Course of

Robot Programming with ROS 2

Day 2

4. Behavior Trees

ikerlan

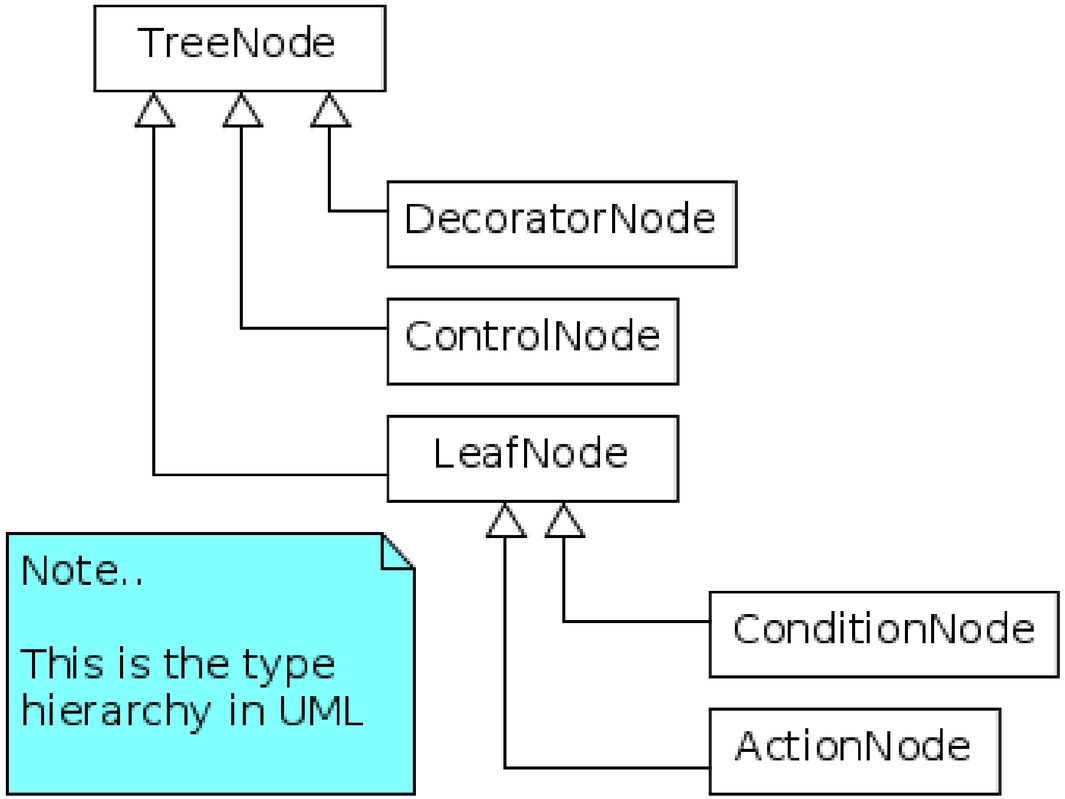
Universidad Rey Juan Carlos

Intelligent Robotics Lab

Behavior Trees?

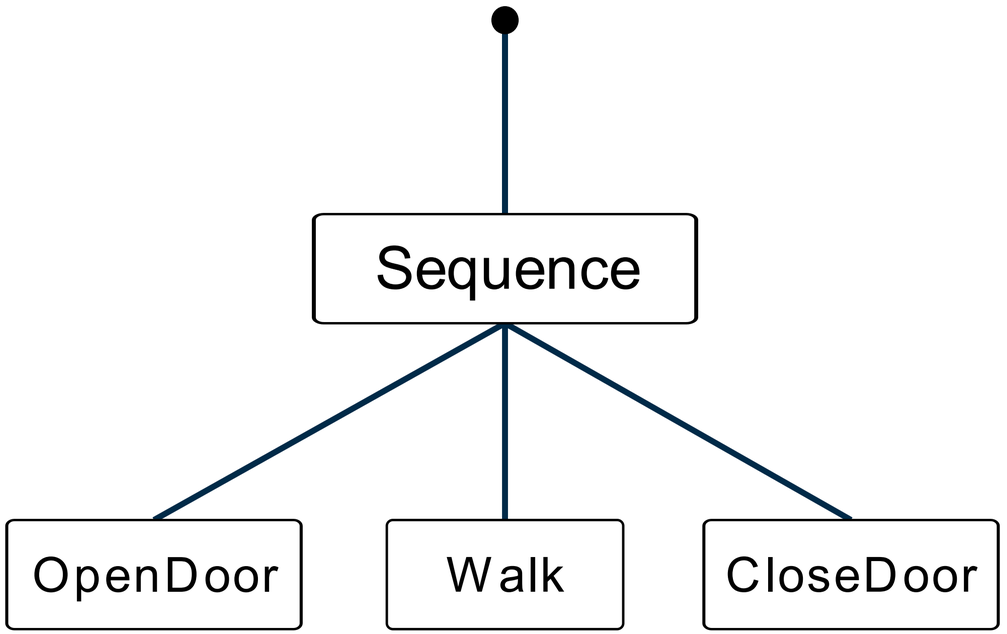
- Mathematical model for plan executions
- Describe agile switching between a set of tasks
- Allow modelling complex tasks
- Modularization, outperforming FSMs

