

## **Interactive tool for creating Behaviour trees for dialogue management**

---

Radiela Milkova Yorgova – 19017297

UFCFXK-30-3 - Digital systems project 2021/2022

## Acknowledgements

I would like to thank Jim Smith for sharing his brilliant ideas  
and providing continuous guidance and support  
throughout this project.

I would like to thank Nathan Duran for his guidance and helpful  
feedback from Project in Progress Day

## Abstract

This project presents an interactive interface for creating behaviour trees for dialogue management. Behaviour trees are mostly used in games and are relatively new to dialogue management. The main focus is to bring out the strengths of BTs, especially their re-usable component, to non-AI experts, allowing them to rapidly create models of their own.

The project represents a pallet of re-usable components that aid the user in modelling BTs, generating the appropriate running code, and testing the BT flow through a dialogue management system. The dialogue management system includes encryption of the conversations for security purposes and their decryption for maintenance.

## Table of Contents

1.	Introduction .....	7
1.1.	Aims and Objectives .....	7
1.2.	Ethical Review.....	8
2.	Literature Review.....	8
2.1.	Behaviour Tree Origin .....	8
2.2.	Finite State Machines (FSM) .....	8
2.3.	Hierarchical State Machines (HSM) .....	9
2.4.	Behaviour Trees .....	9
2.5.	Behaviour Trees and Task-based Dialogue Systems.....	10
2.6.	Behaviour Tree Nodes .....	10
2.7.	Implementation .....	11
2.8.	Tools.....	11
2.8.1.	Behaviour Tree Editor .....	11
2.8.2.	D3JS Tree Editor .....	13
2.8.3.	Collapsible Tree Search.....	15
2.8.4.	Conclusion on Tools .....	15
2.9.	Natural Language Processing.....	16
2.10.	Dialogue Management Systems .....	16
2.10.1.	Traditional Pipeline .....	16
2.10.2.	End-to-end method .....	17
2.11.	Dialogue Manager.....	17
2.11.1.	Dialogue Control .....	17
2.12.	Chatbots.....	18
2.12.1.	Chatbots and Business.....	18
2.	Requirements.....	19
2.1.	Use Case Diagrams.....	19
2.2.	Non-functional Requirements .....	20
2.3.	Functional Requirements.....	21
2.3.1.	Table 1: Functional Requirements for the BT modeler GUI .....	21
2.3.2.	Table 2: Functional Requirements for the Code Generating system .....	22
2.3.3.	Table 3: Functional Requirements for the Dialogue Management system .....	23
2.4.	Acceptance Tests .....	24
2.4.1.	Table 4: Acceptance tests .....	24
4.	Design .....	26
4.1.	User stories .....	26

4.1.1. Table 5: User stories .....	26
4.2. System Architecture .....	28
4.2.1. Figure 15: Train of thought .....	29
4.2.2. Flowchart Diagrams .....	29
4.3. User Interface Design .....	32
4.3.1. Figure 18: GUI Design .....	32
4.4. Pseudocodes .....	33
4.4.1. BT running code generating system .....	33
4.4.2. Chatbot system .....	33
4.5. Text Encryption and Decryption algorithms .....	33
5. Methodology.....	34
6. Implementation .....	35
6.1. Building a GUI for creating BT models .....	35
6.1.1. Implement a download button for extracting BT models as JSON files .....	35
6.1.2. Implement an upload button for loading a chosen JSON BT model .....	37
6.1.3. Make GUI design more approachable .....	38
6.1.4. Stages of GUI design .....	38
6.1.5. Acceptance tests passed from this system.....	40
6.2. Building a BT Code Generating program .....	40
6.2.1. Data processing .....	40
6.2.2. Leaf nodes' functions.....	42
6.2.3. Attaching functions to the leaf nodes .....	42
6.2.4. Running the behaviour tree.....	43
6.2.5. Displaying the tree model and visited nodes .....	43
6.2.6. Acceptance tests passed by this system.....	44
6.3. Dialogue management system – chatbot.....	44
6.3.1. Behaviour data processing .....	45
6.3.2. Conversation data processing .....	45
6.3.3. Chatbot's conversation scope .....	46
6.3.4. Conversation evaluation.....	46
6.3.5. Acceptance tests passed by this system.....	47
6.4. User Testing .....	48
7. Project Evaluation.....	49
8. Conclusions and Further works .....	50
8.1. Future works .....	51
9. Referencing .....	51

10. Appendices.....	54
10.1. Instructions for adapting new BT models to the project system .....	54
10.2. Instructions for changing the scope of the chatbot's conversations .....	54
10.3. Leaf functions provided during user testing of adapting a new BT model to the system .....	55
10.4. Database example for Banking BT model.....	55

## Table of Figures

Figure 1: A behaviour tree model for Pacman's combative behaviour (Colledanchise and Ögren, 2018) .....	9
Figure 2: BT model for parcel tracking dialogue.....	10
Figure 3: Syntax of VS Code Behaviour Tree Editor – the language suggested by Abad, D. (2021).....	11
Figure 4: Visualization of BT model using VS Code Behaviour Tree Editor .....	12
Figure 5: Testing BT model using VS Code Behaviour Tree Editor .....	12
Figure 6: Complete tree cycle of BT model using VS Code Behaviour Tree Editor .....	13
Figure 7: BT model using d3js Tree Editor .....	13
Figure 8: Tracking Delivery Services model with d3js Tree Editor.....	14
Figure 9: JSON file for generating the example model on Figure 8.....	14
Figure 10: A tree model using Collapsible Tree Search .....	15
Figure 11: Traditional Pipeline for Task-oriented Systems. (Jiang Zhao, Ling Li, and Lin, 2019).....	16
Figure 12: Use Case for behaviour trees modeler system.....	19
Figure 13: Use case for generating running code system .....	20
Figure 14: Use case for requesting service from the chatbot system .....	20
Figure 15: Train of thought of the project system .....	29
Figure 16: Flowchart for BT modelling GUI and BT code generating system.....	30
Figure 17: Flowchart diagram for code generating and dialogue management systems.....	31
Figure 18: GUI Design .....	32
Figure 19: Code for recording updates of the graphical BT model in a JSON formatting .....	36
Figure 20: Saving updated tree data code.....	36
Figure 21: Download event and 'download' button function in index.html.....	37
Figure 22: Upload file function .....	38
Figure 23: Design for Project in Progress Day .....	39
Figure 24: The final BT modelling GUI design.....	39
Figure 25: JSON string vs Anytree string.....	40
Figure 26: Pattern for transforming JSON string into Anytree code .....	41
Figure 27: Reusable function for connecting to a database.....	42
Figure 28: The database for the tracking delivery example system.....	42
Figure 29: 1-Tree structure, 2-Service not prompted, 3- Service prompted.....	43
Figure 30: Chatbot behaviour training function .....	45
Figure 31: Functions for recognizing the context of topics and providing a suitable response .....	46
Figure 32: Function for gathering user feedback .....	47
Figure 33: Model for handling banking services customer queries .....	48
Figure 34: Summative user tests functionality feedback .....	49

## 1. Introduction

A Behaviour Tree (BT) is a hierarchical data structure that enables systems to mimic the decision-making process of humans, thus increasing their intelligence. BTs control the behaviour by defining a set number of actions to be executed in response to a trigger by the user. The action-execution is planned beforehand and embedded into the program's logic as the hierarchy of the tree. The data structure can get applied to various systems, including chatbots, video games - the non-player characters (NPCs), automatically operated machines (robots), and more.

The real-world problem addressed by this project is a rapid and approachable implementation of Behaviour Trees (BT) for managing task-oriented dialogues, such as handling customer queries. Chatbot systems for customer queries are gaining more and more popularity as they bring many advantages to businesses. They can reduce the workload for employees by handling mundane tasks. Reduce waiting time and number of clients on the lines for human operators, and establish a connection between users and the company.

As further described in the Research section, Behaviour Trees enhance the performance of task-oriented dialogue systems by managing their dialogue flow. The BT hierarchy gets based on human decision-making logic – how a human thinks the system should react in different situations. That makes the data structure very powerful for decision-making tasks and simplifies its design. The implementation relies on visualizing BT models and their testing.

The project focuses on demonstrating the strengths of behaviour trees, especially their reusability, to non-AI experts. The idea is to provide a system that allows non-AI experts to build BTs from a pallet of easy-to-understand components. The project divides into two main systems - a graphical user interface for creating BT models and a system generating running code from such models. The objective is to use JSON files for the interexchange of data between the two systems. JSON formatting is chosen for its readability and support by multiple programming languages.

### 1.1. Aims and Objectives

The successful implementation of Behaviour Trees relies on two main components – visualization and testing. The first step of the implementation process is creating a BT model – the logic behind what the system should do in given situations. The next step is creating the AI component for the designed tree - this step requires extensive programming time and skills. The final step is testing the BT logic – does the system do what it is supposed to?

The main objective of this project is to ease the method of implementation by generating the AI component for Behaviour Trees from models. That component is the most exhausting and can only get done by professionals. The project aims to provide a graphical user interface that allows visual designing of BT models and their extraction. The goal is to pass these models through a system that would generate running code for the modeled tree. The project also aims to provide a task-oriented dialogue system - a chatbot handling customer queries. The chatbot will allow functional testing of the generated code.

- ❖ Aim is to separate designing a dialogue flow, from creation of underpinning AI component
- ❖ Tool should allow rapid authoring of behaviour trees from a palette of existing natural language processing components
- ❖ Tool should enable visual testing of the tree logic (traversing)
- ❖ 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

## 1.2. Ethical Review

Based on the information provided in the Ethical Review checklist, this project is considered Low risk. The project is a software tool that is intended to have user testing for its use and functionalities. However, the system will not gather any personal data throughout the development process or the intended use of the application.

As further described in the Research section, chatbots could face ethical issues when it comes to the user's privacy and safety. The chatbot system of this project doesn't fall under high-risk categories. It is merely a testing tool for the dialogue flow of behaviour trees. There is, however, an encryption system for hashing the conversations straight after completing the tree cycle. The encryption system ensures that conversions with the chatbot will be safe to store as testing data in the future.

## 2.Literature Review

### 2.1. Behaviour Tree Origin

A Behaviour Tree (BT) is a hierarchical tree data structure that replicates the decision-making process of humans with the help of predefined actions and conditions. It originated from the field of Gaming Technologies and got invented as a tool to allow the modularization of AI in games. Colledanchise, Marzinotto, and Ögren (2014) call it – “a more modular and flexible alternative to Finite State Machines”.

### 2.2. Finite State Machines (FSM)

Finite State Machines (FSM) is the first AI technique to address the problem of modelling the process of human thought (Amal, Joe, and Raylene, 2019). FSMs have a start state and a control unit – a finite number of internal states which process a set alphabet of symbols for input and output. Although FSMs can switch between internal states in some predefined order, they can be in only one at a time and are incapable of remembering state transitions. Chellapilla and Czarnecki (1999) address these issues as the cause for FSMs to scale poorly.



### 2.3. Hierarchical State Machines (HSM)

A more flexible version of FSM is the (HSM) Hierarchical State Machines (Yannakakis, 2000). HSMs are finite state machines whose states themselves can be other state machines. The hierarchy splits the states into two types – common messages handlers and states executing specific functions. The message handlers are at a higher level in the structure and get inherited by the lower-level states, which perform the state-specific functions. However, with the rising demands on AI agent complexity, programmers find FSMs “poorly scaled and difficult to extend, adapt, and reuse” (Iovino et al., 2020).

### 2.4. Behaviour Trees

Behaviour Trees are defined as an improvement of Hierarchical State Machines (Colledanchise and Ögren, 2014), combining them with scheduling, planning, and action-execution. The behaviour of the system is what actions to take and when. It is planned and visualized through a BT model. In comparison with Hierarchical State Machines, BTs are considered scalable, simple to implement, capable of handling complexity, and modular to improve reusability (Lim, Baumgarten, and Colton, 2010). These properties are considered crucial in many applications outside of gaming, which has led to BTs gaining popularity in branches of AI, Robotics, and Computer Science.

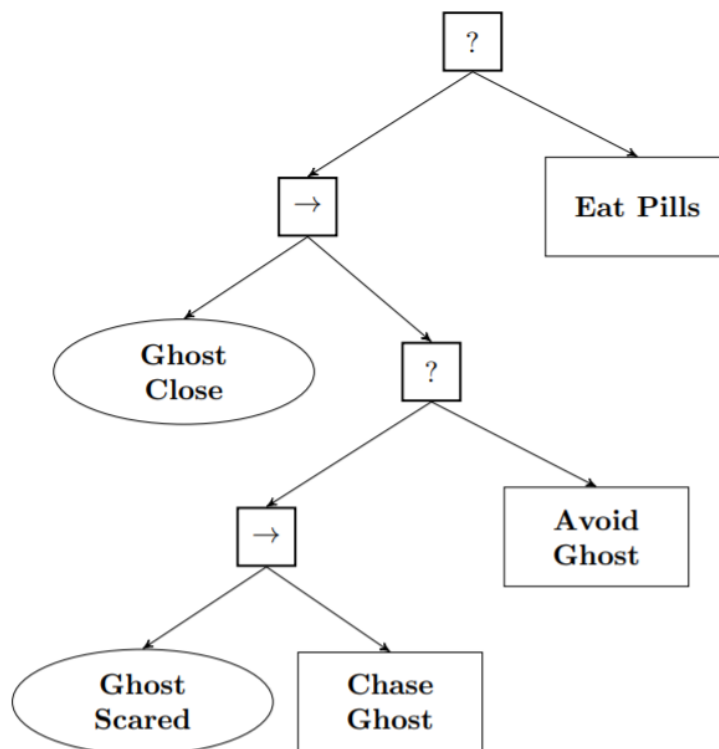


Figure 1: A behaviour tree model for Pacman's combative behaviour (Colledanchise and Ögren, 2018)

Figure 1 displays the behaviour tree behind the decision-making process of Pacman. Eat pills until a ghost approaches. If the ghost is scared - chase it. Otherwise, run away. Similar behaviour trees can

be seen in many video games' non-player characters, as that is what the data structure gets primarily used for.

