CogLogo manual

François Suro

LIRMM - Laboratoire d'Informatique, Robotique et Microélectronique de Montpellier
Université de Montpellier - CNRS
161 Rue Ada, 34090 Montpellier, France
suro@lirmm.fr

March 1, 2020

Contents

1 Introduction

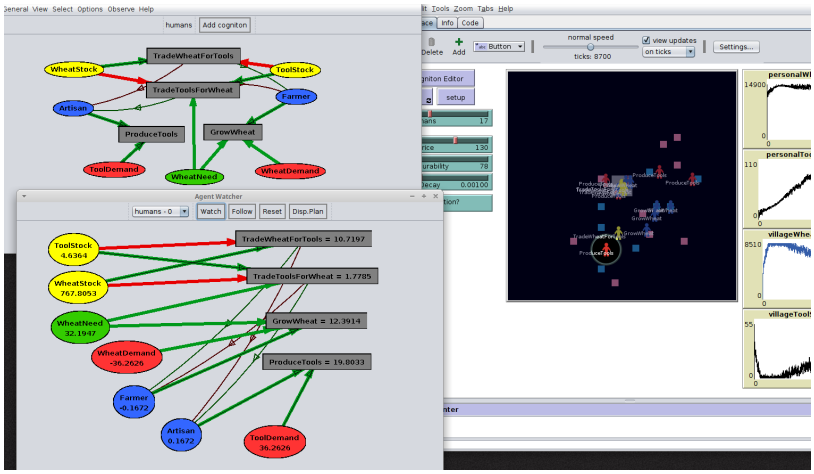


Figure 1: On the right the NetLogo window, on the top left the Cognitive Scheme editor, on the bottom left the agent watcher displaying the state of the cognitons and the calculated values of the plan in an agent's mind

CogLogo [10] is a NetLogo [11] extension written in Java which provides an implementation of the cogniton architecture described in the MetaCiv Meta-Model.

CogLogo and MetaCiv is the result of collective work and research by the SMILE team [1], and contains extracts and translations from the following works :

- Agent-Group-Role [6, 2, 3]
- Agent-Group-Role-Environment [4]
- MASQ [8, 9]
- MetaCiv [5]

2 MetaCiv

MetaCiv is a generic framework for modelling and simulating complex human socio-cognitive systems. It leaves complete freedom of implementation of the environment and agent behaviour and focus on the decision process, taking into account the main aspects when modelling human societies: individual and social, cultural and environmental.

The structural principles that guided the design of MetaCiv are:

- Structure the socio-cognitive items used in social simulations by following the conceptual framework of MASQ.
- Consider both reactive and cognitive aspects to model the agent mind by using a hybrid agent architecture based on *cognitons* [1].

[1] `https://www.lirmm.fr/lirmm_eng/research/teams/smile`

- Integrate social and cultural aspects of interaction by using an extension of the AGR model which merges the cultural aspects of groups (norms, standards, shared beliefs) with individual aspects at the decision making level of each agent. This allows a modeller to have a vision of culture and norms as a source to influence the behavior of the agent and not as regimented or compulsory system, as is usually the case for normative agents [8].

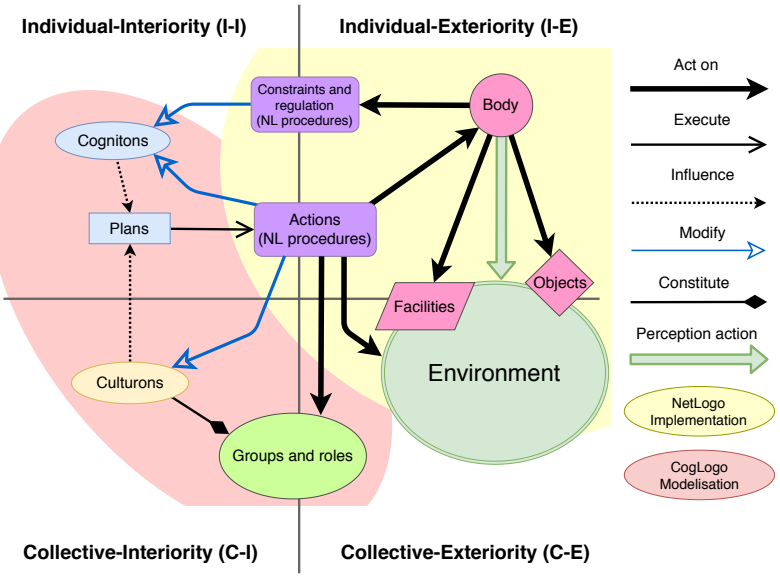


Figure 2: MetaCiv elements and their mapping on the MASQ-quadrants

Figure 2 shows the overall representation of MetaCiv and its essential features depicted using MASQ-ML, the graphical modelling language of MASQ [7].

