```python
# File komega-v08e.py

# Author: G.Doeben-Henisch
# First date: September 4, 2020
# Last change: 11.October 2020, 4:30 pm

############################################
# TOPIC OF NEW CHANGE
'''
Rebuilt the structure of the simulation cycle using a modified PV-state (8b) and a modified rule state (8c) with new Simulation (8d,e)
'''

##################################
# Execution Environment of my local machine:
# (venv) gerd@gerd-ub2:~/env/komega/tst$ python3 komega-v01d.py
#

####################################
# GITHUB
#
# We use a github repository at:
# https://github.com/szmt/komega.git
#
# I am working from a unix-shell using the following github-commands:
# https://git-scm.com/docs/git

######################################
# BACKGROUND THEORY
#
# This code is a translation of a theory described in the blog
# https://www.uffmm.org
#
# There exists meanwhile additionally a German webpage with a comprehensive text including
# epistemology, theory of shared knowledge generation with simulation as well
# as assessment of goals within this process:
'''
URL: https://www.cognitiveagent.org/2020/10/02/mensch-mensch-computer-gemeinsam-planen-und-lernen-erste-notizen/
'''
# This German text covers only a subpart of the uffmm.org website

#############################################
# HMI - CLASS PUBLISH and STORAGE
#
'''
The intended interaction of the user with the system will be realized through an interactive web page. In this experimental program there is no web page but a normal console. Therefore (proposal from Tobias Schmitt) we have a special class 'Publish' which handles all console input and output and the other classes interact with this class Publish. In the context of the web server we can then replace the class Publish by appropriate libraries for HTML web pages.
```

```
'''
######################################
# ACTOR STORY
#
# In the specifications an actor story [AS] has been specified. This AS requires # some basic states which are dedicated for certain tasks to do:

'''
ACTOR STORY

S1: START
S2: EDIT P and V (Problem and Vision description)
S3: EDIT S (actual state)
S4: EDIT X (change rules)
S5: SIMULATION (Applying X to S)
S6: EVALUATION (After the simulation)
S7: STOP

'''

# MAIN IDEA
'''
According to the above mentioned actor story the user will be sitting in front of a system interface [SI] which works first only as a console.
In the beginning the user is placed in a start state S1 showing all options available.
The user can select one of these options and can from start state S1 reach all other states S2-S7.
'''

#############################
# IMPORTS

#############################
# SUPPORTING FUNCTION
#

# No isolated funtions yet, only functions as part of classes.

# CLASSES
#
'''
For every state there exists one working class to do the job.
'''

import kcv8e as kc     #The theory-related classes

###########################
# Main Programm
#


###############
# Start main loop
```

```
#
# The loop will work as long as the value of the variable 'loop' is different to 'N'

loop='Y'
while loop=='Y':

################################
# STATE 1 : START
# Show available options
# Get feedback for selection
# Confirm the selection
# Move to different states

# SHOW ALL AVAILABLE OPTIONS

        kc.ast.menushow()

# Ask back for selection number
        message='Enter a Number [1-7] for Menu Option \n'
        kc.pub.userinput(message)

# Evaluate the selection

        opt=kc.pub.opt
        kc.ast.badoption(opt)

####################################################################
# Call to state Edit Problem P and Vision V
################################
# STATE 2 : EDIT P and V
# Ask Questions related to P and V
# Collect all answers into one problem-vision document
#

        if opt=='2':
                # Where You are
                kc.pub.show(kc.ap)

                #Interaction with Problem Class
                # OVERVIEW ABOUT TOPICS

                message='You will be asked to the following topics:\n'
                kc.pub.useroutput(message)

                kc.app.menushow()

                # DOCUMENTS SO FAR
                kc.app.getpvlist()

                # ENTER PROBLEM

                message='\n Enter a NAME for your problem\n'
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.app.getpname(inp)

                message='\n Enter your PROBLEM in plain text\n'
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.app.getproblem(inp)

                message='\n Enter your VISION of a better state in the future in plain text\n'
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.app.getvision(inp)

                message='\n Enter the NAME of the CITY or REGION you are in\n'
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.app.getregion(inp)


                message='\n TIME model [From, Until,Cycleunit [Y or M or D or H]]\n'
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.app.gettime(inp)

                message='\n Which kinds of PERSONS (individuals or roles) are important? Write a
list, comma separated please :\n '
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.app.getperson(inp)

                ########################################
                # Show final document as dictionary

                kc.app.problemTotal()


