Assignment: Behaviour Trees - REVISED VERSION

IMPORTANT: For this assignment, be sure you are using version 0.3.1 of the AAPE environment. You will find the releases for each operating system as well as the updated guide in the virtual campus in the "AAPE - Autonomous Agents Practice Environment".

In this assignment, we will use Behaviour Trees to implement relatively complex behaviours within the AAPE platform. Our main characters will be the **Astronaut** and **CritterMantaRay** agents.

Testing Environment

- The primary test scene is named **"3-BT-Critter"**. This scene will be used to evaluate your agents (applying to all three scenarios described below).
- *Note:* You may use other scenes for additional testing if needed.

Implementation Scenarios

You are required to implement **three incremental scenarios**. The table below outlines the scoring for each successfully implemented scenario.

Scenario	Score	Comments
Alone	5-6-7	This is mandatory and necessary to pass the assignment. You can get 5, 6 or 7 points implementing this scenario.
Critters	+1	Implementing this scenario will increase your score by up to 1 point.
Collect-and-run	+2	Implementing this scenario will increase your score by up to 2 points.

Assistance Option for Behavior Tree Implementation

If you encounter difficulties implementing the core behaviors (goals) required for the behavior tree in the Alone scenario, you may request assistance. This support includes receiving the code for the primary goals, allowing you to focus on constructing the behavior tree itself and implementing only the simpler goals—avoiding the most complex parts.

Important Considerations:

- **Grading Impact:** Choosing this assistance limits your maximum score to a **5** (out of 7). You will not be eligible for a 6 or 7 in the *Alone* scenario.
- **Opportunity to Improve:** You can still raise your overall score by successfully implementing the *Critters* and *Collect-and-run* scenarios.

Scenario Alone

Our Astronaut has been assigned to the DarkHole deep-space collection outpost. Her mission is to collect a rare species of alien flower. Fortunately, she is alone at the outpost, so gathering the flowers should be straightforward.

The Astronaut starts at the Base outpost (spawn point 0). The number of alien flowers remains constant. Whenever the astronaut collects a flower, a new one will spawn randomly in the harvest zone after a short delay.

Behavior:

- **Wandering & Detection**:
  - The astronaut wanders the outpost searching for alien flowers (tag: `AlienFlower`).
  - When a flower is detected, she moves toward it and automatically collects it (added to her inventory).
- **Inventory Limit**:
  - She can carry only **two flowers** at a time.
  - When her inventory is full, she must return to the *Base Outpost* to unload (`action:leave,AlienFlower,2`).
- **Returning to Base**:
  - To return, she uses the **NavMesh system** (`action:walk_to,Base`).
  - For **testing purposes only**, she may also use instant teleportation (`action:teleport_to,Base`) to speed up the process (since the return trip in this scenario is always safe).

Scenario Critters

We now move to a new collection outpost, situated on a small moon near Saturn. Recently, hostile life forms have been discovered here (`"type": "AAgentCritterMantaRay"`).

Assignment: Implement the behaviour of these life forms.

Behaviour Requirements:

1. **Default State (Roaming):**
    - The critter roams aimlessly, avoiding obstacles.
2. **Astronaut Detection:**
    - If the critter detects the astronaut, it **starts following her** and attempts to bite (touch the astronaut).
    - Biting the astronaut implies the astronaut is stunned for 5 seconds, and if she had flowers in her inventory, she will lose one (this behavior is automatic and managed by AAPE, you don't have to program it in Python)
    - After biting the astronaut, the critter has to **move away** and allow the astronaut to recover.
    - If it loses contact, it **resumes roaming**.

TIPS:

- If you want to try with multiple critters, use the Spawner.py tool and the two Spawn Areas ("HarvestZone" or "SmallHarvestZone").
- Use a manually controlled Astronaut to test your Critters.

Scenario Collect-and-run

Our Astronaut has arrived at the Soylent Green outpost to continue with her task of collecting alien flowers. However, this time is not so easy. We have the critters wandering around and the Astronaut has to avoid them at all cost. A bite from a Critter means the Astronaut gets paralyzed for 5 seconds, moreover, the Critter takes one of the flowers from the Astronaut's inventory.

