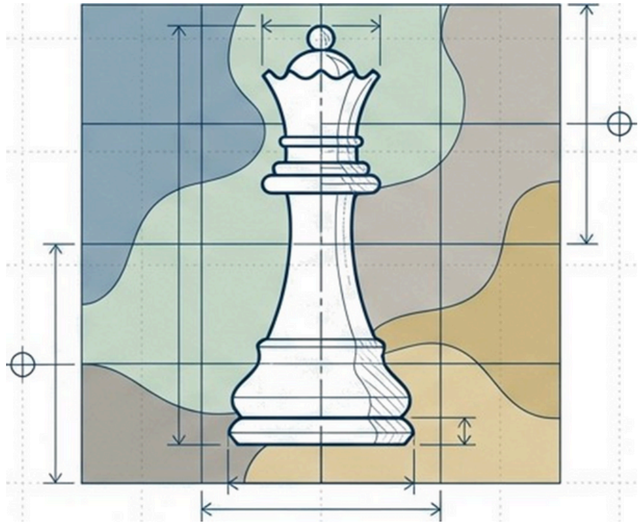
GROUP 05



# QUEENS

## GREEDY ALGORITHM

Akhil S - CB.SC.U4CSE24105
Anagha Govind - CB.SC.U4CSE24106
Chittesh D P - CB.SC.U4CSE24112
Punita Hari - CB.SC.U4CSE24138

# TECH STACK



## FRONTEND LAYER

Next.js 16 → React framework with SSR
TypeScript → Type-safe development
Axios → API communication
React 18 → UI components

REST API

## BACKEND LAYER

Java 17 → Core language
Spring Boot 4.0 → REST API framework
Maven → Build automation
Architecture

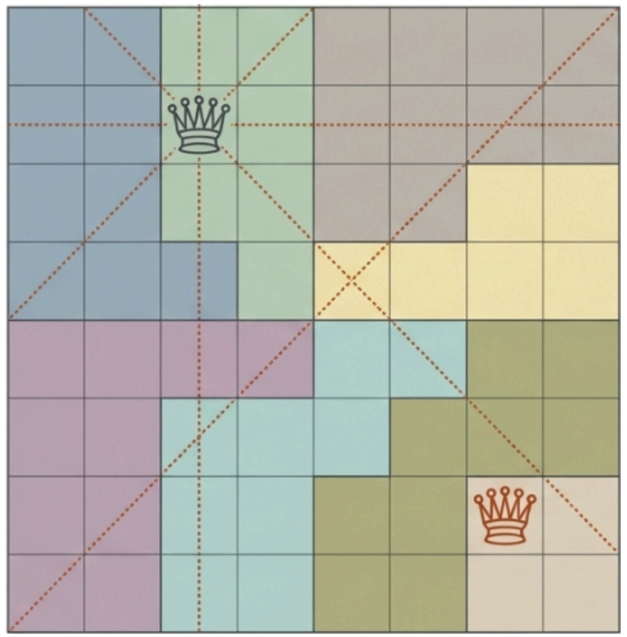
## KEY API ENDPOINTS

| | |
|---|---|
| POST | /api/gave/init : Initiatises a new game. |
| POST | /ap1/game/move : Submits a player's move. |
| POST | /api/gave/ai-move : Requests the AI's next move. |

# A NEW TWIST ON A CLASSIC CHALLENGE: THE GAME RULES

## OBJECTIVE

Force your opponent into a position with no valid moves by strategically placing queens.



## CORE MECHANICS

The Board: An N x N grid divided into exactly N distinct coloured regions.

The Play: Players alternate placing one queen per turn.

