Algorithm

My solution uses an A* algorithm to find safe board states, with the number of threatened queens used as the heuristic.

Two versions of the algorithm are included: astar(args) which calculates a single solution, and returns all moves needed to reach it, and astar_multi(args) which calculates all solutions and returns the final node of each solution.

The data specifying which queen is fixed is stored in constants at the start of the file.

The A* functions take a list of tuple coordinates (in range 0 to 7), representing the starting positions of the eight queens. Then an initial node is created and the loop begins.

The algorithm loops while there are unvisited nodes available and the solution (or solutions) have not been found. It takes a node, checks if it is a solution, then generates child nodes for each possible move each mobile queen can make from the current state, and adds these to the open list to visit.

Complexity & Optimisation

Since the starting randomisation places one queen in each column, and any solutions also need one queen per column, we were able to optimise by only considering vertical moves for each queen, reducing the number of children needed, compared to also generating horizontals and diagonals.

With this we were able to have a time complexity of $O(n^2)$, where n is both the number of queens and dimension of the gameboard, since each mobile queen visits each position in its row.