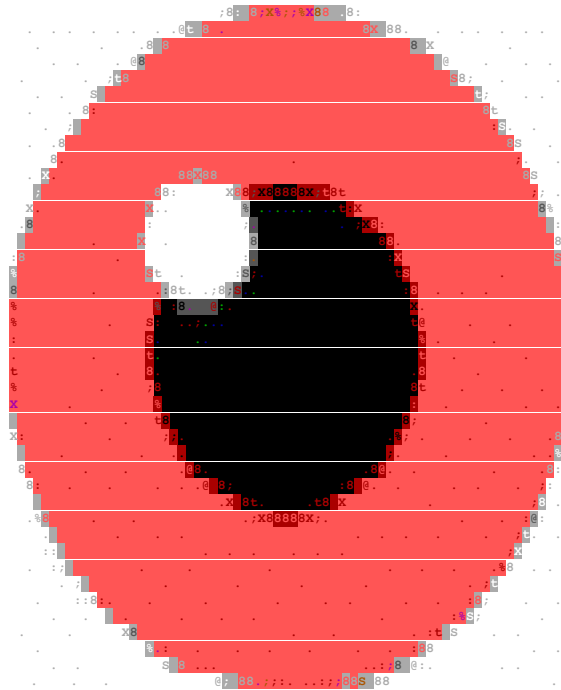


# SHEP Automatic learning AI

v0.0.9



By Dexter RC Shepherd, Undergraduate at the University of Sussex

## Contents

Overview .....	3
Technicalities.....	3
System Requirements .....	3
Test plan.....	3
AI code .....	3
Main SHEP .....	4
User interface.....	4
Development.....	4
AI code .....	4
Personality file .....	4
Data management .....	4
Negative feedback .....	7
Topic break up.....	7
Main SHEP .....	7
User interface.....	7
Testing.....	9
AI code .....	9
Main SHEP .....	10
User interface.....	10
Deployment of the system.....	10
Libraries needed .....	10
Environment .....	11
Future versions .....	11
Legal .....	11
References .....	11

## Overview

Basing off of the last project, the University chatbot [1], I will like to explore the concept of self-learning chat bots. Self-learning chat bots remove the need for an admin making the bot more adaptable to situations.

There are different ways this admin can be replaced, through a controlled method where you would have a datafile of information it can take information from. Alternatively you could allow it to learn off the user, or have something in between. Learning off the user is a quicker method of learning but raises dangers that the AI may take a different path to its set up task [2]. I would like to make an AI which will use a bit of both, its datafile will contain information about itself and information on the project. This is where it will learn it's personality. Everything else will be learned off of user interaction using classical conditioning [3] approaches.

## Technicalities

### System Requirements

The program will be using the old system method of storing the graph in the dictionary. This means the more responses the more replies. I could consider making a file structure dictionary system, to take the pressure off the RAM. This code should run on any normal computer. I hope to put it onto a Raspberry Pi.

The Raspberry Pi offers the ability to embed code on computers to run like a micro-controller. I would then be able to implement a microphone and the Google text to speech library to validate text input for the system.

The user interface will use an LED matrix eye. I will use an i2c microphone with the Raspberry Pi and some other sensors for later versions.

## Test plan

### AI code

Test No	Test	Expected outcome
1	Creates a log of the conversations	In the folder, there will be each log with unique filenames
2	Finds the right database logs to point to	A phrase which points to some set data logs will check these two files. A print statement will be used to show it is searching each datafile.
3	The code finds relevant responses to the input.	The system will answer the question "how are you" with the phrase which is set in one of its databases.
4	The code finds the most relevant output out of a series of outputs saved.	The code will check different files and find which file is the most related
5	The code is adding new information and ways to respond to this information all the time.	When a question is unknown, the AI will ask it back to the user in the most natural way possible.
6		

