BT Studio: a ROS Behaviour-Tree web IDE

Asociación de Robótica e Inteligencia Artificial JdeRobot

CIF G88145909

josemaria.plaza@gmail.com, oscar.robotics@tutanota.com,
javizqh@gmail.com

- Introduction and motivation
- BT-Studio
- How it works?
- For the future
- Examples

Introduction and motivation

Making Behavior trees more accesible

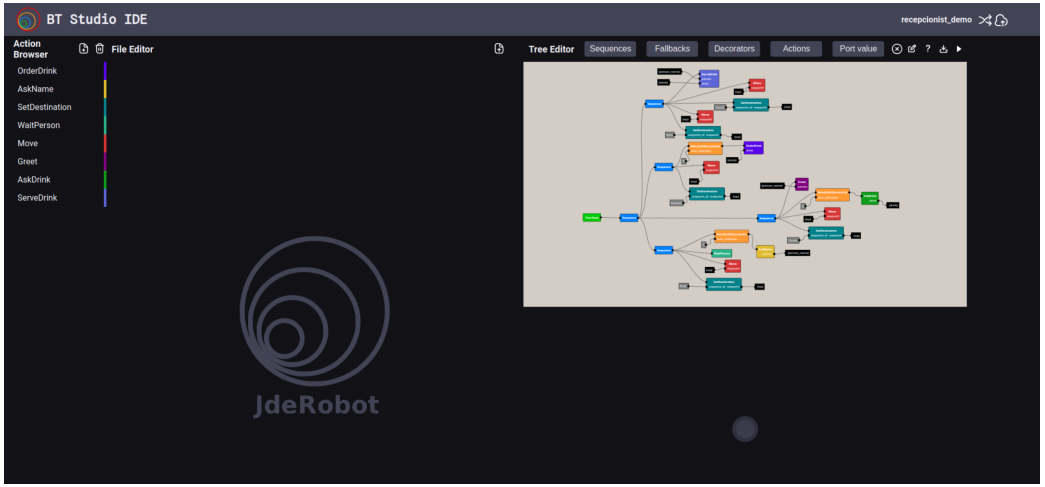
- Uo to date with modern developer trends like the use of Behavior Trees in IA
- Tries to improve on already established tools, for example: Groot and Groot2.
- Built on top of py_trees for better compatibility.
- It tries to provide a similar experience to BehaviorTrees.CPP but for python.
- Fast and streamlined development of fairly complex applications with the 3.8 version according to BehaviorTrees.CPP.
- Reuse of behavior trees and modification in a graphical interface.

BT Studio

- It's primary objective is to facilitate the quick deployment of behavior tree-based robotic applications within ROS 2.
- Develop applications for ROS2 Humble
- Streamlines the process of creating a ROS 2 package
- Free and open-source

