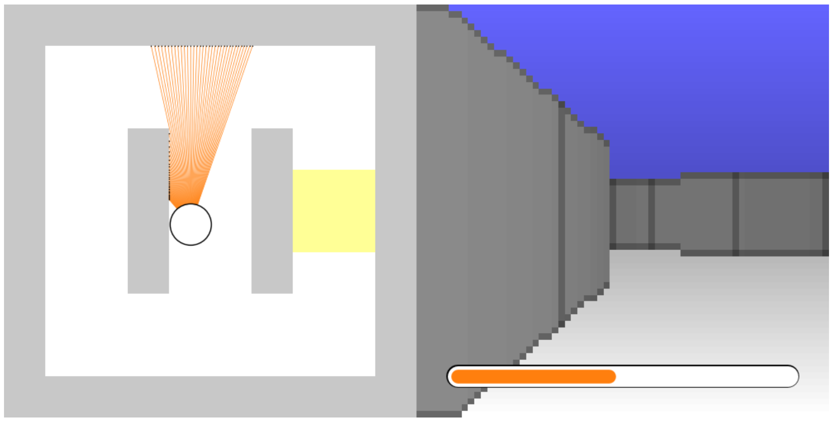
1 Programmatoid braincraft challenge solution

Let us consider the following digital experimental setup, as described in braincraft challenge presentation[1].

1.1 Simplified problem statement



The braincraft challenge[2] bot moves at a constant with sensor inputs and one orientation output, it uses some energy and refill this energy on a given yellow location. The 2D space size is $[0, 1] \times [0, 1]$. The bot starts in the middle and oriented at 90, i.e., upward.

Input variables	
$p_l \in [0_{\text{wall hit}}, 1_{\text{no wall}}[$	Leftward proximity, min value of the [0, +30°] range 32 left sensors.
$p_r \in [0_{\text{wall hit}}, 1_{\text{no wall}}[$	Rightward proximity, min value of the [-30°, 0] range 32 right sensors.
$c_l \in \{0, 4, 5\}$	Leftward color sensor: 4 blue block. 5 red block. 0 no special block.
$c_r \in \{0, 4, 5\}$	Rightward color sensor, idem.
$g_e \in [0_{\text{death}}, 1_{\text{full}}]$	Energy gauge value.
Output variable	
$d_o \in [-5, 5]$	Orientation difference, saturated at ± 5°.
Internal variables, with initial values	
$q_p \in \{0_{\text{leftward}}, 1_{\text{rightward}}\}, \quad q_p\|_{t=0} = 0$	Preferred quarter-turn direction.
$c_p \in \{0, 4, 5\}, \quad c_p\|_{t=0} = 0$	Preferred quarter-turn direction color.
$g_1 \in [0, 1], \quad g_1\|_{t=0} = 1$	Previous energy gauge value.
$g_d \in [0, 1], \quad g_1\|_{t=0} = 1$	Previous energy gauge increase.

With respect to the original we consider only two leftward and rightward sensors, while the wall hit indicator is not used.

[1]`https://github.com/rougier/braincraft/blob/master/README.md#introduction`
[2]`https://github.com/rougier/braincraft`

1.2 A putative controller

1.2.1 Navigation

Heuristic : The bot runs at constant velocity,

(i) ahead by default, thanks to a linear correction, high enough to correct the direction, small enough to avoid oscillations, and

(ii) attempts to perform a quarter-turn in the preferred orientation as soon as passing is detected.

With this navigation mechanism the bot performs either leftward or rightward half-loops, traversing the central corridor.

Programmatoid solution :

$$\begin{array}{rcccc} d_o & = & \underbrace{\gamma\,(p_l - p_r)}_{\substack{\text{linear correction to} \\ \text{maintain direction ahead}}} & + & \underbrace{\alpha\,(t_l - t_r)}_{\substack{\text{quarter- turn} \\ \text{left or right}}} \\ t_l & = & (1 - q_p)\,H(\beta - p_l) & = & H((\beta - p_l) - \upsilon\,q_p) \\ t_r & = & q_p\,H(\beta - p_r) & = & H((\beta - p_r) - \upsilon\,(1 - q_p)) \end{array}$$

where:

w	$\simeq$	$1/4$	Rough estimation of the path-width.
γ	$=$	$\frac{5}{w/2}$	Saturates the correction at 5 if the depth difference is half of the path-width.
α	$=$	5	Saturates the correction at ± 5 to make the quarter-turn.
β	$=$	w	Triggers the quarter-turn if the depth is higher than the path-width.
υ	$=$	2	Transform a boolean product to a threshold.

