

Re: Simulating simple electric circuits

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2007-05/msg02055.html>

- *From:* Bjoern Schliessmann <usenet-mail-0306.20.chr0n0ss@xxxxxxxxxxxxxxxxxxx>
 - *Date:* Mon, 14 May 2007 22:19:43 +0200
-

Arnaud Delobelle wrote:

I'm interested! I was tempted to have a go at it after your initial post, it sounded like a nice little project :)

Here you go, see below (quite long). It works from Python 2.5 and above and should be straightly executable from command line.

(It'll work with lower Python versions if you backtranslate the few "A if (condition) else B" parts.)

We have:

- class CurrentController which does the main work
- classes Source and Ground for convenience when creating circuits
- class Relay for the realisation of logic

Please have a look at the example at the end. It's taken from my actual project. A relay "ZT" is brought up if an outside key is pressed. This forces the normally-"up" latching relay "ZTS" down. This in turn causes relay "FTS" to go "up" (which causes many other complicated stuff).

I'm open for questions and/or suggestions. :) The names used and docstrings may sound a bit strange since I hastily translated the source from german (we have some very special terms regarding relay technology).

I suspect that this is a dead end though, because in more complex designs the creation of circuits becomes cumbersome. Also, for the system I want to model, the circuits are OOH very complex and OTOH I don't have access to all circuit diagrams. :(So I think I'll try next to transfer my problem to digital two-level logic.

I'll go with Dave Baum's suggestion with the event queue and post again here soon with report of my achievements ...

Regards,

Re: Simulating simple electric circuits

Björn

---snip---

```
#!/usr/bin/env python
```

```
class CurrentController(object):
```

```
    def __init__(self):
```

```
        self.currentSources = { }
```

```
        # Contents: {ID1: source1, ID2: source2, ...}
```

```
        self.circuits = { }
```

```
        # Contents: {ID1: [all circuits starting at source 1],  
ID2: [...], ...}
```

```
    def newID(self):
```

```
        ID = 0
```

```
        while ID in self.currentSources:
```

```
            ID += 1
```

```
        return ID
```

```
    def newSource(self, func = None):
```

```
        id = self.newID()
```

```
        s = Source(id, func)
```

```
        self.currentSources[id] = s
```

```
        return s
```

```
    def newGround(self):
```

```
        e = Ground()
```

```
        return e
```

```
    def newRelay(self, name, **argv):
```

```
        r = Relay(name, **argv)
```

```
        return r
```

```
    def changed(self, relay):
```

```
        """
```

```
        Relay relay changed. Re-try all circuits containing it.
```

```
        """
```

```
        print "CurrentController: changed:", relay.name, "up" if relay.up else "down"
```

```
        for ID, circuits in self.circuits.items():
```

```
            for circuit in circuits:
```

```
                if circuit.contains(relay):
```

```
                    circuit.test()
```

```
    def showCircuits(self, activeOnly = False):
```

```
        """
```

```
        Show all circuits.
```

```
        If activeOnly is True, show only active ones.
```

Re: Simulating simple electric circuits

Re: Simulating simple electric circuits

```
"""
print "| Circuits: "
for ID, circuits in self.circuits.items():
    print "| ID: %i" % ID
    for circuit in circuits:
        if (not activeOnly) or circuit.active:
            print circuit

def start(self):
    """
    All circuits are defined -- try them out and start
    monitoring changes.
    """
    for ID, source in self.currentSources.items():
        source.start()

    for ID, circuits in self.circuits.items():
        for circuit in circuits:
            circuit.test()

def toGround(self, telegram):
    """
    Save telegram as valid circuit and monitor it.

    Called from the outside if a telegram reaches "ground"
    (i. e., circuit closed).
    """
    try:
        self.circuits[telegram.ID].append(telegram)
    except KeyError:
        self.circuits[telegram.ID] = [telegram]

class CurrentTelegram(object):
    """
    Class for recording and managing a circuit.
    """
    usedIDs = []

    def __init__(self, id):
        self.relays = []
        # Contents: [(group name, relay object, coil function), ...]
        self.switches = []
        # Format: [(group name, relay object, switch function), ...]

        self.ID = id # ID of the source of this telegram
        self.circuitID = self.newID() # unique telegram ID

        # default priority is 1. Should always be > 0.
```

