Tutorial 5: Genetic algorithms

Tutorial objectives:

- **To test your understanding of genetic algorithms**
- **To get a sense of how GAs work in a relatively simple scenario before having to implement a GA in a more complicated, multi-agent environment for Assignment 2**

Stochastic search concepts

1. What's a *stochastic search* and why is it sometimes a better option for solving a search problem than an *exhaustive search*?

2. How does the *stochasticity* come about in a Genetic Algorithm (GA)?

3. What's the role the chromosome in GA from the state space search point of view?

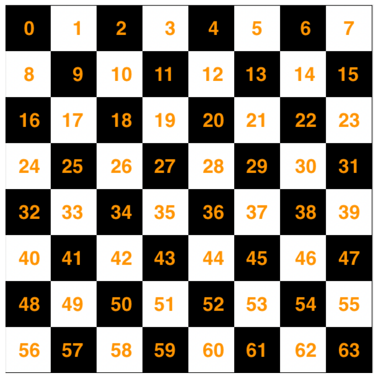
4. Why do we include *mutation* in an implementation of GA given the cross-over process on a computer can be implemented so that it doesn't make accidental mistakes?

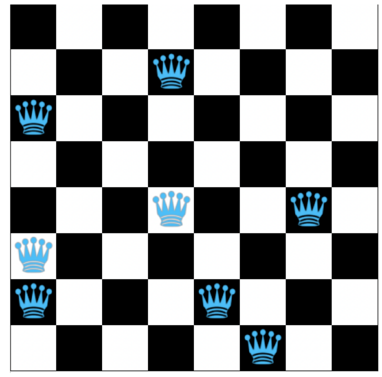
5. Describe and contrast two parent selection methods from the point of view of diversity preservation in the GA population.

GA

Exercise 1: Implement a genetic algorithm to solve the 8-queens problem from Lecture 6. Use the chromosome encoding scheme from the lecture, but on an 8x8 board with 8 queens. This means your chromosome will be a list of 8 integers with values between 0 and 63, which index the locations of 8 queens on 64-square board.

The figure below shows the correspondence of the indexes to locations on the board (on the left) and visualisation of the configuration of the chromosome [[11, 35, 48, 38, 16, 40, 61, 52] (on the right).





Implement whichever parent selection method you prefer. Follow the lecture notes for hints on how to do crossover and mutation. The provided `chess_board.py` and `queen.png` (downloadable from Blackboard) allow you to visualise the positions of the eight queens on the chessboard for a given chromosome. If you place these files in your project folder, you can use visualisations in your scripts by doing:

In order to further reduce the coding workload and let you quickly play with the dynamics of the genetic algorithm, `chess_board.py` implements a method called `nonattacking_pairs`, which takes the chromosome (assumed to be a list of 8 integers) as an argument and returns the number of nonattacking pairs in the board configurations dictated by that chromosome. The lowest score of the fitness function is 0 *nontaking* pairs of queens, and the highest score is 28 *nontaking* pairs of queens (meaning none of the 8 queens can take any of the other 7 queens).

Here's a rough Python-based pseudo-code for your genetic algorithm including the examples for how to use visualisations and the provided fitness function.

```
from chess_board import chess_board

#Initialise the board
board = chess_board()

# Init your population with randomly chosen (valid) chromosomes.

# Iterate over number of generations...

    # Iterate over number of individuals in your population...

        # Evaluate fitness of each individual;
        # let's assume c is the chromosome
        # (an 8-item list of integer values between
        # 0 and 63) of an individual.
        f = board.nonattacking_pairs(c)
```

```
# Pick the best individual and show the
# corresponding board; lets assume c_best
# is the chromosome of the best individual
# in the population.
board.show_state(c_best)

# Check if fitness of the best individual is
# 28...if so, you're done and the shown
# board is the solution to 8-queens problem.
# If you're not done continue with creation
# of new generation.


# For each new child...

    # Select two parents based on fitness.

    # Cross over parents to create the new child.

    # Decide at random if there is going to be a mutation.

# Replace the current population with the new children.
```