Behaviour Requirements:

We want the **Astronaut** to replicate the same behavior demonstrated in the *"Alone"* scenario, but now with an added requirement: **active avoidance of the Critters**.

Your implementation will be assessed based on how successfully the autonomous Astronaut evades the Critters.

Naturally, we expect the Astronaut's behavior to improve compared to the 'Alone' scenario when facing the Critters. The scoring will be based on this demonstrated improvement.

The 3-BT-Critter scene

Figure 1 shows the designated Spawn Areas. These areas serve two purposes:

Agent Spawning: Areas where agents appear (controlled by the "spawn_area" parameter in AAgent-X.json)

AlienFlowers Generation: Areas where AlienFlowers can randomly appear.

To adjust the number of AlienFlowers in each area, you can modify the file “config.json” in the directory “StreamingAssets”[1], which can be found in your directory release. The file has the form:

```
{
 "flowersInHarvestZone": 10,
 "flowersInSmallHarvestZone": 0,
 "flowersInNearBaseZone": 0
}
```

The default config.json file that comes with the release spawns 10 AlienFlowers in the HarvestZone area. Modify the parameters of the areas as needed for testing your agents. Be aware that the evaluation test will be performed by spawning AlienFlowers only in the “HarvestZone” area.

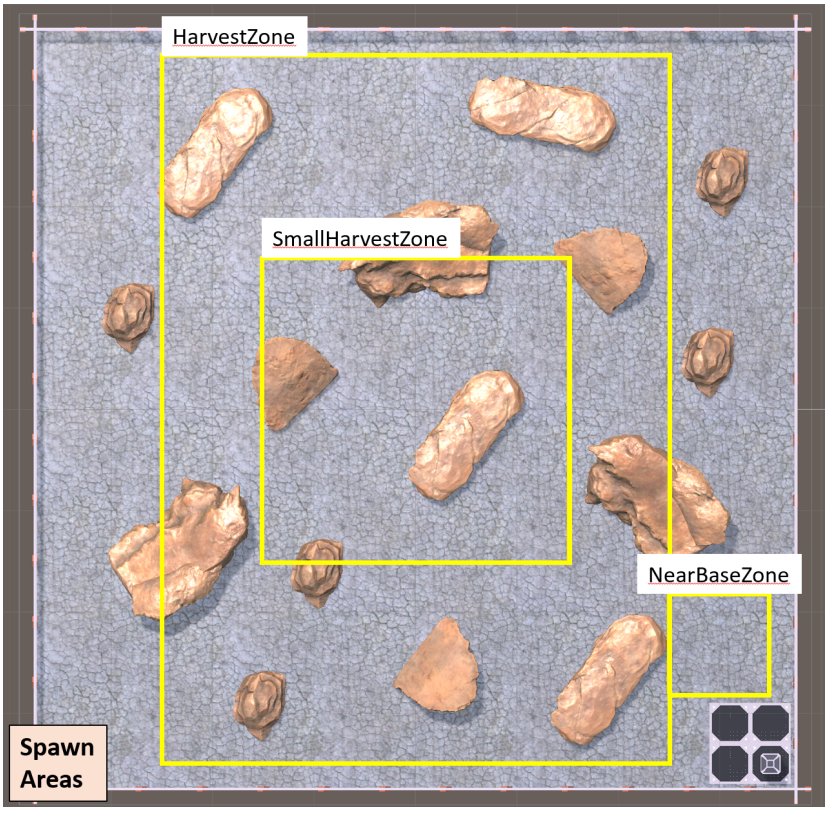


Figure 1

[1] Windows: AAPE_Data\StreamingAssets
Mac: Contents/Resources/Data/StreamingAssets
Linux: Linux-V0.3.0_Data/StreamingAssets

Figure 2 shows the named locations. These are the locations that can be used with the “walk_to” and “teleport_to” actions (“teleport_to” can use also spawn points as target teleport positions). Use the exact name together with the desired action.