2.1 Cogniton-based agent architecture

In MetaCiv, according to MASQ principles, each agent has a physical body (quadrant I-E) which is separated from its mind (quadrant I-I). The mind of an agent is built using an architecture based on cognitons [1], thus unifying cognitive and reactive aspects in the same form. We qualify the architecture of unifying in that it differs from hybrid architectures by not distinguishing a reactive level and a cognitive level, but incorporating all factors allowing an agent to make a decision such as beliefs, percepts, skills, norms, etc. Although this type of architecture is not new *per se* (it was introduced in the early '90s to model reactive agents), it draws its power from its ability to integrate the individual and collective aspects by simply using the same representation.

This architecture consists of two basic elements: *cognitons* and *plans*.

- The cognitons are "basic cognitive units" [1] that can be dynamically added or removed from the mind of the agent. Each cogniton has its own weight in the agent's mind that can evolve during the lifetime of the agent. The cognitons can represent beliefs, knowledge or percepts. All the cognitons present in the mind of an agent at a given moment form its mental state.

- The plans contain the different activities that an agent can perform. Each plan consists of a sequence of actions of varying complexity. To guide the decision making process of the agent of which plan to execute, we assign to each plan a weight determined by the cognitons that can influence it. The choice of the plan can then be according to various strategies: in terms of the largest weight, perform a random draw that favours the strong weight, etc.

In order to compute the weight of a plan, we propagate the weights of the cognitons to each plan through *influence links*. The total weight of a plan can be computed using the formula:

$$weight = \sum_i C_i L_i \tag{1}$$

where i is an influence link, C_i is the weight of the cogniton linked by i to the plan, and L_i is the influence value of the link. The sum is performed on all the influence links to the cognitons that affect the plan. The influence of a cogniton on the plan is therefore described by a multiplicative function. A high weight amplifies the influence of a cogniton, while a low weight reduces it.

Figure 3 illustrates the use of cognitons in a simple context. In this example the cognitons A and B have links that affect the weights of Plan1 and Plan2. Plan1 is influenced only by the cogniton A of weight 3 and the weight of the influence link is 4. Plan2 is affected by both A and B of weights 3 and 2 respectively for which the weights of the influence links are -1 and 6. The final weights of Plan1 and Plan2 are respectively 12 and 9.

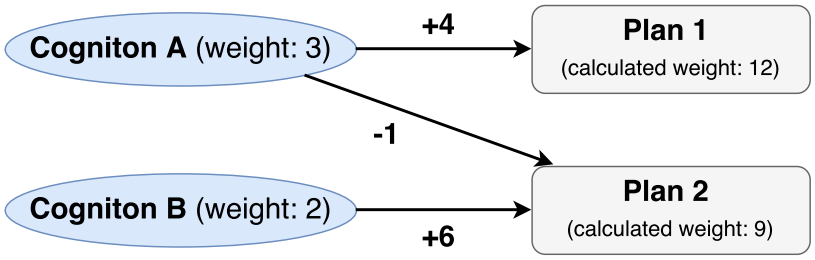


Figure 3: An example of the influence mechanism.

A plan allows an agent to act on its body, on the objects it manipulates, on the surrounding environment or on the groups to which it belongs. It may also affect the mind of the agent by acting directly on the agent's cognitons (through deletion, addition or modification of its weight). This feedback creates a reinforcement mechanism that strengthens or weakens the weights of the cognitons that contributed to the selection of the current plan.

2.2 Groups and culturons

The MetaCiv meta-model extends AGR [3] by associating to each role within a group a set of culturons that represent cultural elements common to all agents playing that role. The culturons work similarly to cognitons: when an agent plays a role, it assimilates all the culturons associated to the role, which will then act on the weights of the plans from its mind, like the cognitons. Figure 4 illustrates this mechanism. In this example the agent belongs to the group α and plays the role β which contains two culturons, X and Y. By playing the role β, Plan2 will also be influenced by the culturons X and Y that alter its total weight.

As in AGR, an agent can belong to several groups and play many roles. All culturons from all the roles played by the agent are taken into account in calculating the weight of the plans. But unlike cognitons, the influence of culturons is also affected by the degree of participation of the agent to the group in which it plays that role. This degree is used as a factor in calculating the real impact of the culturon, as shown in the formula below.

