BTstudio, a web tool for programming robots with Behavior Trees

JdeRobot

Asociación de Robótica e Inteligencia Artificial JdeRobot

https://jderobot.github.io

JoseMaria Cañas, Javier Izquierdo, Oscar Martínez

- Introduction
- Using BT Studio
- How has it been developed?
- Experimental validation
- Conclusions

Introduction

JdeRobot org

- Develops open source sw in Robotics and AI
- Started in 2018, 20+ members
- Projects:
  RoboticsAcademy, Unibotics, BTstudio, VisualCircuit...
- Activities:
  Google Summer of Code (2015,2017-2024), internships
- https://jderobot.github.io, YouTube, LinkedIn, Twitter

Making Behavior Trees more accessible for Robotics applications

- Development trend of using Behavior Trees in Robotics applications, fairly complex ones beyond simple reactive ones. Similar to HFSM.
- Goal: **to facilitate the quick deployment of BT-based robotics applications within ROS2**
- Inspired on already established tools: Groot and Groot2.
- Built on top of py_trees for better compatibility.
- Provides a similar experience to BehaviorTrees.CPP 3.8 but for Python.

BTstudio

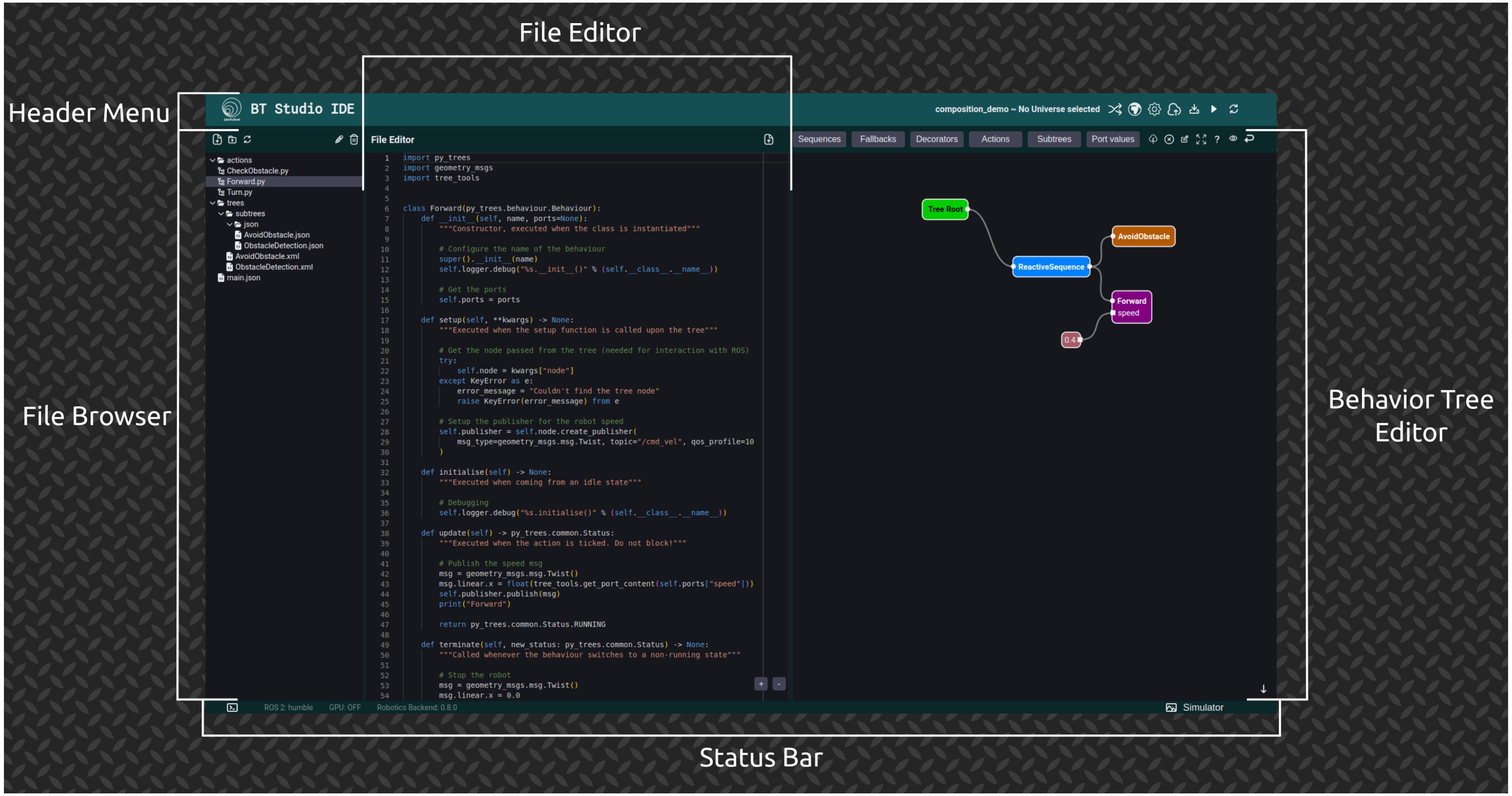
- Web based
- Python language
- Develop applications for ROS2 Humble
- Reuse of behavior trees and modification in a graphical interface
- Free and open-source
- Execution in a dockerized environment (*Robotics Backend*)
- (optional) Streamlines the process of creating a ROS2 package

