Adding knowledge to SecondEGO assistant with MindMap diagrams

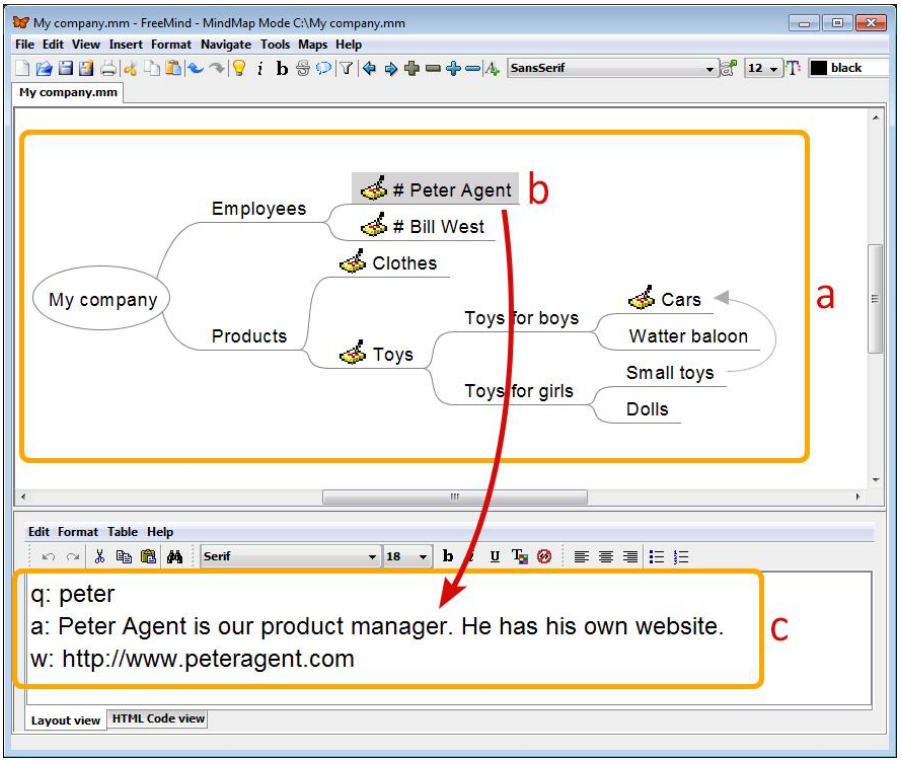
We can add knowledge to SecondEGO assistant in different ways. For fast, small and simple procedures, it is best to use "patterns" (see Dashboard/Train/Patterns) or "frequently asked questions and answers" (see Dashboard/Train/FAQ). For all other cases, it is recommended to use MindMap diagrams. That way, you have a nice overview of your project. It is especially useful if you have a large MindMap.

SecondEGO supports diagram importing from FreeMind, which is a free, GNU General Public Licensed mind mapping application.

INSTRUCTIONS FOR PREPARING MINDMAP DIAGRAMS IN FREEMIND

1 General

A diagram (a) consists of logically connected nodes. They usually go from general to specific (Example: "My company" > "Products" > "Toys" > "Toys for boys" > "Cars").



Every node has a name (b) and most have content (c) that usually specifies the keywords for user question recognition and a suitable response.

2 Diagram building

We have to combine nodes logically to form a whole.

To build a diagram we usually use these commands:

- To select a node, click on it or move toward it using arrow keys.

- To edit the node's name, click F2 or just start typing the new name.
- Delete a node with the "Delete" key.
- To add a node on the same level (New sibling node) press "Enter".
- To add a node on a new level (New child node) press "Insert".
- You can move through nodes with arrow keys.
- To organize a diagram, simply use your mouse and place it on the left of the node, click on the circle that appears and then drag the node to the desired location. One may also use Ctrl + arrow keys for this.
- To connect desired nodes, use the function "Add graphical link".
- To remove this connection, right-click on it and click on "Remove arrow link".

