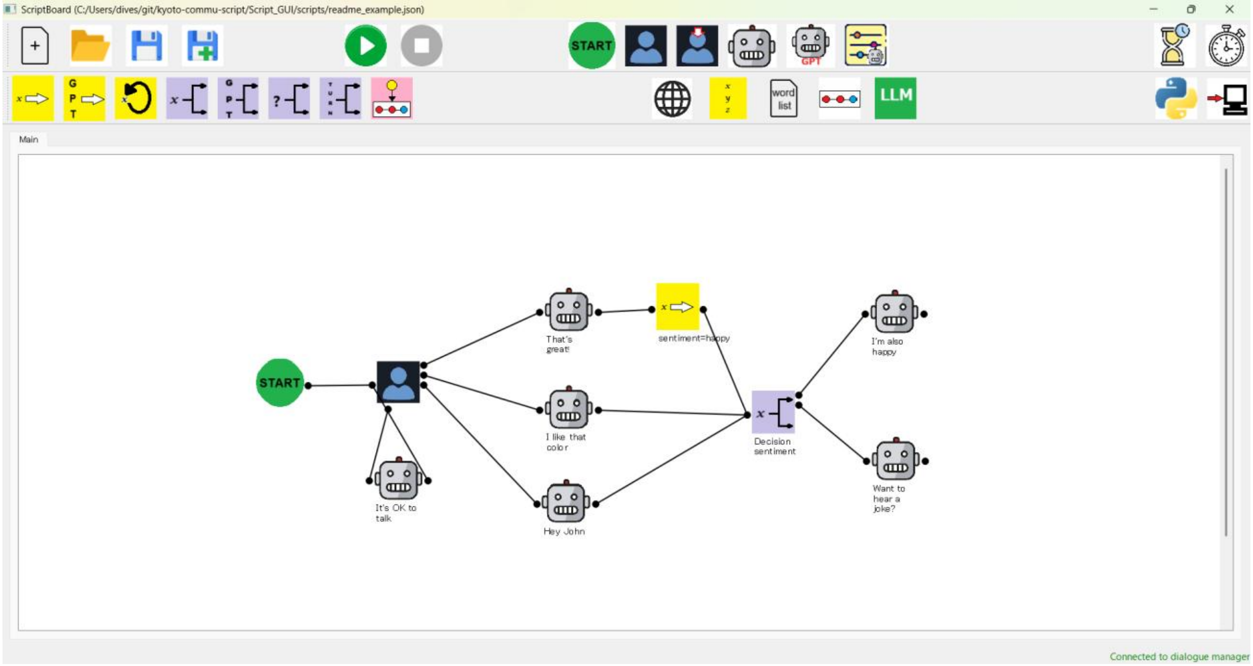
How to use ScriptBoard:

ScriptBoard is a visual programming system for creating conversational scripts and scenarios with an autonomous robot or agent. The basic GUI is shown below:

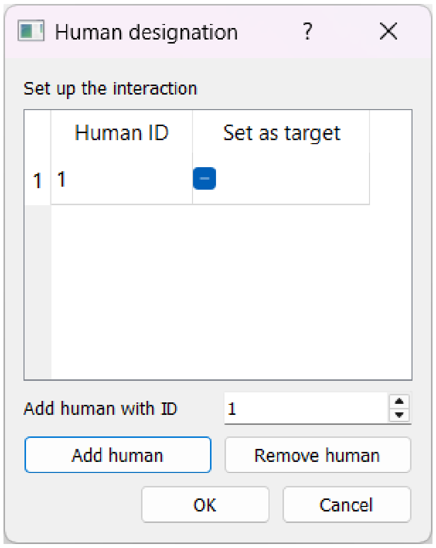


Basic interface of ScriptBoard

Setting up the environment:

🌐

Set the number of humans who will participate in the script by clicking on the "environment" button. Add the appropriate number of humans along with their ID numbers. These numbers should correspond to those used in the dialogue manager. In the example below, one human has been added with an ID of 1, and is designated as the target human. A maximum of one human can be the target but it is possible for a script to have no target humans.



Set up the interaction

	Human ID	Set as target
1	1	☑

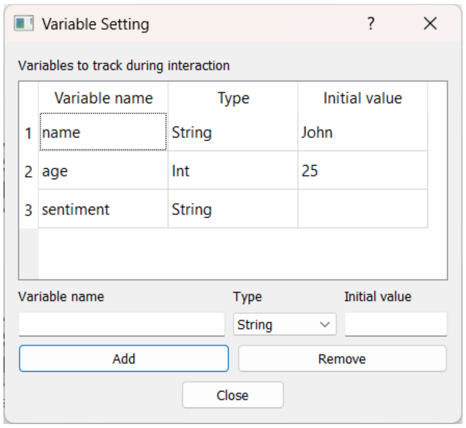
Add human with ID: 1

Environment window

Setting variables:

x
y
z

Variables for the script can be set by clicking on the “variable list” button. These can be accessed and modified by the script to control dialogue flow. **Set the name and type (string, int, float or boolean) of a variable and its initial value.** In the example below the variables “name”, “age” and “sentiment” have the values “John”, 25 and “”(empty value), respectively.



Variables to track during interaction

	Variable name	Type	Initial value
1	name	String	John
2	age	Int	25
3	sentiment	String	

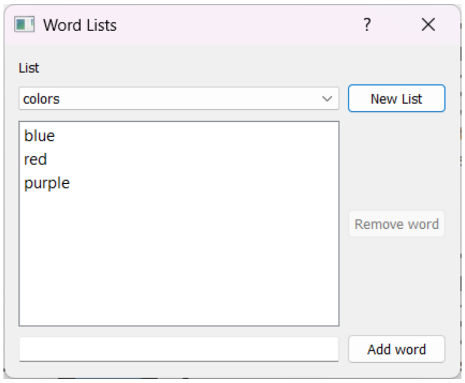
Variable name · Type: String · Initial value

Variable window

Setting word lists:

word
list

Word lists can be created and modified by clicking on the “word list” button. Word lists can be used to check for categories of words in the script. In the example below there is a word list named “colors” which has the words blue, green and purple. Words can be added and removed from this list. Word lists are saved can be found in the “word_lists” folder.



List

colors

blue
red
purple

Word list window

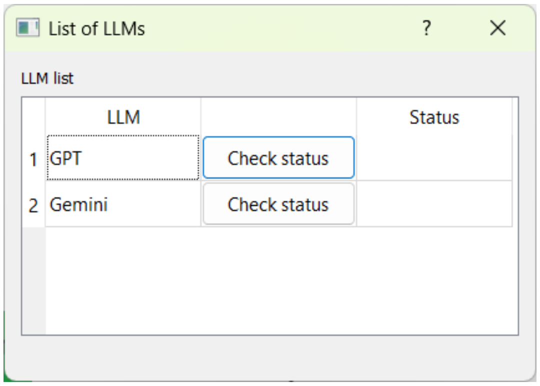
Checking Large Language Models:

LLM

On startup, ScriptBoard automatically creates a server to connect to the APIs of the GPT and Gemini large language models. To use these models in the script, ScriptBoard must be able to connect to the relevant API using a key and model version. **First enter the appropriate login details and save the**

file as *llm_login_info.txt* file in the llm folder. Use the example *llm_login_info_example.txt* file as a guide.