Behavior Trees



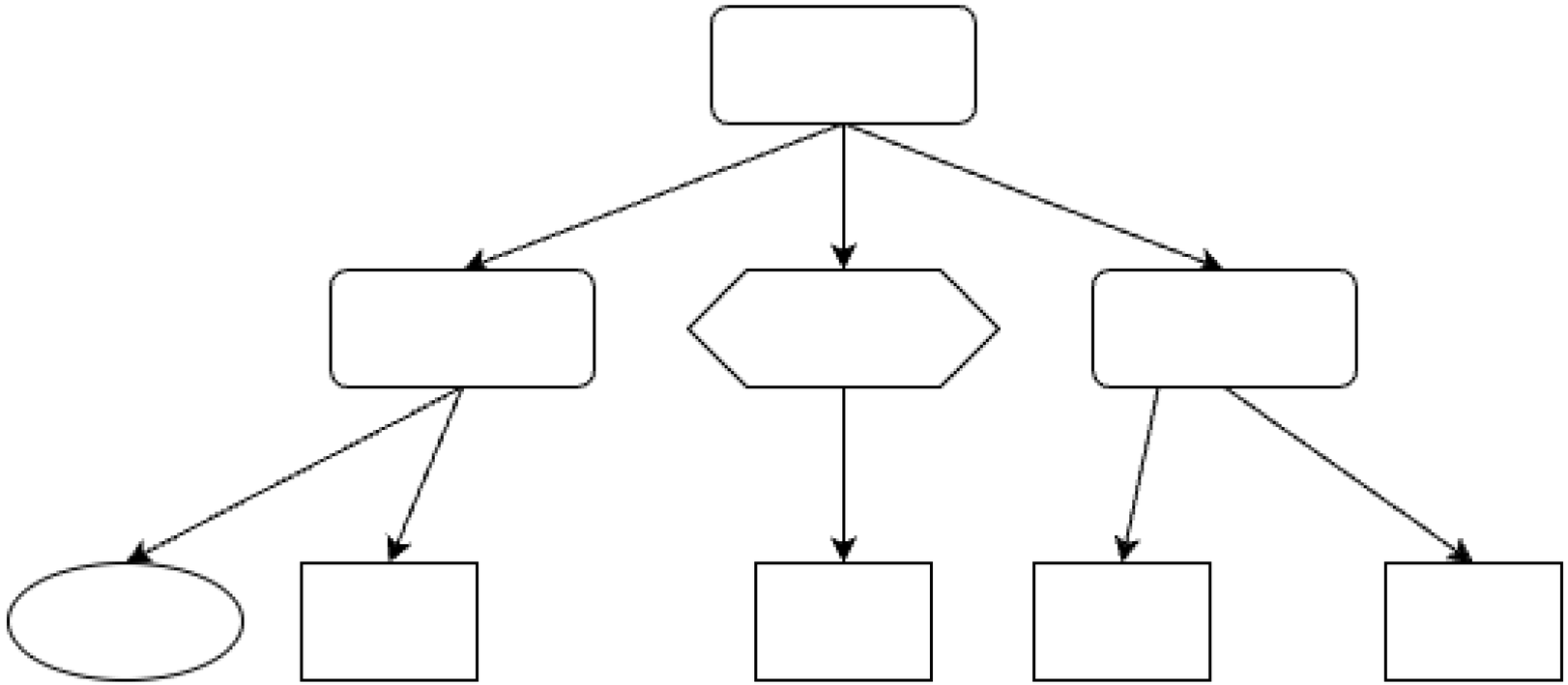
source: behaviortree.dev

Behavior Trees

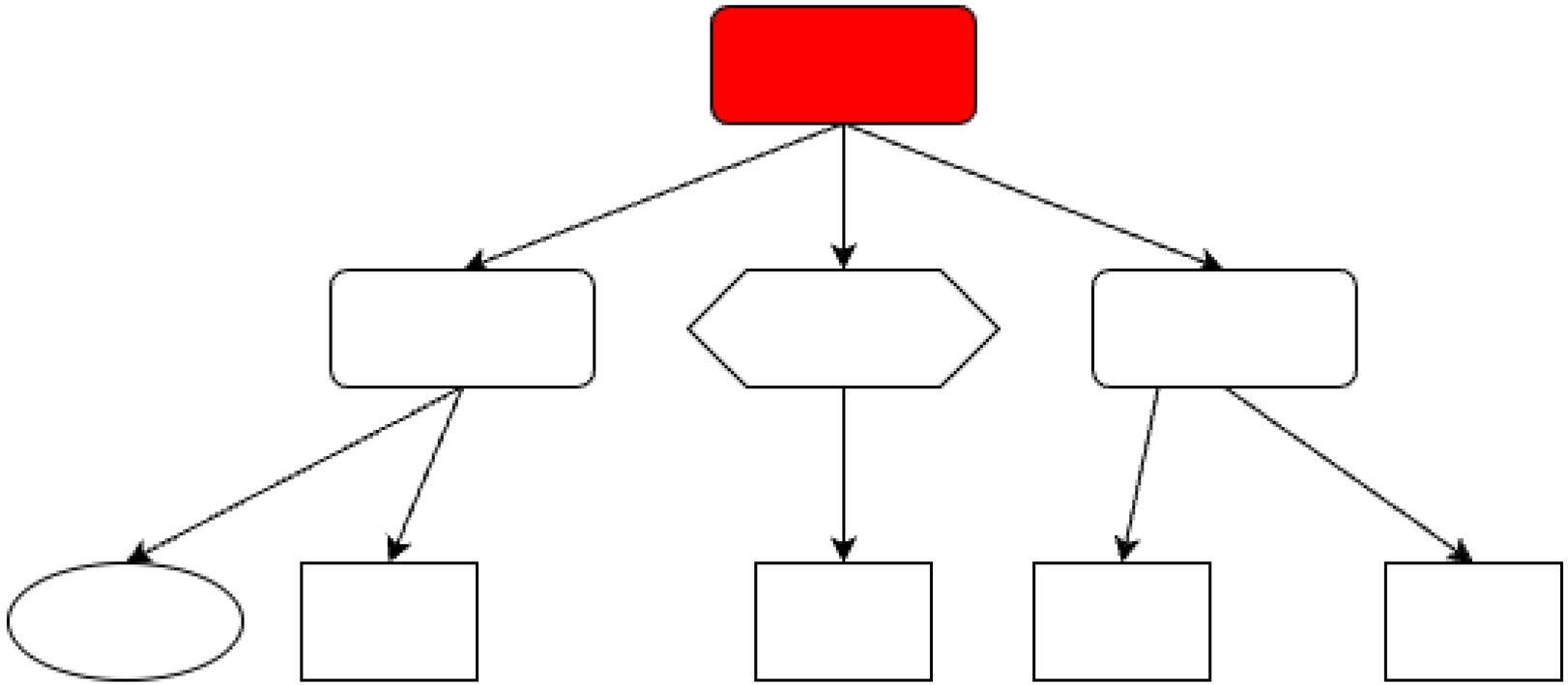


source: behaviortree.dev

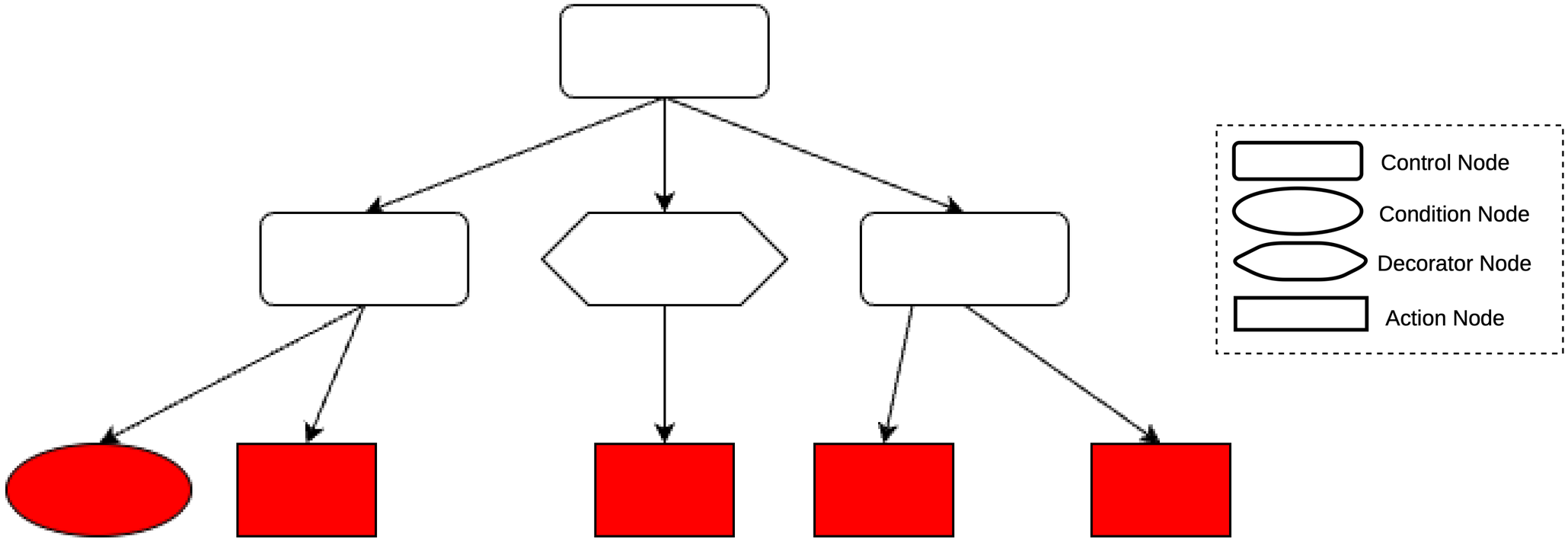
Behavior Trees



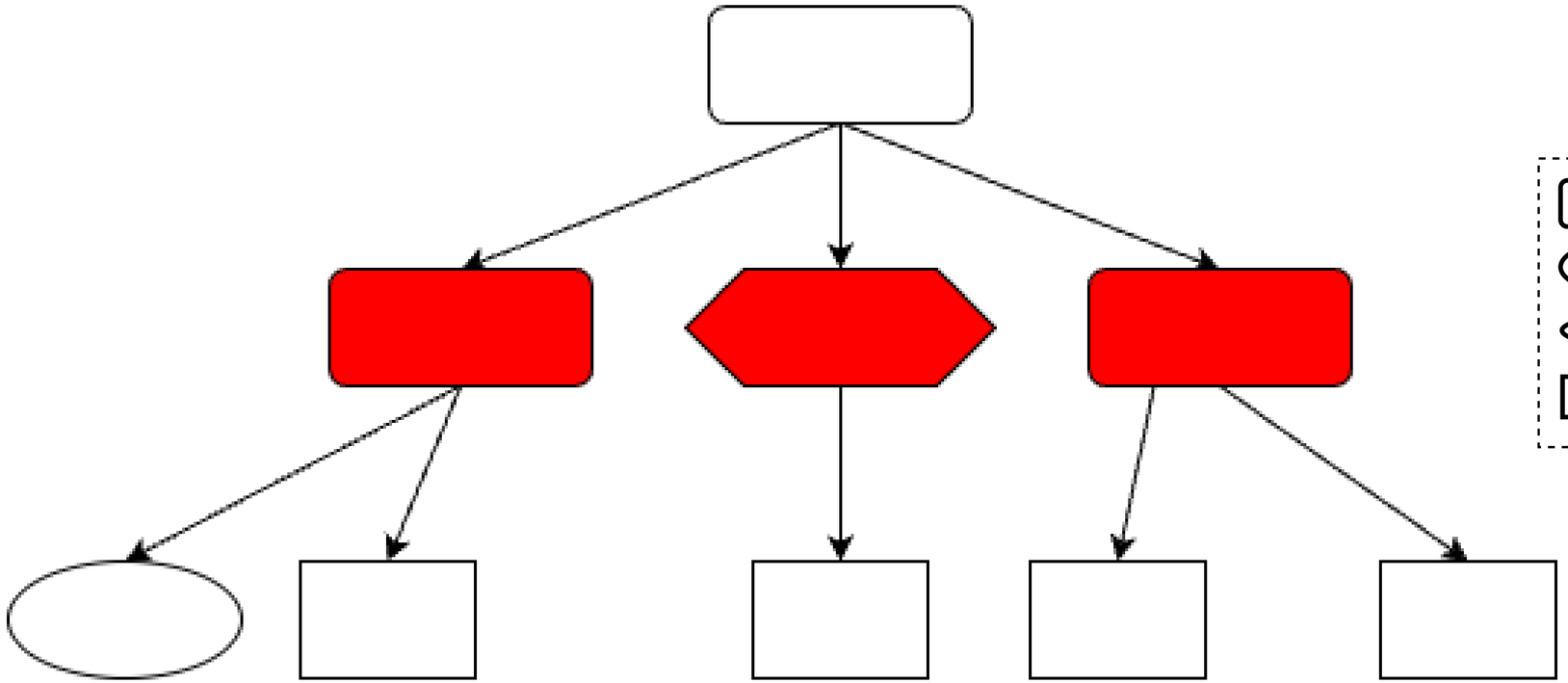
Behavior Trees

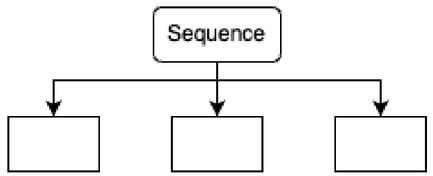


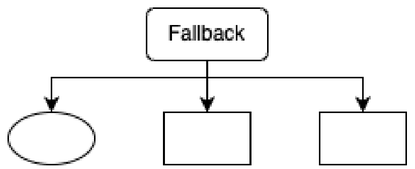
Behavior Trees

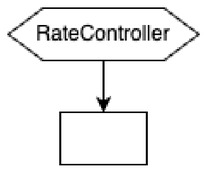


Behavior Trees





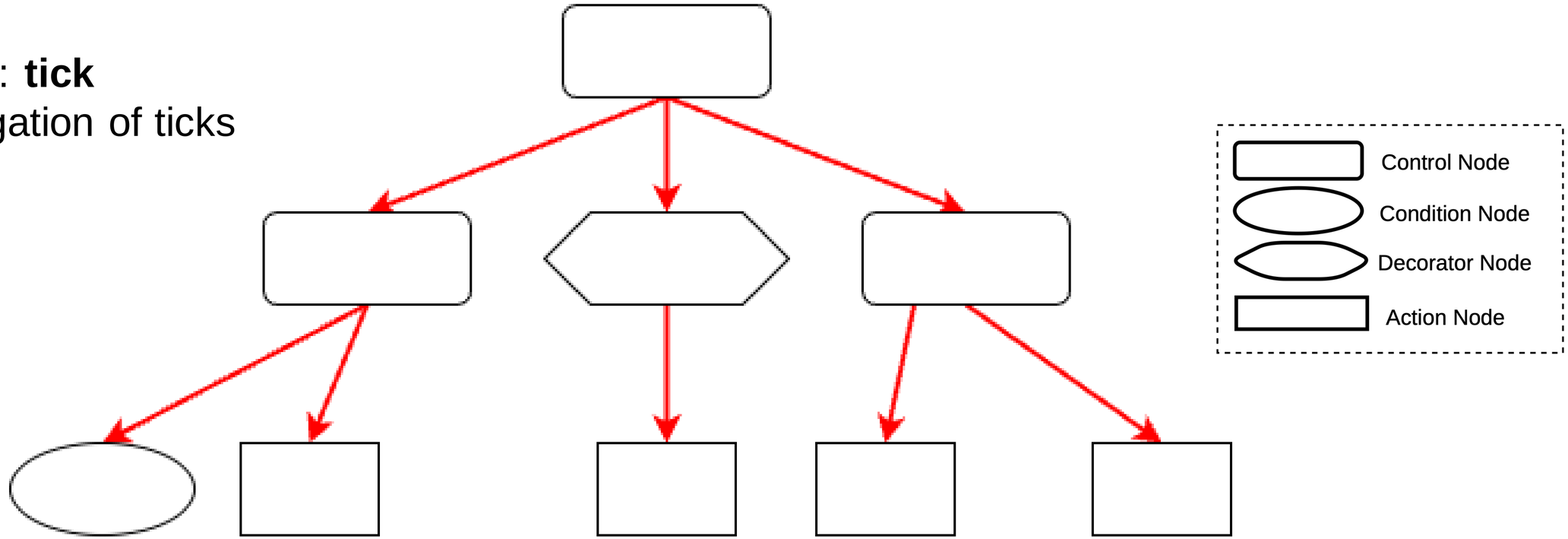




Intelligent Robotics Lab

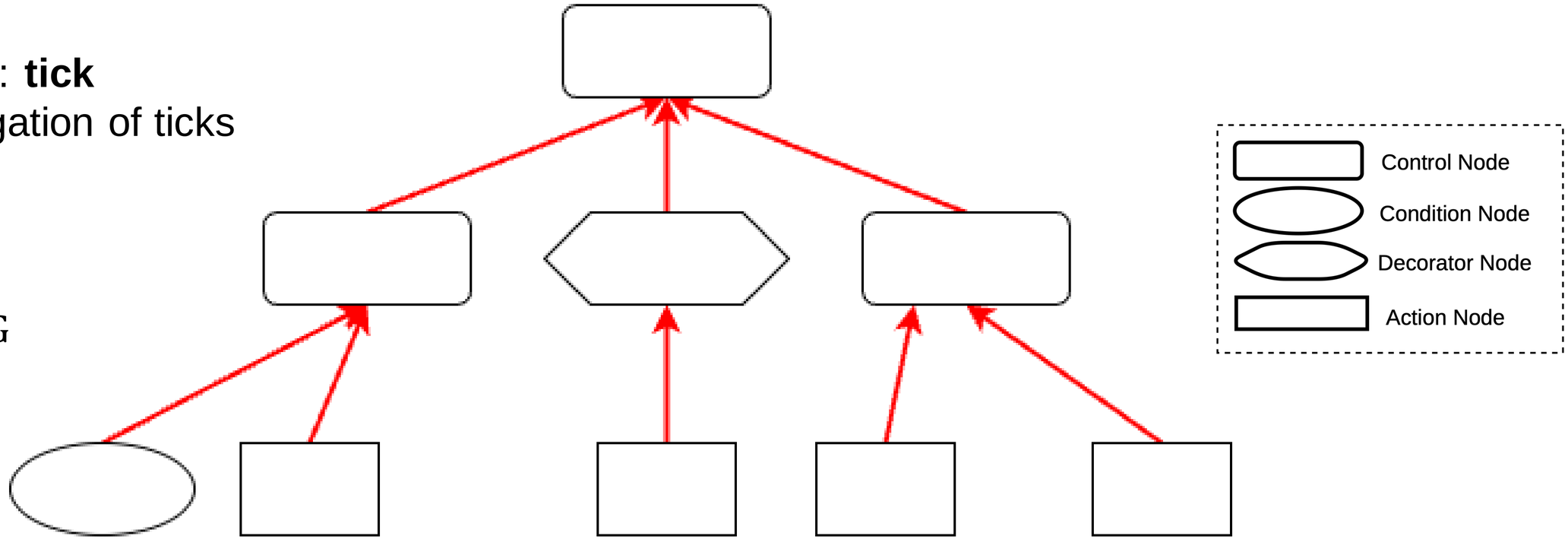
Behavior Trees

- Basic operation: **tick**
- In-depth propagation of ticks



Behavior Trees

- Basic operation: **tick**
- In-depth propagation of ticks
  - SUCCESS
  - FAILURE
  - RUNNING



Behavior Trees

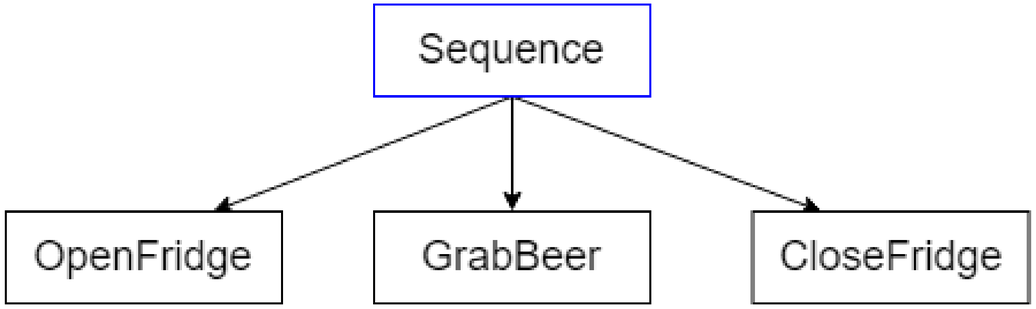
Node type	Children count	Notes
Control	1..N	Ticks a child based on the results of its children
Decorator	1	It may alter the result of the child, tick it multimple times, etc.
Condition	0	Never returns RUNNING
Action	0	Does someting

Behavior Trees

	Value returned by child		
Control Node Type	**FAILURE**	**SUCCESS**	**RUNNING**
`Sequence`	Return FAILURE and restart sequence	Tick next child. Return SUCCESS if no more child	Return RUNNING and tick again
`ReactiveSequence`	Return FAILURE and restart sequence	Tick next child. Return SUCCESS if no more child	Return RUNNING and restart sequence
`SequenceStar`	Return FAILURE and tick again	Tick next child. Return SUCCESS if no more child	Return RUNNING and tick again
`Fallback`	Tick next child. Return FAILURE if no more child	Return SUCCESS	Return RUNNING and tick again
`ReactiveFallback`	Tick next child. Return FAILURE if no more child	Return SUCCESS	Return RUNNING and restart sequence
`InverterNode`	Return SUCCESS	Return FAILURE	Return RUNNING
`ForceSuccessNode`	Return SUCCESS	Return SUCCESS	Return RUNNING
`ForceFailureNode`	Return FAILURE	Return FAILURE	Return RUNNING
`RepeatNode` (N)	Return FAILURE	Return RUNNING N timesbefore returning SUCCESS	Return RUNNING
`RetryNode` (N)	Return RUNNING N times before returning FAILURE	Return SUCCESS	Return RUNNING

Intelligent Robotics Lab

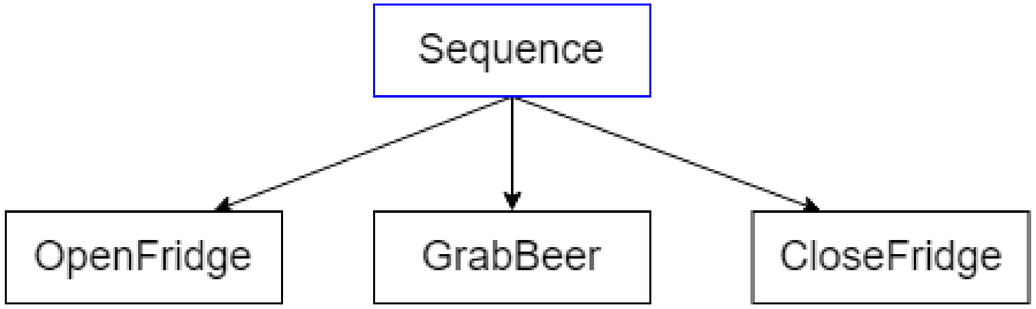
Behavior Trees
Sequences



source: behaviortree.dev

- If the child return SUCCESS, ticks the next one
- If the child return FAILURE, no more children are ticked and the sequence returns FAILURE
- If all children return SUCCESS, the sequence returns SUCCESS

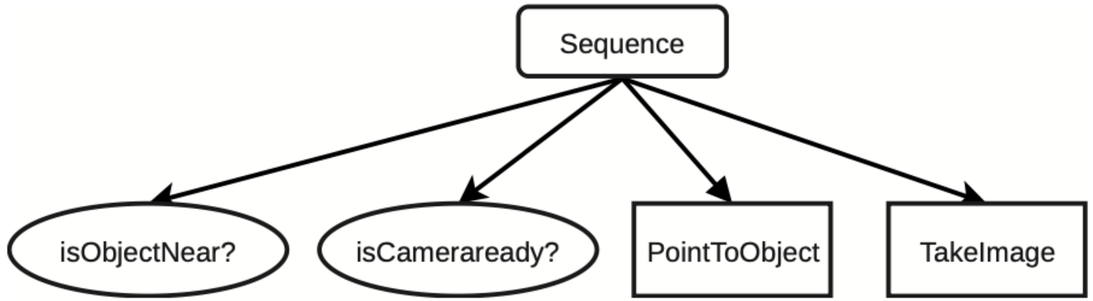
Behavior Trees
Sequences

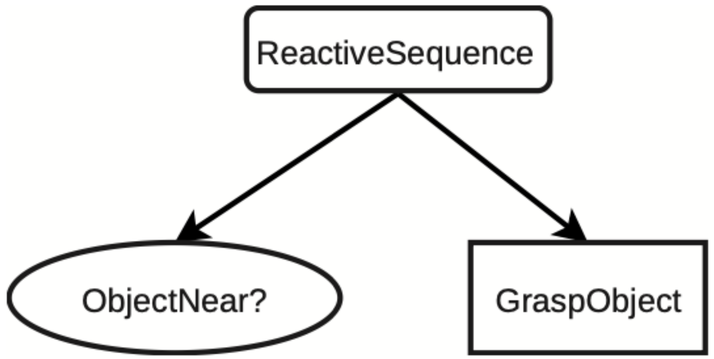


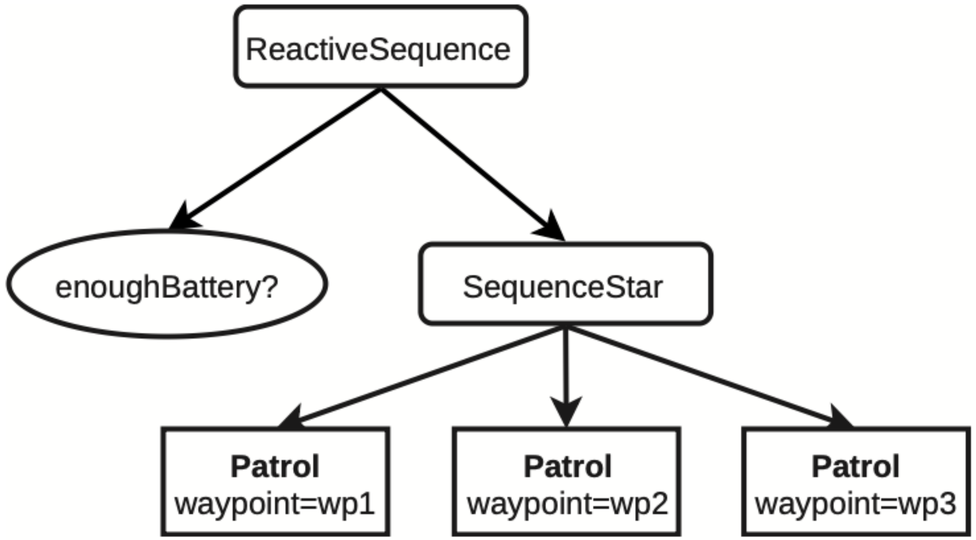
source: behaviortree.dev

What happens if GrabBeer fails?