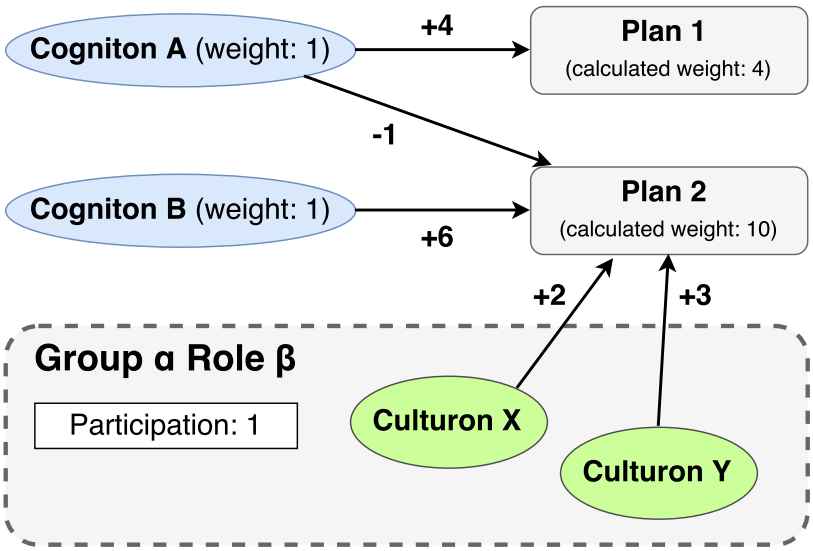


Figure 4: Culturons influences on the agent's plans

$$weight = \sum_i C_i L_i + \sum_j (A_j \sum_k L_k Cu_k) \tag{2}$$

The weight is computed based on the influences coming from all the groups j in which the agent plays at least one role. A_j is the degree of participation in the group, and L_k are the weights of the influence links of the culturons Cu_k associated to the role. The final weight is composed therefore by influences coming from cognitons $C_i L_i$ and culturons.

Each agent belonging to an instance of a group can modify the weights of the culturons for its instance of the group. The weight values of the culturons are shared by all the agents belonging to this instance of the group (a different instance of the same type of group will have its own culturon weight values).

2.3 Environment, buildings, objects and bodies

In MetaCiv, the cogniton architecture is dedicated to modelling the mind of the agent, while the impact of an agent's action on the environment and its objects is obtained through the body of the agent. In Figure 2 we show that MetaCiv, as generic framework, covers only the elements of the cogniton architecture. Modellers are left to use the platform of their choice or their own implementation to describe the environment, the buildings, the objects and the physical interactions between them and the body of the agent.

3 CogLogo

CogLogo [10] is a NetLogo [11] extension written in Java which provides an implementation of the cogniton architecture that integrates smoothly with NetLogo ability to describe an environment, physical agents, objects and interactions.

CogLogo integrates with NetLogo by providing procedures to modify the cognitons weights, organize groups of agents and modify culturons weights. The procedure *cogLogo:choose-next-plan* is called at the modeller's discretion, and returns a string which can be used to call any corresponding NetLogo or user defined procedure.

CogLogo has a graphical editor to design the internal cognitive architecture of the agents: the *Cognitive Scheme*. The *Cognitive Scheme* describes the individual thought process with cognitons and collective elements with culturons.

Multiple Cognitive Schemes can be defined and used in the same model, and can be assigned to different agent kinds (NetLogo's *breeds*). For each Cognitive Scheme we can choose among several decision makers.

CogLogo also provides tools to observe its elements during the simulation. The agent watcher window will show the weight of each cogniton and the calculated weights of the plans for a selected agent in real time.

3.1 Execution cycle

- 1 - The agent calls *cogLogo:choose-next-plan*.
- 2 - Active plans are evaluated: plans that do not have at least one conditional link to an active cogniton are deactivated.
- 3 - Plan weight is calculated: the value of each cogniton is multiplied by the influence link value and summed in the plan.
- 4 - The decision maker is called and the plan is selected according to its mechanism.
- 5 - *cogLogo:choose-next-plan* returns the name of the plan, which can be executed with the *run* command.
- 6 - The plan acts on the environment and on the agent's mind: cogniton values are altered, cognitons can be activated or deactivated, the agent can join or leave groups and change their degree of participation, feedback can be sent through reinforcement links.
- 7 - next simulation step repeats the process.

3.2 Links

The Cognitive Scheme editor is used to create the cognitons and culturons, and the influence links to the plans. The value of each influence link can be defined as a positive or negative real number.