## Main SHEP

Test No	Test	Expected outcome
1	Functions I expected way in previous version	Yes

## User interface

Test No	Test	Expected outcome
1	Displays an eye	The SHEP eye is shown constantly
2	The eye blinks every so often	Ever few seconds the eye blinks
3	The AI shows it is listening	The eye shows a green dot when it is listening for audio and turns off when it is not.
4	Does the AI take in your voice and convert to text	Yes

## Development

### AI code

#### Personality file

The personality file will be used and checked in the beginning. If an answer is not found then it will continue with the normal process. If it is found then that will be returned as the output no ifs or buts. This file will contain data such as "my name is SHEP" and other information about the software, the project, the creator and so on.

This will be a simple text file called "personality.txt".

#### Data management

This section is about the way in which different bits of information is stored and how it impacts the greater system. Also solutions to keep the system fast and sustainable, meaning it will not overspill the RAM or memory. To make a lifelong AI it will need to make sure databases aren't just logging for the sake of it if the data doesn't add anything to the system. Also current variables should not be taking up all the RAM.

The data will be using a couple of graphs to organize what the meaning of a sentence is, and which database logs of conversation each term points to. Each database will then be converted into an array of sentences which flow into each other. It will also be converted to an array of broken up language. This is so the general feel of the conversation can be matched with the current conversation. This will give greater accuracy when finding a response.

```

it is
>what is your name
what is your name
>my name is SHEP
my name is SHEP
>that is a nice name
that is a nice name
>thank you
where are you from
>i am from the UK
that is cool
>
= RESTART: C:\Users\Dexter Shepherd\Documents\AI\LibrarySHEP
>what is your name
2020-07-14-18-00-01-938868
my name is SHEP
>my name is Dexter
2020-07-14-18-00-01-938868
that is a nice name
>thank you
2020-07-14-18-00-01-938868
where are you from
>i am from Brighton
2020-07-14-18-00-01-938868
that is cool
>
= RESTART: C:\Users\Dexter Shepherd\Documents\AI\LibrarySHEP
>hi
hello
>what is your favourite thing to do
what is your favourite thing to do
>i enjoy programming
i enjoy programming
>that is a cool hobby
that is a cool hobby
>
= RESTART: C:\Users\Dexter Shepherd\Documents\AI\LibrarySHEP
>what is your favourite thing to do
i enjoy programming
>cool
thanks
>where are you from
i am from the uk
>

```

Above shows the process of this learning. Where each conversation is carried through and additions made in a datafile log of conversations and called upon when phrases point to this database. There will be potential memory issues with the code as more and more responses get added, but I've got a few ideas in place to fix these. This will be something I get to later down the line.

My next task is to make to code find the most relevant conversation to take the data from. I did this by splitting the last 100 or less words in a conversation and the current conversation down into elements of the language. Then it calculates the sum of each phrase similarity, and converts it to a percentage.

$$P = \frac{\sum S}{w}, w \leq 100$$

Where w is the amount of words in the log, an S is the probabilities of similarity of words with the current conversation.

Below shows the output files and the probability of it being the right file to find an answer. It bases it off of how similar the conversations are.

```

>where are the apples
2020-07-15-10-41-24-870813 0.81
i am not sure
>where are you from
2020-07-15-10-31-23-799939 0.675
i am from the uk
>
= RESTART: C:\Users\Dexter Shepherd\Documents\AI\LibraryS
>what is your name
2020-07-14-18-00-42-137522 0.81
my name is SHEP
>hi
2020-07-14-17-58-46-466391 1.0383333333333333
hello
>hello
testConvo 1.0522222222222224
how are you
>i am good
2020-07-14-18-00-01-938868 0.8303333333333334
perfect
>

```

I was happy with this so far, but found that there would be a potential problem that it always has the same answer for everything. It can only learn new responses if the bot asks you something and you respond differently, then this file is picked based on it's conversational similarity. I will need to get the bot to experiment with it's own language more often. When stuck for a response, perhaps take responses out of the confused file.