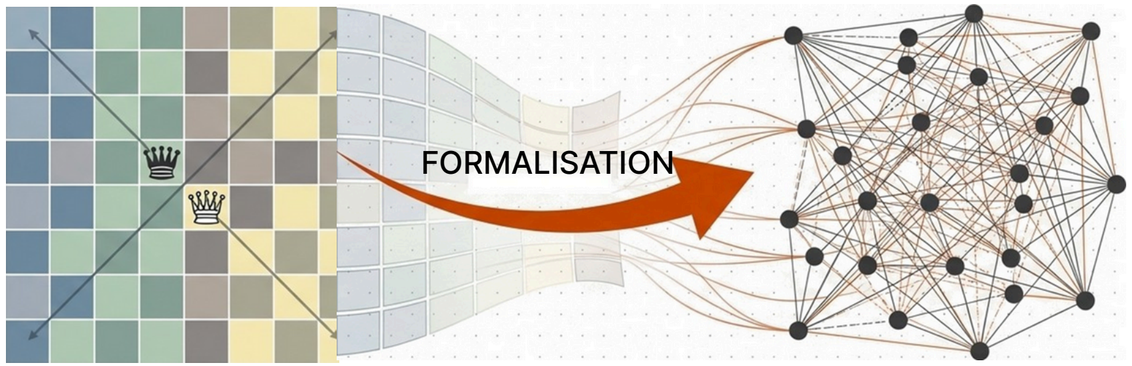
The Play: Players alternate placing one queen per turn.
The Constraints: Queens cannot attack each other (no shared row, column, or diagonal). Each coloured region can contain at most one queen.

The End: The game concludes when a player cannot make a valid move

## WIN CONDITIONS

Strategic Win: Your opponent has no valid moves remaining.
Perfect Game: All N queens are successfully placed on the board

# TRANSLATING GAME RULES INTO A COMPUTATIONAL MODEL




How can a complex set of spatial rules and constraints be represented in a way a computer can efficiently process and strategise with?

We model the game as an undirected graph, where the board's properties become nodes and the game's constraints become edges. This transforms a board game problem into a well-defined graph theory problem.

## FORMAL DEFINITION

The game is modelled as an undirected graph G = (V, E).

Vertices (V): Each of the N2 cells on the board is a vertex

Edges (E): An edge connects two vertices if placing queens on both ce(is would violate a game rule (i.e., they are in the same row, column, diagonal, or region).

Natural Constraint Modelling: Attack vectors (rows, columns, diagonals) and region limits are perfectly represented as edges in a conflict graph. A valid move is an independent set.

Atgorithmic Efficiency: Move validation becomes a simple check of adjacency, and finding valid moves is a straightforward graph traversal.

Mathematical Foundation: The problem elegantly reduces to finding a maximum independent set, where each of the N partitioned regions must contribute exactly one vertex.

# SORTING ALGORITHM: MERGE SORT

Before the AI places a queen, it sorts the N regions by the number of available cells in ascending order. This forces the algorithm to address the most constrained regions first, significantly increasing the probability of finding a successful placement for all N queens. A classic divide-and-conquer algorithm, it recursively splits the list of regions, sorts the halves, and merges them.

## WHY MERGE SORT?

01

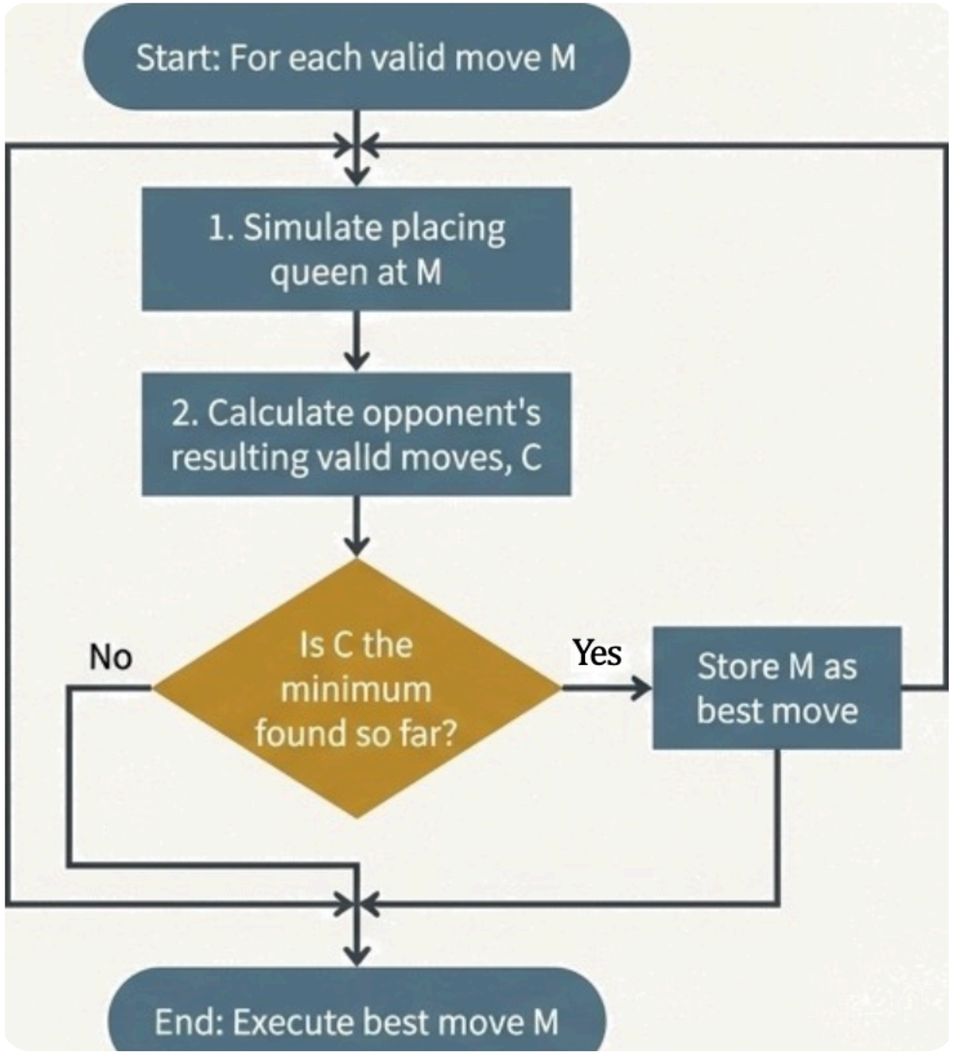
Performance Guarantee: A stable and predictable worst-case time complexity of O(log N).