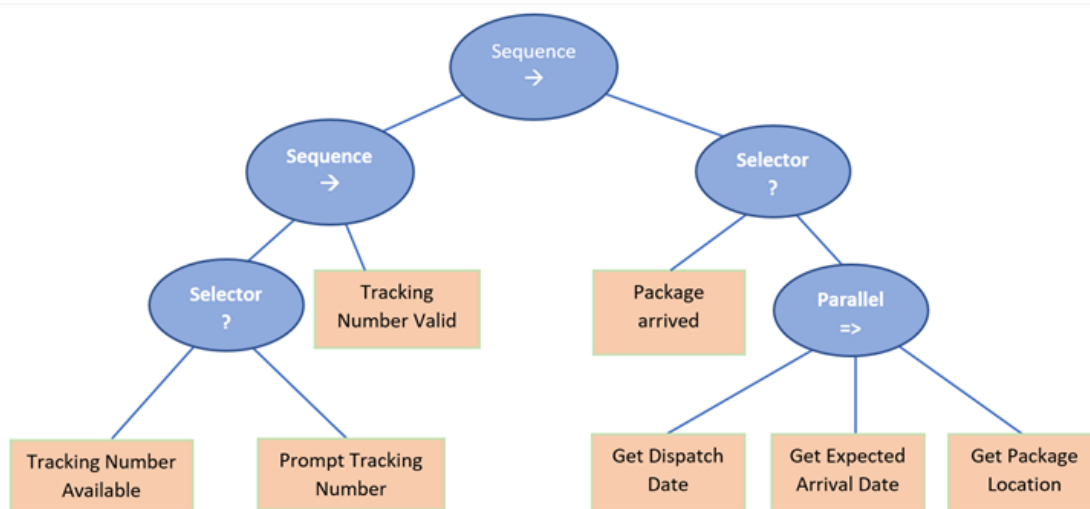


Figure 2: BT model for parcel tracking dialogue

Figure 2 demonstrates the use of behaviour trees for dialogue management, which is relatively new. The blue nodes on the graph represent the hierarchy of the tree. The orange nodes describe the functions to be executed.

The hierarchy of behaviour trees gets constructed from condition and action nodes (Figure 1, 2). The condition nodes look for a specific condition to be met and the action nodes complete a given task when reached. Nirwan (2020) describes condition nodes as "control flow" (Figure 2, blue nodes) and action nodes as "execution" (Figure 2, orange nodes).

## 2.5. Behaviour Trees and Task-based Dialogue Systems

Behaviour Trees can be used to enhance task-based dialogue systems by interpreting the flow of conversations with the control flow and reacting to the user's actions when executing leaf nodes. They model the structure of switching between multiple tasks in autonomous agents. They are described (Colledanchise and Ögren, 2018) as a very efficient way of creating sophisticated programs that are both modular and reactive.

## 2.6. Behaviour Tree Nodes

The control flow nodes in the BT model, Figure 1, are sequence, selector, and parallel. What all the nodes have in common (Simpson, 2014) is that they can be in one out of three states: success, failure, or still running. The traversing of the tree is complete when the root node returns success – meaning that the necessary leaf nodes functions have been executed.

Sequence nodes (→) return success if all their children sequentially do. Selectors (?) return success when either of their children does. Parallel nodes (⇒) execute all their children simultaneously and

only return success if all children are successful. The control flow enables the modularity properties of the Behaviour Tree, as it separates individual states as leaf nodes. Every state operates as a separate module, and their switching does not alter the hierarchy of the tree.

The execution consists of functions that get called when traversing the tree structure. In the case of Figure 1, these functions are for a tracking delivery services system. The user would get prompted to provide a valid tracking number, and the system will answer their query from information in its database. Until a valid tracking number gets provided, the system will not leave the state of prompting one.

## 2.7. Implementation

The successful implementation of Behaviour Trees relies on two crucial components – visualization and testing. As BTs get based on human decision-making logic – how a human thinks the system should react in different situations, creating models is quite straightforward. When creating a visual model of a BT system, there is no need for programming skills. Modelling can be as simplified as using a pen and paper. However, there are various tools available that could make the modelling process more enjoyable.

## 2.8. Tools

Behaviour Trees (BT) include an element of planning and strategy hence modelling and testing of the models are crucial for successful implementation. Modelling BTs is intuitive and straightforward as it relies more on logic than skills. Although the modelling doesn't require the use of software, there are open-source tools that could make the process more convenient. One example is VS Code Behaviour Tree Editor (Dolejší, J., 2022).

### 2.8.1. Behaviour Tree Editor

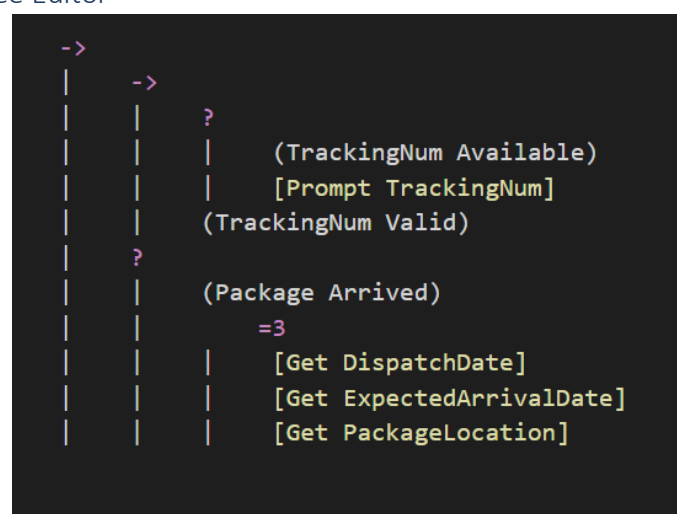


Figure 3: Syntax of VS Code Behaviour Tree Editor – the language suggested by Abad, D. (2021)

The BT Editor visualizes tree models using the language (shown in Figure 3) suggested by Abad, D. (2021). The language consists of sequence, selector, parallel, action, and condition nodes. The example BT model is for Tracking Deliveries services.

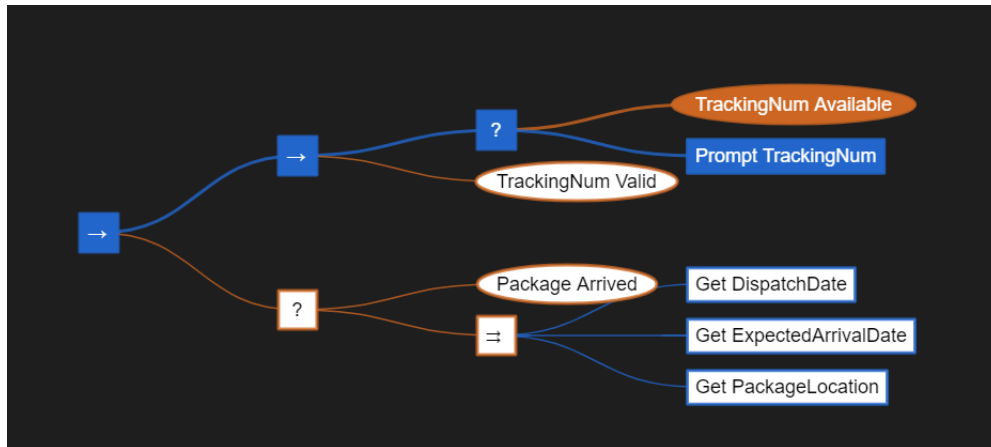


Figure 4: Visualization of BT model using VS Code Behaviour Tree Editor

Figure 4 displays the visualization of BT models using VS Code Behaviour Tree Editor. Along with visual representation, the tool also allows testing of the current model. The user can test the logic of their model by altering nodes' status upon selection. Green nodes are successful, blue nodes are still in progress, and orange - failed.

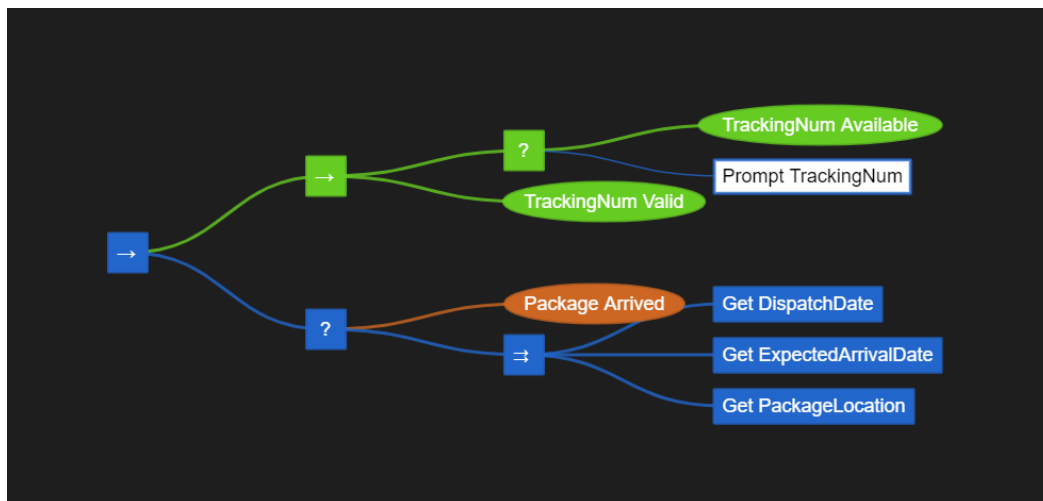


Figure 5: Testing BT model using VS Code Behaviour Tree Editor

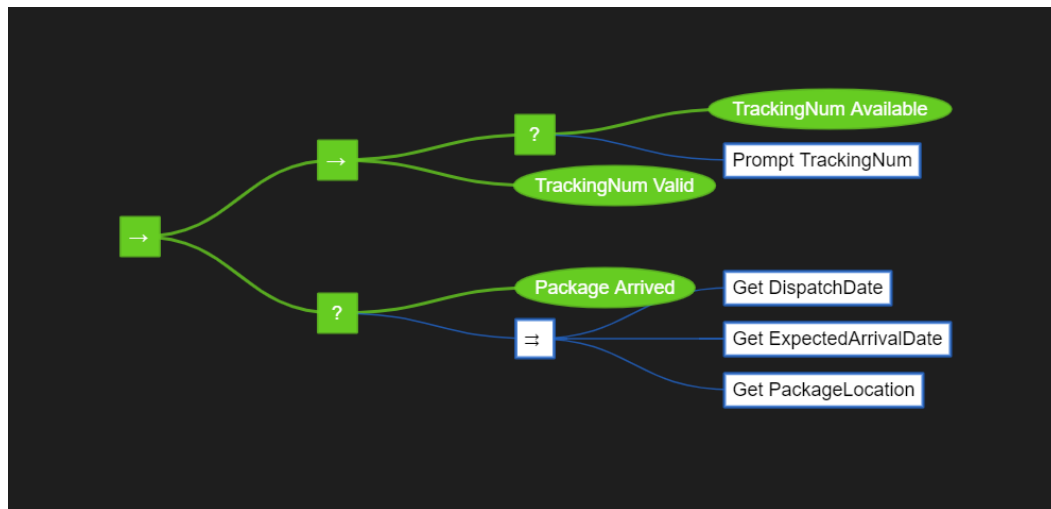


Figure 6: Complete tree cycle of BT model using VS Code Behaviour Tree Editor

Figure 5 shows how the tool reacts to changing nodes' status. The right branch is set to success (green), whereas the left branch is still running (blue). As the root node is a sequence node – all children must sequentially return success, and the tree cycle is not complete. Figure 6 demonstrates a complete tree cycle for the given model. Although quite convenient for personal use, the tool has limitations. There is an option for exporting the tree model to a JSON format, but it does not give the JSON file enough data to make it compatible with other sources.

### 2.8.2. D3JS Tree Editor

Another example of a BT modelling tool is D3js Tree Editor (Feuer, A., 2017). This tool allows users to display and alter tree models generated from a JSON file. Figure 7 shows the tool with its original model.

## D3JS Tree Visualizer

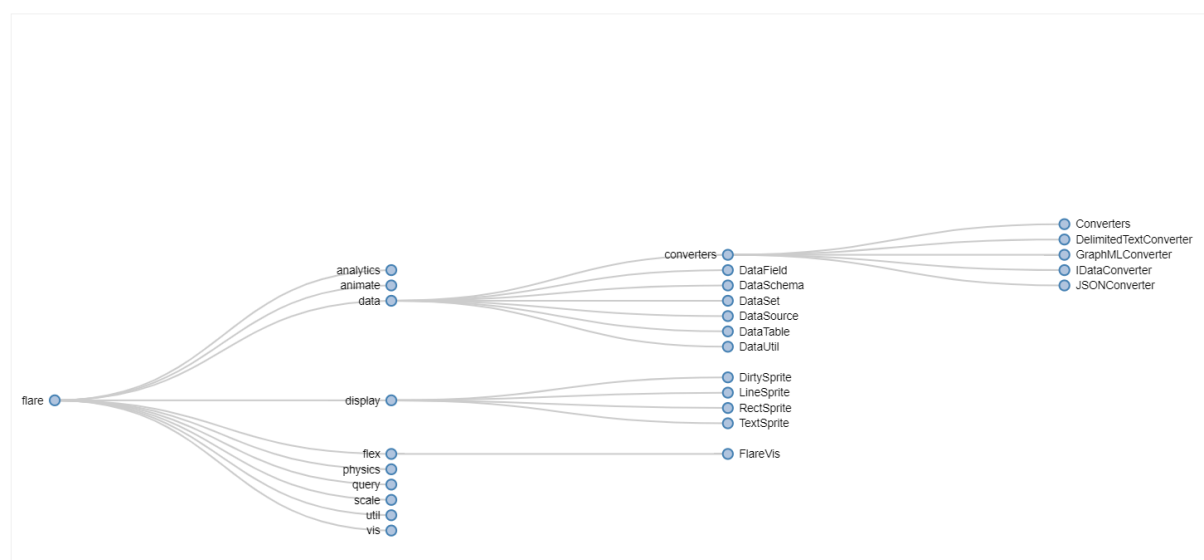


Figure 7: BT model using d3js Tree Editor



Figure 8: Tracking Delivery Services model with d3js Tree Editor

The user can alter the original tree model, by deleting, renaming, or adding new nodes. Figure 8 visualizes the previously used example of BT for tracking delivery services. Nodes can be switched around by drag-and-drop, and there is also an option for collapsing or expanding node's children. The strength of this tool is the means of generating models – using JSON files. This functionality makes it compatible with different programs and gives it great potential.

D3js Tree Editor uses Data-Driven-Documents (d3) Javascript library. The library allows binding data from any format to a Document Object Model (DOM), applying data-driven transformations to the document (Bostock, M., 2021). D3 library enables the D3js Tree Editor to visualize tree models from JSON files. Figure 9 displays the JSON file generating the model example.

