For more information, access: http://www.gilgamesh-ai.org
You can contact me at gilgamesh.ai.project@gmail.com

Gilgamesh

Artificial Intelligence Project Manual

Version 0.19 Beta

Index

Overview

This manual refers to a project made in 2013 in order to help people and programmers which intend to use the artificial intelligence code called *Gilgamesh*. The Gilgamesh is an AI (Artificial Intelligence) Project with its source code made in Java programming language, using an alternative concept, but similar to neural networks. Common users and programmers can use it in a large number of situations (which I attempted to show in this manual).

The main purpose about all this work is to contribute with other as possible, helping people with their own objectives, using the knowledge described here to achieve that. Furthermore, it is expected for programmers use the code or ideas presented here to help other people, finding solutions to problems where computers are better than humans. Originally, the Gilgamesh code has been developed to help in stock market, but it could be used in a wide range of areas. It is expected to receive some suggestions and ideas from you in order to improve it.

History

The Gilgamesh concept was created in the middle of 2013, but their concepts were thought for a long time so it was constructed based on them. All started around 2005, when a group of friends started an uncommitted AI project called Spirit Project. At that time, they made some meetings to learn and study some AI concepts, just for fun. They got good results, but around one year later the project ended up with more unsolved questions and disagreements than when it began. Since the end of Spirit project and during all those years, lots of possible solutions to the unsolved remaining problems came up. Actually, lot of solutions were created for some different concepts and implemented in Gilgamesh (it is up to you to decide if they will serve you or not). In 2013, an opportunity happens to put all the ideas together, releasing the Gilgamesh Artificial Intelligence Project (before it was called as Core Project) and since then, it is being used to predict stock market moves, with satisfactory results.

After a while and being sure about its effectiveness, it was decided to share the code as open source and create this material to help people understand what the creator had in mind. It was created some small programs applied to various areas in order to give a start point for those who wants to use them in a similar situation.

All information and code available on this manual are not an academic study about AI or its math concepts. Its intent is to be a practical work for professionals or companies to get practical results using it. For that, it was attempted to provide all kind of help, about the concepts created and about the code developed, keeping everything as simple as possible, trying to achieve not only programmers, but enthusiasts and common public in general. Most concepts in this manual reflects the creator's opinions and his ideas about AI. It has no intention to argue about mathematical models or AI concepts.

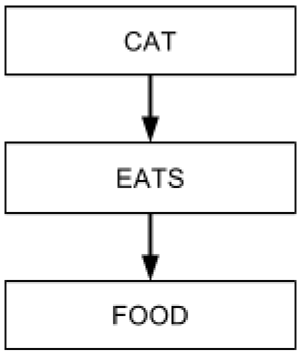
About Concepts

When we talk about AI, we always remember science fiction movies full filled with robots doing human things. However, the AI in real world is much more difficult to understand and to implement into a code.

Basically, AI is the ability of machines thinking like living things, having some kind of what we call intelligence. A lot of studies were done in this area, developing a lot of different concepts about it. In the next lines it is shown some concepts that were used to develop the Gilgamesh concepts.

Almost all the concepts used in Gilgamesh was done thinking about blocks of information connected among themselves. It was like if we would have a bunch of neurons where each one could keep a entire block of information as we know. Besides that, it was attempted to use and adjust the code to be close enough about how some psychological concepts work. In Gilgamesh's concept, if it could combine those psychological concepts with AI concepts, a good AI structure happens. That was what the project tried.

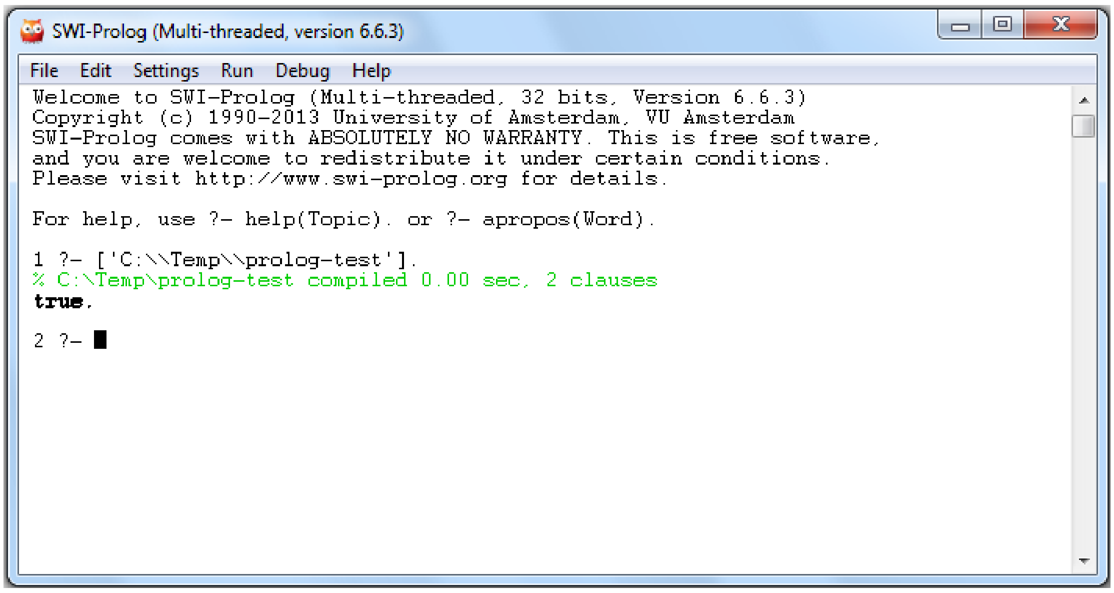
Let's start with the information blocks. As the original concept for an atom that was the smallest and indivisible part of matter, our information block (which is the small piece of information) is called atom in Gilgamesh as well (actually, this concept is the same as used in other AI programs). An atom can be a simple word, a number, a character or a group of all of them. Let's start with a simple connection among some blocks of information (atoms):



Simple enough. We have a bunch of information, connected among them to make some sense. From now on, those blocks of information with their respective connections will be called as a fact (like a compound term). If we could make the computer learn those information's with their connections, we would have some level of intelligence into a program.

One of the first things that computer professionals have in mind when the topic is AI, usually is a computer language called Prolog (that means logic programming). Prolog is one of the languages which best fits in the generic concept of AI. We can find on internet a lot of good implementations about prolog interpreters and tutorials about how to use it.
As the name says, Prolog interpreters can learn relations among blocks of information. Advanced concepts of Prolog is not cover here, but it is possible to do same job as Gilgamesh using those advanced math's in that language.

Let's see what can be done in a prolog interpreter (test was done using SWI Prolog):

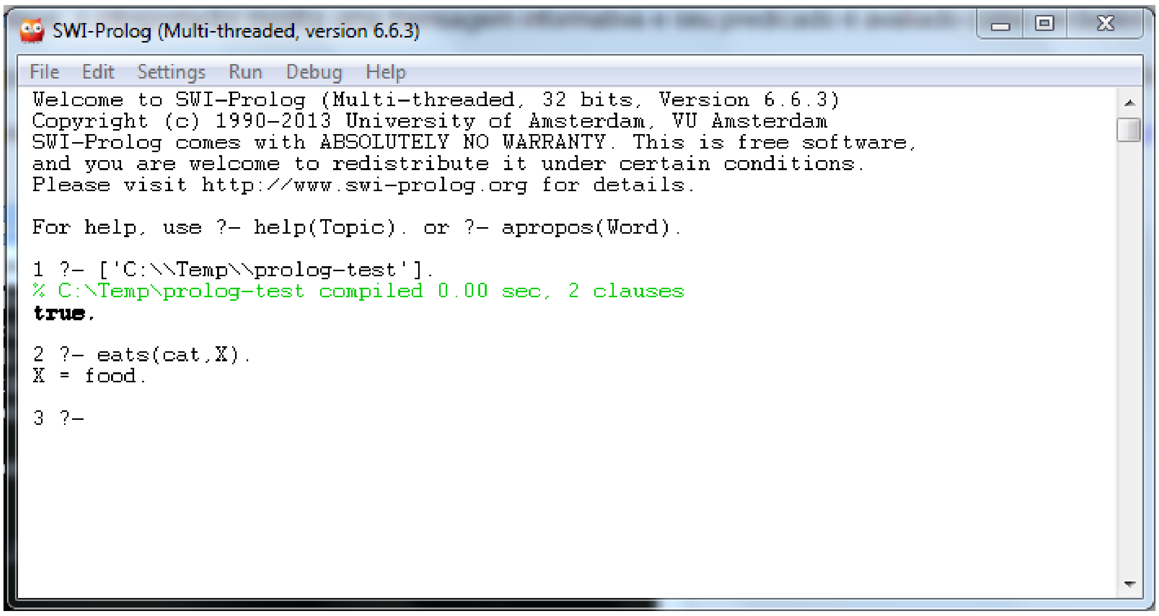

SWI-Prolog (Multi-threaded, version 6.6.3)
File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 6.6.3)
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- ['C:\\Temp\\prolog-test'].
% C:\Temp\prolog-test compiled 0.00 sec, 2 clauses
true.

2 ?-
```

It was loaded a prolog test file (this is the way that it “learns”) with the following block of information:

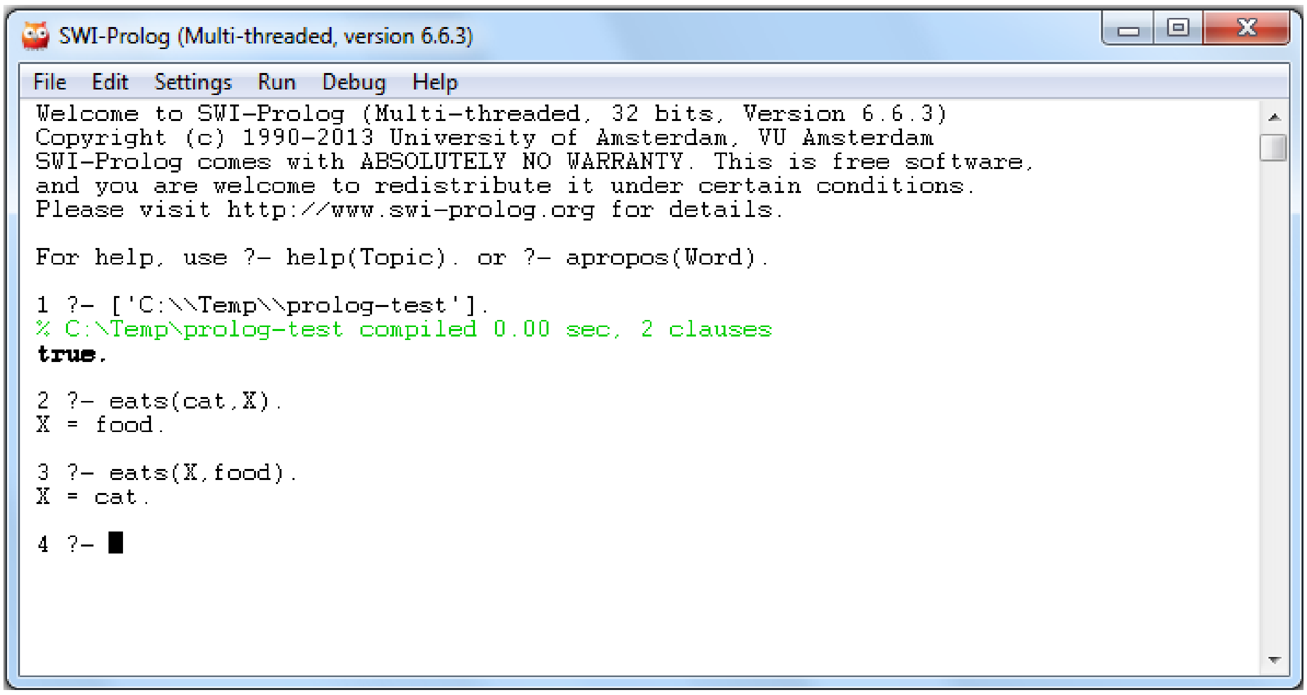
eats(cat, food).

This information tells something like “cat eats food” to prolog. Now, the program has learned about that information and its relationship. To do something useful with that information, we ask it a question in order to obtain a satisfactory answer. If we want to know something like “what does the cat eat?” would be something like:


SWI-Prolog (Multi-threaded, version 6.6.3)
File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 6.6.3)
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- ['C:\\Temp\\prolog-test'].
% C:\Temp\prolog-test compiled 0.00 sec, 2 clauses
true.

2 ?- eats(cat,X).
X = food.

3 ?-
```

We got the right answer for “X” variable: *food*. If we try to do another question like “who does eat food?”, would be something like:


SWI-Prolog (Multi-threaded, version 6.6.3)
File Edit Settings Run Debug Help

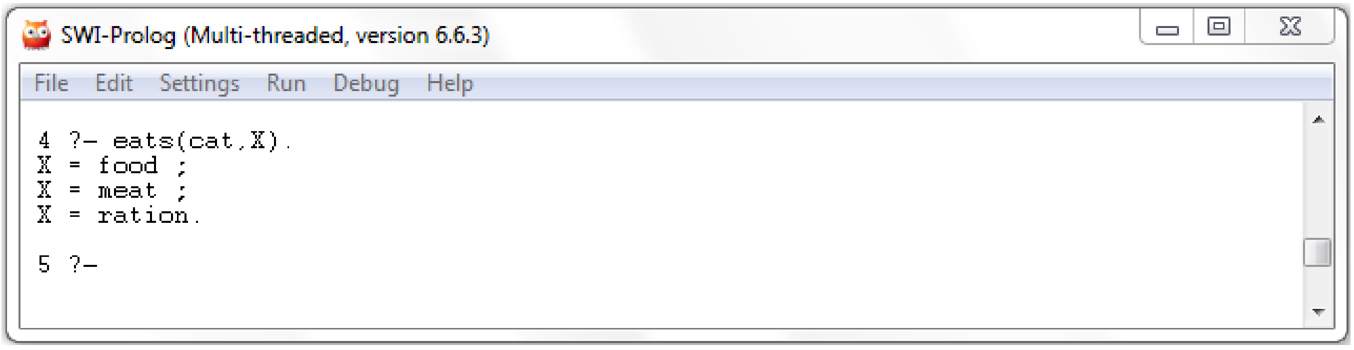
```
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 6.6.3)
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- ['C:\\Temp\\prolog-test'].
% C:\Temp\prolog-test compiled 0.00 sec, 2 clauses
true.

2 ?- eats(cat,X).
X = food.

3 ?- eats(X,food).
X = cat.

4 ?-
```

It gave us the right answer: *cat*.

This is the point where it was observed the first changed idea about Gilgamesh's concepts: the order of the answer can be changed, according your question. For a logic programming that works perfectly, but not for Gilgamesh's concept. Lets advance a little, improving our test providing more information:

```
eats(cat, food).
eats(cat, meat).
eats(cat, ration).
```

Now we have three facts, each one with a specific connection among the atoms. If we ask to Prolog what cat eats, the answer would be:


SWI-Prolog (Multi-threaded, version 6.6.3)
File Edit Settings Run Debug Help

```
4 ?- eats(cat,X).
X = food ;
X = meat ;
X = ration.

5 ?-
```

In the same order we provided the information. If you enter a hundred times the information “eats(cat, meat)” for example, each one would be a valid answer, in the order you entered them (considering no math optimizations). Questions were raised up, like: if we decide to use prolog in its simple way, those results would be in right order? If we want to know which one is truthful, the first would be the right answer? The text above was just to give an idea about how AI in computer works (only if you are not aware about it) in order to compare with Gilgamesh's concepts.

One of the most important concept in Gilgamesh, it is the *memory print* as follows: it was thought how about electricity runs among neurons through their synapses and how they end up working as intelligence. Every conductor of electricity has a property called resistance (Ω) which is the opposition (or difficulty) that the electricity has to pass through it. For synapses, is not different except for the fact that they have the ability to constantly adapt themselves according the electricity received by senses, making a print on its complex connection web.

For Gilgamesh, prints that happens often should *dig or sculpt* an easily path through synapses, being the most *probable* answer among those paths, unless that one changes. Another important point about neurons is that each one has thousands of connections to other neurons (like different algorithms used by neural networks). According those concepts, would not be possible in a simple way to have answers in the way like prolog gave us: it should have a most *probable* answer. To get the most probable answer, we should use something different, like fuzzy logic.

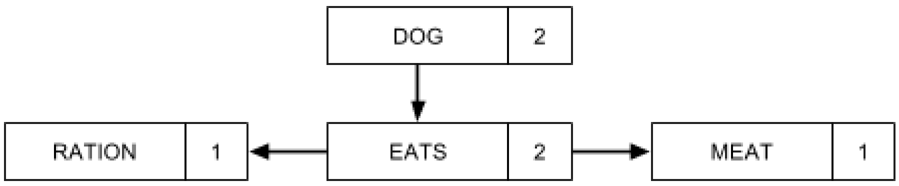
The main point on fuzzy logic is that you don't have only logic answers (like true or false), but you have a best answer from a bunch of information, choosing it according different entries. For example, if we think of how tall is someone, results could be very different if we ask different people to answer that. For someone, a tall guy could be a person 2.20m tall but for you, no. In this case, bringing the fuzzy logic concept to Gilgamesh could result in a better answer, according the entries and their quantity and not according their order.

Despite the fuzzy logic’s concept that says to use a number from 0 to 1 allowing to work with few data, the Gilgamesh project ended up matching that idea with a normal distribution curve (Gaussian), which made more sense and it seemed to have a closer behavior according the way which neurons and synapses work.

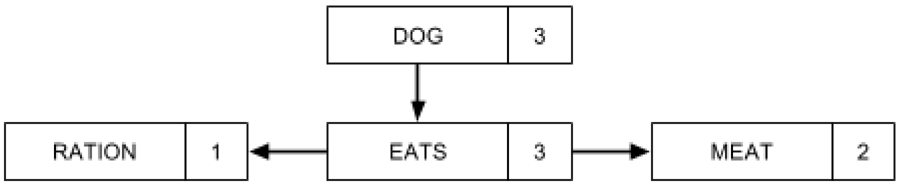
In this point, Gilgamesh's structure was taking shape. Below there is a improved example to explain what was wanted from code. Let's consider the information below:

DOG EATS RATION
DOG EATS MEAT

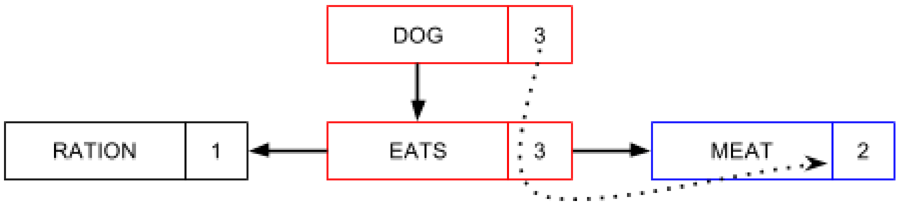
We could connect our atoms like that, indicating how many times each one occurred, using a counter that would work to do *memory prints*, or in fact, using a *memory force* for each atom which creates a *path* (for that, the atom "dog" and “eats” cannot be repeated):



At this point it is not possible to determine a better answer yet, except for the entering order. Now, what would happen if we enter one same information again? Let's suppose that one more time an information is entered like “DOG EATS MEAT” in our model. What would happen?