CogLogo also offers two kinds of links not discussed before, the conditional links and the reinforcement links, which we will briefly explain.

The *conditional links* play a role in the simulation by telling the decision maker which plans are available. Cognitons can be activated and deactivated, and a plan can take part in the selection process only if it has at least one conditional link to an active cogniton (even if the cogniton weight is 0). This means that even if, through other cognitons, a plan obtains the highest calculated weight, it will never be selected without a conditional link to an active cogniton.

The *reinforcement links* provide a simple and more readable way to implement the reinforcement mechanism. While the evolution of certain cognitons are more a matter of perception and regulation (such as hunger, climate influence or age), others are involved in the long term behaviour of the agent and reflect a learning process (such as a social specialization or an attitude towards external factors). When an agent runs a plan, a feedback operation (*CogLogo:feed-back-from-plan*) can be called with a value parameter, this value is multiplied by the weight of the reinforcement link defined in the Cognitive Scheme and added to the corresponding cogniton.

The use of reinforcement links is entirely optional and the same effect can be accomplished by using the regular procedures to set the values of the cognitons. They are simply meant to simplify reinforcement mechanisms and make them visible in the CogLogo interface.

3.3 Decision Makers

3.3.0.1 MaximumWeight : the plan with the maximum weight is selected.

3.3.0.2 WeightedStochastic : the weight of each plan represents the probability for the plan to be selected. if PlanA = 5 and PlanB = 3 , the probability of being selected is : PlanA = 5/8 = 0.625 and PlanB = 3/8 = 0.375 a random function (0,1) is then called to select the plan.

3.3.0.3 BiasedWeightedStochastic : works the same way as the WeightedStochastic, but increase the probability of selecting the better plans accordingly to the bias factor. the plans are sorted from higher probability to lower, covering the range from 0 to 1 (the highest probablity covers the range from 0 to p , then the next plan from p to p+1 ...) the random function result is then elevated to the degree of the bias specified (which will bias the random function towards giving values closer to 0) (a bias of 1 will give the same results as the regular WeightedStochastic, but with a slightly higher computing cost)

3.4 Interface

3.4.1 General

3.4.1.1 Save model : saves all cognitive schemes

3.4.2 View

3.4.2.1 Cognitive Scheme editor : shows the Cognitive scheme editor view of your current cognitive scheme. You can edit cognitons in this view

3.4.2.2 Groups and roles editor : shows the Groups and roles editor view of your current cognitive scheme. You can edit groups, roles and culturons in this view

3.4.3 Select

3.4.3.1 Create new cognitive scheme : Create a new cognitive scheme

3.4.3.2 Cognitive scheme list : lets you select the current cognitive scheme

3.4.4 Options

3.4.4.1 Setting of the current cognitive scheme : the setting window for the current cognitive scheme. You can set the decision maker in this window.

3.4.4.2 Save current cognitive scheme : Save changes to the current cognitive scheme only.

3.4.4.3 Delete current cognitive scheme : Deletes the current cognitive scheme.

3.4.5 Observe

3.4.5.1 Observe a single agent : Opens the agent watcher window.

3.4.6 Help

3.4.6.1 Help : shows the readme.txt

3.4.6.2 Console : shows the CogLogo debugging console. May help you understand what went wrong with your model.

3.4.6.3 About : License and such.

3.5 NetLogo Commands

3.5.1 General

3.5.1.1 OpenEditor : opens/close the editor panel

3.5.1.2 choose-next-plan : returns a string, the name of the next plan to run. (ex : run cognitonsfornetlogo:choose-next-plan)

3.5.1.3 feed-back-from-plan <string planName><double val> : propagates val to all the cognitons (or culturons) linked to the plan by a reinforcement link. the value parameter (val) is multiplied by the value of the reinforcement link, the result is added to the value of the cogniton (or culturon)

3.5.1.4 report-agent-data : reports the value of all cognitons and the resulting weights of the plans to our observation panel

3.5.1.5 reset-simulation : to be called in the setup, resets group count and agent data tracking

3.5.2 Cognitons

3.5.2.1 init-cognitons : must be called for each agent using a cognitive scheme , only during the setup.

3.5.2.2 add-to-cogniton-value <string cognitonName><double val> : adds val to the corresponding cogniton

3.5.2.3 set-cogniton-value <string cognitonName><double val> : sets the value to val of the corresponding cogniton

3.5.2.4 get-cogniton-value <string cognitonName> : returns the value of the corresponding cogniton. if the cogniton is not active the function returns 0.

