Quick Reference for XGrabtor 0.33

Daniel Polani

1 Introduction

The following is a very brief overview over some functions of XGrabtor. It does not attempt to be a complete introduction in either XGrabtor or XRaptor and has only been created to provide a preliminary guide for usage of XGrabtor. For XRaptor, a detailed manual is available under the XRaptor web page. It describes the dynamics of the scenarios as well as their C++ interfaces.

Any more detailed account of the functions of the graphical programming environment XGrabtor will unfortunately have to wait for future releases. Of course, you are welcome to contribute.

2 Basic Philosophy

The main idea of XGrabtor is to smoothen further the XRaptor learning curve and allow to work with (currently the ant) scenarios without having to know C++ and the C++ interfaces of XRaptor, just using a graphical user interface. Consequently, the whole interface is graphical based. In this preliminary manual, we will not explain all the details of the software, but give only an overview over the nodes.

3 List of Nodes

Nodes can be created by right-clicking the mouse. The nodes are grouped into the following types:

- sensors
- engines
- math functions
- delays

3.1 Sensors

3.1.1 Sensed Objects

The *sensed objects* node is used to select what type of objects an agent (ant or queen) can see around it. To fully understand it, one must see that in XRaptor the objects are seen by the agents like on a radar screen, in a circle around them, relative to their position and orientation. There is a limited distance to which an agent can detect objects. Objects outside of that range cannot be detected.

The *sensed objects* node is probably one of the most complex nodes in the system, because it does several things at once; it saves the users the hassle to write object selection mechanisms on their own. And that's how it works:

The node has one input and one or more output connections. The input contact can be used to specify a team to which an ant belongs. The output contacts give indices of objects detected.

On creating the node, a popup menu allows to choose the

- object types (ant, queen and food)
- the number of objects to be detected
- and whether the group (team) will be selected or antiselected for.

To understand the operation in detail, ignore the input contact for a moment. A *sensed objects* node shows in its label a list of object types which it is able to recognize. The output contacts then represent all objects matching one of these types, sorted from top to bottom according to their distance from the current agent. The top output contact therefore denotes the closest object of the given type.

As example, if the selection type is `ant|food`, the *sensed objects* node will recognize any object around it that is either an ant or a food object. Therefore, the output contacts will, from top to bottom, give the index for the closest object that is ant or food, the second contact the index for the second closest object that is ant or food and so on. This, until the last contact is reached. If there are fewer objects in the viewing range than there are output contacts, the output contacts for which there is no corresponding object return an "undefined" index.

If the input ("group", i.e. team) contact is connected to some output contact specifying a group, it selects for objects belonging to that group (team). By that, one can selectively choose to concentrate on ants from a specific group. A very typical input would be for instance the agent's own group. Then the *sensed objects* node will show only agents from one's own group. Food particles will always be selected for because they have no group property.

It is possible to switch on an *antiselection*. In that case, the input contact is called "antigroup" instead of group and the teams selected for are those whose group is *not* given in the contact. In particular, if the input is set to the agent's own group, the sensor detects only enemy ants (and, of course, food if specified).

Note that "antigroup" has the opposite logic of "group". If no input is given, a "group" contact will detect agents of any team. An unconnected or undefined "antigroup", however, will not detect any agent.

Note: the output of *sensed objects* node is an index denoting an object. It would itself be useless unless one can read out specific properties of the given objects. Nodes that provide suitable functionality are

- relative position
- pheromone sensor
- team/group.

3.1.2 Relative Position

The relative position node takes an object index as input and returns its coordinates relative to the current agent as output. Thus, one typically connects it to one of the output contacts of an *sensed objects* node.

Its output contacts, x and y denote the coordinates of the object relative to the current agent's position and orientation. In short, x denotes the agent's coordinate in front of the agent y towards the left of the agent. An observed agent at position $(x, y) = (20, 10)$ would be reached by the current agent stepping 20 units exactly forward (in its own coordinate system) and then jumping left 10 units. For more details, consult the XRaptor user manual.

If the input is not given or the "undefined" index, then $(x, y) = (0, 0)$.

3.1.3 Pheromone Sensor

Like the relative position node, the pheromone sensor accepts an object index (as produced by a *sensed objects* node) as input and returns the pheromone flag set by the given object. The flag is an integer numeric value between 0 and 4 (in the current implementation of the scenario). Its standard value is 0, but it can be set freely by the agents (see Sec. 3.2 below), and its semantics is entirely up to the agent designers.

3.1.4 Team/Group Sensor

Like the relative position node or the pheromone sensor, the team/group sensor accepts an object index (as produced by a *sensed objects* node) as input and returns the team/group index the given object belongs to. Its output can be plugged into the group/antigroup input of another *sensed objects* node.