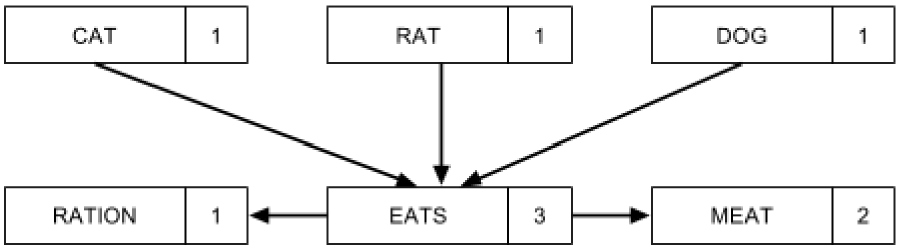
In this case, part about answer order problem could be solved because if we have a program which uses the forces to answer, it could follow the *path* through them to find a best answer. Then if we ask it “what does the dog eat?”, the most probable answer would be “MEAT”:



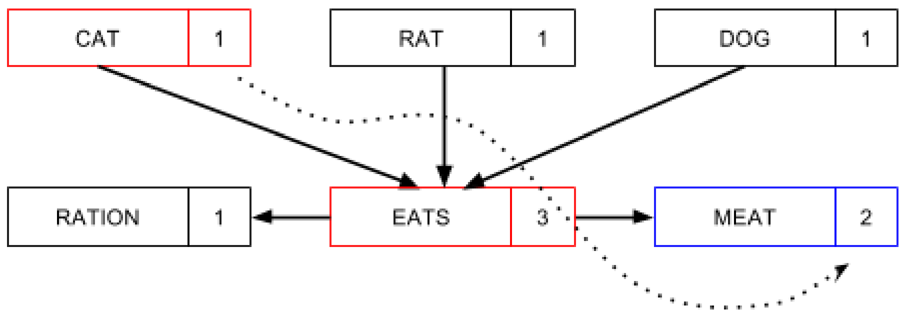
For the first time during the Gilgamesh's development, it sounded as a good solution. But there was a problem in that concept when information is improved with crossed facts. Let's consider now, the following facts:

DOG EATS MEAT
RAT EATS MEAT
CAT EATS RATION

Our model would be something like:



If we ask to this model “what does the cat eat?”, it would give us the wrong answer “MEAT”, because cat has never eaten meat:



And why that answer would be wrong, according Gilgamesh concepts? Because we have never said to the program that there is a fact called “CAT EATS **MEAT**”: the only information that it has is that “CAT EATS **RATION**”. In this case, did not make sense to the project implementing functions to control errors, so that model could not attend the project expectations.

The project retrieves some psychological concepts to solve problems (especially about Ivan Pavlov which did great discoveries about behavior and mind conditioning). The psychological behaviorism concepts were used to teach the code how it should work, applying the concepts of reinforcement and punishment for facts.

Combining some of those knowledge, a bunch of conclusions and decisions were taken about how the code should work. Thoughts of all those concepts ended up in a modeling structure that tries to represent all of them working together, with the following premises:

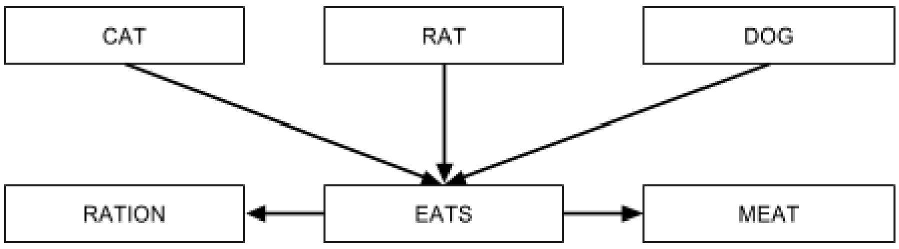
- ✓ The code should keep the answer order, according the fact was informed. It could not have crossed information answers and it should use the most probable answer;
- ✓ If there is no most probable answer, then the code should try to find one, usually, the last entered or reducing the scope of search;
- ✓ The code should behavior like psychological behaviorism concepts: with more positive (reinforcements) or negative (punishment) forces. It should follow this rule to give an best probable answer (analog to fuzzy logic);
- ✓ All forces used to control the atoms should represent the electricity way running through synapses;
- ✓ It should be easy to use.

How It Works

The main focus of Gilgamesh is about connections. The first thing the code does is to register all **atoms** (information blocks) used by the program. The second thing is about the right connection among atoms which is a **fact**. The solution found in this case was to create a link of each atom to the correspondent fact. If you are a programmer, to create a Gilgamesh memory for a specific context, you have to create a Gilgamesh Core. The Gilgamesh Core is the code that do all the main work. To the entries:

DOG EATS MEAT
RAT EATS MEAT
CAT EATS RATION

We have the following atoms with their connections:



The first thing that Gilgamesh Core does is to include all atoms into a collection, like below:

Atom ID	Atom
1	DOG
2	EATS
3	MEAT
4	RAT
5	CAT
6	RATION

In a second step, Gilgamesh Core controls the fact order according its entrance, creating a collection of facts:

Fact ID	**Fact (refers to atom ID)**	**What does it really means**
1	1 -> 2 -> 3	DOG EATS MEAT
2	4 -> 2 -> 3	RAT EATS MEAT
3	5 -> 2 -> 6	CAT EATS RATION

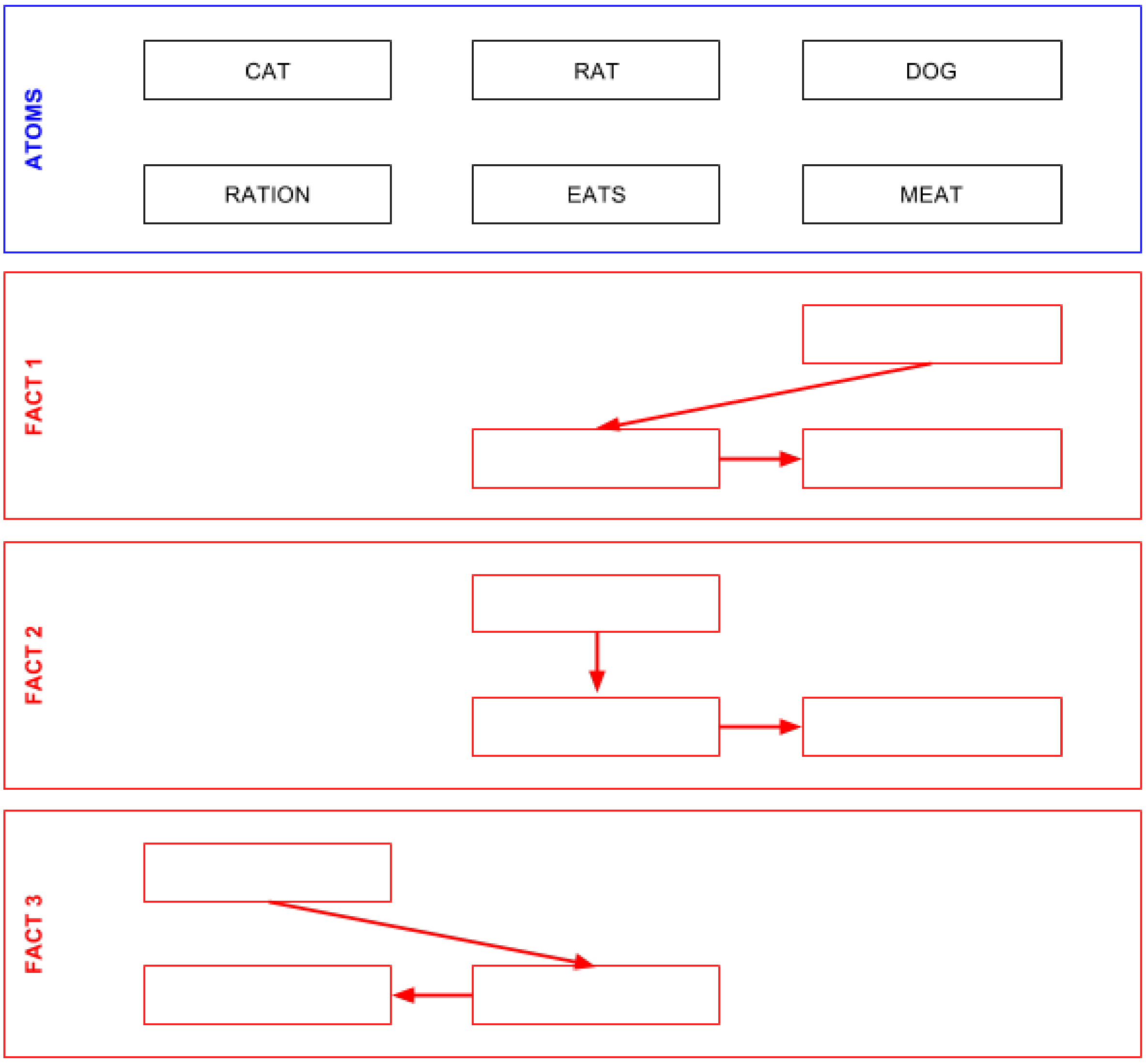
As one last step, the Gilgamesh Core links each atom with its respective facts, setting a memory force (which is by default, just a counter of how many times a fact happens) to that fact. Actually, this force represents the link force which the atom has for that specific fact. By definition, positive values are better forces. Notice in the example that, as all the facts are different from each other, the force will be always 1. So the final atom table is like below:

Atom ID	**Atom**	**[Fact, Force]**
1	DOG	[1, 1]
2	EATS	[1, 1], [2, 1], [3, 1]
3	MEAT	[1, 1], [2, 1]
4	RAT	[2, 1]
5	CAT	[3, 1]
6	RATION	[3, 1]

With this configuration, if we ask to the Gilgamesh something like “What does the cat eat?” like in the example before, now we get the right answer that would be “RATION”. This happens because when the Gilgamesh Core goes to atoms collection and finds two matching information of three in a fact, it will use that fact to answer the question. In this case, the fact ID number 3, points to two of three atoms, in the right order:

CAT EATS? = Fact 3 = CAT EATS RATION (5 -> 2 -> 6)

This is one of the solutions that Gilgamesh provides. If we represents the atoms and the facts in a model, it would be something like that:



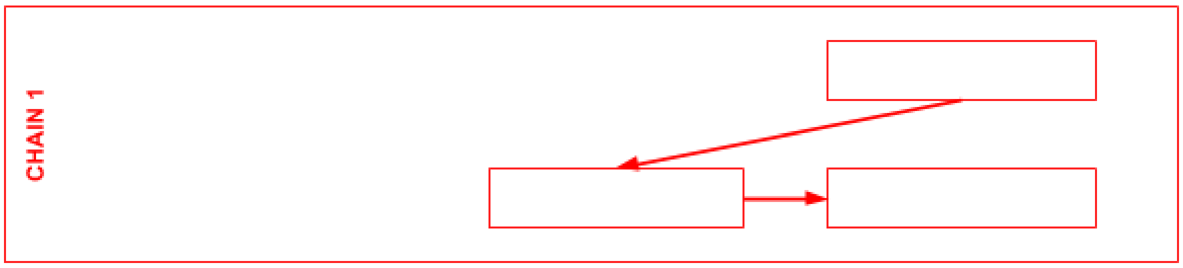
Facts in Gilgamesh tends to represent memory prints in a virtual brain. Now, what if we ask to the code “who does eat meat?”. We would have two possible answers:

RAT EATS MEAT
DOG EATS MEAT

In this case, both answers according Gilgamesh concept are right and we could have the output like above. This happens because we had just one occurrence for each those facts. Now, let's simulate the psychological behaviorism like if our code were a dog brain. What does it happens if we make positive reinforcements? What if, we reinforce one of our facts above again?

DOG EATS MEAT

Then we would have the following memory print twice:



And then we would have this new configuration for atoms:

Atom ID	Atom	[Fact, Force]
1	DOG	[1, 2]
2	EATS	[1, 2], [2, 1], [3, 1]
3	MEAT	[1, 2], [2, 1]
4	RAT	[2, 1]
5	CAT	[3, 1]
6	RATION	[3, 1]

Note that the fact above has occurred twice. Basically, the force says that we have more occurrences of that fact than others. So now, if we ask to the code “Who does eat meat?”, then the right answer (DOG) would be related with the most probable fact known: DOG EATS MEAT.

This is the way that Gilgamesh works. Inside the code, the force works like a sum of numbers which gives you the most probable answer. This happens because the facts are stored according its atoms combination. The answers you get are results of an intrinsic statistical behavior from the code.

But, what if you would like that Gilgamesh give you an answer anyway? Let's say, if you ask something "Does the dog eat ration?". Gilgamesh would not give you an answer since it has not a fact with exact all those asked atoms (DOG, EATS, RATION) what it means that it does not know that fact. For that, there is another solution into code that you can ask it to match a best answer. On this way, the answer would be:

DOG EATS MEAT

To reach this answer, the best that it can do is to calculate question atoms among those that have higher forces to a fact. In this case, the atoms “DOG” and “EATS” have a better force to the fact “DOG EATS MEAT”, so, that fact is chosen. This behavior brings more results than the original one, where the fact has all atoms. In a general way, when Gilgamesh looks for an answer, it can find in two ways:

- ✓ it can bring the higher fact with exact matching for all question atoms or
- ✓ it can bring the higher fact with approximated matching for most question atoms

You can decide which one you are going to use into your solution. However, to simplify things, there is a method inside the code which tries to find the best fact according first rule and, if it does not find an answer, it tries to find the answer using the second method. This combined method will provide you an answer almost all the time. You can choose if you will use the first, second or combined methods.

The Gilgamesh answer concept was created to not provide any answer without a known fact (different for e.g. from Neural Networks which can reach an unexpected new answer). Still today, this make sense according project concepts for a simple reason: it is thought that the creative human process is a result of introspective thinking with things you already know: you cannot create something from nothing (you have things in your mind, even if those things were already there in your brain, due to genetics - you could think about most impressive and creative thing ever: if you break your idea in parts, you will realize that you have joined small parts of a lot of things you already knew).

Gilgamesh versus Neural Networks

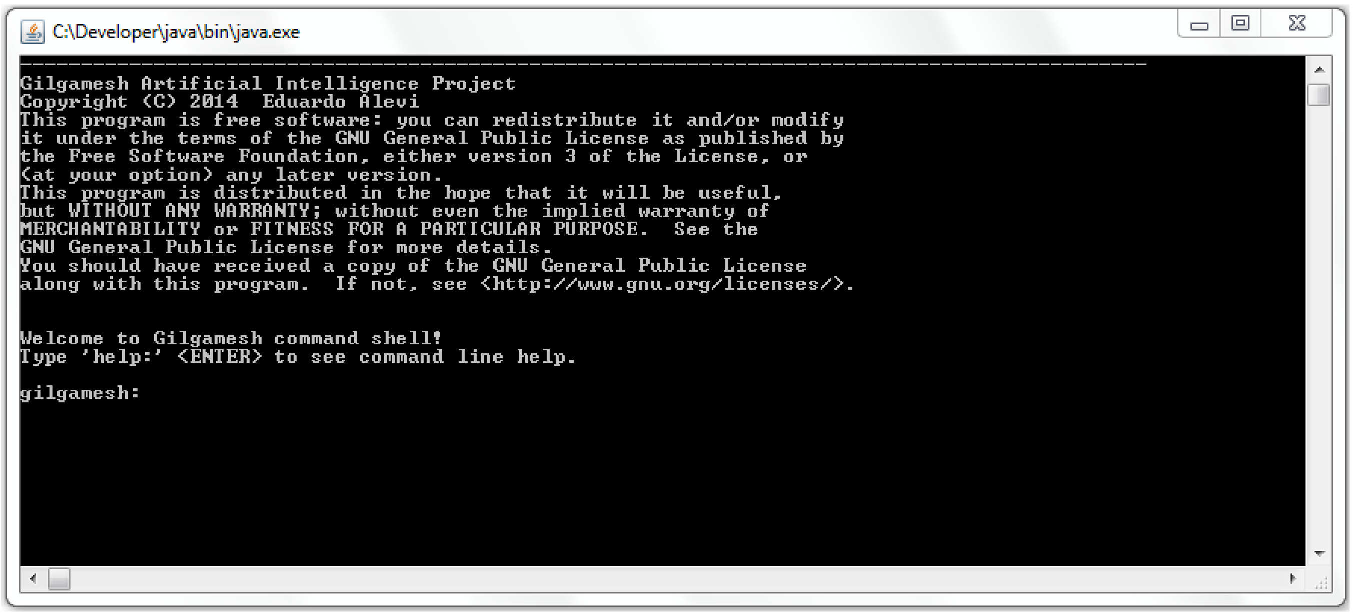
Usually when you create a neural network (like Neuroph or DeepLearning4J), you have to inform a fixed number of neurons for input and output values, besides providing a normalized value for them, even if they are not relevant for you. The main difference In Gilgamesh is that you can provide a variable equivalent input of neurons and obtain a variable equivalent output of neurons as well. This is usefull when you don't have a specific value for a neuron when a fact or situation happens or when you expect to have an unknow output size. The values for Gilgamesh doesn't need to be normalized as well because it matches them into memory.

