* 1. The first step of creating the traffic control task using behavior trees is creating a design.
  2. During the designing process I found two ways to design a system with behavior trees.

# 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.
3. For this assignment I will use the behaviortree.cpp library. <https://www.behaviortree.dev/>
4. The reason for using this library and thus cpp as programming language instead of python is mainly because a graphical editor (groot) is made for the cpp library and I’m simply curious.
5. The design is made in the program Groot. Groot is a visual behavior tree editor. More information can be found here: <https://github.com/BehaviorTree/Groot>
6. After making the design in Groot, Groot exports the BT structure to a xml file. This file can be imported into the code.
7. The whole Groot design can be found in the file “traffic\_control\_ontwerp.xml”

VERTELLEN HOE CPP KUT IS

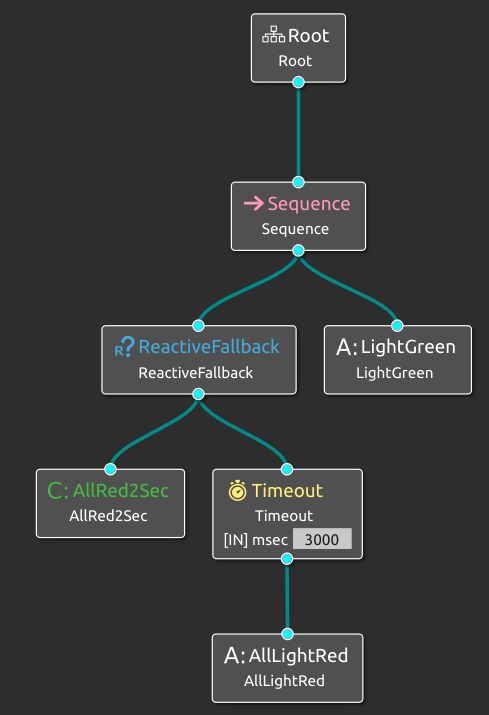
MISSCHIEN PYTHON LIB?

# 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:

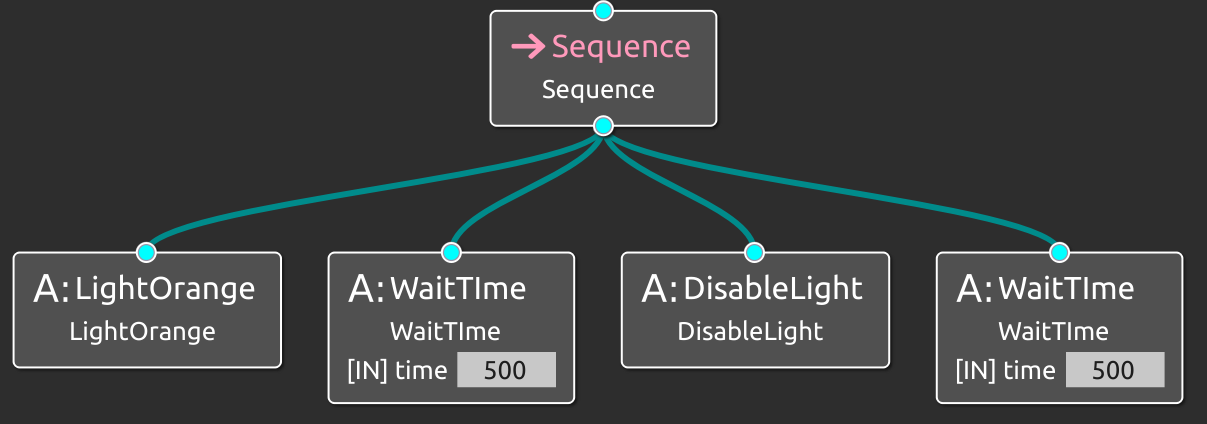
**Turn traffic light green:**

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.

