# Behavior Tree v.4.3 and Groot2

## Behavior Tree v.4.3

Book: https://arxiv.org/abs/1709.00084

Good Tutorial (v3): <https://www.youtube.com/watch?v=kRp3eA09JkM>

Documentation: https://www.behaviortree.dev/docs/intro

Für Groot2 Pro License Request: <https://airtable.com/appqTlC23AiSoB4Fd/shrsC33d8feHZmKXs>

Sourcecode: https://github.com/BehaviorTree/BehaviorTree.CPP

git clone https://github.com/BehaviorTree/BehaviorTree.CPP

Installed with:

mkdir build ; cd build

cmake ..

make

sudo make install

CMakeLists for CPP, ROS1 and ROS2: https://github.com/BehaviorTree/btcpp\_sample

## Install BehaviorTree v4 Package for ROS Noetic

- macht mal das erst später – wenn ihr Tutorial 02\_simple\_ball\_example\_cpp und 03\_blackboard\_ports\_example\_cpp gemacht hat – hatte ja wie gesagt dann Cmake errors nachdem ich auch die ros-noetic behavior tree

sudo apt-get install ros-noetic-behaviortree-cpp

## Groot2

Download Groot2 from <https://www.behaviortree.dev/groot/>

To have an desktop application for Groot2:

- cd /usr/share/applications

- create file in this folder – sudo touch groot2.desktop

- add the following to the file – sudo vim groot2.desktop – exit with ESC and then enter :wq

[Desktop Entry]

Name=Groot2

Exec=/home/**user**/Groot2/bin/groot2

Icon=/home/**user**/Groot2/bin/logo.png

Type=Application

Categories=Development;TextEditor;

Hint: **change** **user** with your own **user –** or check path where you installed Groot2

# Some basic Behavior Tree Knowledge

### Build Behavior Tree in C++ - what do you need?

1. CPP file – BT functionality (Code for the execution nodes, actions, conditions etc.)
2. XML file – BT Design (How is the control flow – Control Flow nodes)
3. CMakeLists.txt – CPP build

### Control Flow nodes (all tick from left to right)

1. Sequence (→) – ticks child nodes until != SUCCESS to tick next one

finish with SUCCESS to tick next one

**In short:**

If a child returns SUCCESS, tick the next one.

If a child returns FAILURE or RUNNING, then no more children are ticked, and the Sequence returns FAILURE or RUNNING.

If all the children return SUCCESS, then the Sequence returns SUCCESS too.

1. Fallback (?) - ticks child nodes until it finds SUCCESS or RUNNING

- will not tick the next child node if one SUCCESS or RUNNING is found

**In short:**

If a child returns FAILURE, tick the next one

If a child returns SUCCESS, then no more children are ticked and the Fallback returns SUCCESS.

If all the children return FAILURE, then the Fallback returns FAILURE too.

1. Parallel (Doppelpfeil) – routes all child nodes

returns SUCCESS if M (defined by user) nodes return SUCCESS

return FAILURE if n-M+1 (where n is number of child nodes) return FAILURE

otherwise return RUNNING

1. Decorator () - added functionality on childnode e.g. it can fail certain # of times

(Delay, Force Failure, Force Success, Inverter, KeepRunningUntilFailure, LoopDouble, Loop String, Precondition, Repeat, RetryUntilSuccessful, RunOnce, Timeout)

### Execution nodes

1. Action – carry out action – return either SUCCESS, RUNNING, FAILURE
2. Condition – check condition – return SUCCESS, FAILURE

### Synchronous vs. Asynchronous nodes

- Synchronous always return SUCCESS or FAILURE

- Asynchronous return SUCCESS, RUNNING or FAILURE – **non-blocking**