The tool comes very close to the idea for the graphical user interface. It allows rapid authoring of BTs models but cannot save or upload them. The D3js Tree Editor also does not distinguish Behaviour Trees from any other tree structure. An intent is to use the tool as a base for the BT modeler system and build a GUI that enables the design, upload, and downloading of BT models.

```

{
  "name": "Sequence",
  "children": [
    {
      "name": "Sequence",
      "children": [
        {
          "name": "Selector",
          "children": [
            {
              "name": "Prompt TrackingNumber",
              "children": [
                {
                  "name": "TrackingNum Available"
                }
              ]
            },
            {
              "name": "TrackingNum Valid"
            }
          ]
        },
        {
          "name": "Selector",
          "children": [
            {
              "name": "Parallel",
              "children": [
                {
                  "name": "Get DispatchDate"
                },
                {
                  "name": "Get ExpectedArrivalDate"
                },
                {
                  "name": "Get PackageLocation"
                },
                {
                  "name": "Package Arrived"
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}

```

Figure 9: JSON file for generating the example model on Figure 8

### 2.8.3. Collapsible Tree Search

## Collapsible Tree Search

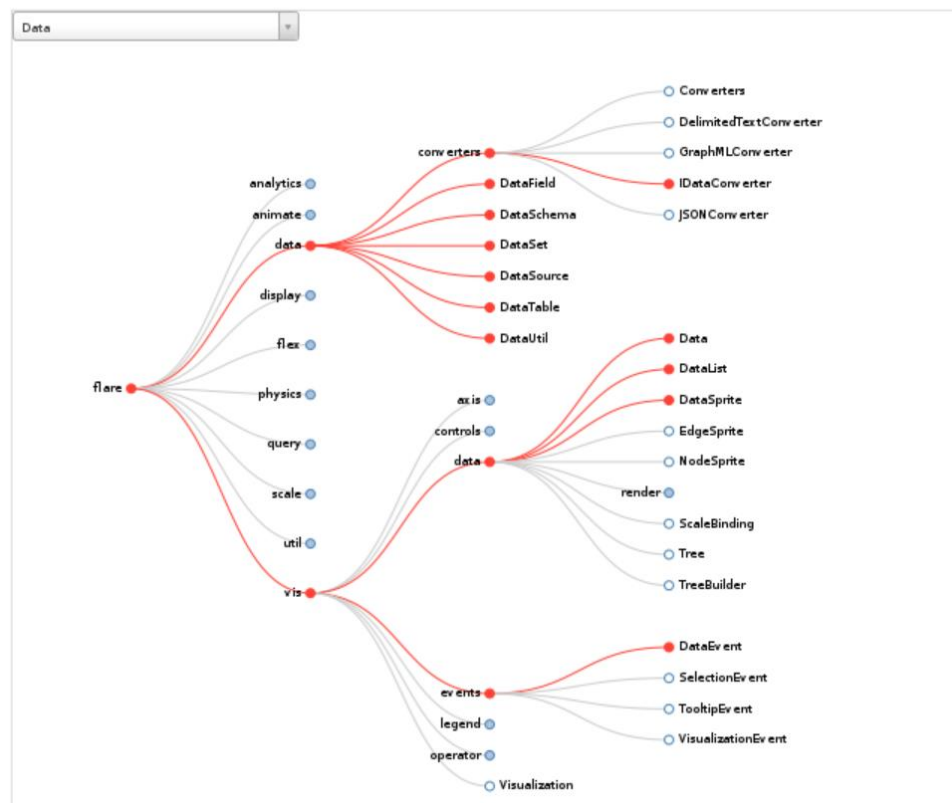


Figure 10: A tree model using Collapsible Tree Search

D3 Tree Editor is based on Collapsible Tree Search tool (Brockmann, 2014). Figure 10 shows a model visualized by the tool and some data search. The model of the tree looks very similar to D3js Tree Editor, as they are both based on the Reingold-Tilford Algorithm – an algorithm for drawing tree structures. The approach for the Collapsible Tree Structure raises issues with the search when the number of nodes goes over 300, says Brockmann (2014) himself. D3js Tree Editor does not have the search functionality, but it allows the user to change the model visually and not through coding. It requires less skills to model with and gives the user full control over the model.

### 2.8.4. Conclusion on Tools

The tools given as examples display a colourful variation of methods for designing tree structures. The tool that is closest to the type of tree modeler intended for this project is the D3JS Tree Editor by Adam Feuer (2017). It allows rapid authoring of BTs models but cannot save or upload them through the GUI – not very user-friendly. The D3js Tree Editor also does not distinguish Behaviour Trees from any other tree structure, which will need modification if used as intended.

## 2.9. Natural Language Processing

The problem of machines understanding and processing human language is one of the earliest problems of Artificial Intelligence. It emerged from the idea of using computers to translate different languages and evolved into real-time human-machine communication (Anyoha, 2017).

The computational techniques developed to solve that problem are known as Natural Language Processing (NLP). One early application of NLP is the rule-based chatbot ELIZA which made human language conversation with a computer possible (Weizenbaum, 1966; pp.36–45). ELIZA still gets mentioned by researchers as the start of systems such as Siri (Coheur, 2020).

NLP typically aims to relate written text and abstract meanings. It can also interpret the meaning of texts containing lexical errors. However, human language can be ambiguous, especially when using figures of speech such as sarcasm or irony.

Analyzing and interpreting such figures of speech is the main issue facing NLP (Levine, 2022). Although understandable as even humans can misinterpret the context of such conversations. NLP is being commercially exploited by most major tech companies – an example is the Microsoft LUIS and QnA systems (Microsoft, no date). The project system aims to be future-proofed and platform agnostic, meaning that it will allow the use of commercial packages as the leaf functions of the tree. These packages include but are not limited to chatbot languages such as AIML or NLTK.

## 2.10. Dialogue Management Systems

Dialogue Management Systems are natural language components which enable conversations between machine and humans. Jiang Zhao, Ling Li, and Lin (2019) describe them as - “the bridge between human and intelligent machine”. The method of communication started as exchanging text messages and has now evolved to speech, with the presence of Spoken Dialogue Systems. Such systems typically provide some client service and are used for human-machine interaction.

In terms of application, Dialogue Systems divide into two main types: task-based and non-task-based. The non-task-based systems are known as “chattering systems” (Jokinen and McTear, 2009). Their purpose is entertainment of the user with generic chit chats without having to complete a given task. In contrast, task-based systems aid the user in completing certain tasks by acquiring data for a specific domain. Task-based dialogue systems include pipeline and end-to-end methods.

### 2.10.1. Traditional Pipeline

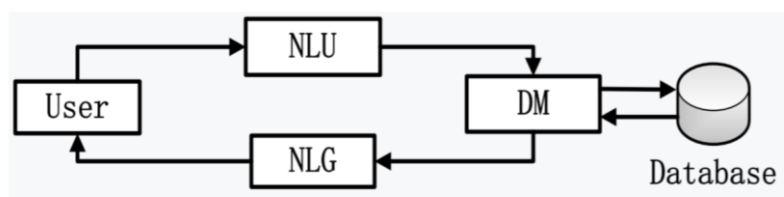


Figure 11: Traditional Pipeline for Task-oriented Systems. (Jiang Zhao, Ling Li, and Lin, 2019)



As shown in Figure 11, Pipeline methods consist of three modules - Natural Language Understanding (NLU), Dialogue Management (DM), and Natural Language Generating (NLG). When the user sends a message, the NLU interprets and translates it into an internal representation. Afterwards, the Dialogue Manager takes a series of actions based on the strategy through that representation. The actions taken are the response to the user's message, and to be delivered, they first need to be translated back into natural language by the NLG. Although modules have clearly defined problems, they are strongly dependent on each other - if one of them causes an error, they will all malfunction.

### 2.10.2. End-to-end method

End-to-end methods use Neural Networks because of their powerful distributed representation in deep learning (Wen et al., 2017). The idea is to design and run a complete model, collect its output, and feed it back to the network using backpropagation. Bordes, Boureau, and Weston (2017) recognize that this method raises issues – uncertainty in the user's input is undetected, and the neural network cannot retrieve a knowledge base. Both methods have their advantages and disadvantages, but recent study (Zhang, Z. et al., 2020) shows that end-to-end system modeling is getting more attention from researchers.

## 2.11. Dialogue Manager

The Dialogue Manager is the central component of dialogue systems (McTear, 2010). Its purpose is to collect interpreted input, interact with external sources, and produce output messages for the user. The dialogue management process splits into two systems: Dialogue Modeling and Dialogue Control. The Dialogue Modeling consists of dialogue state information -conversations history, and a task record – the available and the still to be collected information, formulated by a template. The system determines the context of the dialogue state necessary for the Dialogue Control.

### 2.11.1. Dialogue Control

The Dialogue Control system determines what to do next in the current dialogue state. For a more accurate decision, the system could prompt the user for more input or ask for clarification on a previous input or output. The decision can be pre-scripted (factor-based choices) or dynamic – based on reasoning. When Pre-scripted, the decision-making can be predicted at the design stage and hardcoded into the Dialogue Control. Dynamic decision-making gets based on reasoning about the current dialogue state using evidence from a combination of domain and dialogue knowledge sources.

#### 2.11.1.1. Dimensions of Dialogue Control

In terms of dimensions, dialogue control has a dialogue initiative and a dialogue flow. As the names suggest, the dialogue initiative is who controls the dialogue, and the dialogue flow is the means of implementation. There are three types of initiatives: system-directed, user-directed, and mixed-initiative.

##### User-directed

In user-directed dialogue systems, the user controls the dialogue. Such systems are or resemble question-answering (QA) systems. The user's input is not restricted, and these systems require more advanced language processing capabilities.

### Mixed initiative

Mixed-initiative dialogues allow both the system and the user to control the context. The system initializes the topic, but the user can interrupt and change it. Such systems also require more advanced language processing abilities to adapt to the change in the dialogue context.

### System-directed

System-directed dialogue prompts the user to answer questions to collect data for a specific domain. These systems typically provide some customer service, and they usually get used for commercial systems. Bernsen, Dybkjær, and Dybkjær (2016) consider them “functionally adequate”, although they also point out the existing vocabulary limitations. The advantage of system-directed dialogue is that the scope of the user’s answers is restricted, as the dialogue flow gets determined in advance.

#### 2.11.1.2. Dialogue flow implementation

The dialogue flow gets implemented using agent-based methods, forms and slots filling, or dialogue scripts. Dialogue scripts, also known as finite-state dialogue control, predict the dialogue states and transitions. They define the actions to be taken according to the current state. This method is only suitable for system-directed dialogues, as it does not allow the user to take the initiative.

The form-filling dialogue control consists of slots that get filled by gathering the user’s input and retrieving data from the database. This method is considered more flexible as it allows the user to change the order in which the slots get filled. It is suitable for user-directed but insufficient for true mixed-initiative dialogues.

### 2.12. Chatbots

Chatbot software is the application of dialogue management systems. Daniel and Martin (2021) define chatbots as the simplest kind of dialogue systems that can replicate the unstructured ‘chats’ naturally occurring in human-human conversations.

Chatbots enable interaction via messages and are mainly used for handling customer queries and answering frequently asked questions (FAQ). There are two types of chatbots – rule-based, where the software gets bounded to a pre-scripted dialogue control, and AI conversational agents. Rule-based chatbots trigger pre-set actions upon keywords, and AI chatbots use Machine Learning for dialogue control. Both methods have brought forward efficient applications for entertainment and business purposes.

#### 2.12.1. Chatbots and Business

Chatbots gain more and more popularity in the business sphere for the various advantages they bring. One chatbot system could provide non-stop online support and handle hundreds of clients’ queries a day. Wilson and Daugherty (2019) demonstrate the importance of chatbots for businesses in their review. Their example is the chatbot Aida which handles queries of millions of customers for a major Swedish bank. The chatbot has proved efficient for resolving over 70% of the customer queries. When unable to resolve the issue, Aida would connect the client to a human operator and monitor their conversation for future reference on how to solve such a problem.

### 2.12.2. Ethical issues

There are a few ethical issues haunting chatbot software. Some chatbots, for example, could be unethical when working for a service provider and not the user. Such chatbots are mainly online shop assistants that instead of focusing on making the users happy, would focus on convincing them to spend more money on the service. Chatbots are the connection between business and customers and should always aim for the customers' well-being and content.

Murtarelli, Gregory, and Romenti (2020) share these views and state that chatbots must be kind and encouraging towards their customers. The impact of an untrustful system could be devastating for businesses as it will cause a significant loss of customers.

Other issues arising consider privacy and protection of personal data. There are chatbots that collect personal data without intending to protect it or even – without any need for it. Ethical chatbots must keep customers' conversations private, and if storing any personal data, it must be encrypted first. Amditis, J. (2017) addresses these issues and provides examples of chatbots behaving unethically and their impact on users.

