

ElizaJS - Long Term Memory for Eliza with Rudimentary Context Recognition

Francis Dippnall

Manchester Metropolitan University, United Kingdom

April 17, 2020

Abstract

Contents

0	Disclaimer	3
0.1	Ethical Approval	3
1	Introduction	4
1.1	Course Specific Learning Outcomes	4
1.2	Definitions	4
1.3	Background	5
1.4	Report Outline	5
2	Literature Review	6
2.1	Introduction	6
2.2	Chatterbots	6
2.3	Known Issues	6
2.4	Modern Usage	7
2.5	Methodologies	7
2.5.1	Pattern Matching	7
2.5.2	AIML	7
2.5.3	SQL	7
2.5.4	Markov Chain	7
2.6	Conclusions	7
3	Design	8
3.1	Requirements Analysis	8
3.1.1	Priority 1 - Chatterbot	8
3.1.2	Priority 2 - Long-Term Memory	8
3.1.3	Priority 3 - Believability	8
3.1.4	Priority 4 - Usability	8
3.1.5	Priority 5 - Code Quality and Maintainability	8
3.2	Technology	9
3.2.1	JavaScript	9
3.2.2	Node	9
3.2.3	Express	9
3.2.4	Socket.io	9
3.2.5	MySQL	9
3.3	Memory	10
3.3.1	Introduction	10
3.3.2	Lexical Context	10
3.3.3	Fact Extraction	11
3.3.4	Memory Querying	12
3.4	Data Flow	13
3.4.1	Data Flow Diagram (DFD)	13
3.4.2	Two Types of Data	13
3.4.3	Socket.IO Plan	14
3.5	Database Design	15
3.5.1	Memory Subsystem	15
3.5.2	Evaluation Subsystem	15

4	Implementation	16
4.1	Data Tier	16
4.1.1	Credentials	16
4.1.2	Creating a Connection	16
4.2	Server	17
4.2.1	Handler Deferral	17
4.2.2	Server-Side Validation	17
4.3	Client Script (runeliza)	17
4.3.1	Interfacing with Socket.IO	17
4.3.2	Loading an Instance	17
4.4	Eliza Core	18
4.4.1	talk()	18
4.4.2	readyState	18
4.4.3	_get_response()	18
4.4.4	_run_macros()	19
4.4.5	_case()	19
4.4.6	_output()	20
4.5	Pronoun Handling	20
4.5.1	Pronoun Classes	21
4.5.2	Pronoun Sub-types	21
4.5.3	_estimate_pronoun_class()	21
4.5.4	Pronoun Substitution	21
4.6	Legacy Core	22
4.7	Evaluation System	22
4.7.1	Evaluation Data	22
4.7.2	Likert Evaluation	22
5	Testing	23
6	Evaluation	24
6.1	Priority 1 - Chatterbot	24
6.2	Priority 2 - Long-Term Memory	24
6.3	Priority 3 - Believability	24
6.4	Priority 4 - Usability	24
6.5	Priority 5 - Code Quality and Maintainability	24
6.5.1	+ Class Structure	24
6.5.2	+ Asynchronous Operation	24
6.6	Personal Evaluation	24
7	Conclusion	25
8	References	26
9	Appendix - Feasibility Study	27

List of Figures

1	Data Flow Diagram of the ElizaJS system.	13
2	Entity Relationship Diagram. Evaluation is in blue, Memory in green. . .	15
3	Credentials.json example.	16
4	definition.net, line 320. Pronoun class.	21
5	Evaluation form.	22

0 Disclaimer

No part of this project has been submitted in support of an application for any other degree or qualification at this or any other institute of learning. Apart from those parts of this project containing citations to the work of others, this project is my own unaided work.

0.1 Ethical Approval

This project was given ethical approval before testing began.

***EthOS* reference: 14404**

1 Introduction

This report documents the development of an Eliza (Weizenbaum, 1966) chatterbot reimplementation featuring a long-term memory system; which includes a basic context recognition and management system to improve believability. Throughout this report the MMU Harvard referencing system is used for citations. See section 8 for a complete reference list.

1.1 Course Specific Learning Outcomes

1. Apply skills of critical analysis to real-world situations within a defined range of contexts.
2. Demonstrate a high degree of professionalism characterised by initiative, creativity, motivation and self-management.
3. Express ideas effectively and communicate information appropriately and accurately using a range of media, including ICT.
4. Articulate an awareness of the social and community contexts within their disciplinary field.

1.2 Definitions

1. conversational agent (CA)

A computer program that attempts to converse with a human user using natural language via text or speech messaging.

2. chatterbot

CA that tries to fool the user into believing that it is in fact a human.

3. chatbot

CA for other use such as consumer service. It does not attempt to fool the user.

4. lexical string/statement (LS)

A string of words written in the chosen language (English).

5. lexical operator

An operator string in the chosen language (English). Examples: *is*, *has*, *will be*

6. lexical term

The words before or after a given lexical operator in a string.

7. ES6

ECMA Script 6. The leading JavaScript standard.

1.3 Background

The concept of emulating human conversation has been documented as far back as the 17th century, when Descartes posed the idea that it may be possible to produce an automaton that emulates human speech and appearance so well that it becomes indistinguishable to a person. The revolutionary paper Computing Machinery and Intelligence (Turing, 1950), while not the first mention of “learning machines” in the information age, was one of the driving factors for future public interest in this subset of computing. Later the term “conversational agent” was used to describe a variety of different types of natural language human-computer interface (O’Shea et al, 2011) which included the chatterbot, an agent which attempts to hold a conversation with a human using natural language.

The subject of this project, *Eliza* (Weizenbaum, 1966), was one of the first attempts at creating a chatterbot designed to challenge the modern Turing test (Mauldin, 1994). While it may seem primitive, *Eliza* showed that creating a convincing series of responses was possible by simply performing some processing on the input of the human and weakly conjugating to switch person (e.g. “I have brown hair” might become “you have brown hair”). *Eliza* does have a rudimentary implementation of short-term memory: the human inputs are manipulated into facts that are stored in program memory in an array, and can subsequently be accessed to allow *Eliza* to repeat information it is given as statements - however, any information stored about the conversation is deleted when that conversation ends. Thus, it can be said that *Eliza* does not implement any long-term memory solution.

1.4 Report Outline

The report contains the following sections:

1. Introduction

This introductory section.

2. Literature Review

An analysis of relevant literature regarding artificial intelligence and chatterbots.

3. Design

Documentation of the design process undergone for the project.

4. Implementation

Description of the product itself, including any relevant features and processes.

5. Testing

Testing methodologies and results.

6. Evaluation

Evaluation of the results of the project, including a product and personal evaluation.

7. Conclusion

Final thoughts on the project, summarizing results and evaluation.

8. References

List of academic references cited in the report.

9. Appendix

Report appendix, including the Feasibility Study.

2 Literature Review

This section comprises of a description of the relevant work in the field of artificial intelligence and chatterbots.

