# Analyse

1. “A Behavior Tree (BT) is a way to structure the switching between different tasks in an autonomous agent, such as a robot or a virtual entity in a computer game.” - behaviortree.dev.
2. The goal of this document is to research the best way to design the traffic control assignment using BT.
   1. *What language and libraries to use*

To design behavior trees, you could use the program Groot, Groot is a visual behavior tree editor. More information can be found here: <https://github.com/BehaviorTree/Groot>

After making the design in Groot, Groot exports the BT structure to an xml file. This file can be imported into the code.

The whole Groot design can be found in the file “traffic\_control\_ontwerp2.xml”

There are two languages which have a fully implemented library. These are C++ and Python. C++ has the library behaviortree.cpp and Python has the library py\_trees. For this assignment we need to decide which has the most benefits and is most useful.

**Pros and cons C++/Behaviortree.cpp**

Pros

* The official ROS 2 navigation stack uses this library in its [BT Navigator](https://navigation.ros.org/configuration/packages/configuring-bt-navigator.html) feature. By understanding this library, you incidentally understand more of ROS2 underlying code. Behaviortree.cpp is also more popular than py\_trees. Behaviortree.cpp has 1.3K stars on Github whereas py\_trees only has 254.
* C++ based means it is a lot quicker than Python. This is useful for time critical scenarios (working with milliseconds).
* The designer Groot has been designed to generate an XML file to be used in behaviortree.cpp. The XML file is a file that defines the structure of the behavior tree. The only thing that needs to be done in the code is define the nodes.

Cons

* C++ is a difficult language. You need to keep things like pointers, header files and such in mind. This slows down development time. Python is a lot simpler.
* As the structure is defined at compile time using the XML file, it is static. This makes it difficult to change it during run-time. There are ways to program this, but it is not documented well and has no support in Groot.

**Pros and cons Python/py\_trees**

Pros

* Python is a lot simpler to implement than C++ as it is an interpreted language. This makes development time a lot faster.
* Although py\_trees is not as popular as behaviortree.cpp, it is fully implemented.

Cons

* Some of the terminology and design paradigms are a little bit different from the [Behavior Trees in Robotics](https://arxiv.org/abs/1709.00084) book. For example, instead of Fallback nodes this library uses Selector nodes, and these behave slightly differently.
* Python is not useful in time critical (working with milliseconds precision) scenarios.

Conclusion

For this research we will use the behaviortree.cpp library.

Since it is our first complete research about behavior trees, we will use the C++ library. We do this for the following reasons:

* Groot support. We do not need to worry about porting the tree to python.
* C++ is the standard for ROS2. Since the ROS2 environment is our development goal this seems fitting.
* Unlike py\_trees it follows the terminology and design paradigms of the behavior trees in robotics book.

## Problems with C++

Documentation incomplete.

It sais <https://www.behaviortree.dev/sequencenode/> sequence node returns running it ticks again. No were it could be found that instead of the default SYNCACTIONNODE you need to have the CoroActionNode to return a RUNNING status.

Blackboard is weird and inconsistent.

HIER VERTELLEN OVER DAT BLACKBOARD SOMS WEL SOMS NIET WERKT. DOCUMENTATIE IS NIET DUIDELIJK OOK.

Best way to find information about nodes is in Groot. Items like shared blackboard and checkblackboardstring

VERTELLEN HOE CPP KUT IS

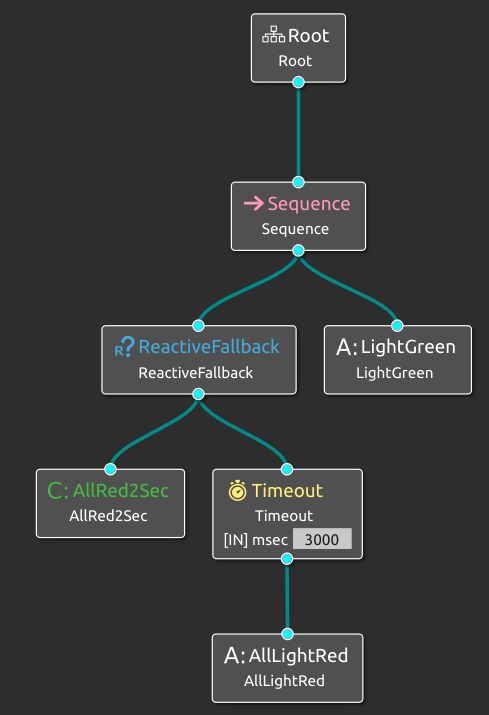
MISSCHIEN PYTHON LIB?