Regarding transforming a boolean product to a threshold, we easily verify that:

$$b \in \{0, 1\}, x \in]-M, M[\rightarrow b\,H(x) = H(x - (1 - b)\,M),$$

for both values of b, while $\max(|\beta - p_l|, |\beta - p_r) \leq 1$; this is developed in the sequel.

We thus have a linear output unit for d_o and two step-unit for t_l and t_r.

1.2.2 Direction choice

Task's strategy :

Task 1 Restrict navigation to the half-loop that contains the energy source, the other does not.

Heuristic If the energy is too low, thus turning in the wrong direction, the direction is changed once.

Task 2 Restrict navigation to the half-loop without a closed path, indicated by a color that has already been seen once.

Heuristic

```
Detect the cue if not yet done
if   $c_p = 0$ and $c_l \neq 0$                        then   $c_p \leftarrow c_l$
if   $c_p = 0$ and $c_r \neq 0$                        then   $c_p \leftarrow c_r$
Set the direction according to the cue
if   $c_p \neq 0$ and $c_l \neq 0$ and $c_p = c_l$     then   $q_c \leftarrow 0$
if   $c_p \neq 0$ and $c_r \neq 0$ and $c_p = c_r$     then   $q_c \leftarrow 1$
Reset the cue at a certain energy decrease
if   $g_e < \kappa$                                    then   $c_p \leftarrow 0$
```

Task 3 Test both energy sources on the left and on the right and decide which one is better.

Heuristic When energy increase occurs if the previous energy increase is higher, change direction.

Programmatoid solution :

$$\begin{array}{rcl}
\lambda & = & \lambda + \Delta\lambda_1 + \Delta\lambda_2 + \Delta\lambda_3 + \cdots \\
\Delta\lambda_1 & = & (\lambda - 1)\, H(\alpha - \varepsilon) \\
\Delta\lambda_2 & = & (\lambda - 1)\, H(\iota - I_{\text{left blue color sensor}}) \\
\Delta\lambda_3 & = & (\lambda - 1)\, (\Upsilon_{\text{again red color}} + \Upsilon_{\text{again yellow color}} + \cdots) \\
\cdots & & \\
\Upsilon_{\text{again this color}} & = & \Upsilon_{\text{seen this color}}\, H(\iota - I_{\text{left this color sensor}}) \\
\Upsilon_{\text{seen this color}} & = & \Upsilon_{\text{seen this color}} + (1 - \Upsilon_{\text{seen this color}})\, H(\iota - I_{\text{this color sensor}}) \\
\cdots & &
\end{array}$$

where:

α is a the energy threshold, corresponding to the energy consumption during one turn.

ι is some color detection threshold.

$\Delta\lambda_1$ raises to one if the energy decreases below a threshold.

$\Delta\lambda_2$ raises to one if the blue color is seen on the left.

$\Delta\lambda_3$ raises to one if some color is seen again.

$\Upsilon_{\text{again this color}}$ raises to one a previously seen color is seen again on the left.

$\Upsilon_{\text{seen this color}}$ raises to one a color is seen for the first time.

$I_{\text{left this color sensor}}$ combines color sensor input.

$I_{\text{this color sensor}}$ combines left and right color sensor input.

Neuronoid implementation : The mollification of this system uses 3 neuronoids and writes:

$$\begin{array}{rcl} d\theta & = & h\left(\gamma\,(\theta_l - \theta_r) + \frac{\pi}{2}\,(\Delta\theta_r - \Delta\theta_l)\right) \\ \Delta\theta_r & = & h(W_\infty\,(\beta - \theta_r) + 2\,W_\infty\,(\lambda - 1)) \\ \Delta\theta_l & = & h(W_\infty\,(\beta - \theta_l) - 2\,W_\infty\,\lambda) \end{array}$$

as easily verified considering the four cases $\beta \lesseqgtr \theta_*$ versus $\lambda \in \{0, 1\}$.

With respect to the programmatoid solution, the output value is "saturated" by the $h(\cdot)$ function while the $\Delta\theta_*$ almost binary values are approximated by the mollification of the step function. This part of the system is feed-forward thus without convergence or stability issue.