2.1 Introduction

Turing described in his historic paper *Computing Machinery and Intelligence* (Turing, 1950) a method by which to approach an answer to the ultimate question of "can machines think?" called the Imitation Game; an altered version of which is still used now as the definitive objective for some fields of artificial intelligence: the Loebner prize for example, is one of the most widely known assessment of computer programs' ability to trick humans into believing they are a human (Mauldin, 1994). This prize has been criticised for corrupting the original idea of Turing into an exploitable process that rewards cheap trickery (Shieber, 1994). It can be said however, that while these methods do not prove that the machine can think, they do show that they can mimic a human's messages sufficiently.

2.2 Chatterbots

Eliza (Weizenbaum, 1966) was one of the earliest responses to Imitation Game. Weizenbaum showed that a program could be written to play the game convincingly by using linguistic tricks, most notably including reflecting questions in a Rogerian style - a method which could create convincing messages without any memory or world state. Parry (Colby, 1971) improved on Eliza by tracking emotional state throughout the conversation using several dimensions such as trust or anger. Parry was also generally paranoid hence the name, a trait which justified the disconnected logic it was using. Colby later showed with Random-Parry that the tracking of emotional state is more believable than a version that randomly decided on an emotion for each message. Many more variations on Eliza and Parry have been developed in time, and are available online (Laven, 1996). On his page, Laven defines the difference between the terms "chatbot" and "chatterbot" - the former is widely used for customer service and telemarketing, whereas the latter is designed to fool users into sounding human-like; although these terms are often used interchangeably. Additionally, the term "bot" can refer to either form and is often used for brevity.

2.3 Known Issues

As noted, Eliza has several limiting factors which have led to its deprecation. Firstly, it was written in the bespoke language "Slip" which has seen no modern usage and is therefore unadaptable. This has caused many reimplementations to be created, such as *elizabeth.js* (Landsteiner, 2005), in JavaScript. These programs are written as artefacts, and use the same tricks as the original without adding any newer features. Secondly, the criticisms of Eliza noted that lacking a sense of state leads to non-sequitural statements and disconnected conversation (Mauldin, 1994). Modern chatterbots will often make use of the internet to gain a layer of understanding, such as *Alice* (Wallace, 1995). *ViDi*, the virtual dietitian (Lokman et al, 2009), is an example of a chatbot that remembers past conversations to help diabetic patients.

2.4 Modern Usage

Chatbots see wide use in automated "tech support" systems and business interfaces (Deryugina, 2010). Deryugina describes how the *Anna* bot is the virtual representative of IKEA, and how it interacts with the website by loading relevant pages to help the user. While goals of chatterbot design do not align exactly with that of commercial chatbots such as *Anna*, the methods that produce human-like messaging are useful in creating a more natural discourse for the customer.

2.5 Methodologies

Conversational agents use linguistic tricks to sound less like a machine and more like a human. The following techniques are a subset of those detailed in *Survey on Chatbot Design Techniques in Speech Conversation Systems* (Abdul-Kader et al, 2015).

2.5.1 Pattern Matching

Perhaps the simplest method is to match the input text to a pattern and respond accordingly. This is the main trick for many chatterbots and while it is effective for basic sentences such as greetings, it will fail if no pattern is found which means wider "catch-all" patterns must also be implemented which can lead to disconnected phrasing as if the agent has ignored the user.

2.5.2 AIML

Created by Richard Wallace, the creator of *Alice*, Artificial Intelligence Markup Language (AIML) is the most widely used language for chatbots today. AIML provides a flexible system for writing the responses of a CA without any code, often seen as an advantage allowing more people interested in the field to create bots. AIML defines sets of patterns and "templates" to formulate a response, which are then processed by a "core" that performs the major decision making.

2.5.3 SQL

Abdul-Kader et al notes that modern chatbots make use of a relational database to store conversation history to be searched for keywords later. They argue it "gives continuity and accuracy" as it allows past comments to be accessed like with human conversation.

2.5.4 Markov Chain

A more advanced technique is the use of Markov Chains to produce a response that has a higher probability of correctness (Bradeško et al, 2012). In theory this leads to fewer non-sequiturs and thus greater believability.

2.6 Conclusions

The techniques looked at in the review will be helpful in producing a believable agent, especially when combined with an effective knowledge base using a relational data structure, however the Markov Chain method deviates from the project scope. Techniques involving SQL are especially relevant and will be developed in this project.

3 Design

This section outlines the design of the ElizaJS project. It covers all aspects of the pre-implementation design - minor production details will be discussed in Implementation (section 4).

3.1 Requirements Analysis

The Feasibility Study provided the majority of requirements for the project, however it is useful to list the up to date version here for reference. This section lists the macro-requirements as "priorities".

3.1.1 Priority 1 - Chatterbot

The project is, at heart, a chatterbot. As such, the first priority is to develop a working chatterbot - that is, a text-based interface for a user to converse with a core, and the core itself. The core needs to be able to interface with the view such that it can receive the user's messages and can respond with its own.

3.1.2 Priority 2 - Long-Term Memory

The resulting agent must be able to have a long-term memory for this project to be successful. This will be defined as follows:

1. Ability to detect and transpose statements using standard English.
2. Mechanism to store parsed statements in a database in a context-irrelevant manner (so they can be understood in any context)
3. Ability to retrieve statements when relevant from the database and output them in a correct context-dependant manner.

3.1.3 Priority 3 - Believability

Eliza was originally meant as a response to the Turing proposition, and as such it tried to imitate human conversational patterns to trick users to believing the agent they were speaking to was a person. For this reason it is important that this long-term memory implementation attempts to be believable and so it should use good grammar and have a wide variety of response keywords to imitate a human agent.

3.1.4 Priority 4 - Usability

The application should be easy to operate by the end user in the long and short term. The view should have a focus on ease of use and provide textual help if needed.

3.1.5 Priority 5 - Code Quality and Maintainability

The server and client side code should be readable, concise, maintainable, well-formatted, and scalable so long as doing so does not conflict with any of the previous priorities. The code should make good use of new ES6 principles.

3.2 Technology

This section describes and explains the main technologies to be used in the project. The is one subsection for each technology.

3.2.1 JavaScript

JavaScript (JS) is an imperative programming language that bridges object-oriented languages such as Python with more functional approaches. The key draw of JS in this project is it can be used for full stack development (see below). Its extensive inbuilt library of methods for strings is also a positive, which the client and server files will make ample use of. On the client side specifically, JS provides asynchronous behaviour management tools like `await` and `setTimeout` which are useful to defer operations until their required time; similar to the `yield` instruction in other languages.

3.2.2 Node

Node.js is a JavaScript runtime environment that executes JavaScript without a browser. Node provides a platform for full-stack development in JavaScript, and is used in many development stacks such as MEAN. This project uses Node instead of Java because Node utilises the asynchronous nature of JS which means requests in Node are run on one "thread" but time slices are allocated mathematically at process level, instead of Java in which each request has its own thread. Additionally, Node servers are lightweight and the Node Package Manager (NPM) handles dependencies automatically which speeds up development.

3.2.3 Express

Express is a Node package that, among other things, provides an abstract interface to a standard HTTP server. Express simplifies the implementation of normal HTTP methods (GET, POST ...), however more importantly it supplies the `static` method which is used to expose a single folder for the public web files, similar to Apache.

