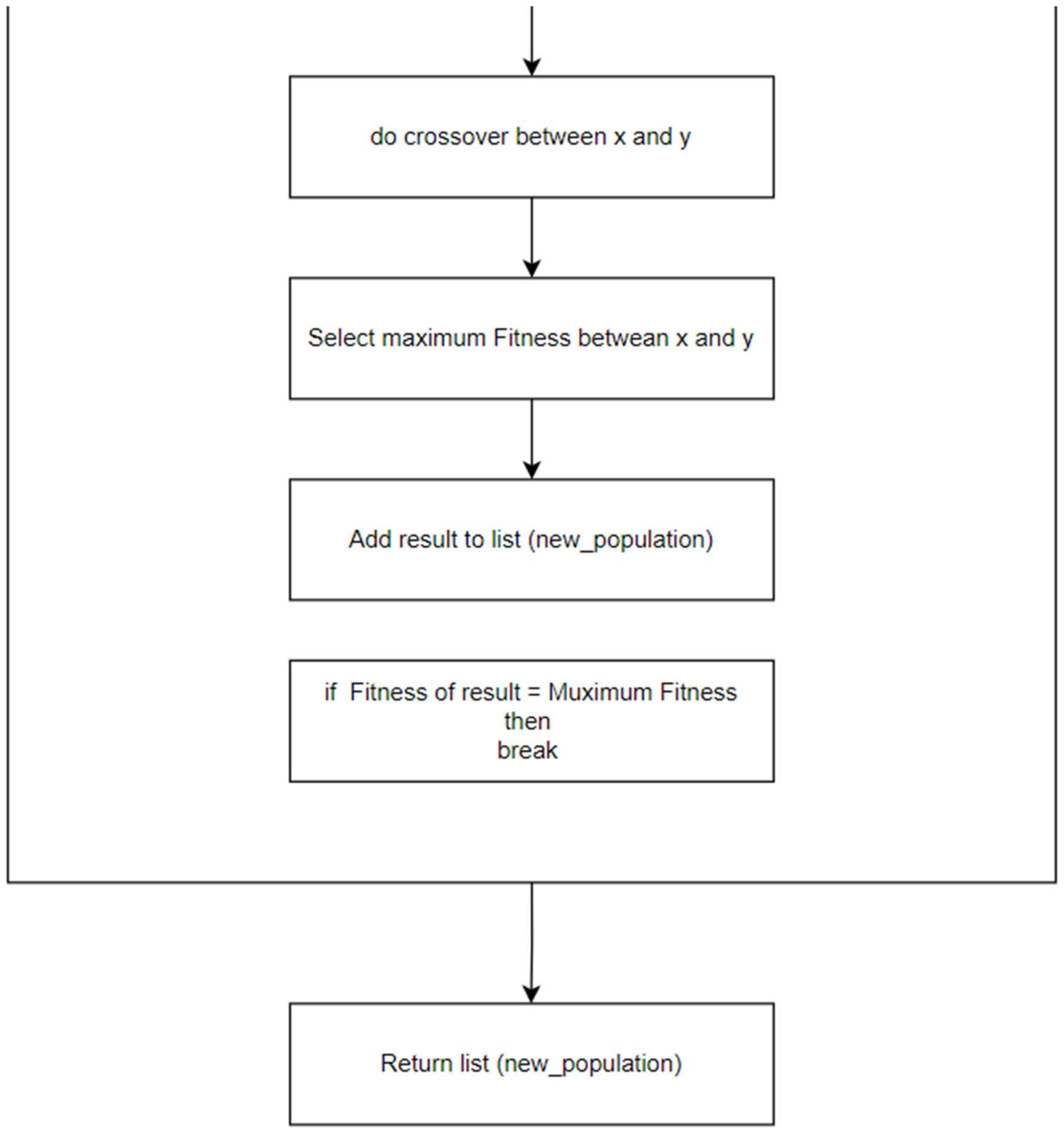
The penalty of one queen is equal to the number of queens she can check. The fitness of the configuration is equal to the sum of all the queens penalties divided by two, deleting redundancy counting. For example, the fitness of the solution presented in Figure 3 is 0: we can easily observe that each queen in this chessboard is the only one in all the lines passing through it; horizontal, diagonal or vertical. In the matrix solution of Figure 4, the fitness is 8: the penalty of the queen in column 1 is 3: it can check the second, the third and the fourth queen in the same line, the second column's queen has a penalty of 4: it can check the 3 other queens of the same line and that situated in the fourth column of the third line. The third queen can check the queens of the same line, its penalty is 3. Similarly, to the second queen in its line, the fourth queen of that first line can check the three other queens in this line and that in the fourth column in the third line. It has a penalty of 4. Finally, the queen in the third line of column 4 can check 2 other queens. So, its penalty is 2. The sum of these penalties is equal to 16, and when dividing it by 2 to reduce redundancy, we will have 8 which is the fitness of this matrix solution