When the conversations are not exactly found in the files, and the response is going to be a response but not related (ie a low probability) instead of returning it and reinforcing it's own mistakes, it could perhaps set this conversation as confused, and try and lead with the users questions in another conversation. It would then delete this conversation from being used as its trained data.

I added in a method which would stop saving to the conversation log and stop directing data there. This is so it will not keep training its mistakes. It will still attempt to respond using it's potentially incorrect data, but will now have a log file of confused data to try out.

After thinking a lot, learning paths of conversations is very difficult as it is open to many changes. I could potentially keep log files in the confused folder get picked based on similarity to the current conversation. If similar, then it will use the last outputted phrase and set this file as the current log. The response the user gives will then be the response. I could apply classical conditioning to get the most frequent reply but for now will just do it this way.

When implementing this, suddenly the code started writing loads of blank files for no reason. This delayed my ability to develop this. Once I solved this issue (I think it was where old files were deleted), I worked back on the "find how to respond and learn" function.

After adding this in, the system was appearing more natural in the way it learns. It was making mistakes, but I think it will be less of a problem once it has learned more conversations.

The responses were starting to be the same one every time. I would like this not to be the case to appear less robotic.

```
>hello
['how are you', 'how are you', 'how are you', 'how are you', 'hello', 'how are you', 'how are you', '
how are you', 'hi', 'how are you']
how are you
>i am good
['i am okay', 'i am okay', 'i am okay', 'i am okay', 'i am okay']
i am okay
>great
['that is great']
that is great
>
```

Here we have an array of potential outputs linked with each interaction. It will pick a response at random. The AI is in it's "dumb" stage. I will get lots of conversational data and train it. I will also experiment with it by implementing a different graph data structure. This will take pressure off the ram and save the data within a file system.

The ram was secured, but the system slowed down considerably. This might be because the new graph data structure is reading every single file.

- 466
- 467
- 468
- 469
- 470
- 471
- 472
- 473

This was an issue, but could be solved by having a page indexer as some sort of database to find the wanted item.

After doing this it improved the feel of the system. I will limit the amount of times the same response can appear so it doesn't become OP in the response results.

After tweaking bits of the code I found it to be flowing a it more naturally. The program would occasionally respond randomly, but that is because it still has much to learn, and it is learning all the time.

### Negative feedback

If the system gets something wrong, how will it correct itself? The negative feedback is already written into the SHEP code, but the AI will need a way to trigger it. Perhaps phrases, or even volume of microphone if using voice recognition. I will leave this to the user by using a method to do all this.

What this negative feedback method will need to do is remove the answer to the question, and then add it to the confused file. When it outputs a statement rather than a question, it could on occasion Follow this up by outputting the confused file data and trying to find a response.

### Topic break up

After researching Cleverbot [4] I discovered that breaking has been done in more efficient ways in the past than the first plan I made to do this. Instead of building up topics and searching each response, why not gather all the possible responses to questions it has learned. "How are you" → "I am good", "I am sad", "I am okay" and check which time it was used shares the most in common with the current. This would mean storing information about each time data is used.

Every conversation log with the user will be kept and used to find responses. This will then be taken further to find which text file matches the user input the most.

When I added the return a random choice from all the potential outputs, I increased the probability on the return of an exact match with the conversation. This was to prevent "hi" always being "hello". Instead what I need to do is gather all the top responses as well as the second to top responses. Then return a random choice form this as well.

### Main SHEP

Main SHEP will remain mostly the same, with some changes. These changes will be to split up processes to be used to search the data. I will use SHEP and the NLP to break down information and get it linking to a key within the AI program.