How to Use It - The Command Shell

The Command Shell is best way for non programmers use Gilgamesh. Remember that Gilgamesh is case sensitive and consequently, its command shell too. Gilgamesh is done in Java so you have to have the Java Runtime installed in your computer (you can download it at www.java.com). You can run Gilgamesh in these ways:

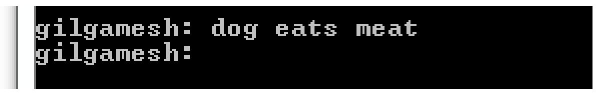
- ✓ executing one of the files “gilgamesh.bat” (Windows) or “gilgamesh.sh” (Linux)
- ✓ running the command shell “java -jar gilgamesh.jar”

After that, you will see a window like this, with a prompt waiting your commands:



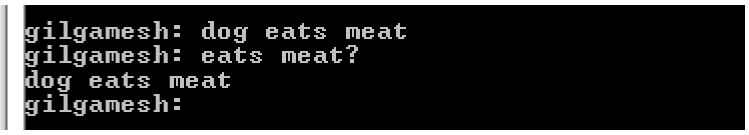
```
------------------------------------------------------------------------------------------
Gilgamesh Artificial Intelligence Project
Copyright (C) 2014  Eduardo Alevi
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.
You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.


Welcome to Gilgamesh command shell!
Type 'help:' <ENTER> to see command line help.

gilgamesh:
```

The Gilgamesh starts with no memory so you have to teach it entering some facts. To teach it, you have to type your atoms (block of information) in a raw way. Gilgamesh is not programmed to understanding natural language what means if you want to enter the information “My dog eats meat”, you should enter only the essential information you need (dog eats meat). It can accept commands too if you put ; (semicolon) at the end or the middle of a typed line. It will understand as a command and its respective parameters (if it has any). If the line ends with a ? (question mark), then it will understand as a question for a single answer. Any other words typed in a line, separated with blank spaces, will be considered as a fact (atoms connected among them). If you want to consider a longer atom (with more than one word), then put those words between ' and ‘ (single quotes). So, to enter a fact, you can simply type the words you want, followed by ENTER key:



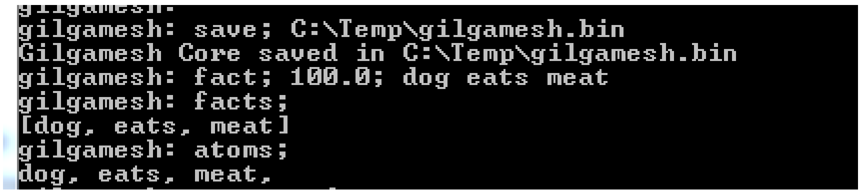
```
gilgamesh: dog eats meat
gilgamesh:
```

If you want to ask something to program, type the atoms you want with "?" at end and press ENTER, like:



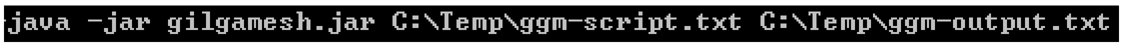
```
gilgamesh: dog eats meat
gilgamesh: eats meat?
dog eats meat
gilgamesh:
```

The atoms order in a question does not matter. For commands, they are executed by its name followed by ";" and theirs parameters if they have. E.g.:



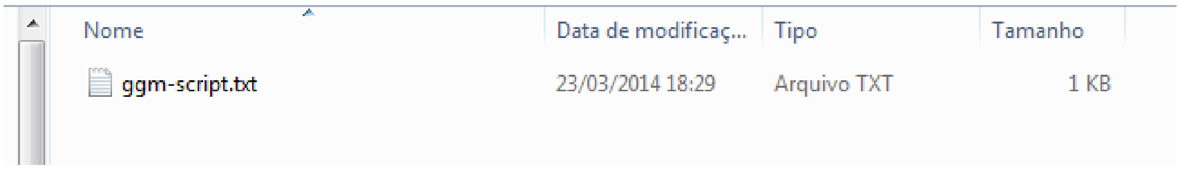
```
gilgamesh: save; C:\Temp\gilgamesh.bin
Gilgamesh Core saved in C:\Temp\gilgamesh.bin
gilgamesh: fact; 100.0; dog eats meat
gilgamesh: facts;
[dog, eats, meat]
gilgamesh: atoms;
dog, eats, meat,
```

If you intend to execute a Gilgamesh script outside the program (for example, if you want to integrate with other systems or make it run as a service), you can execute the command shell passing the script file and the output file (if you intend to) as parameters. That will make Gilgamesh loads the script file, run all commands inside it and writes answers inside another text file. For example, if your script is inside the file "C:\Temp\ggm-script.txt" and you want the answers into another text file in "C:\Temp\ggm-output.txt", then you can run the following command:

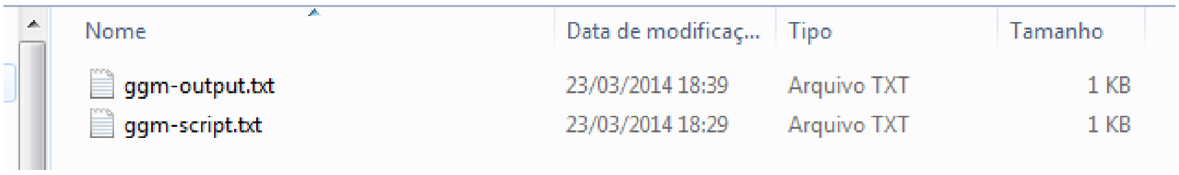


```
java -jar gilgamesh.jar C:\Temp\ggm-script.txt C:\Temp\ggm-output.txt
```

That will make Gilgamesh to create the file "C:\Temp\ggm-output.txt" if it not exists, with its answers inside it:



Nome	Data de modificaç...	Tipo	Tamanho
ggm-script.txt	23/03/2014 18:29	Arquivo TXT	1 KB

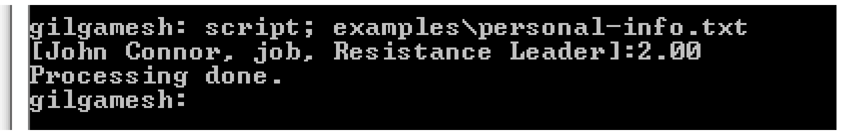


Nome	Data de modificaç...	Tipo	Tamanho
ggm-output.txt	23/03/2014 18:39	Arquivo TXT	1 KB
ggm-script.txt	23/03/2014 18:29	Arquivo TXT	1 KB

As another example, if you execute the script file "personal-info.txt" that goes along with Gilgamesh:

script; examples\personal-info.txt

Then it will give you the answer in terminal, since you have omitted an output file:



```
gilgamesh: script; examples\personal-info.txt
[John Connor, job, Resistance Leader]:2.00
Processing done.
gilgamesh:
```

Commands

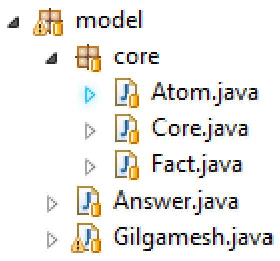
You can call “help;” to see all commands and its respective actions inside the command shell. Below there is a description of each command:

Action	Command
Record a fact into Gilgamesh Core in given order.	<atoms>
Make a question. The atoms order doesn’t matter.	<atoms>?
Same as ?, except that it will suppress the question atoms from its answer.	<atoms>?-
Indicate a comment (mainly used inside scripts). No action is done.	#<atoms>
Show the help menu with all commads.	help;
It will exit from Gilgamesh program. No memory is saved.	exit;
Clean the Gilgamesh Core memory. Is the same as if you close the program and execute it again.	reset;
Create a fact with specific force. If the force is 10 for example, it is the same as if you enter a fact 10 times. If the fact already exists, it will be summed to the current force.	fact;<number>;<atoms>
List all facts in Gilgamesh Core memory.	facts;
List all atoms in Gilgamesh Core memory.	atoms;
Show all connection quantities from atoms.	links;
Same as typing atoms followed by <ENTER>. It will count +1 to fact force.	reinforce;<atoms>
Decrease 1 from the fact force.	punish;<atoms>
Remove a fact. Atoms with no connections will be deleted as well.	remove;<atoms>
Bring a list of best answers, matching all atoms informed.	answer;<atoms>
Bring a list of best answers, matching any of the atoms informed.	answerany;<atoms>
Bring a list of answers, suppressing question atoms.	answersupp;<atoms>
Bring a list of answers matching any atom, suppressing questions atoms (same analogy as distinct).	answerdist;<atoms>
Loads and execute a script (text file), as if you would be typing its commands. If an output file is informed, the answers will be written on that file (integration).	script;<IN>;<OUT>
Show some Gilgamesh Core statistics.	statistics;
Show what you type after ; (semicolon) without doing any action.	echo;<atoms>

For Developers - Using the Code

This section is to programmers which intend to use Gilgamesh sources inside their own codes. If you are not a programmer, perhaps this section can be a little confuse to you so you can go directly to the examples.

Gilgamesh is done in Java programming language. After you unzip the downloaded file, you will see a folder called "src" which contains all sources with their respective packages. That folder can be copied to your IDE source folder (e.g.: Eclipse, Netbeans). The model packages and classes are:



```
model
  core
    Atom.java
    Core.java
    Fact.java
  Answer.java
  Gilgamesh.java
```

Gilgamesh is a really tiny project for what it does. Below are some short explanation about those classes:

- ✓ **Atom**: this class represents a single and indivisible block of information;
- ✓ **Fact**: it's a group of atoms, in a specific order you have entered;
- ✓ **Core**: is the Gilgamesh memory, according a context. The context is a subject or topic you decide, and its scope is up to you;
- ✓ **Answer**: it represents an summarized answer from Gilgamesh Core, according the question you have entered.
- ✓ **Gilgamesh**: the main class, containing some factory methods to help you.

If you want to create a simple Gilgamesh Core like the same used by the command shell, you can instantiate it as below:

```
Core<String> core = Gilgamesh.createStringCore();
```

After that, the Gilgamesh Core memory is ready to be used in a context you want. The context is the scope concept you have to choose, according what you want to do. If you create one Gilgamesh Core memory for crossed contexts, you can have bad or unexpected results (at the beginning, I had bad results in my stock market project due to crossed contexts). For example, if you are using Gilgamesh to control customer data, like in a database, then you should have the that context in mind when developing your code. If you try to control data from another context, let's say, like results for market sales, then the answers can be a little confusing to understand.

Inside the Core class, you have methods with same names used in command shell, like reinforce(), punish(), fact() etc. If you understand how to use the shell commands, then you will understand the methods as well. You can use the Javadoc to get details about methods and how they work. Some implementations were not made intentionally, because it is possible to create your own logic using Gilgamesh classes.

Gilgamesh Core is a generic class what means that you can instantiate it with any type you want. It is possible to create a Core using a super class so you can put in its memory a bunch of different types like strings, dates, numbers or any other type you need. If you intend to create that super class to be used into Gilgamesh, you have to implement the interface Comparable to do that. Another important point is to remember that internally, Gilgamesh will keep facts with your beans in a HashMap so if you want some specific behavior, you should override the hashCode() and equals() methods.

There is a package called "main" where you can find some executable classes. One of them is the Command Shell class. Another one, called GilgameshTest, is the originally test class for Gilgamesh's behavior. If you intend to customize or implement some customized code, the suggestion is to run this class in order to ensure the results will be according Gilgamesh concepts, unless you want to change them. If you run the GilgameshTest class, do not forget to include -ea parameter in JVM arguments to enable assertions.

Coding Example

Let's suppose you want to work connecting strings in facts for a medical application. In your application, you have to connect symptoms with diseases. First of all, you have to create a Gilgamesh core to work with it:

```
Core<String> gilgamesh = Gilgamesh.createStringCore();
```

After this point you have a entire Gilgamesh's memory to work with. Now, you could connect atoms (block of informations) as you wish:

```
gilgamesh.reinforce("flu", "fever");
gilgamesh.punish("flu", "pain");
```

In this case, we are connecting the flu disease with its symptoms. First, we create a connection to fever and then in another line, we create another connection between flu and pain. So far we have a simple connection for force values as 1 for each reinforcement and -1 for each punishment for symptoms (fever and pain), since we told it that connection happens only one time.

However, if your intention is just to create a connection among them to consult later, then you don't have to call that reinforcements or punishments. That is because you don't need to create successive connections for those facts which will imply the statistical behavior of Gilgamesh's memory. In fact, if you just want to create simple connections, the better way to connect them would be:

```
gilgamesh.fact(0.0, "flu", "fever");
gilgamesh.fact(0.0, "flu", "pain");
```

In this case you are not giving forces to those connections but you are only connecting them. To query any of those connections, you could call:

```
List<Answer<Fact<String>>> answers = gilgamesh.getAnswers(false, false, "flu");
```

The parameters are:

- first boolean: if the answer should contains any question atoms. false if the answers should contains all question atoms ("flu" and any other).
- second boolean: indicates that answers should be combined, suppressing the atoms informed (in this case, "flu" and any other);
- next parameters: the atoms you want to search ("flu");

The result list should contain two facts:

```
[[flu, pain]:0,00, [flu, fever]:0,00]
```

According you have entered before.

Now, if you want to use its AI capability, then you have to reinforce or punish a kind of connection (or provide any force calling fact method). Let's suppose you have a bunch of treatment results and you want to know which one is better for a disease. In this case, you have to use its AI behavior, reinforcing or punishing a fact like:

```
gilgamesh.reinforce("flu", "aspirin");
gilgamesh.reinforce("flu", "aspirin");
gilgamesh.reinforce("flu", "aspirin");
gilgamesh.reinforce("flu", "paracetamol");
gilgamesh.reinforce("flu", "paracetamol");
gilgamesh.reinforce("flu", "paracetamol");
gilgamesh.reinforce("flu", "paracetamol");
```

In this case, you are saying that paracetamol is a better treatment for flu than aspirin, since you have entered that connection 4 times compared of 3 for aspirin. So, to query the best treatment you could call again:

```
List<Answer<Fact<String>>> answers = gilgamesh.getAnswers(false, false, "flu");
```

Note that if you don't specify the kind of each connection, it will consider all of them as in the same context, bringing you all the results (facts) with flu on it:

```
[[flu, paracetamol]:4,00, [flu, aspirin]:3,00, [flu, pain]:0,00, [flu, fever]:0,00]
```

So, in this case, you should specify the context you are working on for each fact, entering that information like:

```
gilgamesh.fact(0.0, "disease", "flu", "symptom", "fever");
gilgamesh.fact(0.0, "disease", "flu", "symptom", "pain");
gilgamesh.reinforce("disease", "flu", "treatment", "aspirin");
gilgamesh.reinforce("disease", "flu", "treatment", "aspirin");
gilgamesh.reinforce("disease", "flu", "treatment", "aspirin");
gilgamesh.reinforce("disease", "flu", "treatment", "paracetamol");
gilgamesh.reinforce("disease", "flu", "treatment", "paracetamol");
gilgamesh.reinforce("disease", "flu", "treatment", "paracetamol");
gilgamesh.reinforce("disease", "flu", "treatment", "paracetamol");
```

Including the words "disease", "symptom" and "treatment" you are providing in which context that fact is. Now, if you want to query a result in a disease / symptom context, then you could call:

```
List<Answer<Fact<String>>> answers = gilgamesh.getAnswers(false, false, "disease", "flu", "symptom");
```

And the answer would be:

```
[[disease, flu, symptom, pain]:0,00, [disease, flu, symptom, fever]:0,00]
```

Which brings you the right answer, in the right context (disease / symptom). If you want to query connections in disease / treatment context, then you could ask:

```
List<Answer<Fact<String>>> answers = gilgamesh.getAnswers(false, false, "disease", "flu", "treatment");
```

And the answer would be:

```
[[disease, flu, treatment, paracetamol]:4,00, [disease, flu, treatment, aspirin]:3,00]
```

Note that the paracetamol answer came first, since we have entered it 4 times indicating that would be the best treatment answer.

Practical Examples

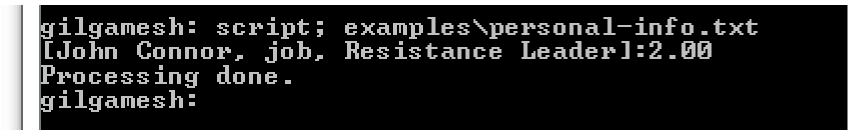
Very often, when I tell people to use AI software's to solve everyday problems, people are used to tell me that is not a good idea. I think that sometimes, people think that AI is too advanced or too complex to use that, what could mean too much work or costs to maintain that kind of software.

So, I have decided to do a whole chapter with short and simple examples, in order to provide the beginning for those who want to use Gilgamesh AI software to solve those kind of problems. I hope the examples can help people to understand how it can be used and how to do that. I do not intend to do complete solutions but just give the light at the end of the tunnel :)

Each example was done thinking how that kind of person or professional can use Gilgamesh to solve problems related to its everyday life. Usually the examples are going to start giving some kind of scenario, telling how to input those data into Gilgamesh and then, how to extract the information you want.

