```
print("Hello World! ¿ ¿ ¿ \n\n")

using Plots
gr()

plot([1,2],[2,3])

module rrt
    export Node, Edge, Obstacle, Room, Point

    struct Point
        x::Int
        y::Int
    end

    struct Node
        id::Int
        iPrev::Int #parent
        state::Point
    end

    struct Edge
        startnode::Int
        endnode::Int
    end

    struct Obstacle
        SW::Point
        NE::Point
    end

    ### Some types?
    struct Room #corners of the room
        x1::Int
        y1::Int #bottom left corner?
        x2::Int
        y2::Int
        obstacleList::Array{Obstacle}
    end

    struct robot
    end

end


function isCollidingNode(pt,obs)
    # this does not work.
    px,py = pt.x, pt.y
    x1,y1 = obs.SW.x, obs.SW.y
    x2,y2 = obs.NE.x, obs.NE.y

    if (px > x1 && px < x2 && py > y1 && py < y2)
        #print("Node in obstacle, discarded.")
        return true
    else
        return false
    end
end


function isCollidingEdge(r, nn, obs)
    # to detect collision, let's just check whether any of the four
    # lines of the rectangular obstacle intersect with our edge
    # ignore collinearity for now

    # Tofix: To make it look prettier in the graph, I am just going to add a 1 grid pt
    # spacer for now, until I figure out how to check for coincidence of point
    # in line. Or really, I should use GeometryShapes library

    x1,y1 = obs.SW.x, obs.SW.y
    x2,y2 = obs.NE.x, obs.NE.y

    x1 -= 1
    y1 -= 1
    x2 += 1
    y2 += 1

    pt1 = rrt.Point(x1, y1)
    pt2 = rrt.Point(x1, y2)
    pt3 = rrt.Point(x2, y2)
    pt4 = rrt.Point(x2, y1)

    # let A, B be r, nn
    coll1 = intersectLineSeg(r, nn, pt1, pt2)
    coll2 = intersectLineSeg(r, nn, pt2, pt3)
    coll3 = intersectLineSeg(r, nn, pt3, pt4)
    coll4 = intersectLineSeg(r, nn, pt4, pt1)

    if (!coll1 && !coll2 && !coll3 && !coll4)
        return false
    else
        #print("you're colliding!\n")
        #@show nn
        return true
    end
end


function inGoalRegion(node, goal)
    goal = rrt.Point(18,18)
    n = node.state
    if node == goal
        return true
    end
    distx = abs(n.x - goal.x)
    disty = abs(n.y - goal.y)
    if ( distx < 2 && disty < 2) #radius of goal
        return true
    end
    return false

end


function ccw(A,B,C)
    # determines direction of lines formed by three points
    return (C.y-A.y) * (B.x-A.x) > (B.y-A.y) * (C.x-A.x)
end


function intersectLineSeg(A,B,C,D) #no ":" at the end!
    return ( (ccw(A, C, D) != ccw(B, C, D)) && ccw(A, B, C) != ccw(A, B, D))
end


function nearestN(r, nodeslist) #dear lord rewrite this so it doesn't need to pass in maxNodeId
    nearestDist = 9999;
    nearestNode = [];
    for n in nodeslist
        dist = distPt(r, n.state)
        if dist < nearestDist
            nearestDist = dist
            nearestNode = n
        end
    end
    return nearestNode
end


function distPt(pt1, pt2)
    x2,y2 = pt2.x, pt2.y
    x1,y1 = pt1.x, pt1.y
    dist = sqrt( (x1-x2)^2 + (y1-y2)^2 )
    return dist
end
```

```
function rrtPathPlanner(niterations)
    nIter = niterations
    maxDist = 3
    #room = Room(0,0,21,21);

    obs1 = rrt.Obstacle(rrt.Point(8,3),rrt.Point(10,18)) #Todo
    #obs2 = rrt.Obstacle(rrt.Point(10,12),rrt.Point(14,12)) #Todo
    obs1 = rrt.Obstacle(rrt.Point(8,5),rrt.Point(10,18)) #Todo

    rrtstart = rrt.Point(1,0)
    goal = rrt.Point(18,18)

    nodeslist = Vector{rrt.Node}()

    startNode = rrt.Node(0,0, rrtstart)
    push!(nodeslist, startNode)
    #@printf("string %s",nodeslist)

    maxNodeID = 1
    isPathFound = false

    for i in 1:nIter
        r = rrt.Point(rand(1:20),rand(1:20)) #new point
        nn = nearestN(r, nodeslist) # parent point

        #if !isCollidingNode(r, obs1)#check node XY first #Tofix: this function
        if !isCollidingNode(r, obs1) && !isCollidingEdge(r, nn.state, obs1) # check edge #rewrite so we can check multiple obstacles...
                nearestDist = distPt(r, nn.state)
                    if nearestDist > maxDist
                        x1,y1 = nn.state.x, nn.state.y
                        x2,y2 = r.x, r.y
                        tantheta = atan2((y2-y1) , (x2-x1))
                        newX = x1 + maxDist * cos(tantheta)
                        newY = y1 + maxDist * sin(tantheta)
                        newPt = rrt.Point(floor(newX), floor(newY)) #Todo: maybe I shouldn't floor?