If the input of the team/group sensor node is unplugged or the "undefined" index, the node returns the agent's own team/group code. This way the agent can determine its own group.

3.1.5 Hunger

The hunger sensor is not connected to *sensed objects* node. It is an independent sensor and returns a real value between 0 and 1 denoting the hunger status of the current agent. A large value (close to 1) means that the agent is going to die very soon, a value close to 0 means that the agent is well fed.

3.2 Actuators

3.2.1 Engine

The engine node controls the movement of the agent. The inputs into the two engines are real values between -1 and 1. If both values are the same positive value, the agent moves straight ahead, if the same negative ones, it moves straight back, if they are different, the agent may also move around. For details of the dynamics, see XRaptor manual.

3.2.2 Gobble Node

The gobble node accepts a real value. If that value is larger than 0.5, the agent tries to eat if it is close to a piece of food, if it is smaller, it will not eat the food and just collide with it. Eating a piece of food is the only way an agent can replenish its energy. See also the gobble action in the XRaptor manual.

3.2.3 Spit Node

`Note:` This function is only available to ants, not to queens.

This node is an engine which takes some of the agent's life energy and reconverts it into a piece of food. The amount of energy is a real number used as input for the spit node.

This process is lossy, so if the agent spends E energy units, the created piece of food only carries $E/2$ units at its creation.

Note: the following statement is specific to the particular dynamics of your current implementation. Numbers may vary. A piece of food rots away at 30000 units. So, to create a piece of food that remains in the simulation for a while, an agent has to spend more than 60000 units, otherwise the piece of food will rot away right away and the simulation will seem to do nothing. As a good rule of thumb, a spit amount of 100000 is probably around the order of magnitude which you may want to consider.

Note: the network created represents the complete, instantaneous calculation and action performed during one simulation cycle. Having a constant input of, say, 100000, connected to a spit node will create some pieces of food and cause the agent to die almost immediately because of hunger. If that is not what you want, you must control the "spitting" action.

3.2.4 Pheromone Flag

Accepts a real value between 0 and 4 and rounds it to the closest integer which is then used as a flag. This flag serves as a signal to other ants and can be read out by other ants using the pheromone sensor (Sec. 3.1.3). See also XRaptor manual.

3.2.5 Child Node

Note: This node only works for queen agents.

It accepts a real number as input. If that number is larger than 0.5, a child ant is created. Note that, similar to spit, a continuous value above 0.5 will have the queen immediately create a number of ants and then die. If that is not what you want, you have to stop the queen creating ants at some point.

3.3 Math Functions

3.3.1 Constant Node

This node creates a constant value at its output. The value can be edited without having to disconnect and delete the node.

3.3.2 Multiplicator

There are two types of contacts, index and value ones. Two indexes cannot be combined, however different incoming value connections can. By default, they are combined additively, *with exception of the multiplicator node*. If several connections converge on a multiplicator input connection, the output of the node returns the *product* of these.

In particular, you can use the multiplicator node as a *gate* to control whether to let a real value through or not.

3.3.3 $-x$

This node inverts the sign of a real number.

3.3.4 $1/x$

This node returns the multiplicatory inverse of the input node.

3.3.5 Step

This node takes any real value as input and returns 0 if the input value is negative and 1 if the input is nonnegative (≥ 0).

Note: It has been identified as a design error that "step" does not return 0 if the input is exactly 0. However, for sake of the consistency during coursework phase, this property is not changed at this time.

3.3.6 Index Gate

Since indexes cannot be arithmetically combined as values can, it is not possible to use an arithmetic construct like the multiplicator to control the flow of an index through the network.

Index gate allows to do that. If the "gate" input is nonnegative, then the "in" index is propagated to the "out" contact. If not, the "out" contact takes the "undefined" index value.

3.4 Delays

As mentioned above, a network denotes an instantaneous flow of information from sensors to actuators. However, no loops are possible with the above nodes, and thus, no internal memory. The "delay" nodes allow to delay the processing of a piece of information until the next simulation cycles. Delays (and only them!) can be part of a loop.

3.4.1 Delay Value

This node delays the transfer of an input value to the output contact by one simulation cycle. For instance, if in the initial simulation step, the input is 1, then during this time step, the output is 0 (the default value). Only in the next time step the output value will be one.

The output can be fed back into the input and affect computation for the next step, if required. You will need to use a strategy along these lines, as the only way to create counters and memory.

3.4.2 Delay Index

The same thing as delay value, but separate for index, as indices require a different handling and do not allow arithmetics.