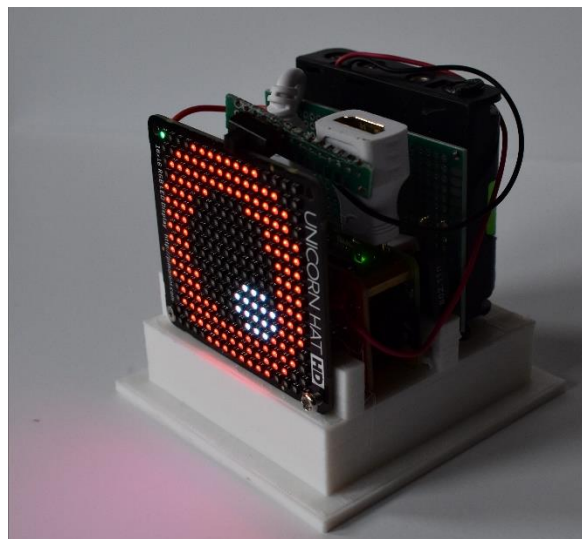
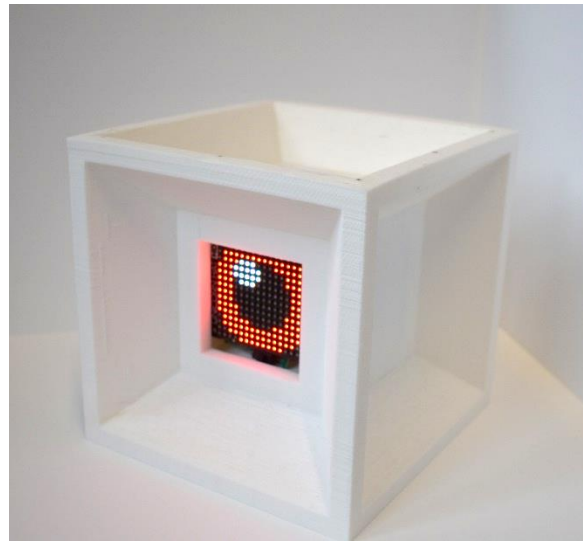
There will be no need to use the admin or client side, as An item either exists or it does not and I will not be using the tools to interact with the chatbot interface, more just using it to provide a better searching method for the AI I am making.

The alterations to the main SHEP code must not break the SHEP code in other processes, for example the uni-bot. It should be a library tool for machine learning. By splitting up the functions, this should be possible.

### User interface

The user interface is very much hardware based, with the LED matrix eye and a microphone. This will need to be presented in a nice way. Using the old Hypercube design from V 0.0.1, I 3D printed a cube to hold the Raspberry Pi Zero, attached to the UNICORN HAT. Between was a PCB HAT which had a

microphone soldered to it, and a battery pack. All of this screws onto the lid of the hypercube so that it can easily fit into the cube.



The eye blinks every few seconds to give a more active feel.

The hardware has a microphone wired in so that it can use speech recognition. I will use the Google speech to text API. I will need to also install a speaker method in the robot and some sort of speech to text library (which should be achievable as I have done it before).

I will need to incorporate some sort of reset if inactive method, so the responses do not get confused. Perhaps after about a minute of inactivity the conversation will reset. This is something I can set up in the user interface side.

For audio output I used the espeak library. I also used the Python speech recognition library and google recognizer. I will need to research into use of these software's in a legal sense.