3.5.2.5 activate-cogniton <string cognitonName> : activates a cogniton and sets its value to 0. All plans connected to this cogniton via dependancy links will be avalaible for the next call of choose-next-plan (if they were not already).

3.5.2.6 deactivate-cogniton <string cognitonName> : deactivates a cogniton. All plans connected to this cogniton via dependancy links will be unavalaible for the next call of choose-next-plan if they are not connected to another active cogniton via dependancy links.

3.5.3 Culturons

3.5.3.1 create-and-join-group <string groupName><string roleName> : create an instance of <groupName>type and join it in the role <roleName>. When joining, the participation (involvement) value is set to 1.0.

3.5.3.2 join-group <string groupName><string roleName><double groupId> : join the instance <groupId>of type <groupName>in the role <roleName>. When joining, the participation (involvement) value is set to 1.0.

3.5.3.3 add-to-participation <string groupName><double val> : adds val to the corresponding group involvement.

3.5.3.4 set-participation <string groupName><double val> : sets the value to val of the corresponding group involvement.

3.5.3.5 get-group-id <string groupName> : returns the group identifier of groupName (a number >= 0) of the agent. if the agent is not in any role of this group type the function returns -1.

3.5.3.6 get-group-role-id <string groupName><string roleName> : returns the group identifier of groupName (a number >= 0) of the agent if the agent has the role of roleName. if the agent is not in this role of this group type the function returns -1.

3.5.3.7 leave-group <string groupName> : leave the group of groupName (if the agent is not in this type of group, does nothing)

3.5.3.8 leave-all-groups : the agent leaves all his groups

3.5.3.9 add-to-culturon-value <String groupName><String culturonName><Double value> : add <value>to the culturon <culturonName>of the group type <groupName>

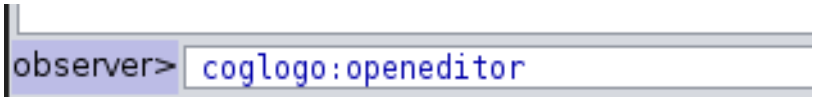
3.5.3.10 set-culturon-value <String groupName><String culturonName><Double value> : set <value>as the value of the culturon <culturonName>of the group type <groupName>

3.5.3.11 get-culturon-value <String groupName><String culturonName> : return the value of the culturon <culturonName>of the group type <groupName>. if the agent is not in any role of this group type the function returns 0.

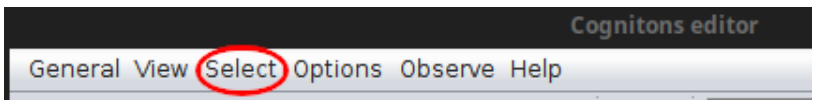
4 A short tutorial

4.1 Creating the cognitive scheme

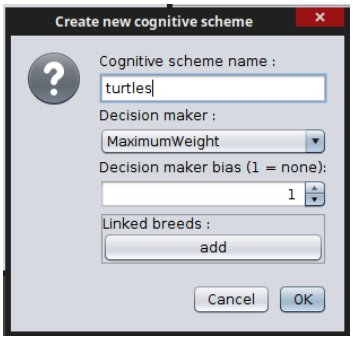
Before you can open the editor, import the coglogo extension (*extensions [CogLogo]* in the code window) and save your model. Open the editor (by command or button).



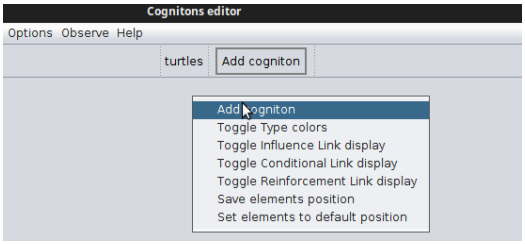
In the menu bar : Select>create new cognitive scheme.



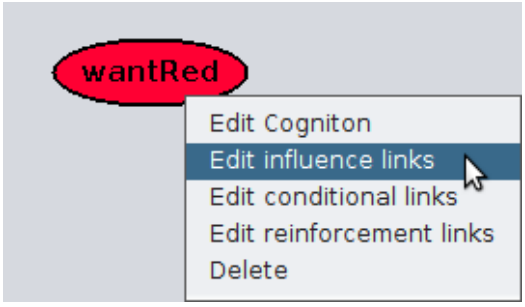
Name the cognitive scheme "turtles" (the breed we use in this model), leave default for other settings.

