Nicholas Pey, Piotr Bonar, Xavier Sánchez, Carlota Criado

Report - Behaviour Trees

Introduction

In this project, we worked with the AAPE platform (version 0.3.1) to build autonomous agents capable of making decisions using Behaviour Trees (BTs). Our focus wasn't just on designing agents that "work", but on making them modular, reactive, and extensible using clean, goal-based logic.

We implemented two main agent types:

- **Astronaut**, whose job is to roam around, detect and collect AlienFlowers, and bring them back to base.
    - We also implemented the "Avoid Critters" which its main goal is to take flowers and bring them to the base at the same time it avoids the critters that are chasing him.
- **Critter**, which is a hostile creature that detects the Astronaut and tries to attack it.

All behaviours are implemented as goals (asynchronous tasks) and integrated into py_trees-based BTs, allowing the agents to respond to changing conditions in real time.

Architecture Overview

The project is composed of three main modules:

- **Goals_BT.py**: Implements low-level atomic behaviours (goals) such as moving forward, avoiding obstacles, facing an object, collecting flowers, etc.
- **BT_Astronaut.py**: Implementation of behaviour nodes and behaviour tree for the Astronaut agent (for both Alone and Avoid)
- **BT_Critter.py**: Implementation of behaviour tree for the Critter agent.

Each behaviour tree is ticked asynchronously and consists of Selector and Sequence nodes to define priorities and task sequences.

Scenario 1: Alone + Avoid (BTAstronaut.py)

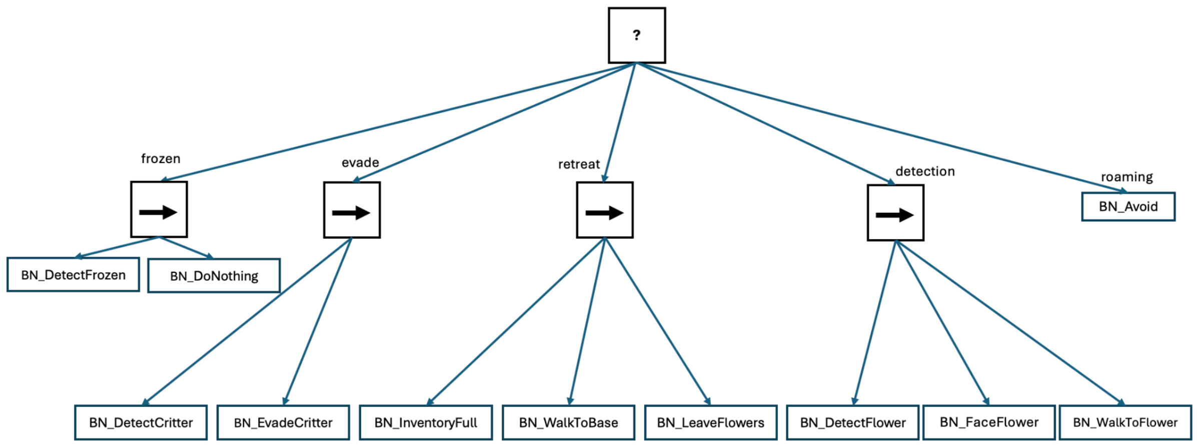
Objective

The goal is to implement a behavior tree that manages the autonomous actions of a single astronaut agent navigating an unknown environment. The agent must be capable of detecting and collecting alien flowers, avoiding obstacles and threats (critters), and returning to base to deposit resources, while adapting to environmental conditions such as being frozen by critters.

Implementation

This scenario has been implemented in file BTAstronaut.py and is using goals from Goals_BT.py from section marked as *NEW ASSIGNMENT - ASTRONAUT*

Behavior Tree



Specification of goals and behaviours:

1. **Waiting for the cooldown to finish (Frozen)**
   At the very top of the tree, we check whether the agent is frozen. If this condition is true, the tree halts execution of all subsequent behaviours by using *BN_DoNothing*. This ensures that the agent remains still and does not perform any unintended actions while in a frozen state.

2. **Evading Critters (Evade)**
   The next priority is evading the critters. We split this into 2 steps:

   BN_DetectCritter - The agent first checks whether the object sensed by its sensor has the tag "CritterMantaRay". If so, the node returns success, allowing going to the next step.

   BN_EvadeCritter - the agent immediately performs a 180-degree turn away from the threat, then runs forward to escape. This logic is triggered using a Sequence that first checks for Critters, and then runs the evasion task.

3. **Map Exploration (Roaming)**
   Uses *BN_Avoid*, which is based on the Avoid class implemented in Assignment 1. Specifically, the difference is that, if the agent has been walking in a straight line for 3

seconds or more, it will temporarily stop and perform a random turn by a randomly selected angle. This prevents the agent from getting stuck roaming along the edges of the map. By introducing randomized turning angles, the agent is more likely to explore the entire map, including its central areas.

4. **Returning flowers back to Base (Retreat)**
   The flower collection and deposit process is also divided into three steps:

   BN_InventoryFull – The agent first uses this node to determine if its inventory contains two flowers. If the inventory is full, the node returns success and the behavior tree progresses to the next step.

   BN_WalkToBase – The node triggers the WalkToBase task. This task instructs the agent to return to the base by sending the message "action", "walk_to,Base". For debugging purposes, it also verifies every 0.5 seconds whether the agent is still en route to its destination.

