Build Your Own Robotic Pet Using Behavior Tree

Due: Monday, July 26th by 10:30am EST

(Credit: This project is adapted from an assignment in the "COMP135: Introduction to Artificial Intelligence" course at Tufts University, taught by Professor Santini in Spring 2021.)

Description:
During previous lectures, we discussed various python data structures including "class" as a way to organize data. In this project, we will use classes in python to implement something interesting.

In 2019, Boston Dynamics released Spot the robotic dog (watch the quick 2 minute video if you've never heard of Spot!). The Anki Vector robot can be a desktop pet that roams around autonomously, it can act as a voice assistant, and it takes care of itself by returning home when its battery is low. Think about the robots introduced by Dr. Jivko Sinapov during last week's faculty guest lecture on Wednesday about autonomous robots. Behind the scenes of those robots making decisions is a program containing code. That is what we'll be working on today - a robotic, autonomous pet living in your terminal!

In this project, you will be using a simple behavior tree that describes and governs the actions of a robotic pet. This is what we'll be using for the lab:

- Local python data structures: class, dictionary, list
- Python class and class hierarchy
- Robotic history and basic knowledge of behavior tree

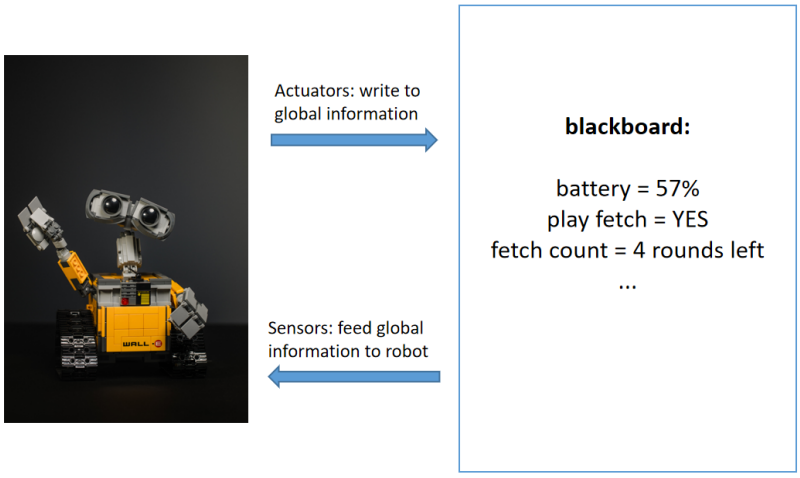
You will be implementing a few *nodes* in the behavior tree, which will allow your robotic pet to make certain decisions, like going to play fetch or not, autonomously (in other words, by itself). You can avoid repeating code in your program by using inheritance if two nodes share certain similarities -- for example, if they lead to the same result. After implementing the remaining nodes, they will then be stitched together to build the behavior tree for our robotic pet. More information about how nodes work is under the "Behavior Tree" section below.

You will then get information from the user and run the behavior tree until every task on the blackboard is finished. Note that in this lab, there is only one task that our robotic pet can complete - however, it will require an entire behavior tree to decide how the robotic pet will complete that task.

Reflex Agent:
We are implementing a reflex agent that acts solely based on the "current situation". By implementing a reflex agent, we are allowing our robotic pet to make decisions on its own - in other words, we are making it autonomous.

Information regarding the current situation is global information that is recorded on a blackboard - think of the blackboard as the robot's mainframe, where it's storing all of its internal information. The robotic pet will refer to its blackboard to help it make decisions. We can use a python dictionary to record such information.

The picture below demonstrates reflex agents' interaction with the global information on the blackboard.