Features

- Edition mode and Execution mode
- Behavior Tree **actions** are created in a *text editor* and a file navigator
- Behavior Tree **structure** is defined using a *visual editor*
- Integrated *execution viewer* and *execution BT monitor* for dockerized execution, VNC based
- Integrated *text console*
- Both on real robots and on simulated robots (Gazebo, Webots...)
- Manage multiple projects and multiple universes (worlds + robots)

User Interface in Edition Mode

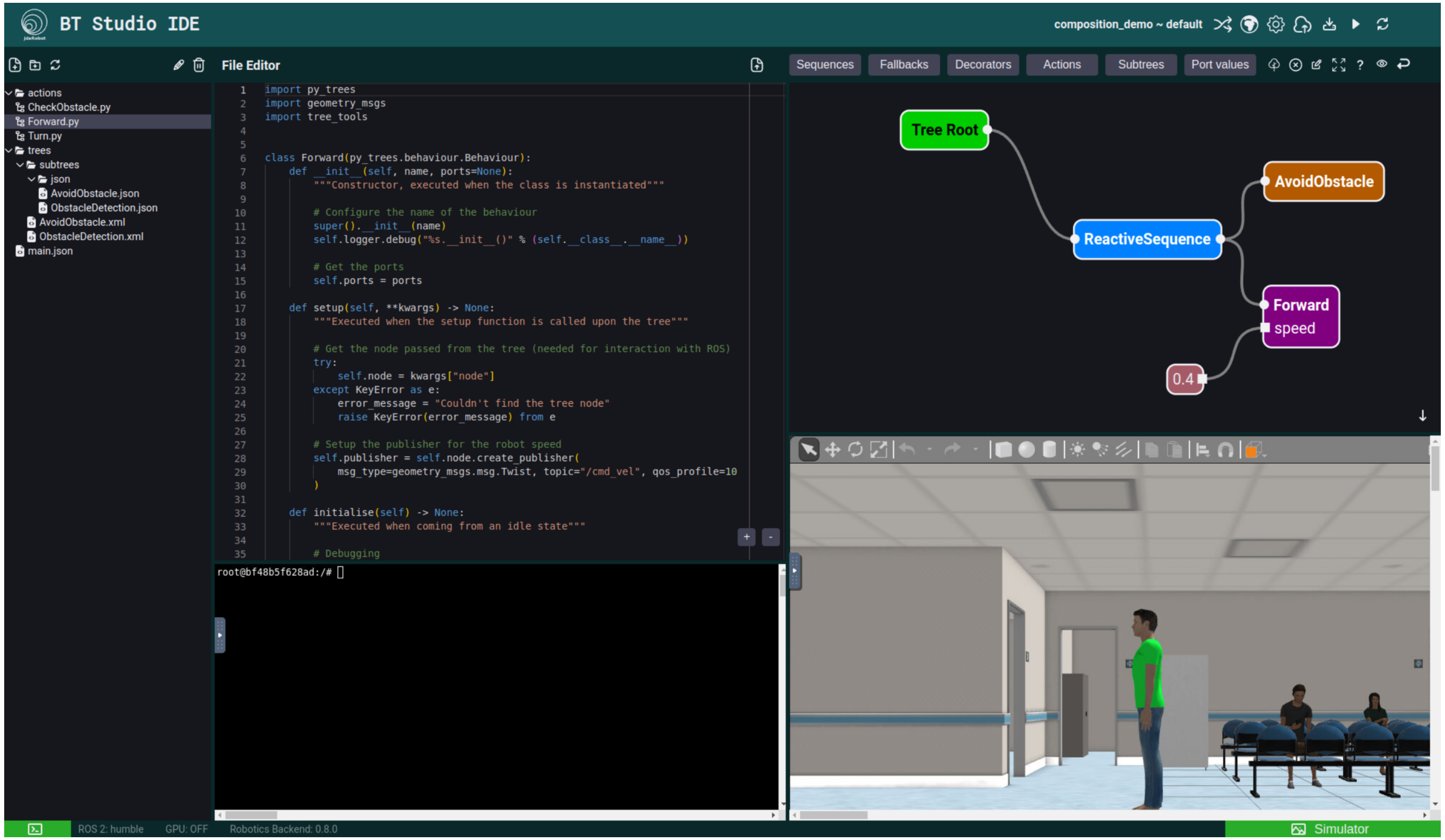
- *File Navigator + Text editor + BT visual editor*



```
import py_trees
import geometry_msgs
import tree_tools


class Forward(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) -> 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

        # Setup the publisher for the robot speed
        self.publisher = self.node.create_publisher(
            msg_type=geometry_msgs.msg.Twist, topic="/cmd_vel", qos_profile=10
        )

    def initialise(self) -> None:
        """Executed when coming from an idle state"""

        # Debugging
        self.logger.debug("%s.initialise()" % (self.__class__.__name__))

    def update(self) -> py_trees.common.Status:
        """Executed when the action is ticked. Do not block!"""

        # Publish the speed msg
        msg = geometry_msgs.msg.Twist()
        msg.linear.x = float(tree_tools.get_port_content(self.ports["speed"]))
        self.publisher.publish(msg)
        print("Forward")

        return py_trees.common.Status.RUNNING

    def terminate(self, new_status: py_trees.common.Status) -> None:
        """Called whenever the behaviour switches to a non-running state"""

        # Stop the robot
        msg = geometry_msgs.msg.Twist()
        msg.linear.x = 0.0
```