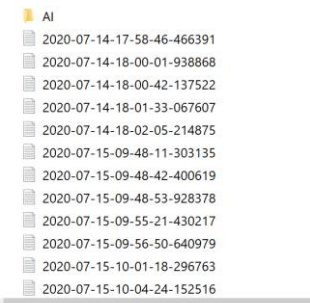
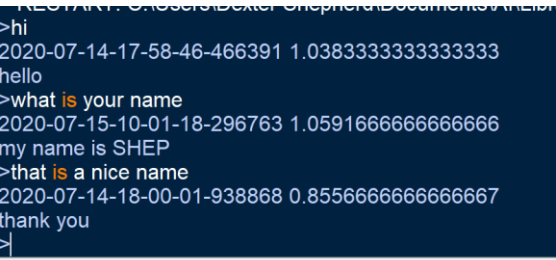
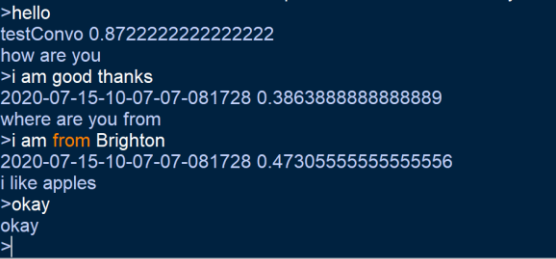
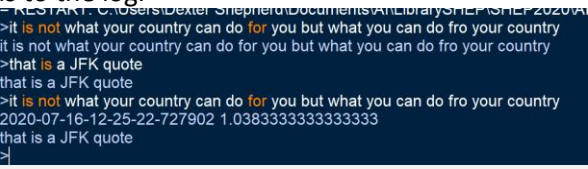
I have used both of them before so got my old code for it, but it was not working.

I installed all the libraries and had a bit of trouble compiling the microphone, but got there in the end.



# Testing

## AI code

Test No	Test	Outcome
1	Creates a log of the conversations	 <p>The files are uniquely named by using the date and time. Each file contains the conversation that I had with the bot.</p>
2	Finds the right database logs to point to	
3	The code finds relevant responses to the input.	 <p>The responses are based off previous responses. It finds the most likely response.</p>
4	The code finds the most relevant output out of a series of outputs saved.	 <p>Each datafile name is outputted in this window, with the probability of this folder combined with the phrase being correct. As you can see, the file changes based on how relevant the conversation is to the log.</p>
5	The code is adding new information and ways to respond to this information all the time.	 <p>The information is added and remembered. Of course, I will make the learning system more complicated.</p>

```

>there are many birds out today
thats nice
>where are you
0.4685714285714285
i am from the uk
>nice
cool
>what is going on
thanks
>no problem
there are many birds out today
>there are indeed
where are you
>

```

Above shows a later snippet of development where it will ask you the question more naturally.

6

## Main SHEP

Test No	Test	Outcome
1	Functions I expected way in previous version	Yes

## User interface

Test No	Test	Outcome
1	Displays an eye	Yes
2	The eye blinks every so often	Yes
3	The AI shows it is listening	Yes, with a green LED
4	Does the AI take in your voice and convert to text	Yes

## Deployment of the system

### Libraries needed

SHEP library 2020, and all the sub libraries:

- Nltk
- Json
- Sys
- Itertools
- Re
- Difflib
- threading

Datetime to uniquely name files. As many of these libraries are inbuilt to Python they are free to use and distribute.

## Environment

### Future versions

This system is very much based on imitating conversation and not taking it further. Our ability to communicate is arguably like this in a fundamental level, but what we can do is form new information from past experiences. This is a concept I would like to research and implement in the next version so that the system actually grasps information and uses it to its best ability.

Another thing I would like to add is the ability to find where questions lead to the same responses. This can then work out what words and structures mean the same thing. This learning of language will potentially help find information in the future.

### Legal

Appending the SHEP2020 License written for the chat bot service, I will be making changes as some of the libraries are not used.

Something which was not a library was espeak, which I used for voice recognition,

I used the Google speech to text library.

GNU General Public License v3 (GPLv3) - For the audio output

BSD License (BSD) - The speech recognition library cannot be used in products without written permission from the author.

## References

[1] University chat bot project proposal, 2020

[2] Microsoft AI Nazi, [https://en.wikipedia.org/wiki/Tay\\_\(bot\)](https://en.wikipedia.org/wiki/Tay_(bot)) (20/06/2020)

[3] Classical conditioning, [https://en.wikipedia.org/wiki/Classical\\_conditioning](https://en.wikipedia.org/wiki/Classical_conditioning) (14/07/2020)

[4] Cleverbot, <https://www.cleverbot.com/>