3.2.4 Socket.io

The `socket.io` Node package allows for reliable low-level communication between the server and client without the need for a page refresh which improves the user experience. `socket.io` supports transfer of many data structures which improves code flexibility to future changes. The `socket.io` package can be attached directly to an HTTP server and served through Express which means less time is spent setting it up than other web socket solutions.

3.2.5 MySQL

MySQL is a relational database modelling system that allows the Node server to interface with a remote database. Said database can be structured to have field-specific constraints to reduce the chance of data redundancy and inconsistency. MySQL is a good choice for this project as opposed to other options such as MongoDB because the MMU Mudfoot system allows use of a remote MySQL server without third-party hosting.

3.3 Memory

This section outlines the design of the long-term memory mechanism to be employed in the ElizaJS system.

3.3.1 Introduction

The original Eliza chatterbot had a rudimentary implementation of memory, which worked by saving facts as defined in the definition file as binary or ternary lexical statements in an array in memory. We can define "memory" in this context as some form of storage of lexical statements of this type which have some meaning in the real-world, that can later be used to influence future responses. This implementation can be referred to as "short-term" memory since the information is not retained after the termination of the program. To successfully implement a long-term memory solution, there must be a sense of fact persistence across multiple conversations.

3.3.2 Lexical Context

The ElizaJS system will have a basic context handler which should allow the user to use common predefined pronouns ("it", "he" etc.) to refer to the most recent item in the current lexical context. The core will track two contextual variables: Active Context Pronoun (ACP), and Active Context Noun (ACN) which will allow it to both translate user messages to a base lexical form (see 0.1.3), and use pronouns in its own responses (see 0.1.0). For this, the core must be able to recognise when:

- **A context is created or replaced**

This is done when a new noun is introduced by the user. The current ACP and ACN are overwritten by the new information.

- **A context is destroyed**

A rarer form of context switching where the context is dereferenced without replacing it with a new one. To avoid over-complication, This shall be assumed when the core receives a message that contains neither a valid fact containing a noun nor a valid pronoun. Examples of context-destroying messages include "Enjoy!" and "Well then."

3.3.3 Fact Extraction

A "fact" will be used to mean an individual item of memory consisting of up to two lexical terms and an operator from a set of pre-defined lexical operators taken from the training corpus. To allow the core to remember facts they must be extracted from each user message. The process for which is explained in detail below.

1. **Search for operators**

The first step is to find the highest-priority lexical operator in the user string. Starting with the message string, iterate over each word in the operator list in descending priority order and search for that word in the message. If no operators are found, the user string does not contain any facts and Eliza should revert to a Null Response. When an operator is found, split the user string at that point so that you have the characters before the operator, the operator itself, and the characters after the operator.

2. **Trim lexical terms**

For each string, remove the portion of the string from the punctuation mark closest to the operator to the extreme end (the end furthest away from the operator) of the string.

3. **Apply context**

Pronouns in the lexical terms must be replaced with the valid context information. If no context information can be found, the core must abandon the fact to avoid compromising memory integrity. For each lexical term, scan for a pronoun from the pre-defined list. First and second-person pronouns may be published unchanged, but if a third-person pronoun is found, check if the pronoun matches the current ACP. If there is no match, abort the fact creation process. Otherwise, replace the pronoun in the lexical term with the current ACN. Repeat until all pronouns in the string have been replaced.

After the completion of the steps above you are left with the pre-operator lexical term (`term1`), the operator itself, and the post-operator term (`term2`) which can be stored in the database along with the username of the user providing the fact (`negotiator`) and the date (`fact date`). Note that due to the subject-verb-object¹ nature of English, the `term1` LS can be assumed to be the *subject* and `term2` the *object*.

¹<https://en.wikipedia.org/wiki/Subject%E2%80%93verb%E2%80%93object>

3.3.4 Memory Querying

The final step of the memory architecture of the system will be the querying process. This will allow the core to respond to user's statements with corresponding factual data. If no memory entry can be found for the queried item, the core can respond with a question, or create a new entry if a fact is supplied by the user message. The following steps are performed after the completion of the Fact Extraction process.

- **Query the fact base**

If the user string contains a valid fact, query the datastore for facts whose *subject* contains or is contained in the extracted user fact. Otherwise:

- **Perform user-requested query**

If the user string contains a match for one phrase in the *keyphrase* list (see Fig 0.), take the text following the keyword in the user string as the *subject* and query the datastore for any fact with an exact match for that subject. Otherwise:

- **Use null response**

No information can be found about the user string, so respond with a null response.

3.4 Data Flow

Before the specific data structure can be designed, it is necessary to plan the data flow of the system. I decided to go with a 3-tier architecture as it allows off-loading of processing to the client which speeds up the server. However, this means the Eliza core is situated on the client side which could lead to some security vulnerabilities that should be considered in the implementation phase. The DFD below outlines the macrostructure of the system.

3.4.1 Data Flow Diagram (DFD)

The general idea is that the user submits their User Data (username) and this is stored in the knowledge base.² Then the user can send messages to the core, similar to the original Eliza, however the core will scan each message for keywords and query the knowledge base for data on that keyword.

The server will act as the middle-man between the client and the knowledge base for security and verification purposes, and will pass the results of these queries back to the core which can then formulate a response based off of the information.

This process repeats until the user quits the chat session, at which point the client will submit a log to the server which will be stored in another data structure (the Evaluation subsystem, see below).

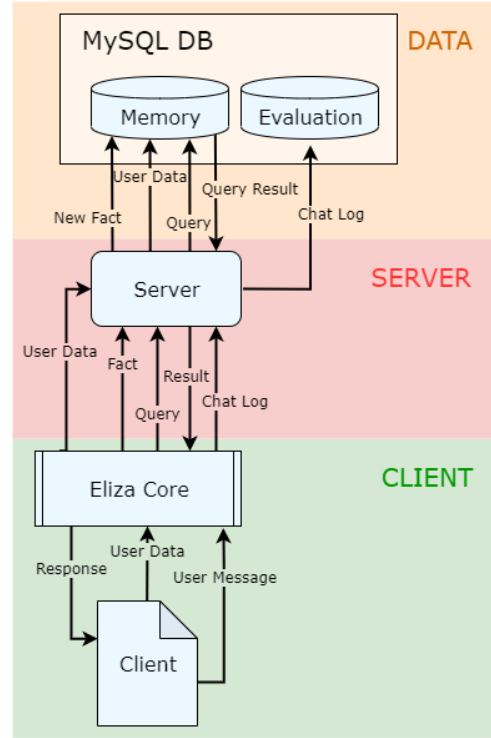


Figure 1: Data Flow Diagram of the ElizaJS system.

3.4.2 Two Types of Data

Looking at the DFD it is clear that there are two distinct types of data being transmitted: conversation logs, and facts to be stored in the long-term memory. These two aspects should be completely separate in the data structure as the core should be able to access memory data at will but raw conversation data is solely for evaluation. As such, it is imperative that the database contains two isolated subsystems: one for storing evaluation data which will be referred to as *Evaluation* and another for the knowledge base named *Memory*. The next section will cover the design of each of these subsystems and explain their purpose in more detail.

²The username is also stored on the Evaluation subsystem.

3.4.3 Socket.IO Plan

The `socket.io` system uses a message-based communication structure. The server can send messages to any client and the clients can send messages to the server. It is standard practice to plan the communications in table format so that an understanding about the use of `socket.io` in this project can be reached.