Behavior Tree:

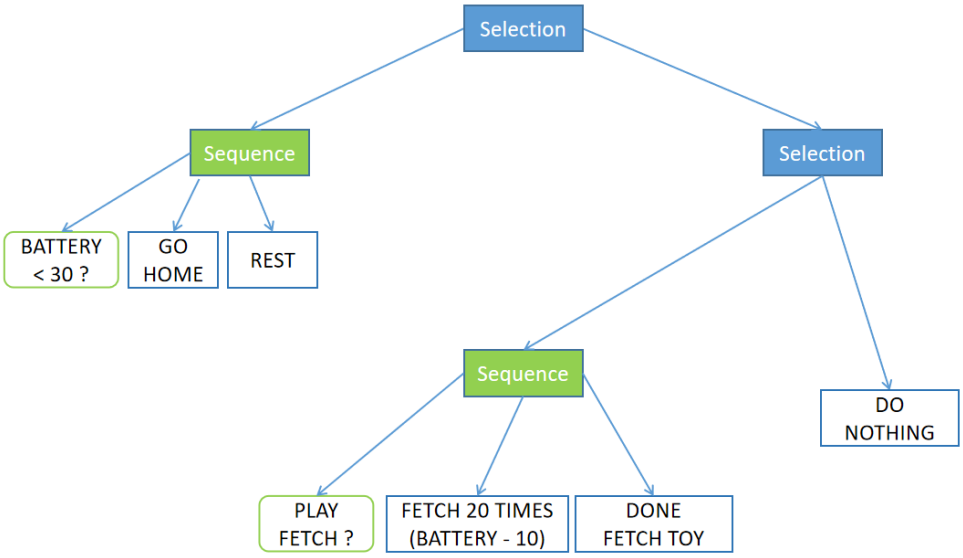
- Wikipedia introduces behavior trees as mathematical structures that describe "switchings between a finite set of tasks in a modular fashion".
- Behavior trees are made up with nodes. There are 3 types of nodes that have been implemented in the behavior tree.
- Node types:
    - Task node: each node represent an action of the agent [**YOU WILL NEED TO FILL OUT THE TASK NODES**]
    - Condition node: each node represent [Given]
    - Composite node: each node strings other nodes together [Given]
        - Sequence: children nodes executed in order; fails as long as one child fails
        - Selection: children nodes executed in order; succeeds as long as one child succeeds

Implementation:

- Blackboard
    - Blackboard is used to record information that is accessible by the robot while it is running. You can think of it as the robot's mainframe which stores important information for it to know. For example, some of the data that the blackboard records are the robot's battery level, the robot's unfinished tasks, etc.

- Some suggestions for blackboard implementation is to use a python dictionary.

- The behavior tree that is being implemented in our code is the one below. It captures a few actions of the robot:



  - The robot will first check if its battery is under 30%.
    - If yes, then it would go home and recharge and start the procedure.
    - If not, then it has enough battery to proceed to other tasks. It will check if the user wants to play fetch.
      - If the user does want to play fetch, then the robot will play fetch 5 times (note, if battery goes below 30%, then it will go home, rest, then continue playing the remaining games of fetch)
      - If the user does not want to play fetch, then the robot will do nothing.
  - The robot will terminate (or, exit the program) when there are no unfinished actions left on the blackboard.

- You will implement in total 5 classes out of 13 classes:
  - Level 0: 1 class representing a raw node,
  - Level 1: 3 classes inheriting from raw node with each representing a node type (task, condition, composite),
  - Level 2: 9 classes each representing a different node on the behavior tree inheriting from corresponding node type. [**YOU WILL IMPLEMENT THE 5 TASK NODES IN LEVEL 2**]

Support Code & Your Task / Expectation:
Your goal is to finish implementing the classes and blackboard, build the behavior tree, take users input and run the tree.

We have provided starter code for each class. Your job is to:

- understand and write your understanding of how inheritance works,
- understand and write your understanding of how a behavior tree shoots information through the tree,
- understand and write your understanding of what the blackboard does
- finish implementing each class (the remaining 5 level 2 classes are unfinished for you to implement):
    - GIVEN:
        - Sequence (given, but understand how it works)
        - Selection (given, but understand how it works)
        - check_battery (given, but understand how it works)
        - play_fetch (given, but understand how it works)
    - TO DO:
        - go_home **(implement, specific instruction in replit)**
        - Rest **(implement, specific instruction in replit)**
        - done_fetch **(implement, specific instruction in replit)**
        - Fetch **(implement, specific instruction in replit)**
        - do_nothing **(implement, specific instruction in replit)**
- build the behavior tree (given, understand how it works)
- The main function should:
    - Ask the user for information including: what is the starting battery level, and do you want your pet to play fetch.
    - Run the behavior tree.
    - Understand and write your understanding of what the second while loop does

- Extra credit: +30pts if you can build on top of the finished tree with at least 2 new tasks or condition nodes.

