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.

- `imread`: Loads the original image from the specified path (`'noisy_colorful_image.jpg'` in this case).
- `imshow`: Displays the loaded image on the screen.
- `puzzleFolder`: The folder where the 16 puzzle pieces are stored.
- `numPieces` **and** `gridSize`: Define the number of puzzle pieces (16) and the grid size (4x4), indicating that the original image will be split into 4x4 blocks.

```
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
```

`rgb2gray` : Converts the original color image (`originalImage`) into a grayscale image (`grayOriginal`). This simplifies the matching process, as grayscale images have only one channel, while color images have three channels (RGB).

`size` : Retrieves the dimensions of the grayscale image (`grayOriginal`), i.e., the number of rows and columns.

`blockRows` **and** `blockCols` : Calculate the number of rows and columns each block should have. This ensures that the original image is split into `gridSize` blocks. The `floor` function is used to ensure integer values when dividing.

```
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
```

`originalBlocks` : A cell array is initialized to store the blocks of the original image.

The nested `for` loops iterate over the grid size (4x4). For each block the `rowRange` and `colRange` define the pixel indices for each sub-image.

- `rowRange` **and** `colRange` : These ensure that each block extracts the correct part of the original image.
- `originalBlocks{i, j}` : Stores each 4x4 block of the grayscale image.

```
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
```

`puzzlePieces`: A cell array to store the 16 puzzle pieces.

A `for` loop loads each puzzle piece from the specified folder (`puzzleFolder`).

- `fullfile`: Creates the full path to each puzzle piece by combining the folder name and the piece filename (`piece1.jpg`, `piece2.jpg`, etc.).
- `imread`: Reads each puzzle piece and stores it in the `puzzlePieces` cell array.

`finalArrangement`: A matrix to store the final placement of puzzle pieces, which will later be used to reconstruct the image.

`usedPieces`: A logical array to keep track of which puzzle pieces have been used. Each entry corresponds to a puzzle piece; `false` means unused, and `true` means 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
```

`bestMatch` **and** `bestPiece`: These variables store the smallest difference (`diff`) and the corresponding puzzle piece index.

Nested `for` **loops**:

- The outer loops iterate over each block in the original image.
- The inner loop compares each unused puzzle piece with the current block, resizing the puzzle piece and converting it to grayscale if necessary.
- `diff`: Calculates the difference between the block and the resized puzzle piece. The `sum` of squared differences is used as a similarity measure (the lower the `diff`, the better the match).
- `usedPieces`: Ensures that each piece is used only once by marking it as used once assigned.

```
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.

`reconstructedImage`: An empty grayscale image is initialized to store the final reconstructed image.

Nested loops: These loops iterate over the grid, resizing the assigned puzzle piece and placing it in its corresponding location in `reconstructedImage`.

```
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.