1.2.3 Computation unit: neuronoid

By "neuronoid" we name the not very biologically plausible[3] simplest biological neuron or neuron small ensemble inspired by mean-field modelisation[4] of the Hodgkin–Huxley neuronal axon model[5].

$$\begin{array}{cc} \tau \frac{\partial v_i}{\partial t}(t) + v_i(t) = z_i(t), & z_i(t) \overset{\text{def}}{=} h\left(\sum_j w_{ij}\, v_j(t) + w_i\right), \\ h(v) \overset{\text{def}}{=} \frac{1}{1+\exp(-4\,v)}, & H(v) \overset{\text{def}}{=} \left\{ \begin{array}{lll} 1 & \text{if} & v > 0 \\ 0 & \text{if} & v < 0 \end{array} \right. \end{array}$$

where v_i is the membrane potential, so that:

$$\begin{array}{rcl} v(t) & = & 1/\tau \int_0^t z(s)\, exp(-(t-s)/\tau)\, ds + v(0)\, exp(-t/\tau) \\ & = & z(0) + (v(0) - z(0))\, e^{-t/\tau}\big|_{z(t)=z(0)}\,. \\ & = & z(t)|_{\tau=0} \\ & \simeq & (1-\gamma)\, v(t-\Delta t) + \gamma\, z(t-\Delta t)) \\ & = & \sum_{s=0}^{t-1} z(s)\, \gamma\, (1-\gamma)^{t-s+1}\, z(s) + v(0)\, (1-\gamma)^t \\ & = & z(0) + (v(0) - z(0))\, (1-\gamma)^t|_{z(t)=z(0)} \\ & = & z(t)|_{\gamma=1}\,. \end{array}$$

writing also the corresponding discrete approximation using an Euler schema with $0 < \gamma < 1, 0 < \tau$:

$$\begin{array}{c} \gamma \overset{\text{def}}{=} 1 - \exp(-1/\tau) \Leftrightarrow \tau = 1/\log(1/(1-\gamma)) \\ \lim_{\gamma \to 0} \tau = +\infty, \lim_{\gamma \to 1} \tau = 0. \end{array}$$

Here, $h(\cdot)$ is the normalized sigmoid with $h(-\infty) = 0, h(0) = 1/2, h'(0) = 1, h(+\infty) = 1$, linearly related to the hyperbolic function:

$$h(v) = \frac{1+\tanh(2\,v)}{2}.$$

It is the mollification of the step function, called also Heaviside function, $H(\cdot)$, as detailed below.

It is thus a very common 1st order "neuronoid" model, but with an adjustable bias (or offset) w_i.

1.3 Programmatoid and neuronoid computation

1.3.1 Programmatoid computation

We name "programmatoid" computation the conception of an input-output straight-line program[6] implementing test operator on numerical value expressions using the step function, considering 1 as the true value and 0 as the false value. The $H(v)$ implements the test of the positive sign of v, while for $v_i \in \{0, 1\}$:

[3]https://en.wikipedia.org/wiki/Biological_neuron_model#Relation_between_artificial_and_biological_neuron_models

[4]https://inria.hal.science/cel-01095603v1

[5]https://en.wikipedia.org/wiki/Hodgkin-Huxley_model

[6]https://en.wikipedia.org/wiki/Straight-line_program

programmatoid conjunction The $v_0 = H(v_1 + v_2 + \cdots)$ formula performs a *or* operation.

programmatoid disjunction The $v_0 = v_1\, v_2 \cdots$ formula performs a *and* operation.

programmatoid negation The $v_0 = (1 - v_1)$ formula performs a negation.

so that we can combine any boolean expression on any test of value sign, thus value comparison, and value interval inclusion, switches between two expressions, etc. This defines real semi-algebraic[7] sets of degree 1.

Using local feedback we can also design several functions detailed in the Appendix of this section, while we also can combine neuronoid and programmatoid functions to design temporal functions.

1.3.2 Mollification of the step function

The step function is related to a conditional expression by a simple relation:

$$H(v) = \mathit{conditional\ value} > 0\ ?\ 1 : \mathit{conditional\ value} < 0\ ?\ 0 : H(0),$$

where $condition\ ?\ value\ if\ true : value\ if\ false$ is a conditional expression.