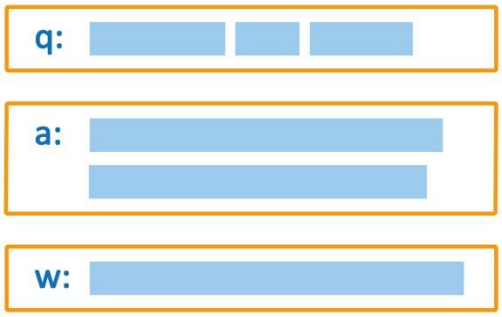
For more information and suitable commands, see FreeMind help.

MindMap diagrams have to be saved in the default format (.mm).

3 Instructions for creating nodes

Every node needs a unique name.

Content is divided into sections. Section "q" (questions) consists of keywords we use to recognize the question. Section "a" (answers) contains possible answers and suitable responses to that question. Section "w" (web page) contains a link to the desired URL (redirects automatically). Sections must follow this order (q, a, w). A node can have many "q" sections and only one section "a" and one section "w". Section labels (q, a, w) have to be the first letter (no spaces, characters...) in a row, followed by a colon (:).



If a node does not have content, it is considered a connecting node, used for connection purposes to create a logically connected diagram.

3.1 Section "q"

This section contains keywords the virtual assistant uses to recognize the user's question. Keywords can be words or phrases. To separate keywords, use a comma, semicolon or place it in a new row (operators OR). If SecondEGO has the entered words in his language database, it can recognize inflected forms of those words. Otherwise, the other word forms have to be entered by hand.

In MindMaps, SecondEGO does not differentiate uppercase from lowercase letters.

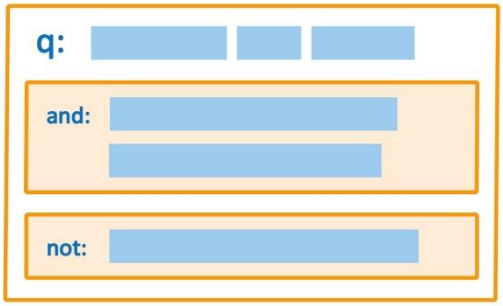
The keyword sequence is arbitrary.

Example:

q: auto, automobile; car, motorcar	=	q: CAR Motorcars Automobile Auto

We can use a subsection AND (max. five times) at the end of section "q" and a subsection NOT (max. once). Subsection AND has to come before NOT.

Subsection labels (and, not) have to be the first word (no spaces, characters...) in a row, followed by a colon (:).



Example:

```
q: keyword1; keyword2
and: keyword3
and: keyword4; keyword5
not: keyword6
```

The logic behind OR, AND, NOT operators is identical to the logic in the patterns (see DASHBOARD/Train/Patterns/Help), more on this topic in chapter 5.

If we need more AND/OR/NOT combinations of keywords, we can add more "q" sections.

Example:

```
q: why
and: use, useful, have
and: secondego, ego, agent, assistant

q: how
and: can, does, will, would, could
and: secondego, ego, agent, assistant
and: help
```

We usually enter keywords into the node, but not always. If we do, the virtual assistant can recognize the question of the user and answer accordingly. This type of node is also called an "entry point."

A node without keywords cannot recognize the user's question, so this node is only accessible by clicking on a "choice" the virtual assistant gives (see chapter 3.5 Choices).

3.2 Section "a"

This section contains answers the virtual assistant can provide. We can have more than one answer. Separate them with a semicolon or place them in a new row (operators OR). We can enter them as we please because the virtual assistant will randomly pick one.

Example:

a: Of course!; Certainly! Absolutely!	=	a: Absolutely! Certainly! Of course!

We can also make our answers bold [B], italic [I] and [URL] (link to website or mail).

Examples:
[B]Big Ben[/B] is in [I]London[/I].
[B][I]Big Ben[/I][/B] is in London.
[URL="http://en.wikipedia.org/wiki/Italy"]Italy[/URL] is in Europe.
[URL=http://en.wikipedia.org/wiki/Italy]Italy[/URL] is in Europe.
Our email address is [URL="mailto:info@secondego.com"]info@secondego.com[/URL].
Our email address is [URL=mailto:info@secondego.com]info@secondego.com[/URL].

3.2.1 Answer in multiple rows

If we wish to present the default operators OR as a part of the answer (more than one row, or include a semicolon), we have to write the entire answer between and .

