OS2 project documentation

Project description:

The "Make a Square" project involves creating a 4x4 square using a set of pieces that can be rotated or flipped. The goal is to utilize all the pieces to form a perfect square shape. The project accepts input specifying the number of pieces and their shapes, then determines and displays possible solutions if they exist.

Problem Statement:

- **Objective:** Form a 4x4 square using a specified set of 4 or 5 pieces, allowing rotation and flipping of pieces.
- **Constraints:** All pieces must be used, and not all sets of pieces may form a square.

Input Specifications:

- The input starts with the number of pieces.
- Each subsequent line specifies a piece:
    - Two integers representing the number of rows and columns in the piece.
    - Followed by lines indicating the shape of the piece with '0' or '1' characters, where '1' represents the solid shape.

Output Specifications:

- The program should output all possible solutions.
- A 4x4 square should be created, with each piece filling its corresponding location.
- Solid parts of each piece are represented by numeric characters ('1' for piece #1, '2' for piece #2, and so on).
- For unsolvable cases, the program should output "No solution possible".

Code documentation:

1. main Method

- Handles user interaction by taking input for the number of puzzle pieces and their details.
- Initiates the puzzle-solving process and displays the solution or notifies the user if no solution exists.

2. solvePuzzle Method

- Coordinates the solving process by distributing tasks to multiple threads using Java's **ExecutorService**.
- Manages the recursive backtracking algorithm to explore various configurations of piece placements on the board.

3. solveHelper Method

- Recursively explores different orientations and positions of pieces to find a valid solution.
- Utilizes backtracking to backtrack when a solution is not viable in the current configuration.

4. Piece Manipulation Methods

- **isValidSolution**: Checks if the board is completely filled, determining if a solution is valid.
- **canPlacePiece**: Verifies if a piece can be placed in a specific position on the board.
- **placePiece**: Places a piece on the board at a specified position.
- **rotatePiece**: Rotates a piece by 90 degrees based on the specified number of rotations.
- **flipPiece**: Flips a piece horizontally.
- **removePieceNumber**: Removes a piece number from the list of available piece numbers.
- **copyBoard**: Creates a deep copy of the board to manage changes during the solving process.

5. Thread Management

- Utilizes Java's **ExecutorService** and multi-threading (**ExecutorService.invokeAll**) to parallelize the search for a solution, enhancing efficiency.

GUI:

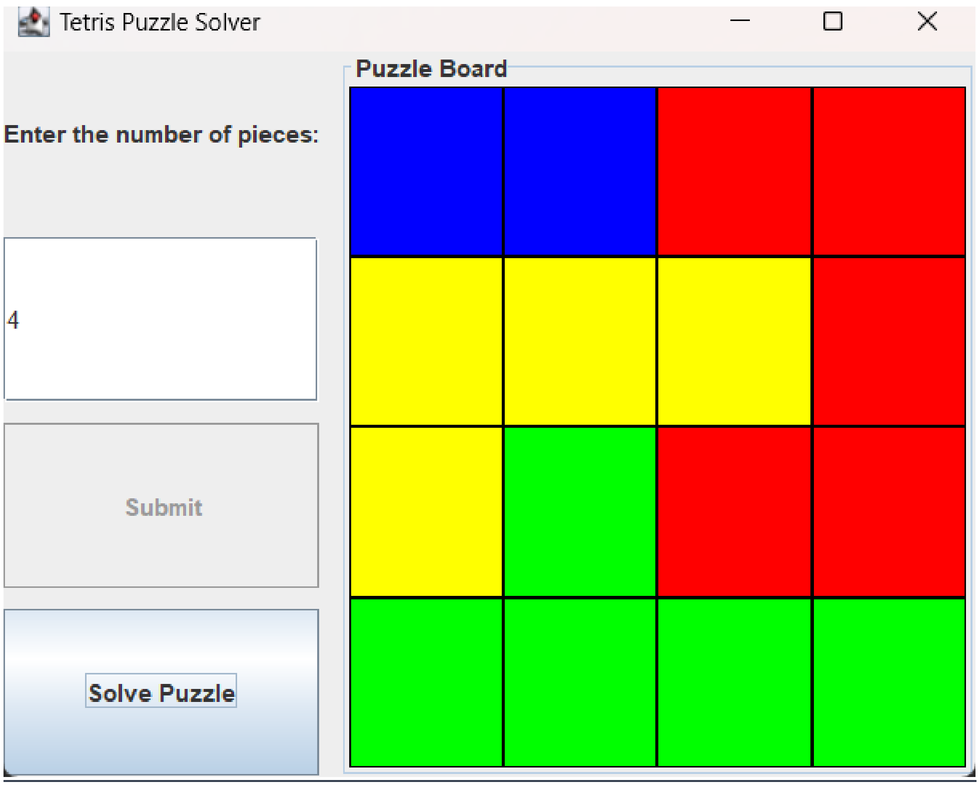
TetrisPuzzleSolverGUI Class:

- **Frame Initialization:**
    - Inherits **JFrame** to create the main window for the application.
    - Sets the window title to "Tetris Puzzle Solver".
    - Defines window dimensions and layout.
- **Input Panel Creation:**
    - Creates an input panel using **JPanel** to gather user input.
    - Includes input fields for the number of pieces (**numPiecesField**), a "Submit" button (**submitButton**), and a "Solve Puzzle" button (**solveButton**).
- **Event Listeners:**
    - **submitButton** ActionListener: Validates the input for the number of pieces and enables the "Solve Puzzle" button upon successful submission.
    - **solveButton** ActionListener: Prompts users to input the details of each piece (rows, columns, and shape) and then attempts to solve the puzzle.
- **Puzzle Solving:**
    - The **solvePuzzle** method processes user-provided details for each piece and attempts to solve the puzzle using the **TetrisPuzzleSolver** class's **solvePuzzle** method.
    - Displays a message dialog with the solved puzzle configuration if a solution is found or notifies the user if no solution exists.

- **Output Display:**
    - Updates the GUI to display the solved puzzle board:
        - Clears the **puzzleBoardPanel**.
        - Creates panels representing each cell of the solved puzzle board.
        - Sets the background color of each panel based on the piece number.
        - Renders the updated puzzle board within the GUI.
- **Main Method:**
    - Invokes the GUI within the Event Dispatch Thread (EDT) using **SwingUtilities.invokeLater**.

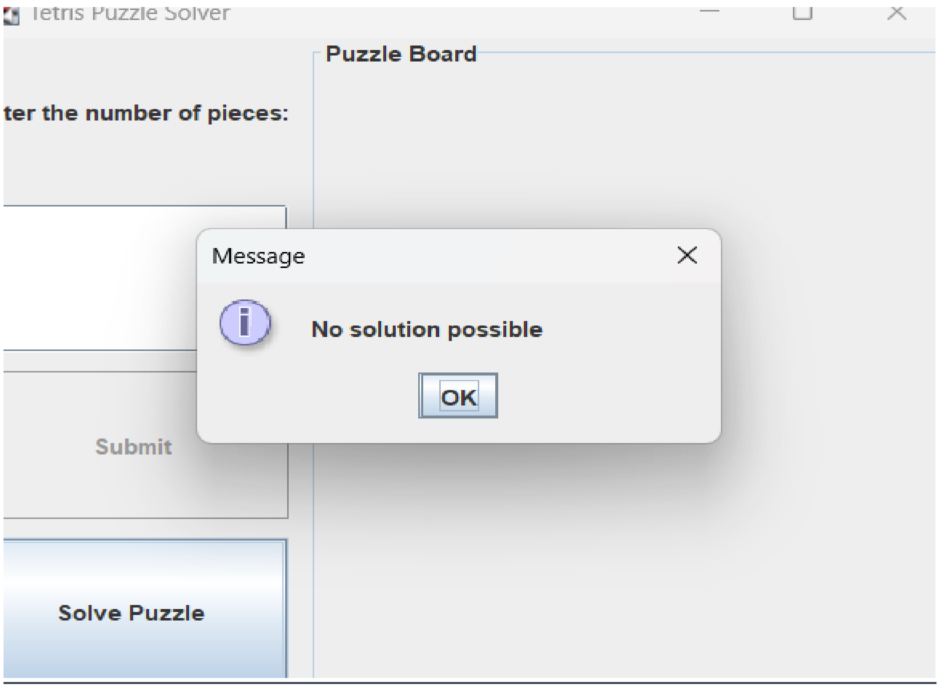
Usage:

1. **Launch the Application:**
    - Run the program to open the Tetris Puzzle Solver GUI window.
    - Input the number of puzzle pieces and submit to enable the "Solve Puzzle" button.
2. **Input Puzzle Pieces:**
    - Click the "Solve Puzzle" button to input details for each puzzle piece (rows, columns, shape).
    - The GUI guides the user through input prompts via message dialogs.
3. **View Solution:**
    - Upon successful puzzle solving, the GUI displays the solved board configuration within the application window.
4. **Error Handling:**
    - The GUI provides error messages for invalid inputs and alerts users if no solution is possible.



We tried many samples in the project, by giving it different pieces and it finds the way to combine those pieces. if there’s a possible combination to the pieces, it will be drawn on the gui as we have shown above.

if the program doesn’t find a way to combine the pieces, it prints no solution found!



No solution possible