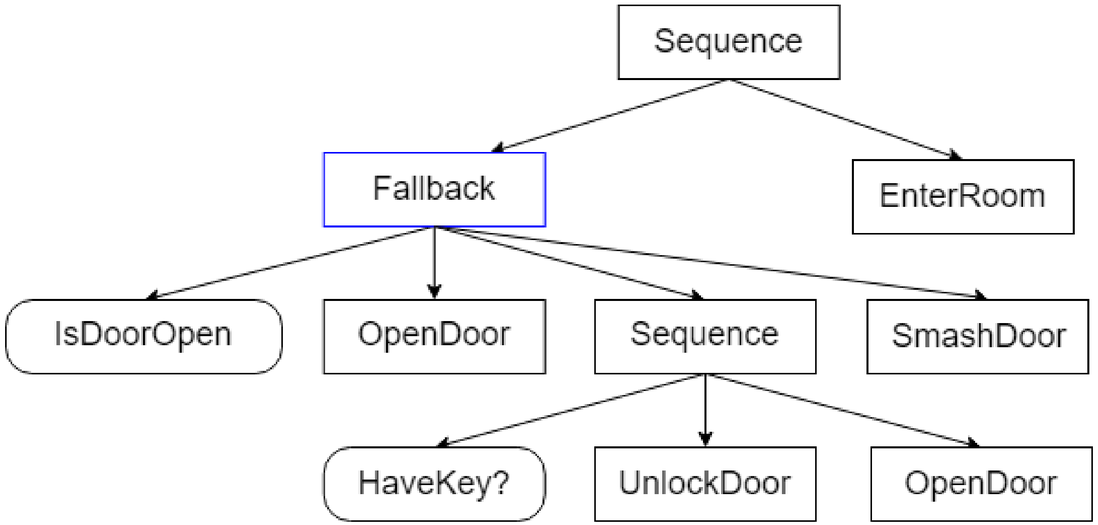
Behavior Trees
Sequences







Behavior Trees
Fallback

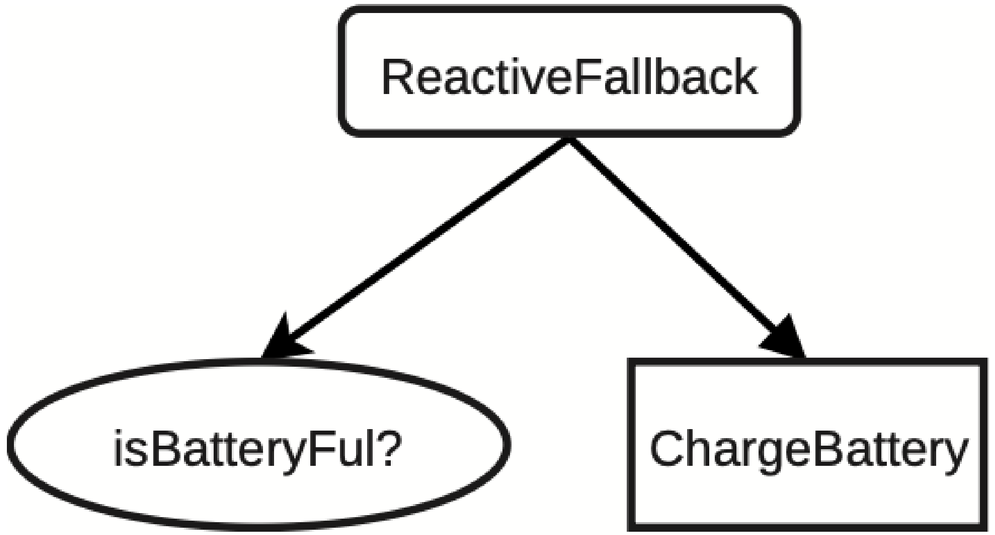


source: behaviortree.dev

- If the door is closed, try to open it, if imposible, try to open it with a key, and if it is not posible as well, smash it.

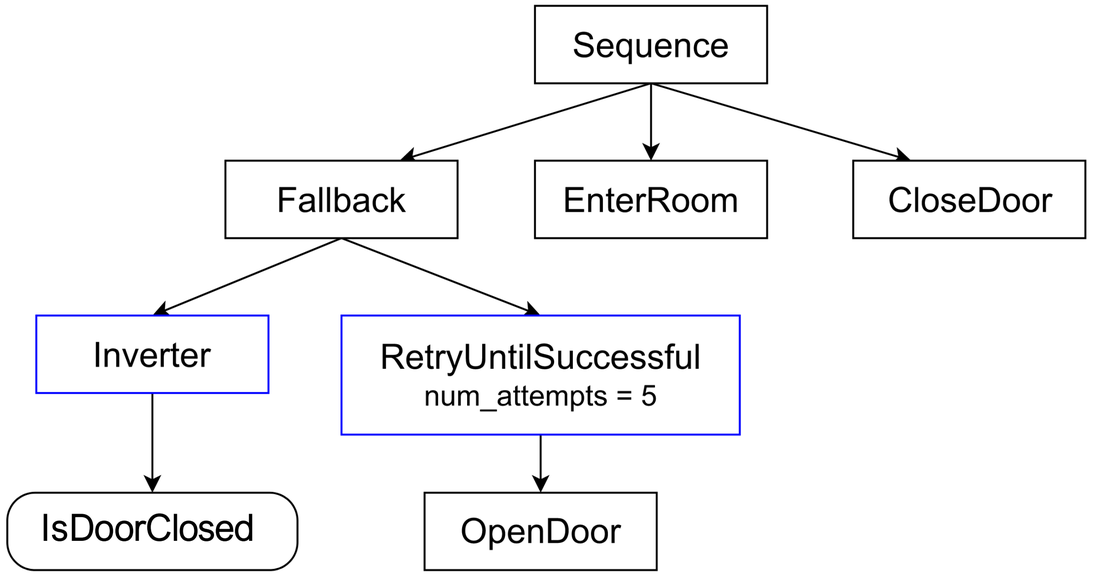
- Once the door is open, enter the room.

Behavior Trees
Reactive Fallback



- Action ChargeBattery always returns RUNNING, so the condition is constantly checked
- Once the battery is full, the condition node returns SUCCESS and the reactive behavior stops

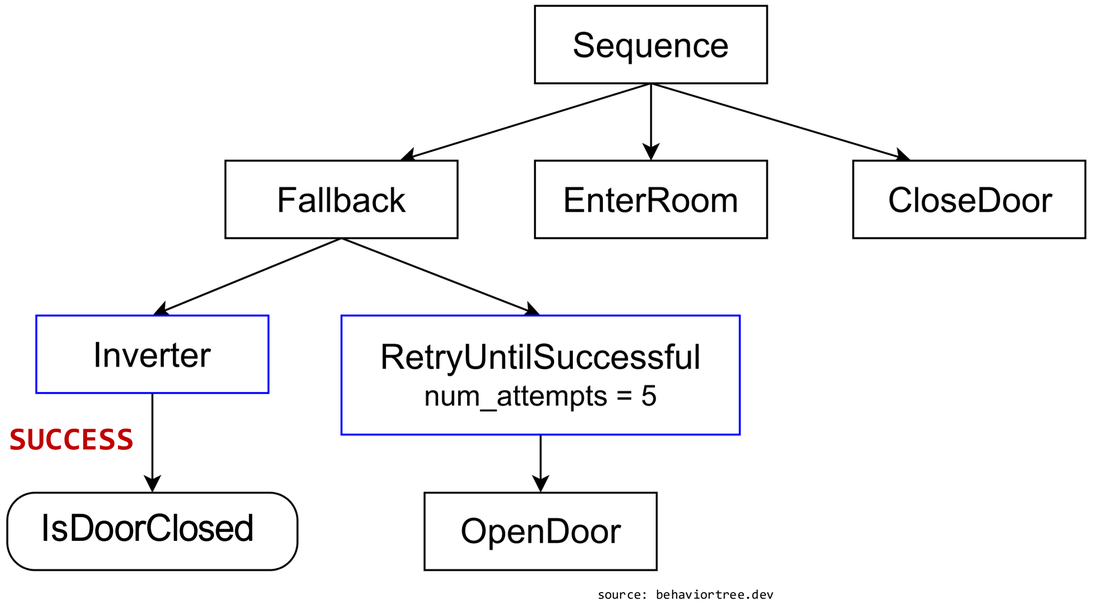
Behavior Trees
Decorators



source: behaviortree.devcccccc

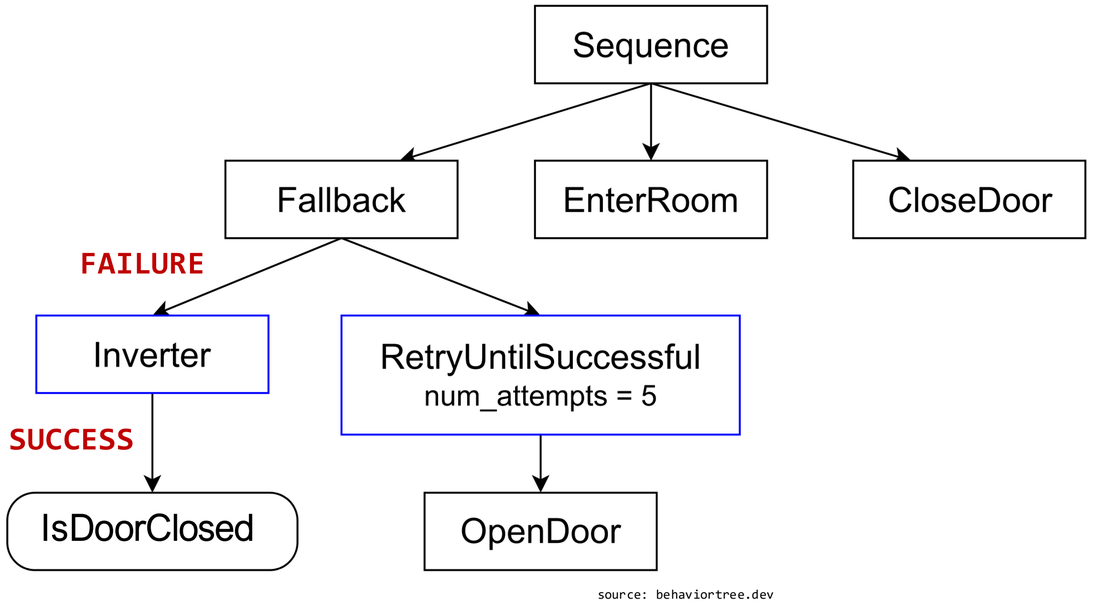
- If the door is closed, then try to open it. Try up to 5 times, otherwise give up
- If the door cannot be opened --> the sequence fails
- Otherwise --> enter the room and close the door

Behavior Trees
Decorators



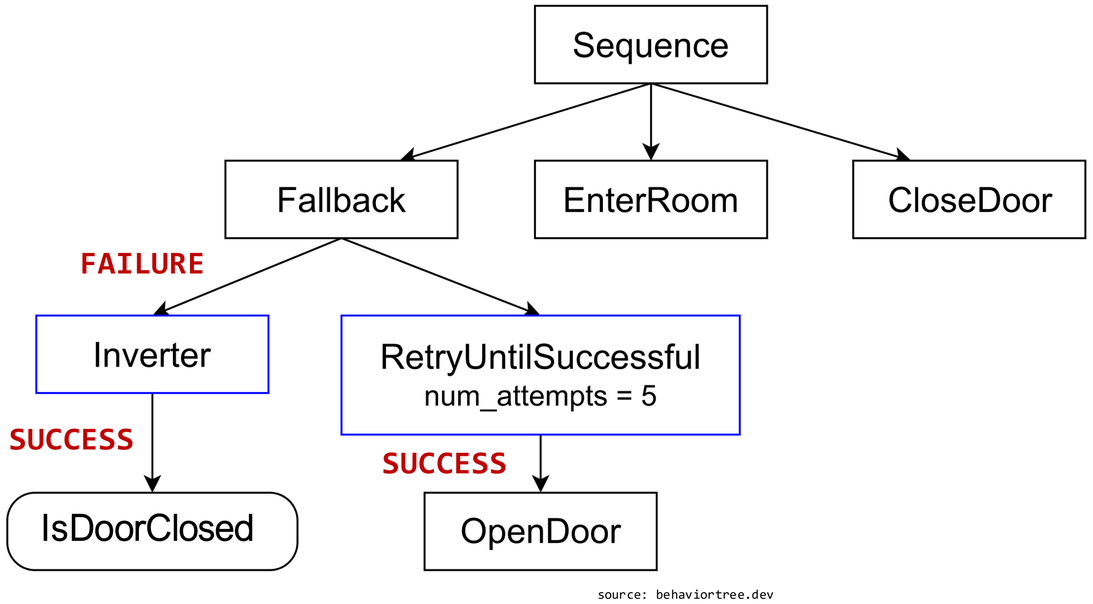
- If the door is closed, then try to open it. Try up to 5 times, otherwise give up
- If the door cannot be opened --> the sequence fails
- Otherwise --> enter the room and close the door

Behavior Trees
Decorators



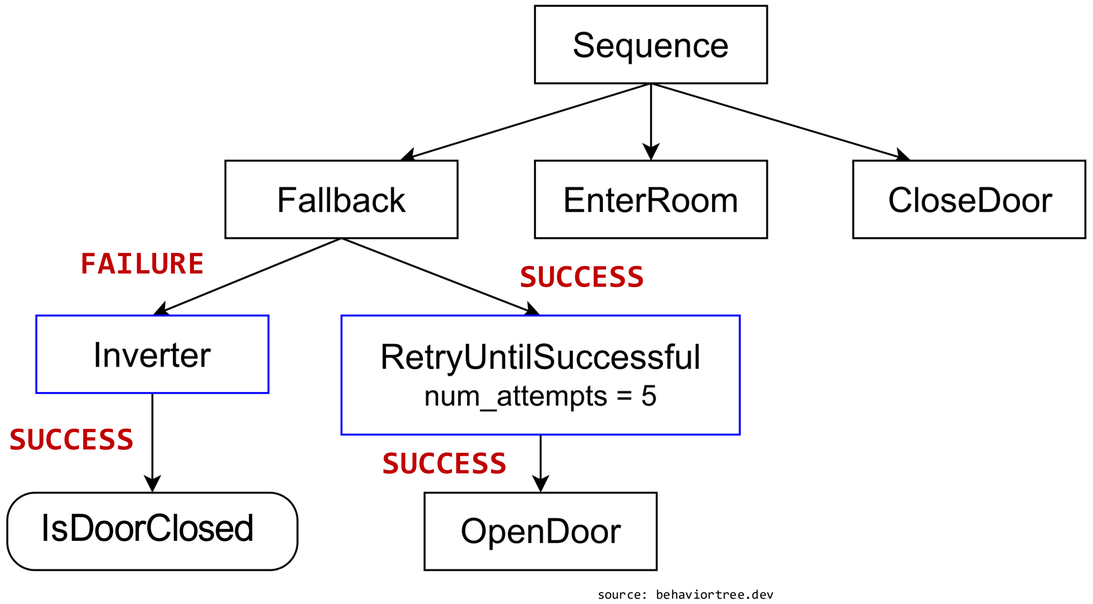
- If the door is closed, then try to open it. Try up to 5 times, otherwise give up
- If the door cannot be opened --> the sequence fails
- Otherwise --> enter the room and close the door

Behavior Trees
Decorators



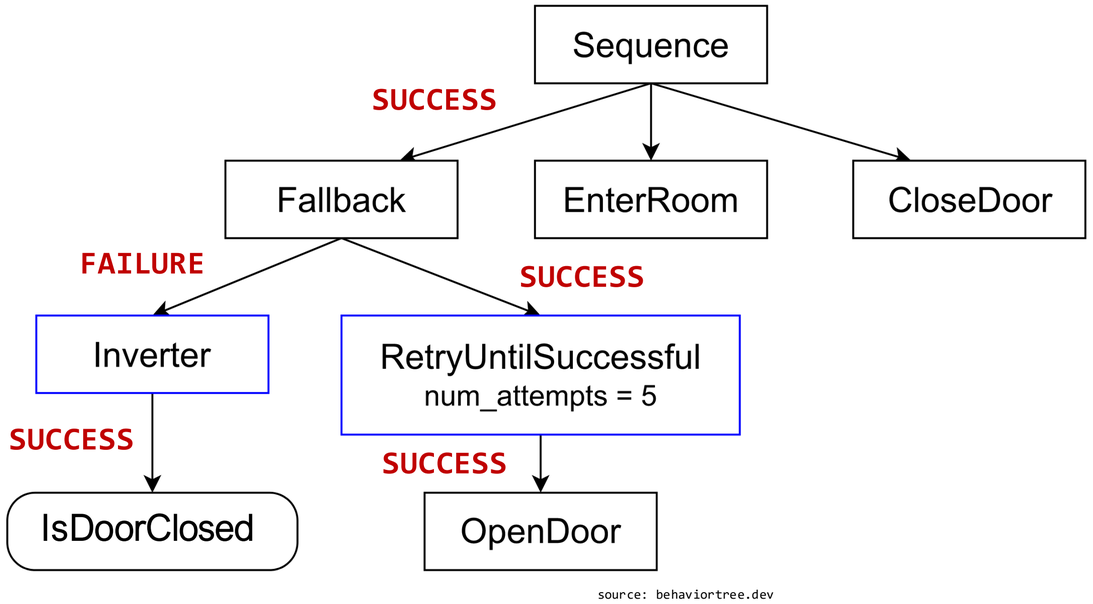
- If the door is closed, then try to open it. Try up to 5 times, otherwise give up
- If the door cannot be opened --> the sequence fails
- Otherwise --> enter the room and close the door