####################################################################
# Call to state Edit Actual State S
################################
# STATE 3 : EDIT S
# Collect single expressions
# Collect all expressions into one  document describing S
# The document S is organized as a set of expressions!
# In the DB every S document is associated with a name.

        elif opt=='3':
                # Where You are
                kc.pub.show(kc.ass)

                # Set document S to zero
                kc.aas.emptydocs()
```

```
# Actual list of state descriptions

kc.aas.getslist()

# Ask for a document S to be loaded
message="Do You want to load a document S? [Y,N]\n"
kc.pub.userinput(message)
inp=kc.pub.opt
if inp == 'Y':
        message='Enter the name of the wanted document:\n'
        kc.pub.userinput(message)
        fname=kc.pub.opt
        kc.ass.stateName=fname
        #############################################
        # LOAD
        docs=set()
        docs=kc.st.d[fname]
        message='This is the content of the document:\n'
        kc.pub.useroutput(message)
        kc.pub.useroutput(str(docs))
        kc.aas.stateAll=docs
else:
        message='Enter a NAME for the new state description:\n'
        kc.pub.userinput(message)
        fname=kc.pub.opt
        kc.aas.getsname(fname)

Sloop='Y'
while Sloop=='Y':
        # Interaction with actual state S document
   message='Enter an expression for your state description  in plain text : \n'
   kc.pub.userinput(message)
   inp=kc.pub.opt
   kc.aas.getexpression(inp)
   message="STOP Editing S  != Y', CONTINUE = 'Y'  \n"
   kc.pub.userinput(message)
   inp=kc.pub.opt
   Sloop=inp

######################################################
# Keeping the document

message='Your final State Description document is now :\n'
kc.pub.useroutput(message)
docs=kc.aas.stateAll
kc.pub.useroutput(docs)

##############################################
# STORE DOCUMENT PERMANENTLY

kc.aas.storeSdocument()
```

```
#
####################################################################
# Call to state Edit Change Rules X
################################
# STATE 4 : EDIT X
# Collect single expressions for change rules
# Collect all expressions into one  document describing X
#
##################################################
# FORMAT OF A RULE WITHOUT ACTOR
#
# IF: CONDITION THEN: PROBABILITY - E-MINUS - E-PLUS
###############################################################
#V0:.1 Expression......[0,1]......1 Expression .1 Expression
#
# In the first version we assume the most simple case which is possible!

    elif opt=='4':
        # Where You are
        kc.pub.show(kc.ax)

        # A list of all X-documents so far
        kc.axx.getxlist()

        # Ask for a document X to be loaded
        message="Do You want to load a document X? [Y,N]\n"
        kc.pub.userinput(message)
        fileinp=kc.pub.opt
        if fileinp == 'Y':
            message='Enter the name of the wanted document:\n'
            kc.pub.userinput(message)
            fname=kc.pub.opt
            kc.axx.ruleDocName=fname
            ##############################################
            # LOAD
            kc.axx.getdocx(fname)
            message='Your Rules  document is as follows :\n'
            kc.pub.useroutput(message)
            docx=kc.axx.rulesAll
            kc.pub.useroutput(docx)
        else:
            message='Enter the name of the new rules document:\n'
            kc.pub.userinput(message)
            fname=kc.pub.opt
            kc.axx.ruleDocName=fname

        # Set AllRules to zero
        if fileinp != 'Y':
            kc.axx.AllToZero()

        XAllLoop='Y'
```

```
while XAllLoop=='Y':

        # Set Rule to zero
        kc.axx.RuleToZero()

        # Set Condition to zero
        kc.axx.CondToZero()

        # Generate the Condition as a set
        XCondLoop='Y'
        while XCondLoop=='Y':
                message=kc.axx.rcat[0]+' : \n' #Shows CONDITION
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.axx.getcond(inp)

                message="CONTINUE  Editing Condition = 'Y', STOP != 'Y'  \n"
                kc.pub.userinput(message)
                inp=kc.pub.opt
                XCondLoop=inp

        # Append Condition to rule
        kc.axx.rule.append(kc.axx.cond)

        # Get the Probability
        # Set Probability to Zero
        kc.axx.ProbToZero()

        # Generate Probability and Append to rule
        message="Enter a probability between 0.0 and 1.0 \n"
        kc.pub.userinput(message)
        inp=kc.pub.opt
        kc.axx.getprob(inp)

        # Set EMinus to zero
        kc.axx.EminusToZero()

        # Generate the set EMinus
        XEMinusLoop='Y'
        while XEMinusLoop=='Y':