## 2. Requirements

### 2.1. Use Case Diagrams

#### 2.1.1. Figure 12: Use Case for behaviour trees modeler system

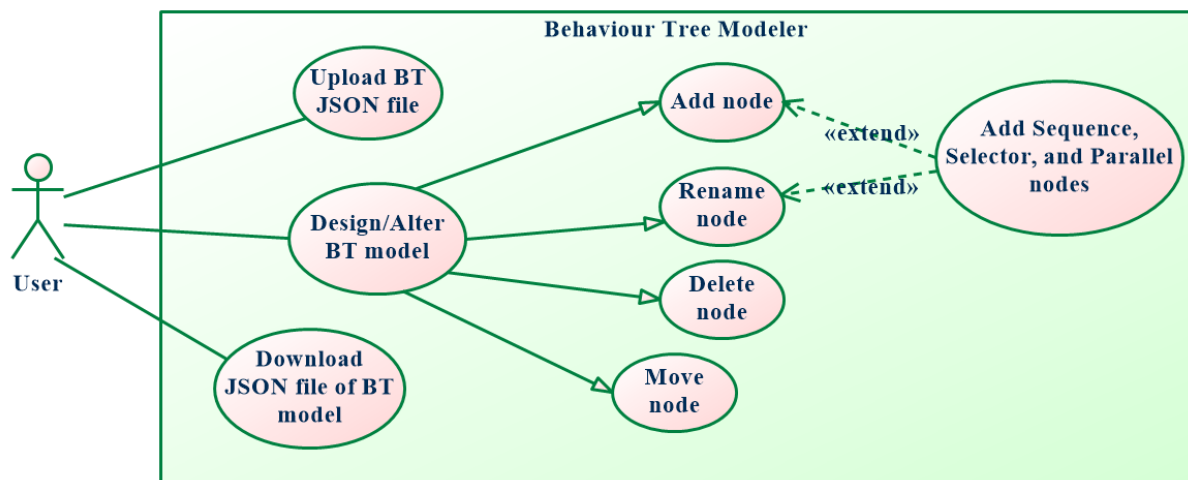


Figure 12: Use Case for behaviour trees modeler system

2.1.2. Figure 13: Use case for generating running code system

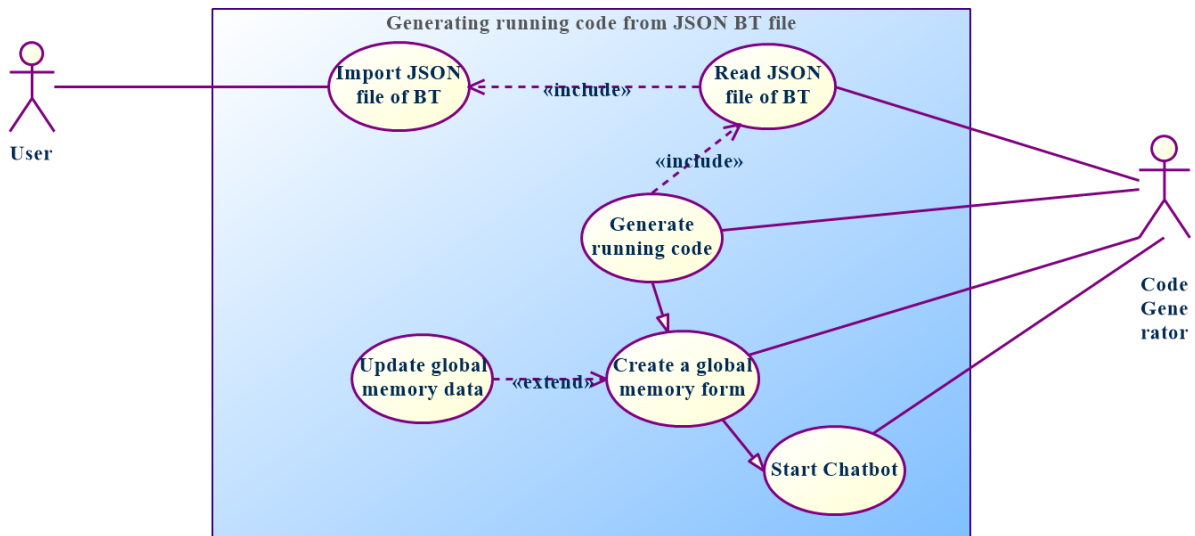


Figure 13: Use case for generating running code system

2.1.3. Figure 14: Use case for requesting service from the chatbot system

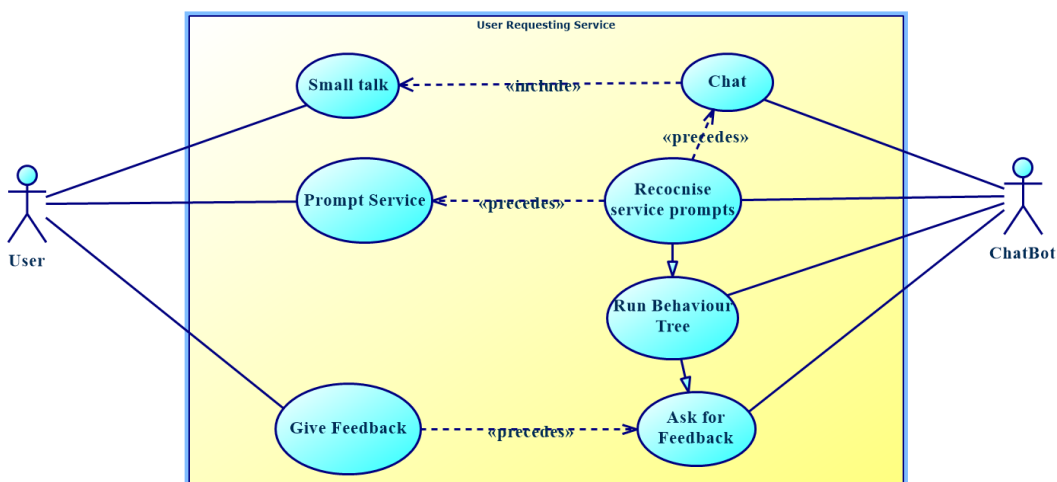


Figure 14: Use case for requesting service from the chatbot system

## 2.2. Non-functional Requirements

The following non-functional requirements were extracted from aims and objectives, and research:

- ❖ The system should provide a simple method for designing Behaviour Tree models
- ❖ A User-friendly Graphical User Interface and data representation
- ❖ There should be a fast input method, such as drag-and-drop, for altering the models
- ❖ An option for extracting/downloading the modeled tree

- ❖ An option for loading a Behaviour Tree back into the GUI for alteration
- ❖ A simple method of parsing a BT model and transforming it into executable code
- ❖ Limited altering necessary for changing between models for different services
- ❖ A form of global/blackboard memory for recording the tree progress
- ❖ A form of displaying the graphical representation of the generated tree class
- ❖ A form of displaying the tree cycle or visited nodes
- ❖ The project should provide a dialogue system for testing of the generated BTs
- ❖ The dialogue system could be in the form of a rule-based chatbot with pre-scripted dialogue – as suitable for task-oriented dialogues
- ❖ The chatbot system should facilitate understanding of the dialogue flow
- ❖ Limit the scope of the chatbot-user conversations to - the task at hand and small talk
- ❖ A feedback mechanism provided by the chatbot
- ❖ A learning method to improve the bot's appropriate recognition of its scripted topics
- ❖ Standard encryption of user conversations
- ❖ Decryption algorithm for maintenance checks on the bot's conversations

Further analysis of these non-functional requirements will draw out the formal functional requirements. The functional requirements will provide more details on the system's use and functionalities. The main components for building the set system will get constructed in the functional requirements section – the next step to design and implementation.

### 2.3. Functional Requirements

The functional requirements are displayed as table forms, including the requirements' description, prioritization using the MoSCoW scheme, and justification for their given priority. The MoSCoW scheme is: (M) Must have, (S) Should have, (C) Could have, and (W) will not have at this stage. The requirements specified as fundamental features are extracted from the aims and objectives of the project.

#### 2.3.1. Table 1: Functional Requirements for the BT modeler GUI

The requirements' names are an abbreviation of Behaviour Tree Modeler (BTM).

Requirement	Description	Priority	Priority Justification
All1.	The ability of all systems to interact with JSON files	M	Fundamental feature. The project requires data exchange between different systems.
BTM1.	A system that reads JSON files of a Behaviour Tree	M	Fundamental component
BTM2.	The system to generate graphical models of the tree	M	Fundamental feature

BTM3.	The ability for the user to add new nodes	M	Fundamental feature
BTM4.	The ability for the user to rename nodes	M	Fundamental feature
BTM5.	The ability for the user to delete existing nodes	M	Fundamental feature
BTM6.	The ability for the user to create sequence and selector nodes	M	Fundamental feature
BTM7.	The ability for the user to create parallel nodes	S	Parallel nodes are not as often used, but they are part of the structural nodes for Behaviour Trees
BTM8.	The ability for the user to download the BT model as JSON file	M	Fundamental feature
BTM9.	The ability for the user to upload a chosen BT model for alteration	S	There is a default BT model provided by the system. However, the design would be more user-friendly with the option for continuing the work on a chosen model.
BTM10.	A fast-input method, such as drag-and-drop	S	A convenient way for the user to make changes to an existing BT model using the graphical tool.
BTM11.	The ability for the user to graphically see the BT flow during the modelling stage	W	There is a visual representation of the behaviour tree's structure and leaf nodes. However, visualization could be improved by also displaying how the tree would flow when certain nodes are successful.

2.3.2. Table 2: Functional Requirements for the Code Generating system

The names of the requirements are an abbreviation of Behaviour Tree Code Generator (BTCG)

Requirement	Description	Priority	Priority Justification
BTCG1.	A system that reads JSON files of BTs	M	Fundamental feature
BTCG2.	The system to generate running code for the JSON tree	M	Fundamental feature

BTCG3.	The system to recognise sequence and selector nodes	M	Fundamental feature
BTCG4.	The system to recognise parallel nodes	S	Parallel nodes are not as often used, but they are part of the structural nodes for Behaviour Trees
BTCG5.	The system to work with a form of blackboard/global memory	M	Fundamental feature
BTCG6.	Automated generation of the Behaviour Tree structure	M	Fundamental feature
BTCG7.	The ability for the user to see the graphical representation of the generated BT instance	M	Fundamental feature
BTCG8.	The ability for the user to see the tree cycle and visited nodes	M	Fundamental feature

2.3.3. Table 3: Functional Requirements for the Dialogue Management system

The names of the requirements are an abbreviation of Dialogue Management System (DMS)

Requirement	Description	Priority	Priority Justification
DMS1.	A dialogue system for functional testing of the BT model and generated code	M	Fundamental feature
DMS2.	A rule-based chatbot that adopts the behaviour tree structure	C	The project needs an interactive system for observing the BT models in action. Rule-based chatbots do well with task-oriented dialogues.
DMS3.	The ability for the chatbot to accurately interpret the context of the conversation	M	Fundamental feature
DMS4.	The ability of the chatbot to allow user-initiative within necessary limitations	S	For a more natural conversation, the chatbot should be capable of small talk. However, it is a task-oriented dialogue system,

			and the task must be the main focus.
DMS5.	The ability to gather user feedback upon task completion	S	Completing the tree cycle, does not necessarily mean that the user's query has been solved. Gathering feedback is essential for improvement.
DMS6.	A learning capability for more accurate context interpretation	S	Without a learning capability, the system will be much less convenient as it will repeat its mistakes.
DMS7.	Encryption of user-system conversations	M	Security
DMS8.	Decryption of encrypted conversations for maintenance purposes	S	Part of the chatbot's maintenance is monitoring its conversations and outcome. Ethics require encryption before storing user-machine conversations.

## 2.4. Acceptance Tests

The acceptance tests will also be provided in a table form with their ID, name of the tested requirement, description of the test, and the expected outcome.

### 2.4.1. Table 4: Acceptance tests

Test ID	Tested Requirement	Test Description	Expected Outcome
AT1.	BTM1 and BTM2	Test if the system can successfully generate graphical tree models from a JSON file	The system can import JSON BT data and display the graphical model on the modelling GUI
AT2.	BTM3	Test adding a new node to the graphical model	User to be able to add new nodes to the BT model
AT3.	BTM4	Test renaming existing nodes	User to be able to rename nodes in the BT model
AT4.	BTM5	Test deleting a node	User to be able to remove nodes from the BT model
AT5.	BTM8	Test downloading current model as a JSON file	User to be able to download/extract their BT



			model from the GUI in JSON format
AT6.	BTM9	Test uploading JSON file to the GUI	User to be able to upload a chosen BT model to the GUI from a JSON file
AT7.	BTM10	Test drag-and-drop input method for modifying the model	User to be able to alter the BT model using drag-and-drop input method
AT8.	BTCG1	Test if the code generating system can import JSON files	Code Generating system to be able to successfully import BT data through a JSON file
AT9.	BTCG2	Test if the system can generate running code from the JSON file of the BT model	System to be able to generate running code for the behaviour tree in the JSON file
AT10.	BTCG3 and BTM6	Test if the system recognises sequence and selector nodes from the BT model	System to successfully distinguish selector and sequence nodes in the generated code
AT11.	BTCG4 and BTM7	Test if the system recognises parallel nodes from the graphical model	System to successfully distinguish parallel nodes in the generated code
AT12.	BTCG5	Test if the system successfully utilizes its form of global memory	System to be able to store and update tree cycle data to a form of global memory
AT13.	BTCG6	Test if the system can automatically generate a behaviour tree structure	System capable of automatically generating BT structures from given JSON data
AT14.	BTCG7	Test the system's interpretation of the BT structure on the model	The system can provide an accurate graphical representation of the BT model
AT15.	BTCG8	Test the system's tree traversing accuracy	The tree cycle is running as intended in the model
AT16.	DMS1 and DMS2	Test if the dialogue system can adopt the BT code without conflict	System successfully running BT code from the dialogue management system
AT17.	DMS3	Test if the chatbot accurately interprets task-oriented context	System successfully recognising user intents and providing accurate responses
AT18.	DMS4	Test if the chatbot allows user-initiative of not too far off topics	System successfully recognises and handles small talk triggered by the user

AT19.	DMS5	Test if the chatbot always asks for feedback when its service has been triggered	System always asks for feedback after completing the behaviour tree cycle
AT20.	DMS6	Test the chatbot's learning functionality	System capable of learning from user feedback
AT21.	DMS7	Test encryption	Conversations are instantly hashed after user says goodbye
AT22.	DMS8	Test decryption system	Conversations are successfully decrypted, and the plain text is identical with the original conversation
AT23.	All1	Test the compatibility of all systems to work with JSON files	All systems interexchange data from JSON formatting without conflicts

## 4.Design

### 4.1. User stories

The next step after finalizing the project requirements is extracting user stories from them. This step will further describe the expected functionality of the project. The user stories are in the form of tables and they are following the given format:

As an {actor}, I need {to be able to do something}, so that {some reason}.

Each story has a unique ID and only concerns one actor at a time. User stories describe what functionalities the actors need and a justification or the reason why. Stories are related to one or more Acceptance Tests - further justification of the need for the specified tests.

4.1.1. Table 5: User stories

User Story ID	Story		User Acceptance tests
US1.	As a	user,	AT1 and AT23
	I need	a GUI that can show me graphical models of JSON behaviour trees,	
	so that	I can see its visual representation.	

US2.	As a	user,	AT2, AT3, AT4, and AT7
	I need	the GUI to allow me to alter the displayed model,	
	so that	I can create my own behaviour tree model.	
US3.	As a	user,	AT2, AT3, AT10, and AT11
	I need	to be able to add sequence, selector, and parallel nodes	
	so that	my tree model is distinguished as a behaviour tree model.	
US4.	As a	user,	AT5 and A23
	I need	to be able to download my graphical model,	
	so that	I can load it into the behaviour tree code generating system.	
US5.	As a	user,	AT6
	I need	to be able to upload a chosen BT model,	
	so that	I can make quick alterations.	
US6.	As a	non-AI expert,	AT8, AT9, AT13, and AT23
	I need	to get a running code for my behaviour tree model from the code generating system,	
	so that	I don't have to write the code myself.	
US7.	As a	non-AI expert,	A10, A11, A12, A14, and A15
	I need	to be able to see a visual representation of the tree traversing,	
	so that	I can assess if the flow is the way I intended.	
US8.	As a	non-AI expert,	AT17
	I need	To be able to interact with a dialogue management system,	
	so that	I can test the dialogue flow with the behaviour tree	

US9.	As a	user/client,	AT18
	I need	to be able to introduce conversation topics	
	so that	I can get straight to the point of what I need help with.	
US10.	As a	user/client,	AT21
	I need	The conversations with the chat system to be secure	
	so that	My privacy and personal data are protected	
US11.	As a	user/maintenance,	AT19
	I need	the chatbot system to ask users/clients for feedback,	
	so that	I can monitor how well the system handles customers queries and jump in the chat when needed.	
US12.	As a	user/maintenance,	AT20
	I need	The chatbot system to have a form of learning from user feedback,	
	so that	I don't have to do it manually.	
US13.	As a	User/maintenance,	AT22
	I need	To be able to decrypt hashed user-chatbot conversations,	
	so that	I can monitor the efficiency of the chatbot system.	

## 4.2. System Architecture

From the User stories and Requirements sections, the formulated design of the project consists of two main systems and three sub-systems. Figure 15 displays all project components and their relations. The decryption program has a dashed outline because it does not partake in the main run – it needs to be run separately. The boxes in yellow represent the files for exchanging data between the programs.

The boxes in green are the main systems – a BT modelling GUI and a code generating system. The boxes in blue are the sub-systems – an interactive dialogue management system (chatbot) and programs for encrypting and decrypting conversations. The dialogue management system is considered a sub-system because its purpose is to test how well the generated code works.