Behavior Trees
Decorators



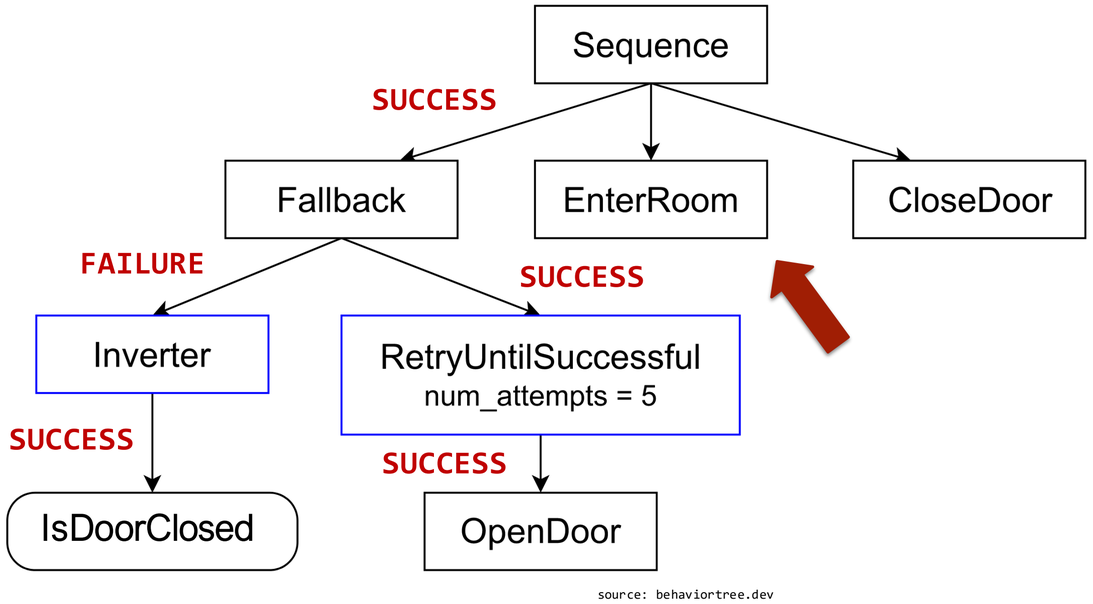
- If the door is closed, then try to open it. Try up to 5 times, otherwise give up
- If the door cannot be opened --> the sequence fails
- Otherwise --> enter the room and close the door

Behavior Trees
Decorators



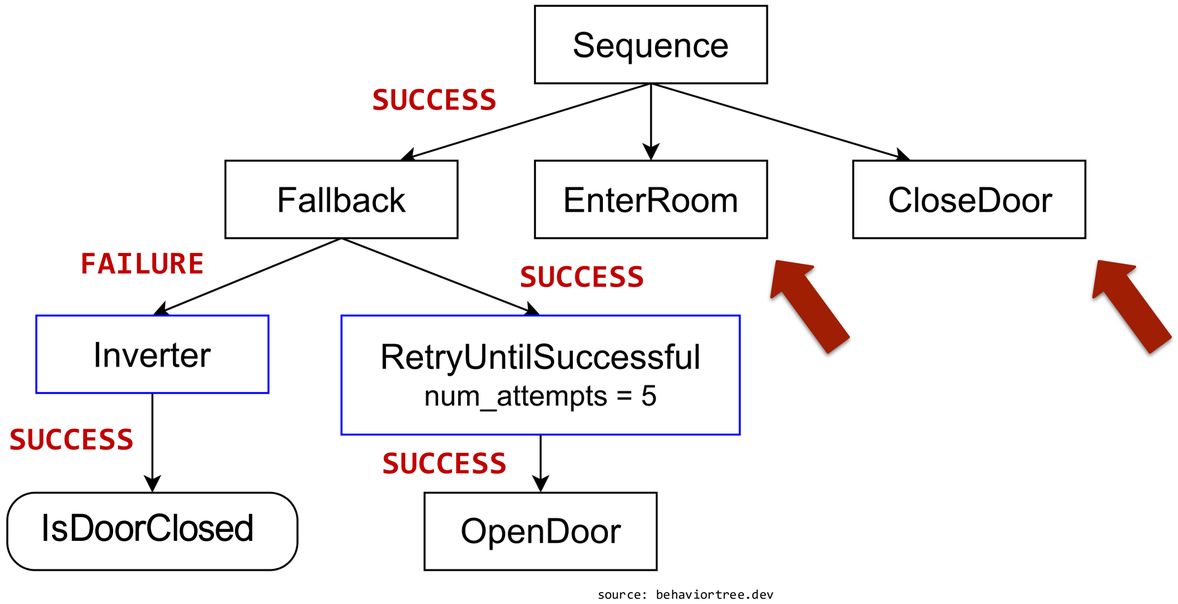
- If the door is closed, then try to open it. Try up to 5 times, otherwise give up
- If the door cannot be opened --> the sequence fails
- Otherwise --> enter the room and close the door

Behavior Trees
Decorators



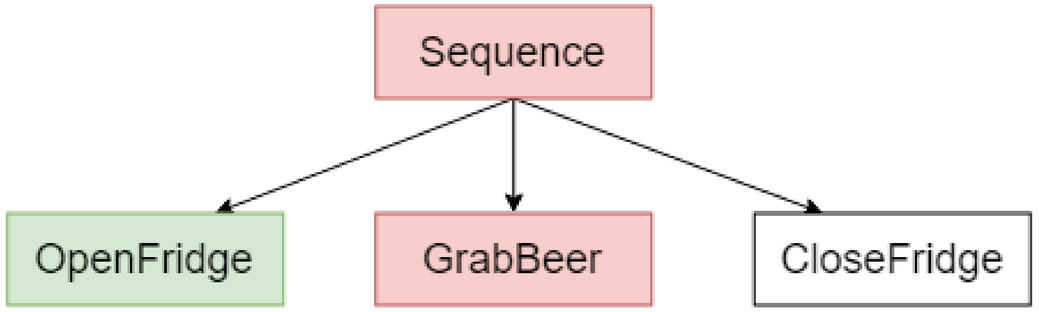
- If the door is closed, then try to open it. Try up to 5 times, otherwise give up
- If the door cannot be opened --> the sequence fails
- Otherwise --> enter the room and close the door

Behavior Trees
Decorators



- If the door is closed, then try to open it. Try up to 5 times, otherwise give up
- If the door cannot be opened --> the sequence fails
- Otherwise --> enter the room and close the door

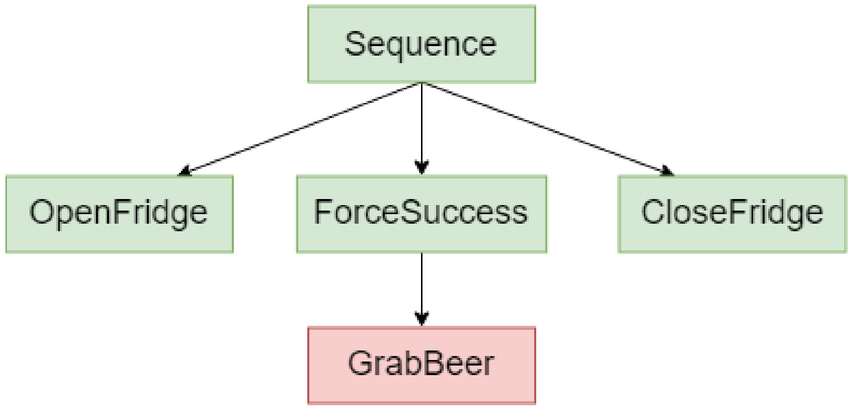
Behavior Trees
Example revisited

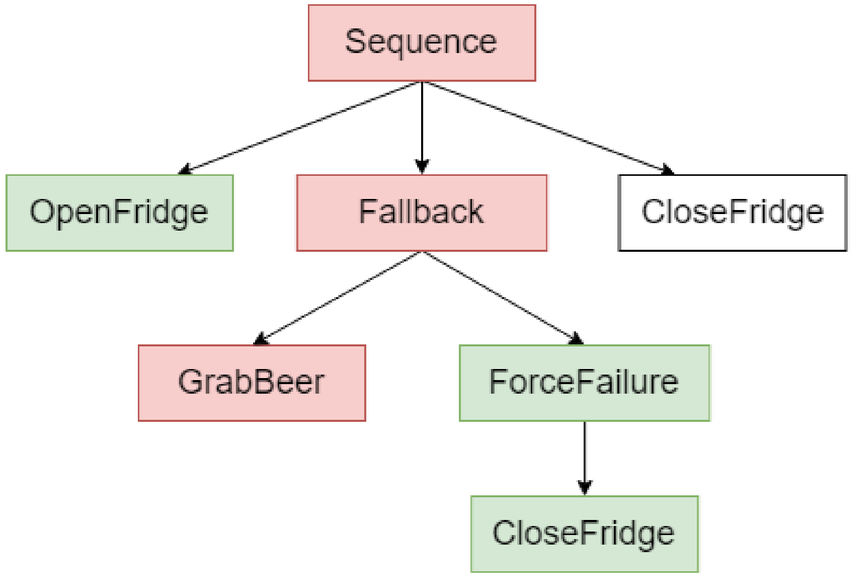


source: behaviortree.dev

What happens if GrabBeer fails?

Behavior Trees
Example revisited

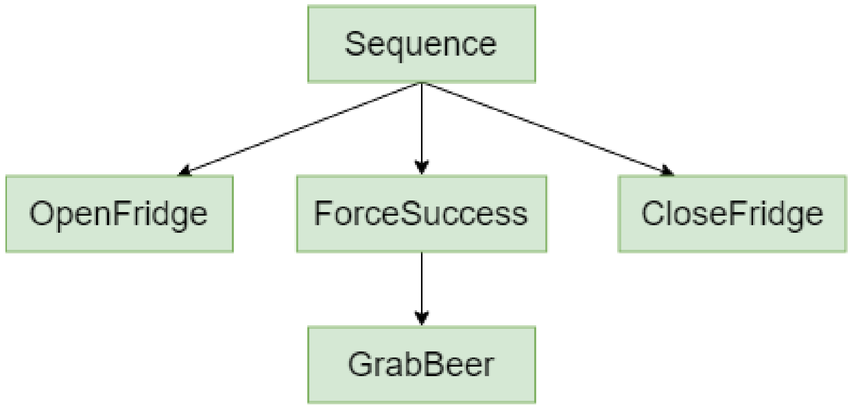


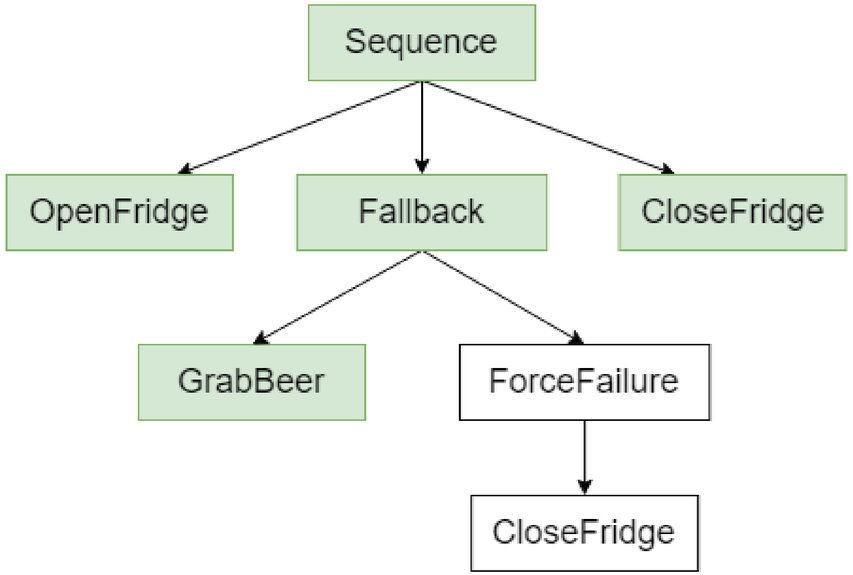


source: behaviortree.dev

- Left tree --> does not matter if the beer could not be grabbed
- Right tree --> the fridge is always closed

Behavior Trees
Example revisited

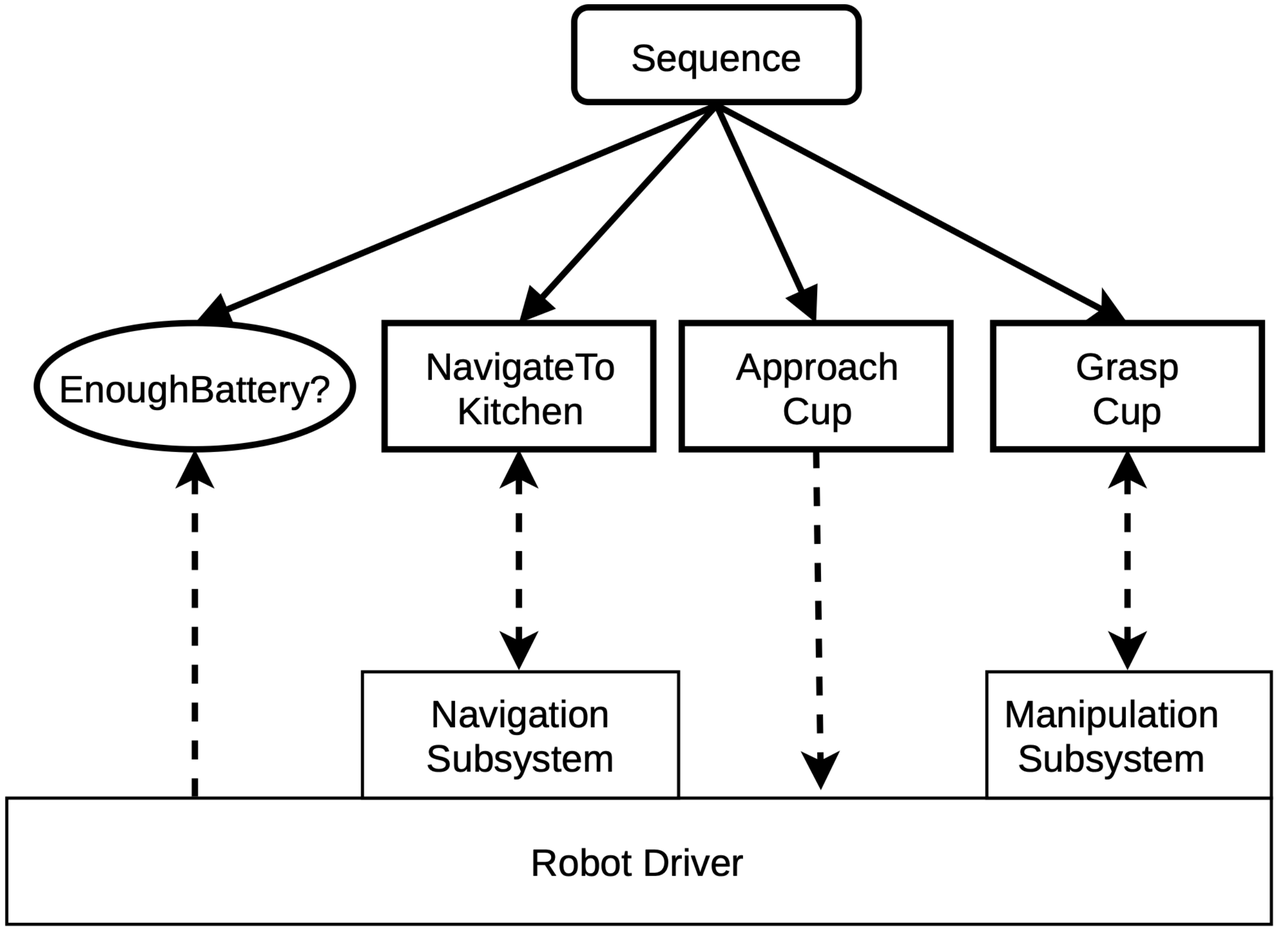




source: behaviortree.dev

- Left tree --> does not matter if the beer could not be grabbed
- Right tree --> the fridge is always closed