**The applied Deferential Evolution Algorithm** for our N-Queen problem solver is:

```
read size of board
create a list (population) of 100 random chromosome
check
while maxFitness not in list(population)
        call Differential Evolution (function)
            create empty list(new_population)
            create list (probabilities) of prop of 100 chromosome
            for size of population
                    create two best chromosomes (x, y) of type list based on higher propabilty
                    do mutation(function) to x
                    do crossover between x and y
                    select maximum fitness between x and y
                    add result to list(new_population)
                    if fitness of result = maximum fitness then break
                            return list(new_population)
                            else
                            loop in list(population)
    if fitness(function) = maxFitness then
            return chromosome
```

## 8.Block Diagram Representation


Read the size of N_Queen_board
Create a list (population) of N(100) chromosomes
check while maxFitness is not in list(population)
Call Differential Evalution (Function)
loop in list(population)
if Fitness(function) = maxFitness
Return Chromosome
create empty list (new_population)
create list (propabiliries) of propability of N(100) chromosome
loop in population list
Create two best chromosome (x,y) of type list based on higher propability
do mutation to x


do crossover between x and y
Select maximum Fitness betwean x and y
Add result to list (new_population)
if Fitness of result = Muximum Fitness then break
Return list (new_population)

# IV. Tools, Discussion & Results.

## 9. Development platform.

**<u>Programming languages used:</u> Python.**

**<u>IDEs and tools:</u> PyCharm & Visual Studio.**

**<u>Libraries used:</u>**

### 1. Tkinter:

Out of all the GUI methods, Tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter is the fastest and easiest way to create the GUI applications.

Creating a GUI using tkinter is an easy task. provides classes which allow the display, positioning, and control of widgets. Properties of the widgets are specified with keyword arguments.

MessageBox Widget is used to display the message boxes in the python applications. This module is used to display a message using provides a number of functions.

### 2. Random:

In Python, random numbers are not generated implicitly; therefore, it provides a random module in order to generate random numbers explicitly. random module in Python is used to create random numbers. It is one of the most extensively tested generators in existence.

The random() method is implemented in C, executes in a single Python step, and is, therefore, threadsafe.

## 10. Results and Samples of the Output and GUI

The application is tested and applied many times with different number of Queens (N). And it could achieve all the requirements with high level of accuracy and precision and the solution obeyed all the constraints and differential evolution algorithm.

