CODING

The following coding is written in Python programming Language 3.7:

```
##First of all the toolkit Tkinter is imported. Then a window is called where
the canvas widget is packed to
import tkinter as tk
window = tk.Tk()
w = tk.Canvas(window, width = 500, height = 500, bg = 'white')
w.pack()

##These are the global variables that will be used the functions
nodes = [] #a list that will sequencially store vertices as they created
edges = [] #a list that will sequencially store edges as they created in a
list
nodesDict = {} #a dictionary that will store the bending points or control
points around each vertex
edgesDict = {} #a dictionary that will store shadow lines and potent control
vertices of every edge
handshakes = {} #inspired from the handshake theorem, 'handshakes' is a
dictionary that will records the two endpoints of an edge
#'startNode' and 'endNode' will store the endpoints of a newly created edges
startNode = None
endNode = None
#'const' is a constant variable that will determine the distance of the
surrounding control points from a vertex
const = [(-10,-10),(0,-10),(10,-10),(10,0),(10,10),(0,10),(-10,10),(-10,0)]

def createNode(e):
    nodes.append(w.create_oval(e.x-20,e.y-20,e.x+20,e.y+20, fill = '#34A2FE',
    tags = 'nodes'))
def dragNode(e):
    w.coords(nodes[-1], e.x-20,e.y-20,e.x+20,e.y+20)
def releaseNode(e):
    global nodesDict
    x0,y0,x1,y1 = w.coords(nodes[-1])
    centerx,centery = x0+20,y0+20
    nodesDict[nodes[-1]] = {'topleft':(x0,y0), 'top':(centerx,centery-
     30),'topright':(x1,y0), 'right':(centerx+30,centery),
     'bottomright':(x1,y1),'bottom':(centerx,centery+30),
     'bottomleft':(x0,y1), 'left':(centerx-30,centery)}
def createEdge(e):
    global startNode,endNode,newEdge,edgesDict
    overlapped = w.find_overlapping(e.x-5,e.y-5,e.x+5,e.y+5)
    for Object in overlapped:
        if w.type(Object) == 'oval':
            startNode = Object
            ex0,ey0 = w.coords(startNode)[0]+20,w.coords(startNode)[1]+20
            edges.append(w.create_line(ex0,ey0,e.x,e.y,tags = 'edges',smooth
            = True))
            newEdge = edges[-1]
            w.tag_lower(newEdge,nodes[0])
```

```
                edgesDict[newEdge] = 
                {'shadowLine1':w.create_line(w.coords(newEdge),fill = 'white'),
                'shadowLine2':w.create_line(w.coords(newEdge),fill = 'white')}
                w.tag_lower(edgesDict[newEdge]['shadowLine1'], edges[0])
                w.tag_lower(edgesDict[newEdge]['shadowLine2'],
                edgesDict[newEdge]['shadowLine1'])
                edgesDict[newEdge]['controlnodes'] = [node for node in 
                w.find_withtag('nodes') if (startNode != node)]
                break

def addremoveControlPoints(edge,ex,ey,lineEndTipBox):
    #setup
    global nodesDict,edgesDict,const
    potentNodes = edgesDict[edge]['controlnodes']
    shadowLine1 = edgesDict[edge]['shadowLine1']
    shadowLine2 = edgesDict[edge]['shadowLine2']
    controlPoints = tuple(w.coords(edge)[2:len(w.coords(edge))-2])
    xyPairedControlPoints = 
     tuple(zip(controlPoints[0::2],controlPoints[1::2]))
    w.coords(shadowLine1,w.coords(edge)[-4],w.coords(edge)[-3],ex,ey)
    if controlPoints:
        w.coords(shadowLine2, w.coords(edge)[-6],w.coords(edge)[-5],ex,ey)
    else:
        w.coords(shadowLine2,
        w.coords(edge)[0],w.coords(edge)[1],w.coords(edge)[0],w.coords(edge)[
        1])
    #for actual adding and removing controlpoints
    for node in potentNodes:
        for (key,val),c in zip(nodesDict[node].items(),const):
            box = w.find_overlapping(val[0]+c[0],val[1]+c[1], val[0] - c[0],
             val[1] - c[1])
            #w.create_rectangle(val[0]+c[0],val[1]+c[1], val[0] - c[0],
            val[1] - c[1], outline = 'blue')
            oldbox = w.find_overlapping(val[0],val[1],val[0]-2*c[0],val[1]-
            2*c[1])
            #w.create_rectangle(val[0],val[1],val[0]-2*c[0],val[1]-2*c[1],
            outline = 'green')
            boxx = w.find_overlapping(val[0] + 2*c[0], val[1] + 2*c[1],
            val[0] - c[0], val[1] - c[1])
            #w.create_rectangle(val[0] + 2*c[0], val[1] + 2*c[1], val[0] -
            c[0], val[1] - c[1], outline = 'red')
            if (shadowLine1 in box) and (node not in lineEndTipBox) and (val
             not in xyPairedControlPoints) and ((val[0]-c[0], val[1]-c[1])
             not in xyPairedControlPoints) :
                controlPoints += val
                xyPairedControlPoints += (val,)
                nodesDict[node][key] = (val[0]+c[0], val[1]+c[1])
            if xyPairedControlPoints and ((shadowLine2 in boxx and
               shadowLine2 not in oldbox) or (node in lineEndTipBox)) and
               (val[0]-c[0], val[1]-c[1]) in xyPairedControlPoints:
                controlPoints = controlPoints[0:len(controlPoints)-2]
                nodesDict[node][key] = (val[0]-c[0], val[1]-c[1])
    return controlPoints

def dragEdge(e):
    global startNode,endNode,newEdge,ControlPoints,edgesDict