#### 4.2.1. Train of thought

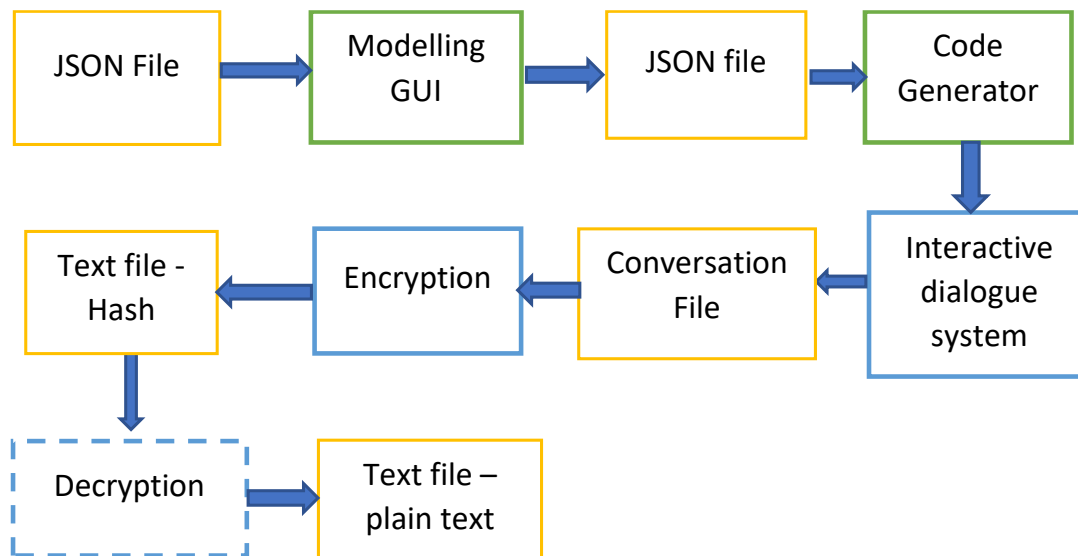


Figure 15: Train of thought of the project system

#### 4.2.2. Flowchart Diagrams

The flowchart diagrams below (Figure 16, 17) visualize the running process of all system components with their relations. The modelling GUI requires a default JSON file for the initial display of the BT graphical model. The code generating system also requires a JSON file with a BT model that would be the source for generating an executable code for the specified tree. The dialogue management system exists to adopt the BT code and run the tree when triggered by the user. After completing the tree cycle, the system would ask for feedback and encrypt the conversation.

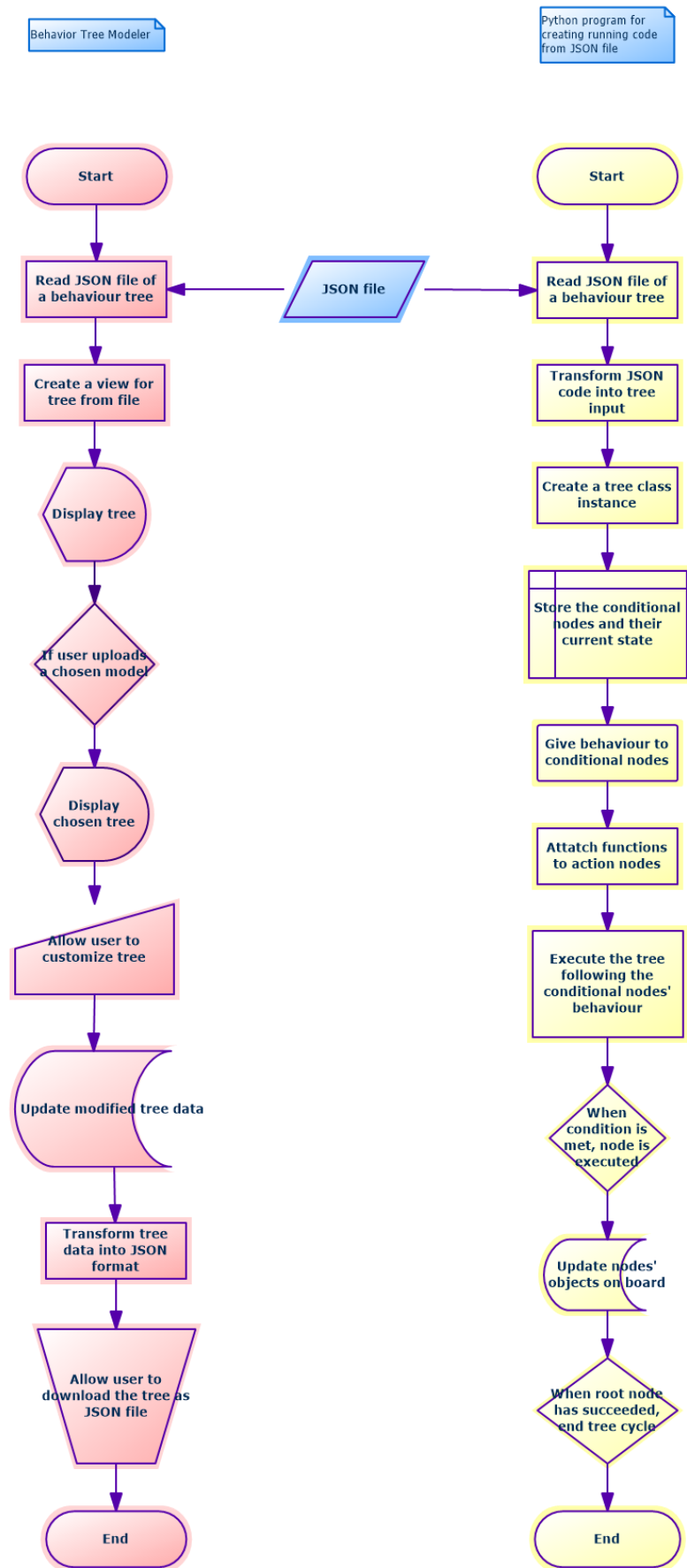


Figure 16: Flowchart for BT modelling GUI and BT code generating system

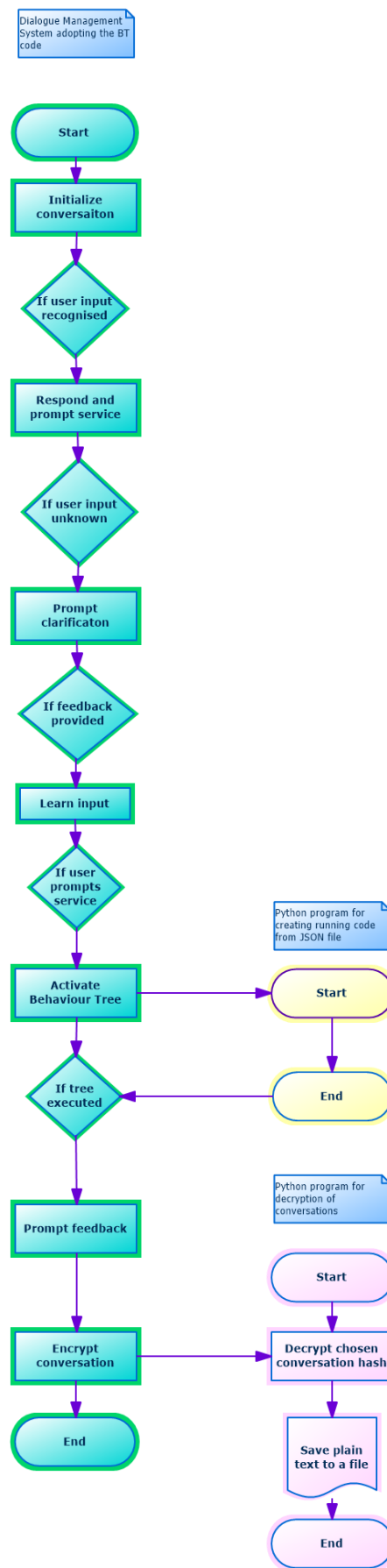


Figure 15

Figure 17:Flowchart diagram for code generating and dialogue management systems

### 4.3. User Interface Design

As mentioned in *Research(Tools)*, there is an existing system that is very close to the idea for the BT modeller. The tool, D3JS Tree Editor (Adam Feuer, 2017), provides a GUI in the form of a webpage. The design provided includes displaying a tree structure from a JSON file using the Reingold-Tilford algorithm. The tool enables users to make modifications to the default tree by adding, renaming, and deleting nodes. Users can also drag and drop existing nodes to different positions, zoom in and out of the model, and collapse/expand the node's children.

Essentially, what must be added to the existing system is - a functionality for downloading and uploading JSON files of tree models and a way to visualize the tree structure as a behaviour tree. There is also room for improvement on the user interface for a more user-friendly design. Nodes get sorted alphabetically, and having complete control over the flow of the tree is not possible. A reset button could be convenient for reloading the initial model if something goes wrong during design.

Figure 18 structures the design from the required improvements listed above. The visual representation of the trees could use more contrast. Nodes could be in different colours depending on whether they are structural nodes (sequence, selector, parallel) or action nodes (leaf nodes). Change of background and tree model colours could aid for easier reading and interpretation of the model.

#### 4.3.1. Figure 18: GUI Design

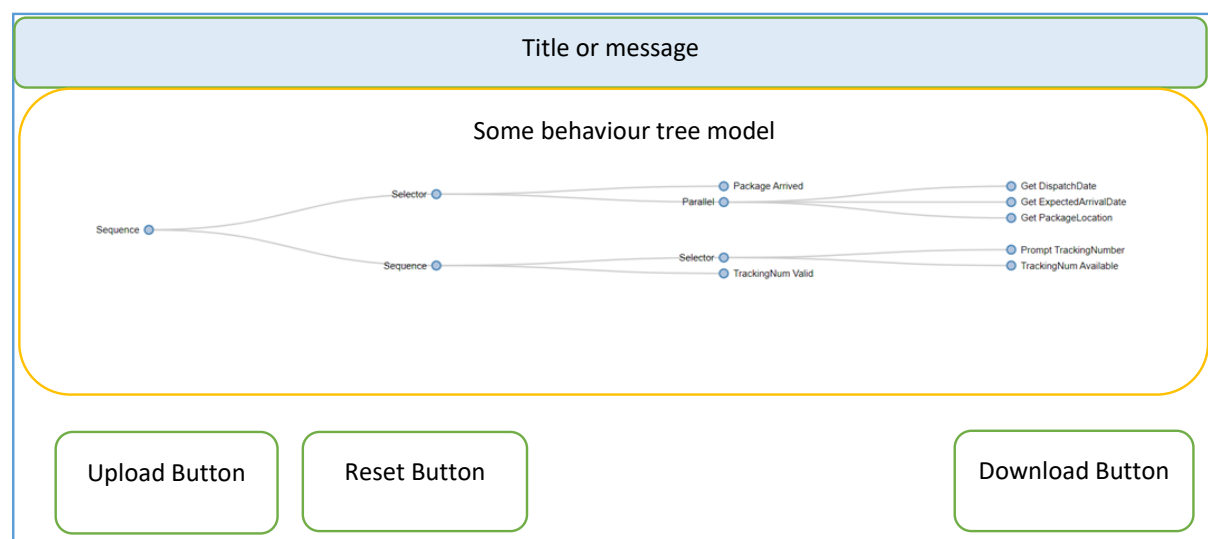


Figure 18: GUI Design



## 4.4. Pseudocodes

### 4.4.1. BT running code generating system

1. Read JSON data as a JSON string
2. Set tree data by altering the JSON string into an Anytree code
3. Generate running code from the tree data
4. Find and extract sequence, selector, and parallel nodes
5. Create global memory objects for extracted sequence, selector, and parallel nodes
6. Create global memory objects for the rest of the nodes (as leaf nodes)
7. Add functions to leaf nodes to execute when reached
8. Run the tree and activate nodes when appropriate
9. If tree cycle is completed, stop

### 4.4.2. Chatbot system

1. Read JSON file of scripts as a dictionary
2. Create objects from the different conversation topics and add them to global memory
3. Learn behavior from memory
4. Use behavior in chat with the user
5. If input unknown, prompt user for clarification
  - 5.a. Learn input
6. If service requested, run behavior tree
7. If BT executed, ask for feedback
  - 7.a. If feedback positive, say goodbye
  - 7.b. If feedback negative, ask if human assistance is needed
8. Encrypt conversation
9. End

## 4.5. Text Encryption and Decryption algorithms

As the project intends to use a chatbot, it should also consider ways to encrypt and decrypt user-chatbot conversations. Encryption is necessary for ethical chatbots, as the privacy of the user must come first. Decryption will be needed for doing maintenance of the bot's conversations. The encryption/decryption algorithms show a link between the project system and the module of Cryptography. The following is the structure and analysis of one such algorithm:

**Word**  $\longrightarrow$  **w + o + r + d**  $\longrightarrow$  **sha1(w) + sha1(o) + sha1(r) + sha1(d)**

Every word in the conversation gets separated into chars. Each char is separately encrypted, while the total length of the word gets recorded as a passive key. These keys are only useful for putting the text back together after deciphering and have nothing to do with security. The decision to choose to hash individual characters got driven by the idea of faster decryption with brute force.

Hash **042dc4512fa3d391c5170cf3aa61e6a638f84342**



All hashes from the same encryption algorithms have the same lengths, regardless of their plain text. For SHA-1, the length is 40 bits. Everyone with this knowledge could separate the big hash into individual hashes and decipher them. In this case, that would be particularly easy, as the hashed


password space is 1 (individual chars). It is meant to be fast. To cope with that issue, bits of the hashes get shifted to certain key positions before getting integrated into the big hash.

*This is what a big hash looks like:*

```
7cf184f475c67a68d5828329ecb19349720b0cae58e6b3a4ba14a112e090df7fc6029add0f3555cc07c3
42be856e56f00e7f4347842e2e21b774e61d07c342be856e56f00e7f4347842e2e21b774e61d7a81af3
e7d591a83c713f8761ea1efe93dcf36150ab8318a1bcaf639e678dd3d02e2b5c343ed4111ca73ab6...
```

The decryption algorithm for this application extracts the big hash from the database and separates it back to individual hashes of 40 bits each. The hashes gathered got previously shuffled using a certain key pattern. Using the same pattern, each bit gets back to its original place.

042dc442512fa3d39143c5170cf3a8f8a61e6a63 → Shuffled hash  
042dc4512fa3d391c5170cf3aa61e6a638f84342 → Original hash



The whole idea of hashing individual chars comes from the need for fast decryption for big texts (such as chatbot conversations). Brute forcing 1bit strings is an extremely fast and easy implementation of deciphering, even on large text files. The algorithm decrypts letters while maintaining their sequence. Once all the letters are back to plain text, words get formulated by the passive key values - the length of every word in the original text. The system recognizes all punctuational signs and cannot get broken by entering unknown symbols with the text. It also successfully restores all capital letters and punctuation, keeping the data's integrity intact.

Testing of this program has given a successful result - texts of all sizes get deciphered in seconds. The program proves suitable for the AI chatbot, as it will ensure fast access for maintenance through the bot's conversations and its ethical use.

## 5. Methodology

Agile methodology is a flexible way of managing projects by breaking them into smaller tasks that are easier to handle. Tasks get formulated as sprints that can have a different time assigned to them - depending on their difficulty. Every sprint gets presented to the stakeholders, which ensures their content with the project. Issues found can get presented instantly, and their solution would not hurt the entirety of the project. Finding issues earlier in implementation would contribute to their faster solution and continuous improvement of the project.

The project follows the Agile methodology due to its flexibility and product quality. Supervisor meetings would take the form of fortnight sprints. Each sprint had two weeks of implementation time, and the tasks were equally split. Every supervisor meeting would start with the developer (me) presenting the work done so far to the stakeholder (my supervisor) and any occurring problems. At the end of the meeting, we would go over the tasks to be delivered for the next meeting/sprint.