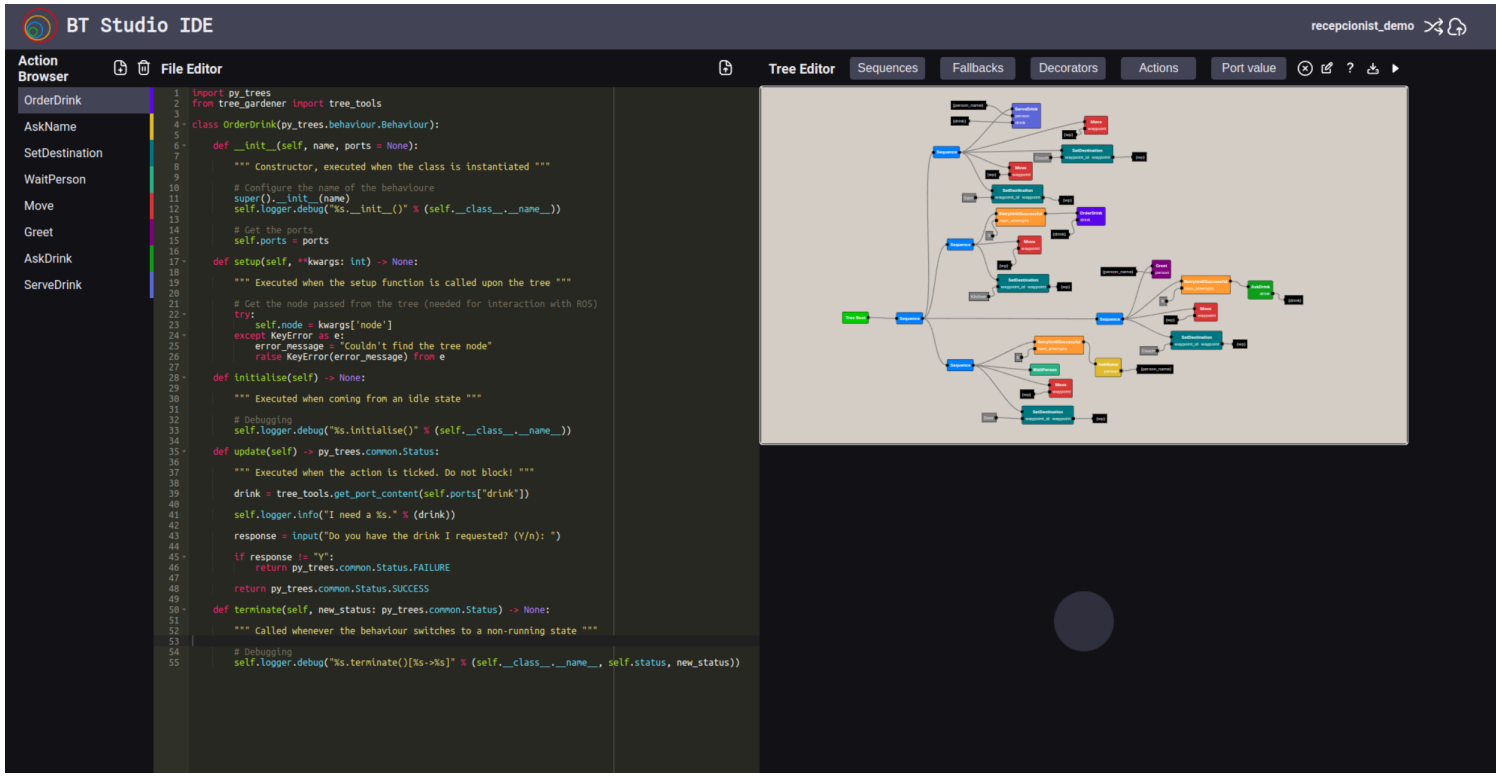
Features

- Manage multiple projects
- Programming in the Python language
- Edit the behavior tree actions in the diagram editor
- Define the behaviour tree structure using a graphical interface
- Both on real robots and on simulated robots (Gazebo, Webots...)



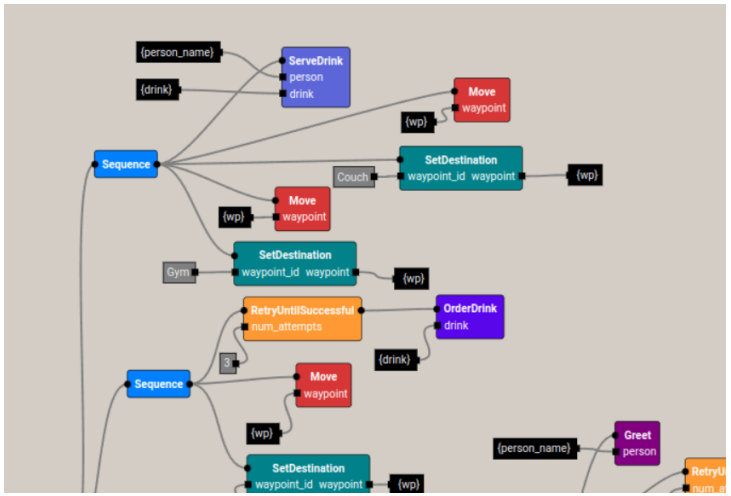
User Interface

- **Text Editor + BT Editor + Vnc Visualizer**
- Program the actions while modifying the behaviour tree



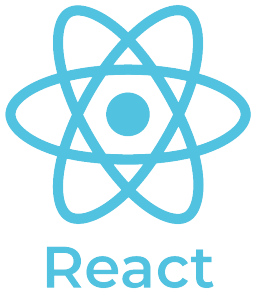
Behavior Tree Editor

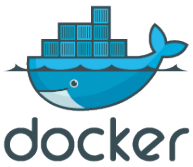
- Intuitive and reactive editor
- Customizable colors for each action
- Order from bottom to top
- Everything you need for developing BT based applications



Developers: How it works?

