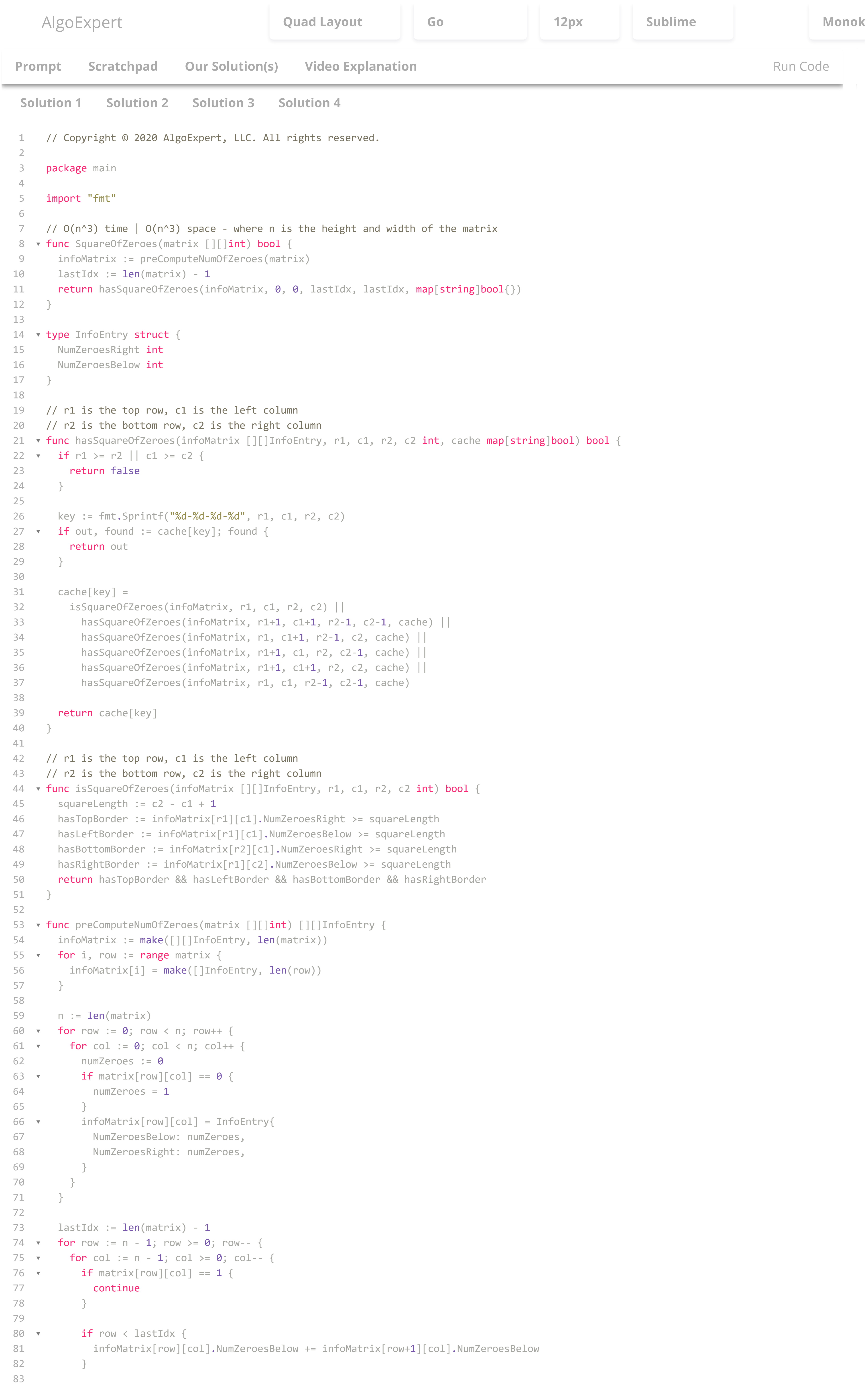

```go
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

package main

import "fmt"

// O(n^3) time | O(n^3) space - where n is the height and width of the matrix
func SquareOfZeroes(matrix [][]int) bool {
  infoMatrix := preComputeNumOfZeroes(matrix)
  lastIdx := len(matrix) - 1
  return hasSquareOfZeroes(infoMatrix, 0, 0, lastIdx, lastIdx, map[string]bool{})
}

type InfoEntry struct {
  NumZeroesRight int
  NumZeroesBelow int
}

// r1 is the top row, c1 is the left column
// r2 is the bottom row, c2 is the right column
func hasSquareOfZeroes(infoMatrix [][]InfoEntry, r1, c1, r2, c2 int, cache map[string]bool) bool {
  if r1 >= r2 || c1 >= c2 {
    return false
  }

  key := fmt.Sprintf("%d-%d-%d-%d", r1, c1, r2, c2)
  if out, found := cache[key]; found {
    return out
  }

  cache[key] =
    isSquareOfZeroes(infoMatrix, r1, c1, r2, c2) ||
      hasSquareOfZeroes(infoMatrix, r1+1, c1+1, r2-1, c2-1, cache) ||
      hasSquareOfZeroes(infoMatrix, r1, c1+1, r2-1, c2, cache) ||
      hasSquareOfZeroes(infoMatrix, r1+1, c1, r2, c2-1, cache) ||
      hasSquareOfZeroes(infoMatrix, r1+1, c1+1, r2, c2, cache) ||
      hasSquareOfZeroes(infoMatrix, r1, c1, r2-1, c2-1, cache)

  return cache[key]
}

// r1 is the top row, c1 is the left column
// r2 is the bottom row, c2 is the right column
func isSquareOfZeroes(infoMatrix [][]InfoEntry, r1, c1, r2, c2 int) bool {
  squareLength := c2 - c1 + 1
  hasTopBorder := infoMatrix[r1][c1].NumZeroesRight >= squareLength
  hasLeftBorder := infoMatrix[r1][c1].NumZeroesBelow >= squareLength
  hasBottomBorder := infoMatrix[r2][c1].NumZeroesRight >= squareLength
  hasRightBorder := infoMatrix[r1][c2].NumZeroesBelow >= squareLength
  return hasTopBorder && hasLeftBorder && hasBottomBorder && hasRightBorder
}

func preComputeNumOfZeroes(matrix [][]int) [][]InfoEntry {
  infoMatrix := make([][]InfoEntry, len(matrix))
  for i, row := range matrix {
    infoMatrix[i] = make([]InfoEntry, len(row))
  }

  n := len(matrix)
  for row := 0; row < n; row++ {
    for col := 0; col < n; col++ {
      numZeroes := 0
      if matrix[row][col] == 0 {
        numZeroes = 1
      }
      infoMatrix[row][col] = InfoEntry{
        NumZeroesBelow: numZeroes,
        NumZeroesRight: numZeroes,
      }
    }
  }

  lastIdx := len(matrix) - 1
  for row := n - 1; row >= 0; row-- {
    for col := n - 1; col >= 0; col-- {
      if matrix[row][col] == 1 {
        continue
      }

      if row < lastIdx {
        infoMatrix[row][col].NumZeroesBelow += infoMatrix[row+1][col].NumZeroesBelow
      }

      if col < lastIdx {
        infoMatrix[row][col].NumZeroesRight += infoMatrix[row][col+1].NumZeroesRight
      }
    }
  }
  return infoMatrix
}

```