I have tried to put each example in at maximum, two pages. In most of examples, I have generated random data in order to related them to a real scenario, providing the closest situation you could find on each kind of problem. Anyway, if you have an idea about using Gilgamesh in a real scenario or you want to solve any questions about one of the examples, you can send me an email at gilgamesh.ai.project@gmail.com. I will try to help as possible as I can :)

Remember: to load an example script, call in the command shell:



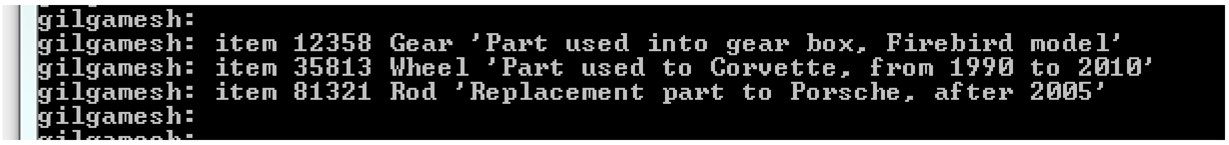
```
gilgamesh: script; examples\personal-info.txt
[John Connor, job, Resistance Leader]:2.00
Processing done.
gilgamesh:
```

Stock Flow Control (examples/stock-flow.txt)

This example is about controlling parts flow in a factory, just to demonstrate how Gilgamesh can keep information, without any fact reinforcements. Let's suppose you have a small industry and wants to register all material that enters and exits your plant. Remember that Gilgamesh is an AI system so it will learn facts - if you want a system that makes calculations, then probable you should customize it through source code or exporting data to a worksheet. To register the material flow, you just need to enter all information. Example:

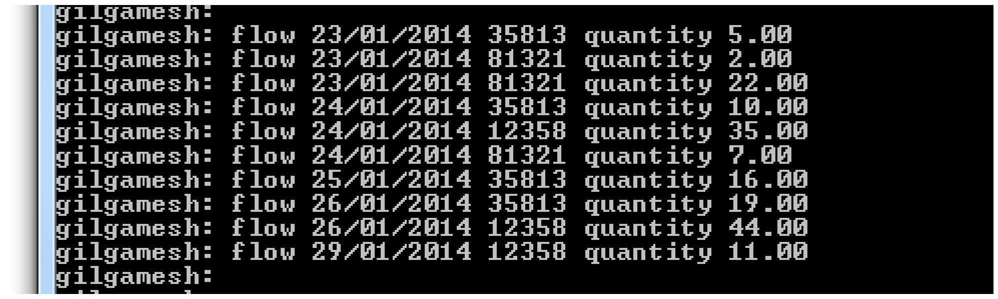
Type	**Number**	**Part**	**Name Description**
Item	12358	Gear	Part used into gear box, Firebird model
Item	35813	Wheel	Part used to Corvette, from 1990 to 2010
Item	81321	Rod	Replacement part to Porsche, after 2005

Now what we have to do is to inform what they are:



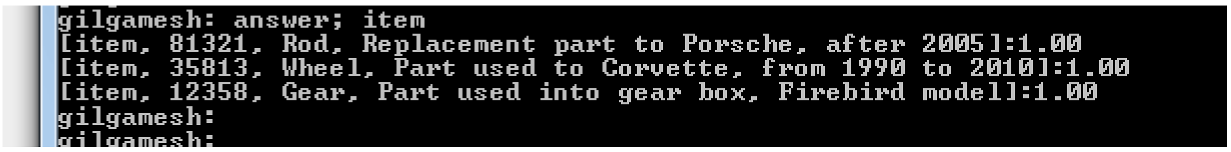
```
gilgamesh:
gilgamesh: item 12358 Gear 'Part used into gear box, Firebird model'
gilgamesh: item 35813 Wheel 'Part used to Corvette, from 1990 to 2010'
gilgamesh: item 81321 Rod 'Replacement part to Porsche, after 2005'
gilgamesh:
```

After this point you have information inside Gilgamesh Core memory where you can consult any information about those items. This is important because, in fact, it is up to you how to connect your fact atoms. So now you can enter some stock during time:



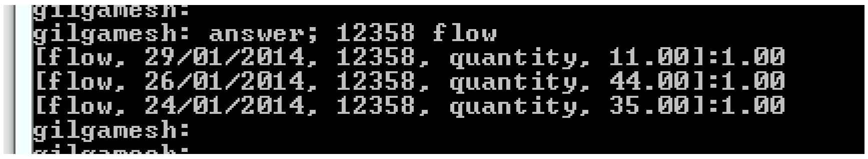
```
gilgamesh:
gilgamesh: flow 23/01/2014 35813 quantity 5.00
gilgamesh: flow 23/01/2014 81321 quantity 2.00
gilgamesh: flow 23/01/2014 81321 quantity 22.00
gilgamesh: flow 24/01/2014 35813 quantity 10.00
gilgamesh: flow 24/01/2014 12358 quantity 35.00
gilgamesh: flow 24/01/2014 81321 quantity 7.00
gilgamesh: flow 25/01/2014 35813 quantity 16.00
gilgamesh: flow 26/01/2014 35813 quantity 19.00
gilgamesh: flow 26/01/2014 12358 quantity 44.00
gilgamesh: flow 29/01/2014 12358 quantity 11.00
gilgamesh:
```

Every time when you want to query those data, you just need to ask Gilgamesh according what you want to know. Let's suppose you want to get part numbers registered into memory:



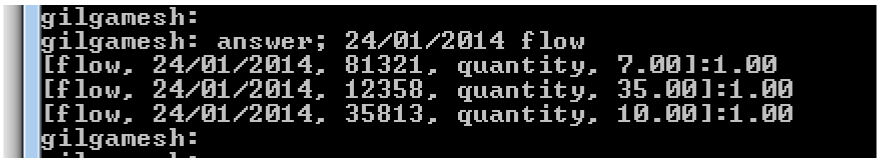
```
gilgamesh: answer; item
[item, 81321, Rod, Replacement part to Porsche, after 2005]:1.00
[item, 35813, Wheel, Part used to Corvette, from 1990 to 2010]:1.00
[item, 12358, Gear, Part used into gear box, Firebird model]:1.00
gilgamesh:
gilgamesh:
```

After that, you could ask all entries about a specific item:



```
gilgamesh:
gilgamesh: answer; 12358 flow
[flow, 29/01/2014, 12358, quantity, 11.00]:1.00
[flow, 26/01/2014, 12358, quantity, 44.00]:1.00
[flow, 24/01/2014, 12358, quantity, 35.00]:1.00
gilgamesh:
```

And then query which parts entered in an specific date:



```
gilgamesh:
gilgamesh: answer; 24/01/2014 flow
[flow, 24/01/2014, 81321, quantity, 7.00]:1.00
[flow, 24/01/2014, 12358, quantity, 35.00]:1.00
[flow, 24/01/2014, 35813, quantity, 10.00]:1.00
gilgamesh:
```

If you teach Gilgamesh with specific data types (item, part, description etc) it can work similar to a common relational database, when the point is to connect fields. However, there is a certain flexibility using Gilgamesh because you can dynamically define type of atoms, connection among them and how long they will be.

Unfortunately so far, you cannot do any calculations just using the shell command: you have to do any kind of source code customization. Despite those problems, you still can control data like above (actually, this example shows the simplest way to do that).

Mechanical Diagnosis (examples/mechanical-diagnosis.txt)

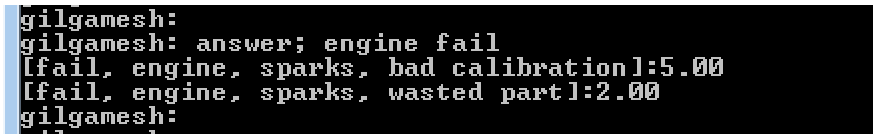
Let’s suppose you have a small mechanical workshop where you fix cars and you would like to improve the quality of your work, anticipating problems that could happen. For all those fixes that you did right and knew about the cause, you could enter one record into Gilgamesh. Below there is a scenario of 1 week of work:

Defect	**Part**	**Fix**	**Cause**
noise	wheel	bearing	bad calibration
fail	engine	sparks	bad calibration
noise	engine	valve	wasted part
fail	suspension	dumper	material defect
noise	suspension	dumper	bad calibration
noise	suspension	spring	material defect
fail	engine	sparks	wasted part
fail	wheel	bearing	material defect
fail	engine	sparks	bad calibration
noise	engine	valve	wasted part
fail	suspension	dumper	material defect
noise	suspension	dumper	bad calibration
noise	suspension	spring	material defect
fail	engine	sparks	wasted part
noise	wheel	bearing	bad calibration
fail	engine	sparks	bad calibration
noise	suspension	dumper	bad calibration
noise	suspension	spring	material defect
fail	engine	sparks	bad calibration
noise	suspension	dumper	bad calibration
noise	engine	valve	wasted part
fail	engine	sparks	bad calibration

You could enter those facts like:

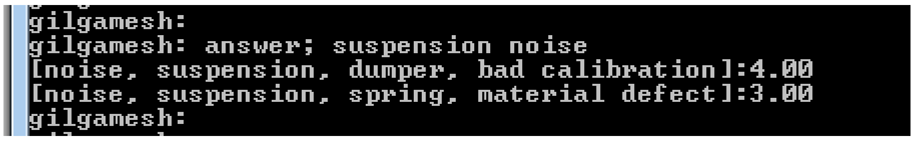
noise wheel bearing 'bad calibration'

After entered all information above, we can start asking questions. So, what if you want to know what is the most common cause to the engine fail?



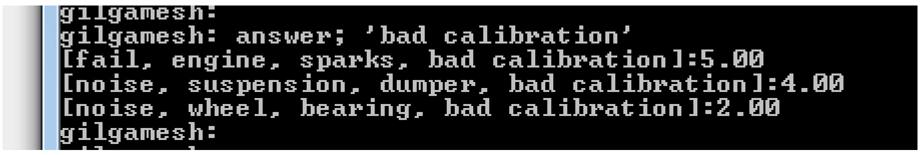
```
gilgamesh:
gilgamesh: answer; engine fail
[fail, engine, sparks, bad calibration]:5.00
[fail, engine, sparks, wasted part]:2.00
gilgamesh:
```

Now we know that bad calibrations are the most common cause in engine fails. We could check causes for suspension noise:



```
gilgamesh:
gilgamesh: answer; suspension noise
[noise, suspension, dumper, bad calibration]:4.00
[noise, suspension, spring, material defect]:3.00
gilgamesh:
```

The most common is bad calibration too. What if we look defects related to bad calibrations?



```
gilgamesh:
gilgamesh: answer; 'bad calibration'
[fail, engine, sparks, bad calibration]:5.00
[noise, suspension, dumper, bad calibration]:4.00
[noise, wheel, bearing, bad calibration]:2.00
gilgamesh:
```

So we found out that bad calibrations are causing noise in wheels too. In this case, some actions could be taken to verify all calibrations in all cars you fix before delivery the car to the customer. That could improve the quality of your work.

Medical Prediction (examples/medical-prediction.txt)

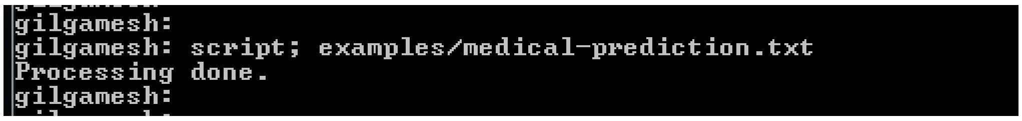
This is probably the example I like most. That is because this is the kind of system which could really help people (actually, it is one of the reasons I have decided to made Gilgamesh as open source). In my opinion, one prediction system like this could reduce medical errors and make the diagnosis process faster and more accurate, helping medics to take decisions. It could be used to find solutions for diseases which we do not know any treatment yet. So, let’s consider these things:

Disease: the disease diagnosed by a medic
Symptom: is what the patient feels.
Cause: the real problem to be treated.
Treatment: what was done or drug used to cure people.

We will connect them as below (there are other ways to connect them, though):

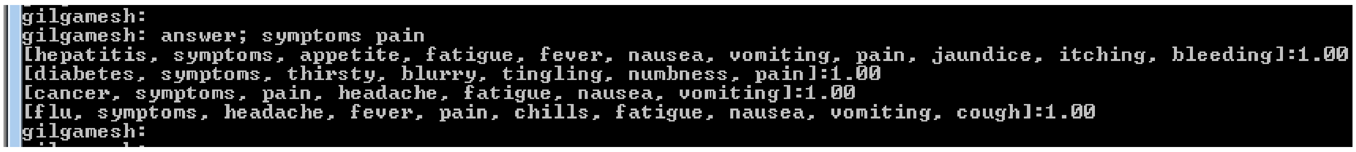
- **one disease can have one or more symptoms;**
- **one disease can have one or more causes;**
- **one disease has different treatments and we want to know which one is better;**

For this example, we can have more than a single atom of same type into a fact (btw, I kept atoms as simple as possible). For diseases with their symptoms and causes, single and few facts were used. For treatments, shorter and more facts were created and there is one row for each treated patient. That allows to Gilgamesh chooses the best treatment for each disease. As more information you enter, better answers you will get. After load the script:



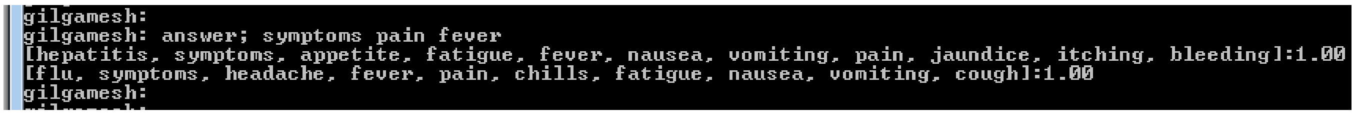
```
gilgamesh:
gilgamesh: script; examples/medical-prediction.txt
Processing done.
gilgamesh:
```

Gilgamesh has some medical data and it would be ready to be used by a medic. During a patient consult, a medic can verify these symptoms : "pain", "fever" and "cough". Then, he could starting ask to Gilgamesh find what was the most probable disease. Looking for “pain”, it would bring us all diseases which have pain as symptom:



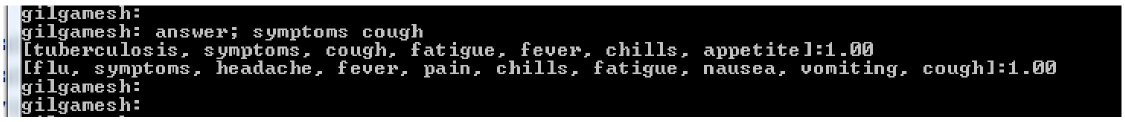
```
gilgamesh:
gilgamesh: answer; symptoms pain
[hepatitis, symptoms, appetite, fatigue, fever, nausea, vomiting, pain, jaundice, itching, bleeding]:1.00
[diabetes, symptoms, thirsty, blurry, tingling, numbness, pain]:1.00
[cancer, symptoms, pain, headache, fatigue, nausea, vomiting]:1.00
[flu, symptoms, headache, fever, pain, chills, fatigue, nausea, vomiting, cough]:1.00
gilgamesh:
```

Including “fever” the list would be shorter, since only flu and hepatitis have both symptoms together:



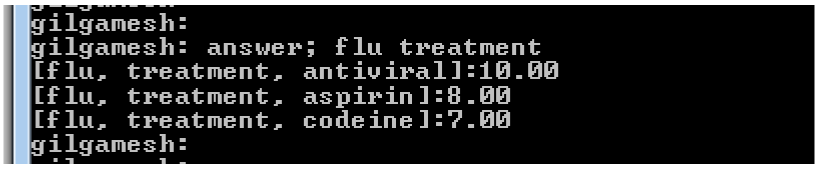
```
gilgamesh:
gilgamesh: answer; symptoms pain fever
[hepatitis, symptoms, appetite, fatigue, fever, nausea, vomiting, pain, jaundice, itching, bleeding]:1.00
[flu, symptoms, headache, fever, pain, chills, fatigue, nausea, vomiting, cough]:1.00
gilgamesh:
```

And finally, including “cough”:



```
gilgamesh:
gilgamesh: answer; symptoms cough
[tuberculosis, symptoms, cough, fatigue, fever, chills, appetite]:1.00
[flu, symptoms, headache, fever, pain, chills, fatigue, nausea, vomiting, cough]:1.00
gilgamesh:
gilgamesh:
```

He got only flu as possible answer. Now, based on all medical treatment records, the medic could find out what is the best treatment for flu:



```
gilgamesh:
gilgamesh: answer; flu treatment
[flu, treatment, antiviral]:10.00
[flu, treatment, aspirin]:8.00
[flu, treatment, codeine]:7.00
gilgamesh:
```

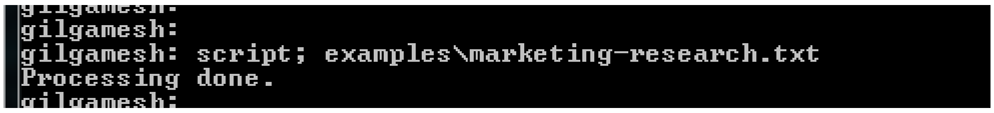
According treatments that have happened last period, an antiviral treatment is being more effective than others. So, it is possible to conclude that the system is providing some help to medic diagnostics.

Of course that the example above is very rude but the intention is to show how Gilgamesh could be used as a learning system, crossing medical data to find best answers to diseases treatments, among other things. I hope that someone likes this example (and use it in practical ways).

Marketing Research (examples/marketing-research.txt)

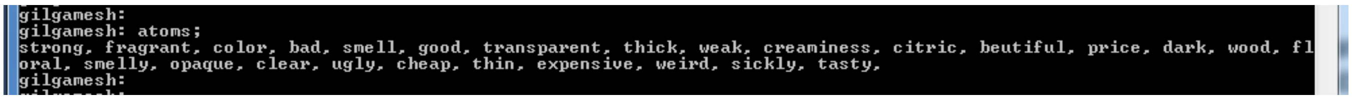
Marketing professional are used to do market researchs in order to identify what customers want. Usually, there are two main kinds of researchs: quantitative and qualitative. Quantitative researchs gives you result in numbers, so it is easy to measure that. In the other hand, qualitative researchs are difficult to measure, since they are related about quality descriptions. Gilgamesh can be used to help in both researchs, but specially for qualitative ones.

Let’s suppose you have interviewed a bunch of people in a qualitative research about a shampoo new brand. Now you have all answers, you would you like to know some customers intentions related to your product. In this example, we are going to simplify adjectives in order to simplify the example. First, we have to load the example:



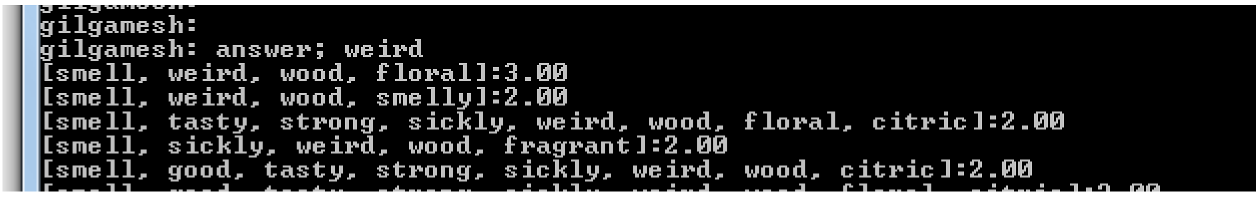
```
gilgamesh:
gilgamesh:
gilgamesh: script; examples\marketing-research.txt
Processing done.
gilgamesh:
```

Now we have all opinions about those customers. First of all, let's find out what words respondents have used:



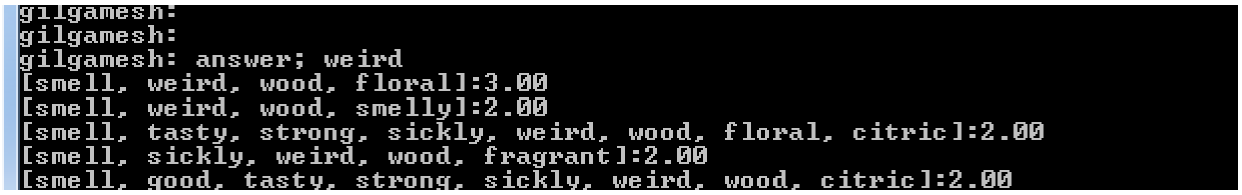
```
gilgamesh:
gilgamesh: atoms;
strong, fragrant, color, bad, smell, good, transparent, thick, weak, creaminess, citric, beutiful, price, dark, wood, fl
oral, smelly, opaque, clear, ugly, cheap, thin, expensive, weird, sickly, tasty,
gilgamesh:
```

So now, let’s find out what customers think about our shampoo. if you want to know what was the opinion which used more "weird" in answers:



```
gilgamesh:
gilgamesh: answer; weird
[smell, weird, wood, floral]:3.00
[smell, weird, wood, smelly]:2.00
[smell, tasty, strong, sickly, weird, wood, floral, citric]:2.00
[smell, sickly, weird, wood, fragrant]:2.00
[smell, good, tasty, strong, sickly, weird, wood, citric]:2.00
```

That means that more people think that smells of wood and floral are weird (remember that my data is random :D). So, let’s see what people have realized about the citric smell in our product:



```
gilgamesh:
gilgamesh:
gilgamesh: answer; weird
[smell, weird, wood, floral]:3.00
[smell, weird, wood, smelly]:2.00
[smell, tasty, strong, sickly, weird, wood, floral, citric]:2.00
[smell, sickly, weird, wood, fragrant]:2.00
[smell, good, tasty, strong, sickly, weird, wood, citric]:2.00
```

It seems they liked it. You could ask anything you want, like "What do customers think when they have answered 'good price'?", "Has someone answered 'smell strong weird'?" or "What opinions contains 'strong'?". The possibilities are infinite, all depends what you want to know.