Sender	Message Name	Send When?	Data Sent	On Receipt
Client	sign on	Client submits the user data form.	username: string password: string	Server validates and verifies the client data, and responds with a sign on result message.
Server	sign on result	Server has processed the client data.	success: boolean new user: boolean? reason: string?	Client displays the chat container if success is true, else display the reason for failure.
Client	query factbase	Client has extracted a term and wishes to query the factbase.	query: string	Server performs the query and responds with a query result message.
Server	query result	Server has completed the query.	success: boolean results: object? reason: string?	Client uses the query result to compile an Eliza response.
Client	new fact	Client has extracted a fact from the user and wants to push it to the factbase.	fact: object	Server validates the new fact and inserts it into the database.
Client	conversation log	Client has finished the conversation.	username: string, log: object, bot type: string, blind: boolean	Server serialises and stores the conversation log in the datastore with additional data.
Server	log received	Server has logged the conversation to the datastore.	success: boolean reason: string?	Client displays the evaluation page.
Client	user evaluation	Client submits the evaluation form.	results: object	Server stores the evaluation in the database.
Server	evaluation received	Server has completed storing evaluation in datastore.	success: boolean reason: string?	Client displays a thank you message.

3.5 Database Design

The Entity Relationship Diagram (ERD) of the MySQL database used in the project is shown in the figure below. It shows the two disconnected subsystems and four tables. Each table name is prefixed by "Eliza" to distinguish these tables from the other ones in the database used.

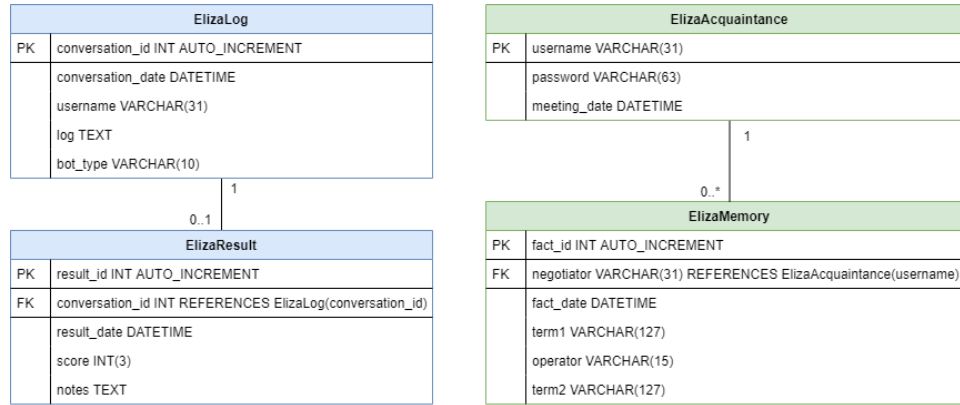


Figure 2: Entity Relationship Diagram. Evaluation is in blue, Memory in green.

3.5.1 Memory Subsystem

The Memory system is the long-term memory structure of ElizaJS. It contains two tables:

- **ElizaAcquaintance** - A store of the users that Eliza has spoken with or is currently speaking to. Allows persistent conversations by providing a password.
- **ElizaMemory** - the actual fact-base for the core. Each entry contains a reference to the user that provided the fact, the date-time when it was obtained; in addition to the fact itself, which is stored as two lexical terms and an operator.

There is more information on the Memory process in section 3.2.

3.5.2 Evaluation Subsystem

The Evaluation subsystem is used to collect information for analysis purposes. This data is not relevant to the chatterbot core and is therefore only accessible via an admin view. This system uses two tables:

- **ElizaLog** - stores full text logs of every conversation on record paired with the submission data and username. The `log` field is delimited by a double-slash ("`//`") meaning this string must be removed if present in any message. The `bot_type` field shows which version of Eliza this conversation is with.
- **ElizaResult** - stores user evaluation data. As an optional addition to the conversation log, users can provide feedback notes and a score which will be useful in the Evaluation phase.

4 Implementation

This section outlines the implementation of the ElizaJS project including any specific production decisions made during development. Note that line numbers refer to the code on submission (21 Feb 2020).

4.1 Data Tier

The data tier of ElizaJS is composed of a MySQL database (see 3.5) and some server-side code which allows interfacing between the server and data tiers.

4.1.1 Credentials

To provide an interface for the database, I followed a Credentials system which I have been using on different projects for several years now. This system involves creating a `credentials.json` file (see figure 4) in the root folder which contains all of the connection details for the MySQL server.

```
1 {  
2     "HOST": "localhost",  
3     "PORT": 3306,  
4     "USER": "root",  
5     "DATABASE": "elizaajs",  
6     "PASS": ""  
7 }
```

Figure 3: Credentials.json example.

Importantly, this file is included in the `.gitignore` file so Git will not post this to any branches of the repository. This protects users from having their connection credentials leaked. Next, `server.js` wraps these connection details into a DB object for later use. This can be seen at line 29.

4.1.2 Creating a Connection

The server file now needed a simple method to create a connection to the database using the given information. This is done via the `createConnection` (line 98), a process which simplifies the code and improves readability and maintainability.

4.2 Server

The `server.js` file comprises the server tier which is responsible for routing facts between the client and data tiers. As previously noted, this is done via `socket.io` messages. Generally, the socket plan (see 3.4.3) was followed correctly.

4.2.1 Handler Deferral

This service has a deferred system for receiving messages which only enables the majority of handlers after the user sends a valid `sign on` message. This prevents a client from submitting and querying data until they are actually in a conversation, which should improve performance and security. The function responsible for attaching the deferred handlers is called `addElizaServices` and can be seen at line 207.

4.2.2 Server-Side Validation

An important role of the server tier is validation. `server.js` has several validation functions: `validUsername` at line 68, and `validateFact` on line 108 for example. These methods provide a final check ensuring only entirely valid data is stored in the database. As an additional measure, the server uses `mysql.escape`³ on all user data values which guards against SQL injection⁴ attacks and otherwise invalid characters such as non-escaped apostrophes.

4.3 Client Script (runeliza)

The `runeliza.js` file handles the view. It interfaces with the page DOM to instantiate an Eliza instance and binds it to the global variable `currentElizaInstance` after the user successfully signs in to the service.

4.3.1 Interfacing with Socket.IO

The `runeliza.js` script file handles all socket communication to and from the server on the client side. This means that no references to `socket` are required in the core file in line with separation of concerns. To allow the core to make requests and receive the server's responses, a `methods` object is supplied on instantiation containing socket-specific communication functions (see line 176).

4.3.2 Loading an Instance

The script contains a `loadEliza` function (line 23) which asynchronously instantiates the selected Eliza core. This function supports the passing of `options` which can alter some behaviour of the core, `methods` to expose to the core, and a `callback` to execute after instantiation is complete.

³<https://github.com/mysqljs/mysql#escaping-query-values>

⁴https://www.w3schools.com/sql/sql_injection.asp

4.4 Eliza Core

The `ElizaBotNew.js` file handles the core Eliza Chatterbot logic. This section explains the the major functions of the core, including the `definition.net` file. Line numbers may refer to either the former or latter file based on context.