## 6. Implementation

The goal for implementation is to provide a simple interface for non-AI experts that will allow them to build BT models from a pallet of existing components. As mentioned in the Research section, one of the strengths of behaviour trees is their re-usable component. The functions in the tree are separated as individual modules (leaf nodes), and their switching does not alter the tree structure. Essentially, the system should provide an interface that produces behaviour trees from graphical models and appropriate leaf nodes' functions.

### 6.1. Building a GUI for creating BT models

The first step of the implementation process is building a graphical user interface (GUI) that allows alterations and extraction of a BT model as a JSON file. Supervisor meetings have pin-pointed a similar open-source tool as the base for the BT modelling GUI. The tool is the D3JS Tree Editor by Feuer (2017) - see (Research, Tools). It enables modifying a default tree model by adding, renaming, deleting, and moving nodes around. As mentioned in Research, there is no option for extracting the edited model, or uploading a selected one. The GUI can be more user-friendly with a more appropriate design – nodes get sorted alphabetically, which denies users complete control over the flow.

This step of the project implementation ended up as the most difficult and time exhausting. D3JS Tree Editor consists of multiple files, some of which exceed 5 000 lines of code. The program files are in different languages – JavaScript, HTML, CSS, and JSON. Some implementation issues were caused by a lack of experience working with JavaScript, especially with the d3 library. Other issues concerned understanding and building upon the existing code. Supervisor meetings have provided continuous guidance and support whenever such problems would appear.

#### 6.1.1. Implement a download button for extracting BT models as JSON files

This is one of the parts that prolonged implementation time the most. The D3JS Tree Editor did not have the functionality of saving the altered graphical model data back into a JSON string. Supervisor meetings aided in finding the exact file to modify and what approach to take for recording updates in JSON formatting. The conclusion of supervisor meetings and research is that the 'dndTree1.js' file has an 'update' function that visually updates the models on the models upon alteration. The 'update' function has been improved (see added code in Figure 19) and can now additionally record the changes back into a JSON string. The system now gets string data from a circular JSON that only includes important information – all nodes in the tree and their relation.

```

641 //added by jim for 'save as' functionality
642 console.log(root); //root contains everything you need
643
644 const getCircularReplacer = (deleteProperties) => { //func that allows a circular json to be stringified
645   const seen = new WeakSet();
646   return (key, value) => {
647     if (typeof value === "object" && value !== null) {
648       if(deleteProperties){
649         delete value.id; //delete all properties you don't want in your json (not very convenient but a good temporary solution)
650         delete value.x0;
651         delete value.y0;
652         delete value.y;
653         delete value.x;
654         delete value.depth;
655         delete value.size;
656         delete value._children;
657       }
658       if (seen.has(value)) {
659         return;
660       }
661       seen.add(value);
662     }
663     return value;
664   };
665 };
666
667 var myRoot = JSON.stringify(root, getCircularReplacer(false)); //Stringify a first time to clone the root object (it's allow you to
668 var newTree = JSON.parse(myRoot);
669 newTree= JSON.stringify(newTree, getCircularReplacer(true)); //Stringify a second time to delete the propeties you don't need
670 //console.log(newTree); //You have your json in newTree

```

Figure 19: Code for recording updates of the graphical BT model in a JSON formatting

```

667 var myRoot = JSON.stringify(root, getCircularReplacer(false)); //Stringify a first time to clone the root object (it's allow you to
668 var newTree = JSON.parse(myRoot);
669 newTree= JSON.stringify(newTree, getCircularReplacer(true)); //Stringify a second time to delete the propeties you don't need
670 //console.log(newTree); //JSON in newTree
671
672 //Store the updated tree (added by Radiela)
673 sessionStorage.setItem("newTree", newTree);
674 console.log("update ", newTree);
675

```

Figure 20: Saving updated tree data code

Another issue encountered was the means of storing the updated tree JSON data. The data must get saved somewhere in order to import it into a JSON file when requested by the user. The issue was finding a way to store user-updated data into a file to download because of JavaScript security restrictions. The way around it (see Figure 20) is to store such data in session storage - which will only keep the modified JSON data in the current session.

The implementation of the download button itself was relatively straightforward. The system uses built in 'blob' constructor to represent the JSON data and place it into a file for download. The file is downloaded as a link, built from the tree data, a name for the file, and its MIME type. Figure 21 shows the code behind the downloading process and the 'download' button of the GUI.

```

164
165 // Build a download link
166 a.href = "data:" + mimeType + "charset=utf-8," + escape(data);
167
168 // Save data to file
169 if (window.MSBlobBuilder) {
170     var blob = new MSBlobBuilder();
171     blob.append(data);
172     return navigator.msSaveBlob(blob, fileName);
173 } /* end if(window.MSBlobBuilder) */
174
175
176 // The download event
177 if ('download' in a) {
178     a.setAttribute("download", n);
179     a.innerHTML = "downloading...";
180     D.body.appendChild(a);
181     setTimeout(function () {
182         var e = D.createEvent("MouseEvents");
183         e.initMouseEvent("click", true, false, window, 0, 0, 0, 0, 0, false, false, false, false, 0, null);
184         a.dispatchEvent(e);
185         D.body.removeChild(a);
186     }, 66);
187     return true;
188 }; /* end if('download' in a) */
189
190
191 <script>
192 function save() { // Added by Radiela, until END of FILE
193     // Get updated tree from session
194     var tree = sessionStorage.getItem("newTree");
195
196     // Use the download function above
197     var t = download(tree, 'BTree.json', 'application/json');
198 }
199 </script>

```

Figure 21: Download event and 'download' button function in index.html

### 6.1.2. Implement an upload button for loading a chosen JSON BT model

The D3JS Tree Editor loads a default tree model on the GUI webpage from a JSON file. The default tree model provided is particularly large, hence not a very convenient model for complete alteration. It will take a long time and effort to clean up the model before actually starting to build another. Even if the default tree was smaller - for example, just a root node, users will not be able to reuse pre-built models. A simple solution to this problem is adding an upload button that allows users to display a JSON file of their choosing. This way, users can reuse previously designed trees and choose models requiring fewer alterations.

The upload functionality was easier to implement, as the tree generating function was already discovered while looking for a solution to the problems with the download option. The system uses a 'search files' form to help the user select a JSON file. When a file gets chosen, the system gathers its path from the webpage and extracts just the file name. In order to display the tree from the new file, the system must replace the default model with the selected one in the tree generating function. To see code of the above mentioned, look at Figure 22 below.

```

217 <div id="upload">
218   <script>
219     function send() {
220       var file = document.getElementById("myFile"); // Uploaded file by user
221       var path = file.value; // The path of the file
222       console.log(path);
223
224       var file_name = path.substring(12); // Extract just the file name from the path
225       console.log(file_name);
226       var treeJSON = d3.json(file_name, draw_tree);
227       $('#tree-container').html(treeJSON); // Replace the previous tree model with selected file
228     }
229   </script>

```

Figure 22: Upload file function

### 6.1.3. Make GUI design more approachable

As mentioned in Research, the GUI could be more user-friendly. One major issue is that the tree automatically sorts the existing nodes alphabetically. That means that users do not have complete control over the flow of the tree. One way to fix that is to find the source of the sorting function and change or remove it. In this case, the default sorting is in the form of LILO - the last one in is the last one out. The function for alphabetical sorting had to be removed to use the default, line 171 from the 'dndTree1.js' file.

Another thing that could improve the GUI is if it could visualize the BT flow upon selected nodes considered successful. Unfortunately, the time proved insufficient for the amount of research required to implement this functionality. Instead, to make the GUI more user-friendly, the system visually distinguishes action nodes from structural (sequence, selector, parallel) nodes. The circle of structural nodes is painted pink, and the circle of action nodes is bright green (see Figure 24). Another functionality added to the GUI is a reset button. It reloads the default model back to the page and is convenient to use if something goes wrong during design. More colours and formatting are added to the GUI for better readability.

### 6.1.4. Stages of GUI design

#### 6.1.4.1. Figure 23:



Figure 23: Design for Project in Progress Day

#### 6.1.4.2. Figure 24: The final BT modelling GUI design

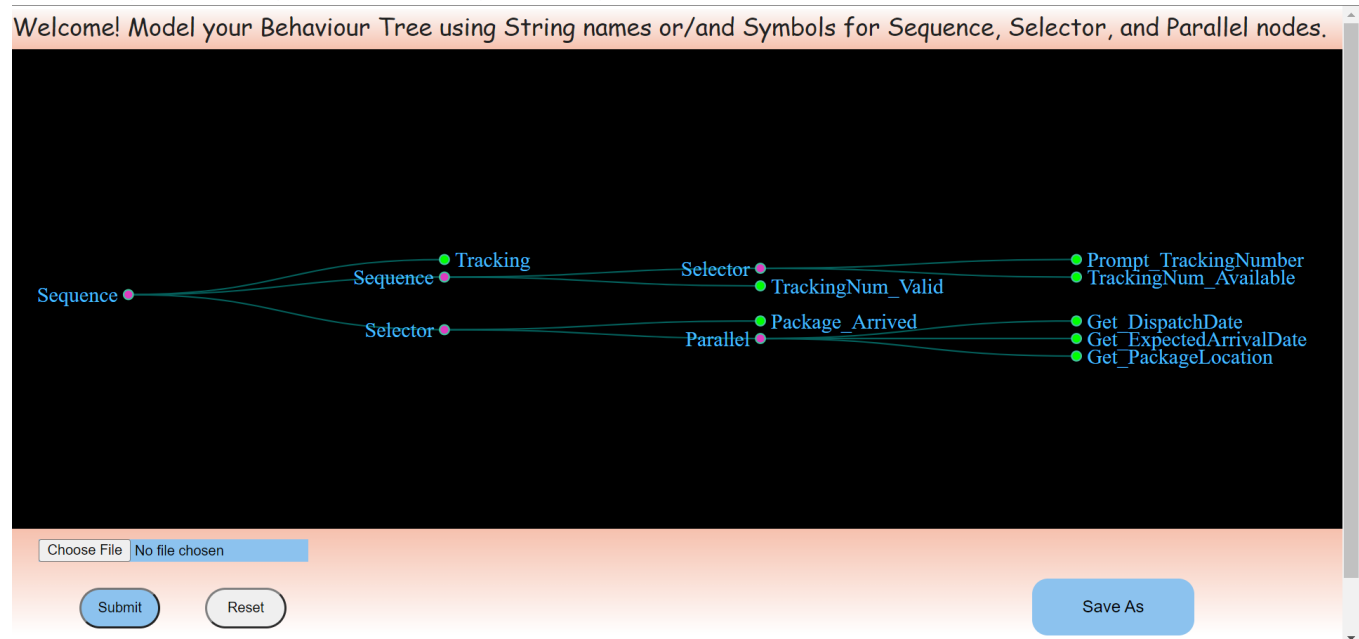


Figure 24: The final BT modelling GUI design

#### 6.1.5. Acceptance tests passed from this system

Test ID	Tested Requirement	Test Description	Expected Outcome
AT1.	BTM1 and BTM2	Test if the system can successfully generate graphical tree models from a JSON file	The system can import JSON BT data and display the graphical model on the modelling GUI
AT2.	BTM3	Test adding a new node to the graphical model	User to be able to add new nodes to the BT model
AT3.	BTM4	Test renaming existing nodes	User to be able to rename nodes in the BT model
AT4.	BTM5	Test deleting a node	User to be able to remove nodes from the BT model
AT5.	BTM8	Test downloading current model as a JSON file	User to be able to download/extract their BT model from the GUI in JSON format
AT6.	BTM9	Test uploading JSON file to the GUI	User to be able to upload a chosen BT model to the GUI from a JSON file
AT7.	BTM10	Test drag-and-drop input method for modifying the model	User to be able to alter the BT model using drag-and-drop input method

## 6.2. Building a BT Code Generating program

The next step of the implementation process is building a Python program for generating running code from a JSON file of a behaviour tree. Supervisor meetings provided guidance for the approach to be taken for designing the system. Research has concluded that the Anytree python library has a function syntax very similar to the syntax of a JSON string (see Figure 25 below). Anytree is used for building tree class instances and allows tree traversing - running the tree structure.

Example of the JSON string imported from the JSON file:

```
{“name” : “sequence”, “children”:[{“name” : “child one”}, {“name” : “child two”}]}
```

Example of Anytree code:

```
Node (name = sequence, children = [Node (name = “child one”, Node (name = “child two”))])
```

*Figure 25: JSON string vs Anytree string*

### 6.2.1. Data processing



The code generating system starts with reading a JSON file of a behaviour tree and transferring it into a JSON string. As mentioned, there is a pattern (see Figure 26) between the JSON string data and Anytree code. The JSON string gets transformed according to the pattern, and the string undergoes the 'eval()' function, which executes it as a running code. The executed code builds the tree structure from the JSON model.

There is a 'Board' class that serves the purpose of structuring nodes' data that would go onto a blackboard/global memory object. The nodes' data gets gathered from a sequence of functions in the code. The "Set data" function extracts structural (sequence, selector, parallel) nodes from the JSON data, recognizing both their string names and allocated symbols. "Create board objects" traverses the tree and allocates nodes as objects in the global memory object. Each node has a name, an index for repeating names, child nodes, a parent, and a ticked state – whether the node has returned success or not.

When running the tree, the nodes' success state is appropriately updated in the global memory by a 'tick node' function. The tree cycle gets completed when the 'root' node is ticked or successful. That would mean that the tree has executed all necessary functions for the purpose it serves. Setting up and running the tree provides reusable code as it can handle different models in the same way.

There is a database section in the code that provides a reusable connection to the user's database and functions to extract whole tables and specific information (see Figure 27). The user only has to update the DB source details with their own and choose a table to work with.

```
36
37 def set_tree(data):
38     """Remove redundant symbols and Structure tree nodes"""
39
40     data = data.replace("{", "Node(")
41     data = data.replace("}", ")")
42     data = data.replace('"name"', 'name')
43     data = data.replace('"children"', 'children')
44     data = data.replace(":", " =")
45     data = data.replace("_children = null,", "")
46
47     return data
```

Figure 26: Pattern for transforming JSON string into Anytree code

```

408 def get_database_column(column):
409
410     # Change the database source, if using a database
411     con = pymysql.connect(host="localhost", user="root", password="", database="delivery tracking") # MySQL database
412     data = []
413
414     with con.cursor() as curs:
415         # Get all items from ('packages') - chosen Table to work with
416         curs.execute("SELECT * FROM 'packages'")
417         packages = curs.fetchall()
418         # print(packages)
419
420         # Get all tracking numbers - get specific items from the Table data
421         num = len(packages)
422         for i in range(0, num):
423             item = packages[i][column]
424             data.append(item)
425
426     con.close()
427     return data

```

Figure 27: Reusable function for connecting to a database

### 6.2.2. Leaf nodes' functions

Generating the code for building a BT model from a chosen JSON file is automated and reusable for different JSON models. Behaviour trees will always have sequence and selector nodes structuring the hierarchy of the tree. The only thing that changes for the different models is the functions in the leaf nodes – the functionality of the tree. For the example of the delivery tracking system, such functions would be prompting user's input or tapping into a database (see Figure 28) to check or collect some data. There is a section in the code specified for the leaf node functions. The user can update them to suit their chosen model - the system sustains reusability.