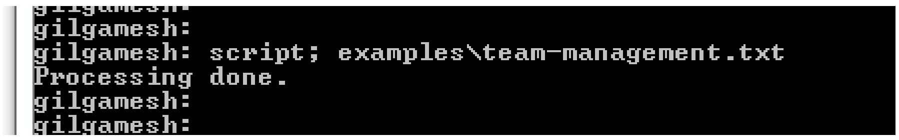
Of course that those random data I have generated to the example sometimes do not make any sense (you have atoms like 'good' and 'bad' in the same fact). But the point is to show how you could use Gilgamesh memory to improve your ability to identify market trends, customer wishes etc.

Team Management (examples/team-management.txt)

Most examples so far are related to connect a few information types. But you can take some advantages from Gilgamesh if you connect information of same type or if you don't know (or don't want) to classify those information. This example will show you how to connect data of same type (players, in this case). Let's suppose the following example: you are a paintball team manager and have the possibility to assembly your team between 2 and 4 players. To the championship, you would like to know which team configuration is better in order to have the best performance. Usually what team managers do is to choose all those best players and put them together. Very often, combining all best players doesn't mean that the team will be the best one. Now, what you have to do is: find the best players into the best team configuration. Your players are:

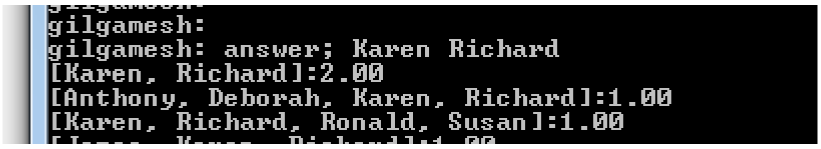
James, David, Christopher, George, Ronald, John, Richard, Daniel, Kenneth, Anthony, Robert, Charles, Paul, Steven, Kevin, Michael, Joseph, Mark, Edward, Jason, William, Thomas, Donald, Brian, Jeff, Mary, Jennifer, Lisa, Sandra, Michelle, Patricia, Maria, Nancy, Donna, Laura, Linda, Susan, Karen, Carol, Sarah, Barbara, Margaret, Betty, Ruth, Kimberly, Elizabeth, Dorothy, Helen, Sharon, Deborah

So you will enter to Gilgamesh last year game results. You can do that loading the file:



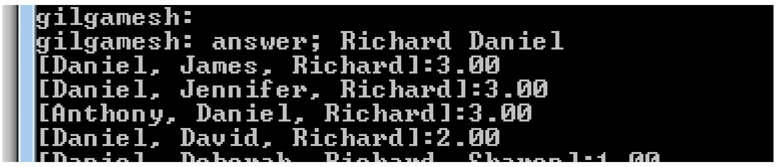
```
gilgamesh:
gilgamesh: script; examples\team-management.txt
Processing done.
gilgamesh:
gilgamesh:
```

After that, you could ask something: “What if I put Karen and Richard in a team?”: Let’s see:



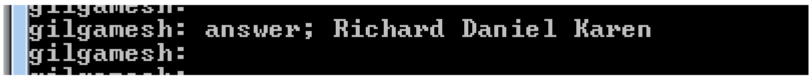
```
gilgamesh:
gilgamesh: answer; Karen Richard
[Karen, Richard]:2.00
[Anthony, Deborah, Karen, Richard]:1.00
[Karen, Richard, Ronald, Susan]:1.00
```

Now, what about only Richard and Daniel?



```
gilgamesh:
gilgamesh: answer; Richard Daniel
[Daniel, James, Richard]:3.00
[Daniel, Jennifer, Richard]:3.00
[Anthony, Daniel, Richard]:3.00
[Daniel, David, Richard]:2.00
```

If we try to put Richard, Daniel and Karen in the same team, what are the results?



```
gilgamesh: answer; Richard Daniel Karen
gilgamesh:
```

No answers. So, we can see that we have a better configuration if we put Jennifer, James or Anthony together with Richard and Daniel in the same team. Again, there are a lot of ways you can make questions to the code. It will depends what you want to know.

Customized Programming Language (examples/custom-prog-lang.txt)

Sometimes, a common user needs to do some kind of work which requests programming knowledge. That implies in a learning curve that often is not requested to that professional main role. To have that task done, the professional has to request some kind of work from a programmer.

Gilgamesh can help with this task, learning a new language and helping to simplify that issue. Actually, it can learn a pseudo code, that could be used by any common user, since he knows what he is doing inside Gilgamesh. The result is much more intuitive than most programming languages (it would be similar what Ruby on Rails does), allowing a user without any programming experience, can do some kind of programming. Let's see what we can do.

Supposing that we would like to allow a common user to create simple html pages, to include them into a human resource website, we could teach Gilgamesh to learn common tasks that the user could do, matching (or "translating") them to html code. We could teach something:

```
'start the page' <html>
```

From now on, every time when we type into the console:

```
'start the page'?-
```

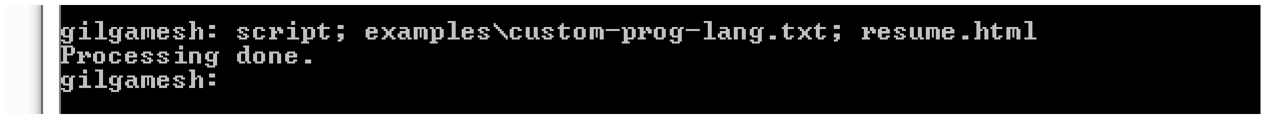
The answer would be:

```
<html>
```

What means that Gilgamesh could do the programmer work to the user. Some more examples:

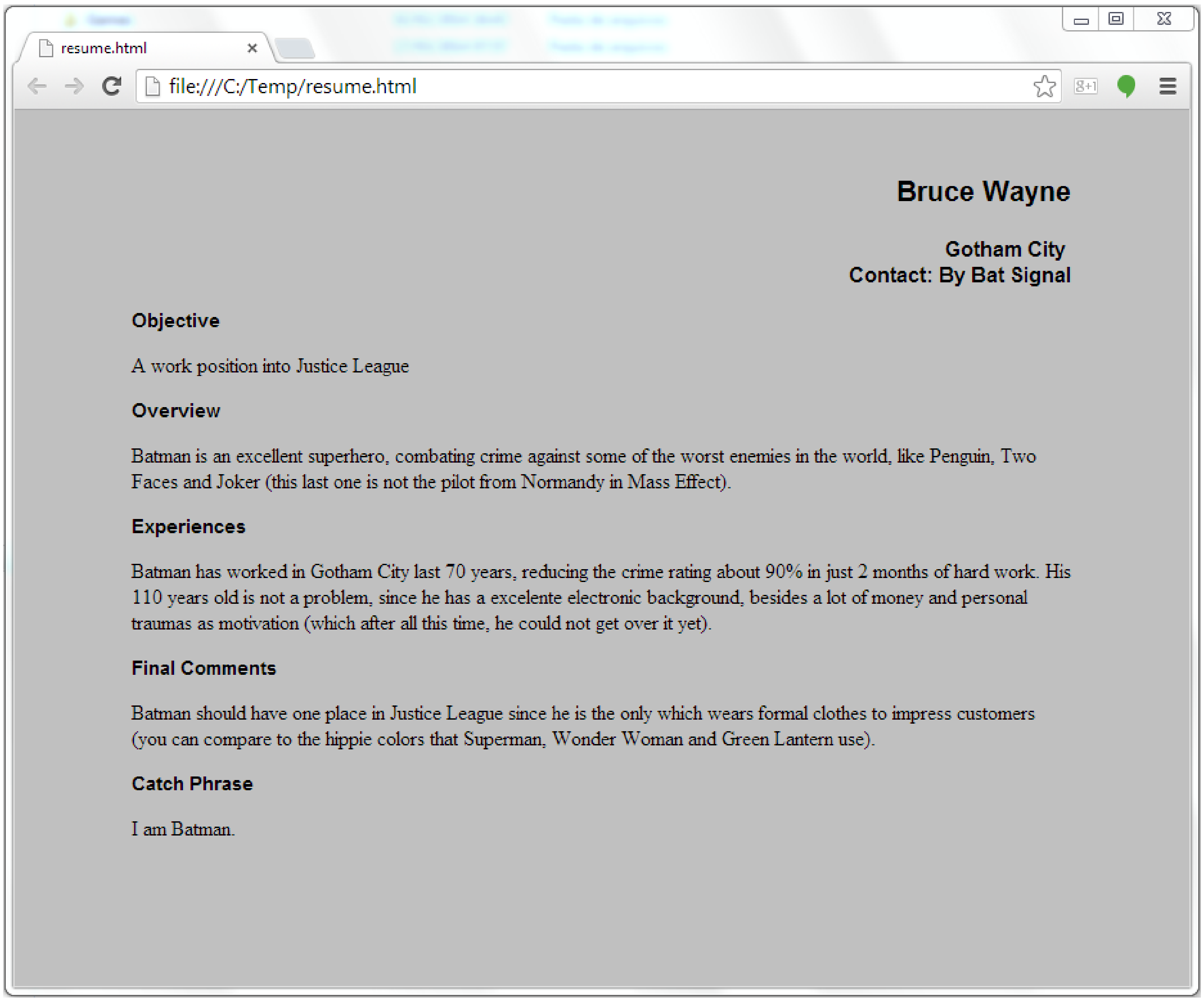
```
'finish the page' </html>
'set the name style' '<style>p.name { font: bold 22px arial,sans-serif; text-align: right;}</style>'
'open a title paragraph' '<p class="title">'
```

So you could program using Gilgamesh just asking it to answer for what you want to do. E.g.: 'start the page', 'close the page', 'set the name style' etc. Executing the script examples\custom-prog-lang.txt as below, will creates a html file called resume.html. Inside that file, you will see the programming result of a typed pseudo code.



```
gilgamesh: script; examples\custom-prog-lang.txt; resume.html
Processing done.
gilgamesh:
```

You can open it in a web browser, like Chrome or Firefox:



Bruce Wayne

Gotham City
Contact: By Bat Signal

Objective

A work position into Justice League

Overview

Batman is an excellent superhero, combating crime against some of the worst enemies in the world, like Penguin, Two Faces and Joker (this last one is not the pilot from Normandy in Mass Effect).

Experiences

Batman has worked in Gotham City last 70 years, reducing the crime rating about 90% in just 2 months of hard work. His 110 years old is not a problem, since he has a excelente electronic background, besides a lot of money and personal traumas as motivation (which after all this time, he could not get over it yet).

Final Comments

Batman should have one place in Justice League since he is the only which wears formal clothes to impress customers (you can compare to the hippie colors that Superman, Wonder Woman and Green Lantern use).

Catch Phrase

I am Batman.

Traffic Control (examples/traffic-control.txt)

Imagine now that you work at traffic control. Your target is to reduce the traffic jam in rush hours so you have to synchronize three strategic semaphores (A, B and C) in order to achieve that. The time spent in traffic jams you have to reduce is about a period of 132 minutes during the rush hours. It would be possible to do that if you know what is the best configuration for each semaphore. You have the following information:

- **Traffic Jam Duration: is the time which the traffic jam persists in downtown during rush hours.**
- **Semaphore Switch Time: is the time each semaphore has before switch its state (from red to green and vice versa).**

You did a test during a lot of days, taking note of all traffic jam duration and about semaphore times. Let's suppose that some of your records are something like:

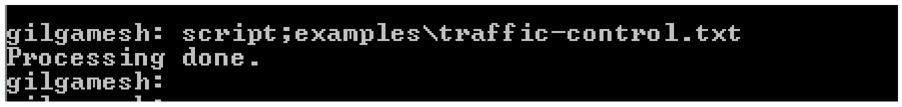
Traffic Duration	**Switch Time A**	**Switch Time B**	**Switch Time C**
128 minutes	35 seconds	50 seconds	40 seconds
167 minutes	20 seconds	15 seconds	30 seconds
149 minutes	45 seconds	30 seconds	25 seconds
...			

A
B
C
São Paulo
Measured Traffic Area
Cia Engenharia Tráfego Cet
Rua Melo Freire
Av. Conde de
Rua Cantagalo
Rua Azevedo Soares
Jardim Anália Franco
Av. Salim Farah Maluf
Rua da Mooca
Av. Vereador Abel Ferreira
Av. Sapopemba
Rua do Oratório
Av. Paes de Barros
Aclimação
Paraíso
Parque da Independência
Rua Orfanato
Av. Rio Branco
Av. São João
da Consolação
050
SP-160

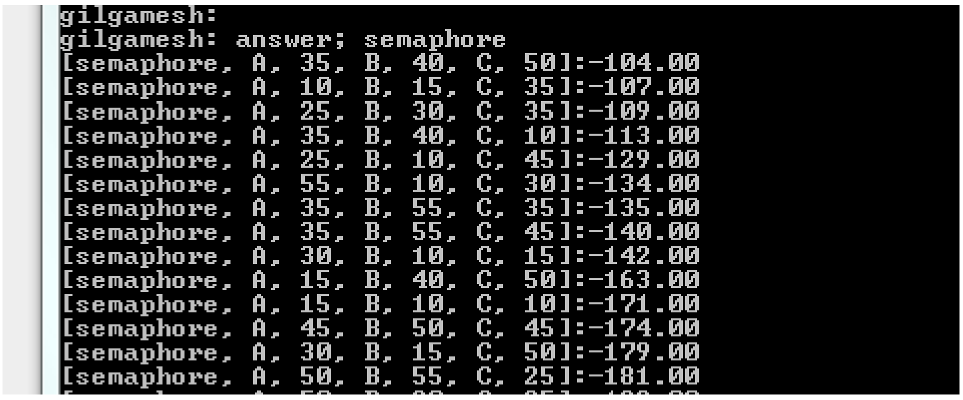
Some of those configurations can happen more than one time, having different duration for each measure. For example, you could test the same switch configuration for a whole week (e.g.: A 35s, B 50s and C 40s), so you would have 5 measures with same configuration but different traffic durations. Now, how is it possible to know which configuration has the best performance to reduce the traffic jam? Just feed those information into Gilgamesh, making the traffic jam duration as its fact force, like:

```
fact; -128; semaphore A 35 B 50 C 40
```

However, there is a trick here. As Gilgamesh sorts answers from the highest to lowest forces, it would bring us the worst case as first answer. Since we want to know the shortest time, we have to multiply our duration times by -1 to invert its results (this is why we used -128 instead 128). Then when we ask Gilgamesh to answer, it will bring us the shortest duration time instead the long one. Running the script:


```
gilgamesh: script;examples\traffic-control.txt
Processing done.
gilgamesh:
```

It will load all records about traffic jam and their respective semaphore configurations. Now, a simple question could bring us the best configuration for those semaphores:


```
gilgamesh:
gilgamesh: answer; semaphore
[semaphore, A, 35, B, 40, C, 50]:-104.00
[semaphore, A, 10, B, 15, C, 35]:-107.00
[semaphore, A, 25, B, 30, C, 35]:-109.00
[semaphore, A, 35, B, 40, C, 10]:-113.00
[semaphore, A, 25, B, 10, C, 45]:-129.00
[semaphore, A, 55, B, 10, C, 30]:-134.00
[semaphore, A, 35, B, 55, C, 35]:-135.00
[semaphore, A, 35, B, 55, C, 45]:-140.00
[semaphore, A, 30, B, 10, C, 15]:-142.00
[semaphore, A, 15, B, 40, C, 50]:-163.00
[semaphore, A, 15, B, 10, C, 10]:-171.00
[semaphore, A, 45, B, 50, C, 45]:-174.00
[semaphore, A, 30, B, 15, C, 50]:-179.00
[semaphore, A, 50, B, 55, C, 25]:-181.00
```

So it is possible to conclude that the best configuration to reduce traffic jam would be:

Switch Time to Semaphore A: 35 seconds
Switch Time to Semaphore B: 40 seconds
Switch Time to Semaphore C: 50 seconds

That makes your traffic jam duration will remain only for 104 minutes. Work done.

It is possible to include in your fact the week day, day time or any information you would think it is relevant to identify the best configuration. Still would be possible that A, B and C would not be a single semaphore, but a whole group of them working together. The possibilities are many.

Human Resource Database (examples/human-resource-db.txt)

Sometime, human resource professionals have a lot of problems to find a candidate that fits to some company position, among all those resumes they received. That happens because each person has its own way to do its resume. Very often, we see some companies creating their own human resource database, asking to candidates to register their resumes on those systems. As this registering has no standards about how to do that, some companies can ask different questions about candidates information, which sometimes, can end up in a mess or in annoying forms to fill the templates.