4.4.1 `talk()`

`talk()` (line 127) is the main external method of `ElizeBotNew`. It takes a single parameter `message` and performs the following operations in order:

1. Checks `readyState` to ensure that the bot is ready for input
2. Outputs the user's given message to the output container
3. Processes the bot's response using `_get_response()`
4. Formats the response for human readability using `_run_macros()`
5. Outputs the response from the bot and sets the `readyState` back to normal

Note that these steps also apply to the legacy version (see 4.5). The following subsections describe these steps in more detail.

4.4.2 `readyState`

`readyState` is a property of the core that stores its current status. There are four available states, represented by an integer value: `null`, meaning that the core is not loaded yet; `0` means the core is processing its response; `1` shows that the bot is waiting for text to be printed on the screen (when using slow text); and `2` indicates that the bot is ready for input. The `readyState` of the instance is shown by the colour of the wrapper: `null`, `0`, `1`, `2` is represented by grey, red, orange and green respectively.

4.4.3 `_get_response()`

This function (line 162) holds the chatterbot logic that results in a response from the bot. Initially this was a normal synchronous function that returned the result, however the inclusion of queries meant a callback-oriented approach is now used. The function has several priorities that are executed in descending order of importance. If a valid response is found at any time, execution of the function is stopped via `return`.

1. Explicit query requests (line 175)

The highest priority action is to search for query keywords. This is done using the definitions listed at line 444. If a keyword is found, the code extracts a lexical term from the message and queries the database on this term, then responds with the result.

2. Fact extraction (line 330)

This step follows the instructions in this paper (see 3.3.3), so will be omitted for brevity.

3. Fallback keywords (line 478)

If neither a query request nor a fact can be found in the message, look for fallback keywords. These are defined in the definition file at line 484 and represent greetings, expressions, and other technically irrelevant chatter. If one of these is found, the responses are predefined underneath the keyword set.

4. Noun messages (line 494)

The last resort checks if the entire unedited message is a valid lexical term. If so, the bot will respond with an affirmative reflection similar to the original Eliza.

5. Non-noun messages (line 506)

If none of the above steps are used, simply print a random non-noun response defined at line 198.

4.4.4 `_run_macros()`

After the response has been produced, it must be formatted such that it makes sense and has correct grammar and casing. This function performs several operations on the output message:

1. Replace macro-keywords

The function uses JS `replace` to detect any macro keywords defined under "special commands" at line 39 and replace them with their value. This improves believability as it allows the bot to refer to the current context inline as a human would.

2. Transpose to second person

In the case where the context refers to the user, second person is used ('you', 'your').

3. Case output

The `_case()` function cases the output message using a series of rules. (See 4.4.5)

4.4.5 `_case()`

`_case()` (line 831) is a synchronous method that takes a single parameter `line`, and cases (converts each letter to uppercase or lowercase) it *correctly* based on a set of rules. The function follows the following steps:

1. Format the line

It can be assumed that in English, only the first letter of any word is ever capitalised. This assumption ignores some structures like acronyms of course, however it is fitting for this project's scope. Thus, the function begins by lowercasing all of the characters in the string, and splitting it into an array of words using the space character ' '. It then iterates over the words and checks each one to see if it needs to be capitalised.

2. Remove punctuation

Each "word" still contains punctuation marks at this point, so they are removed and saved into some variables to be reinserted later.

3. Capitalise special words

Any "special" words must now be capitalised. This mainly includes irregular capitalisation of 'I', 'I'm', and so on, but also includes some relevant brands like GitHub.

4. Capitalise special words

Any "special" words must now be capitalised. This mainly includes irregular capitalisation of 'I', 'I'm', and so on, but also includes some relevant brands like GitHub.

5. Capitalise names

Here, the `Array.binarySearchBoolean` method defined in `ElizaBot.js` is utilised to search the thousands of names for a match. If any name is found, it is capitalised.

6. Add punctuation

Punctuation is added back into the string in the same place as it was before formatting. This includes possessive suffixes.

7. Capitalise first letters

A final rule of English casing is the first letter of all sentences are capital. To achieve this, the function iterates over every character looking for sentence termination characters ('.', '?', and '!') and capitalises the next alphabetic character after that one.

4.4.6 `_output()`

`_output()` is the function used to display any message to the interface. By default it displays text in a manner similar to the MS-DOS implementation (one character at a time) but it can be configured to display text immediately.

4.5 Pronoun Handling

A large portion of this program goes to managing pronouns in speech. This section explains the mechanisms which allow fluid pronoun handling and usage. In English there are many types of words, however most of these forms are exceptionally rare and can be ignored for these purposes. Recall the program has an ability to remember context already (see 3.3.2). This system uses a function `_estimate_pronoun_class()` to assign a pronoun class from a set defined in `definitions.net` 06, to that lexical term. From then on, instances of pronouns in that class can be assumed to refer to that context.

4.5.1 Pronoun Classes

While not technically "classes" these are objects that describe the attribute of a pronoun set and list a hopefully exhaustive instance list of pronouns. For example, the third-person singular genderless class, shown in Figure 6 below, contains four pronouns. It also describes the attributes (third-person, singular, genderless).

```
320  PC third-person singular genderless
321  PR it
322  PR it
323  PR itself
324  PR its
```

Figure 4: `definition.net`, line 320. Pronoun class.

4.5.2 Pronoun Sub-types

The order of the pronouns in the list does matter. As described on line 299, the format is *nominative*, *objective*, *objective-self*, *possessive*. This explains why the first two are the same in the previous example, as "it" is both nominative and objective⁵.

4.5.3 `_estimate_pronoun_class()`

This method, found on line 592 in `ElizaBotNew.js` sounds complex, but in reality is quite simple. It takes 2 parameters: `term`, the lexical term to estimate the pronoun class of, and `negotiator`, an optional parameter that specifies the negotiator of the term. The function then estimates the plurality of the term based on whether it ends in 's', and the gender of the term according on some basic rules. If the term contains a word that has a gender then that is assumed to be the gender for the term. Such words include the user or negotiator's username, in which case the gender is their gender; words that have gender assigned to them by the language, defined on line 359 of `definitions.net`, like "mother" (female) and "brother" (male), and also personal names, in which case the gender is that of the name.

4.5.4 Pronoun Substitution

The implementation of the pronoun class system allows to replace instances of terms by a suitable pronoun and vice-versa: to replace instances of pronouns with the term they refer to. The former is done via macros (see 4.4.4) and the latter is done in the fact extraction process (see 3.3.3). Importantly, the pronouns possessiveness must be taken into account during this process, for example "Bob" should transpose to "he" whereas "Bob's" would instead be "his". That is why the pronoun sub-types are ordered (see 4.7.2), as it makes choosing a possessive pronoun easy (its the last one in the class).

⁵https://www.grammar-monster.com/glossary/personal_pronouns.htm

4.6 Legacy Core

`ElizaBotOld.js` contains a JS reimplementaion of the original Eliza, modelled to use similar string logic which uses the exact same definition file as the original. Developing this legacy core was useful for two reasons: first, it provided a starting point to extend with long-term memory functionality to reduce repetition; and second, the legacy core can be used in testing and evaluation to compare the believability (see 3.1.3) of both new and old chatterbots. The one difference between the original and this legacy core is that the legacy core has no memory of any kind, whereas the original has a volatile short-term memory.