The step function approximates sigmoid with huge slope at zero, i.e.:

$$\forall v \neq 0, H(v) = \lim_{W_\infty \rightarrow +\infty} h(W_\infty\, v),\, h'(W_\infty\, v)|_{v=0} = W_\infty$$

while the convergence is also obtained for $v = 0$ in the distribution sense with $H(0) = h(0) = 1/2$. More precisely:

$$|H(\cdot) - h(\cdot)|_{\mathcal{L}_1} = \frac{\log(2)}{2} \frac{1}{W_\infty}$$

and, for $0 < \epsilon_\infty \ll 1 < v_\infty$:

$$h(W_\infty\, v_\infty) = 1 - \epsilon_\infty \Leftrightarrow v_\infty = \frac{\log(1-\epsilon_\infty) - \log(\epsilon_\infty)}{4\, W_\infty} = \frac{-\log(\epsilon_\infty)}{4\, W_\infty} + O\left(\epsilon_\infty\right).$$

Numerically, the convergence is very fast, e.g. $W_\infty = 2$, for $\epsilon_\infty = 0.1\%$:

ϵ_∞	10^{-1}	10^{-2}	10^{-3}	10^{-6}
W_∞	0.55	1.15	1.73	3.46

A step further, the local variation of sigmoid writes:

$$\begin{array}{rcl} h(v) & = & h(v_0) + [4\, \exp(-8\, v_0) + O\left(\exp(-12\, v_0)\right)]\, (v - v_0) + O\left((v - v_0)^2\right) \\ & \simeq & h(v_0) + 4\, \exp(-8\, v_0)\, (v - v_0). \end{array}$$

Numerically, values decrease very rapidly:

v_0	1	2	5	10
$4\, \exp(-8\, v_0) \simeq$	10^{-1}	10^{-3}	10^{-8}	10^{-17}

1.3.3 Neuronoid implementation of programmatoid computation

We call "neuronoid" computation the conception of an input-output transform based on feed-forward and recurrent combination of neuronoids, as defined previously.

[7] `https://en.wikipedia.org/wiki/Real_algebraic_geometry`

By successive combination, any programmatoid computation involving the $H(\cdot)$ function can be approximated by a neuronoid, with $\tau \simeq 0$.

A step ahead, we considering neuronoid with $\tau > 0$ it seems obvious that we can designed temporizing mechanisms, oscillators and sequence generator, sleep sort mechanism, etc. (not detailed here because not used at this stage, see appendix).

Neuronoid implementation : The mollification uses 3 neuronoids for cases 1 and 2 plus 3 neuronoids by color for case 3. The derivation of the neuronoid equations is straightforward after the previous one.

1.4 Appendix: a few programmatoid and neuronoid components

1.4.1 Conditional expression

For two inputs $v_{i1}(t) \in \{0,1\}, v_{i0}(t) \in \{0,1\}$, an output $v_o(t) \in \{0,1\}$ and a control $v_l \in \{0,1\}$, the condition expression equation, for $0 < W_\delta \ll 1 < W_\sigma$:

$$\begin{aligned}
v_o(t+1) &= v_l(t) == 1\ ?\ v_{i1}(t) : v_{i0}(t) && (1)\\
&= v_l(t)\, v_{i1}(t) + (1 - v_l(t))\, v_{i0}(t) && (2)\\
&= H(v_{i1}(t) - W_\sigma\,(1 - v_l(t)) - W_\delta) + H(v_{i0}(t) - W_\sigma\, v_l(t) - W_\delta) && (3)\\
&\simeq h(W'_\infty\,(W_\omega\, v_{i1}(t) - W_\sigma\,(1 - v_l(t)) - W_\delta))\\
&+ h(W'_\infty\,(W_\omega\, v_{i0}(t) - W_\sigma\, v_l(t) - W_\delta)). && (4)
\end{aligned}$$

To explain these design choices, let us notice that:

- The W_σ value is used at the programmatoid level to ensure that given the $v_l(t)$ binary switch value, it constraints the step function output to correspond to the desired value. Here, an expression including a product by a binary function, is replaces by a sum. The rationale is that there is no explicit multiplication between two variables at the neuronoid level. This trick allows one a straightforward neuronoid approximation.

- The W_δ value is used at the programmatoid level to avoid the ambiguous 0 value and ensure that for $v \simeq 0$ we obtain $H(v - W_\delta) = 0$.

- The W'_∞ gain is used to approximate the step function by a sigmoid, as discussed previously.