Files Given:

project4_starterCode – python starter code for implementing behavior tree

Helpful Resources:

- Some good place to read about what behavior tree is and some further explorations:
  - https://en.wikipedia.org/wiki/Behavior_tree_(artificial_intelligence,_robotics_and_control)
  - https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

Expected output:

Input: 1 100 1
Sample output:

```
Would you want to start the
stimulation: (0 for no, 1 for yes)1
Current battery level (please input
value from 0 to 100): 100
Do you want your pet to play fetch: (0
for no, 1 for yes)1

**********blackboard values **********
{'BATTERY_LEVEL': 100, 'PLAY_FETCH':
1, 'FETCH_COUNT': -1}
**********blackboard values **********

ENOUGH BATTERY_LEVEL:  100 %
doing fetch
DO PLAY_FETCH ON ROUND # 20
ENOUGH BATTERY_LEVEL:  99 %
doing fetch
DO PLAY_FETCH ON ROUND # 19
ENOUGH BATTERY_LEVEL:  98 %
doing fetch
DO PLAY_FETCH ON ROUND # 18
ENOUGH BATTERY_LEVEL:  97 %
doing fetch
DO PLAY_FETCH ON ROUND # 17
ENOUGH BATTERY_LEVEL:  96 %
doing fetch
DO PLAY_FETCH ON ROUND # 16
ENOUGH BATTERY_LEVEL:  95 %
doing fetch
DO PLAY_FETCH ON ROUND # 15
ENOUGH BATTERY_LEVEL:  94 %
doing fetch
DO PLAY_FETCH ON ROUND # 14
ENOUGH BATTERY_LEVEL:  93 %
doing fetch
DO PLAY_FETCH ON ROUND # 13
ENOUGH BATTERY_LEVEL:  92 %
doing fetch
DO PLAY_FETCH ON ROUND # 12
ENOUGH BATTERY_LEVEL:  91 %
doing fetch
DO PLAY_FETCH ON ROUND # 11
ENOUGH BATTERY_LEVEL:  90 %
doing fetch
DO PLAY_FETCH ON ROUND # 10
ENOUGH BATTERY_LEVEL:  89 %
doing fetch
DO PLAY_FETCH ON ROUND # 9
ENOUGH BATTERY_LEVEL:  88 %
doing fetch
DO PLAY_FETCH ON ROUND # 8
ENOUGH BATTERY_LEVEL:  87 %
doing fetch
DO PLAY_FETCH ON ROUND # 7
ENOUGH BATTERY_LEVEL:  86 %
doing fetch
DO PLAY_FETCH ON ROUND # 6
ENOUGH BATTERY_LEVEL:  85 %
doing fetch
DO PLAY_FETCH ON ROUND # 5
ENOUGH BATTERY_LEVEL:  84 %
doing fetch
DO PLAY_FETCH ON ROUND # 4
ENOUGH BATTERY_LEVEL:  83 %
doing fetch
DO PLAY_FETCH ON ROUND # 3
ENOUGH BATTERY_LEVEL:  82 %
doing fetch
DO PLAY_FETCH ON ROUND # 2
ENOUGH BATTERY_LEVEL:  81 %
doing fetch
DO PLAY_FETCH ON ROUND # 1
ENOUGH BATTERY_LEVEL:  80 %
doing fetch
done PLAY_FETCH
All tasks complete!

**********blackboard values **********
{'BATTERY_LEVEL': 80, 'PLAY_FETCH':
0, 'FETCH_COUNT': 0}
**********blackboard values **********
```

Thank you :)