User Interface in Execution Mode

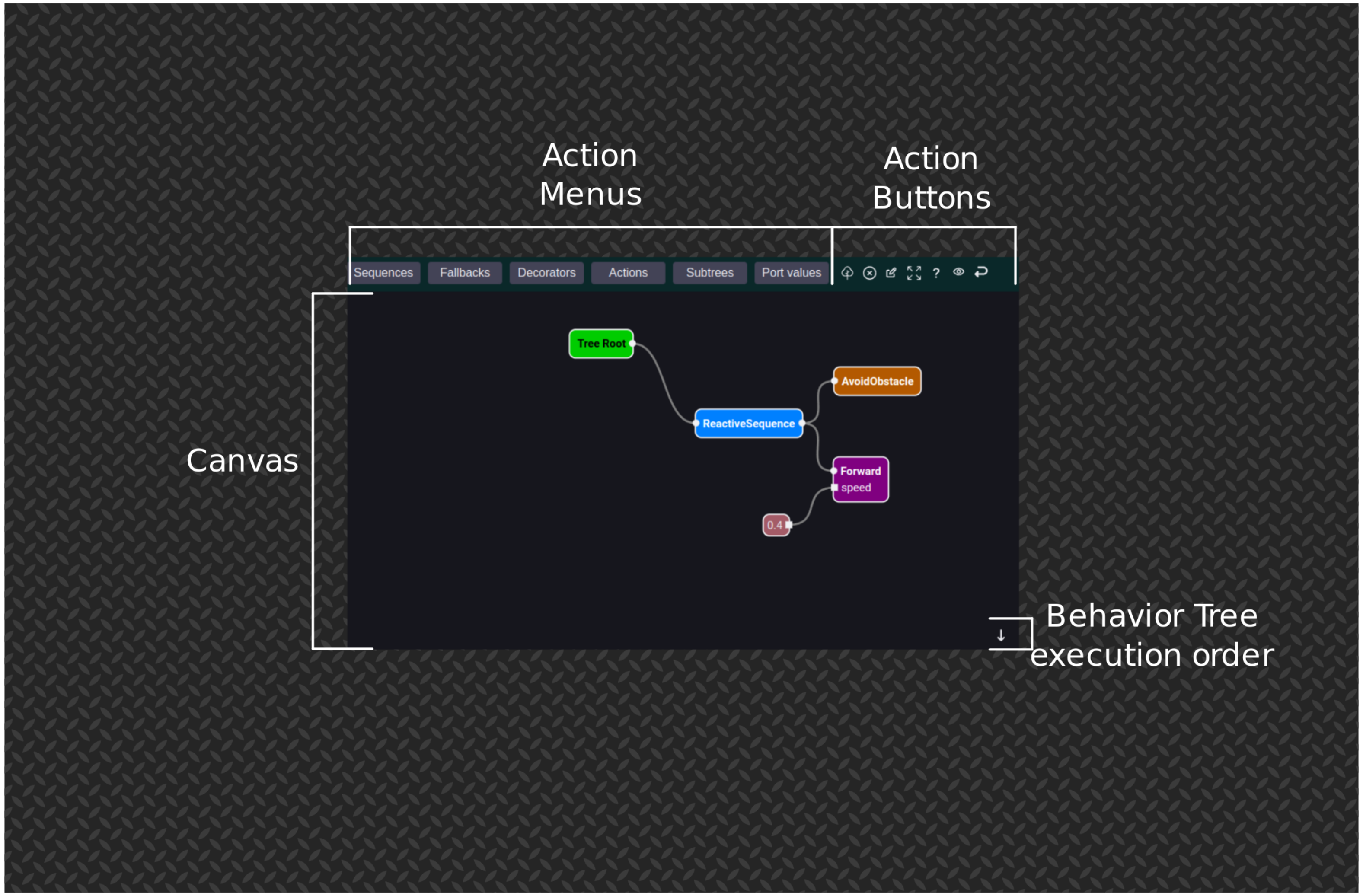
- *Text editor + Visual BT monitor + Console + Execution viewer*
- Fast feedback to the developer



```python
import py_trees
import geometry_msgs
import tree_tools


class Forward(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) -> 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

        # Setup the publisher for the robot speed
        self.publisher = self.node.create_publisher(
            msg_type=geometry_msgs.msg.Twist, topic="/cmd_vel", qos_profile=10
        )

    def initialise(self) -> None:
        """Executed when coming from an idle state"""

        # Debugging
```

```
root@bf48b5f628ad:/# 
```

Visual Behavior Tree Editor



- Intuitive and reactive editor
- Customizable colors for each action
- Customizable order of BT execution (bottom-to-top or top-to-bottom)
- Support for Subtrees and composition
- Everything you need for developing BT based applications

