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

package main

type Coord struct {
  X, Y int
}

type Direction int

const (
  None Direction = iota - 1
  Up
  Down
  Left
  Right
)

// O(n^2) time | O(n^2) space - where n is the number of coordinates
func RectangleMania(coords []Coord) int {
  coordsTable := getCoordsTable(coords)
  return getRectangleCount(coords, coordsTable)
}

type CoordSet map[Coord]struct{}
type CoordsTable map[Coord]map[Direction]CoordSet

func getCoordsTable(coords []Coord) CoordsTable {
  table := CoordsTable{}
  for _, coord1 := range coords {
    directions := map[Direction]CoordSet{
      Up:    CoordSet{},
      Right: CoordSet{},
      Down:  CoordSet{},
      Left:  CoordSet{},
    }
    for _, coord2 := range coords {
      coord2Direction := getCoordDirection(coord1, coord2)
      if coord2Direction != None {
        directions[coord2Direction][coord2] = struct{}{}
      }
    }
    table[coord1] = directions
  }
  return table
}

func getCoordDirection(coord1, coord2 Coord) Direction {
  if coord2.Y == coord1.Y {
    if coord2.X > coord1.X {
      return Right
    } else if coord2.X < coord1.X {
      return Left
    }
  } else if coord2.X == coord1.X {
    if coord2.Y > coord1.Y {
      return Up
    } else if coord2.Y < coord1.Y {
      return Down
    }
  }
  return None
}

func getRectangleCount(coords []Coord, coordsTable CoordsTable) int {
  count := 0
  for _, coord := range coords {
    count += clockwiseCountRectangles(coord, coordsTable, Up, coord)
  }
  return count
}

func clockwiseCountRectangles(coord Coord, coordsTable CoordsTable, direction Direction, origin Coord) int {
  if direction == Left {
    if _, found := coordsTable[coord][Left][origin]; found {
      return 1
    }
    return 0
  }
  rectangleCount := 0
  nextDirection := direction.NextClockwise()
  for nextCoord := range coordsTable[coord][direction] {
    rectangleCount += clockwiseCountRectangles(nextCoord, coordsTable, nextDirection, origin)
  }
  return rectangleCount
}

func (d Direction) NextClockwise() Direction {
  switch d {
  case Up:
    return Right
  case Right:
    return Down
  case Down:
    return Left
  case Left:
    return Up
  }
  return None
}

```