Re: Simulating simple electric circuits

```
# It's used for weighing of multiple currents, see Relay class.
self.prio = 1
self.active = False

def newID(self):
    ID = 0
    while ID in self.usedIDs:
        ID += 1
    self.usedIDs.append(ID)
    return ID

def __str__(self):
    relays = [""] # for an extra newline at the beginning of the string to return
    for group, r, f in self.relays:
        type = repr(f)
        temp = "||| Group %s, Relay %s %s" % (group, r.name, type)
        relays.append(temp)
    relays = "\n".join(relays)

    switches = [""]
    for group, r, f in self.switches:
        type = repr(f)
        temp = "||| Group %s, Relay %s %s" % (group, r.name, type)
        switches.append(temp)
    switches = "\n".join(switches)

    return ""||| CurrentTelegram
    || Source %i
    || Relays: %s
    || Switches: %s
    || %sactive
    ||"" % (self.ID, relays, switches, "" if self.active else "not ")

def __repr__(self):
    return self.__str__()

def copy(self):
    """
    Generate and return a copy of this telegram
    (for current "forks")
    """
    other = CurrentTelegram(self.ID)
    other.relays = self.relays[:]
    other.switches = self.switches[:]
    other.prio = self.prio
    other.active = self.active
    # self.active should at this point normally be always False
    # circuits aren't added after CurrentController.start is
    # called
    return other
```

Re: Simulating simple electric circuits

```
def addSwitch(self, byWho, relay, f):
    """
    Add a switch to the circuit. Parameters:
    – byWho: Name of caller (for information only)
    – relay: Relay object
    – f: A callable. If it returns True or equivalent,
    switch is "closed", else "open".
    """
    self.switches.append((byWho, relay, f))

def addRelay(self, byWho, relay, f):
    """
    Add a relay to the circuit. Parameters:
    – byWho: Name of caller (for information only)
    – relay: Relay object
    – f: A callable ("coil function"). It's called
    as f(True) if the circuit gets active,
    and f(False) if it becomes inactive.
    """
    self.relays.append((byWho, relay, f))

def contains(self, relay):
    return self.containsRelay(relay) or self.containsSwitch(relay)

def containsRelay(self, r):
    for _, relay, __ in self.relays:
        if relay is r:
            return True
    else:
        return False

def containsSwitch(self, r):
    for _, relay, __ in self.switches:
        if relay is r:
            return True
    else:
        return False

def mayBecomeActive(self):
    """
    Determine if circuit is "closed" (i. e. all switches
    are closed so current can flow).
    """
    for name, relay, f in self.switches:
        if not f():
            break
    else:
        return True
    return False

def becomeActive(self):
```

Re: Simulating simple electric circuits

```
"""
Make circuit active: Give "current" to all relays
in this circuit.
"""
if self.active: return
self.active = True

for name, relay, f in self.relays:
# Also give circuitID for relay to know who activated it
f(True, self.circuitID)

def stop(self):
"""
Make circuit inactive: Relay falls
"""
if not self.active: return
self.active = False

for name, relay, f in self.relays:
# Also give circuitID for relay to know who deactivated it
f(False, self.circuitID)

def test(self):
if self.mayBecomeActive():
self.becomeActive()
else:
self.stop()

# TODO: Eliminate circle currents

currentController = CurrentController()

class Source(object):
def __init__(self, ID, neighbour = None):
self.connections = []
self.ID = ID
if neighbour: self.verbindungen.append(neighbour)

def start(self):
"""
Create CurrentTelegrams and send them to
all connected.
"""
for c in self.connections:
t = CurrentTelegram(self.ID)
c(t)

class Ground(object):
c = currentController
def input(self, t):
```