Example (five-row answer):

```
a:
<p>
A car can be:
red,
green,
blue,
white.
</p>
```

3.3 Section "w"

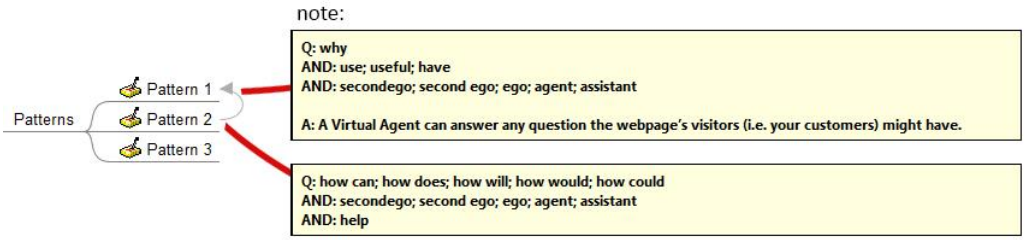
This section contains a link to a website. The virtual assistant will redirect to it from the current site. To open a link in a new tab, we have to use "wn" instead of "w". The label "w" is usually used for web pages that have the virtual assistant on basic domain and the label "wn" for other websites in new window.

Examples:
w: http://www. myxpage.com (redirects from current page)

wn: http://www. wikipedia.org (opens the link in a new tab)

3.4 Using an existing response

Answers often repeat themselves, especially in larger diagrams. To avoid entering it multiple times, assign the reply from a node that already contains it. To do this, we have to use the function "Add graphical link".



We can also use this if we wish to supplement our answer with another one from a different node.

3.5 Choices

The virtual assistant can offer the user answers prepared in advance (choices). By doing this, we create a guided path. The virtual assistant then guides the user along this path.

To make a node a choice, put "#" in front of the name (see picture above). The user sees this as one of the options (# is hidden to the user). It is not necessary to put a space between the # and word, but it is recommended.

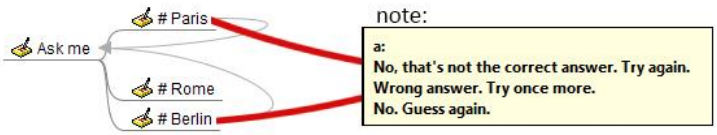
The example above (what the user sees using SecondEGO):

If a node has a section "a" and no section "q" it is only accessible by clicking on a choice that leads to it.

If a node does not have a # in front of the name, it is an ordinary node, and the names are not displayed. In this case, the name is just for orientation purposes (important in big diagrams).

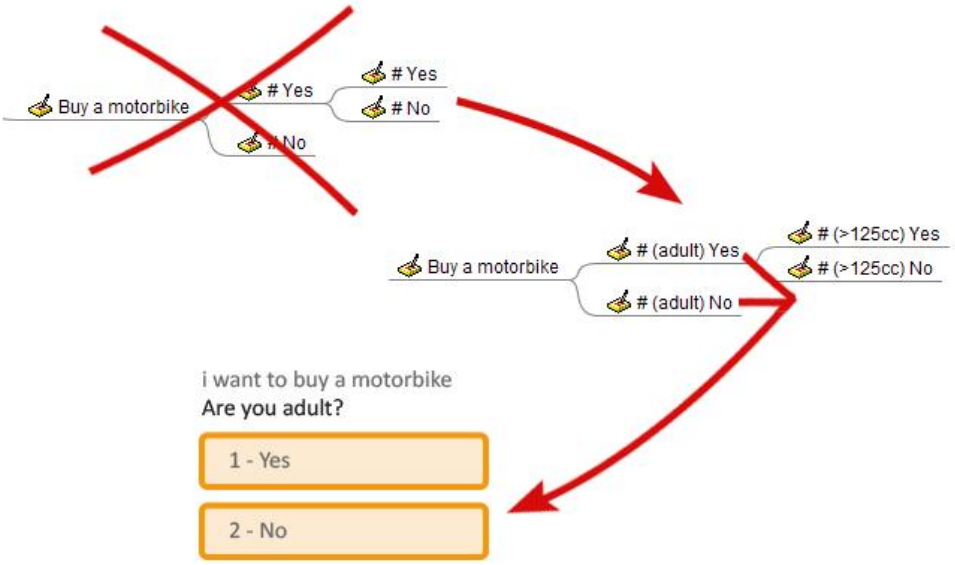
3.5.1 Choices with a loop

Sometimes we wish to return the user to the previous node if he answers incorrectly for example. This connection is like the one described in chapter 3.4 (Using an existing response).