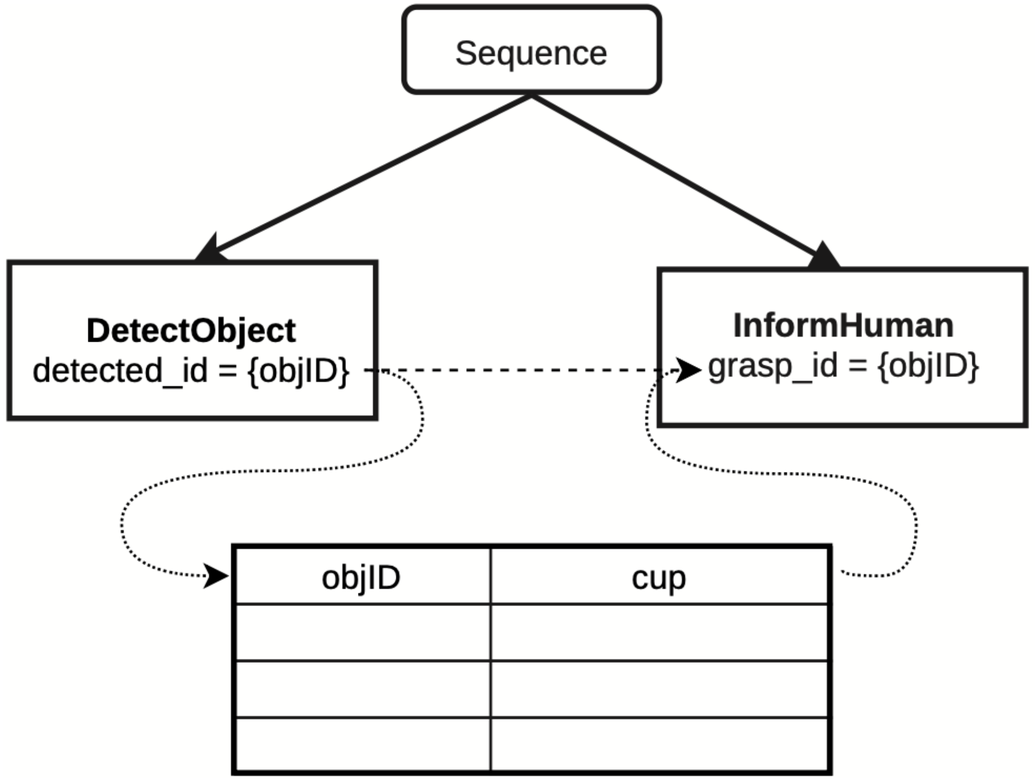
Behavior Trees
Integration with ROS 2



Behavior Trees
Blackboard

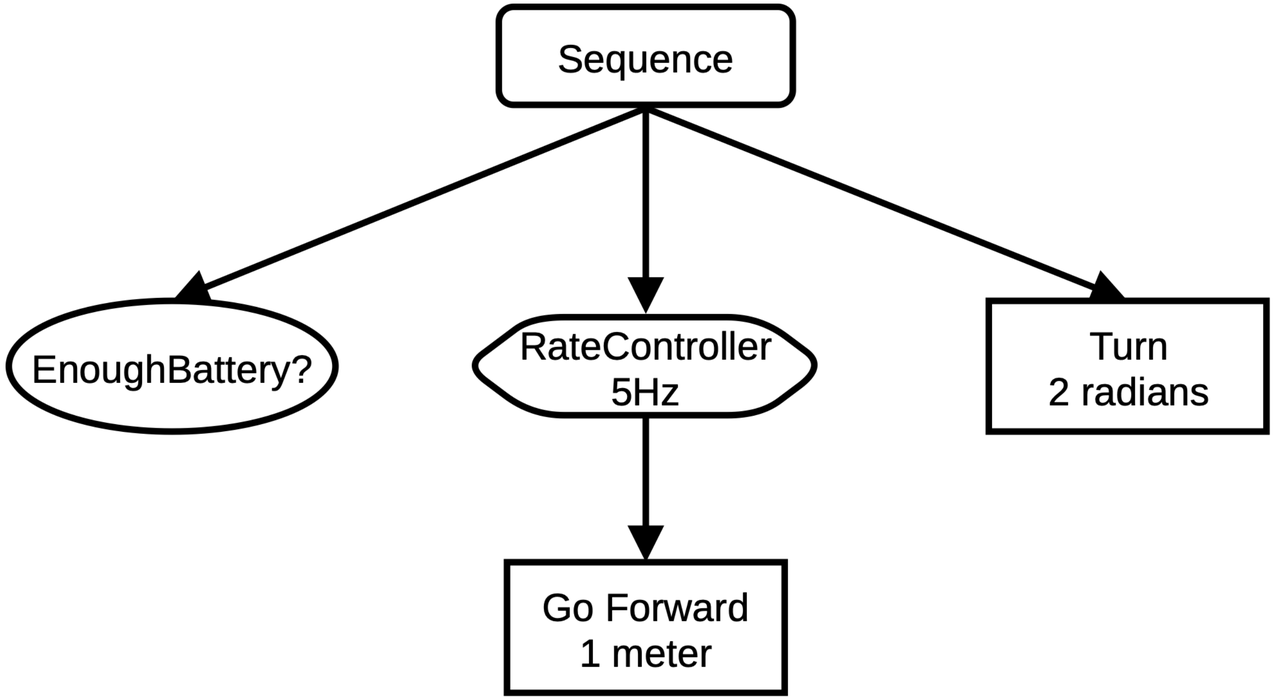
- Key/value storage
- All nodes in a tree can access
- Enable input/output ports connected amongst nodes

Behavior Trees
Blackboard



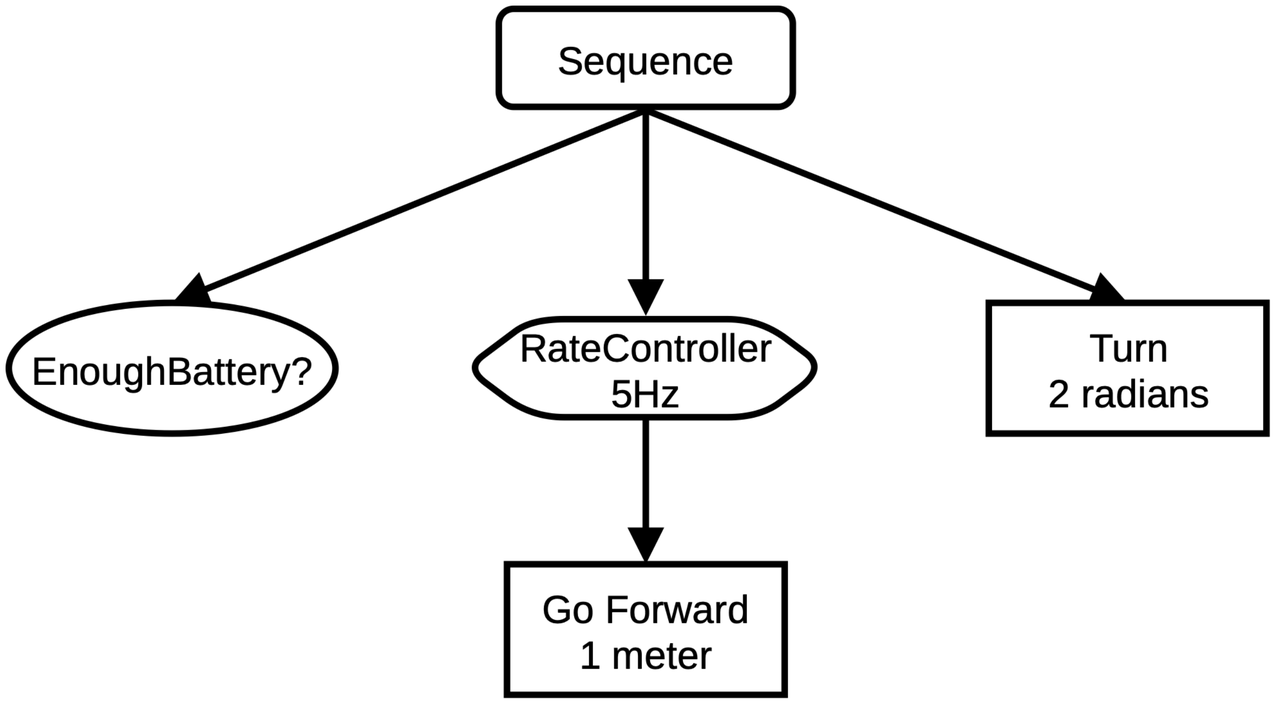
Behavior Trees
Compact XML specification

```
<BehaviorTree ID="BehaviorTree">
    <Sequence>
        <EnoughBattery/>
        <RateController Rate="5Hz">
            <GoForward distance="1.0"/>
        </RateController>
        <Turn angle="2.0"/>
    </Sequence>
</BehaviorTree>
```

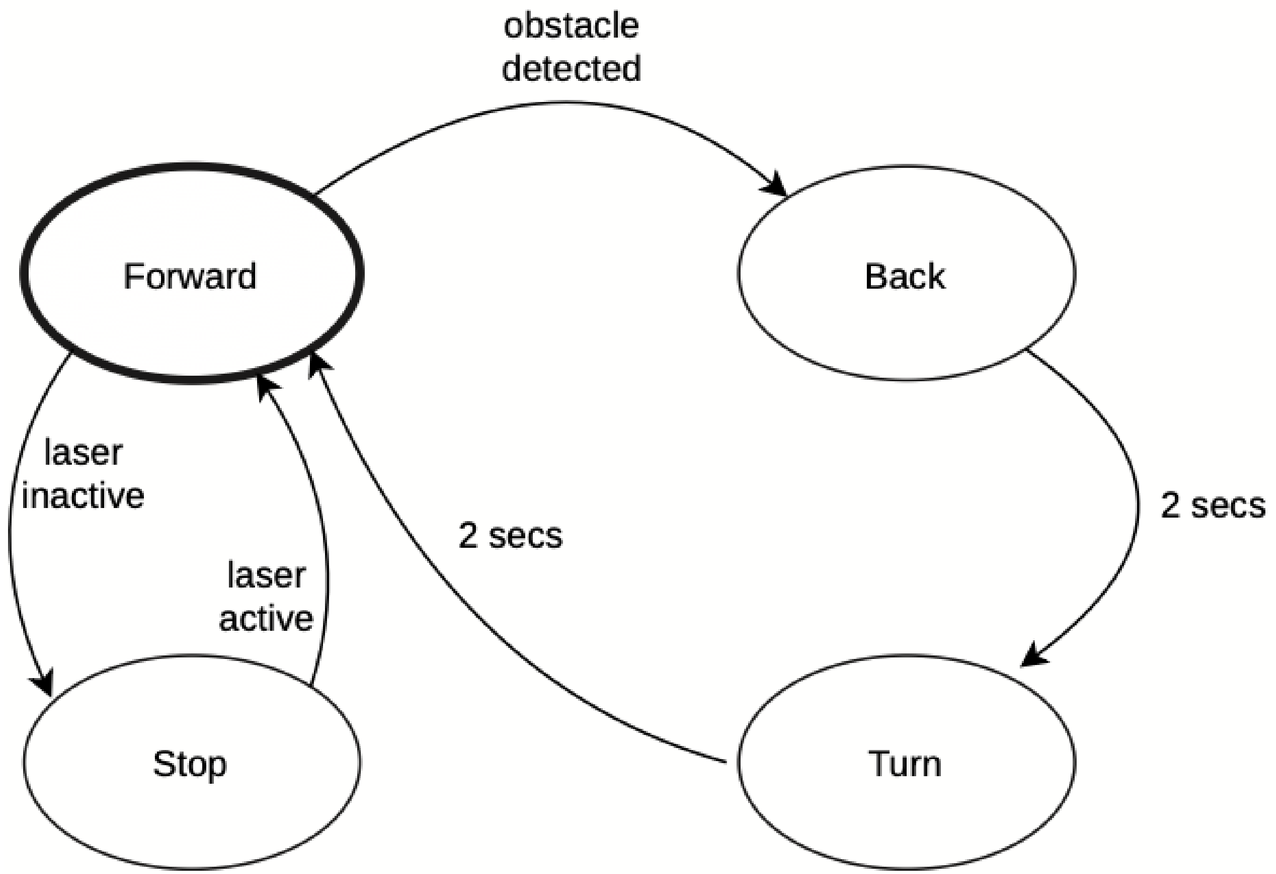


Behavior Trees
Extended XML specification

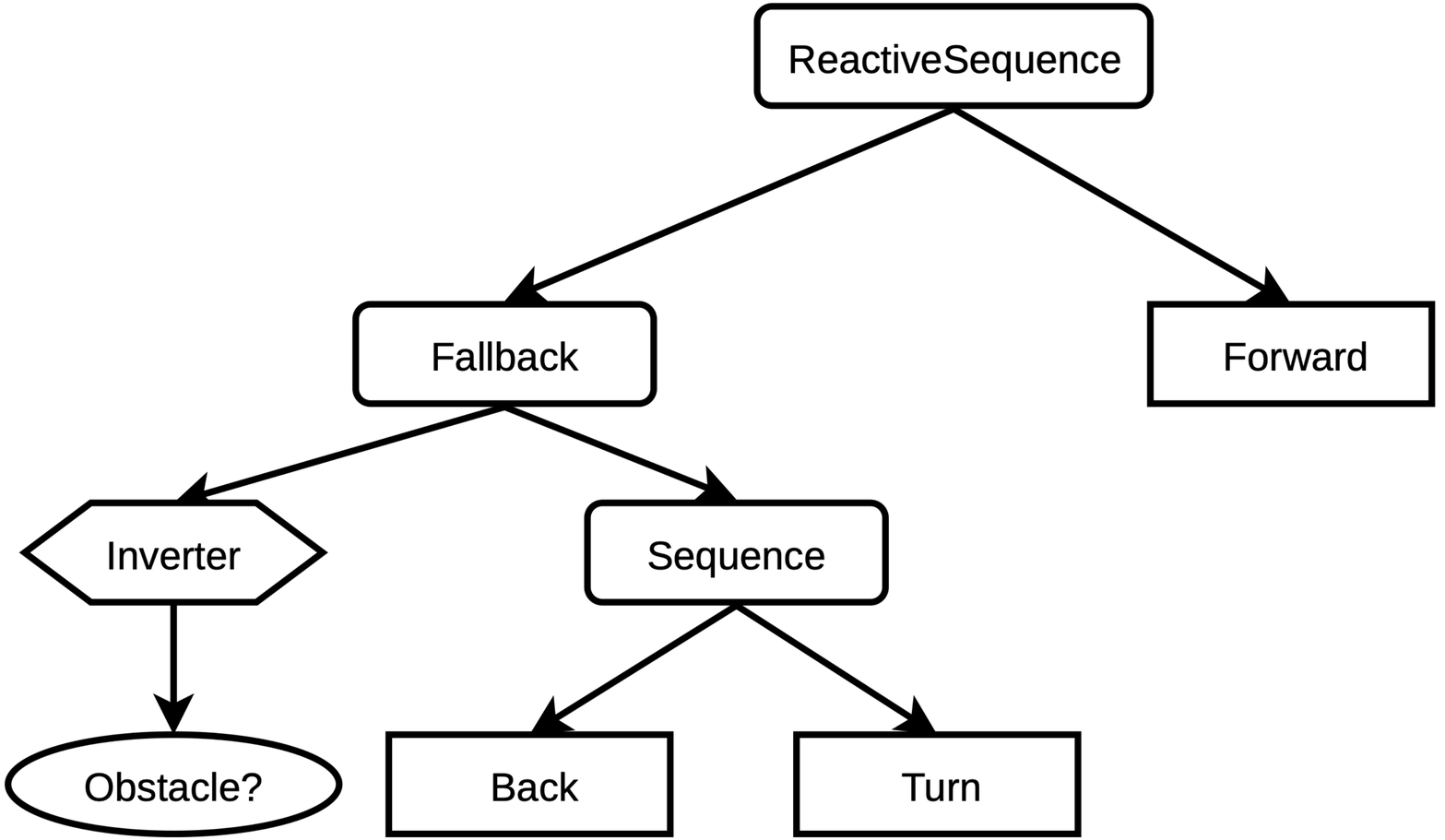
```xml
<?xml version="1.0"?>
<root main_tree_to_execute="BehaviorTree">

    <BehaviorTree ID="BehaviorTree">
        <Sequence>
            <Condition ID="EnoughBattery"/>
            <Decorator ID="RateController" Rate="5Hz">
                <Action ID="GoForward" distance="1.0"/>
            </Decorator>
            <Action ID="Turn" angle="2.o"/>
        </Sequence>
    </BehaviorTree>

    <TreeNodesModel>
        <Condition ID="EnoughBattery"/>
        <Action ID="GoForward">
            <input_port name="distance"/>
        </Action>
        <Decorator ID="RateController">
            <input_port name="Rate"/>
        </Decorator>
        <Action ID="Turn">
            <input_port name="angle"/>
        </Action>
    </TreeNodesModel>
</root>
```