4.7 Evaluation System

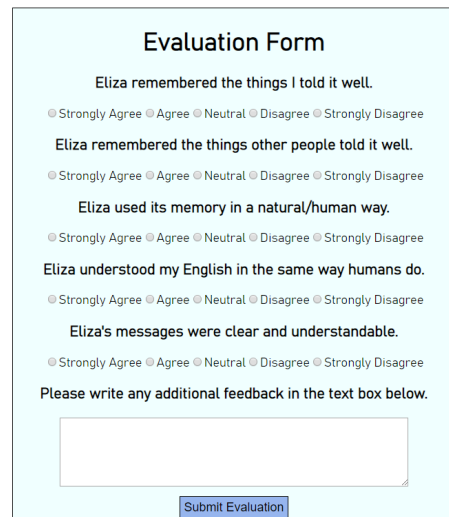
The inclusion of a legacy core in the final product necessitated an evaluation system. After a conversation finishes, the user is directed to a thank you page with an evaluation form. (see figure 5).

4.7.1 Evaluation Data

When the form is submitted, an evaluation is stored in the database linked to the logged conversation. Also included in the data is the `bot_type`, either old or new, which allows an evaluation to be performed on the believability of the new system compared to the old one.

4.7.2 Likert Evaluation

The evaluation system uses Likert scales (Likert, 1932) for each question which should provide an accurate representation of the core’s properties given a large enough sample group. Users can also optionally enter additional feedback using the text field.



The screenshot shows a web form titled "Evaluation Form" with a light blue background. It contains five evaluation questions, each followed by a Likert scale with five radio buttons labeled "Strongly Agree", "Agree", "Neutral", "Disagree", and "Strongly Disagree". The questions are: "Eliza remembered the things I told it well.", "Eliza remembered the things other people told it well.", "Eliza used its memory in a natural/human way.", "Eliza understood my English in the same way humans do.", and "Eliza's messages were clear and understandable." Below these questions is a text box for additional feedback, preceded by the instruction "Please write any additional feedback in the text box below." At the bottom right of the form is a blue button labeled "Submit Evaluation".

Figure 5: Evaluation form.

5 Testing

Testing was performed on a set of 7 participants with no knowledge of the field.

6 Evaluation

This section consists of a system evaluation, including qualitative considerations of the five priorities outlined in the requirements analysis (see 3.1), followed by a personal evaluation. Positive points are preceded with a '+' whereas negatives have a '-'.

6.1 Priority 1 - Chatterbot

6.2 Priority 2 - Long-Term Memory

6.3 Priority 3 - Believability

6.4 Priority 4 - Usability

6.5 Priority 5 - Code Quality and Maintainability

Overall, I believe this project has a high degree of code quality and maintainability, with the use of good programming practices such as ES6 class notation and modularisation.

6.5.1 + Class Structure

The core file uses the ES6 class notation to define the `ElizaBotNew` class. It also privatises methods and attributes - private members of the class are preceded with an underscore, and the class includes several *getters* and *setters* (line 18) that control the private methods, similar to an encapsulated object in Java. The `constructor` function handles asynchronously loading the `definition.net` file and the names files (see 6.6.6) using `XMLHttpRequest` objects. These ideas are in line with common practices in the industry; they improve maintainability in the long term.

6.5.2 + Asynchronous Operation

The `talk()` method is set to `async` which allows code to run in parallel with it, including the browser's interface. A side-effect of this is that the `_output` function is also `async` and needs to be awaited as to avoid visual bugs.

6.6 Personal Evaluation

7 Conclusion

8 References

- Weizenbaum, J. (1966). *ELIZA—a computer program for the study of natural language communication between man and machine*. Communications of the ACM, 9(1), pp.36-45.
- Turing, A.M. (1950). *Computing Machinery and Intelligence*. Mind 49: 433-460.
- O’Shea, J., Bandar, Z. and Crockett, K. (2011). *Systems engineering and conversational agents*. In Intelligence-Based Systems Engineering (pp. 201-232). Springer, Berlin, Heidelberg.
- Mauldin, M.L. (1994). *Chatterbots, tinymuds, and the turing test: Entering the loebner prize competition*. In AAAI (Vol. 94, pp. 16-21).
- Shieber, S.M., (1994) *Lessons from a restricted Turing test*. arXiv preprint cmp-lg/9404002.
- Colby, K.M. (1975). *Artificial paranoia: a computer simulation of paranoid process*. Pergamon Press.
- Epstein, R. (1992). *The quest for the thinking computer*. AI magazine, 13(2), pp.81-81.
- Loebner, H.G. and Shieber, S.M., (1994) *Response-Lessons from a Restricted Turing Test*. Communications of the ACM-Association for Computing Machinery-CACM, 37(6), pp.79-88.
- Colby, K.M., Weber, S. and Hilf, F.D. (1971). *Artificial paranoia*. Artificial Intelligence, 2(1), pp.1-25.
- Deryugina, O.V. (2010). *Chatterbots*. Scientific and Technical Information Processing, 37(2), pp.143-147.
- Landsteiner, N. (2005). *Eliza (elizabot.js)*, Mass:Werk. [Online] [Accessed on 14th April 2020] <https://www.masswerk.at/elizabot/>
- Laven, S. (1996). *The Simon Laven Page*. [Online] [Accessed on 14th April 2020] <http://www.simonlaven.com/>
- Wallace, R. (1995). *A.L.I.C.E. (Artificial Linguistic Internet Computer Entity)*. [Online] [Accessed on 15th April 2020] Accessible via *Pandorabots*: <https://www.pandorabots.com/pandora/talk?botid=b8d616e35e36e881>
- Lokman, A.S. and Zain, J.M. (2009). *An architectural design of Virtual Dietitian (ViDi) for diabetic patients*. In 2009 2nd IEEE International Conference on Computer Science and Information Technology (pp. 408-411). IEEE.
- Abdul-Kader, S.A. and Woods, J.C. (2015). *Survey on chatbot design techniques in speech conversation systems*. International Journal of Advanced Computer Science and Applications, 6(7).
- Bradeško, L. and Mladenović, D. (2012). *A survey of chatbot systems through a loebner prize competition*. In Proceedings of Slovenian Language Technologies Society Eighth Conference of Language Technologies (pp. 34-37).
- Likert, R., (1932). *A technique for the measurement of attitudes*. Archives of psychology.

9 Appendix - Feasibility Study

Feasibility Study

Project Title: Eliza Plus – Long Term Memory

Student: Francis Dippnall – 17003003

Supervisor: James O'Shea

Contents

1. Course-Specific Learning Outcomes	2
2. Background	2
3. Project Aim.....	3
4. Objectives.....	3
4.1. Research.....	3
4.1.1. Literature Study	3
4.1.2. Existing Solutions	3
4.1.3. Potential Problems.....	3
4.2. Design.....	3
4.2.1. Design Documentation	3
4.2.2. System Design	4
4.2.3. Problem-Solution Records	4
4.2.4. Methodology.....	4
4.3. Delivery	4
4.4. Report	4
4.5. Evaluation	4
5. Domain.....	4
6. Required Resources	5
7. Schedule.....	5
8. Ethical Approval	6
9. References	11

1. Course-Specific Learning Outcomes

1. Apply skills of critical analysis to real-world situations within a defined range of contexts.
2. Demonstrate a high degree of professionalism characterised by initiative, creativity, motivation and self-management.
3. Express ideas effectively and communicate information appropriately and accurately using a range of media, including ICT.
4. Articulate an awareness of the social and community contexts within their disciplinary field.