   BN_LeaveFlowers – Once the agent has arrived at the base, this node is executed. It launches the LeaveFlowers task, which sends the message "action", "leave,AlienFlower,2" to the agent, indicating that it should deposit two alien flowers.

5. **Collecting Flowers (Detection)**
   We split the approaching of the flower into 3 steps.

   BN_DetectFlower – The agent first checks whether the object sensed by its sensor has the tag "AlienFlower". If so, the node returns success, allowing progression to the next step.

   BN_FaceFlower – triggers the FaceFlower task. This determines whether the agent should turn left or right based on the flower's position relative to the center sensor (index 2). It calls the DirectedTurn class, which is an adaptation of the Turn class from Assignment 1. Unlike the original, DirectedTurn receives explicit direction and angle parameters, making the turn deterministic rather than random.

   BN_WalkToFlower – activates the WalkToFlower task. This node returns success when the number of flowers in the agent's inventory increases, indicating that a flower has been successfully collected.

Scenario 2: Critter (BTCritter.py)

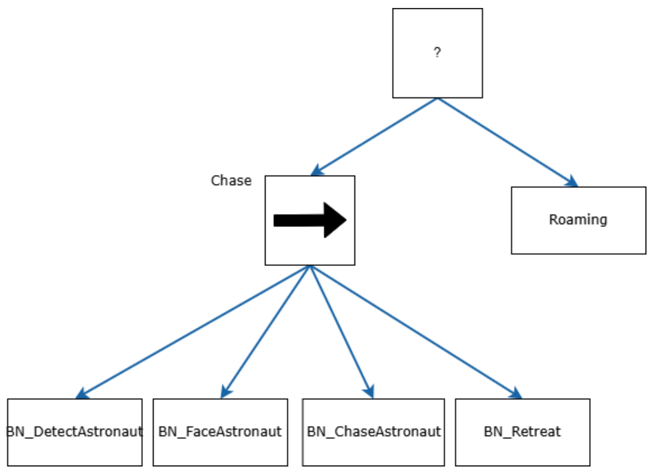
Objective

The CritterMantaRay (or just Critter) is designed to be the opposite of the Astronaut. It doesn't collect anything or explore randomly, it's a predator. Its only job is to detect the Astronaut, move toward it, and simulate an attack. After reaching the Astronaut, it retreats a bit, then continues roaming until it detects another target.

Implementation

To implement this behaviour, we built a dedicated BT_Critter class that assembles the behaviour tree using py_trees. Compared to the Astronaut's tree, this one is simpler and more focused, but still reactive.

Behaviour Tree



The Critter's BT is simple on purpose:

- It's purely reactive: detect → chase → retreat.
- It has no memory, no inventory, no goal beyond chasing.
- The structure is clean: if the attack fails, it tries again; if there's no target, it roams.

And because we used asyncio tasks for all goal logic, the Critter can interrupt its current movement at any time if it suddenly detects the Astronaut again.

How is it behaving in practice?

1. Critter starts roaming with BN_Avoid.
2. It detects an Astronaut → switches to Chase sequence.
3. Faces the Astronaut, walks to it, confirms proximity.
4. Turns 180º and walks away (retreat).
5. Returns to roaming.

This cycle can repeat as many times as needed, and the Critter never gets stuck in a state, it either acts or falls back cleanly.

Specification of goals and behaviours:

1. **Chasing the Astronaut (Sequence)**
   This is the core attack logic. It has been made a sequence, so that when one of the nodes fails, the whole behaviour would return failure. Also, it will only run if the Astronaut is actually visible. Otherwise, the BT falls back to the next node.

   BN_DetectAstronaut - checks whether the Astronaut is currently visible using the Critter's front-facing raycast sensors. If no Astronaut is detected, the node returns failure, causing the entire attack sequence to abort. The behavior tree then falls back to the default roaming behavior.

   BN_FaceAstronaut - turns the agent in the direction of Astronaut. It uses a directed turn based on the sensor index (left/mid/right) so the Critter can align itself correctly before chasing.

   BN_ChaseAstronaut - Once aligned, the Critter moves forward until it gets very close to the Astronaut. We consider the attack "successful" when the distance is under a threshold (implemented with 0.6 meters) and checks if we are not stuck very close to the astronaut (to account for collision or sensor blind spots). At that point, we stop the movement, because the Astronaut has been bitten.

   BN_Retreat - After the chase is done, the Critter turns 180º and walks forward until safe distance is reached, simulating a disengagement. This also avoids the agent getting stuck on the Astronaut's collider.

2. **Roaming (Fallback)**

   If the Critter doesn't detect any Astronauts, it falls back to roaming behavior. This uses the same *BT_Avoid* mechanism as the Astronaut, allowing the Critter to navigate the map while avoiding obstacles. Unlike the Astronaut, however, the Critter does not use random movement—its patrol behavior is purely reactive, driven by sensor-based avoidance. This enables it to eventually encounter an Astronaut again.

Conclusion

Working with Behaviour Trees in this project has shown how effective they are for structuring autonomous agent behaviour in a clean, modular way. Instead of relying on long, nested conditionals or ad hoc logic, we were able to break the agent's decision-making into small, reusable behaviours that can be combined and prioritized clearly.

The Astronaut agent, for example, needed to switch between exploring, collecting flowers, avoiding Critters, and returning to base, all depending on its current state and the environment. By using selectors and sequences, we could define this logic in a readable and extensible way. The Critter, even with a simpler goal (detect and attack), still benefited from the same structure, allowing it to react immediately when an Astronaut was nearby.