The location “Base” is detected by the sensor of the Astronaut ({‘name’: ‘Base’, ‘tag’: ‘Location’...}).

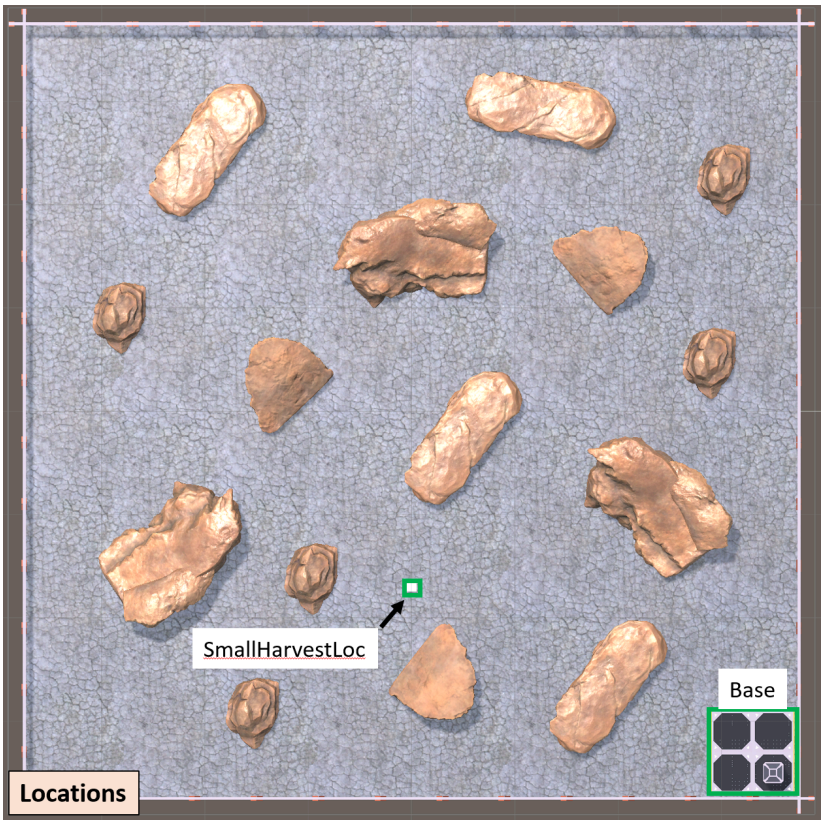


Figure 2

Figure 3 highlights the spawn points. These are the positions where you can spawn agents as specified in the "spawn_point" parameter in the AAgent-X.json file. You can also use these positions to teleport an agent using the tag “SpawnPointX” where X is the number in the figure.