2. Background

The concept of emulating human conversation has been documented as far back as the 17th century, when Descartes posed the idea that it may be possible to produce an automaton that emulates human speech and appearance so well that it becomes indistinguishable to a person. The revolutionary paper *Computing Machinery and Intelligence* (Turing 1950), while not the first mention of “learning machines” in the information age, was one of the driving factors for future public interest in this subset of computing. Later the term “conversational agent” was used to describe a variety of different types of natural language human-computer interface (O’Shea et al, 2011) which included the chatterbot, an agent which attempts to hold a conversation with a human using natural language.

The subject of this project, Eliza (Weizenbaum 1966), was one of the first attempts at creating a chatterbot designed to challenge the modern Turing test (Mauldin, 1994). While it may seem primitive, Eliza showed that creating a convincing series of responses was possible by simply performing some processing on the input of the human and weakly conjugating to switch person (e.g. “I have brown hair” might become “you have brown hair”). Eliza does have a rudimentary implementation of *short-term* memory: the human inputs are manipulated into facts that are stored in program memory in an array, and can subsequently be accessed to allow Eliza to rudimentarily repeat information it is given as statements - however, any information stored about the conversation is deleted when that conversation ends. Thus, it can be said that Eliza does not implement any long-term memory solution.

3. Project Aim

The aim of this project is the successful implementation of a long-term memory mechanism applied to a representation of the Eliza conversational core. This mechanism should allow the system to “remember” facts collected about the environment and the user. The data collected from this process will have to persist between conversations - it will have to be stored in some form of long-term data structure.

The core should then be able to pull these facts from the long-term memory and use them in later conversations to bring Eliza closer to emulating a believable human agent.

4. Objectives

This project has a set of abstract objectives, each with their own sub-objectives, as listed below.

4.1. Research

4.1.1. Literature Study

The research into the background of the project should include references to prominent AI researchers from multiple subsets of the field from 1950-2019. It should consider a wide range of potentially conflicting opinions from different reputable sources.

4.1.2. Existing Solutions

As part of the design process, this project will require continual comparison to existing solutions, especially Eliza, but also competitors and/or improvements to the original core such as Parry (Colby, 1975). Evaluations should be made in the context of the year the solutions were produced in.

4.1.3. Potential Problems

Research into common issues with other conversational agent systems must include searching for the flaws of those systems, (and what others have done to solve them) so that they can be addressed in the design phase.

4.2. Design

The design of solutions to non-trivial problems like this must be consistent and systematic. As a minimum it should include the following:

4.2.1. Design Documentation

Diagrams and graphs describing the design process, including Data Flow Diagrams, Entity Relationship Diagrams, and pseudo-code which can simplify complex sections of the design.

4.2.2. System Design

The system will be designed in accordance with a mainstream design pattern such as the Model-View-Controller model.

4.2.3. Problem-Solution Records

Textual recordings of problems encountered during the design and implementation stages of development, with detailed descriptions of both the problem itself and the steps taken to resolve those problems.

4.2.4. Methodology

Work on the project will follow an Agile design philosophy and follow a standard objective-oriented methodology.

4.3. Delivery

The deliverable objectives for this project are shown below.

- A prototype application
- A final product
- A project report
- A product showcase

4.4. Report

After the submission of the final version of the program, a complete report on each component of design, implementation, testing and evaluation will be submitted.

4.5. Evaluation

The product will be evaluated by a set of participants within the selected domain. Logs will be taken recording the conversations they had with the agent and a simple survey will be taken to determine how effective the memory was, in terms of human-like behaviour.

5. Domain

The 'domain' of this project refers to the type of chatterbot personality the product will have - it is required to build a functional agent in a short timespan to limit the scope of available conversations and allow a fuller implementation of the conversation space.

The selected personality for this project is a video games expert, who will talk to users about games they have played recently and remember user's ratings for games using the long-term memory mechanism.

6. Required Resources

- Development computers
- Programming environment (IDE)
- Web hosting

7. Schedule

The schedule for the project, as shown in Fig. 1, is in Gantt chart form.

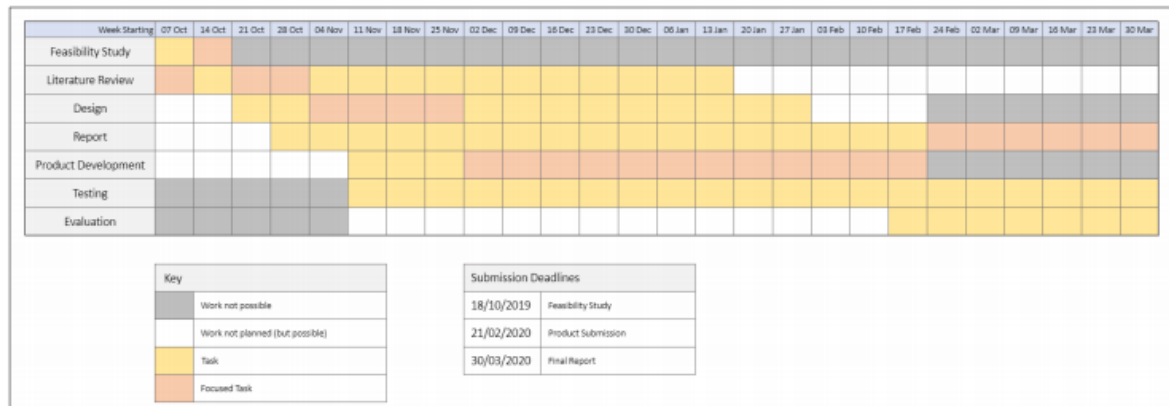


Figure 1 - Project Schedule

8. Ethical Approval

START HERE - Basic Information

This form must be completed for all student projects.

Before you proceed

Some activities inherently involve increased risks or approval by external regulatory bodies, so a proportional ethics review is not recommended and a full ethical review may be required.

These may include:

- i. Approval from an external regulatory body (including, but not limited to: NHS (HRA), HMPPS etc.);
- ii. Misleading participants;
- iii. Research without the participants' consent;
- iv. Clinical procedures with participants;
- v. The ingestion or administration of any substance to participants by any means of delivery;
- vi. The use of novel techniques, even where apparently non-invasive, whose safety may be open to question;
- vii. The use of ionising radiation or exposure to radioactive materials;
- viii. Engaging in, witnessing, or monitoring criminal activity;
- ix. Engaging with, or accessing terrorism related materials;
- x. A requirement for security clearance to access participants, data or materials;
- xi. Physical or psychological risk to the participants or researcher;
- xii. The project activity takes place in a country outside of the UK for which there is currently an active travel warning issued by the authorities (see info button);
- xiii. Animals, animal tissue, new or existing human tissue, or biological toxins and agents.

