UFCFXK-30-3: Digital Systems Project

Student name: Radiela Yorgova

Project Title: **Interactive tool for creating behaviour trees for dialog management**

Scan me!

Abstract:

A graphical user interface to allow non-experts to rapidly create behaviour trees for managing task-oriented dialogues – such as handling customer queries.

Aims and objectives:

- Aim is to separate designing a dialogue flow, from creation of underpinning AI component
- Tool allows rapid authoring of behaviour trees from a palette of existing natural language processing components
- Tool should also facilitate understanding the flow of conversation and improving generation of appropriate behaviour according to the context
- Evaluation: user studies establishment of behaviour trees for handling dialogues such as solving customers queries

Research:

Behaviour Trees (BTs) model the structure of switching between multiple tasks in autonomous agents. According to Colledanchise and Ögren (2018), they are a very efficient way of creating sophisticated programs that are both modular and reactive. The tree structure is hierarchical, and the individual states are leaf nodes - every state is an individual module, and their switching does not alter the tree structure.

BTs have two types of structural nodes – condition and action. Condition nodes construct the hierarchy of the tree, whereas action nodes complete a certain task when reached. Nirwan (2020) describes condition nodes as "control flow" and action nodes as "execution". Behaviour Trees can be used to enhance task-based dialog systems by interpreting the flow of conversations with the "control flow" and reacting to the users' actions when executing leaf nodes.

When working with BTs, visualization is part of the testing. Adam Feuer (2017) provides a system for modeling trees by creating, renaming, or deleting nodes, which is suitable for the task. It provides a clear overview of the chosen BT model. He uses the d3 library of JavaScript for visualizing the tree from JSON tree data.

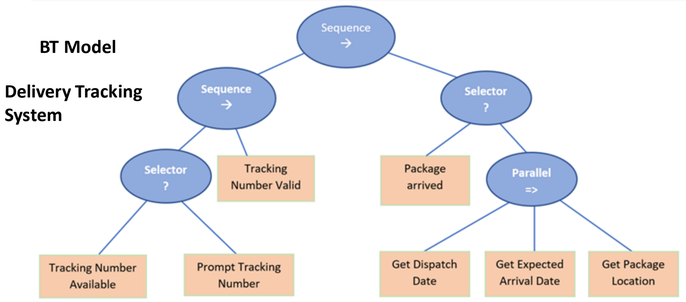
Key requirements:

	MoSCoW
Functional	
• A system that reads JSON files	Must
• The ability to add child nodes	Must
• The ability to remove child nodes	Must
• The ability to generate new JSON files after alternation	Must
• The ability to display a tree from JSON file	Must
• A system to generate running code from a JSON file	Must
• The ability of all systems to interact with JSON files	Must
• A simple method for parsing JSON tree and transforming it into executable code	Must
• The ability to add sequence/selection nodes	Must
• The ability to add parallel nodes	Could
Non-functional	
• The ability of all systems to interact with JSON files	Must
• A simple method for parsing JSON tree and transforming it into executable code	Must
• A form of global blackboard/memory	Must