3.5.2 Unique choice name

Every node needs a unique name and the same goes if the node is a choice. But many times we have the same answers for different questions (Yes/No questions). In this case, we have to add a description in parentheses.



If the parentheses come after the # sign, it is considered a part of the name (choice) but is not shown to the user.

Using parentheses also gives us a better overview of our diagram.

3.5.3 Overlapping choices

A group of options can repeat itself if we have a complex diagram. To avoid writing everything twice, we can redirect to an existing part of the chart. To redirect connect your node with an additional connection (function "Add graphical link").

Connection to a node with a # tag is treated as a choice (if it is not the only one) and differs from additional connections to nodes without # tags, which means using an existing response or complementing the original one (see chapter 3.4). Both connections are mutually exclusive, and priority is given to the choice.

3.6 Variables

We can use variables to save useful information we obtain during the conversation (between SecondEGO and the user).

A variable's name consists of:

- type of variable: »int« (integer) or »str« (string)
- variable's name: random string (letters of the English alphabet, numbers, underscore)

Variable names are case sensitive and can't start with a number. If the variable name starts with an underscore, it's a global variable, visible in other MindMap modules.

Examples:
»int age« - local number »age«
»int _number1« - global number »_number1«
»str name« - local string »name«
»str _country« - global string »_country«

You can also use global system variables, but changing them is not recommended as it can compromise the assistant:
»str _se_agent_name« - virtual assistant's name (can't be empty)
»int _se_agent_type« - virtual assistant's type (0 = formal, 1 = friendly)
»int _se_agent_gender« - virtual assistant's gender (0 = unknown, 1 = male, 2 = female)
»int _se_agent_age« - virtual assistant's age (0 = unknown)
»str _se_agent_place« - virtual assistant's place of residence (empty string = unknown)
»str _se_agent_country« - virtual assistant's country of residence (empty string = unknown)

3.6.1 Assigning variables

Assigning variable values is done at the end of every answer in the section "a". If we have multiple answers that assign variables, only the chosen one is executed. If a reply contains more variable assignments, all are carried out.

Assigning has the following form:

{variable_name operator number_or_string}

Assigning operators:
= assign to
+= add to
-= subtract from

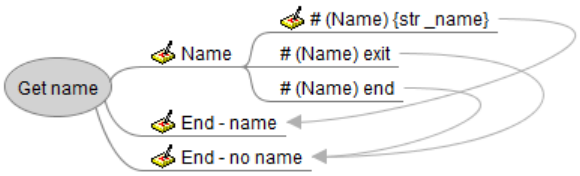
We can only use "=" if we have a string variable, and we have to put the word(s) in quotation marks (example: "France").

If we have integer variables, we can only assign new integers to that variable (… -2, -1, 0, 1, 2 ...).

a: Age: 20 {int _age = 20}	a: Country: France {str _country = "France"} {int country = 1}

Assigning variables from user input

If we wish to save the user's input, we have to use a choice with a string name.



We used {str_name} as the name of an option. In this case, the virtual assistant does not show the option's window (like in chapter 3.5.3). Instead, the user's input is assigned to the variable (if user writes his name, {str_name} is set to what he wrote). If we offer additional options (in the picture; "exit" and "end"), they are not displayed either but are compared to the user's input. If the user writes "exit" or "end" in this case, the word is not set as the user's name and the user is redirected to a suitable diagram. There are no additional choices by default; we have to set them ourselves.

You can see how to do this in the example diagram Survey.mm.

3.6.2 Displaying variables

We can display them as part of an answer:

a: She is {int _age} years old.	a: She is {int _age} years old and she lives in {str _country}.