After running ScriptBoard, the availability of the models can be checked by clicking on the LLM button which brings up the window below:



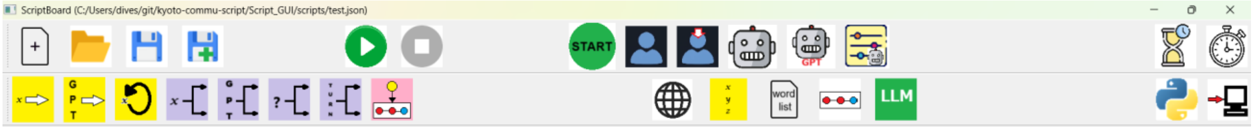
LLM list

	LLM		Status
1	GPT	Check status	
2	Gemini	Check status	

LLM availability window

To check the status of each of the LLMs, click the button. If the LLM is unavailable ScriptBoard will display the reason for this.

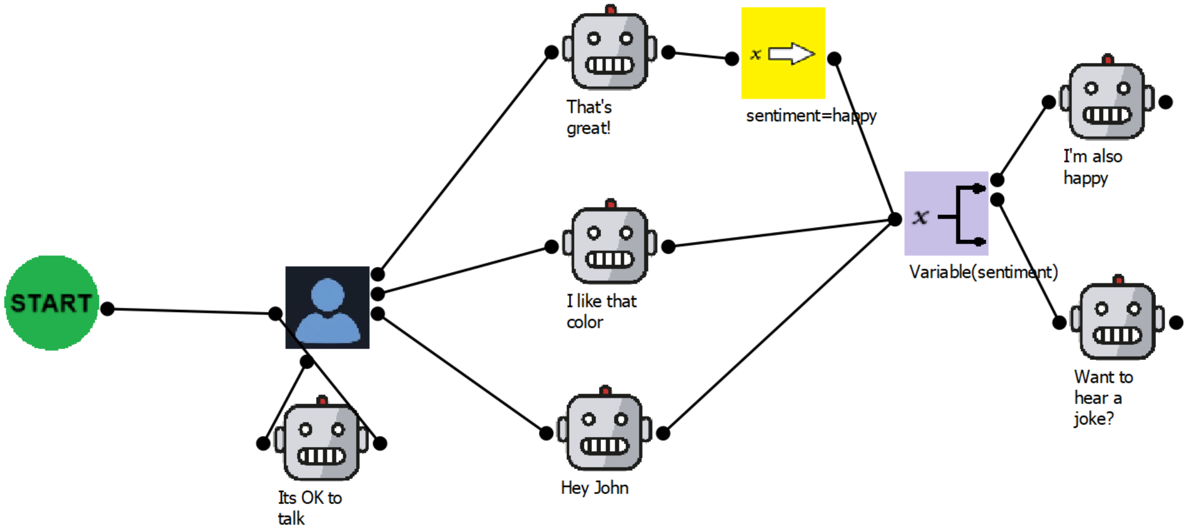
Connecting nodes:



ScriptBoard (C:/Users/dives/git/kyoto-commu-script/Script_GUI/scripts/test.json)
START
GPT
x
G P T
x
?
x y z
word list
LLM

Toolbar

To use the system, drag **nodes** from the toolbar onto the canvas. Nodes can be connected to create a flow of conversation. Nodes contain **connectors** which allow connections to other nodes. Left connectors are input connectors and right connectors are output connectors. In the below example, 1 start node, 1 human node and 3 robot nodes have been dragged on the canvas.



Example script with basic nodes

The script will begin at the Start node and wait for speech from the human. If the human is silent for 5 seconds, then the robot will say "It's OK to talk". According to the human's speech, the dialogue then flows to one of three robot nodes and will say the corresponding speech. If the robot says "That's great", the "sentiment" variable is set to "happy". This variable is then used to make a decision which flows to one of two robot nodes before ending. Any number of nodes can connect to an input connector, but an output connector can only connect to one node. The example above will be used to introduce the basic nodes in this script.

Human node:

Human nodes are entered when the script requires some input (speech) from the user. This speech is sent from the dialogue manager in the form of speech recognition results. Double clicking on a human node will open the human node window.

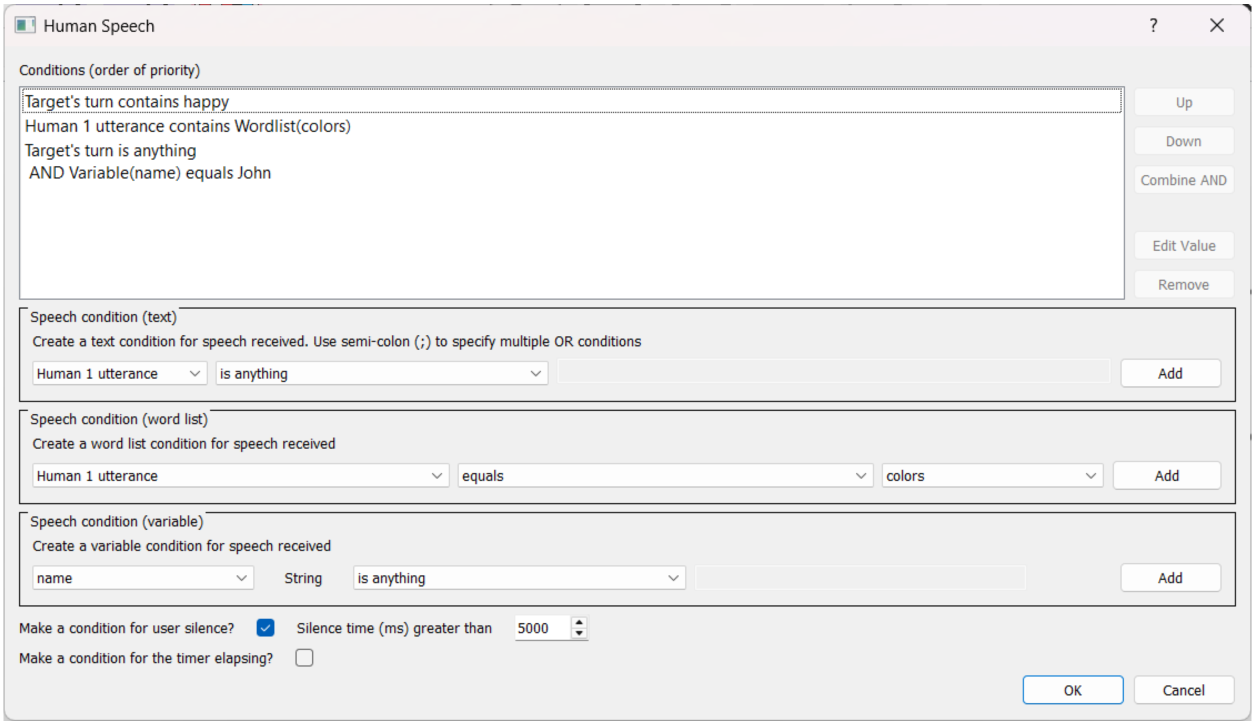
In the below example, there are three specified **conditions**.