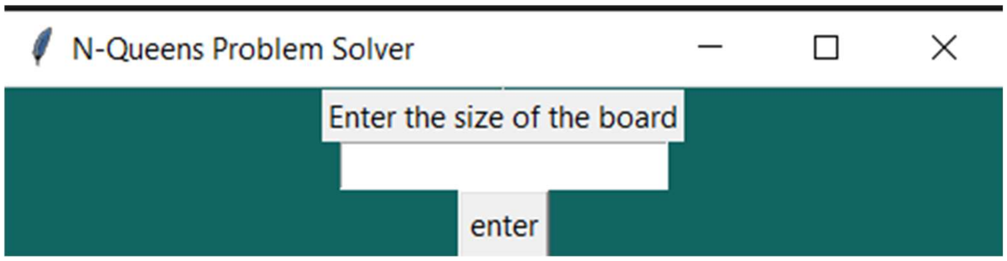
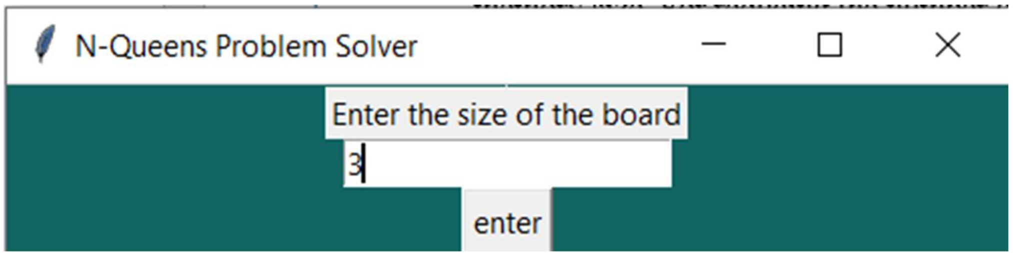
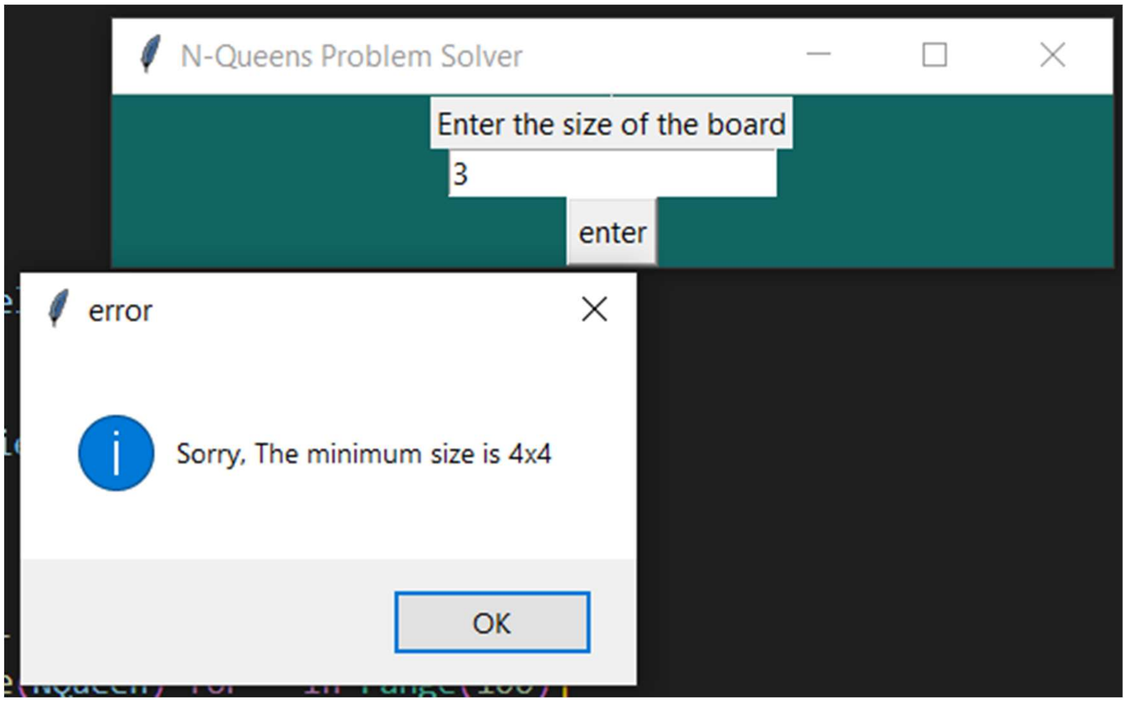


**Fig.7 represents the starting page of the application.**

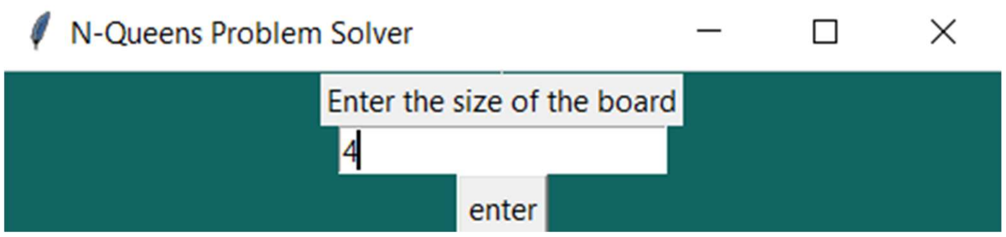
**First test: The user enters number of queens = 3**