If any of these activities are fundamental to your project, please contact your supervisor to determine if a full application is required.

This form must be completed for each research project which you undertake at the University. It must be approved by your supervisor (where relevant) PRIOR to the start of any data collection.

In completing this form, please consult the University's [ACADEMIC ETHICAL FRAMEWORK](#) for ethical research.

A1 Please confirm that you will abide by the University's Academic Ethical Framework in relation to this project.

- ☒ Yes
☐ No

A2 Are you submitting this application as a learning experience, for a unit which already has ethical approval? (please confirm with your supervisor)

- ☐ Yes
☒ No

A3 Student details

Title	First Name	Surname
<input type="text"/>	<input type="text" value="Francis William"/>	<input type="text" value="Dippnail"/>
Email <input type="text" value="francis.w.dippnail@stu.mmu.ac.uk"/>		

A3.1 Manchester Metropolitan University ID number

A4 Supervisor

Title	First Name	Surname
<input type="text" value="Dr."/>	<input type="text" value="James"/>	<input type="text" value="O'Shea"/>
Faculty <input type="text" value="Science and Engineering"/>		
Telephone <input type="text" value="0161 247 1546"/>		
Email <input type="text" value="j.d.oshea@mmu.ac.uk"/>		

A5 Which Faculty is responsible for the project?

A6 Course title

A7 Project title

A8 What is the proposed start date of your project?

A9 When do you expect to complete your project?

A10 Please describe the overall aims of your project (3-4 sentences). Research questions should also be included here.

Implementation of long-term memory mechanism for conversational agent "eliza". Research into similar instances of conversational agent data storage, and their solution to the long-term problem in the context of the modern Turing test.

A11 Please describe the research activity

The project mostly consists of development of a solution to a programming problem.
After the development of the program has completed, a web service will allow participants to interact with the conversational agent. This system will log the conversations and they will be stored in text format on the server. Other information collected is composed of participant's names and other non-private data such as their age.
Participants who visit the site will have to give their consent via an online form prior to accessing the program and prior to any of their data being stored.
The logs will be analysed to find out how successful the implementation of the long-term memory is.
Information will be deleted at most two weeks after the project end date, and participants who wish to withdraw from the study can do so by submitting a form in the page which will automatically remove all of their data from the server.

A12 Please provide details of the participants you intend to involve (please include information relating to the number involved and their demographics; the inclusion and exclusion criteria)

Participants will be included without any exclusion criteria (provided they give informed consent) when they use the service. The service will be advertised in the selected project domain via email and social media. Predicted participant numbers are between 5-30. Vulnerable Persons as per the UK's Office of the Public Guardian will not be used in this research.

A13 Please upload your project protocol

Type	Document Name	File Name	Version Date	Version	Size
Project Protocol	Feasibility Study	Feasibility Study.pdf	16/10/2019	1.1	304.3 KB

Project Activity

B1 Are there any Health and Safety risks to the researcher and/or participants?

- ☐ Yes
☒ No

B2 Please select any of the following which apply to your project

- ☐ Aspects involving human participants (including, but not limited to interviews, questionnaires, images, artefacts and social media data)
☐ Aspects that the researcher or participants could find embarrassing or emotionally upsetting
☐ Aspects that include culturally sensitive issues (e.g. age, gender, ethnicity etc.)
☐ Aspects involving vulnerable groups (e.g. prisoners, pregnant women, children, elderly or disabled people, people experiencing mental health problems, victims of crime etc.), but does not require special approval from external bodies (NHS, security clearance, etc.)
☐ Project activity which will take place in a country outside of the UK
☒ None of the above

B2.4 Is this project being undertaken as part of a larger research study for which a Manchester Metropolitan application for ethical approval has already been granted or submitted?

- ☐ Yes
☐ No

Data

F1 How and where will data and documentation be stored?

Logs of participant's conversations on the web server will be kept in text format. These will be deleted after the project is completed as previously stated in question A11.

F2 Will you be collecting personal data or sensitive personal data as part of this project?

- ☐ Yes
☐ No

Insurance

F3 Does your project involve:

- ☐ Pregnant persons as participants with procedures other than blood samples being taken from them? (see info button)
☐ Children aged five or under with procedures other than blood samples being taken from them? (see info button)
☐ Activities being undertaken by the lead investigator or any other member of the study team in a country outside of the UK as indicated in the info button? If 'Yes', please refer to the 'Travel Insurance' guidance on the info button
☐ Working with Hepatitis, Human T-Cell Lymphotropic Virus Type iii (HTLV iii), or Lymphadenopathy Associated Virus (LAV) or the mutants, derivatives or variations thereof or Acquired Immune Deficiency Syndrome (AIDS) or any syndrome or condition of a similar kind?
☐ Working with Transmissible Spongiform Encephalopathy (TSE), Creutzfeldt-Jakob Disease (CJD), variant Creutzfeldt-Jakob Disease (vCJD) or new variant Creutzfeldt-Jakob Disease (nvCJD)?
☐ Working in hazardous areas or high risk countries? (see info button)
☐ Working with hazardous substances outside of a controlled environment?
☐ Working with persons with a history of violence, substance abuse or a criminal record?
☒ None of the above

Additional Information

G1 Do you have any additional information or comments which have not been covered in this form?

- ☐ Yes
☐ No

G2 Do you have any additional documentation which you want to upload?

- ☐ Yes
☐ No

Signatures

H1 I confirm that all information in this application is accurate and true. I will not start this project until I have received Ethical Approval.

- ☐ I confirm
☐ I do not confirm

H2 Please notify your supervisor that this application is complete and ready to be submitted by clicking "Request" below. Do not begin your project until you have received confirmation from your supervisor - it is your responsibility to ensure that they do this.

Signed: This form was signed by Nicholas Costen (N.Costen@mmu.ac.uk) on 18/10/2019 4:53 PM

H3 By signing this application you are confirming that all details included in the form have been completed accurately and truthfully.

Signed: This form was signed by Francis William Dippnall (francis.w.dippnall@stu.mmu.ac.uk) on 18/10/2019 4:07 PM

9. References

- Turing, A.M., 1950. Computing Machinery and Intelligence. *Mind* 49: 433-460.
(archive available at <http://cogprints.org/499/1/turing.html>)
- Weizenbaum, J., 1966. ELIZA---a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1), pp.36-45.
- Colby, K.M., 1975. *Artificial paranoia: a computer simulation of paranoid process*. Pergamon Press.
- Mauldin, M.L., 1994, August. Chatterbots, tinymuds, and the turing test: Entering the loebner prize competition. In *AAAI* (Vol. 94, pp. 16-21).
(available at <http://new.aaai.org/Papers/AAAI/1994/AAAI94-003.pdf>)
- Epstein, R., 1992. The quest for the thinking computer. *AI magazine*, 13(2), pp.81-81.
(available at <https://www.aaai.org/ojs/index.php/aimagazine/article/download/993/911>)
- Shieber, S.M., 1994. Lessons from a restricted Turing test. *arXiv preprint cmp-lg/9404002*.
- O'Shea, J., Bandar, Z. and Crockett, K., 2011. Systems engineering and conversational agents. In *Intelligence-Based Systems Engineering* (pp. 201-232). Springer, Berlin, Heidelberg.