- A text condition which will trigger if the turn of the target human contains the string “happy”.
- A word list condition which will trigger if the utterance of Human 1 contains a string found in the word list “colors”.
- A multi-condition which will trigger if the turn of the target human is anything **and** the variable “name” equals “John”.

Conditions use target and non-target utterances and turns as comparison objects. Turns are checked when a human’s **turn has ended**, while utterances are checked whenever a **new ASR result has been received**.

A range of string comparators can be chosen including equals, contains and starts with.

The conditions are set in the order of priority. In the example below, if both 1st and 3rd conditions are met (e.g. the target human says “I’m happy” and the variable “name” is John), only the 1st condition is triggered.



Human Speech

Conditions (order of priority)

Target's turn contains happy
Human 1 utterance contains Wordlist(colors)
Target's turn is anything
AND Variable(name) equals John

Speech condition (text)

Create a text condition for speech received. Use semi-colon (;) to specify multiple OR conditions

Human 1 utterance | is anything

Speech condition (word list)

Create a word list condition for speech received

Human 1 utterance | equals | colors

Speech condition (variable)

Create a variable condition for speech received

name | String | is anything

Make a condition for user silence? ☑ Silence time (ms) greater than 5000

Make a condition for the timer elapsing? □

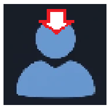
Human node window

When the “OK” button is clicked, three connectors will be created corresponding to the conditions. Nodes can then be connected to them. Therefore, the example contains the following three dialogue branches:

- If the target’s turn contains happy, the system will say “That’s great!”
- If the utterance of Human 1 contains a word in the wordlist “colors”, the system will say “I like that color”.
- If the target’s turn is anything and the variable “name” equals “John, the system will say “Hey John” since John is the value of the “name” variable.

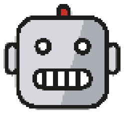
User silence is also set in this example. If a silence message is received which is greater than 5 seconds, the robot will say “It’s OK to talk”. **Silence is known only through messages from the dialogue manager and is not calculated by ScriptBoard. It is recommended to send silence messages only during the OFFER_TO_HUMAN turn state to avoid interrupting the human while they are speaking.**

Human Target node:

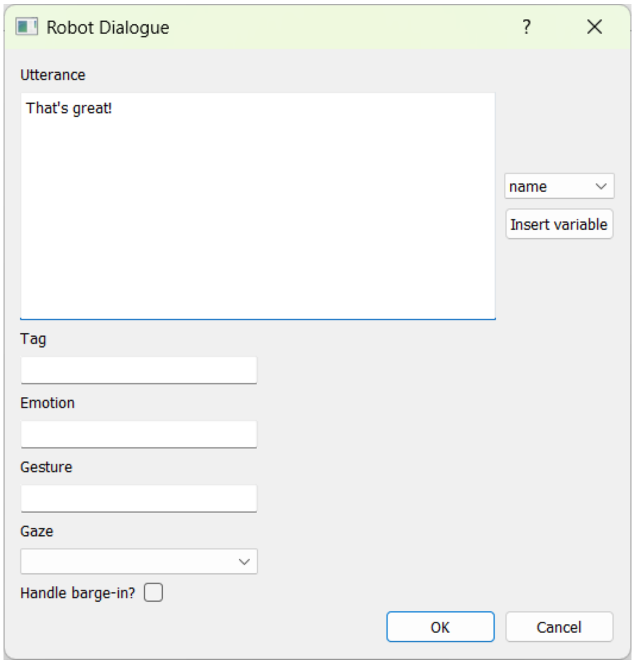


This node allows to set a specific human as a “target” during the script so that specific conditions can be set for multi-party interactions. The list of targets is the same as those set in the Environment window.

Robot node:



Robot nodes are entered when the system will say something. Double clicking on a robot node will open the robot node window.

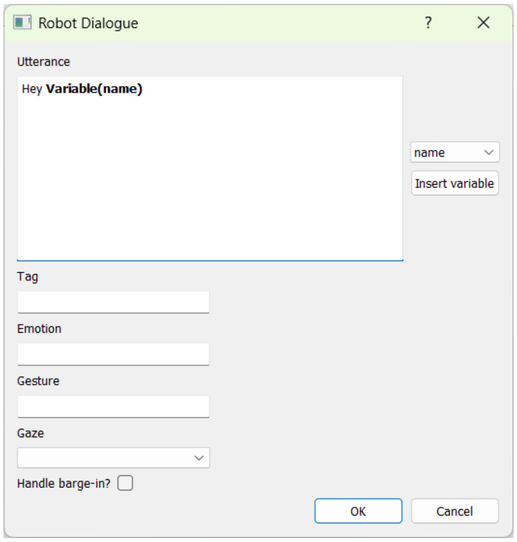


That's great!

Robot node window

The robot will say whatever text is in the Utterance window. It is also possible to add other information such as a text-to-speech tag, emotion, gesture, and if the robot should gaze at a particular human. All this information will be added to the message sent to the dialogue manager.

It is also possible to use a variable value in the utterance text by choosing a variable and clicking the Insert variable button. An example of this functionality for one of the robot nodes is shown below. In this case, the variable “name” is used directly in the utterance text and will be said by the robot. Because the value of the “name” variable has been set as “John”, the robot will say “Hey John”. Only String and Int variables can be used in this way.



Utterance: Hey **Variable(name)**

name

Tag:

Emotion:

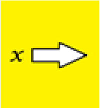
Gesture:

Gaze:

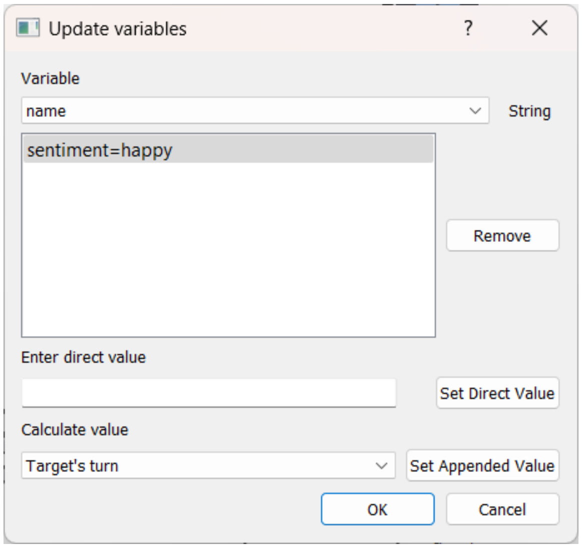
Handle barge-in? ☐

Robot node window with a variable value used in an utterance

Variable update node:



This node is used to update variables. In the above example, if the robot says "That's great" then it enters this node. Double-clicking on the node opens the window below:



Variable: name String

sentiment=happy

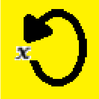
Enter direct value:

Calculate value: Target's turn

Variable update window

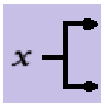
The “Variable” combo box allows the selection of a variable to update. Variables can either be set directly, set as a concatenation with another variable value (in the case of a String), or calculated (in the case of an Int or Float). In this case the value of the variable “sentiment” is set to “happy”. Note that only nodes which have initially been set in the variable window can be updated.