                        node = rrt.Node(maxNodeID, nn.id, newPt) #throw out r, but try to steer toward it
                    else
                        node = rrt.Node(maxNodeID, nn.id, r)
                    end
                maxNodeID += 1
                push!(nodeslist, node)

                if inGoalRegion(node, goal)
                    #print("Goallllll! This is the winning node:\n")
                    #@show node  #winning node
                    #@printf("Goallll! Found after %d iterations", i)
                    isPathFound = true
                    break
                end
        else
            end
        end

    #@show nodeslist
    #@printf("\nPath found? %s Length of nodeslist: %d\n", isPathFound, length(nodeslist))
    cost = costWinningPath(nodeslist)
    return cost, isPathFound, nodeslist
end


function plotPath(isPathFound, nlist) #rewrite so don't need to pass in isPathFound, obs1, rrtstart, goal, room

    ### <COPIED FOR NOW #Todo fix hardcoding
    obs1 = rrt.Obstacle(rrt.Point(8,5),rrt.Point(10,18)) #Todo

    rrtstart = rrt.Point(1,0)
    goal = rrt.Point(18,18)
    ### / COPIED FOR NOW>


    foo = rand(1)
    h = plot()

    @printf("%s", "plotted\n")
    plot!(h, show=true, legend=false, size=(600,600),xaxis=((-5,25), 0:1:20 ), yaxis=((-5,25), 0:1:20), foreground_color_grid=:lightcyan)

    nIter = -1#fix hardcoding
    title!("RRT nIter = $(nIter), Path Found $(isPathFound)")

    # plot room
    dim = 21
    roomx = [0,0,dim,dim];
    roomy = [0,dim,dim,0];
    plot!(roomx, roomy, color=:black, linewidth=5)

    # plot start and end goals
    circle(1,0, 0.5, :red)
    circle(18,18, 0.5, :forestgreen)
    rectEnd = rrt.Obstacle(rrt.Point(17,17),rrt.Point(19,19)) #Todo
    rectObs(rectEnd)

    # plot obstacles
    rectObs(obs1)

    # plot all paths
    plotEdges(nlist)

    # plot all points
    x = [v.state.x for v in nlist]
    y = [v.state.y for v in nlist]
    scatter!(x,y, linewidth=0.1, color=:grey)

    print("Done plotting $(length(nlist)) nodes\n")




    # plot winning path
    if isPathFound
        cost = plotWinningPath(nlist)
        @printf("\n!!!! the cost of the path was %d across %d nodes  !!!! \n", cost, length(nlist))
        #nEnd = nlist[end]
    end
    # for n in nlist
        # if n.id == 0
            # continue
        # else
            # plotEdge(n,nlist)
        # end
    # end
    # display winning path cost
end

#module pHelp()

    ### Some helper functions
    function circle(x,y,r,c_color)
        th = 0:pi/50:2*pi;
        xunit = r * cos.(th) + x;
        yunit = r * sin.(th) + y;
        plot!(xunit, yunit,color=c_color,linewidth=3.0);
    end

    function rectObs(obstacle)
        x1,y1 = obstacle.SW.x, obstacle.SW.y
        x2,y2 = obstacle.NE.x, obstacle.NE.y
        r = 0.2
        obsColor = :blue

        plot!([x1,x1,x2,x2,x1], [y1,y2,y2,y1,y1], color=obsColor, linewidth=2)
    end



    function plotEdges(nlist)
        edgeXs, edgeYs = [], []

        for n in nlist
            pt1 = n.state
            iPrev = n.iPrev
            nPrev = findNode(iPrev, nlist)
            pt2 = nPrev.state

            x1,y1 = pt1.x, pt1.y
            x2,y2 = pt2.x, pt2.y

            push!(edgeXs, x1, x2, NaN)
            push!(edgeYs, y1, y2, NaN)
        end
        plot!(edgeXs, edgeYs, color=:orange, linewidth=1.5)
        print("Done plotting $(length(nlist)) edges\n")

    end




    function costWinningPath(nlist)
        curNode = nlist[end]
        guhPath = [ rrt.Point(curNode.state.x, curNode.state.y)]
        while true
            iPrev = curNode.iPrev
            curNode = findNode(iPrev, nlist)
            x,y = curNode.state.x, curNode.state.y
            push!( guhPath, rrt.Point(x,y))
            if curNode.id == 0
                push!( guhPath, rrt.Point(x,y))
                break
            end
        end
        cost = 0
        for i in 2:length(guhPath)
            cost += distPt(guhPath[i],guhPath[i-1])
        end

        return cost
    end

    function plotWinningPath(nlist)
        curNode = nlist[end]
        xPath = [curNode.state.x]
        yPath = [curNode.state.y]
        guhPath = [ rrt.Point(curNode.state.x, curNode.state.y)]
        while true
            iPrev = curNode.iPrev
            curNode = findNode(iPrev, nlist)
            x,y = curNode.state.x, curNode.state.y
            push!(xPath, x)
            push!(yPath,y)
            push!( guhPath, rrt.Point(x,y))
            if curNode.id == 0
                push!(xPath, x)
                push!(yPath,y)
                break
            end
        end
        print("plotted winning path")
        cost = 0
        for i in 2:length(guhPath)
            cost += distPt(guhPath[i],guhPath[i-1])
        end

        plot!( xPath, yPath, color = :green, linewidth=3)
        #@show nlist
        return cost
    end

    function findNode(id, nodeslist)
        for n in nodeslist
            if n.id == id
                return n
            end
        end
    end
#end

cost, isPathFound, nlist = rrtPathPlanner(150) #maxIter
plotPath(isPathFound,nlist)
```