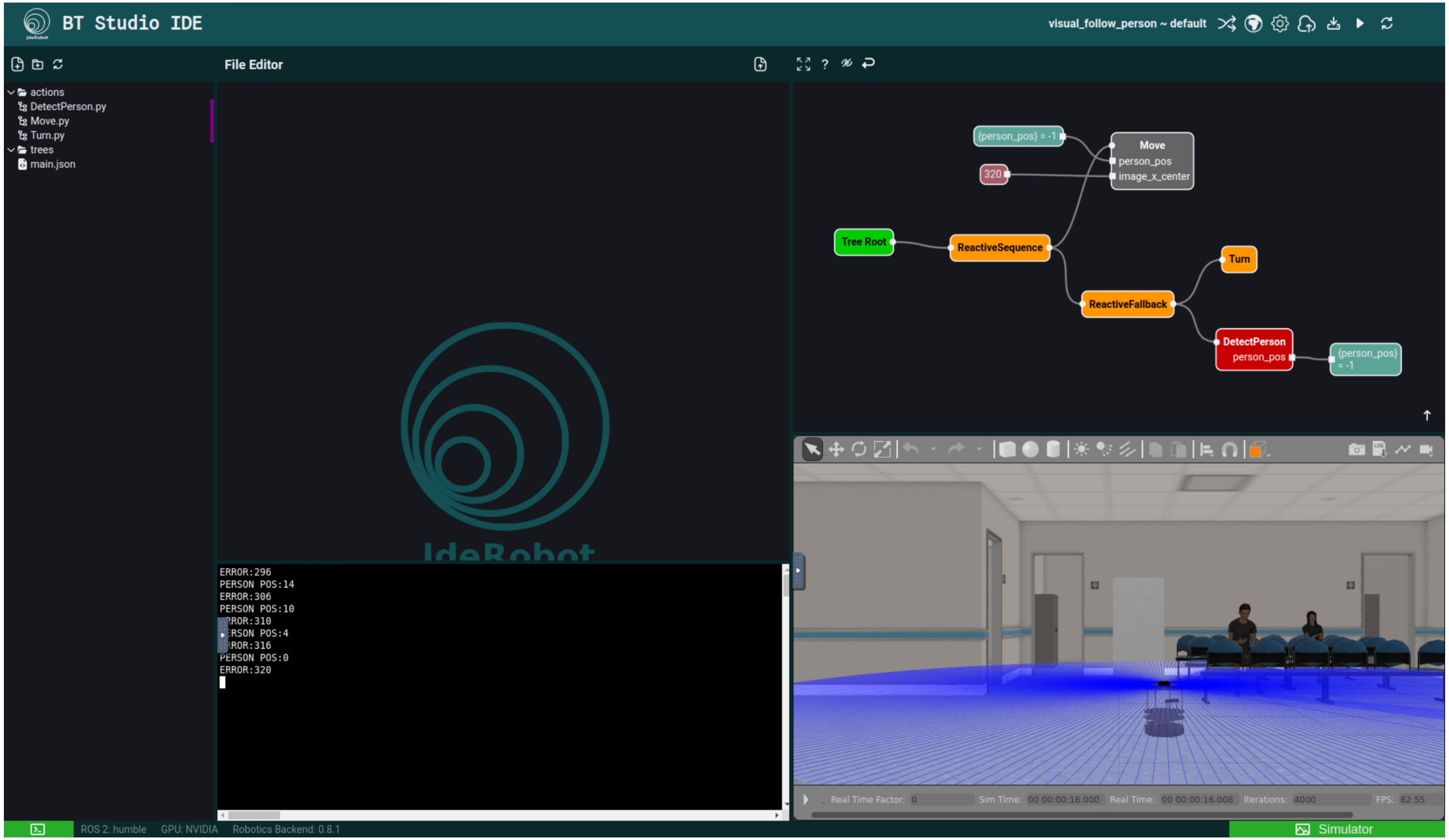
Subtrees

- BT composition
- Speed up the development of complex robotics applications
- Reusing of existing BTs for common functionality
- Library

Execution monitoring of the Behavior Tree

Monitor the execution status of the behavior tree from inside the dockerized execution

- Real time updates
- Move through the subtrees seamlessly
- Also displays the content of blackboard tags