- Web tecnologies
  - backend: Django
  - frontend: React, HTML5, CSS
- Robotics tecnologies
  - ROS2
  - Based around py_trees
- DevOps tecnologies
  - Docker





Action Structure

- The structure is the same as py_trees actions

```python
import py_trees

class TemplateAction(py_trees.behaviour.Behaviour):

    def __init__(self, name, ports = None):

        """ Constructor, executed when the class is instantiated """

        # Configure the name of the behaviour
        super().__init__(name)
        self.logger.debug("%s.__init__()" % (self.__class__.__name__))

        # Get the ports
        self.ports = ports

        ...

    def setup(self, **kwargs: int) -> None:

        """ Executed when the setup function is called upon the tree """

        # Get the node passed from the tree (needed for interaction with ROS)
        try:
            self.node = kwargs['node']
        except KeyError as e:
            error_message = "Couldn't find the tree node"
            raise KeyError(error_message) from e

        ...

    def initialise(self) -> None:

        """ Executed when coming from an idle state """

        ...

    def update(self) -> py_trees.common.Status:

        """ Executed when the action is ticked. Do not block! """

        ....

        return new_status

    def terminate(self, new_status: py_trees.common.Status) -> None:

        """ Called whenever the behavior switches to a non-running state """
        ...
```

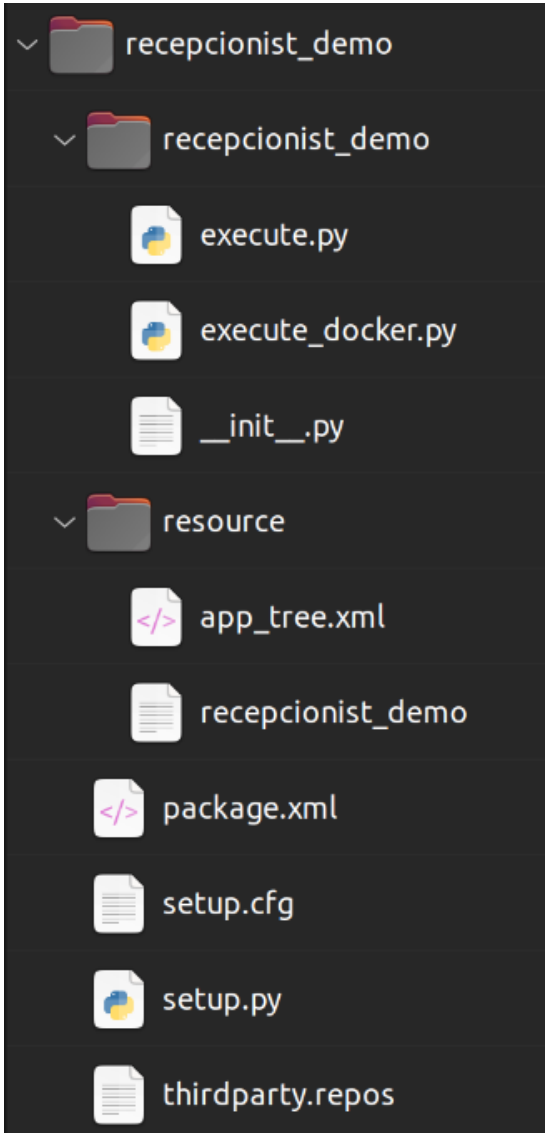
Traslation process

- Traslating from the user code and the diagram is done in the backend
- The 2 parts are combined into a xml single file divided into 2 sections: the BahaviorTree and the Code
- In the BehaviorTree section resides the Behavior Tree and is the same as what is generated by Groot2.
- The code section is used instead of external files for containing each action source code

Application Package

- ROS2 humble is needed
- A testing enviroment is provided with the Webots simulator and a tree execution visualizer as thirdparty repos.
- Compile and run the app using the executor provided
- The actions and behaviour tree are merged into a single xml source file.

- `app_tree.xml`: behavior tree and source code
- `execute.py`: launcher for the application
- `execute_docker.py`: launcher for dockerized execution
- The rest is the same as a basic ROS package



Dockerized execution

- Wait first to get it working
- Using the Robotics Backend docker for easy development and ready to use templates.

For the future

- Merge in Unibotics, online execution.
- Configure launchers

Working demos

Recepcionist Demo

- Wait for dockerized execution