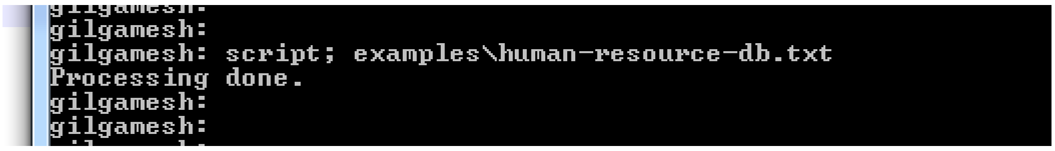
It is possible to use Gilgamesh to help with this process. Basically, the only thing they should do is to input all information available in candidate resumes, using its name and what about that information refers as a ID. Let's suppose that we have the following candidate information:

name, address, objective, skills, experience

So you can load those informations, linking the facts with key words you want, like above. For this example, we use them like below:

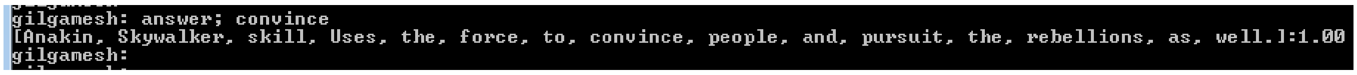
Captain Kirk address Enterprise Vessel, where no man has gone before

Running the script as below:



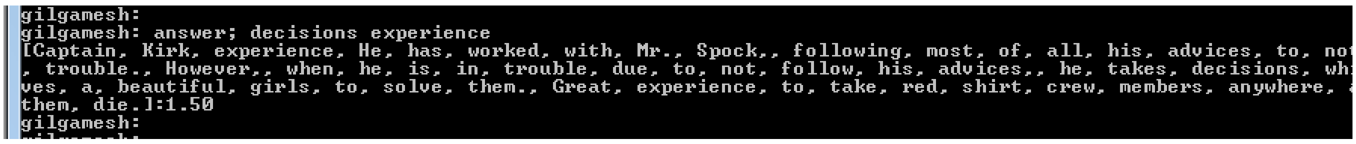
```
gilgamesh:
gilgamesh: script; examples\human-resource-db.txt
Processing done.
gilgamesh:
gilgamesh:
```

You will load some resumes, like if you had typed them entering all information available in a format like the pattern described before. After that, you can start querying some information, asking for key words you are looking for. Now, let's suppose that you want to find a person to work in sales department so you need someone with convincing skills. Let's take a look for the word 'convince':



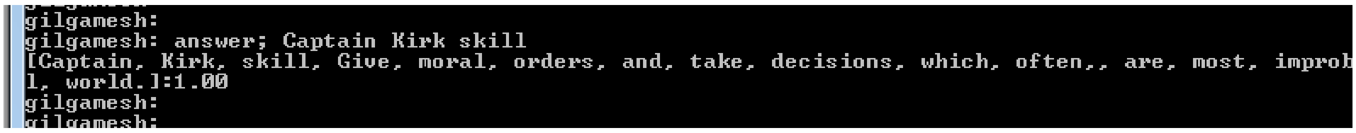
```
gilgamesh: answer; convince
[Anakin, Skywalker, skill, Uses, the, force, to, convince, people, and, pursuit, the, rebellions, as, well.]:1.00
gilgamesh:
```

Note you have found that there one person with this characteristic as one of his skills. You could search for people which has the experience to take decisions. So:



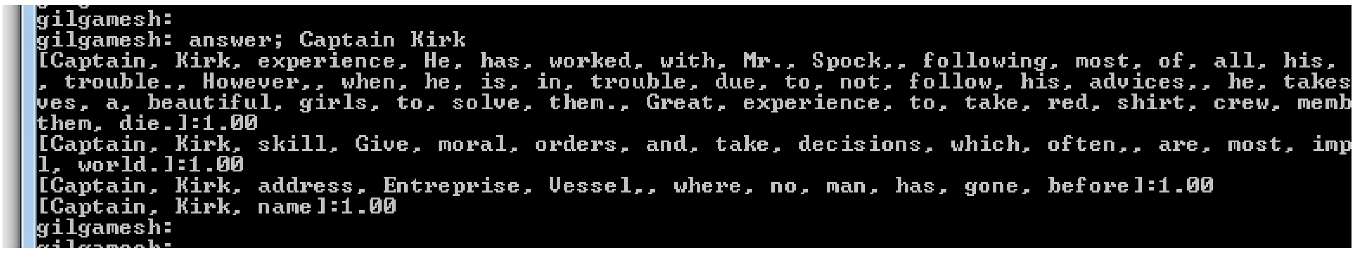
```
gilgamesh:
gilgamesh: answer; decisions experience
[Captain, Kirk, experience, He, has, worked, with, Mr., Spock,, following, most, of, all, his, advices, to, not
, trouble., However,, when, he, is, in, trouble, due, to, not, follow, his, advices,, he, takes, decisions, wh
ves, a, beautiful, girls, to, solve, them., Great, experience, to, take, red, shirt, crew, members, anywhere,
them, die.]:1.50
gilgamesh:
```

You can see that it is possible to query any word you want, since you enter each resume word as an atom, making bigger facts. It is possible to query much more information according your needs and details about the job opportunity you have, matching them with candidates information. After knowing who has those needs you want, you can ask specific questions like personal details:



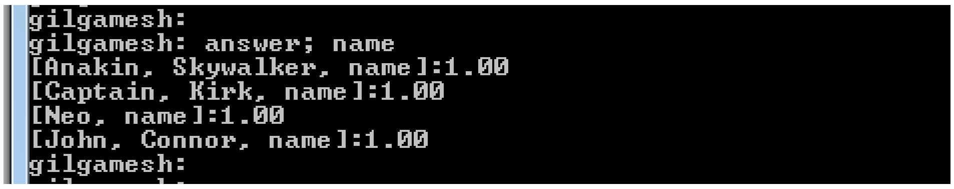
```
gilgamesh:
gilgamesh: answer; Captain Kirk skill
[Captain, Kirk, skill, Give, moral, orders, and, take, decisions, which, often,, are, most, improb
l, world.]:1.00
gilgamesh:
gilgamesh:
```

Or all information about that candidate, in a single shot:



```
gilgamesh:
gilgamesh: answer; Captain Kirk
[Captain, Kirk, experience, He, has, worked, with, Mr., Spock,, following, most, of, all, his,
, trouble., However,, when, he, is, in, trouble, due, to, not, follow, his, advices,, he, takes
ves, a, beautiful, girls, to, solve, them., Great, experience, to, take, red, shirt, crew, memb
them, die.]:1.00
[Captain, Kirk, skill, Give, moral, orders, and, take, decisions, which, often,, are, most, imp
l, world.]:1.00
[Captain, Kirk, address, Entreprise, Vessel,, where, no, man, has, gone, before]:1.00
[Captain, Kirk, name]:1.00
gilgamesh:
```

It is still possible to query all information of one type, like all names for example:



```
gilgamesh:
gilgamesh: answer; name
[Anakin, Skywalker, name]:1.00
[Captain, Kirk, name]:1.00
[Neo, name]:1.00
[John, Connor, name]:1.00
gilgamesh:
```

With those results, would be possible to generate reports, candidate summaries, common characteristics and information among them etc.

Symbol Translation (examples/symbol-translation.txt)

Gilgamesh can be used to translate texts. When I am talking about translation, I mean from any language since you provide basic information to the task. In this example, I am going to try providing one way to do that.

The first thing to know is that symbols can represent different things like syllables, things, sounds etc. For this example, I have created some symbols which will act as hieroglyphs transmitting a message. As hieroglyphs, unique symbols can have a specific meaning while combined symbols can have another one.

As smaller the information is (fact), a higher force it should have. If you intend to use this example as reference for another similar translation work, I suggest you to define a scale for fact forces, in order to fit your symbol combinations into it. For now, we just have a combination for 2 symbols what means that for our unique symbols, we will have the force 2 and for combined ones, 1. The first thing we have to do is to teach Gilgamesh about how to connect symbols and their meanings. In this case let's consider our symbols below:

Symbol	Type	Meaning
(-_-)	Unique	sleep
_[]_	Unique	door
===	Unique	path
/\/	Unique	snake
_/_	Unique	house
...	Unique	go
(:-)	Unique	happy
(o.O)	Unique	surprise
(o_o)	Unique	me
/*\	Unique	garden
_[]_ /*\	Combined	weapon
... (-_-)	Combined	death

Note that combined symbols use a combination of already known symbols to have a different meaning (like hieroglyphs). So, our message is:

```
(o_o) (-_-)
/*\ /\/ ... _/\_ _[]_
(o_o) (o.O) _[]_ /*\ ... (-_-) /\/
(o_o) (:-)
```

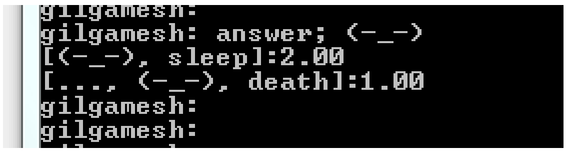
For facts, we will enter like below:

```
    Unique: fact;2;(-_-) sleep
Combined: fact;1;_[]_ /*\ weapon
```

If you are a developer using Gilgamesh source code, you could develop a system to first, trying to start finding a best match for combined symbols and then reducing the size of facts to the unique one. The decreasing force to the bigger facts is important because when you try to look for an unique symbol, it will bring you all answers which have that symbol, ordering them from the unique to the combined ones. Let's suppose you have to find a best translation until 5 combined symbols. In this case, we could use something like:

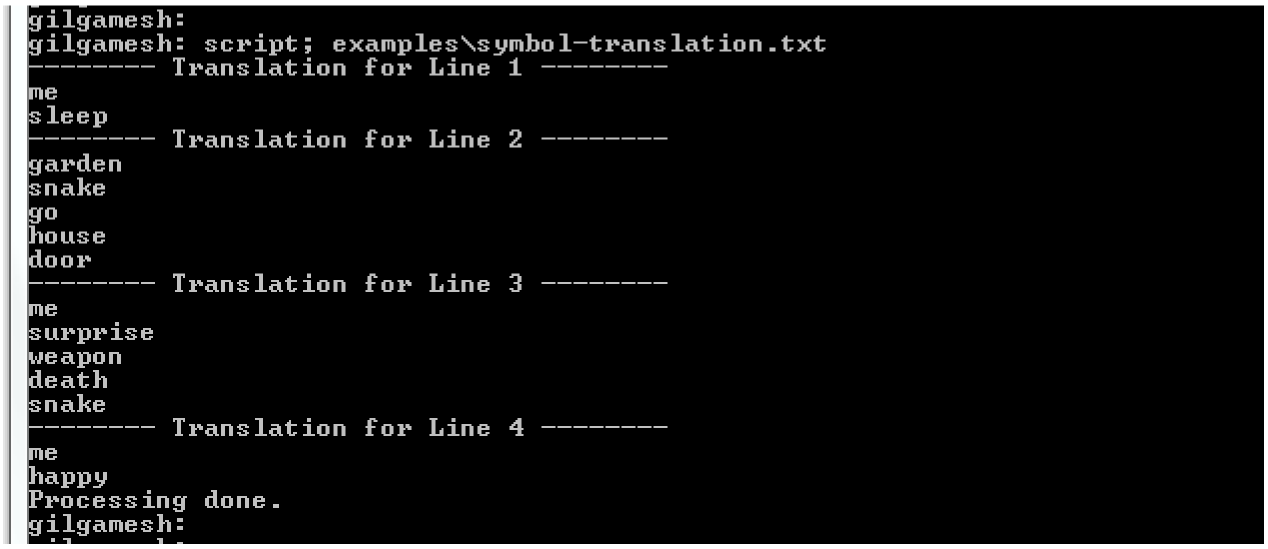
1 unique symbol = force 5
2 combined symbols = force 4
3 combined symbols = force 3
4 combined symbols = force 2
5 combined symbols = force 1

For our example, if we ask to Gilgamesh about an answer using only the symbol (-_-) which has unique and combined meanings, it would bring us first the unique answer:



```
gilgamesh:
gilgamesh: answer; (-_-)
[(-_-), sleep]:2.00
[..., (-_-), death]:1.00
gilgamesh:
gilgamesh:
```

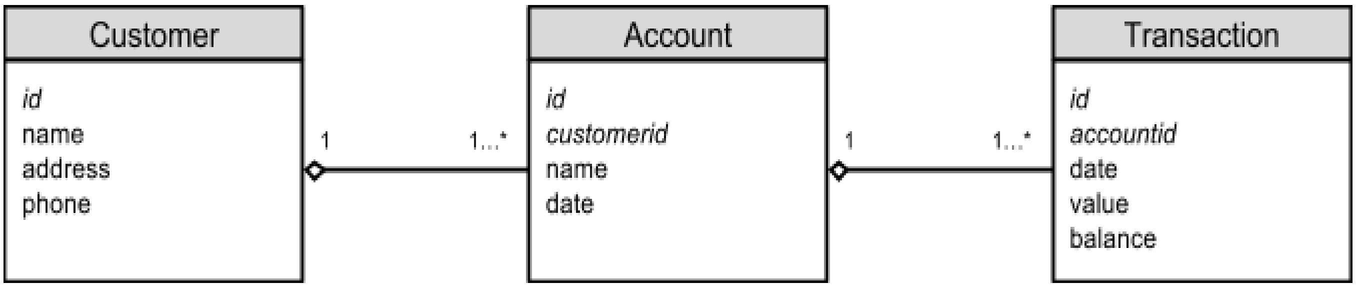
So, running the script "symbol-translation.txt" we can have the translation answer for our message above, like Indiana Jones translating a caveman's story:



```
gilgamesh:
gilgamesh: script; examples\symbol-translation.txt
-------- Translation for Line 1 --------
me
sleep
-------- Translation for Line 2 --------
garden
snake
go
house
door
-------- Translation for Line 3 --------
me
surprise
weapon
death
snake
-------- Translation for Line 4 --------
me
happy
Processing done.
gilgamesh:
```

Bank Account System (examples/bank-account.txt)

In this example we are going to do a bank account system, acting as a traditional bank account database. Let's consider the following entity relationship model:



In our system, we can have many customers. Each customer can have different accounts (current and investment) and each one of those accounts can have a lot of debit and credit transactions. Besides that, we have to control the balance for each account, which usually could be done using some calculation and recording it in the table "Transaction" inside a relational database.

To get control of each fact taught to Gilgamesh, we can create a id which will refers to just one fact inside its memory, like a single row in a database table. Actually, this is similar how a common system works: users and programmers are used to have a code to identify things in systems, which usually is a number. In this case, we will create standard code for each table above. The customers will have the code "CXXX", where "XXX" is a increasing number or counter. So, we can teach Gilgamesh as the example below:

customer C001 name Harrison Ford
customer C001 address AAA street 12
customer C001 phone 1234-1234

That allows to Gilgamesh (and you) retrieve any single and specific fact when needed. Every time when we have to include another customer, we need to increment our customer code id (e.g. C001, C002, C003 and so on). Now, let's see how we have composed the lines above:

Table	Customer ID	Column	Value
customer	C001	name	Harrison Ford
customer	C001	address	AAA street 12
customer	C001	phone	1234-1234
customer	C002	name	Angelina Jolie
customer	C002	address	BBB street 23

and so on...

For accounts, we have to include the composite id to guarantee our fact integrity:

Customer Foreign ID	Table	ID	Column	Value
C001	account	A001	type	current
C001	account	A001	date	21/04/2013
C001	account	A002	type	investment
C001	account	A002	date	22/04/2013
C002	account	A001	type	current

and so on...

Note that we have to repeat the account id (A001, A002 etc) because we have just two kind of accounts (current and investment). As we are creating a fact matching with one customer (C001, C002 etc) there is no problem to duplicate those ids. Another important point is to know that we just need to repeat the account id because we intend to generate duplicated ids to accounts. If our intention was to use a single account id for each record (fact), then would not be necessary to include customer ids (it would be working as unique key for any fact). Continuing, we will do the same for transactions:

Customer Foreign ID	Account Foreign ID	Table	ID	Date Value	Type Value	Monetary Value
C001	A001	transaction	T001	01/01/2013	debit	-35,13
C001	A001	transaction	T002	01/01/2013	credit	48,03
C001	A001	transaction	T003	01/01/2013	credit	15,94
C001	A001	transaction	T004	01/01/2013	credit	31,59
C002	A002	transaction	T005	01/01/2013	debit	-34,95

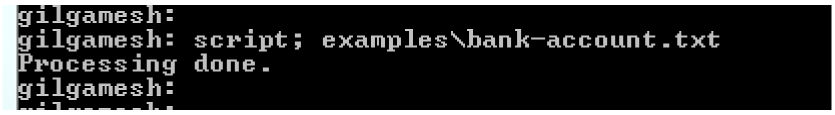
and so on...

For transactions, we will create a link among date, type of transaction (debit/credit) and its value. We need to keep that structure to ensure the fact will describe one specific condition that happens sometime. And we have to input balances for each account and day (note that we split the balance as another table):

Customer Foreign ID	Account Foreign ID	Table	Date Value	Monetary Value
C001	A001	balance	01/01/2013	159,90
C001	A001	balance	02/01/2013	192,92
C001	A001	balance	03/01/2013	309,85

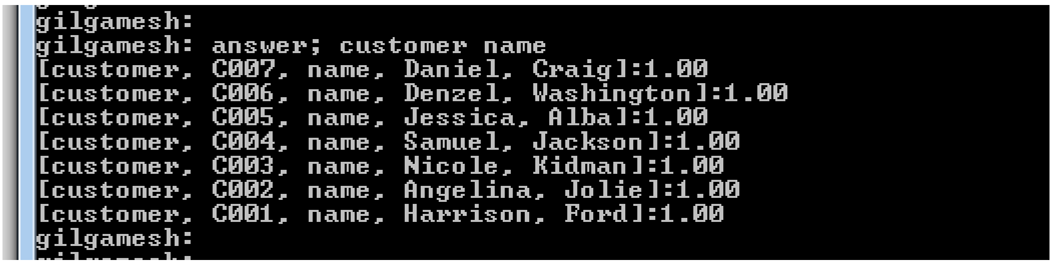
and so on...