Behavior Trees
Bump and go with FSM



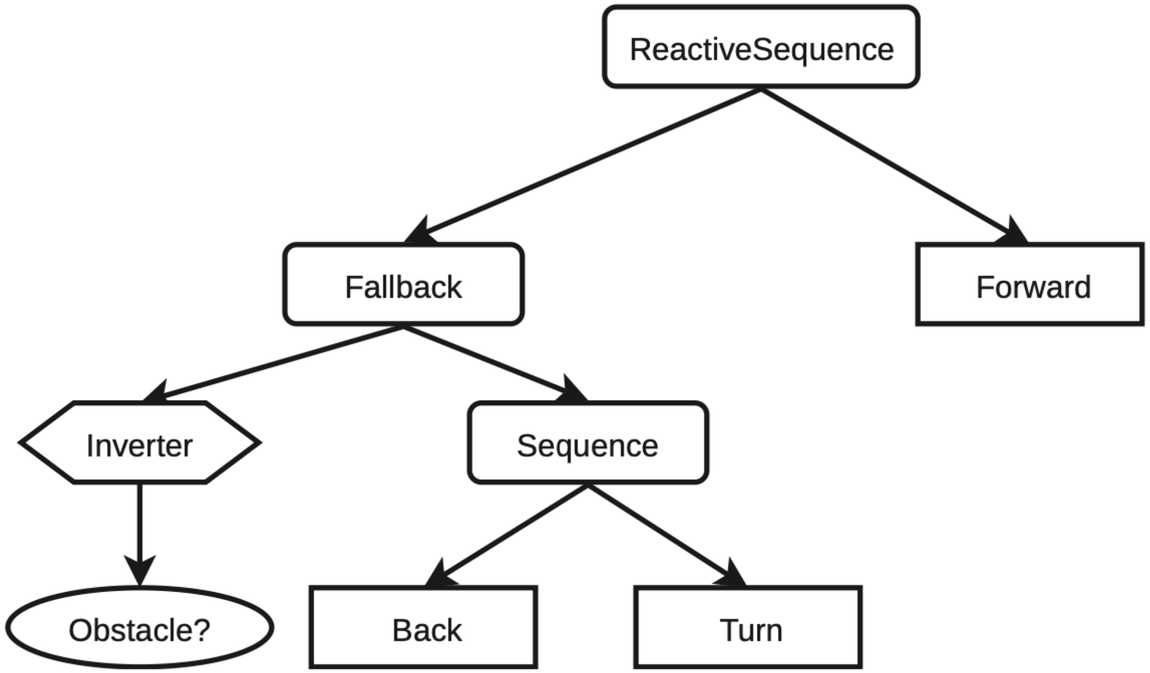
Behavior Trees
Bump and go with BT

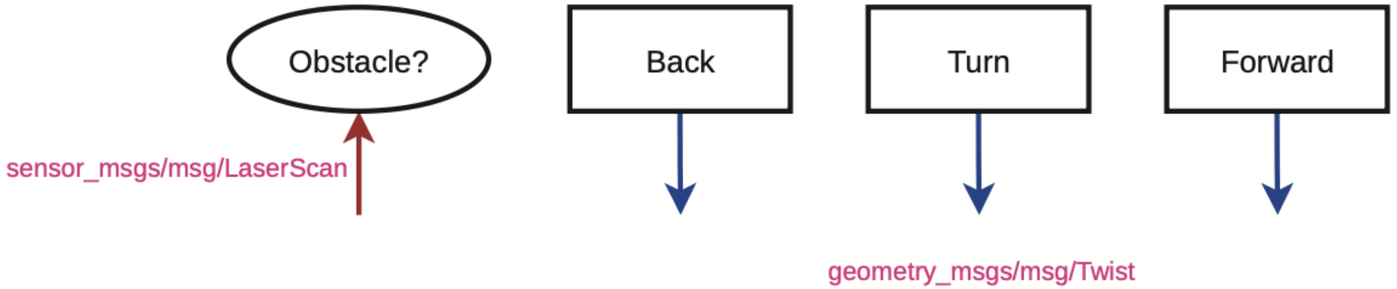


Bump&Go with Behavior Trees

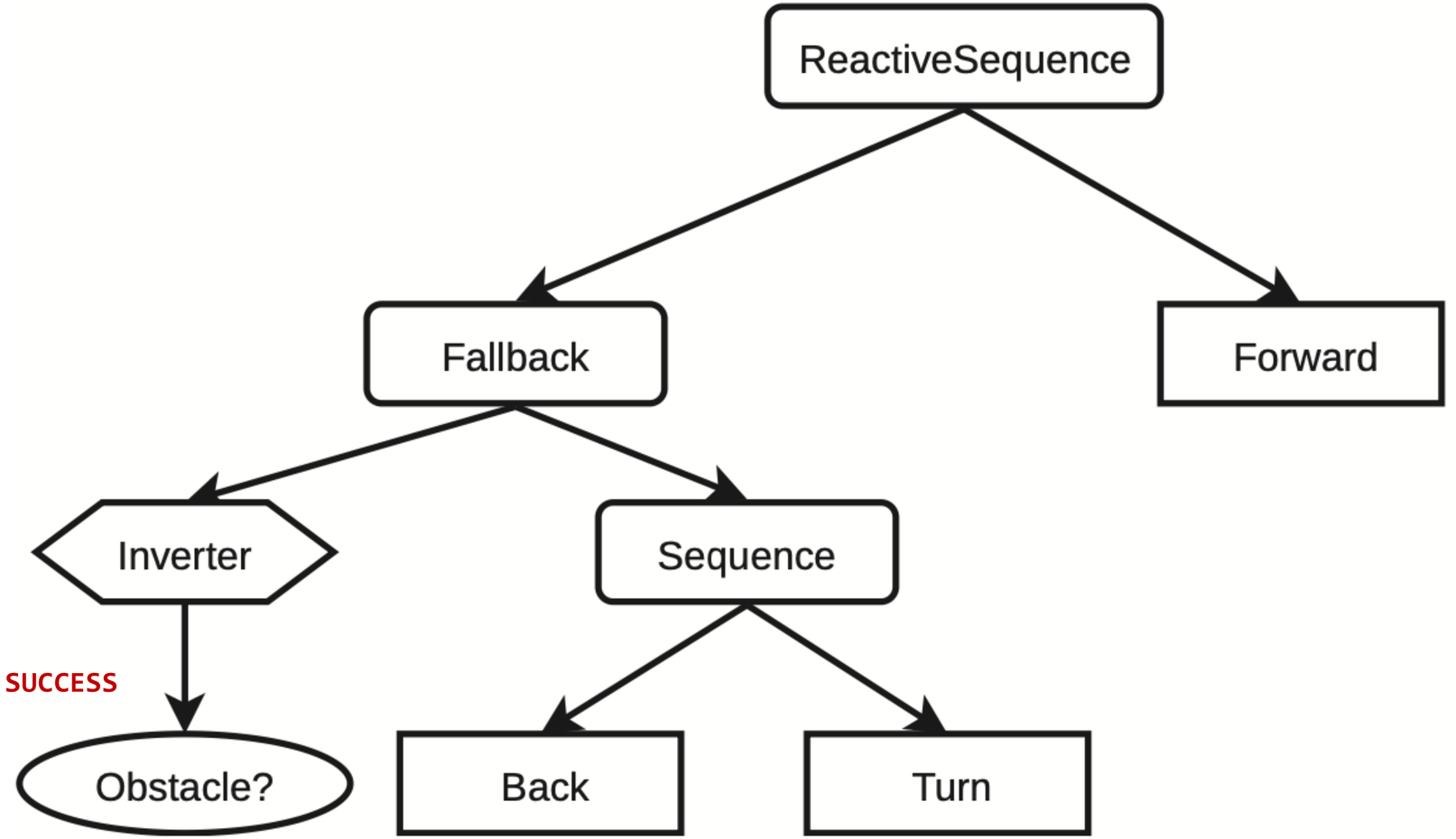
Bump&Go with Behavior Trees

- **Goal**: a Bump&Go behavior
- New concepts:
  - **Implement Behavior Tree Nodes with ROS2 comms**
  - **Behavior Tree execution**