Tracking 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
en660m998	BS01 0AU, Bristol, United Kingdom	2022-02-22	2022-03-04	London, United Kingdom
wf22588u76	BS16 5GF, Bristol, United Kingdom	2022-03-02	2022-04-05	Chelsea, United Kingdom

Figure 28: The database for the tracking delivery example system

### 6.2.3. Attaching functions to the leaf nodes

There is a section in the code specified as 'Attaching functions to leaf nodes' that is responsible for attaching appropriate functions to action nodes. The code represents a sequence of if-statements that allocate a specified function to a specific node. This component is meant to be simple to understand and alter, even by non-AI experts.

The final step of setting up the behaviour tree is to make sure that when a node successfully fulfils its purpose, it gets the state of 'ticked' (successful). In the section for setting up the leaf nodes, there is a function call to a 'tick node' action after the completion of each leaf function. It is advised not to leave out the call as it is responsible for tracking the tree traversing.

#### 6.2.4. Running the behaviour tree

According to design specifications, the tree should get adopted by a dialogue system that would start its cycle when requested by the user. The dialogue system is further explained in the next section but essentially it recognizes users' prompts for service and calls the 'run tree' function.

#### 6.2.5. Displaying the tree model and visited nodes

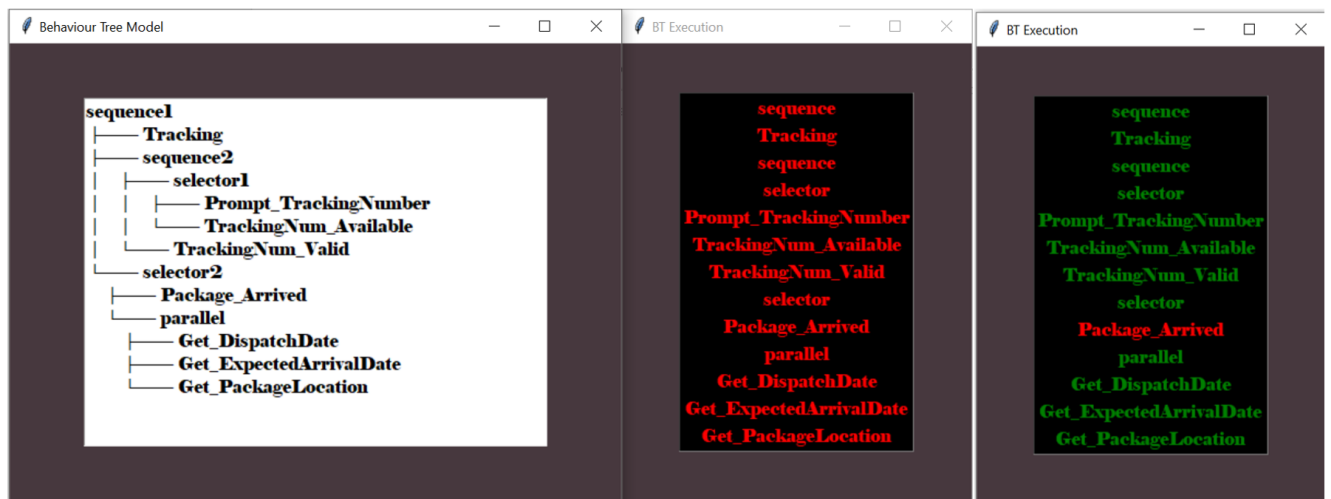


Figure 29: 1-Tree structure, 2-Service not prompted, 3- Service prompted

At the end of the program run, users can see the graphical model the way the system sees it (see Figure 29). They can also see all the visited nodes. The nodes' colour is initially red, and it only changes to green if the given node has returned success. This program component serves the purpose of visual testing the tree logic. Users can check how accurately the system has interpreted their JSON behaviour tree models and whether the tree flow is as intended. This system is also reusable, as it generates the graphical tree and nodes' view from the generated tree class instance.

### 6.2.6. Acceptance tests passed by this system

AT8.	BTCG1	Test if the code generating system can import JSON files	Code Generating system to be able to successfully import BT data through a JSON file
AT9.	BTCG2	Test if the system can generate running code from the JSON file of the BT model	System to be able to generate running code for the behaviour tree in the JSON file
AT10.	BTCG3 and BTM6	Test if the system recognises sequence and selector nodes from the BT model	System to successfully distinguish selector and sequence nodes in the generated code
AT11.	BTCG4 and BTM7	Test if the system recognises parallel nodes from the graphical model	System to successfully distinguish parallel nodes in the generated code
AT12.	BTCG5	Test if the system successfully utilizes its form of global memory	System to be able to store and update tree cycle data to a form of global memory
AT13.	BTCG6	Test if the system can automatically generate a behaviour tree structure	System capable of automatically generating BT structures from given JSON data
AT14.	BTCG7	Test the system's interpretation of the BT structure on the model	The system can provide an accurate graphical representation of the BT model
AT15.	BTCG8	Test the system's tree traversing accuracy	The tree cycle is running as intended in the model

### 6.3. Dialogue management system – chatbot

The project requires a dialogue management system to adopt and test the logic of the behaviour tree. Such a system could be a rule-based or AI-based chatbot. Initially, research suggested using NLTK to create a chatbot with a two-layer Neural Network. However, the NLTK chatbot would take too long to learn - not very convenient for fast alterations. The chatbot system must recognize a service prompt by the user, and flaws in that recognition are not tolerable.

As the BTs are meant to manage task-oriented dialogues, a rule-based chatbot was considered sufficient. The chatbot system learns its behaviour from a JSON file containing tags – the conversation topics available to the system, patterns – the chatbot's recognition of the context, and responses – a set of responses for specified conversation topics. Supervisor meetings suggested including a user-initialized small talk for a more natural conversation flow.

### 6.3.1. Behaviour data processing

The chatbot system starts with learning a preset behaviour from reading the JSON file 'intents' as a dictionary. The system also uses a form of blackboard/global memory that gets fed with the behaviour data extracted from the dictionary and structured by the 'Brain' class (see Figure 30). Each behaviour has a conversation topic, patterns for topic recognition, and a suitable response. Essentially the bot's training consists of uploading behaviour data to global memory that can be read throughout execution.

```
43 def train_behaviour(data):
44     behaviour = []
45
46     # Loop through JSON file and extract behaviour data
47     for intent in data['intents']:
48         memory = Brain()
49
50         category = copy.deepcopy(intent['tag'])
51         memory.tag = category
52
53         message = copy.deepcopy(intent['pattern'])
54         memory.patterns = copy.deepcopy(message)
55
56         response = copy.deepcopy(intent['responses'])
57         memory.responses = copy.deepcopy(response)
58
59         behaviour.append(memory)
60
61     return behaviour
```

Figure 30: Chatbot behaviour training function

### 6.3.2. Conversation data processing

As mentioned in the research, ethical chatbots encrypt their user-system conversations to protect the user's privacy and personal data. The chatbot system of this project records the chat in a 'Conversation' text file, which gets encrypted as the last task of the system. The type of encryption is specified in the Design section. Encryption deletes the conversation file and moves its hashed content to a 'Hash' text file, which could be switched with an actual database.

The purpose of saving these conversations is to observe the chatbot's success. A decryption system is also necessary for recovering the initial text from the 'Hash'. The program for decryption is not included in the main run as it should only get accessed by maintenance staff. The run time for decryption is within 2 seconds, tested from multiple runs and different lengths of conversations.

### 6.3.3. Chatbot's conversation scope

The JSON file 'intents' contains all the topics the chatbot can handle – recognizing greetings, small talk, gratitude, whether the user is happy or unhappy, whether a human presence has been requested, and catching service prompting. It interprets the context from keywords and phrases in the user's input (see Figure 31). When the intent is unrecognized, the chatbot would ask for clarification based on its available topics. If feedback gets provided, the chatbot will add the input to the appropriate pattern component.

```
88 def find_tag(user_input, intents):
89     # Find the category of the user's input
90     tag = ""
91     for item in intents:
92         if user_input in item.patterns: # If the exact input exists
93             tag = item.tag
94         else:
95             words = user_input.split(" ") # If not, split the sentence into keywords
96             for word in words:
97                 if word in item.patterns: # Check for keywords
98                     tag = item.tag
99     return tag
100
101
102 def respond(category, intents):
103     # Respond to certain categories
104     for obj in intents:
105         if obj.tag == category:
106             r = random.randint(0, len(obj.responses) - 1)
107             print(obj.responses[r])
108             write_to_file("Bot: " + obj.responses[r] + "\n")
```

Figure 31: Functions for recognizing the context of topics and providing a suitable response

### 6.3.4. Conversation evaluation

Feedback from Project in Progress Day introduced the question – what will happen after the tree cycle? How can the system be sure that the user's query got answered?

The chatbot system can recognize service prompts. In this case - when the user has a tracking delivery query. When service is requested, the chatbot runs the BT managing the task-based dialogue. When the tree run terminates, the system will prompt the user for feedback. If the user is unhappy with the outcome of their conversation, they can request a human operator. The feedback gets recorded as part of the conversation for spotting maintenance issues (see Figure 32).

```

111 def feedback(intents):
112     # When the service function has been executed, ask for feedback
113     query = "Was I able to help you today?"
114     write_to_file("Bot: " + query + "\n")
115     print(query)
116
117     answer = input("You:")
118     write_to_file("User: " + answer + "\n")
119
120     tag = find_tag(answer, intents)
121     if tag == "yes":
122         respond("thanks", intents)
123     elif tag == "no":
124         print("I seem to have failed my task.")
125         print("Would you like to speak to a member of staff?")
126
127         new_inp = input("You: ")
128         if new_inp == "yes":
129             respond("human_needed", intents)
130         else:
131             respond("goodbye", intents)
132     else:
133         respond("goodbye", intents)

```

Figure 32: Function for gathering user feedback

### 6.3.5. Acceptance tests passed by this system

AT16.	DMS1 and DMS2	Test if the dialogue system can adopt the BT code without conflict	System successfully running BT code from the dialogue management system
AT17.	DMS3	Test if the chatbot accurately interprets task-oriented context	System successfully recognising user intents and providing accurate responses
AT18.	DMS4	Test if the chatbot allows user-initiative of not too far off topics	System successfully recognises and handles small talk triggered by the user
AT19.	DMS5	Test if the chatbot always asks for feedback when its service has been triggered	System always asks for feedback after completing the behaviour tree cycle
AT20.	DMS6	Test the chatbot's learning functionality	System capable of learning from user feedback
AT21.	DMS7	Test encryption	Conversations are instantly hashed after user says goodbye
AT22.	DMS8	Test decryption system	Conversations are successfully decrypted, and the plain text is identical with the original conversation

## 6.4. User Testing

The project got tested by a group of non-AI experts with basic programming knowledge. They got introduced to the concept of behaviour trees through real-world examples – NPCs in video games and the dialogue management example in Figure 2. Their task was to create a new BT model for managing banking service queries. They were provided with a model (Figure 33), pre-built leaf nodes functions and a database appropriate for the new service (*see Appendix*). Testers were also prompted to use the provided instructions for adapting BT models.

All testers managed to complete the task within half an hour. The outcome of all tests passes the user acceptance tests. They were further asked for feedback on the project functionality, what they liked the most, and what could be improved. Their summative functionality feedback is in Figure 34. Mostly, people liked the project idea and the fact that the chatbot goes straight to the point when solving their query. The testing established that there is room for improvement on the chatbot system – it could be more intelligent; the use of a UI would make the process neater.

6.4.1. Figure 33: Model for handling banking services customer queries

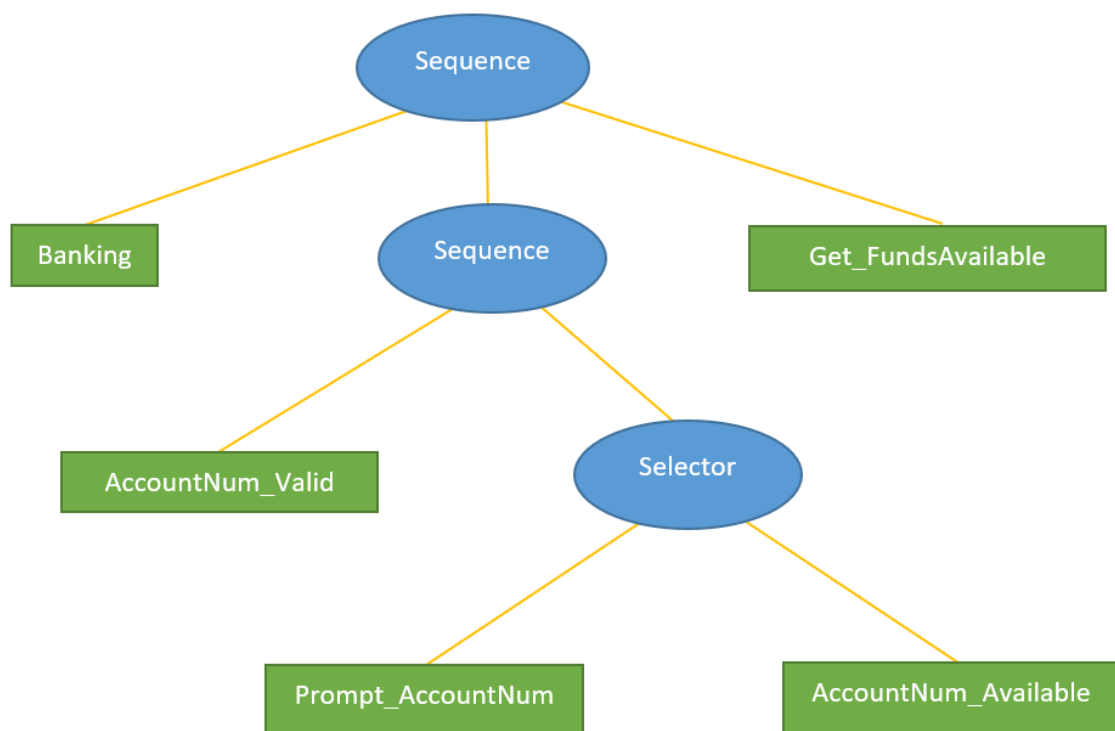


Figure 33: Model for handling banking services customer queries



#### 6.4.2. Figure 34: Summative user tests functionality feedback

From 1 to 5, how would you rate the intelligence of the chatbot?

1	2	3	4	5
---	---	---	---	---

From 1 to 5, how straightforward was the conversation?

1	2	3	4	5
---	---	---	---	---

From 1 to 5, how well did the bot do its job?

1	2	3	4	5
---	---	---	---	---

From 1 to 5, how would you rate the time it takes to generate a new BT code?

1	2	3	4	5
---	---	---	---	---

From 1 to 5, how well does the chatbot recognize whether your answer is happy or unhappy?

1	2	3	4	5
---	---	---	---	---

From 0 to 5, how well did the chatbot understand your service prompt?