Re: Simulating simple electric circuits

```
self.c.toGround(t)

class Relay(object):
    c = currentController
    """
    Continuous current relay/latching relay

    TODO: Delay!
    """
    def __init__(self, name, latching = False, initiallyUp = False, minStrength = 1):
        self.up = False if not initiallyUp else True
        self.name = name
        self.latching = True if latching else False

        # In here, pairs "Circuit-ID: strength" of currents through
        # raising coil are saved
        self.currents = {}

        # This helps decide when Relay goes to "up":
        # If the sum of all "flowing" strengths in
        # self.currents is >= minStrength.
        #
        # Example: minStrength = 2.
        # => The relay will go to "up" if at least
        # - one current of strength "2"
        # - two currents of strength "1" each
        # - ...
        # are active.

        self.minStrength = minStrength

    def strengthBigEnough(self):
        """
        Test the magnitude of the sum of all currents through
        the "raise" coil. If it's big enough, return True,
        else False.
        """
        total = 0
        for _,strength in self.currents.items():
            total += strength

        return True if total >= self.minStrength else False

    def normalSw(self):
        """
        Normal switch -- is closed if relay is "up",
        else open.
        """
        return True if self.up else False

    def reverseSw(self):
```

Re: Simulating simple electric circuits

```
"""
Reverse switch -- opens if relay is "up",
else it's closed.
"""
return True if not self.up else False

def normalCoil(self, status, circuitID, strength = 1):
    """
    Send a "current" through the normal coil.

    Relay goes "up" if status is True or equivalent.

    Else, if no currents remain and relay is no
    latching relay, it goes to "down".
    """
    if status:
        assert circuitID not in self.currents
        self.currents[circuitID] = strength
        if self.strengthBigEnough():
            self.goUp()
        else:
            del self.currents[circuitID]

    if not self.strengthBigEnough():
        if not self.latching:
            self.goDown()

def forceDownCoil(self, status, circuitID):
    """
    Relay goes down, no matter what.
    """
    if status:
        self.goDown()
    # CurrentController will be notified of change here
    # and will retry all circuits through normal coil,
    # so it's okay to ignore self.currents here

def goUp(self):
    if not self.up:
        self.up = True
        self.c.changed(self)

def goDown(self):
    if self.up:
        self.up = False
        self.c.changed(self)

def __str__(self):
    return "Relay %s: %s%s (Currents: %r)" % (self.name,
        "up" if self.up else "down",
        " (latching relay)" if self.latching else "",
```

Re: Simulating simple electric circuits

```
self.currents)
def __repr__(self):
return self.__str__()

# following not in main method for usability of "python -i"

# create some relays
ZT = currentController.newRelay("ZT")
FTS = currentController.newRelay("FTS", latching = True)
ZTS = currentController.newRelay("ZTS", latching = True, initiallyUp = True)

# create one source and one ground
s = currentController.newSource()
g = currentController.newGround()
name = "tester"

def circuit1(t):
t.addSwitch(name, ZT, ZT.normalSw)

u = t.copy() # fork!
u.addRelay(name, ZTS, ZTS.forceDownCoil)
g.input(u)

t.addSwitch(name, ZTS, ZTS.reverseSw)
t.addRelay(name, FTS, FTS.normalCoil)
g.input(t)

s.connections.append(circuit1)

# example circuit:
#
# V (+)
# |_____
# ||
# |_| ZT sw (closes when up) |_| ZTS sw (opens when up)
# ||
# | ^ |
# O | ZTS ("force down" coil) O FTS (normal coil)
# | ("up" at beginning) |
# ||
# ---- GND ---- GND
#

def main():
currentController.start()
print "Putting ZT up with my finger ..."
ZT.goUp()

currentController.showCircuits()

if __name__ == "__main__":
```

Re: Simulating simple electric circuits

main()

----snap----

--

BOFH excuse #401:

Sales staff sold a product we don't offer.

.