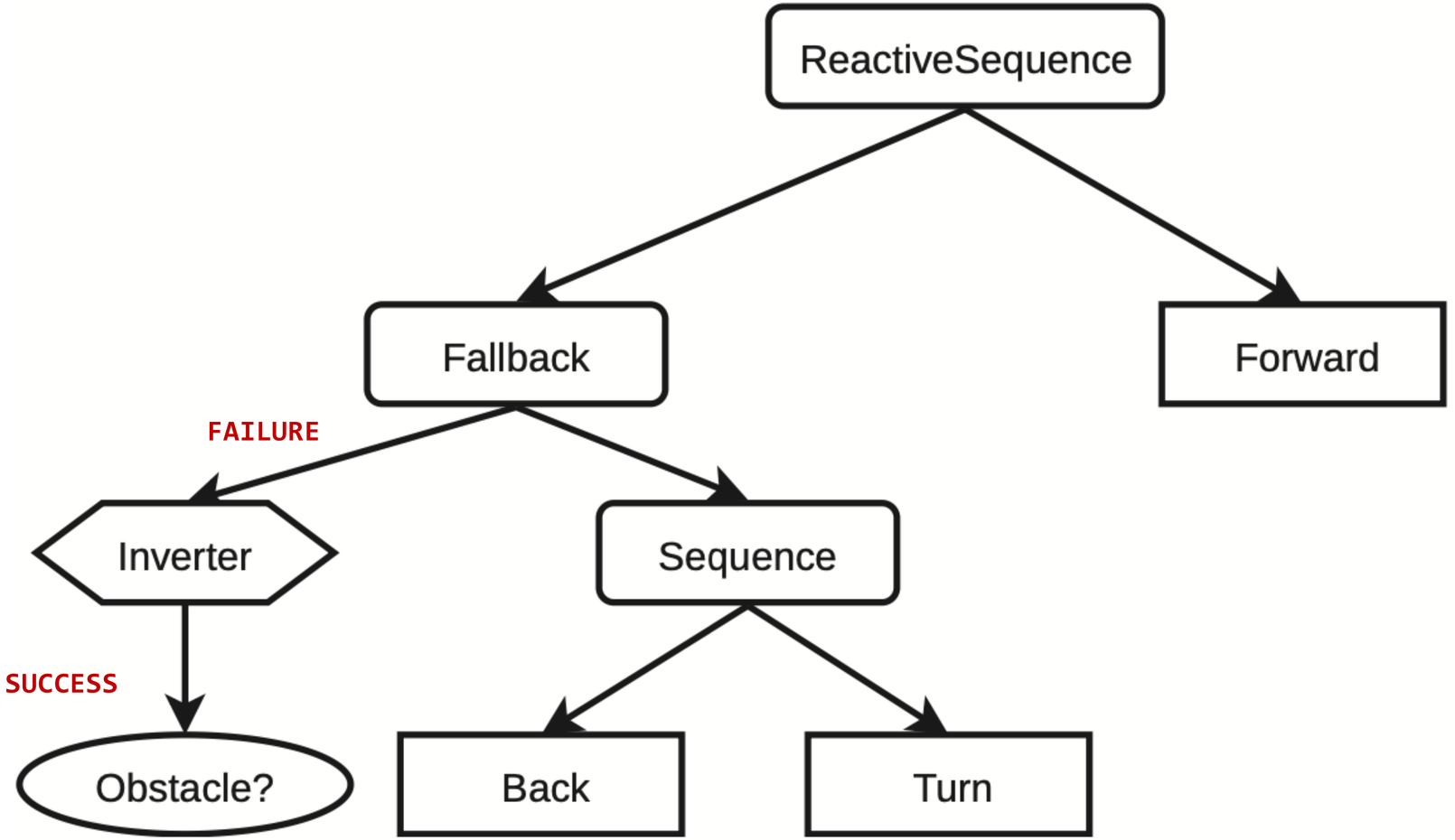




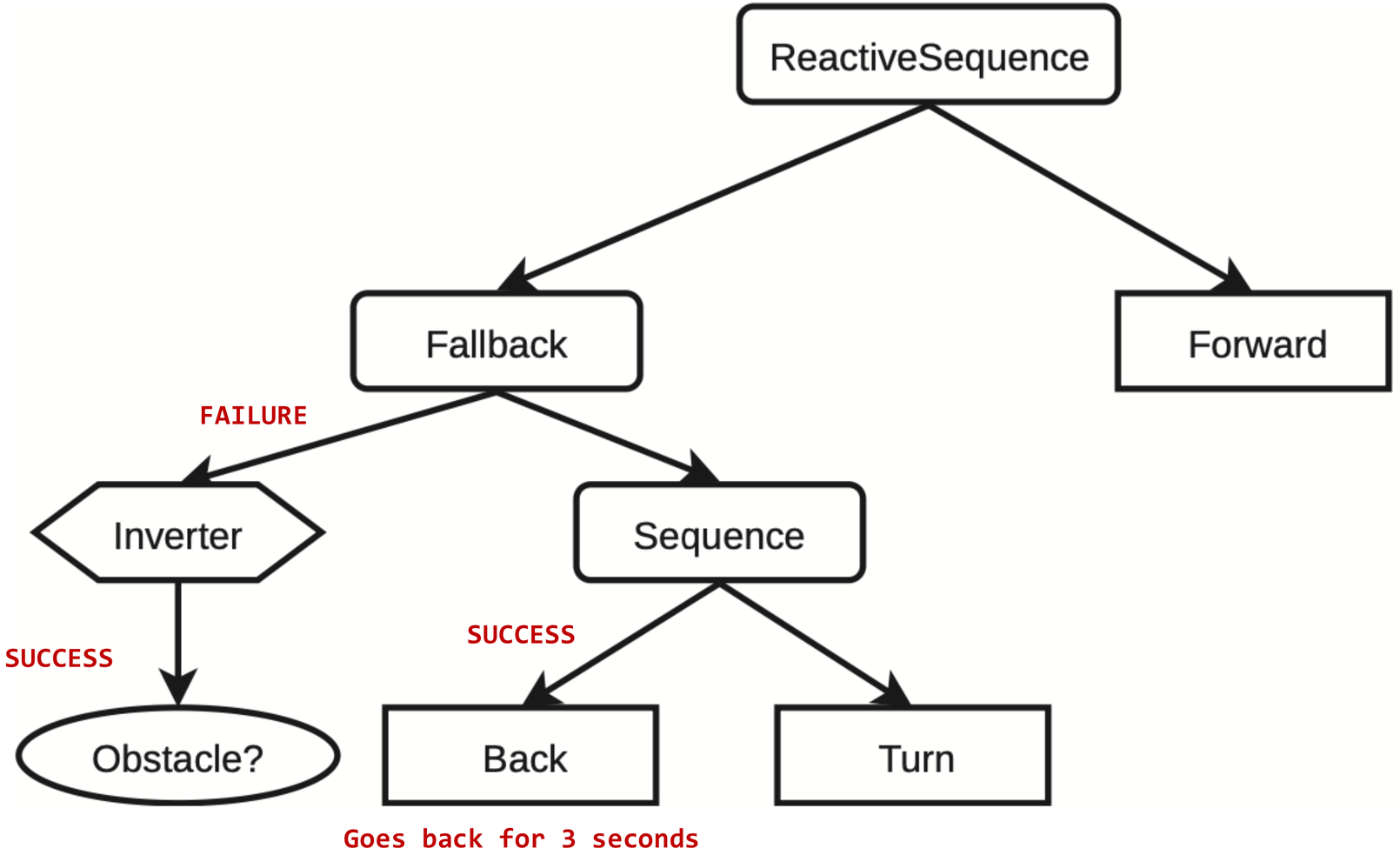
Bump&Go with Behavior Trees



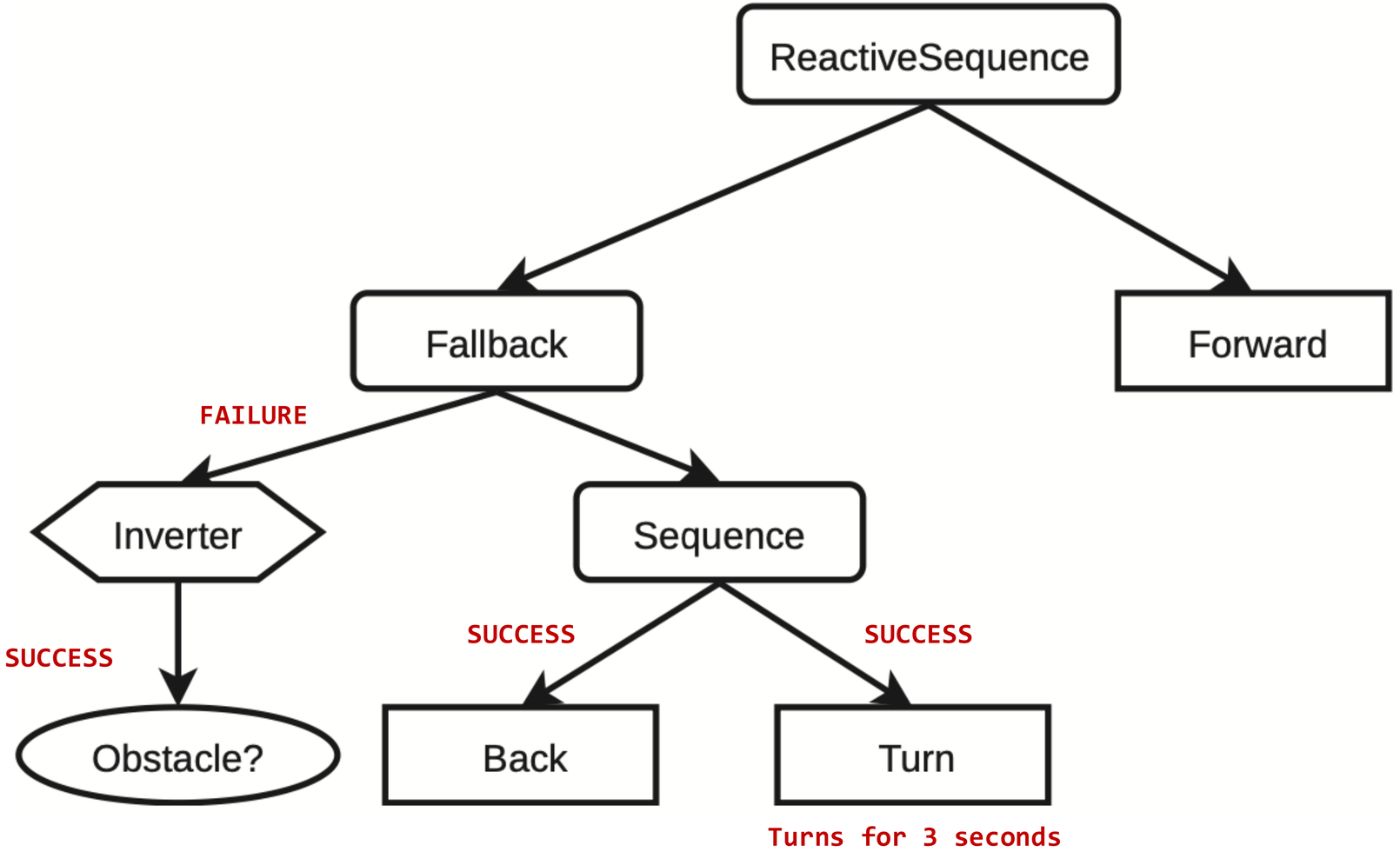
Bump&Go with Behavior Trees



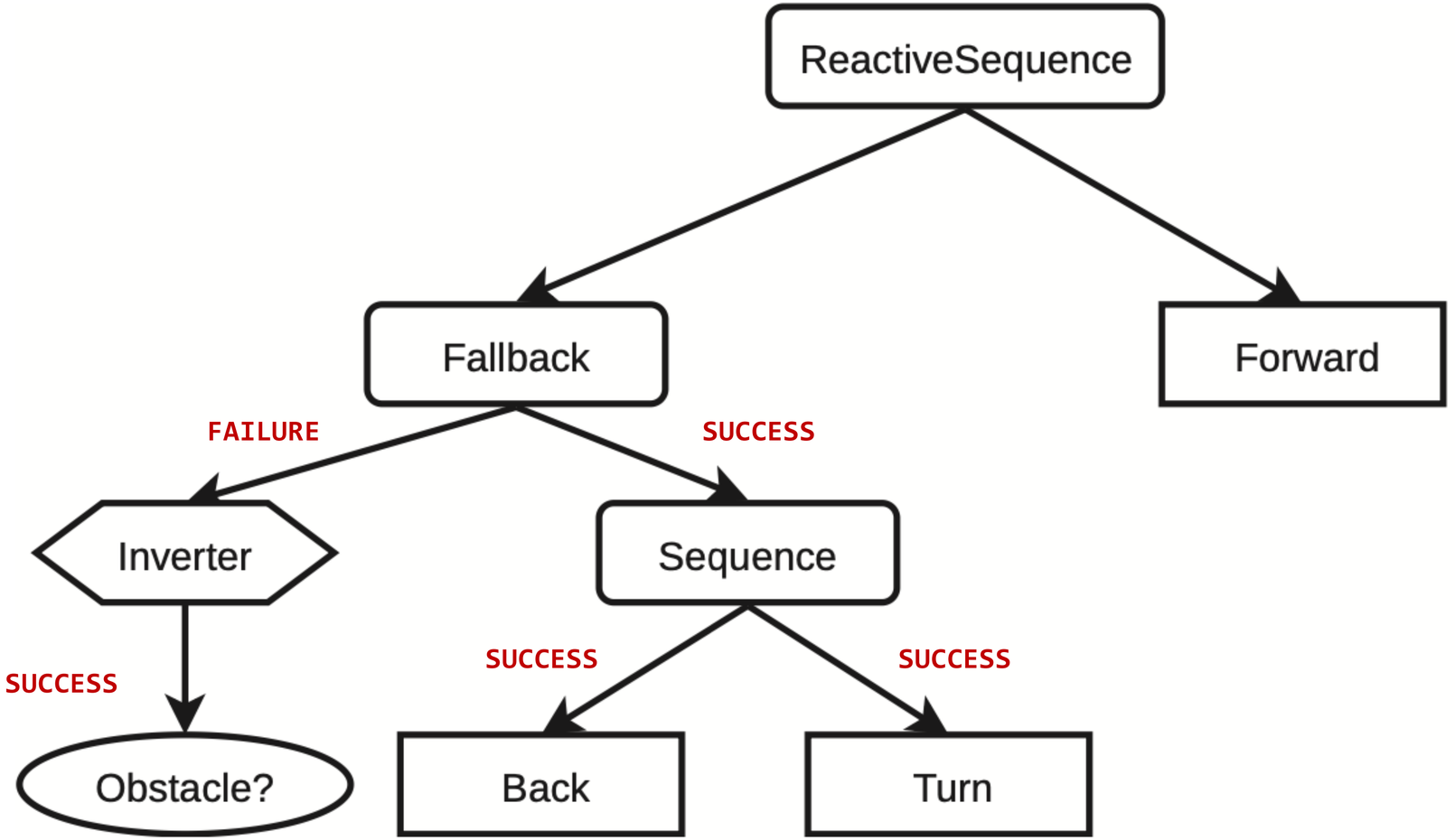
Bump&Go with Behavior Trees



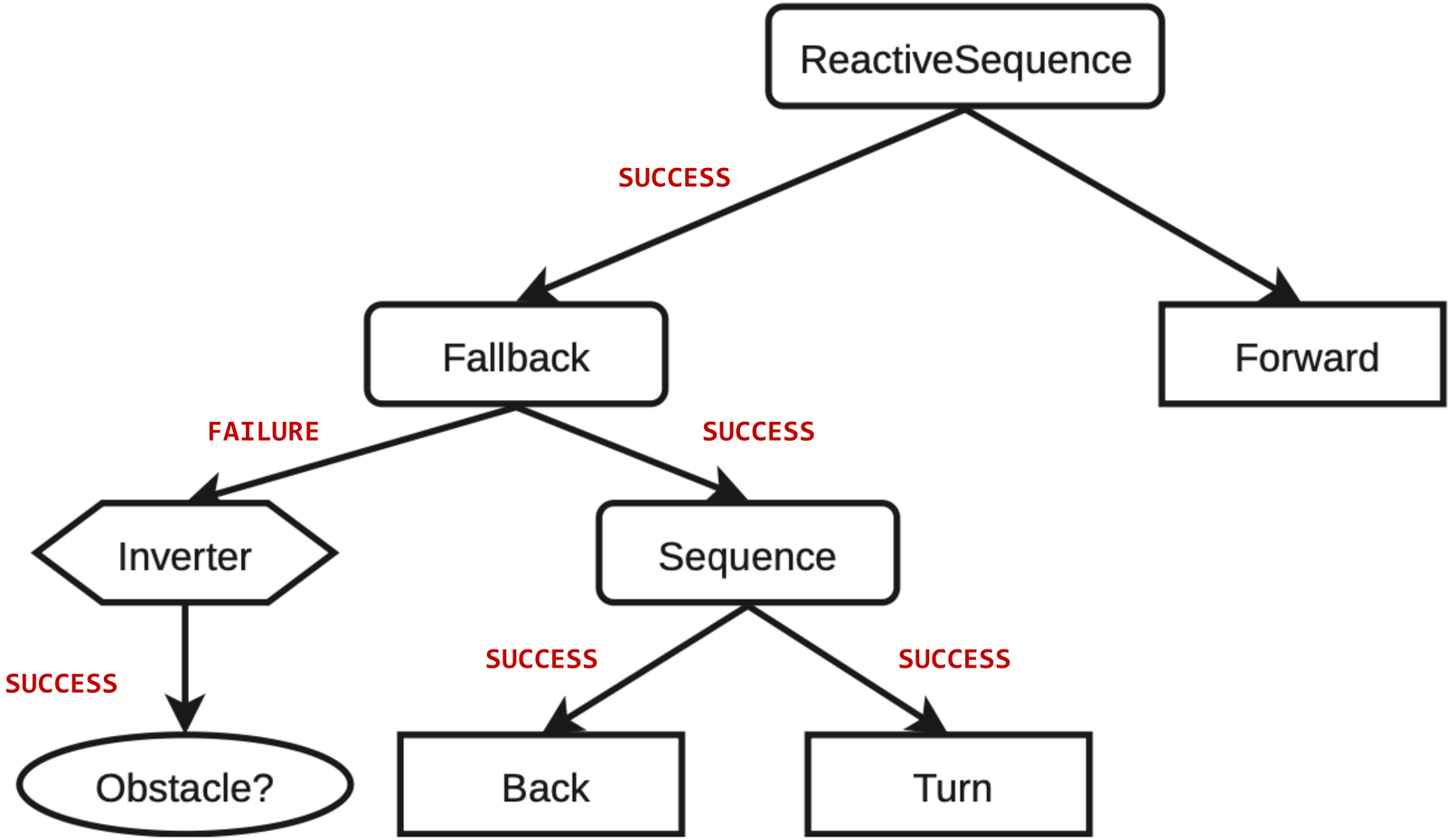
Bump&Go with Behavior Trees



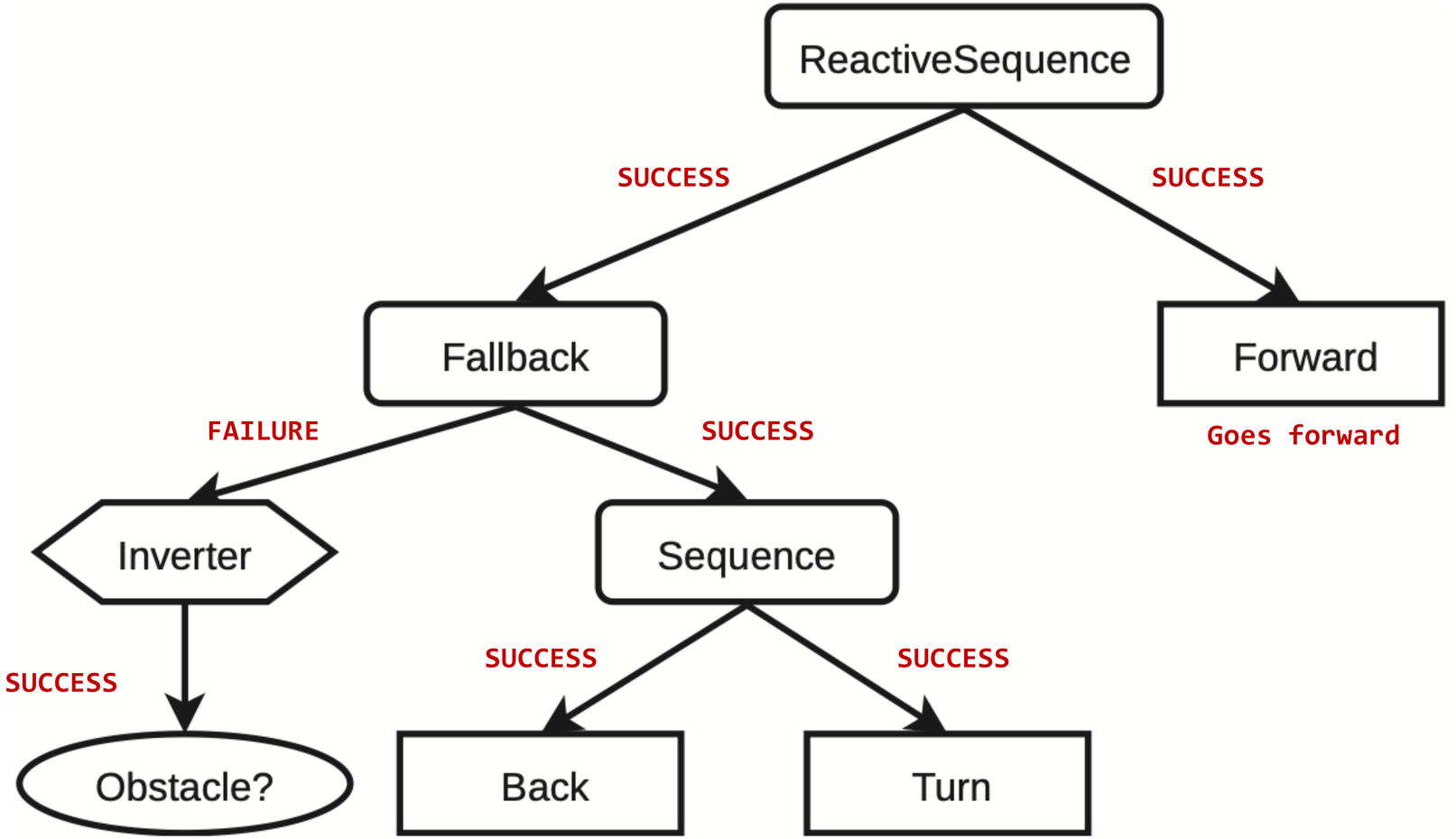
Bump&Go with Behavior Trees



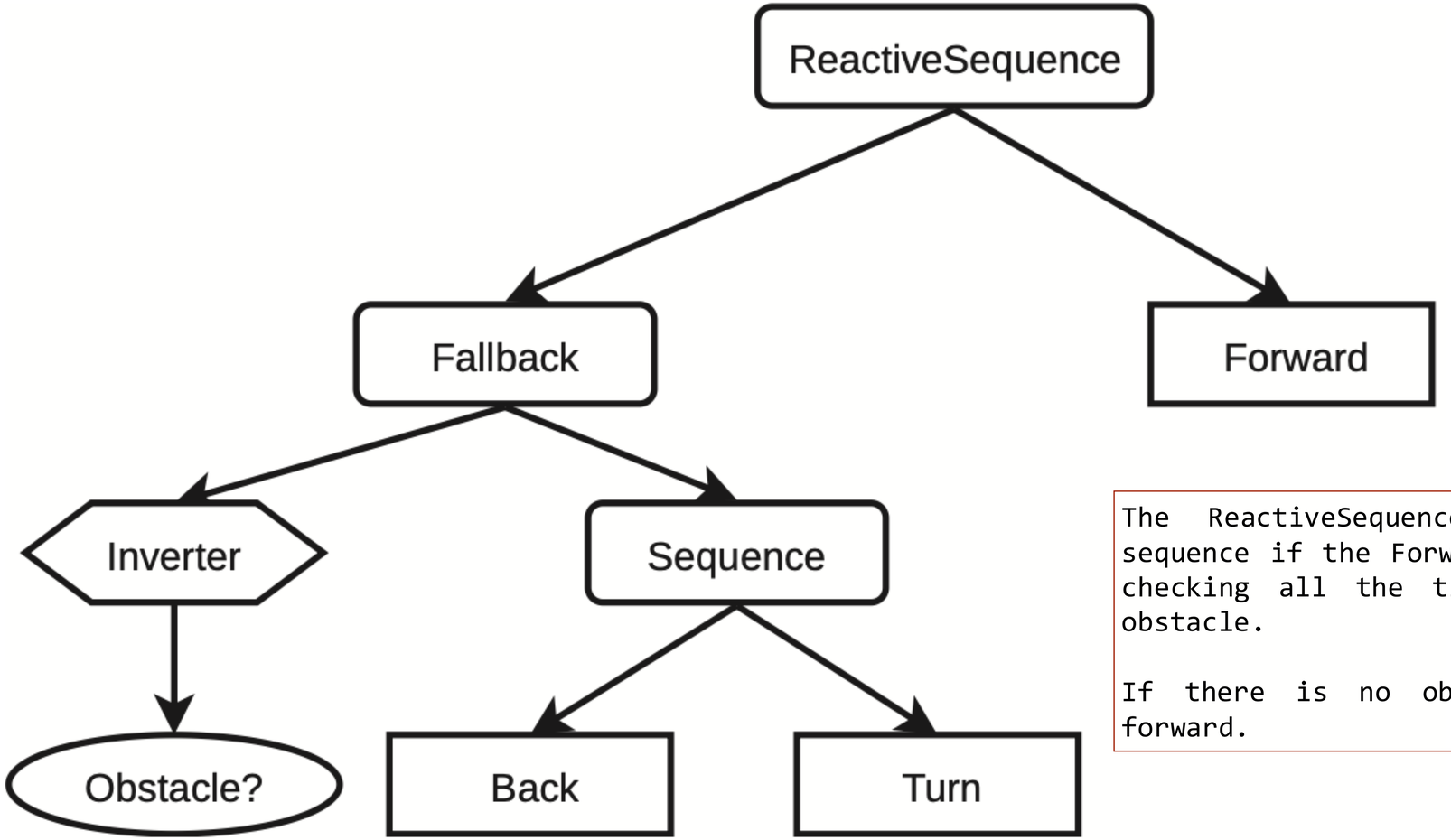
Bump&Go with Behavior Trees



Bump&Go with Behavior Trees



Bump&Go with Behavior Trees



The ReactiveSequence always restarts the sequence if the Forward node returs RUNNING, checking all the time wether there is an obstacle.

If there is no obstacle, keeps on going forward.

Bump&Go with Behavior Trees

Package content

```
br2_bt_bumpgo
├── behavior_tree_xml
│   └── bumpgo.xml
├── cmake
│   └── FindZMQ.cmake
├── CMakeLists.txt
├── include
│   └── br2_bt_bumpgo
│       ├── Back.hpp
│       ├── Forward.hpp
│       ├── IsObstacle.hpp
│       └── Turn.hpp
├── package.xml
├── src
│   ├── br2_bt_bumpgo
│   │   ├── Back.cpp
│   │   ├── Forward.cpp
│   │   ├── IsObstacle.cpp
│   │   └── Turn.cpp
│   └── bt_bumpgo_main.cpp
└── tests
    ├── bt_action_test.cpp
    └── CMakeLists.txt
```

Intelligent Robotics Lab

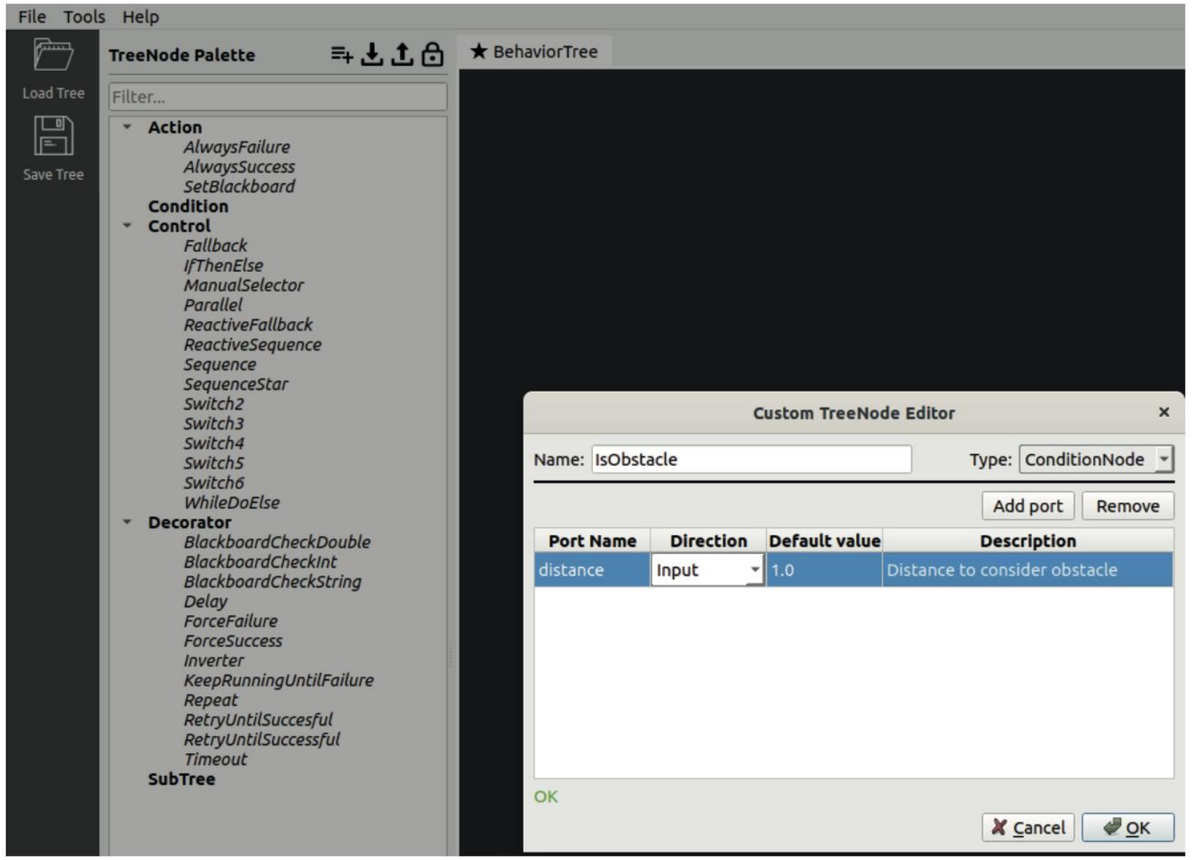
Bump&Go with Behavior Trees

Using Groot to create the Behavior Tree

```
$ ros2 run groot Groot
```

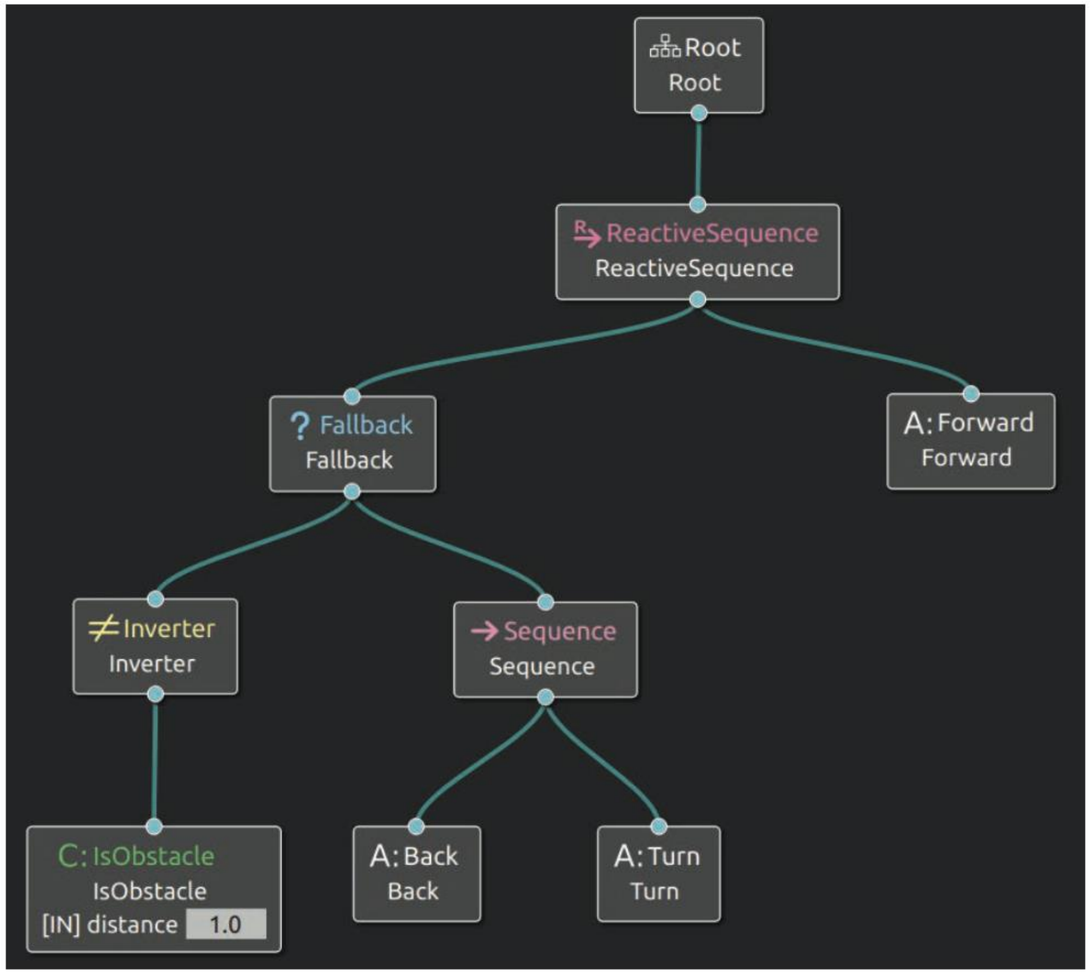
Bump&Go with Behavior Trees

Using Groot to create the Behavior Tree



Bump&Go with Behavior Trees

Using Groot to create the Behavior Tree




```
<?xml version="1.0"?>
<root main_tree_to_execute="BehaviorTree">
    <BehaviorTree ID="BehaviorTree">
        <ReactiveSequence>
            <Fallback>
                <Inverter>
                    <Condition ID="IsObstacle" distance="1.0"/>
                </Inverter>
                <Sequence>
                    <Action ID="Back"/>
                    <Action ID="Turn"/>
                </Sequence>
            </Fallback>
            <Action ID="Forward"/>
        </ReactiveSequence>
    </BehaviorTree>
    <TreeNodesModel>
        <Action ID="Back"/>
        <Action ID="Forward"/>
        <Condition ID="IsObstacle">
            <input_port default="1.0" name="distance">Dist to consider obst</input_port>
        </Condition>
        <Action ID="Turn"/>
    </TreeNodesModel>
</root>
```

Intelligent Robotics Lab

Bump&Go with Behavior Trees

BT nodes implementation

```
class Forward : public BT::ActionNodeBase
{
public:
  explicit Forward(
    const std::string & xml_tag_name,
    const BT::NodeConfiguration & conf);

  BT::NodeStatus tick();

  static BT::PortsList providedPorts()
  {
    return BT::PortsList({});
  }

private:
  rclcpp::Node::SharedPtr node_;
  rclcpp::Time start_time_;
  rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr vel_pub_;
};
```

Intelligent Robotics Lab

Bump&Go with Behavior Trees

BT nodes implementation: Forward

```
Forward::Forward(
  const std::string & xml_tag_name,
  const BT::NodeConfiguration & conf)
: BT::ActionNodeBase(xml_tag_name, conf)
{
  config().blackboard->get("node", node_);

  vel_pub_ = node_->create_publisher<geometry_msgs::msg::Twist>("/output_vel", 100);
}

BT::NodeStatus
Forward::tick()
{
  geometry_msgs::msg::Twist vel_msgs;
  vel_msgs.linear.x = 0.3;
  vel_pub_->publish(vel_msgs);