# Design

During the design process I found two ways behavior trees could be used for the project; for small tasks or the whole process using BTs(behavior trees).

## Small tasks

The first way is to utilize behavior trees for only small tasks.



Each task has its own simple BT(behavior tree). These small trees can be used within the code to execute an task. For example:

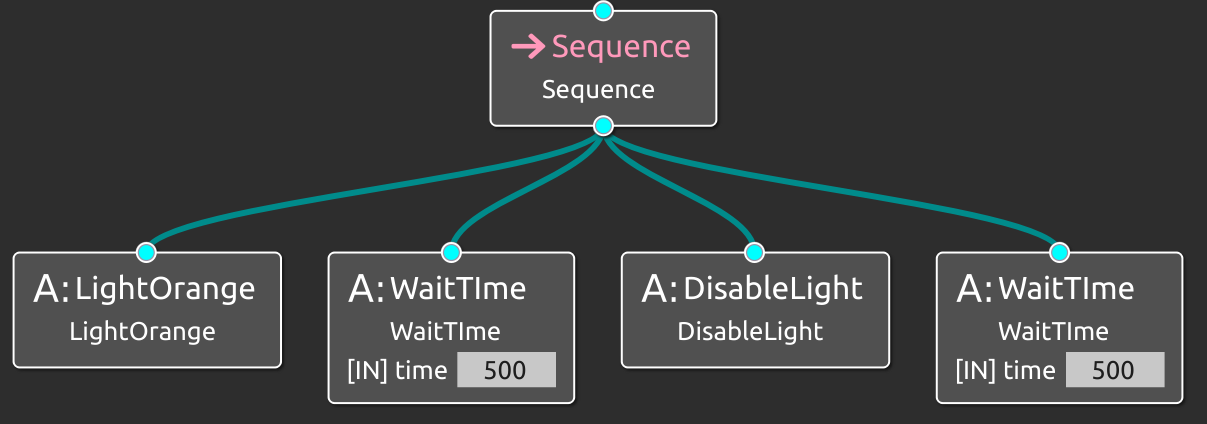
**Turn traffic light green:**

In this example we use a BT to check if all traffic lights have been turned off for at least 2 seconds. When this task has been completed, its turns the lights green.

This BT can be used combined with other kinds of code to just finish the task.