For example:

```
action:teleport_to,SpawnPoint3
```

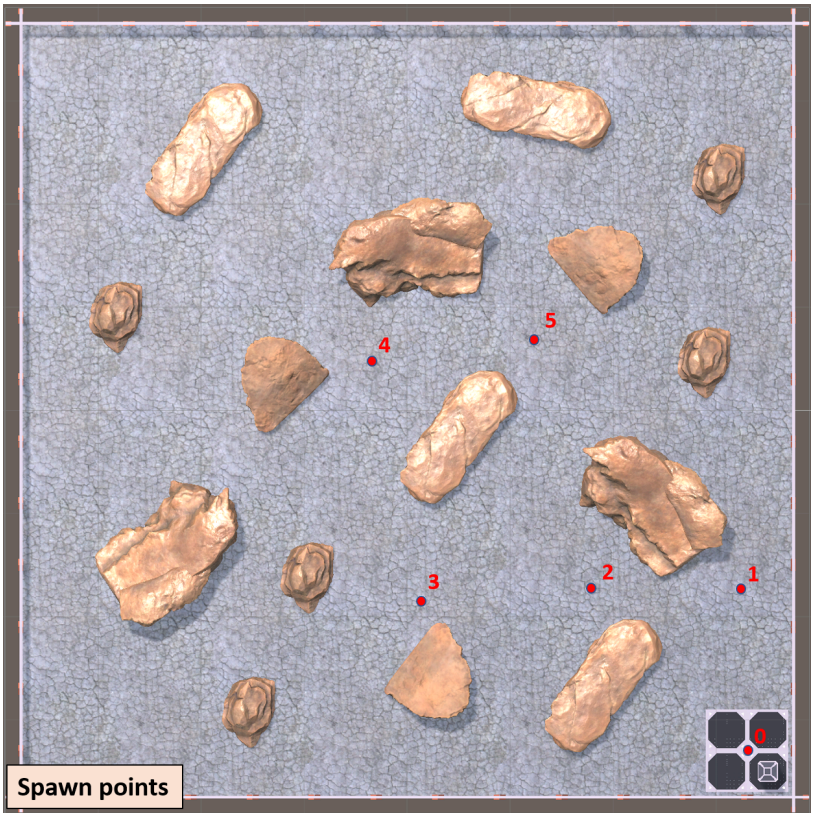


Figure 3

Figure 4 shows the area around the (only) container in this scene, where the Astronaut has to stay in order to interact with that container. If you *walk_to* the *Base* location, you will automatically be inside the area. This area is detected by the sensor of the Astronaut ({'name': 'Container', 'tag': 'Container'...}).

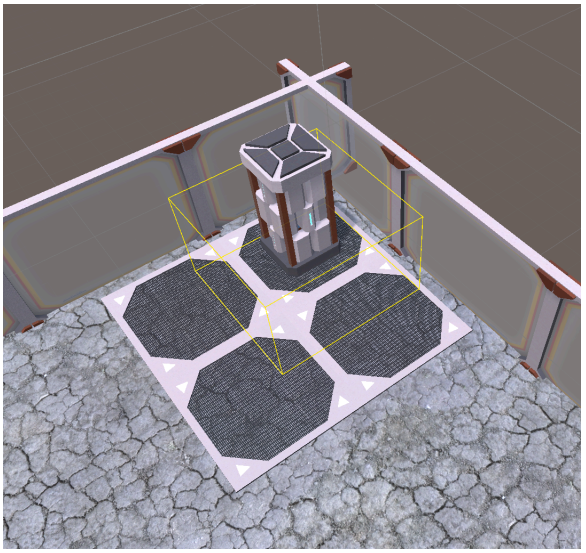


Figure 4

Finally, The **tags** of the different actors are the following:

 AlienFlower  Astronaut  CritterMantaRay

The rest of the elements of the scenario are tagged as “Rock” (each one of the rock formations) and “Wall” (the walls that delimit the perimeter).

Provided code

An example behaviour tree, `BTRoam.py`, is provided. This is the same one we discussed in class. Keep in mind that this is just a reference example.

The file `Goals_BT.py` includes three example goals (DoNothing, ForwardDist, and Turn), so you can try the behaviour tree in the BTRoam.py file. You should replace this file with the code you submitted in the Basic Behaviors assignment and add any other necessary goals.

`AAgent-1.json` is the configuration file for the **Astronaut**, while `AAgent-2.json` is for the **Critter** agent. You may modify these files as needed or create new ones. Additionally, `APackAstroCritters.json` is an example configuration file for the **Spawner** (`Spawner.pt`), which spawns **one Astronaut and ten Critters**.

The AAgent_BT.py is the agent's main code. Initially, you only need to modify that file to register your goals and behaviour trees.

```
# Reference to the possible goals the agent can execute
self.goals = {
    "DoNothing": Goals_BT.DoNothing(self),
    "ForwardDist": Goals_BT.ForwardDist(self, -1, 5, 10),
    "Turn": Goals_BT.Turn(self),
}

# Reference to the possible behaviour trees the agent can execute
self.bts = {
    "BTRoam": BTRoam.BTRoam(self)
}
```

AAgent_BT code section where you have to register the goals and behaviour trees so they can be called from the AAPE interface (“Send message” button).