                message=kc.axx.rcat[2]+' : \n' #Shows EMinus
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.axx.geteminus(inp)
                message="CONTINUE  Editing EMinus = 'Y', STOP != 'Y'  \n"
                kc.pub.userinput(message)
                inp=kc.pub.opt
                XEMinusLoop=inp

        # Append EMinus to rule
        kc.axx.rule.append(kc.axx.eminus)
```

```
            # Set EPlus to zero
            kc.axx.EplusToZero()

            # Generate the set EPlus
            XEPlusLoop='Y'
            while XEPlusLoop=='Y':
                message=kc.axx.rcat[3]+' : \n' #Shows EPlus
                kc.pub.userinput(message)
                inp=kc.pub.opt
                kc.axx.geteplus(inp)
                message="CONTINUE  Editing EPlus = 'Y', STOP != 'Y'  \n"
                kc.pub.userinput(message)
                inp=kc.pub.opt
                XEPlusLoop=inp

            # Append EPlus to rule
            kc.axx.rule.append(kc.axx.eplus)

            # Convert the rule into a dictionary
            ruledict=dict(zip(kc.axx.rcat, kc.axx.rule))

            # Add the new rule-dictionary to rulesAll
            kc.axx.rulesAll.append(ruledict)

            # Shwo all rules so far
            message='Your Rules  document with name '+str(kc.axx.ruleDocName)+' is 
now :\n'
            kc.pub.useroutput(message)
            docx=kc.axx.rulesAll
            kc.pub.useroutput(docx)

            # Asking for Continuatioin with another rule
            message="CONTINUE  Editing X = 'Y', STOP != 'Y'  \n"
            kc.pub.userinput(message)
            inp=kc.pub.opt
            XAllLoop=inp

        #####################################################
        # Store the document

        kc.axx.storeFinal(docx)

#
##################################################################
# Call to state  Simulation SIM
###############################
# STATE 5 : Run the simulation
# PREPARATION
# (1) Take a state description S and an appropriate set of  change rules X
# (either actually edited or from a file)
# SIMULATION CYCLE
# (2) Select all rules X* whose conditions are fulfilled by S.
# (3) For each rule x in X*:
# (3.1) Apply the E-Minus part and remove the E-Minus expression from S
# (3.2) Apply the E-Plus part and add th e-Plus expressions to S
# (3.3) Show the new version of S after applying X* to S
# (3.4) If no Stop then repeat from (2)

        elif opt=='5':
                kc.pub.show(kc.asim)

# PREPARATION
# (1) Take a state description S and an appropriate set of  change rules X
# from a file
# Ask for a document S to be loaded

                message="Do You want to load a state description S? [Y,N]\n"
                kc.pub.userinput(message)
                inp=kc.pub.opt
                if inp == 'Y':
                        kc.st.loaddata()
                        message='Your State Description document is as follows :\n'
                        kc.pub.useroutput(message)
                        kc.assim.s=kc.aas.stateAll
                        kc.pub.useroutput(kc.assim.s)

# Ask for a document X to be loaded
                message="Do You want to load a rule document X? [Y,N]\n"
                kc.pub.userinput(message)
                inp=kc.pub.opt
                if inp == 'Y':
                        kc.st.loaddatar()
                        message='Your Rules  document is as follows :\n'
                        kc.pub.useroutput(message)
                        kc.assim.x=kc.axx.rulesAll
                        kc.pub.useroutput(kc.assim.x)

# SIMULATION CYCLE

                s=kc.assim.s
                x=kc.assim.x

                XSimLoop='Y'
                while XSimLoop=='Y':
                        kc.assim.xapply(s,x)
                        s=kc.aas.stateAll
                        message='New set S : \n'
                        kc.pub.useroutput(message)
                        kc.pub.useroutput(s)

                        # Asking for Continuatioin with the simulation
                        message="CONTINUE  Simulation = 'Y', STOP != 'Y'  \n"
                        kc.pub.userinput(message)
                        inp=kc.pub.opt
                        XSimLoop=inp


#
##########################################################################
# Call to state  Evaluation EV
#

        elif opt=='6':
                kc.pub.show(kc.aev)

#
##########################################################################
# Call to state Stop STP
#

        elif opt=='7':
                kc.pub.show(kc.astp)

####################################################################
# End of Loop



# Clarify how to continue

        message="STOP MAIN LOOP != 'Y', CONTINUE = 'Y'  \n"
        kc.pub.userinput(message)
        inp=kc.pub.opt
        loop=inp
```