3.6.3 Comparing variables

This is called a condition. Conditions may additionally be used to select the appropriate response in the section "a" and the implementation of the comparison of keywords in the section "q".

Conditions have the following form:

{variable_name operator integer_or_string}

Comparing operators:
== equals

<		less than
>		more than
<>	!=	different
<=	=<	equals or less than
>=	=>	equals or more than

If we are comparing string variables we can only use "==", "<>" or "!=".

Examples: {int num == 0}, {int _counter > -2}, {str answer1 == "A"}, {str _country != "France"}

We usually use conditions in the section "a". This is used to offer the user an appropriate answer according to condition(s).

```
a:
{int num == 0} There is no hits.
{int num > 0} Number of hits: {int num}.
```

If no condition is met, the user gets an answer from another node or a knowledge module, so it is best to include an additional answer just in case:

```
a:
{int num == 0} You are wrong.
Your answers are not correct.
```

If we have more conditions for one answer, the default operator between them is AND. If the conditions are met (for more than one answer), the assistant returns the answer that met the most conditions:

```
a:
{str answer1 == "A"} You are right.
{str answer1 == "A"} {str answer2 == "C"} You are genius.
{str answer1 == "B"} You are wrong.
Your answers are not correct.
```

Although conditions are mostly used in the section "a", they can also be used in carrying out a comparison of keywords in the section "q". This comparison is only made between those keywords that meet the conditions above.

We can "catch" a node with different keywords, depending on the variable's value:

```
q:
{str _ language == "English"} hello, hi
{str _ language == "German"} hallo
```

This method can also be used to turn on/off entry points in different situations.

3.7 Functions

Instead of an answer in the section "a", we can call a function, the result of which is an appropriate response. Currently, it is possible to call two functions: "requestion" and "sendemail."

3.7.1 Function "requestion"

The "requestion" function triggers the search for new keywords that are the only parameter of this function:

<function requestion>
Parameter: keywords
</function>

If you want to transfer the information or control from a module (FAQ, Patterns, MindMap, AVO) to another one, you can use the function "requestion". The only parameter is list of keywords, which correspond to the pattern/node in the target module.

Module 1

```
q: aaa
a: <function requestion>bbb</function>
```

→

Module 2

```
q: bbb
a: The answer is bbb.
```

If (in the above example) the user enters the string "aaa", searching of string "bbb" is executed instead of answer. If this is in a second module, the answer is transferred to another module. The answer to the question of "aaa" is, therefore, "The answer is bbb.".

You can look at the requestion function in the example diagram Survey.mm.

3.7.2 Function "sendemail"

When the assistant returns an answer it can also send a message to specific email (in case we want to forward the information to the operator). To do this, we can call the function "sendemail" which has five parameters:

<function sendemail>
1. parameter: e-mail of the operator
2. parameter: subject
3. parameter: message content, which can contain variables
4. parameter: answer if the message was sent
5. parameter: answer if the message was not sent
</function>

Example:

```
a:
<function sendemail>
admin@company.com
Results of the survey no. 12
<p>
Age: {int _age}
Answers:
1: {str _ans1}
2: {str _ans2}
3: {str _ans3}
</p>
```

```
<p>
Thank you for your answers.
They have been submitted.
</p>
Thank you for your answers.
</function>
```

4 Managing MindMap diagrams

The MindMap diagrams (file .mm) can be managed in the tab Dashboard/Train/MindMap. Managing MindMap diagrams is easy. They can be imported, activated/deactivated, deleted or downloaded.

Link for importing is on the top left. Click on it if you want to import a MindMap diagram.

a) Activate/deactivate MindMap diagram (only in case of successful import)
b) Name of the MindMap diagram
c) Activate/deactivate fuzzy search
d) Number of nodes
e) Date and time of importation
f) Status:

- OK (Import successful without warnings or errors.)
- Warnings (Import successful but with warnings. Check warnings by clicking on the link.)
- Errors (Import failed because of syntax errors in the MindMap diagram. Check the errors by clicking on the link.)
- Invalid MM (MindMap diagram is not in the .mm format.)

g) Download MindMap diagram
h) Deleting the diagram. Before deleting check if you have a backup!