02

Stability: Maintains the relative order of regions with an equal number of cells.

03

Pedagogy: A clear demonstration of a foundational divide-and conquer strategy.

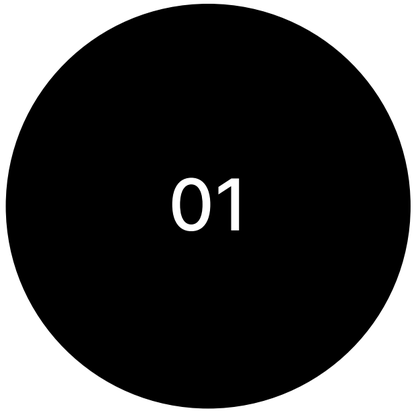



# GREEDY ALGORITHM

The AI makes its decision by looking just one move ahead for itself and one for its opponent. For each of its own valid moves, it performs the following evaluation:

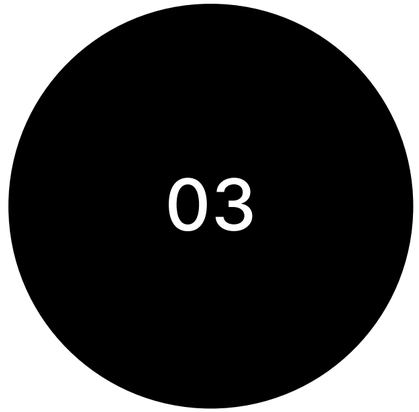
1. Simulate: Tentatively place a queen in a valid position.
2. Calculate: Count the total number of valid moves the opponent would have on the next turn.
3. Decide: Choose the move that minimises the opponent's future options.
4. Tie-Breaker: If multiple moves result in the same minimum for the opponent, choose the one that maximises its own future options.

# WHY A GREEDY APPROACH?




It only looks 2 plies ahead (its move + opponent's response).


It makes a locally optimal choice without exploring the entire game tree.




It is fast and avoids backtracking once a decision is made.

The AI is challenging but not frustratingly perfect. It is still beatable by a clever human player. This trade-off prioritises a smooth, responsive user experience over brute-force, perfect play. It consistently beats random players 85% of the time, proving its effectiveness. It is challenging, but beatable; rewards strategic thinking.

# INDIVIDUAL CONTRIBUTION

## AKHIL S

Backend core logic and game state management. Region generation on the board. Algorithm debugging and testing. System integration and deployment.

## CHITTESH D P

Frontend development and UI implementation. API integration and communication layer. Merge sort algorithm implementation. Frontend-backend connectivity

## ANAGHA GOVING

Greedy algorithm design and implementation. Algorithm complexity analysis. Report writing and documentation. Project structure and organisation.

## PUNITA HARI

Graphical representation and visual design. Color generation and aesthetics. PowerPoint presentation creation. UI styling and layout.

# EXPLORE THE COMPLETE PROJECT ON GITHUB

*Thank You*