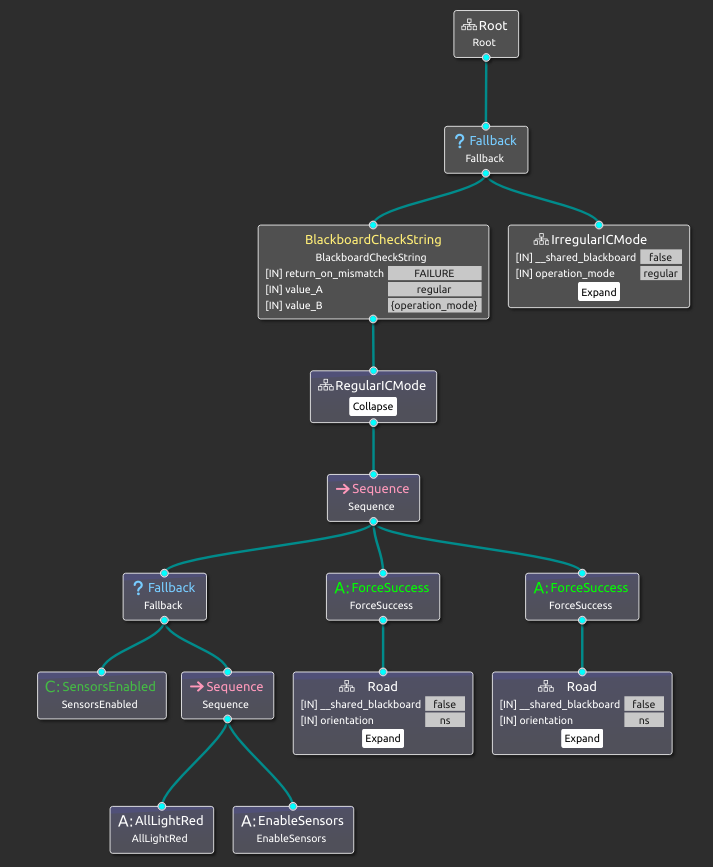
**Flash orange:**

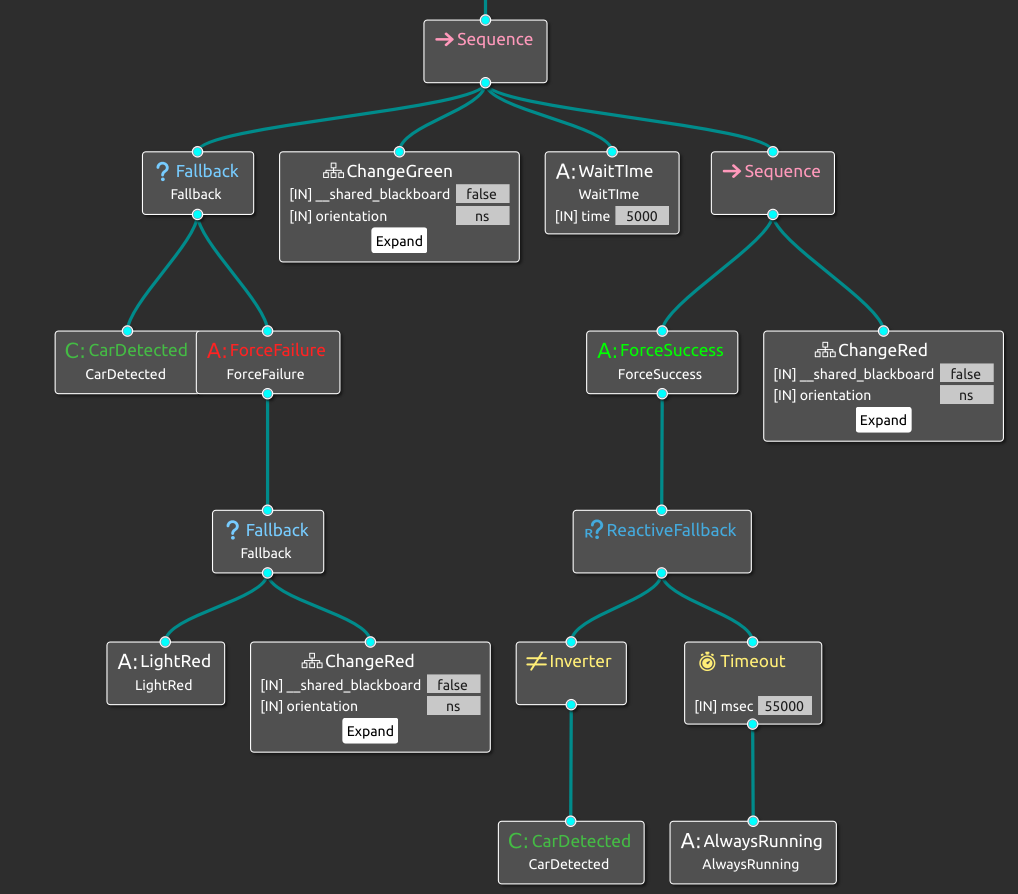
Another simple example where a tree leaf flashes the light orange

****

## 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