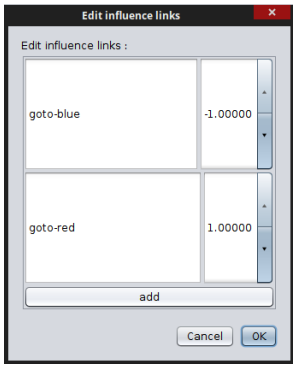


Right click on the background >add cogniton (or use the button). Name it "wantRed" and tick the "is added at birth?" box.

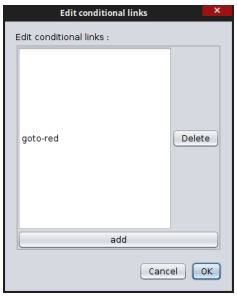


Right click on the "wantRed" cogniton >edit influence link. Add a link to the plan "goto-red" (type in the text box) and set its value to 1. Add a link to the plan "goto-blue" and set its value to -1 and close the window (ok).





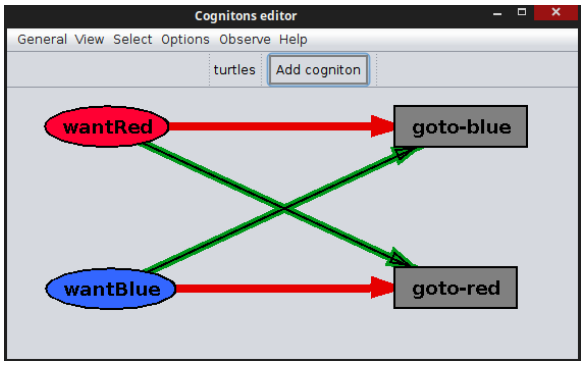
Right click on the "wantRed" cogniton >edit conditional link. Add a link to the plan "goto-red" and close the window (ok).



Now we repeat the process for the "wantBlue" cogniton :

Right click on the background >add cogniton (or use the button). Name it "wantBlue" and tick the "is added at birth?" box set the default value to 1. Right click on the "wantBlue" cogniton >edit influence link. Add a link to the plan "goto-blue" (type in the text box) and set its value to 1. Add a link to the plan "goto-red" and set its value to -1 and close the window (ok). Right click on the "wantBlue" cogniton >edit conditional link. Add a link to the plan "goto-blue" and close the window (ok).

The *turtles* cognitive scheme should look like this:



4.2 Using the cognitive scheme in NetLogo

The following shows the complete **setup** procedure. Each agent using a cognitive scheme must call the *coglogo:init-cognitons* procedure (line 8).

```
1  to setup
2    clear-all
3    reset-ticks
4    set redpatch patch 9 9
5    set bluepatch patch -9 9
6    set greenpatch patch 0 -9
7    create-turtles 1 [
8      coglogo:init-cognitons
9      set color white
10     set size 3 ; easier to see
11     setxy random-xcor random-ycor
12   ]
13   ask redpatch [
14     set pcolor red
15     ask neighbors [set pcolor red]
```

```
  ]
  ask bluepatch [
    set pcolor blue
    ask neighbors [set pcolor blue]
  ]
  ask greenpatch [
    set pcolor green
    ask neighbors [set pcolor green]
  ]
end
```

In the **go** procedure, each agent calls *coglogo:choose-next-plan* which returns the name of the selected plan as a character string. The run primitive is able to run a NetLogo procedure from a character string corresponding to its name.

coglogo:report-agent-data makes the data available to the CogLogo Agent Watcher. Here reporting is done each tick, you can choose to report every 100 ticks or only report for a specific agent.

```
to go
  ask turtles [
    act-on-cognitons              ; cognitons evolve
    run coglogo:choose-next-plan  ; plan chosen by cognitons
    coglogo:report-agent-data     ; send data to agent watcher interface
  ]
  tick
end
```

In **act-on-cognitons** we set the values of the cognitons.

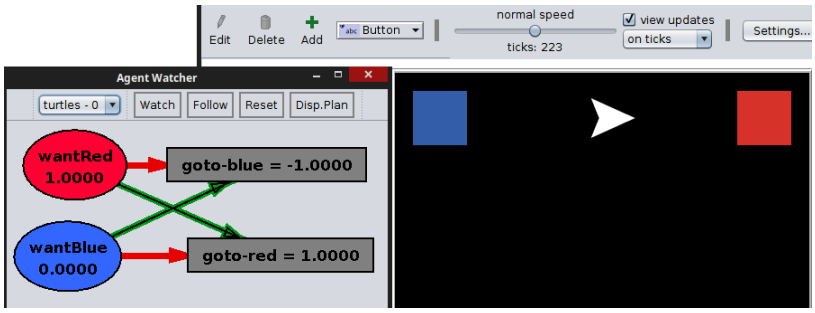
```
to act-on-cognitons
  if pcolor = red[
    coglogo:set-cogniton-value "wantRed" 0
    coglogo:set-cogniton-value "wantBlue" 1
  ]
  if pcolor = blue[
    coglogo:set-cogniton-value "wantRed" 1
    coglogo:set-cogniton-value "wantBlue" 0
  ]
end
```

