```
# Tested working with testman, 2/2

from bblist import *
from schematic import schematic
from formulate import solve, autosolve
import random, sys

if __name__ == '__main__':

    # Another case study for the solver: an N-channel remote.
    # Build strategy is enforcing uC and LED to be together, while allowing
    # large bridges betwen battery and bat, as well as bat and uC.

    # Modified to take arguments. First argument is the channel #,
    # second argument is the workspace square.

    #n = int(sys.argv[1])
    #ws = (int(sys.argv[2]), int(sys.argv[2]), 2)
    n = 6
    ws = (10, 7, 2)
    problem = schematic(ws, name = ("Remote-" + str(n)))
    print "Making " + problem.name + " in " + str(ws)

    lip = Lipo("LiPo")

    for chan in range(1, n+1):
        but = But("Button " + str(chan))
        led = Led("LED " + str(chan))
        uc = Micro("uC " + str(chan))
        problem.addConnection(lip, 'PWR', but, 'B1', 3)
        problem.addConnection(but, 'B2', uc, 'PWR', 3)
        problem.addConnection(uc, 'OUT2', led, 'LED')

    # Call the solver, which positions the Bitblox correctly.

    fixed = autosolve(problem)

    # Draw the assembly resulting from the solution.
    fixed.showSolution()
```