**Project code explanation**

|  |
| --- |
| clc;  clear all;  close all;  originalImage = imread('noisy\_colorful\_image.jpg'); % Load the original image  imshow(originalImage); % Display the original image  puzzleFolder = 'puzzleimages'; % Folder containing the puzzle pieces  numPieces = 16; % Total number of puzzle pieces  gridSize = 4; % The grid size (4x4 for 16 pieces) |
| * clc: Clears the command window, providing a fresh workspace. * clear all: Clears all variables from the workspace to prevent interference with previous variables. * close all: Closes all open figure windows, ensuring that any previous plots or images are removed.   This section prepares the MATLAB environment by clearing unnecessary variables and closing figures**.** |

|  |
| --- |
| grayOriginal = rgb2gray(originalImage);  [rows, cols] = size(grayOriginal); % Get the size of the grayscale image  blockRows = floor(rows / gridSize); % Number of rows per block  blockCols = floor(cols / gridSize); % Number of columns per block |
|  |

|  |
| --- |
| originalBlocks = cell(gridSize, gridSize);  for i = 1:gridSize  for j = 1:gridSize  rowRange = (i-1)\*blockRows + 1 : min(i\*blockRows, rows); % Ensure rows stay within bounds  colRange = (j-1)\*blockCols + 1 : min(j\*blockCols, cols); % Ensure columns stay within bounds    originalBlocks{i, j} = grayOriginal(rowRange, colRange);  end  end |
|  |

|  |
| --- |
| puzzlePieces = cell(numPieces, 1);  for k = 1:numPieces  pieceFilename = fullfile(puzzleFolder, sprintf('piece%d.jpg', k));  puzzlePieces{k} = imread(pieceFilename);  end  finalArrangement = zeros(gridSize, gridSize); % Store the arrangement of pieces  usedPieces = false(numPieces, 1); % Array to track which pieces have been used |
|  |

|  |
| --- |
| for i = 1:gridSize  for j = 1:gridSize  bestMatch = inf;  bestPiece = 0;    for k = 1:numPieces  if ~usedPieces(k) % Only consider unused pieces  piece = puzzlePieces{k};  resizedPiece = imresize(piece, [blockRows, blockCols]); % Resize piece to match block size    % Convert puzzle piece to grayscale if RGB  if size(resizedPiece, 3) == 3  resizedPiece = rgb2gray(resizedPiece);  end    % Calculate similarity between block and piece  diff = sum((double(originalBlocks{i, j}) - double(resizedPiece)).^2, 'all');    if diff < bestMatch  bestMatch = diff;  bestPiece = k;  end  end  end    % Assign the best matching piece to the current block  finalArrangement(i, j) = bestPiece;  usedPieces(bestPiece) = true; % Mark this piece as used  end  end |
|  |

|  |
| --- |
| disp('Final arrangement of puzzle pieces (as matrix):');  disp(finalArrangement);  reconstructedImage = uint8(zeros(rows, cols)); % Initialize an empty image  for i = 1:gridSize  for j = 1:gridSize  rowRange = (i-1)\*blockRows + 1 : i\*blockRows;  colRange = (j-1)\*blockCols + 1 : j\*blockCols;    piece = imresize(puzzlePieces{finalArrangement(i, j)}, [blockRows, blockCols]);    % Convert puzzle piece to grayscale if RGB  if size(piece, 3) == 3  piece = rgb2gray(piece);  end    % Place the resized piece into the reconstructed image  reconstructedImage(rowRange, colRange) = piece;  end  end |
| Displays the final arrangement of puzzle pieces as a 4x4 matrix, showing which piece is assigned to each block. |

|  |
| --- |
| figure;  imshow(reconstructedImage);  title('Reconstructed Grayscale Image'); |
|  **imshow**: Displays the reconstructed grayscale image.   **title**: Adds a title to the image |

The code performs the following tasks:

1. **Loads the original image** and the puzzle pieces.
2. **Converts the original image** to grayscale and divides it into 4x4 blocks.
3. **Resizes and compares each puzzle piece** with the blocks, selecting the best match using the sum of squared differences.
4. **Reconstructs the original image** by placing the puzzle pieces in the correct positions.

The final output is a grayscale reconstruction of the original image, using the correctly arranged puzzle pieces.