Each plan in the cognitive scheme MUST have a corresponding NetLogo procedure. The name of the procedure must correspond exactly.

```
to goto-blue
  face bluepatch
  fd 1
end
```

Don't forget to define goto-red

Try your model, the turtle should go back and forth between the blue and red patch.



4.3 Adding reinforcement links to the cognitive scheme

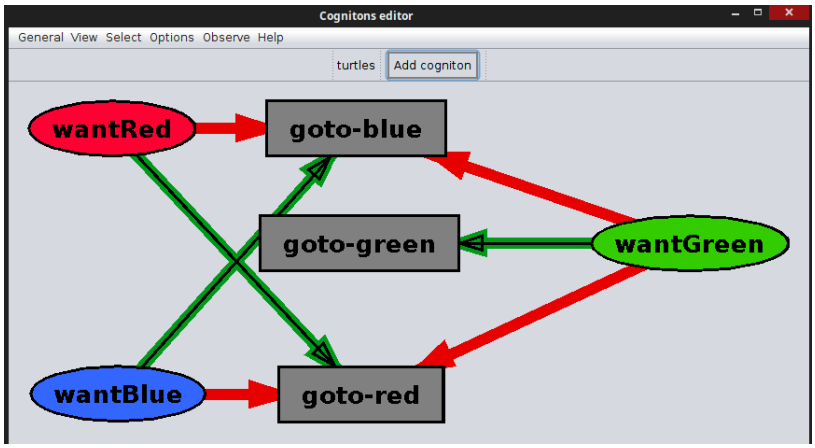
We will repeat the previous process for a "wantGreen" cogniton.

Right click on the background >add cogniton (or use the button). Name it "wantGreen" and tick the "is added at birth?" box

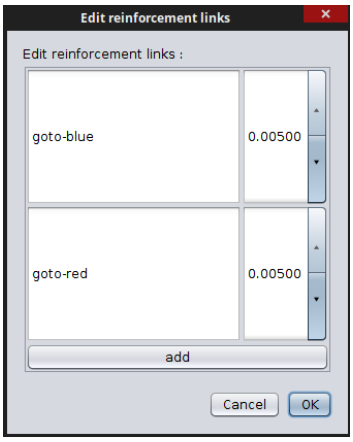
Right click on the "wantGreen" cogniton >edit influence link. Add a link to the plan "goto-green" (type in the text box) and set its value to 1 Add a link to the plan "goto-red" and set its value to -1 Add a link to the plan "goto-blue" and set its value to -1 and close the window (ok)

Right click on the "wantGreen" cogniton >edit conditional link. Add a link to the plan "goto-green" and close the window (ok).

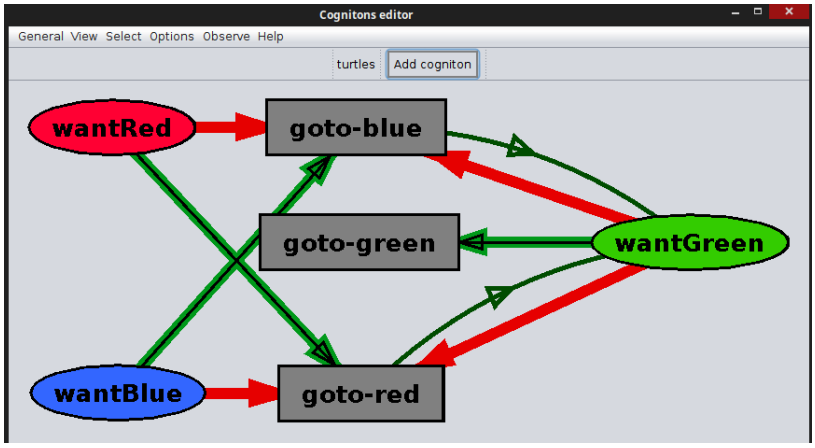
At this point your model should look like this:



Right click on the "wantGreen" cogniton >edit reinforcement link. Add a link to the plan "goto-red" and set its value to 0.005. Add a link to the plan "goto-blue" and set its value to 0.005 and close the window (ok)