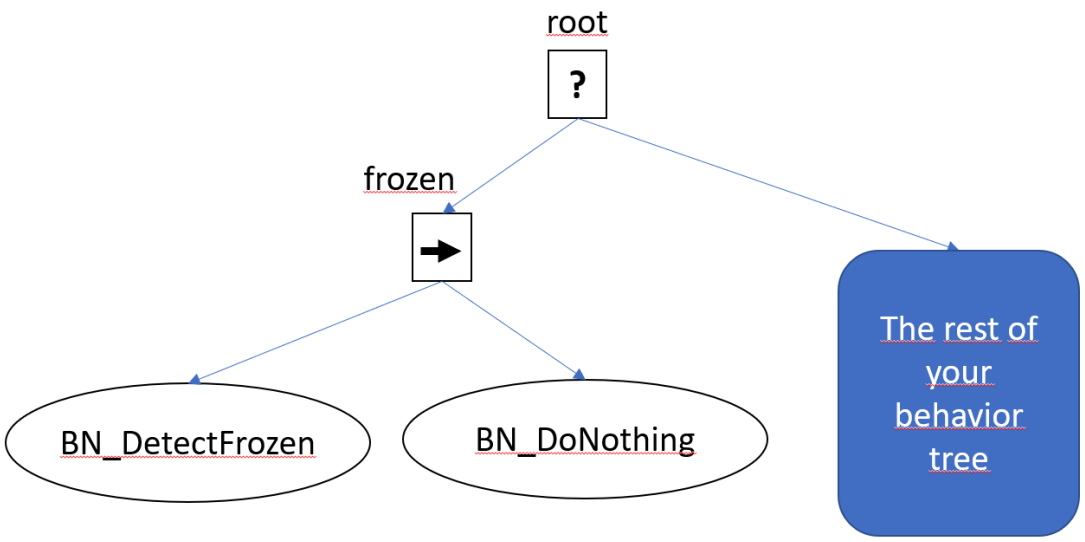
In this file, you have the possibility to modify the class **AAgent** to add extra code if you decide to add new functionality that cannot be implemented as goals or a behaviour tree. This is allowed, but you have to mark your code clearly like this:

```
<current code in the AAgent class>

#------------------------------ CUSTOM CODE ------------
<your custom code>
#-------------------------------------------------------

<current code in the AAgent class>
```

You cannot modify the provided functions or the current properties of the agent class. You are only allowed to add new elements.

IMPORTANT NOTE ABOUT THE IMPLEMENTATION OF THE BEHAVIOR TREE

To manage correctly the period when the astronaut is stunned after a critter's bite, your behavior tree has to comply with the following structure:



Since the AAPE simulation blocks incoming messages to the astronaut while stunned, this structure ensures your behavior tree does not send actions to the simulation during that period. This prevents potential issues (such as unexpected behavior) when the astronaut resumes using the behavior tree after recovering.

BN_DetectFrozen should have this implementation:

```
class BN_DetectFrozen(pt.behaviour.Behaviour):
    def __init__(self, aagent):
        self.my_goal = None
        # print("Initializing BN_DetectInventoryFull")
        super(BN_DetectFrozen, self).__init__("BN_DetectFrozen")
        self.my_agent = aagent
```

```
        self.i_state = aagent.i_state

    def initialise(self):
        pass

    def update(self):
        if self.i_state.isFrozen:
            return pt.common.Status.SUCCESS
        return pt.common.Status.FAILURE

    def terminate(self, new_status: common.Status):
        pass
```

BN_DoNothing is the one you already have in the example.

The part of the behavior tree will be:

```
frozen = pt.composites.Sequence(name="Sequence_frozen",
memory=True)
frozen.add_children([BN_DetectFrozen(aagent),BN_DoNothing(aagent)]
)

# THE REST OF YOUR BEHAVIOR TREE DEFINITION

self.root = pt.composites.Selector(name="Selector_root", memory=False)
self.root.add_children([frozen,
                        THE_ROOT_COMPOSITE_OF_THE_REST_OF_YOUR_TREE])
```

Submission

Please submit the **AAgent_Python** folder compressed in a single zip file with all the files you consider necessary to test your implementation.

Include a *pdf* document where you explain how you implemented the behaviour trees and highlight the main features of your implementation (5 pages maximum). Include also instructions about how to test your agents.