Variable reset node:

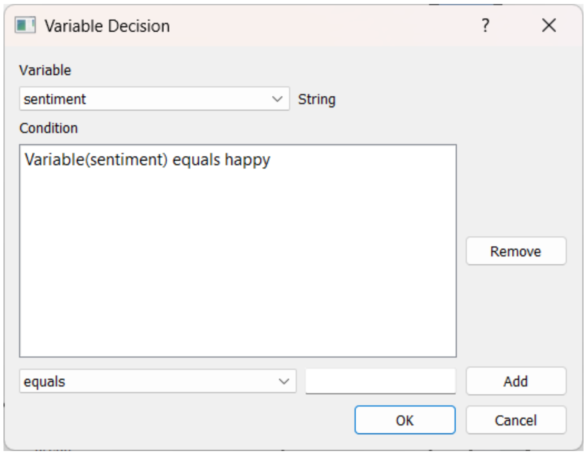


This node resets all the variables to their initial values.

Variable decision node:



This node decides on the dialogue flow based on the value of a variable. In the above example, the decision is related to the value of the “sentiment” variable and connects to two output robot nodes. Double-clicking on the node brings up the window below:



Variable Decision

Variable: sentiment String

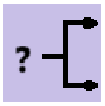
Condition: Variable(sentiment) equals happy

equals

Variable decision window

This node checks if the “sentiment” variable is equal to “happy”. If this is true, the robot says “I’m also happy”. For any other value of “sentiment” the robot says “Want to hear a joke?”.

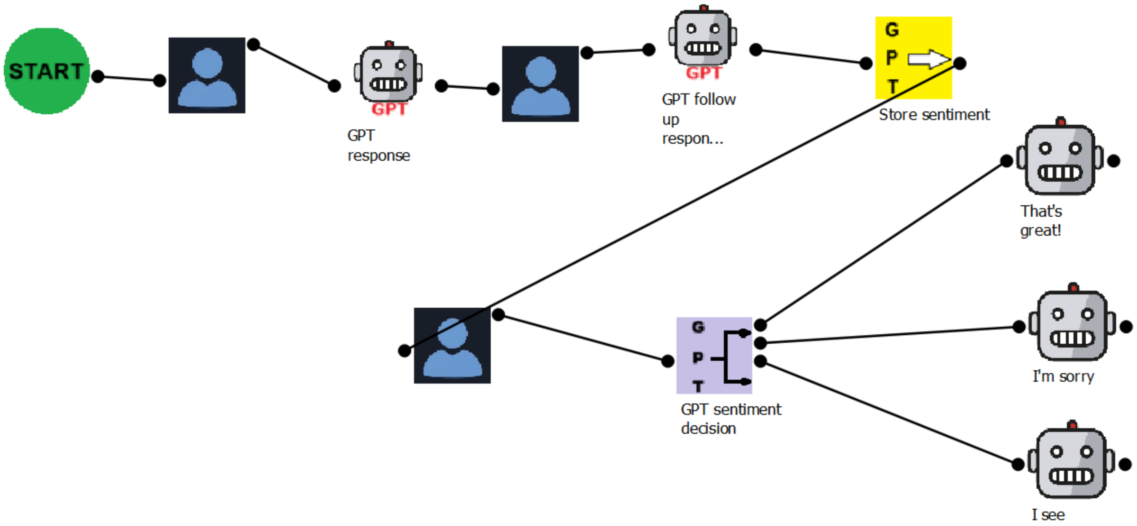
Random decision node:



This node chooses a dialog path randomly, each path being equally likely. Any number of output connectors can be used with this node.

Using LLM Nodes:

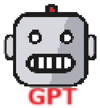
LLMs are a powerful tool for creating scripts by reducing the need for manual language processing and response generation. The script below shows an example of a script using LLMs. We set one human as a target, set one variable named “sentiment” with an initial empty value and assume that the condition for exiting the human node is “Target’s turn is anything”.