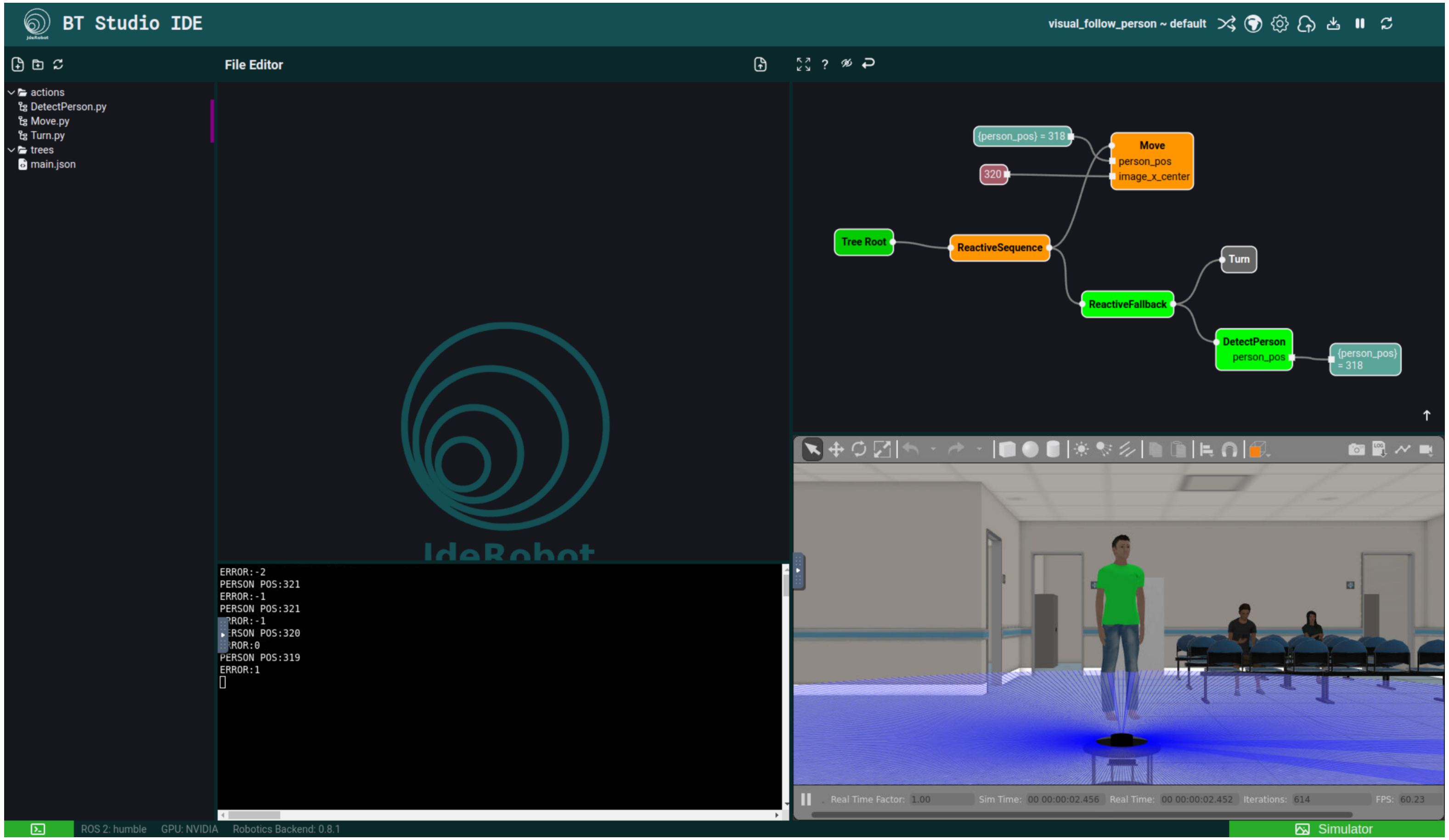


```
ERROR:296
PERSON POS:14
ERROR:306
PERSON POS:10
RROR:310
ERSON POS:4
RROR:316
PERSON POS:0
ERROR:320
```