1	2	3	4	5
---	---	---	---	---

Figure 34: Summative user tests functionality feedback

## 7. Project Evaluation

Bringing out the strengths of Behaviour Trees has been a fun and interesting experience. The Research section aims to be as broad as the words budget would allow, bringing the concept of the project together for the reader. The research data gets split into easy-to-handle topics providing all the necessary background information for the appropriate reader's assessment. One of the topics is the ethical status of the project and ethical issues that could face chatbot systems. Encryption of chatbot-user conversations before their storage is a must to ensure the user's privacy and personal data safety.

Requirements got gathered through supervisor feedback and the outcome of the research. They split the project into three systems – a BT modeller, a running code generating system, and a dialogue management system – a chatbot. Each of the three programs has a use-case diagram visualization that confirms the value of the set requirements. The functional requirements are extracted from the non-functional and are the base for structuring the user acceptance tests. Each requirement is included in at least one acceptance test, ensuring their fulfilment.

The Design section formulates the structure of the desired product from the requirements. The first step is bringing out user stories from the use-cases and requirements lists. The user stories involve three stakeholders – the general user – could be an AI expert, a non-AI expert, and a member of staff for maintaining the chatbot system. This section also displays the train of thought behind the project system – all system components and their relations. The functionality of all systems gets visualized in flowchart diagrams. Further explanation of the intents for the chatbot and the code generating system is provided in pseudocode. This section also goes over details of the encryption and decryption algorithms.

Implementation goes over the systems from the Design section in much more detail – down to the code level. The implementation of the BT modelling GUI proved the most problematic - it took the longest time, and there is still room for improvement. Supervisor meetings provided constant guidance whenever problems would occur – this section had the most dedicated supervisor meetings. The implementation issues revolve around confusion with the open-source code or limitations in JavaScript and d3 library.

In contrast, the implementation of the BT code generator was the easiest and took the least time. Supervisor guidance was taken into account, and the product was ready within the implementation time appointed. Feedback from Project in Progress Day suggested considering the question – what happens after the tree cycle gets completed? How can the system be sure that the user's query got answered? In response to the feedback, the chatbot system now gathers user feedback when service has been requested. Part of the feedback process is to prompt a human assistant if needed – this functionality would ensure solving the user's query.

The user testing outcome confirmed that the system passes all acceptance tests. However, there are still things that can be improved – mainly the chatbot system. Users did consider the chatbot system intelligent, and it did recognize all their service requests. However, they have pointed out that the system could be more intelligent, and it would be neater if it had a user interface. As the chatbot is rule-based, it interprets conversations using keywords and phrases. That sometimes compromises the natural flow of the conversation. Overall, users perceived the project as successful, and their testing fulfilled the laid-out requirements and acceptance tests.

## 8. Conclusions and Further works

A major goal of this project was to separate the creation of the underpinning AI component from the design of the dialogue flow. The aim was met by developing a program that reads JSON files of task-oriented dialogues and generates the appropriate code for the given tree structure. That makes BTs approachable for non-AI experts.

The system allows the rapid authoring of the BTs from a pallet of existing components, meaning that the user can use their own functions (or pre-set) as the leaf nodes. The tool also allows visual and functional testing through the two Python programs. Users can see both how the JSON model got interpreted by the system and which nodes have been visited during the tree cycle.

Functional testing gets supported by the Python chatbot system. The chatbot facilitates understanding of the conversation flow and generates appropriate behaviour according to the context. The system adopts the behaviour tree structure for managing task-oriented dialogues – such as handling customer queries for a delivery tracking system.

## 8.1. Future works

As mentioned in the project evaluation, there is room for improvement in the behaviour tree modelling GUI. The current system visually distinguishes structural nodes from actual nodes in the colour of their circles. The Python program displays the interpreted flow from the JSON model and the nodes visited during the tree cycle. However, it would be more convenient if the BT flow gets visualized during the design stage instead of testing. The user feedback on improvement suggests the chatbot should have a proper graphical user interface as well. User testing also suggests working on a more natural flow of the conversations with the chatbot.

## 9. Referencing

1. Adam Feuer (2017) d3js Tree Editor. Source:

<https://gist.github.com/adamfeuer/042bfa0dde0059e2b288>

2. Colledanchise, M. and Ögren, P. (2014) How Behavior Trees modularize robustness and safety in hybrid systems *IEEE Xplore*. 1 September 2014 [online]. 1482–1488. Available from:

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6942752>

3. Lim, C.-U., Baumgarten, R. and Colton, S. (2010) Evolving Behaviour Trees for the Commercial Game DEFCON. *Applications of Evolutionary Computation*. [online]. pp.100–110. Available from:

[https://link.springer.com/chapter/10.1007/978-3-642-12239-2\\_11](https://link.springer.com/chapter/10.1007/978-3-642-12239-2_11)

4. Nirwan, D. (2020) *Designing AI Agents' Behaviors with Behavior Trees*. Available from:

<https://towardsdatascience.com/designing-ai-agents-behaviors-with-behavior-trees-b28aa1c3cf8a>.

5. Colledanchise, M. and Ögren, P. (2018) *Behavior Trees in Robotics and AI: An Introduction*.

arXiv:1709.00084 [cs]. [online]. Available from: <https://arxiv.org/abs/1709.00084>.

6. Simpson, C. (2014) Behavior trees for AI: How they work Game Developer. 18 July 2014 [online].

Available from: <https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work>.

7. Dolejší, J. (2022) *VS Code Behavior Tree editor GitHub*. 21 January 2022 [online]. Available from:

<https://github.com/jan-dolejsi/vscode-btree>

8. Abad, D. (2021) *Behavior Tree GitHub*. 2 December 2021 [online]. Available from:

[https://github.com/Oxabad/behavior\\_tree/](https://github.com/Oxabad/behavior_tree/)

9. Bostock, M. (2021) D3.js - Data-Driven Documents d3js.org. [online]. Available from:

<https://d3js.org>.

10. Brockmann, P., (2014) CollapsibleTree Search Gist. [online]. Available from:

<https://gist.github.com/PBrockmann/0f22818096428b12ea23>

11. Anyoha, R. (2017) *The History of Artificial Intelligence Science in the News*. 28 August 2017

[online]. Harvard University. Available from: <https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/>

12. Weizenbaum, J. (1966) ELIZA---a computer program for the study of natural language communication between man and machine. *Communications of the ACM*. [online]. 9 (1), pp.36–45. Available from: <https://dl.acm.org/doi/abs/10.1145/365153.365168>
13. Coheur, L. (2020) From Eliza to Siri and Beyond. *Information Processing and Management of Uncertainty in Knowledge-Based Systems*. [online]. pp.29–41. Available from: [https://doi.org/10.1007/978-3-030-50146-4\\_3](https://doi.org/10.1007/978-3-030-50146-4_3)
14. Levine, S. (2022) *It's the Golden Age of Natural Language Processing, So Why Can't Chatbots Solve More Problems?* Medium.1 March 2022 [online]. Available from: <https://towardsdatascience.com/its-the-golden-age-of-natural-language-processing-so-why-can-t-chatbots-solve-more-problems-bada37a427cf>
15. Microsoft (nodate) Language Understanding (LUIS) Documentation - *Azure Cognitive Services* docs.microsoft.com. [online]. Available from: <https://docs.microsoft.com/en-us/azure/cognitive-services/luis/>.
16. Jiang Zhao, Y., Ling Li, Y., and Lin, M. (2019) A Review of the Research on Dialogue Management of Task-Oriented Systems. *Journal of Physics: Conference Series*. [online]. 1267, p.012025. Available from: <https://iopscience.iop.org/article/10.1088/1742-6596/1267/1/012025>
17. Jokinen, K. and McTear, M. (2009) Spoken Dialogue Systems. *Synthesis Lectures on Human Language Technologies*. [online]. 2 (1), pp.1–151. Available from: <https://www.morganclaypool.com/doi/pdf/10.2200/S00204ED1V01Y200910HLT005>
18. Wen, TH., Vandyke, D., Mrksic, N., et al. (2017) A Network-based End-to-End Trainable Task-oriented Dialogue System. *arXiv:1604.04562* [cs, stat]. [online]. Available from: <https://arxiv.org/abs/1604.04562>.
19. Bordes, A., Boureau, Y.-L. and Weston, J. (2017) Learning End-to-End Goal-Oriented Dialog. *arXiv:1605.07683* [cs]. [online]. Available from: <https://arxiv.org/abs/1605.07683>
20. Zhang, Z., Takanobu, R., Zhu, Q., Huang, M., and Zhu, X. (2020) SCIENCE CHINA Technological Sciences. *REVIEW. Recent Advances and Challenges in Task-oriented Dialog Systems*. [online]. c Science China Press and Springer-Verlag. Available from: <https://arxiv.org/pdf/2003.07490.pdf>
21. McTear, M. (2010) *Chapter 9 - The Role of Spoken Dialogue in User–Environment Interaction* Aghajan, H., Delgado, R.L.-C. and Augusto, J.C. (eds.) ScienceDirect.1 January 2010 [online]. Oxford, Academic Press, 225–254. Available from: <https://www.sciencedirect.com/science/article/pii/B9780123747082000097?via%3Dihub>
22. Bernsen, N.O., Dybkjær, H., and Dybkjær, L. (2016) Exploring the limits of system-directed dialogue. *Centre for Cognitive Science, Roskilde University*. Access from: [https://www.researchgate.net/profile/Hans-Dybkjaer/publication/221484557\\_Exploring\\_the\\_limits\\_of\\_system-directed\\_dialogue\\_dialogue\\_evaluation\\_of\\_the\\_danish\\_dialogue\\_system/links/56f13fce08aec9e096b31938/Exploring-the-limits-of-system-directed-dialogue-dialogue-evaluation-of-the-danish-dialogue-system.pdf](https://www.researchgate.net/profile/Hans-Dybkjaer/publication/221484557_Exploring_the_limits_of_system-directed_dialogue_dialogue_evaluation_of_the_danish_dialogue_system/links/56f13fce08aec9e096b31938/Exploring-the-limits-of-system-directed-dialogue-dialogue-evaluation-of-the-danish-dialogue-system.pdf)

23. Colledanchise, M., Marzinotto, A. and Ögren, P. (2014) *Performance analysis of stochastic behavior trees* IEEE Xplore.1 May 2014 [online]. 3265–3272. Available from: <https://ieeexplore.ieee.org/abstract/document/6907328/citations#citations>
24. Amal, D. A., Joe, C., Raylene, Y. (2019) Basics of Automata Theory. *Stanford.edu.2019* [online]. Available from: <https://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/automata-theory/basics.html>
25. Chellapilla, K. and Czarnecki, D. (1999) *A preliminary investigation into evolving modular finite state machines* IEEE Xplore.1 July 1999 [online]. 2, 1349–1356 Vol. 2. Available from: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=782607>
26. Yannakakis, M. (2000) Hierarchical State Machines. *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics*. [online]. pp.315–330. Available from: [https://link.springer.com/chapter/10.1007/3-540-44929-9\\_24](https://link.springer.com/chapter/10.1007/3-540-44929-9_24)
27. Iovino, M. et al. (2020) *A Survey of Behavior Trees in Robotics and AI*. arXiv:2005.05842 [cs]. [online]. Available from: <https://arxiv.org/abs/2005.05842>.
28. Daniel, J. and Martin, J. (2021) *Speech and Language Processing* [online]. Available from: <https://web.stanford.edu/~jura/slp3/24.pdf>.
29. Wilson, H.J. and Daugherty, P.R. (2019) *How Humans and AI Are Working Together in 1,500 Companies* Harvard Business Review. 4 April 2019 [online]. Available from: <https://hbr.org/2018/07/collaborative-intelligence-humans-and-ai-are-joining-forces>.
30. Murtarelli, G., Gregory, A. and Romenti, S. (2020) A conversation-based perspective for shaping ethical human–machine interactions: The particular challenge of chatbots. *Journal of Business Research*. [online]. Available from: <https://www.sciencedirect.com/science/article/pii/S0148296320305944>.
31. Amditis, J. (2017) *What are some ethical considerations that chatbots raise, and how should they be addressed?* Medium.22 March 2017 [online]. NJ Mobile News Lab. Available from: <https://medium.com/mobilenewslab/what-are-some-ethical-considerations-that-chatbots-raise-and-how-can-those-be-best-be-addressed-7f7fed23557>.

## 10. Appendices

### 10.1. Instructions for adapting new BT models to the project system

---

1. Download BT model from the GUI system and place it in the project folder
2. Update files in Python project
  - 2.a. Go to 'BT\_Generator.py' --> Reading JSON function --> line 15, replace the file name
3. Go to 'BT\_Generator.py' --> line 411 in get\_database\_column() function
  - 3.a. Replace the database source values with your own
  - 3.b. Replace 'packages' with the name of the desired table in your DB
4. In the same program, go to line 242 - big\_traversing() function
  - 4.a. Lines 252 to 255 are the database columns. Replace their values with your own.
5. In the same program, change the action(leaf) node functions
  - 5.a. Go to line 347 - leaf functions' section
  - 5.b. Place your functions in that section. Remove redundant functions.
6. Go back to line 242 - big\_traversing()
  - 6.a. Replace nodes' names with the ones on your new BT model
  - 6.b. Replace their functions with the new ones from the previous step
  - 6.c. Make sure that every node is using 'tick\_node()' after executing their action
7. Replace trigger word "Tracking" with another word chosen
  - 7.a. Go to 'BT\_Generator.py'
    - 7.a.1. Use (ctrl + f) to open a search bar in the code and replace all "Tracking" instances
  - 7.b. Go to Chatbot program and do the same ^
8. Go to intents.json
  - 8.a. Use (ctrl + f) to open a search bar and look for "Tracking" tag
  - 8.b. Replace the name of the tag with the service of the new model
  - 8.c. Update the 'patterns' with appropriate data for recognizing service prompts
9. Run Main

### 10.2. Instructions for changing the scope of the chatbot's conversations

- 1 Find the file "test.json" in the project folder and open it.
  2. The chatbot is set to gather its scope of conversation from "intents.json". Replace "intents" with "test.json" and run.
  3. While running make the chatbot learn something new and it will automatically update "intents.json" with the altered structure in "test.json".
  4. Once "intents.json" is modified, place it back into the function for reading JSON files.
- \* If needed, the "intents.json" file can be fully erased and loaded back from "test.json", by giving the chatbot an unknown input.

### 10.3. Leaf functions provided during user testing of adapting a new BT model to the system

```
def prompt_account_number():
    user_input = input("Please enter your account number: ")

    return user_input

def validate(user_input):
    # Validation of user input
    valid_numbers = get_database_column(0) # Tracking numbers
    validation = False

    if user_input in valid_numbers:
        validation = True
        print("Validation Successful!")
        print(" ")

    return validation

def get_order_id(user_input):

    tracking_numbers = get_database_column(0)
    order_id = tracking_numbers.index(user_input)

    return order_id

def get_funds_available(column, row):
    data = get_database_column(column)
    date = data[row]

    return date
```

### 10.4. Database example for Banking BT model

account number	funds_available
wf22588u76	2000.5
en660m998	500.01