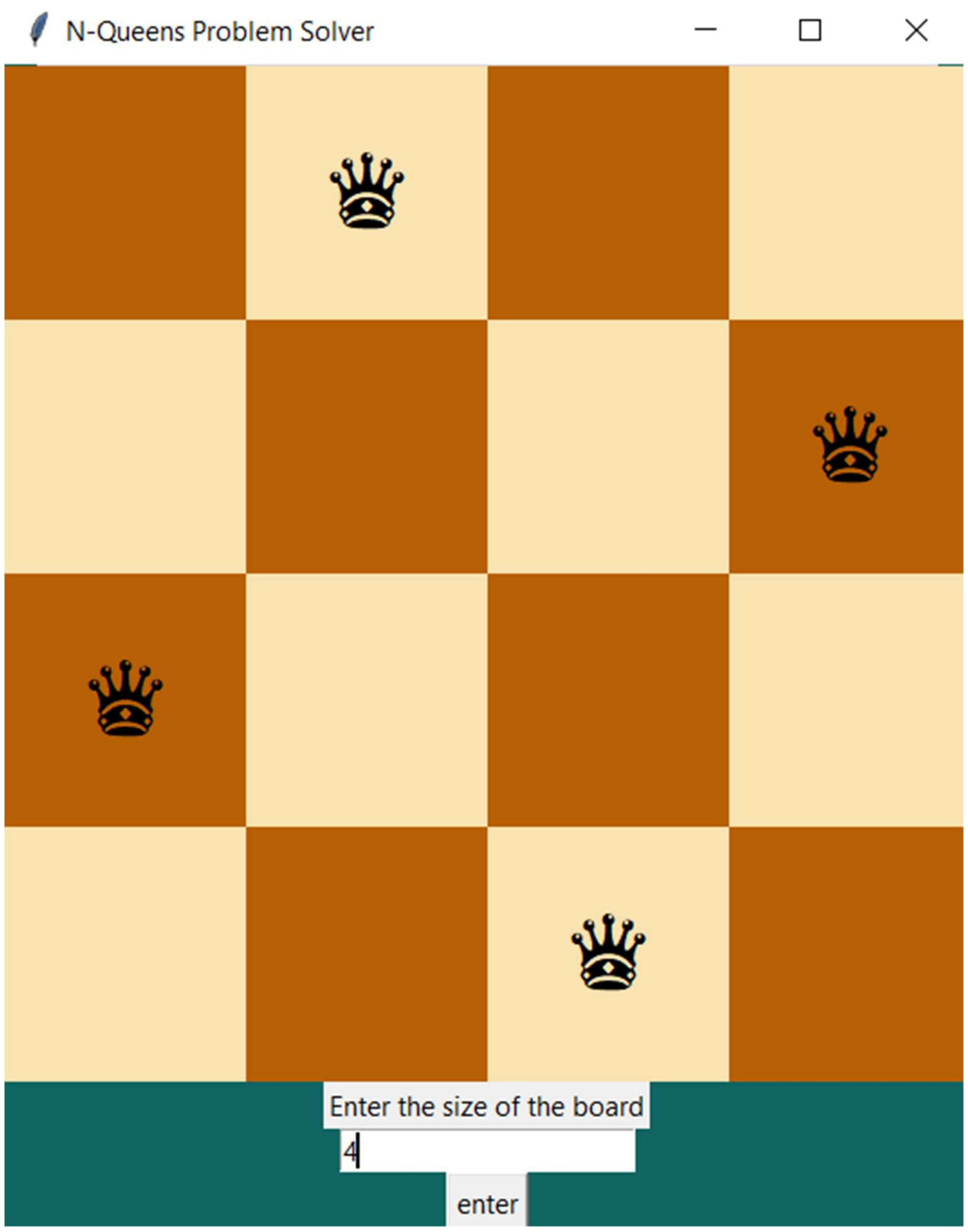




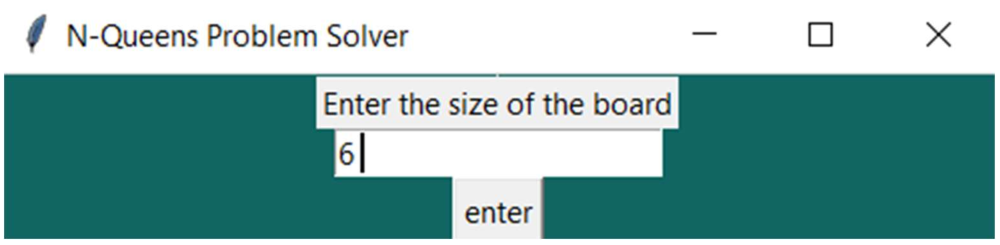
Sorry, The minimum size is 4x4

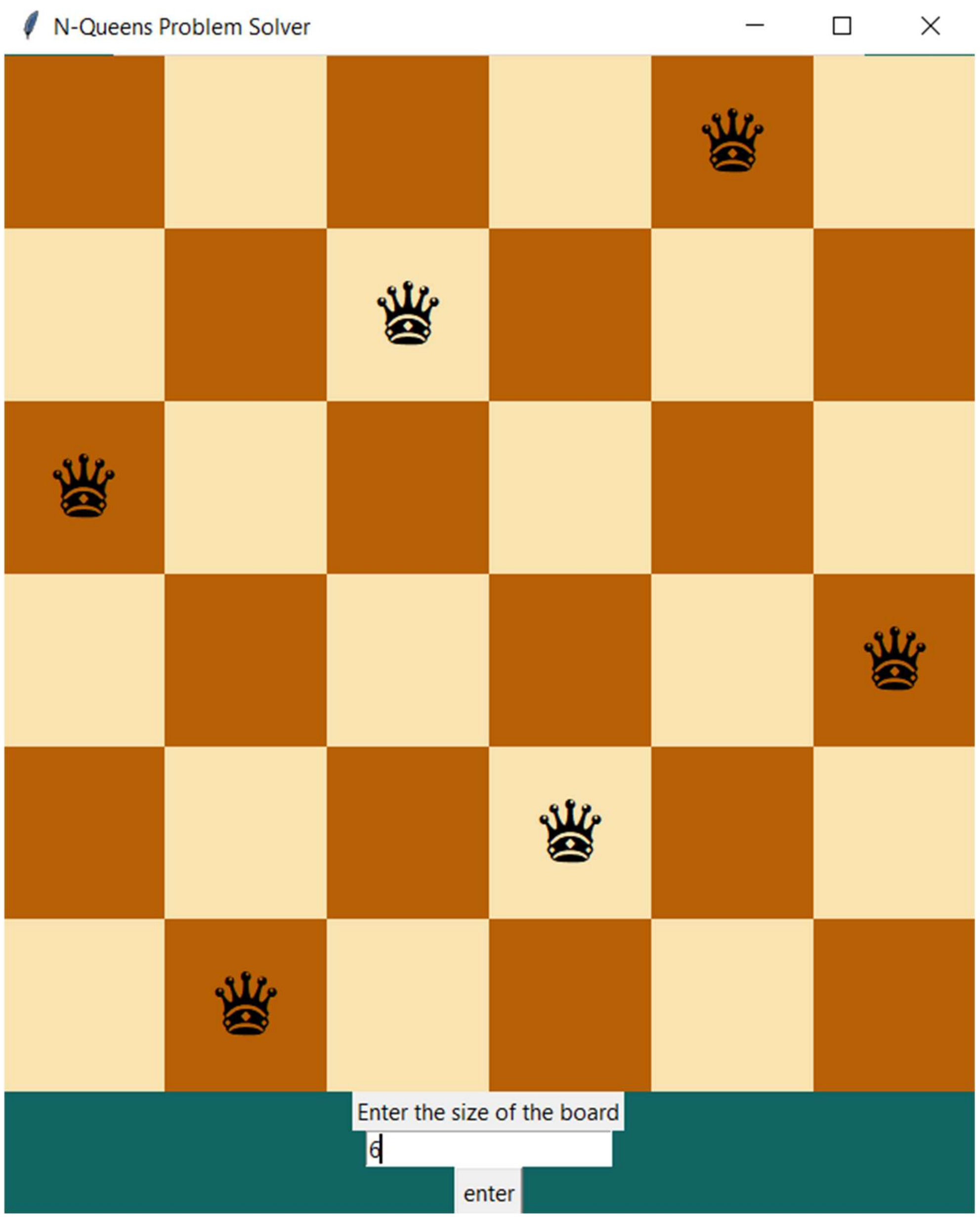
**Second test: The user enters number of queens = 4.**





**Third test: The user enters number of queens = 6.**





## V. Future Work.

Conventional DEAs operate on a set of individuals (chromosomes) forming a population. To be more representative this population must contain a fit number of chromosomes. This makes the solution space very large. So, the classical DEAs are usually very expensive in terms of processing time and memory size. For reducing the number of chromosomes and consequently reducing the heavy computation time, an alternative is proposed: it is the quantum-inspired differential evolution algorithm. A QDEA is a QDA with quantum coding solutions. This representation will reduce the computation time by decreasing the number of chromosomes. Moreover it gives a better global solution. Like in DEA, initial solutions are encoded in N chromosomes representing the initial population. The difference in a QDEA is that each chromosome does not encode only one solution but all the possible solutions by putting them within a superposition.