Real Time Factor: 0 Sim Time: 00 00:00:16.000 Real Time: 00 00:00:16.008 Iterations: 4000 FPS: 62.55

Dockerized Execution

- Use multiple universes seamlessly
- Ready to use universes provided by the Robotics Backend
- Use your own custom universes
- Control the flow of execution: Run, Pause or Restart

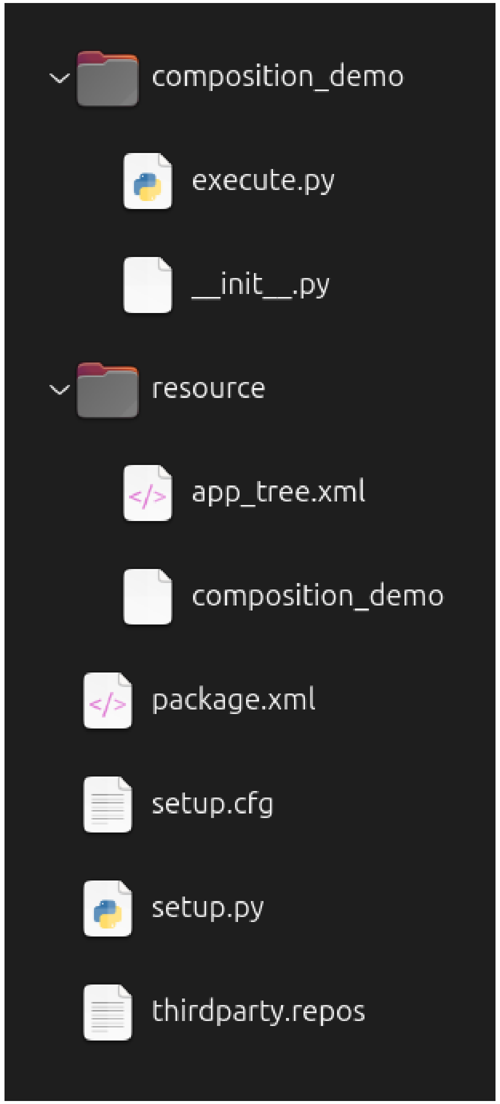


```
ERROR:-2
PERSON POS:321
ERROR:-1
PERSON POS:321
ERROR:-1
PERSON POS:320
ERROR:0
PERSON POS:319
ERROR:1
```

Local Application Package

- ROS2 Humble or Jazzy is needed
- A testing environment is provided with the Webots simulator and a tree execution visualizer as third-party repos.
- Compile and run the app using the executor provided
- The actions and behavior tree are merged into a single xml source file.

- `app_tree.xml`: behavior tree and source code
- `execute.py`: launcher for the application
- The rest is the same as a basic ROS2 package