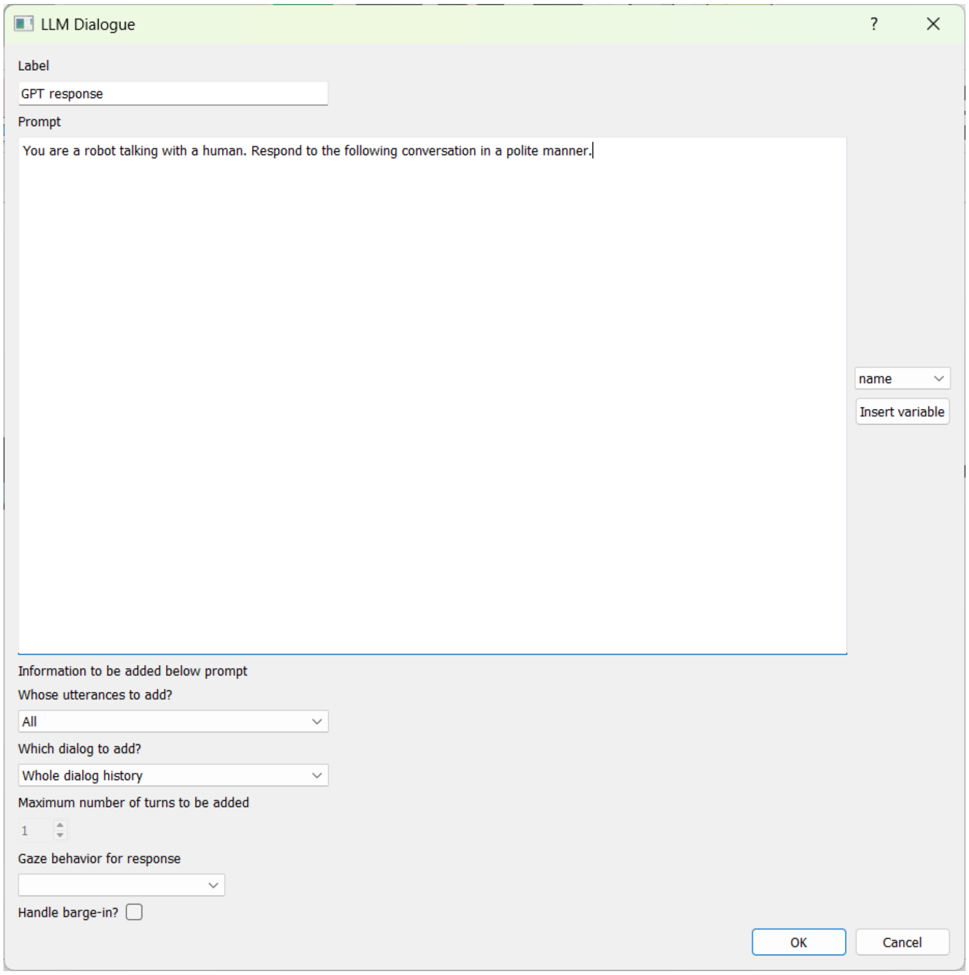
Example of script using LLM functions

1. The human says something and the robot follows up with a GPT response
2. Same as 1 but this time GPT also estimates the sentiment of the conversation and stores it into the “sentiment” variable that was initialized previously.
3. The human continues the talk and GPT also estimates the sentiment of the conversation. This time the robot response is manually set and is dependent on the output of GPT.

Robot LLM response nodes:



LLMs can be used to generate a system response instead of manual entry. This can be done by connecting a Robot LLM response node. **Right-clicking this node on the toolbar allows the selection of GPT or Gemini LLM response generation**. Double-clicking this node on the canvas brings up the following window:



Label

GPT response

Prompt

You are a robot talking with a human. Respond to the following conversation in a polite manner.

Information to be added below prompt

Whose utterances to add?

All

Which dialog to add?

Whole dialog history

Maximum number of turns to be added

1

Gaze behavior for response

Handle barge-in? □

Robot LLM response window

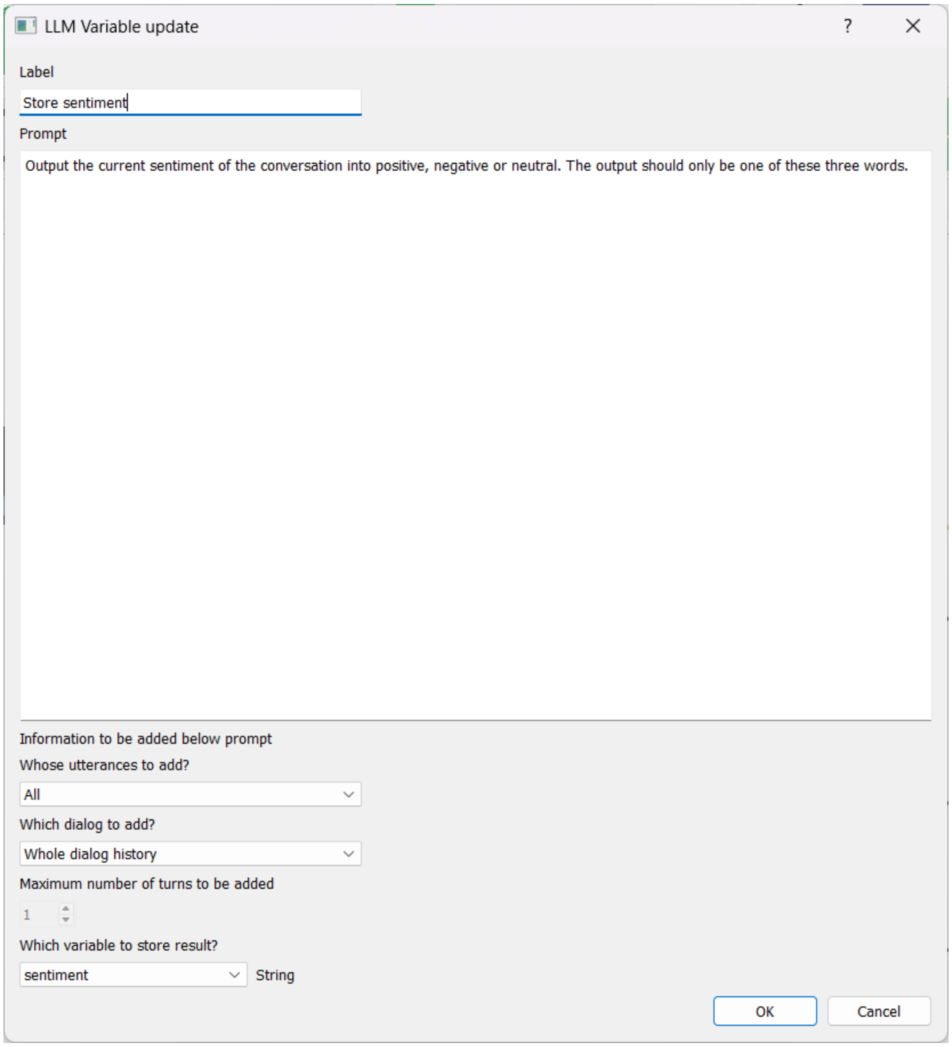
The prompt for the LLM can be entered in the Prompt window. If conversation history needs to be added to the prompt, it can be done by selecting information such as which utterances to use and how much dialogue history should be included. In the above example, the prompt will add the entire dialogue history for all participants (i.e. human(s) and robot).

ScriptBoard will send this information to the LLM and the generated response will be sent to the dialogue manager.

Robot LLM variable update nodes:

G
P ⇨
T

LLMs can also be used to update variables in a ScriptBoard scenario. Double-clicking this node will open the window below:



LLM Variable update

Label

Store sentiment

Prompt

Output the current sentiment of the conversation into positive, negative or neutral. The output should only be one of these three words.

Information to be added below prompt

Whose utterances to add?

All

Which dialog to add?

Whole dialog history

Maximum number of turns to be added

1

Which variable to store result?