- Informally, at the neuronoid level, we mimic the programmatoid mechanisms, assuming that suitable $W'_\infty, W_\omega, W_\sigma, W_\delta$ values will reproduced the programmatoid conditional expression. It works, although the parameter adjustment is not obvious, as derived now.

Let us now verify the equivalence between these equations:

- It is obvious to verify that line (1) and (2) are equivalent.

- The fact line (2) and (3) are equivalent is verified by this truth table:

$v_l(t)$	$v_{i1}(t)$	$v_{i0}(t)$	$v_{i1}(t) - W_\sigma\,(1 - v_l(t)) - W_\delta$	$v_{i0}(t) - W_\sigma\,v_l(t) - W_\delta$	$v_o(t+1)$
1	0	0	$-W_\delta < 0$	$-W_\sigma - W_\delta < 0$	$0 + 0 = 0 = v_{i1}(t)$
1	0	1	$-W_\delta < 0$	$1 - W_\sigma - W_\delta < 0$	$0 + 0 = 0 = v_{i1}(t)$
1	1	0	$1 - W_\delta > 0$	$-W_\sigma - W_\delta < 0$	$1 + 0 = 1 = v_{i1}(t)$
1	1	1	$1 - W_\delta > 0$	$1 - W_\sigma - W_\delta < 0$	$1 + 0 = 1 = v_{i1}(t)$
0	0	0	$-W_\sigma - W_\delta < 0$	$-W_\delta < 0$	$0 + 0 = 0 = v_{i0}(t)$
0	0	1	$-W_\sigma - W_\delta < 0$	$1 - W_\delta > 0$	$0 + 1 = 1 = v_{i0}(t)$
0	1	0	$1 - W_\sigma - W_\delta < 0$	$-W_\delta > 0$	$0 + 0 = 0 = v_{i0}(t)$
0	1	1	$1 - W_\sigma - W_\delta < 0$	$1 - W_\delta > 0$	$0 + 1 = 1 = v_{i0}(t)$

- The approximation of line (3) at line (4) by continuous quantities corresponds now to:

  $$v_*(t) \in \{[0, \epsilon_\infty], [1 - \epsilon_\infty, 1]\} = \{\simeq 0, \simeq 1\}, \epsilon_\infty \ll 1/2,$$

  in words: values to be either below or above a threshold close to either 0 or 1. The truth table now involves intervals and has been generated using the computer algebra piece of code associated to this document[8]. Considering the following intuitive design constraints:

  $$0 < W_\delta < \{W_\omega, W_\sigma\}, 0 < W_\omega < W_\sigma, 0 < \epsilon_\infty < 1$$

  the computer algebra derivations show that the approximation at line (4) is valid as soon as:

  $$W_\omega\,\epsilon_\infty < W_\delta, W_\sigma\,\epsilon_\infty < W_\delta, W_\delta + (W_\omega + W_\sigma)\,\epsilon_\infty < W_\omega.$$

  Furthermore, let us consider the margin:

  $$0 < \mu = \min(W_\delta - \epsilon_\infty\,W_\sigma, W_\omega - (W_\omega + W_\sigma)\,\epsilon_\infty - W_\delta),$$

  yielding:

  $$|v_o(t+1) - 1/2| > h(W'_\infty\,\mu).$$

  We thus can adjust $W'_\infty = W_\infty/\mu$ in order $v_o(t+1) \in \{[0, \epsilon_\infty], [1 - \epsilon_\infty, 1]\}$ for further calculation.

  For instance $\{W_\delta = 1, W_\omega = 2, W_\sigma = 4, \epsilon_\infty = 1/8\}$ is a suitable solution with $\mu = 1/4$.

1.4.2 RS input/output gate

For an input $v_i(t) \in \{0, 1\}$, an output $v_o(t) \in \{0, 1\}$, and a control $v_l \in \{0, 1\}$, the equation:

$$\begin{array}{rcl} v_o(t+1) & = & v_l(t) == 1\ ?\ v_o(t)\ :\ v_i(t), \\ & = & v_l(t)\,v_o(t) + (1 - v_l(t))\,v_i(t) \\ & = & H(v_o(t) - W_\sigma\,(1 - v_l(t)) - W_\delta) + H(v_i(t) - W_\sigma\,v_l(t) - W_\delta) \\ & \simeq & h(W'_\infty\,(W_\omega\,v_o(t) - W_\sigma\,(1 - v_l(t)) - W_\delta)) \\ & + & h(W'_\infty\,(W_\omega\,v_i(t) - W_\sigma\,v_l(t) - W_\delta)). \end{array}$$

implements a 1 bit memory, i.e., also called a RS gate, and reusing the previous conditional instruction parameters.