The complete model should look like this:



4.4 Adding reinforcement links to the cognitive scheme

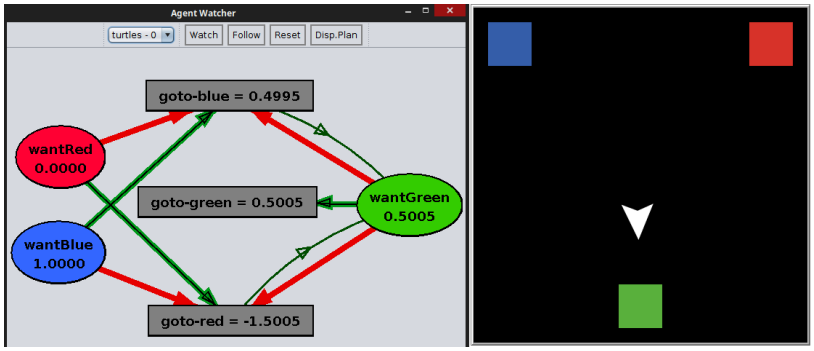
Every use of the goto-blue and goto-red plans will reinforce the "wantGreen" cogniton. Simply add the *coglogo:feed-back-from-plan "goto-*" *value** to the corresponding plan.

```
to goto-blue
  face bluepatch
  fd 1
  coglogo:feed-back-from-plan "goto-blue" 1.1
end

to goto-red
  face redpatch
  fd 1
  coglogo:feed-back-from-plan "goto-red" 1.1
end

to goto-green
  face greenpatch
  fd 1
end
```

To complete the model, we will reset the "wantGreen" cogniton to 0 when the agent reaches the green patch.

```
to act-on-cognitons
  if pcolor = red[
    coglogo:set-cogniton-value "wantRed" 0
    coglogo:set-cogniton-value "wantBlue" 1
  ]
  if pcolor = blue[
    coglogo:set-cogniton-value "wantRed" 1
    coglogo:set-cogniton-value "wantBlue" 0
  ]
  if pcolor = green[
    coglogo:set-cogniton-value "wantGreen" 0
  ]
end
```

Try you model, the turtle should go back and forth between the blue and red patch. After enough reinforcement is accumulated, the turtle should go to the green patch. Reaching the green patch will reset the value of wantGreen cogniton, the turtle will resume the red-blue cycle



References

[1] Jacques Ferber. *Les Systèmes Multi-agents, Vers une intelligence collective.* InterEditions, Paris, 1995.

[2] Jacques Ferber, O Gutknecht, Fabien Michel, et al. Agent/group/roles: Simulating with organizations. In *ABS'03: Agent Based Simulation*, 2003.

[3] Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations: an organizational view of multi-agent systems. *Agent-Oriented Software Engineering (AOSE) IV*, 2004.

[4] Jacques Ferber, Fabien Michel, and José-Antonio Báez-Barranco. AGRE : Integrating Environments with Organizations. In *Environments for Multi-agent Systems*, volume 3374 of *Lecture Notes in Computer Science*, pages 48–56. Springer, 2005.

[5] Jacques Ferber, Julien Nigon, Gautier Maille, Tristan Seguin, Sven Holtz, and Tiberiu Stratulat. De masq à metaciv: un cadre générique pour modéliser des sociétés humaines dans une approche transdisciplinaire, 2014.

[6] Olivier Gutknecht. *Proposition d'un modèle organisationnel générique de systèmes multi-agents et examen de ses conséquences formelles, implémentatoires et méthologiques.* PhD thesis, Université Montpellier II-Sciences et Techniques du Languedoc, 2001.

[7] Denis Phan, editor. *La structuration sociale de l'espace scolaire : une ontologie multi-points de vue, intégrée mais non réductrice.* Hermes-Lavoisier, Londres-Paris, 2014.

[8] Tiberiu Stratulat. *Systèmes d'agents normatifs: concepts et outils logiques.* PhD thesis, Université de Caen, 2002.

[9] Tiberiu Stratulat, Jacques Ferber, and John Tranier. MASQ: towards an integral approach to interaction. *Proc of 8th Int. Conf. on Autonomous Agents and Multiagent Systems*, pages 813–820, 2009.

[10] François Suro. CogLogo. `https://github.com/suroFr/CogLogo`, 2017. Equipe SMILE, Laboratoire d'informatique, de robotique et de microélectronique de Montpellier, France.

[11] U. Wilensky. Netlogo. `http://ccl.northwestern.edu/netlogo`, 1999. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.