5 Identifying answers

In addition to using an exact match method to determine the keywords, we can also use fuzzy matching. Exact matching has higher priority than fuzzy matching, but we can combine these two approaches as well.

5.1 Exact Match

The exact match method is the default method for comparing keywords to user questions; this cannot be deactivated.

5.1.1 Keywords

A node "captures" what the user asked if this issue contains at least one of these keywords in the section "q" (semicolon or new row between keywords is an OR operator). We can also use phrases:

```
q: sports car, bicycle, dog
```

"Is your **sports car** blue?" – true
"Does Simon have a **bicycle** – true
"A sports **car**, a **dog** or a **bicycle**?" – true
"Did you see my **dogs**?" – true
"Do you have a sports **bicycle** and a car?" – true
"Do you have a fast car?" – false

If a language module for the chosen language exists, it can identify all forms of a word (dog/dogs in English).

Phrases can also be whole sentences:

```
q:
what is the capital city of france
do you know any capital cities
```

"**what is the capital city of france**" – true
"**What is the capital city of France**?" – true
"**do you know any capital city**" – true
"the capital city of france" – false
"I wonder...**what is the capital city of france** today?" – true

Comparing words is case-insensitive.

5.1.2 Use of additional conditions

If we use the operator AND, the user's question must contain at least one keyword from the additional condition:

```
q: where is
and: big ben, bigben
and: located
```

"where is bigben" – false
"where is bigben located" – true
"where is located big ben " – true
"where bigben is" – false
"where bigben is located" – false

5.1.3 Punctuations

If a keyword in the section "q" does not include punctuations, matching does not depend on it:

```
q: where is italy
```

"**where is italy**" – true
"**Where is Italy**" – true
"**where is italy**?" – true
"**Where is Italy**?" – true

If the keyword in the section "q" contains punctuation (with the exception of commas and semicolons that represent the operators OR), matching will be successful only if the user's question contained punctuation in the same place:

```
q: where is italy?
```

"where is italy" – false
"Where is Italy" – false
"**where is italy?**" – true
"**Where is Italy?**" – true

5.2 Fuzzy matching

Fuzzy matching is not the default method for comparing keywords to user questions, but it can be included and combined with exact matching.

We usually enable it when we use whole sentences as keywords:

```
q:
what is the capital city of france
do you know any capital cities
```

There is a slim chance the user will write that exact sentence and therefore "catch" the appropriate node:

what is the capital city of france
Paris

what is the capital of france
I have no reply!

The solution is fuzzy matching (see chapter 4.). The search engine tries to find the question and answer that is "closest" to the user's question:

what is the capital city of france
Paris

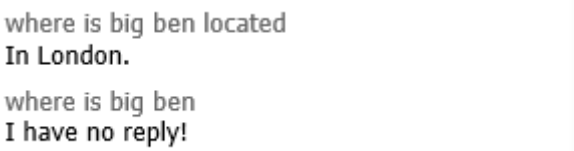
what is the capital of france
what is the capital city of france
Paris

Unlike exact matching, when fuzzy matching is enabled the question (to which the displayed answer belongs) is displayed in bold letters before the answer. The semantic meaning of the user's question may be different than what the assistant understands.

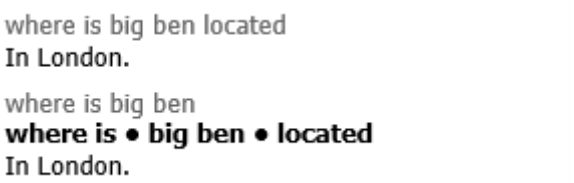
Fuzzy matching can also be activated if we have regular keywords (not whole sentences).

q: where is
and: big ben, bigben
and: located

Using exact matching (excluding any fuzzy matching) we get the following answers:



Answers we get using fuzzy matching with the same keywords:



We have to be careful what keywords we write first (AND operators) because the bold question will always contain the first word from every subsection within the section "q".

By using fuzzy matching we can, therefore, answer more questions, but we cannot be entirely sure whether the response is (always) correct. However, adding new keywords (phrases) can gradually solve these problems.