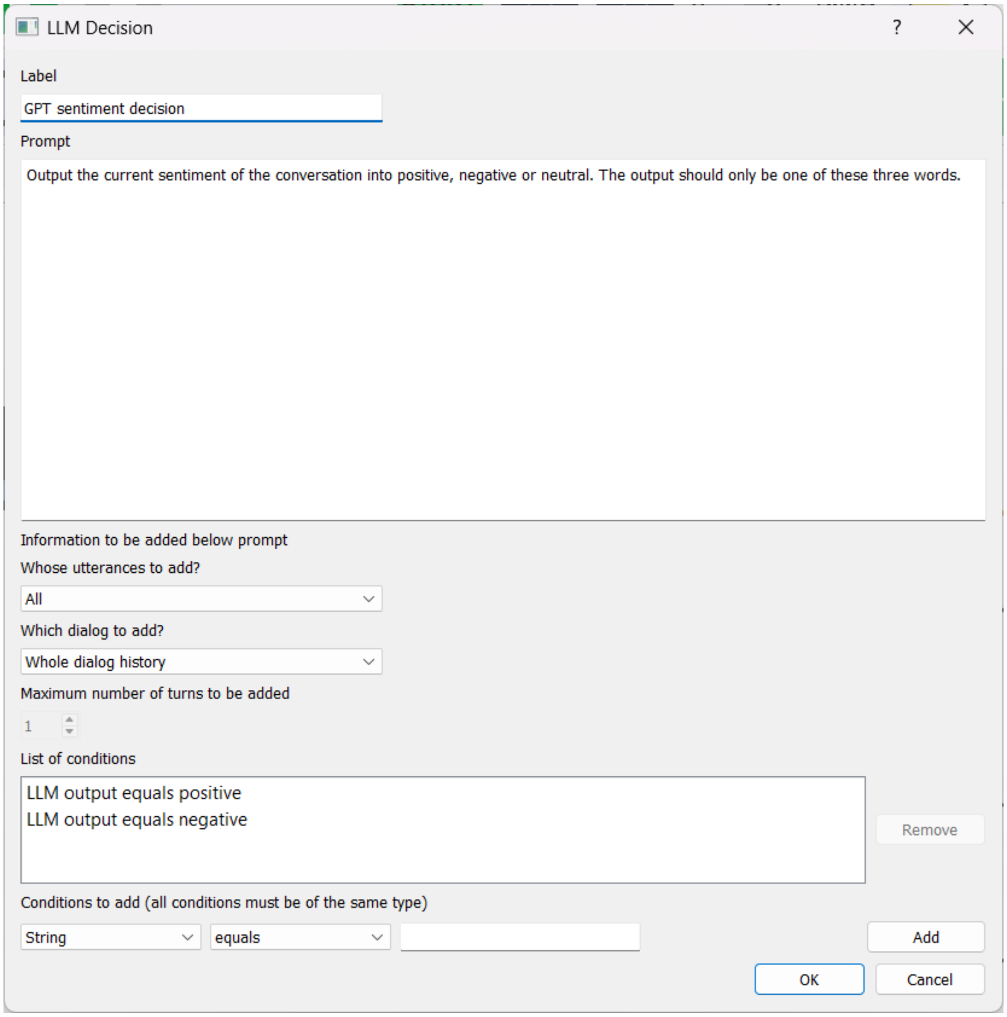
sentiment String

Robot LLM variable update window

This is much the same as the Robot LLM response node. In this example, the LLM is provided with the dialogue history and is asked to output the sentiment into one of three values. The result of the LLM response will be stored in the variable named “sentiment”, but is not used in this example.

Robot LLM decision node:

LLMs can also be used to make decisions to branch the dialogue into different dialogue flows. Double-clicking on this node will bring up the window below:



Label

GPT sentiment decision

Prompt

Output the current sentiment of the conversation into positive, negative or neutral. The output should only be one of these three words.

Information to be added below prompt

Whose utterances to add?

All

Which dialog to add?

Whole dialog history

Maximum number of turns to be added

1

List of conditions

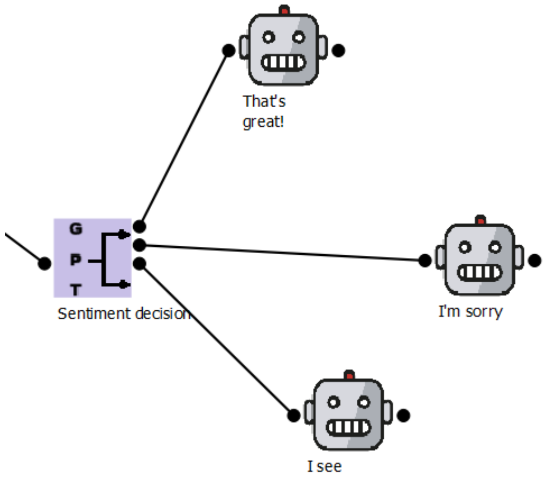
LLM output equals positive
LLM output equals negative

Conditions to add (all conditions must be of the same type)

String equals

Robot LLM variable decision window

This example is similar to the Robot LLM variable update node, except that instead of storing the output into a variable, it is used to directly make a branching decision. Conditions can be added by entering the appropriate condition to the List of conditions, which will create an output connector for each, plus an additional “other” connector. In the above example, there are two conditions: the LLM output equals “positive” and the LLM output equals “negative”. Once these are added, the node on the canvas is also updated.

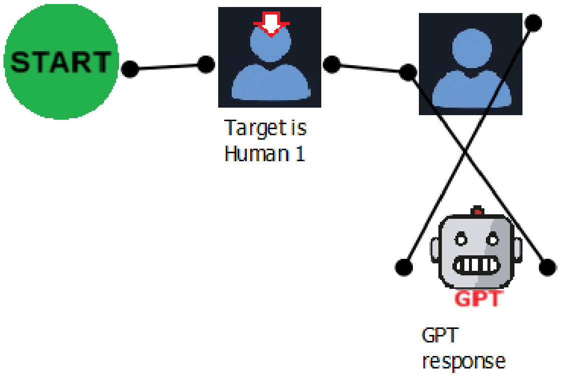


LLM decision branching into three robot nodes

If the LLM outputs positive, the system says “That’s great”, if it is negative the system says “I’m sorry” and for any other output the system says “I see”.

A simple ChatGPT script:

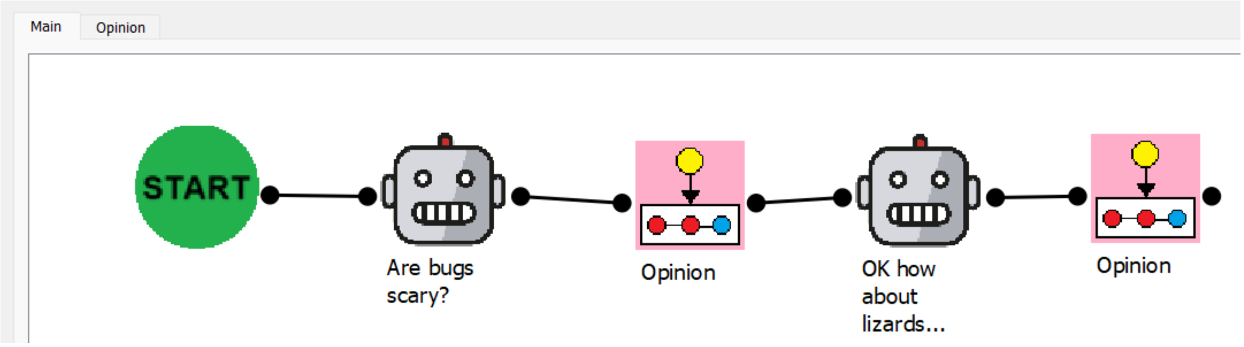
A script where the user talks with ChatGPT is shown below. The human node condition is “Target’s turn is anything” and the GPT prompt is left blank.