Now, let's execute our script to load all data for this example:

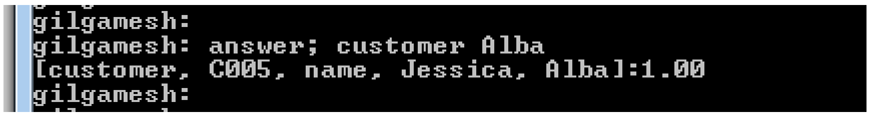

```
gilgamesh:
gilgamesh: script; examples\bank-account.txt
Processing done.
gilgamesh:
```

And let's see what we can do:

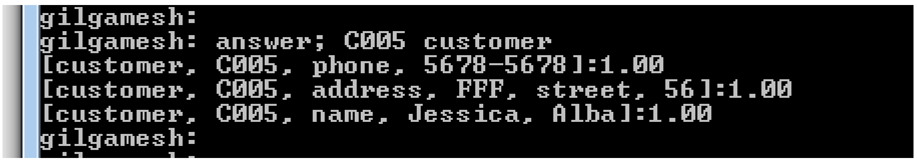
- To query all customer names:


```
gilgamesh:
gilgamesh: answer; customer name
[customer, C007, name, Daniel, Craig]:1.00
[customer, C006, name, Denzel, Washington]:1.00
[customer, C005, name, Jessica, Alba]:1.00
[customer, C004, name, Samuel, Jackson]:1.00
[customer, C003, name, Nicole, Kidman]:1.00
[customer, C002, name, Angelina, Jolie]:1.00
[customer, C001, name, Harrison, Ford]:1.00
gilgamesh:
```

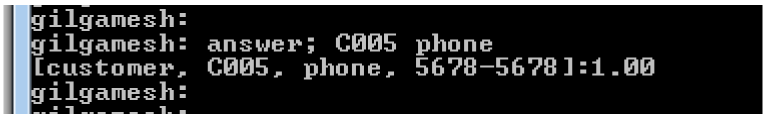
- To query the customer code using a surname:


```
gilgamesh:
gilgamesh: answer; customer Alba
[customer, C005, name, Jessica, Alba]:1.00
gilgamesh:
```

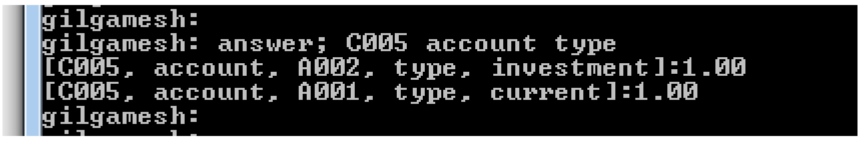
- To query all customer information from her code:


```
gilgamesh:
gilgamesh: answer; C005 customer
[customer, C005, phone, 5678-5678]:1.00
[customer, C005, address, FFF, street, 56]:1.00
[customer, C005, name, Jessica, Alba]:1.00
gilgamesh:
```

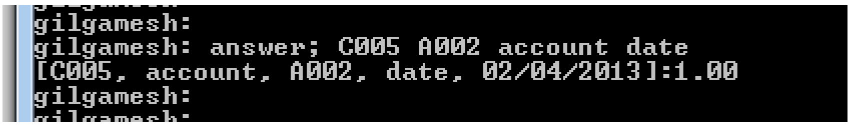
- To query a specific information (phone) from her code:


```
gilgamesh:
gilgamesh: answer; C005 phone
[customer, C005, phone, 5678-5678]:1.00
gilgamesh:
```

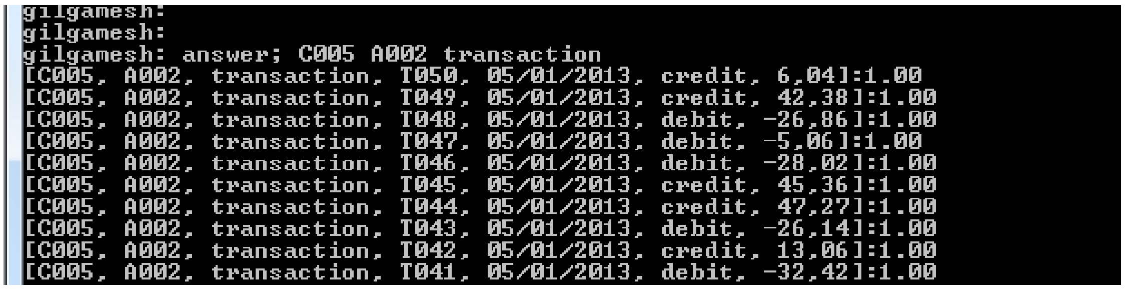
- To query which account types she has:


```
gilgamesh:
gilgamesh: answer; C005 account type
[C005, account, A002, type, investment]:1.00
[C005, account, A001, type, current]:1.00
gilgamesh:
```

- To query when she has created the investment account:

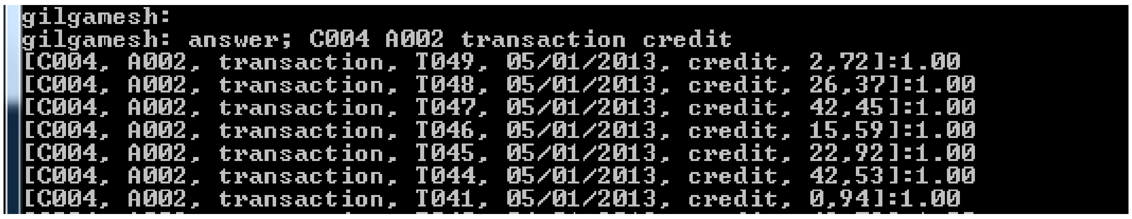

```
gilgamesh:
gilgamesh: answer; C005 A002 account date
[C005, account, A002, date, 02/04/2013]:1.00
gilgamesh:
```

- To query all transactions from her investment account:



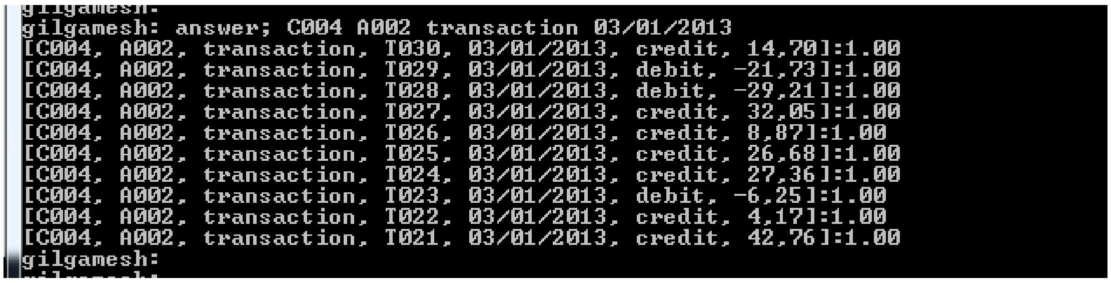
```
gilgamesh:
gilgamesh:
gilgamesh: answer; C005 A002 transaction
[C005, A002, transaction, T050, 05/01/2013, credit, 6,04]:1.00
[C005, A002, transaction, T049, 05/01/2013, credit, 42,38]:1.00
[C005, A002, transaction, T048, 05/01/2013, debit, -26,86]:1.00
[C005, A002, transaction, T047, 05/01/2013, debit, -5,06]:1.00
[C005, A002, transaction, T046, 05/01/2013, debit, -28,02]:1.00
[C005, A002, transaction, T045, 05/01/2013, credit, 45,36]:1.00
[C005, A002, transaction, T044, 05/01/2013, credit, 47,27]:1.00
[C005, A002, transaction, T043, 05/01/2013, debit, -26,14]:1.00
[C005, A002, transaction, T042, 05/01/2013, credit, 13,06]:1.00
[C005, A002, transaction, T041, 05/01/2013, debit, -32,42]:1.00
```

- To query all her credit transactions from investment account:



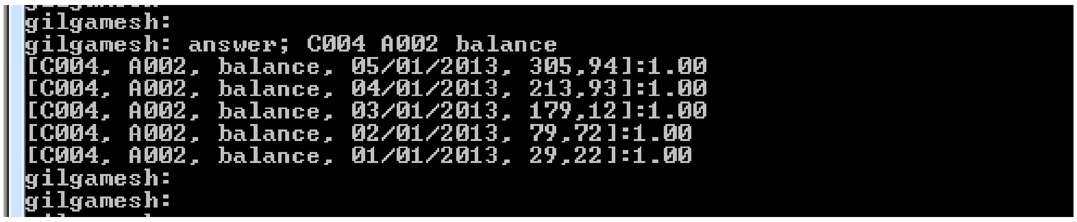
```
gilgamesh:
gilgamesh: answer; C004 A002 transaction credit
[C004, A002, transaction, T049, 05/01/2013, credit, 2,72]:1.00
[C004, A002, transaction, T048, 05/01/2013, credit, 26,37]:1.00
[C004, A002, transaction, T047, 05/01/2013, credit, 42,45]:1.00
[C004, A002, transaction, T046, 05/01/2013, credit, 15,59]:1.00
[C004, A002, transaction, T045, 05/01/2013, credit, 22,92]:1.00
[C004, A002, transaction, T044, 05/01/2013, credit, 42,53]:1.00
[C004, A002, transaction, T041, 05/01/2013, credit, 0,94]:1.00
```

- To query all her transactions in a specific date from investment account:



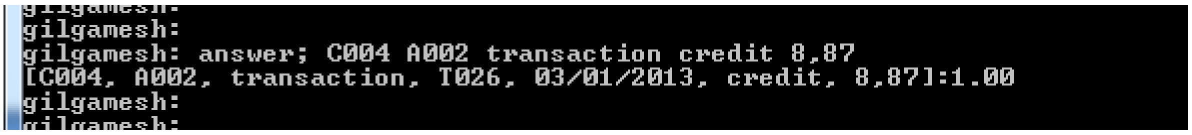
```
gilgamesh:
gilgamesh: answer; C004 A002 transaction 03/01/2013
[C004, A002, transaction, T030, 03/01/2013, credit, 14,70]:1.00
[C004, A002, transaction, T029, 03/01/2013, debit, -21,73]:1.00
[C004, A002, transaction, T028, 03/01/2013, debit, -29,21]:1.00
[C004, A002, transaction, T027, 03/01/2013, credit, 32,05]:1.00
[C004, A002, transaction, T026, 03/01/2013, credit, 8,87]:1.00
[C004, A002, transaction, T025, 03/01/2013, credit, 26,68]:1.00
[C004, A002, transaction, T024, 03/01/2013, credit, 27,36]:1.00
[C004, A002, transaction, T023, 03/01/2013, debit, -6,25]:1.00
[C004, A002, transaction, T022, 03/01/2013, credit, 4,17]:1.00
[C004, A002, transaction, T021, 03/01/2013, credit, 42,76]:1.00
gilgamesh:
```

- To query all her balances for investment account:



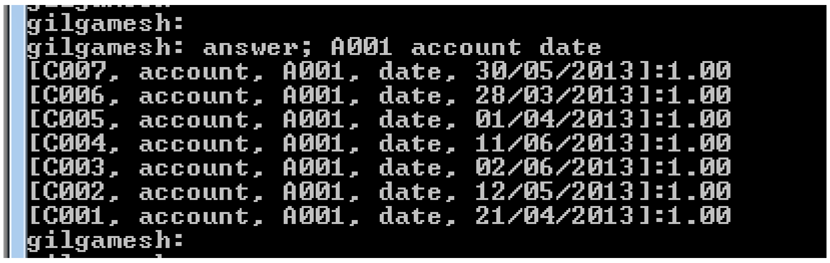
```
gilgamesh:
gilgamesh: answer; C004 A002 balance
[C004, A002, balance, 05/01/2013, 305,94]:1.00
[C004, A002, balance, 04/01/2013, 213,93]:1.00
[C004, A002, balance, 03/01/2013, 179,12]:1.00
[C004, A002, balance, 02/01/2013, 79,72]:1.00
[C004, A002, balance, 01/01/2013, 29,22]:1.00
gilgamesh:
gilgamesh:
```

- To query if she has done a $8,87 credit deposit:



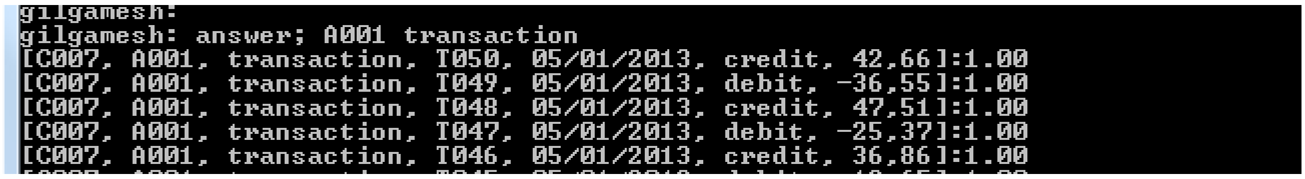
```
gilgamesh:
gilgamesh: answer; C004 A002 transaction credit 8,87
[C004, A002, transaction, T026, 03/01/2013, credit, 8,87]:1.00
gilgamesh:
gilgamesh:
```

- To query all dates from all customers current accounts:



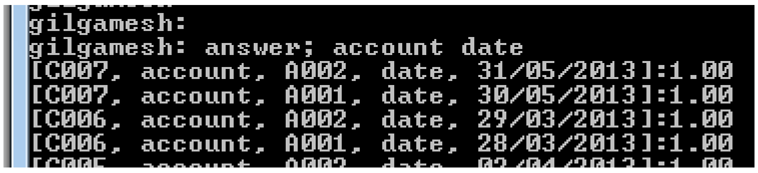
```
gilgamesh:
gilgamesh: answer; A001 account date
[C007, account, A001, date, 30/05/2013]:1.00
[C006, account, A001, date, 28/03/2013]:1.00
[C005, account, A001, date, 01/04/2013]:1.00
[C004, account, A001, date, 11/06/2013]:1.00
[C003, account, A001, date, 02/06/2013]:1.00
[C002, account, A001, date, 12/05/2013]:1.00
[C001, account, A001, date, 21/04/2013]:1.00
gilgamesh:
```

- To query all transactions for current accounts:



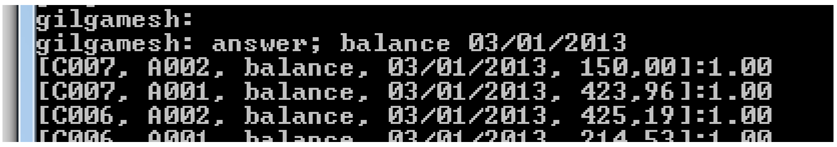
```
gilgamesh:
gilgamesh: answer; A001 transaction
[C007, A001, transaction, T050, 05/01/2013, credit, 42,66]:1.00
[C007, A001, transaction, T049, 05/01/2013, debit, -36,55]:1.00
[C007, A001, transaction, T048, 05/01/2013, credit, 47,51]:1.00
[C007, A001, transaction, T047, 05/01/2013, debit, -25,37]:1.00
[C007, A001, transaction, T046, 05/01/2013, credit, 36,86]:1.00
```

- To query all account dates from all customers:



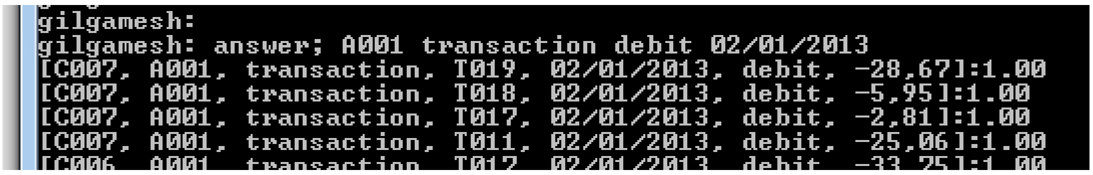
```
gilgamesh:
gilgamesh: answer; account date
[C007, account, A002, date, 31/05/2013]:1.00
[C007, account, A001, date, 30/05/2013]:1.00
[C006, account, A002, date, 29/03/2013]:1.00
[C006, account, A001, date, 28/03/2013]:1.00
```

- To query all balances for all accounts in a specific date:



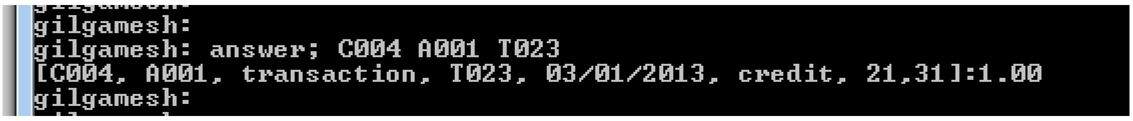
```
gilgamesh:
gilgamesh: answer; balance 03/01/2013
[C007, A002, balance, 03/01/2013, 150,00]:1.00
[C007, A001, balance, 03/01/2013, 423,96]:1.00
[C006, A002, balance, 03/01/2013, 425,19]:1.00
```

- To query all debit transactions from all customers which happens in a specific date:



```
gilgamesh:
gilgamesh: answer; A001 transaction debit 02/01/2013
[C007, A001, transaction, T019, 02/01/2013, debit, -28,67]:1.00
[C007, A001, transaction, T018, 02/01/2013, debit, -5,95]:1.00
[C007, A001, transaction, T017, 02/01/2013, debit, -2,81]:1.00
[C007, A001, transaction, T011, 02/01/2013, debit, -25,06]:1.00
```

- To query a specific customer, account and transaction ids by codes:

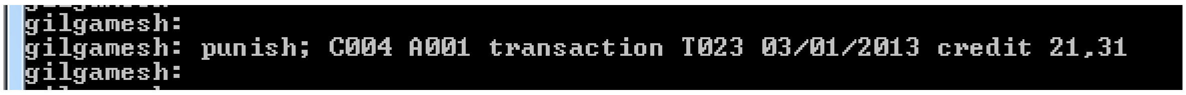


```
gilgamesh:
gilgamesh: answer; C004 A001 T023
[C004, A001, transaction, T023, 03/01/2013, credit, 21,31]:1.00
gilgamesh:
```

Now, let's suppose you want to update or delete one fact. There are two ways you can do that:

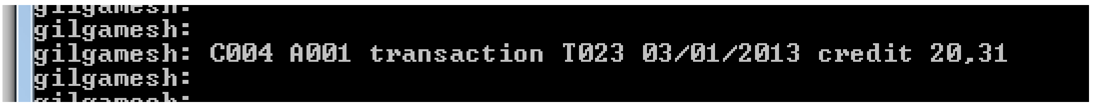
- You can remove the fact from Gilgamesh memory, calling "remove" command or
- You can give a different weight for that fact to keep a history of changes;

If you intend to keep the history of changes, you can punish a fact like:



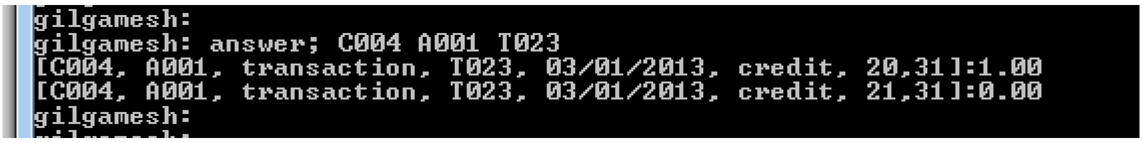
```
gilgamesh:
gilgamesh: punish; C004 A001 transaction T023 03/01/2013 credit 21,31
gilgamesh:
```

Then, you can enter the right fact as below:



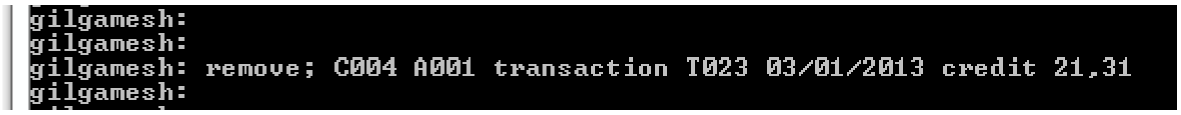
```
gilgamesh:
gilgamesh: C004 A001 transaction T023 03/01/2013 credit 20,31
gilgamesh:
```

And when you query about that fact again, it will bring to you the right one first:



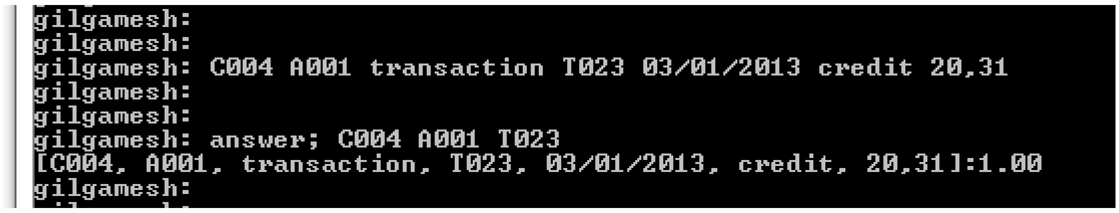
```
gilgamesh:
gilgamesh: answer; C004 A001 T023
[C004, A001, transaction, T023, 03/01/2013, credit, 20,31]:1.00
[C004, A001, transaction, T023, 03/01/2013, credit, 21,31]:0.00
gilgamesh:
```

Or if you want to remove the old fact and insert a new one:



```
gilgamesh:
gilgamesh:
gilgamesh: remove; C004 A001 transaction T023 03/01/2013 credit 21,31
gilgamesh:
```

And insert the new one:



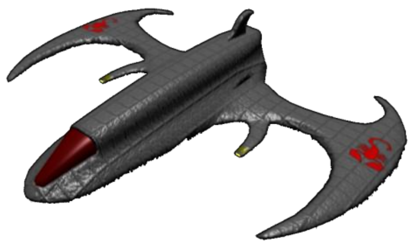
```
gilgamesh:
gilgamesh:
gilgamesh: C004 A001 transaction T023 03/01/2013 credit 20,31
gilgamesh:
gilgamesh:
gilgamesh: answer; C004 A001 T023
[C004, A001, transaction, T023, 03/01/2013, credit, 20,31]:1.00
gilgamesh:
```

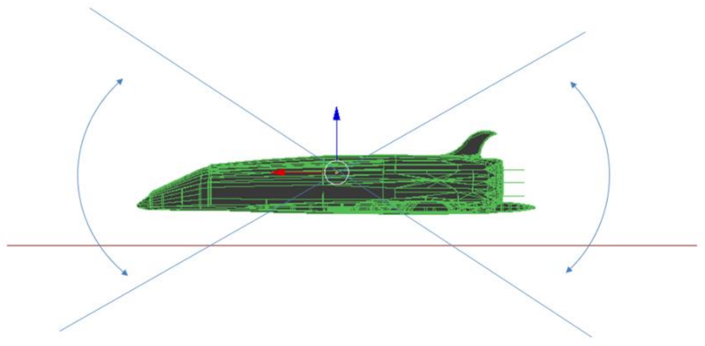
That’s it.