    if startNode != None and endNode == None:
        x0,y0 = w.coords(newEdge)[0],w.coords(newEdge)[1]
        lineEndTipBox = w.find_overlapping(e.x-5,e.y-5,e.x+5,e.y+5)
        ControlPoints = addremoveControlPoints(newEdge,e.x,e.y,lineEndTipBox)
        w.coords(newEdge,(x0,y0)+ControlPoints+(e.x,e.y))
        for Object in lineEndTipBox:
            if Object in edgesDict[newEdge]['controlnodes']:
                endNode = Object
                ControlPoints =
               addremoveControlPoints(newEdge,e.x,e.y,lineEndTipBox)
               w.coords(newEdge,(x0,y0)+ControlPoints+(w.coords(endNode)[0]+2
               0,w.coords(endNode)[1]+20))

def releaseEdge(e):
    global startNode,endNode,newedge,edgesDict,nodesDict,ControlPoints,const
    if startNode:
        w.itemconfig(startNode, tags = 'nodes')
        if endNode:
            for nodePair in handshakes.values():
                if nodePair == (startNode,endNode) or nodePair ==
                 (endNode,startNode):
                    if ControlPoints:
                        xyPairedControlPoints =
                       tuple(zip(ControlPoints[0::2],ControlPoints[1::2]))
                        for node in edgesDict[newEdge]['controlnodes']:
                            for (key,val),c in
                              zip(nodesDict[node].items(),const):
                                 if ((val[0]-c[0], val[1]-c[1]) in
                                  xyPairedControlPoints):
                                     nodesDict[node][key] = (val[0]-c[0],
                                    val[1]-c[1])

                    w.delete(edgesDict[newEdge]['shadowLine1'],edgesDict[newE
                    dge]['shadowLine2'])
                    del edgesDict[newEdge]
                    x = edges.pop()
                    w.delete(x)
                    startNode = None
                    endNode = None
                    return
            handshakes.update({newEdge:(startNode, endNode)})
            print(handshakes)
            startNode = None
            endNode = None
        elif endNode == None:
            if ControlPoints:
                xyPairedControlPoints =
                 tuple(zip(ControlPoints[0::2],ControlPoints[1::2]))
                for node in edgesDict[newEdge]['controlnodes']:
                    for (key,val),c in zip(nodesDict[node].items(),const):
                        if ((val[0]-c[0], val[1]-c[1]) in
                         xyPairedControlPoints):
                            nodesDict[node][key] = (val[0]-c[0], val[1]-c[1])

            w.delete(edgesDict[newEdge]['shadowLine1'],edgesDict[newEdge]['sha
            dowLine2'])
            del edgesDict[newEdge]
            x = edges.pop()
            w.delete(x)
            startNode = None

def chooseNode(e):
    global edgesDict, handshakes
    if w.type('current') == 'oval':
        w.addtag_withtag('actNode','current')
        node = w.find_withtag('current')[0]
        x0, y0, x1, y1 = w.coords(node)
        for edge in handshakes.keys():
            if node in handshakes[edge]:
                if w.coords(edge)[0] == x0+20  and w.coords(edge)[1] ==
                 y0+20:
                    coords = w.coords(edge)
                    coords = list(zip(coords[0::2],coords[1::2]))
                    coords.reverse()
                    newcoords = ()
                    for coord in coords:
                        newcoords += coord
                    w.coords(edge, newcoords)
                    w.itemconfig(edge, tags = ('edges','actEdge'))
                if w.coords(edge)[-2] == x1-20  and w.coords(edge)[-1] == y1-
                 20:
                    w.itemconfig(edge, tags = ('edges','actEdge'))
        for edge in w.find_withtag('actEdge'):
            edgesDict[edge]['controlnodes'] = [node for node in
            w.find_withtag('nodes') if (node not in handshakes[edge])]
def moveNode(e):
    w.coords('actNode', e.x-20,e.y-20,e.x+20,e.y+20)
    for edge in w.find_withtag('actEdge'):
        controlPoints =
         addremoveControlPoints(edge,e.x,e.y,w.find_withtag('actNode'))
        w.coords(edge,
         (w.coords(edge)[0],w.coords(edge)[1])+controlPoints+(e.x,e.y))
def releaseMovingNode(e):
    node = w.find_withtag('actNode')[0]
    x0,y0,x1,y1 = w.coords(node)
    centerx,centery = x0+20,y0+20
    nodesDict[node] = {'topleft':(x0,y0), 'top':(centerx,centery-
                      30),'topright':(x1,y0),
                      'right':(centerx+30,centery),'bottomright':(x1,y1),
                      'bottom':(centerx,centery+30), 'bottomleft':(x0,y1),
                      'left':(centerx-30,centery)}
    w.dtag('actNode','actNode')
    w.dtag('actEdge','actEdge')


w.bind('<Button-1>', createNode)
w.bind('<ButtonRelease-1>', releaseNode)
w.bind('<B1-Motion>', dragNode)
w.bind('<Button-3>', createEdge)
w.bind('<B3-Motion>', dragEdge)
w.bind('<ButtonRelease-3>', releaseEdge)
w.bind('<Button-2>', chooseNode)
w.bind('<B2-Motion>', moveNode)
w.bind('<ButtonRelease-2>', releaseMovingNode)
window.mainloop()
```