My First Sprouts Solver

Create a functional program that can play Sprouts against a human or another AI using a search algorithm with optimizations.

Phase 1: The Foundation (Game Logic)

- **Board Representation:** Choose a data structure. A graph is recommended (use an adjacency list or matrix to represent dots and lines).
- **State Management:** Create functions to accurately represent the current state of the board.
  - `function add_dot_and_line(dot1, dot2, new_dot_position)`
  - `function get_dot_lives(dot)` - to track remaining connections (1, 2, or 3).
- **Move Generation:** Write a function that identifies all legal moves from a given position.
  - `function find_all_legal_moves(current_state)`
- **Game Termination:** Implement a check to determine if the game has ended (no legal moves are available).
  - `function is_game_over(current_state)`

Phase 2: The Search Algorithm (AI Core)

- **Minimax Function:** Implement the core recursive Minimax algorithm.
  - `function minimax(state, depth, is_maximizing_player)`
- **Evaluation Function:** Create a simple evaluation for terminal nodes.
  - If the current player has no moves, return -1 (loss).
  - If the opponent has no moves, return +1 (win).
- **Recursive Calls:** Ensure the algorithm correctly alternates between maximizing and minimizing players as it goes deeper into the decision tree.

Phase 3: Optimization

- **Alpha-Beta Pruning:** Integrate alpha and beta variables into your Minimax function to prune search branches.
  - Pass `alpha` and `beta` values through your recursive calls.
  - Add the core condition: `if beta <= alpha, break` the loop.
- **Move Ordering:** Implement a simple heuristic to sort moves before searching.
  - Suggestion: Prioritize moves that leave the opponent with fewer subsequent moves (a "mobility" heuristic).
- **Isomorphism (Optional but Recommended):** Implement a way to create a canonical representation (a unique ID or hash) for each board state to avoid re-calculating duplicate positions.

Phase 4: Advanced Features

- **Heuristic Evaluation:** Develop a function to evaluate non-terminal positions.

    - Start with mobility (number of available moves).
    - Consider more advanced metrics like region analysis.
- **Opening Book:** Pre-calculate and store the optimal first few moves for common starting positions (e.g., n=2, n=3).
- **User Interface:** Build a simple command-line or graphical interface to play against your AI.