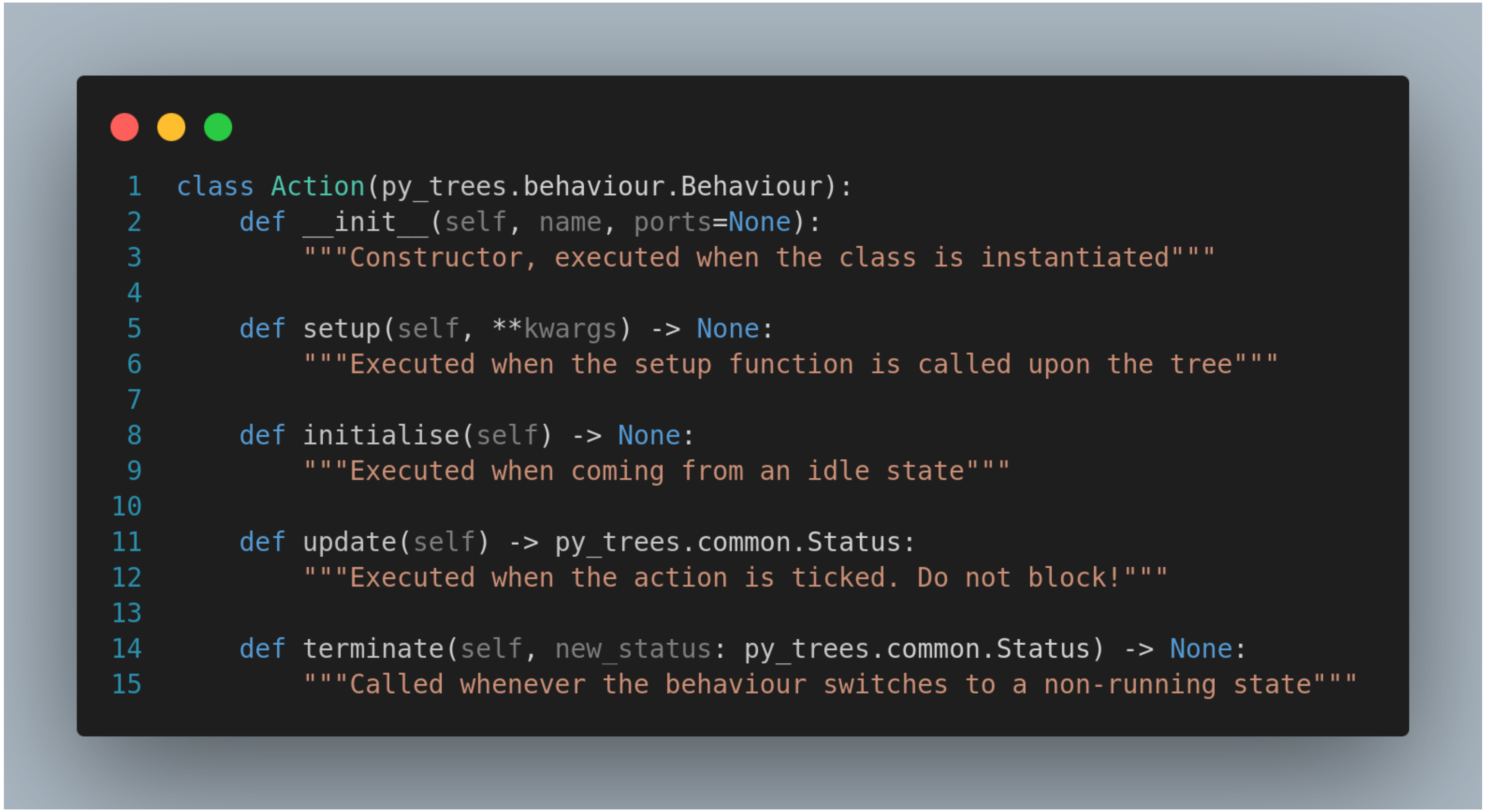
```
composition_demo
    execute.py
    __init__.py
resource
    app_tree.xml
    composition_demo
package.xml
setup.cfg
setup.py
thirdparty.repos
```

Developers: How does it work?

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

Action Structure

- The structure is the same as py_trees actions



```python
class Action(py_trees.behaviour.Behaviour):
    def __init__(self, name, ports=None):
        """Constructor, executed when the class is instantiated"""

    def setup(self, **kwargs) -> None:
        """Executed when the setup function is called upon the tree"""

    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!"""

    def terminate(self, new_status: py_trees.common.Status) -> None:
        """Called whenever the behaviour switches to a non-running state"""
```

Translation process

- Translating 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 BehaviorTree 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

Working demos

Follow Person Demo

- Video

Bump & Go Demo

- Video

Receptionist Demo

- Video

Conclusions

- BTstudio facilitates the quick deployment of behavior tree-based robotics applications within ROS2
- Integrated in Unibotics robot programming website

- Present BTstudio to the open source community and ROS community
- Develop more relevant robotics applications and subtree library
- Jump to BehaviorTree.CPP 4.X