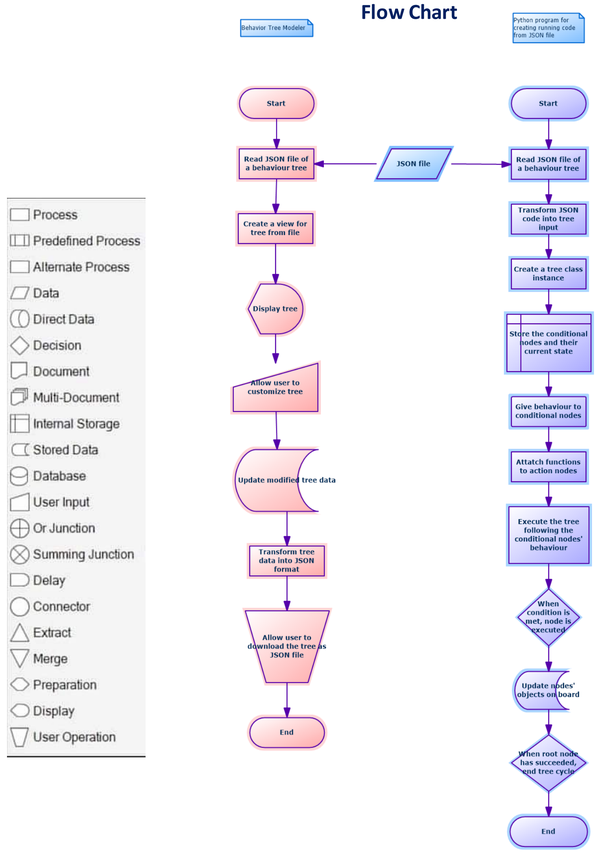
Design, Implementation & Testing

BT Model

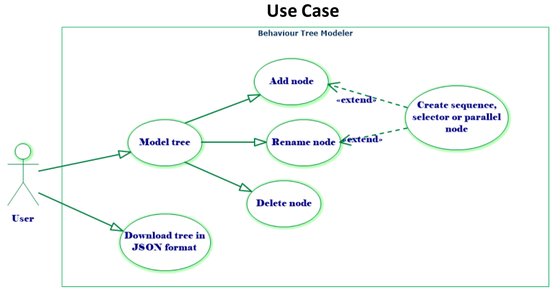
Delivery Tracking System



Flow Chart

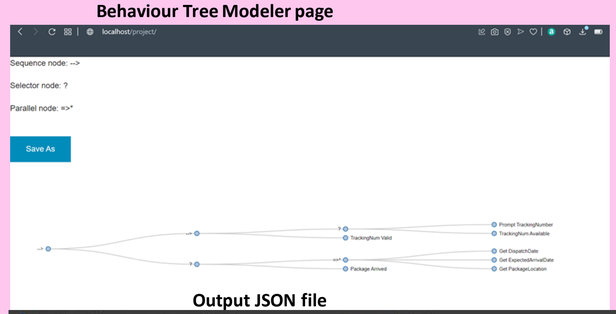


Use Case



Implementation artefacts

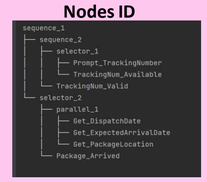
Behaviour Tree Modeler page



Output JSON file

Python program for getting running code from JSON file

Nodes ID



Example database

Traking number	Delivery address	Dispatch date	Expected Arrival	Current location
df112888k9	BS16 1ZU, Bristol, United Kingdom	2022-02-18	2022-02-25	BS16 1ZU, Bristol, United Kingdom
ar660m998	BS01 0AU, Bristol, United Kingdom	2022-02-22	2022-03-04	London, United Kingdom
wf2258fu76	BS16 5GF, Bristol, United Kingdom	2022-03-02	2022-04-05	Chelsea, United Kingdom

Testing

Test	Testing Outcome	MoSCoW
Test if the JavaScript system successfully reads JSON files	Success	Must
Test adding a child node	Success	Must
Test removing a child node	Success	Must
Test if the system displays a tree and whether it's accurate	Success	Must
Test if the system generates JSON files successfully	Success	Must
Test if the Python program successfully reads JSON files	Success	Must
Test if the system generates executable code from JSON files	Success	Must
Test if all systems can work with JSON files	Success	Must
Test if sequence/selection nodes are added to the tree successfully	Success	Must
Test if parallel node is successfully added to the tree	Success	Could
Test if traversing the tree is accurate	Success	Must

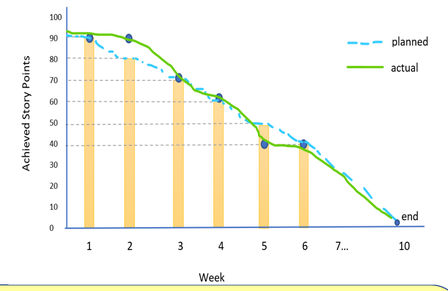
Planning and Management:

1 week/10 points/task

1) Design first behaviour tree
2) Create a system that can load a tree from JSON file and display it
3) Allow user to modify tree
4) Allow user to download modified tree as JSON file

2 weeks/20 points/task

5) Design Python program that reads JSON file and creates running code from the file
6) Connect a chatbot to the program, that recognizes whether the user's request has been answered
7) Display the execution of the tree



References:

- Colledanchise, M. and Ögren, P., (2018), *'Behavior Trees in Robotics and AI: An Introduction'. arXiv:1709.00084* [cs]
- Nirwan, D. (2020) *Designing AI Agents' Behaviors with Behavior Trees*. Available from: https://towardsdatascience.com/designing-ai-agents-behaviors-with-behavior-trees-b28aa1c3cf8a.
- Adam Feuer (2017*) d3js Tree Editor.* Source: https://gist.github.com/adamfeuer/042bfa0dde0059e2b288
- Bostock, M. (no date) *D3.js - Data-Driven Documents*. Available from: https://d3js.org.
- PyPI (2020) *Anytree – powerful and lightweight Python Tree Data Structure.* Available from: https://pypi.org/project/anytree