[8]`https://github.com/vthierry/braincraft/raw/master/data/programmatic-solution.mpl.txt.out`

The key point is that it is now a recurrent system, which stability is obvious at the programmatoid level, but not necessarily at the neuronoid level, since we have a continous equation. More precisely:

$$v_o(t+1) = h(W'_\infty\, W_\omega\, v_o(t) - W_\beta(t)) + h_\beta(t),$$
$$\left\{ \begin{array}{rcl} W_\beta & \overset{\text{def}}{=} & W_\sigma\,(1 - v_l(t)) + W_\delta \\ h_\beta(t) & \overset{\text{def}}{=} & h(W'_\infty\,(W_\omega\, v_i(t) - W_\sigma\, v_l(t) - W_\delta)) \in [-1, 1] \end{array} \right.$$

with a non contracting recurrent function, since:

$$h'(W'_\infty\, W_\omega\, v) > 1 \text{ for } |v| \leq W'_\infty\, W_\omega.$$

monostable binary value The $v_0 = v_0 + (1 - v_0)\, H(v_1)$ formula sets v_0 to 1 for ever, as soon v_1 has raised once to 1.

and by combination RS gates, bistable mechanisms, etc. (not detailed here because not used at this stage, see appendix).

To be done.

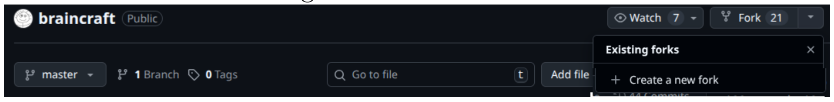
2 Using the braincraft challenge setup

Reference: The braincraft challenge[9].

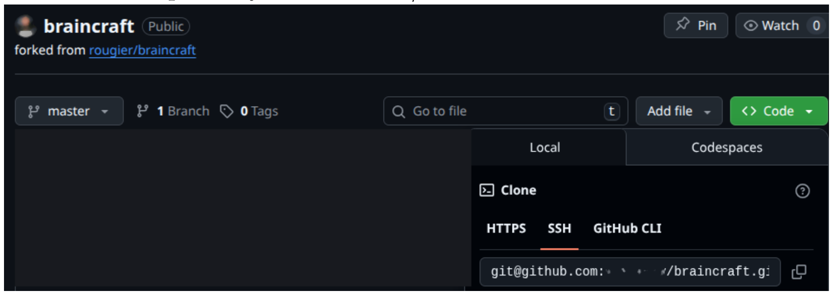
You must be familiar with basic `git` usage and basic `python` programmation.

2.1 Installation of the setup

- Connect to `https://github.com` with your login.
- Go to the braincraft challenge[10] and create a new fork:



- Download the repository in SSH read/write mode:



-In the braincraft local git directory, run `make test`

[9]`https://github.com/rougier/braincraft`
[10]`https://github.com/rougier/braincraft`

+ You may have to run `make install`, before.
+ You are advised to use a virtual environment, running `make venv`

2.2 Running at the programmatic level

API doc[11]
The ‘challenge_callback_1.py‘ file contains the support routines

2.3 Running at the artificial neural network level

```
* Note : vthierry veut PAS gagner la compétition is veut just vérifier des hypothèse quant l

- Warningup : duration (bot don’t move before warmup period is over) just 1 to allow a 1st i

- Quelles dimensions pour Win (quelles entrées où ? le truc de dimension P), W, et Wout(lign

- Avis sur calcul de depth et couleur ?
 + depth: prendre le min des capteurs de gauche/droite ou vaut mieux leur moyenne pondérée ?
 + couleur: les murs ont le vert comme couleur par defaut ? prendre la valeur de couleur la

- Pour expérimenter l’approche programmatique avant de passer à l’approche connexiviste :
 - comment ‘‘débrancher´´ le réseau et just avoir (P inputs) -> output ? sans reimplementer

- Pour expérimenter l’approche connectiviste sans apprentissage ... juste implémenter def tr
```

[11] https://raw.githack.com/vthierry/braincraft/master/braincraft/doc/challenge_callback.html