  return BT::NodeStatus::RUNNING;
}

}  // namespace br2_bt_bumpgo

#include "behaviortree_cpp_v3/bt_factory.h"
BT_REGISTER_NODES(factory)
{
  factory.registerNodeType<br2_bt_bumpgo::Forward>("Forward");
}
```

Intelligent Robotics Lab

Bump&Go with Behavior Trees

BT nodes implementation: Turn

```
BT::NodeStatus
Turn::tick()
{
  if (status() == BT::NodeStatus::IDLE) {
    start_time_ = node_->now();
  }

  geometry_msgs::msg::Twist vel_msgs;
  vel_msgs.angular.z = 0.5;
  vel_pub_->publish(vel_msgs);

  auto elapsed = node_->now() - start_time_;

  if (elapsed < 3s) {
    return BT::NodeStatus::RUNNING;
  } else {
    return BT::NodeStatus::SUCCESS;
  }
}
```

Intelligent Robotics Lab

Bump&Go with Behavior Trees

BT nodes implementation: isObstacle

```
void
IsObstacle::laser_callback(sensor_msgs::msg::LaserScan::UniquePtr msg)
{
  last_scan_ = std::move(msg);
}

BT::NodeStatus
IsObstacle::tick()
{
  double distance = 1.0;
  getInput("distance", distance);

  if (last_scan_->ranges[last_scan_->ranges.size() / 2] < distance) {
    return BT::NodeStatus::SUCCESS;
  } else {
    return BT::NodeStatus::FAILURE;
  }
}
```

Bump&Go with Behavior Trees

Build the BT nodes

```
add_library(br2_forward_bt_node SHARED src/br2_bt_bumpgo/Forward.cpp)
add_library(br2_back_bt_node SHARED src/br2_bt_bumpgo/Back.cpp)
add_library(br2_turn_bt_node SHARED src/br2_bt_bumpgo/Turn.cpp)
add_library(br2_is_obstacle_bt_node SHARED src/br2_bt_bumpgo/IsObstacle.cpp)

list(APPEND plugin_libs
  br2_forward_bt_node
  br2_back_bt_node
  br2_turn_bt_node
  br2_is_obstacle_bt_node
)

foreach(bt_plugin ${plugin_libs})
  ament_target_dependencies(${bt_plugin} ${dependencies})
  target_compile_definitions(${bt_plugin} PRIVATE BT_PLUGIN_EXPORT)
endforeach()

install(TARGETS
  ${plugin_libs}
  ARCHIVE DESTINATION lib
  LIBRARY DESTINATION lib
  RUNTIME DESTINATION lib/${PROJECT_NAME}
)
```

Intelligent Robotics Lab

Bump&Go with Behavior Trees

Running the code

```
int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);

  auto node = rclcpp::Node::make_shared("patrolling_node");

  BT::BehaviorTreeFactory factory;
  BT::SharedLibrary loader;

  factory.registerFromPlugin(loader.getOSName("br2_forward_bt_node"));
  factory.registerFromPlugin(loader.getOSName("br2_back_bt_node"));
  factory.registerFromPlugin(loader.getOSName("br2_turn_bt_node"));
  factory.registerFromPlugin(loader.getOSName("br2_is_obstacle_bt_node"));

  std::string pkgpath = ament_index_cpp::get_package_share_directory("br2_bt_bumpgo");
  std::string xml_file = pkgpath + "/behavior_tree_xml/bumpgo.xml";

  auto blackboard = BT::Blackboard::create();
  blackboard->set("node", node);
  BT::Tree tree = factory.createTreeFromFile(xml_file, blackboard);

  auto publisher_zmq = std::make_shared<BT::PublisherZMQ>(tree, 10, 1666, 1667);

  rclcpp::Rate rate(10);

  bool finish = false;
  while (!finish && rclcpp::ok()) {
    finish = tree.rootNode()->executeTick() != BT::NodeStatus::RUNNING;

    rclcpp::spin_some(node);
    rate.sleep();
  }

  rclcpp::shutdown();
  return 0;
}
```

Intelligent Robotics Lab

Bump&Go with Behavior Trees

Running the code

```
$ ros2 launch br2_tiago sim.launch.py
```

```
$ ros2 run br2_bt_bumpgo bt_bumpgo --ros-args -r input_scan:=/scan_raw -r
output_vel:=/key_vel -p use_sim_time:=true
```

Bump&Go with Behavior Trees

Running the code