Drone, Robot and Game Learning (examples/drone-controller.txt)

Let's suppose you want to make a intelligent drone or robot which is capable to learn by itself some body's control or movement. It is possible to use Gilgamesh's code to make it learn positions getting information from movement sensors (you probably would do that using the source code, but the concept can be demonstrated through Gilgamesh console). The idea of this example is to show how Gilgamesh could learn and use its memory in an ongoing process, as soon as its memory is increasing up. That idea could be used in games too, to control objects or AI characters for a specific task.

Now, let's suppose you have a drone or space vessel in a game which should learn the rotation in one axis. The best angle is 0, identifying that your vessel is in the best rotation to fly (flat position). The Gilgamesh memory should know what is the correction to that angle which would put your vessel again in the best rotation (eg.: if the angle of your vessel is 15 degrees, then the memory has to know that it should correct the angle in -15 degrees to reach 0 for rotation). For each measure of time, a sensor like a gyroscope sends you the current rotation of your object, telling in which degree it is:





To make the code learn, the execution will be in a loop, monitoring and running a logic in this order:

1. get the current sensor rotation value;
2. if there is a correction value in Gilgamesh's memory for that angle:

   - get it and apply a "mutation" on it (small increment or decrement value) or

   - generate any random value;
3. apply the obtained correction
4. check the result after apply the correction and keep it in Gilgamesh's memory;

Let's make some assumptions:

- if the correction value applied makes our vessel turns its axis near 0 degrees, then we can consider it as a good value. Otherwise, would be a bad correction since our vessel didn't turn near 0 degrees to the flat position. In this case, we can create a measure to indicate that, as if it would be the sensor response after apply the correction (this result will be used as fact force in Gilgamesh's memory):

  result = (minimum value between correction and rotation) - (maximum value between correction and rotation)

  Eg.: rotation = 14, correction = 3 (any random value at first time) then the result = (3 - 14) = -11

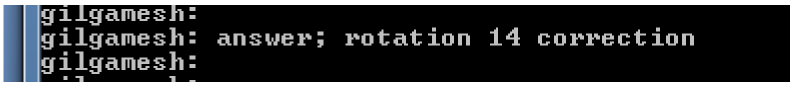
  The -11 is the result after we apply our correction (which will be a random value at first time). This value represents a sensor response indicating what happens after we apply our correction. Any value near 0 is a good value (our vessel rotation is near the flat position: 0 degrees).

- the mutation value is necessary to have variations in our correction value. Think of it like a genetic mutation that happens naturally during new generations. Actually, it could be interpreted like that, since the result of our correction force could be understood like natural selection :) The mutation assumed here will be among -1, 0 or +1 values.

Now, let's simulate the learning process.

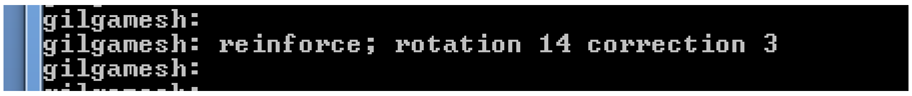
Time: Second 1

The sensor tells to Gilgamesh that the vessel rotation is 14 degrees. We have to check if there is a correction value for this angle:



```
gilgamesh:
gilgamesh: answer; rotation 14 correction
gilgamesh:
```

No values. So as we don't have any correction value in its memory, it will generate a random value. Let's guess any value, for example 3. In this case, we can teach it to apply this correction, reinforcing the value since any correction value is better than nothing:

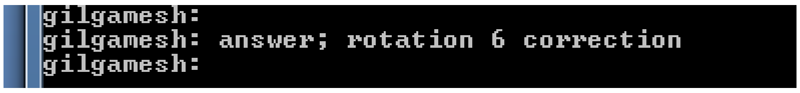


```
gilgamesh:
gilgamesh: reinforce; rotation 14 correction 3
gilgamesh:
```

Ok, now it knows what was the rotation and the correction applied.

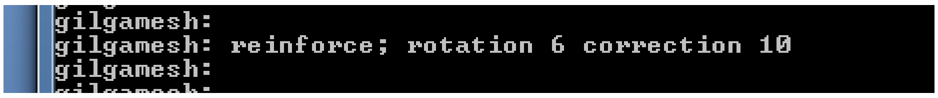
Time: Second 2

The sensor tells to Gilgamesh that the vessel rotation is 6 degrees. Let's search again for previous correction values:



```
gilgamesh:
gilgamesh: answer; rotation 6 correction
gilgamesh:
```

No values too. As we don't have any correction value for this angle, we will generate another random value again. Let's guess another value, let's suppose, 10. Teaching it:

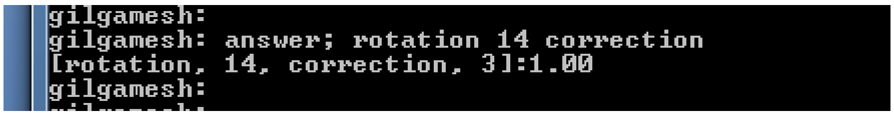


```
gilgamesh:
gilgamesh: reinforce; rotation 6 correction 10
gilgamesh:
```

After that, we have a correction value for angle 6 and random values continue to be inserted until we get a value from Gilgamesh's memory.

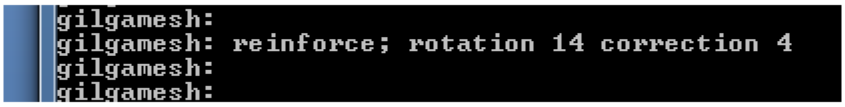
Time: Second 3

The sensor tells to Gilgamesh that the vessel rotation is 14 degrees again. Let's search again for a correction value:



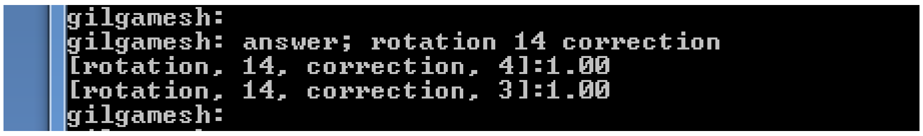
```
gilgamesh:
gilgamesh: answer; rotation 14 correction
[rotation, 14, correction, 3]:1.00
gilgamesh:
```

So now we found a correction value. Before consider this correction value, let's sum 1 as a mutation value (which could be -1, 0 or 1). In this case, our correction value will be 4 (3+1) and now we should compare them: -10 (4-14) is greater than the previous result -11 (3-14)? If yes, we reinforce the new correction, otherwise we reinforce the old value. In this case:



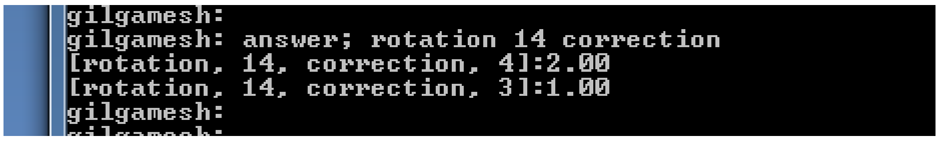
```
gilgamesh:
gilgamesh: reinforce; rotation 14 correction 4
gilgamesh:
gilgamesh:
```

After this point, we can see what Gilgamesh is doing "under the hood":



```
gilgamesh:
gilgamesh: answer; rotation 14 correction
[rotation, 14, correction, 4]:1.00
[rotation, 14, correction, 3]:1.00
gilgamesh:
```

The mutation value added to the correction, has generated a better axis adjustment (towards to 0 degrees) for our rotation. So when this step happens again, we will have:

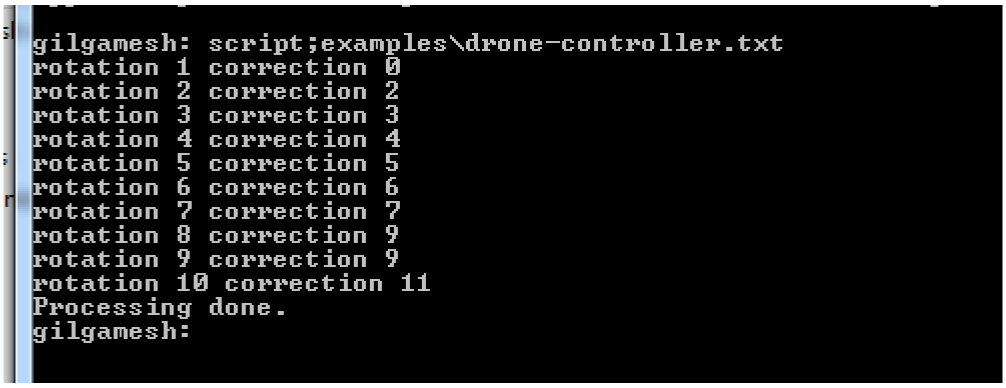


```
gilgamesh:
gilgamesh: answer; rotation 14 correction
[rotation, 14, correction, 4]:2.00
[rotation, 14, correction, 3]:1.00
gilgamesh:
```

And so on.

Getting Answers

The script "drone-controller.txt" into examples folder gives an idea about that kind of control. After run that script which execute those steps above a hundred times, you will get values like:



```
gilgamesh: script;examples\drone-controller.txt
rotation 1 correction 0
rotation 2 correction 2
rotation 3 correction 3
rotation 4 correction 4
rotation 5 correction 5
rotation 6 correction 6
rotation 7 correction 7
rotation 8 correction 9
rotation 9 correction 9
rotation 10 correction 11
Processing done.
gilgamesh:
```

This happens because Gilgamesh will be learning a better value after each iteration, creating a real memory of which value is better to achieve the flat position for our vessel (0 degrees). As more as you input data, more accurate will be its answers. Note that for rotation 1, 8 and 10 it didn't get the best answer yet because we had entered 100 lines of information. If we continue this process, it would have best answers soon.

Now, think that you can use it in any kind of sensor you want. This is one way of many that it could be used in a robot, drone or game object, to make it learn during the running time.

Identifying Signal Patterns (examples/alien-signal.txt)

Gilgamesh can be used to identify some kind of signal pattern in a sequence of data without too much effort. Let's suppose that you work on SETI and you have the duty to analyse some piece of information received from outer space. You want to identify if the signal received has any pattern on it, which could mean a signal sent from any intelligent life form. In this case, let's consider that our signal is formed by letters (not numbers, to keep it easy) and you have a signal like that:

D M Z Q R O D K W F N B C Q Y T L M J X ...

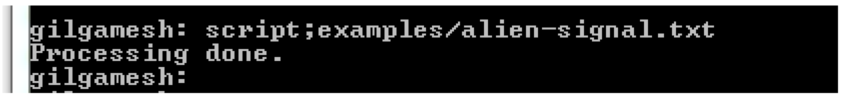
As we don't have any idea how big could be a pattern, let's starting testing it from 2 and 10 combinations. For each step in the sequence, we will insert into Gilgamesh's memory a range of numbers from 2 to 10. Eg.:

Step 1:
D M (from 1 to 2 letters)
D M Z (from 1 to 3 letters)
D M Z Q (from 1 to 4 letters)
D M Z Q R (from 1 to 5 letters)
D M Z Q R O (from 1 to 6 letters)
D M Z Q R O D (from 1 to 7 letters)
D M Z Q R O D K (from 1 to 8 letters)
D M Z Q R O D K W (from 1 to 9 letters)
D M Z Q R O D K W F (from 1 to 10 letters)

Step 2 (walks 1 letter in the sequence):
M Z (from 1 to 2 letters)
M Z Q (from 1 to 3 letters)
M Z Q R (from 1 to 4 letters)
M Z Q R O (from 1 to 5 letters)
M Z Q R O D (from 1 to 6 letters)
M Z Q R O D K (from 1 to 7 letters)
M Z Q R O D K W (from 1 to 8 letters)
M Z Q R O D K W F (from 1 to 9 letters)
M Z Q R O D K W F N (from 1 to 10 letters)

And so on.

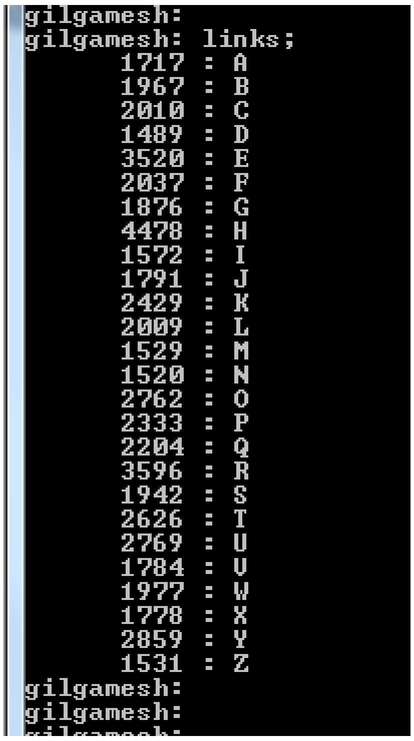
To run our example, execute the script "alien-signal.txt":



```
gilgamesh: script;examples/alien-signal.txt
Processing done.
gilgamesh:
```

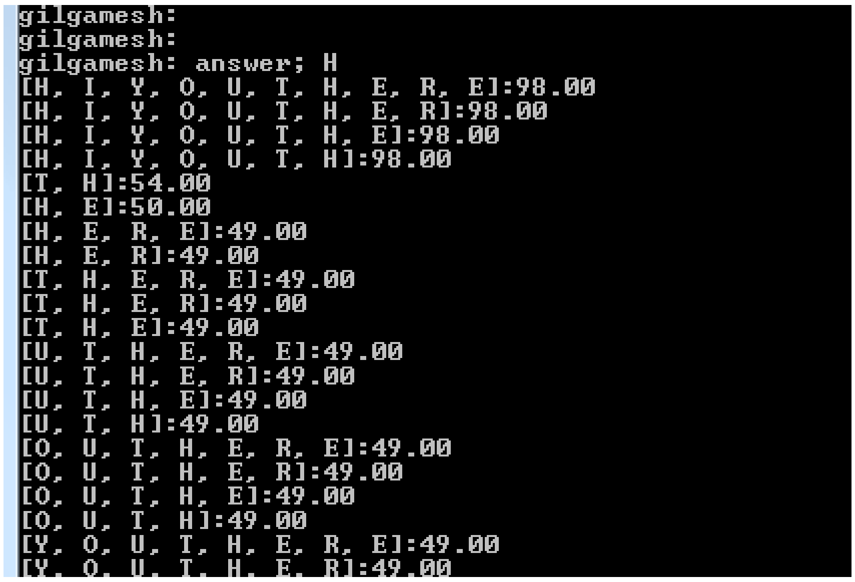
After this step, Gilgamesh knows all patterns into the signal and we are able to start analyzing the signal.

Let's take a look for most common atoms calling "links" command:



```
gilgamesh:
gilgamesh: links;
       1717 : A
       1967 : B
       2010 : C
       1489 : D
       3520 : E
       2037 : F
       1876 : G
       4478 : H
       1572 : I
       1791 : J
       2429 : K
       2009 : L
       1529 : M
       1520 : N
       2762 : O
       2333 : P
       2204 : Q
       3596 : R
       1942 : S
       2626 : T
       2769 : U
       1784 : V
       1977 : W
       1778 : X
       2859 : Y
       1531 : Z
gilgamesh:
gilgamesh:
```

We can see that the most common letter is "H" - that gives us a clue. Now, if we ask about most common pattern using "H" we will see:



```
gilgamesh:
gilgamesh:
gilgamesh: answer; H
[H, I, Y, O, U, T, H, E, R, E]:98.00
[H, I, Y, O, U, T, H, E, R]:98.00
[H, I, Y, O, U, T, H, E]:98.00
[H, I, Y, O, U, T, H]:98.00
[T, H]:54.00
[H, E]:50.00
[H, E, R, E]:49.00
[H, E, R]:49.00
[T, H, E, R, E]:49.00
[T, H, E, R]:49.00
[T, H, E]:49.00
[U, T, H, E, R, E]:49.00
[U, T, H, E, R]:49.00
[U, T, H, E]:49.00
[U, T, H]:49.00
[O, U, T, H, E, R, E]:49.00
[O, U, T, H, E, R]:49.00
[O, U, T, H, E]:49.00
[O, U, T, H]:49.00
[Y, O, U, T, H, E, R, E]:49.00
[Y, O, U, T, H, E, R]:49.00
```

Where the most common pattern is "H I Y O U T H E R E" with force as 98: intelligent alien life discovered! :D