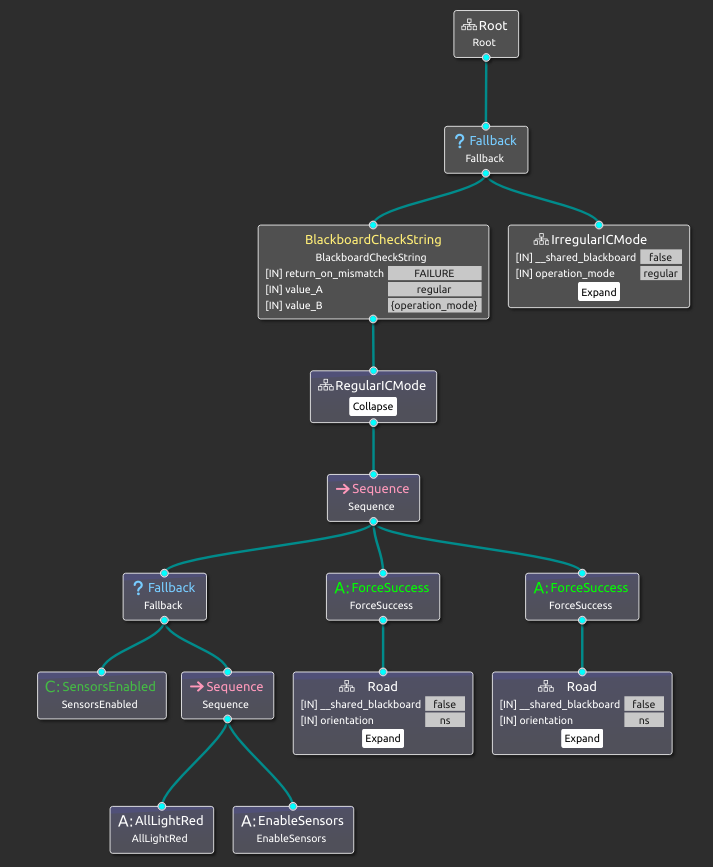
**Flash orange:**

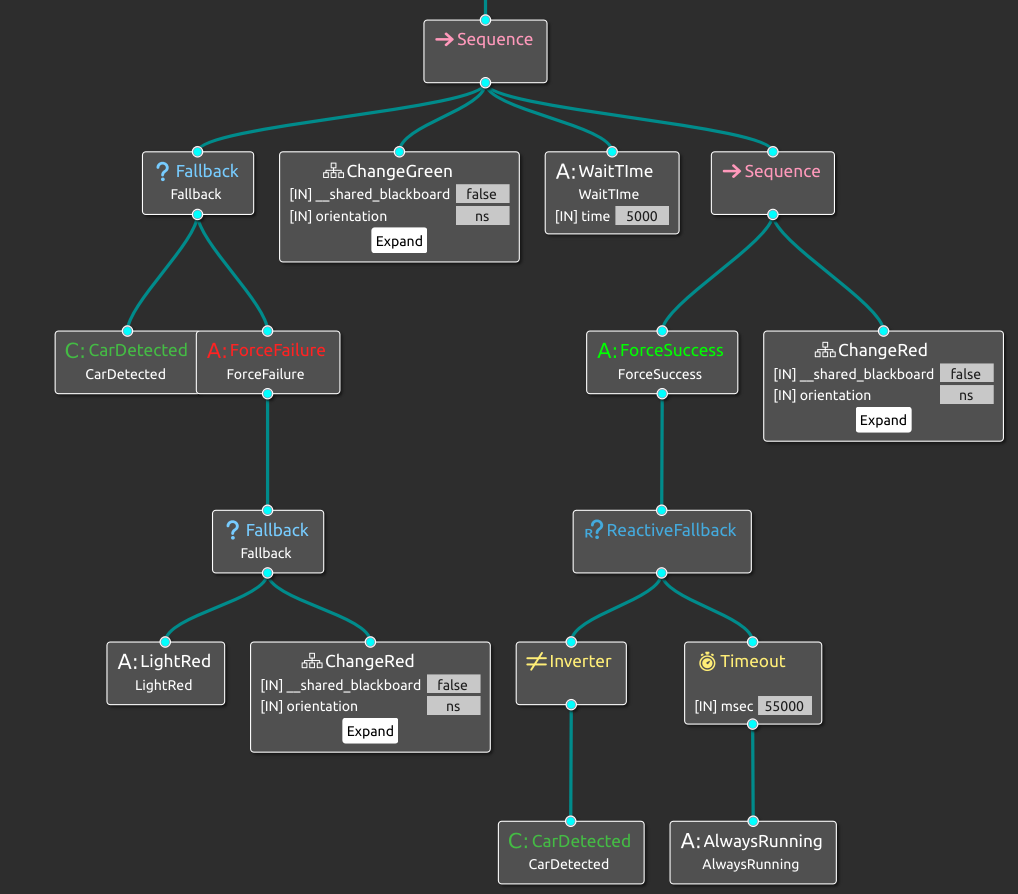
Another simple example where a tree leaf flashes the light orange

****

## Whole system

A different usage for behavior trees is to create the whole program logic using BTs.

This example shows how BT can be used to control the whole system. At the top we check if the system is in regular or irregular mode. For this scenario its regular. First we make sure the sensors are enabled. After it starts ticking the road leafs.

Each road checks if cars are detected. If there are no cars it returns ERROR and the next road leaf is ticked. If there are cars it changes the lights to green, waits a bit and than stays green for a maximum of 60 seconds or if there are no cars left.

# Testing

Testing is very difficult with behavior trees. As the name suggests, the tree ticks leaves based on the behavior a.k.a. situation. It is difficult to test every possible scenario. The bigger the tree gets, the harder it is to find all possible scenarios. Even if you think you have found all possible scenarios there is a lot of room for human error. At the moment my best guess to test the trees is to make sure each tree is as small as possible and make an counter in the blackboard that checks how often a node is called. This can be compared what the user expects to happen.

# Advice

I cannot recommend using BTs as the whole architecture of a project. It is annoying how an individual node is “hardcoded”. It is difficult to give it the right context. For example, if I want to change a light color with the node “SetLight”, I first have to set the global ports “orientation” and “color” before I can call that node. This makes it very cluttered and redundant. This being said, I think BTs can be usefull for small tasks. Think of things like doing an action with a robot arm. This should be fairly simple as in open arm, move forward to pos x, close arm and return.

ALLES DOCUMENTEREN DAT IK HEB GEDAAN IK DOE GOEIE DINGEN MAAR DOCUMENTEER ZE NIET

HIER VERGELIJKEN VAN WAAROM NIET WAT KAN BETER NIET OVERSPRINGEN NAAR ANDERE CONTEXT ZOALS EEN ROBOT

in

# sources

<https://automationsr.com/introduction-to-behavior-trees/> ← pros cons BT libraries