Input: 1 100 0
Sample output:

```
Would you want to start the
stimulation: (0 for no, 1 for yes)1
Current battery level (please input
value from 0 to 100): 100
Do you want your pet to play fetch: (0
for no, 1 for yes)0

**********blackboard values **********
{'BATTERY_LEVEL': 100, 'PLAY_FETCH':
0, 'FETCH_COUNT': -1}
**********blackboard values **********

ENOUGH BATTERY_LEVEL:  100 %
start do_nothing
All tasks complete!

**********blackboard values **********
{'BATTERY_LEVEL': 100, 'PLAY_FETCH':
0, 'FETCH_COUNT': -1}
**********blackboard values **********

Thank you :)
```

Input: 0
Sample output:

```
Would you want to start the
stimulation: (0 for no, 1 for yes)0

**********blackboard values **********
{'BATTERY_LEVEL': 100, 'PLAY_FETCH':
1, 'FETCH_COUNT': -1}
**********blackboard values **********

Thank you :)
```

Input: 1 33 1
Sample output:

```
>>>python
coding101_amyYao_solution.py
Would you want to start the
stimulation: (0 for no, 1 for yes)1
Current battery level (please input
value from 0 to 100): 33
Do you want your pet to play fetch: (0
for no, 1 for yes)1

**********blackboard values **********
{'BATTERY_LEVEL': 33, 'PLAY_FETCH':
1, 'FETCH_COUNT': -1}
**********blackboard values **********

ENOUGH BATTERY_LEVEL:  33 %
doing fetch
DO PLAY_FETCH ON ROUND # 20
ENOUGH BATTERY_LEVEL:  32 %
doing fetch
DO PLAY_FETCH ON ROUND # 19
ENOUGH BATTERY_LEVEL:  31 %
doing fetch
DO PLAY_FETCH ON ROUND # 18
ENOUGH BATTERY_LEVEL:  30 %
doing fetch
DO PLAY_FETCH ON ROUND # 17
LOW BATTERY_LEVEL:  29 %
start go_home
start resting
RECHARGED BATTERY_LEVEL:  100 %
ENOUGH BATTERY_LEVEL:  100 %
doing fetch
DO PLAY_FETCH ON ROUND # 16
ENOUGH BATTERY_LEVEL:  99 %
doing fetch
DO PLAY_FETCH ON ROUND # 15
ENOUGH BATTERY_LEVEL:  98 %
doing fetch
DO PLAY_FETCH ON ROUND # 14
ENOUGH BATTERY_LEVEL:  97 %
doing fetch
DO PLAY_FETCH ON ROUND # 13
ENOUGH BATTERY_LEVEL:  96 %
doing fetch
DO PLAY_FETCH ON ROUND # 12
ENOUGH BATTERY_LEVEL:  95 %
doing fetch
DO PLAY_FETCH ON ROUND # 11
ENOUGH BATTERY_LEVEL:  94 %
doing fetch
DO PLAY_FETCH ON ROUND # 10
ENOUGH BATTERY_LEVEL:  93 %
doing fetch
DO PLAY_FETCH ON ROUND # 9
ENOUGH BATTERY_LEVEL:  92 %
doing fetch
DO PLAY_FETCH ON ROUND # 8
ENOUGH BATTERY_LEVEL:  91 %
doing fetch
DO PLAY_FETCH ON ROUND # 7
ENOUGH BATTERY_LEVEL:  90 %
doing fetch
DO PLAY_FETCH ON ROUND # 6
ENOUGH BATTERY_LEVEL:  89 %
doing fetch
DO PLAY_FETCH ON ROUND # 5
ENOUGH BATTERY_LEVEL:  88 %
doing fetch
DO PLAY_FETCH ON ROUND # 4
ENOUGH BATTERY_LEVEL:  87 %
doing fetch
DO PLAY_FETCH ON ROUND # 3
ENOUGH BATTERY_LEVEL:  86 %
doing fetch
DO PLAY_FETCH ON ROUND # 2
ENOUGH BATTERY_LEVEL:  85 %
doing fetch
DO PLAY_FETCH ON ROUND # 1
ENOUGH BATTERY_LEVEL:  84 %
doing fetch
done PLAY_FETCH
All tasks complete!
```

**********blackboard values **********
{'BATTERY_LEVEL': 84, 'PLAY_FETCH':
0, 'FETCH_COUNT': 0}
**********blackboard values **********

Thank you :)