Basic GPT chatbot script

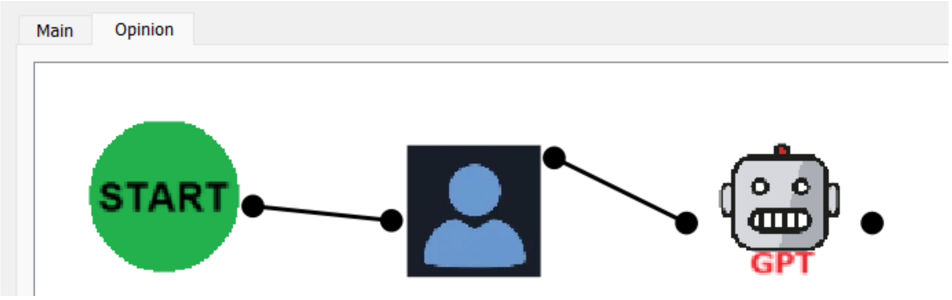
Subsequences:

Subsequences allow the script to be divided and reused. The script below is an example of the use of subsequences. Every script has a Main sequence, which can be seen in the Main tab. This tab contains two Subsequence icons named Opinion.



Main tab with subsequence nodes

There is also a tab named Opinion where the dialogue flow of the subsequence is set. In this tab, the following script is used, where a human speaks and then is responded to by GPT.



Opinion tab showing subsequence

The overall flow of this script is:

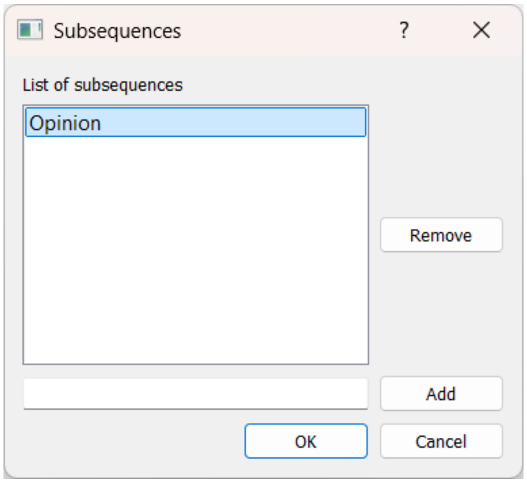
1. Robot asks “Are bugs scary?”
2. Opinion subsequence is entered, which waits for human speech and then uses with a GPT response
3. Opinion subsequence is exited, robot asks “OK how about lizards?”

4. Opinion subsequence is entered, which waits for human speech and then uses with a GPT response

A subsequence can be used multiple times and will exit when there are no more nodes left to process.

Adding and removing subsequences:

Clicking on this button opens the subsequences window which lists all subsequences. Subsequences can also be added or removed. If a subsequence is added a new tab will appear with the name of the subsequence. Like the main sequence, a subsequence must have a Start node to identify its entry point.



Subsequences

List of subsequences

Opinion

Subsequence list window

Subsequence node:

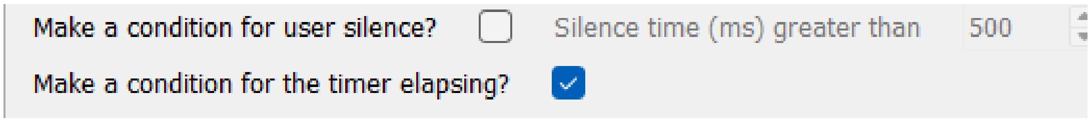
The subsequence node can be dragged from the toolbar. When it is double-clicked, a list of subsequences will appear which can be selected. When the script enters this node, the subsequence will be executed.

Wait node:

When the script enters this node it will pause for a specified amount of time.

Timer node:

This node sets a timer for a specified amount of time in seconds. It is used in conjunction with the Human node. In the Human node window, check the box named “Make a condition for the timer elapsing?”. This creates a connector at the bottom of the human node. When this timer elapses, it will then branch from this connector.

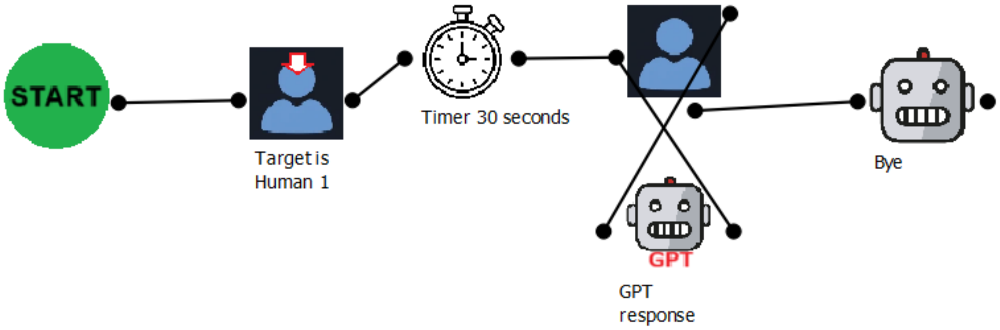


Make a condition for user silence? □ Silence time (ms) greater than 500

Make a condition for the timer elapsing? ☑

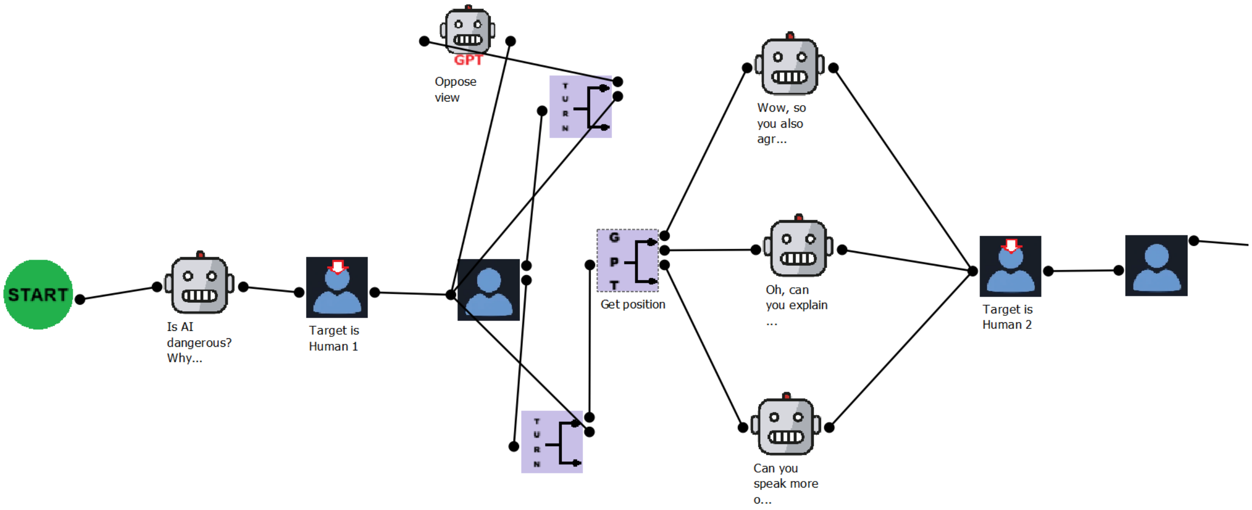
Timer checkbox in human node window

In the below GPT chat example, a timer has been set for 30 seconds. The human and robot will talk with each other for 30 seconds and then the robot will say “Bye”.



Multiparty interactions:

Multiparty scripts can be written where there are two or more human users. In this case one human can be designated as a target and different responses can be set depending on who was the previous speaker. The below script shows an example of an interaction with a robot and two humans (IDs 1 and 2).

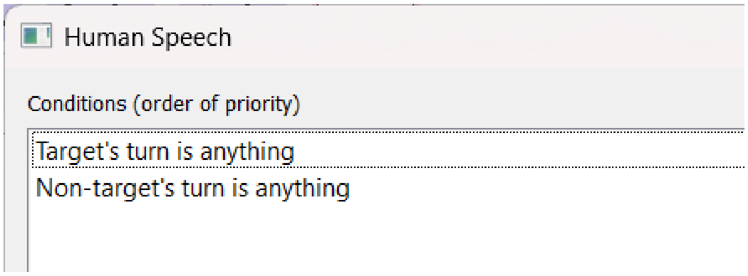


Multiparty interaction example

1. The robot asks if AI is dangerous and sets the target as Human 1
2. When Human 1 (target) ends their turn there are two possibilities
    a. The system is ready to begin their turn. In this case GPT generates a response and the conversation continues.
    b. Human 2 has started their turn. In this case, the robot stays silent and the conversation continues.
3. When Human 2 (the non-target) ends their turn there are two possibilities

a. The system is ready to begin their turn. In this case GPT outputs whether the position of Human 2 is in agreement with Human 1. Depending on this result, three possible robot responses can be used. After this response, Human 2 becomes the target.
b. Human 1 has started their turn. In this case, the robot stays silent and the conversation continues.

The Human node has two simple conditions – whether the target or non-target human has finished their turn. These correspond to the different branches from the human node



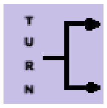
Human Speech

Conditions (order of priority)

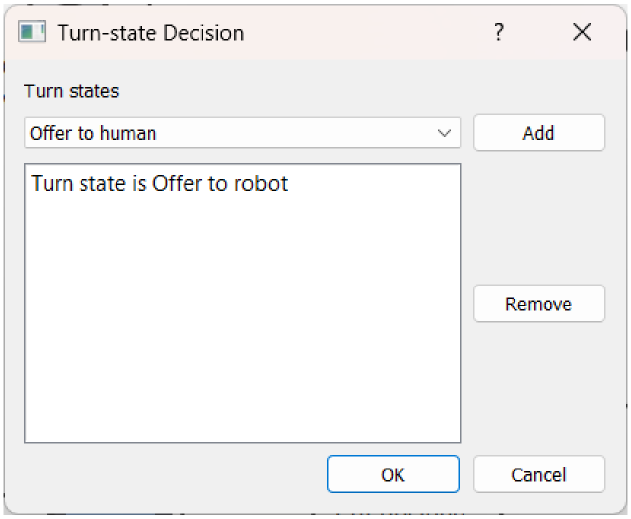
Target's turn is anything
Non-target's turn is anything

Human node conditions in the multi-party interaction

Turn decision node:



This node is mainly used for multiparty interactions, when there is a need to distinguish between a human->human turn switch and a human->robot turn switch. Double-clicking this node brings up the window below:



Turn-state Decision

Turn states

Offer to human

Turn state is Offer to robot

Turn-state decision window

Four states (human, robot, offer to human, offer to robot) can be added as conditions. A common use case is to check the turn state when a human turn has ended and waiting for a system response (turn state = offer to robot) or if the turn state has ended and another human has taken the turn (turn state = human). In the example the turn state is checked. If it is an offer to robot, then the robot should generate a response. In any other state, the conversation continues.

By specifying different behaviors based on the turn state and speakers ScriptBoard can flexibly manage interactions with multiple humans. Note that the turn state updates must come from the dialogue manager for this approach to be effective.

Advanced Nodes:

Python script node:

When the script enters this node it will execute the selected Python script. Python scripts must be placed in the “functions” folder and contain a “run” function as the entry point taking the argument “processor”. For example:

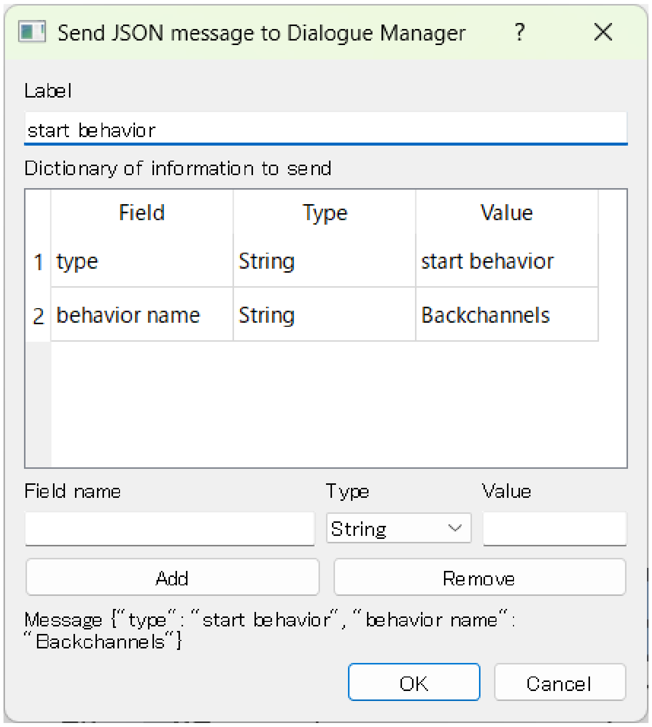
```
def run(processor):

    print(“Hello”)
```

The dictionary containing the variables in the interaction can be accessed through `processor.variable_dict`. This node can be used if there is functionality which cannot be achieved with any of ScriptBoard’s regular nodes.

Python script node:

This node allows the script to send a JSON message to the dialogue manager. When this node is double-clicked the following window will appear:



Label

start behavior

Dictionary of information to send

	Field	Type	Value
1	type	String	start behavior
2	behavior name	String	Backchannels

Field name

Type: String

Value

Message {“type”: “start behavior”, “behavior name”: “Backchannels”}

Through this interface a dictionary will be created, in which keys and values may be added and converted into a JSON message. The processing of the message itself should be handled by the dialogue manager. For consistency, it is recommended to always have a “type” field denoting the type of message.