An AI Chatbot To Help Autistic Children Develop Their Social Skills

by

George Pearce

CS310 Third Year Project

Supervisor: Ian Saunders

Department of Computer Science
University of Warwick

Abstract

One of the things autistic people find most challenging is social interaction. More specifically, being able to understand and employ the social skills common amongst the majority of the population, or the "Predominant Neurotype" (PNT) [1, 2]. Naturally, the best time for an autistic person to develop these skills is in the early stages of their development, and there have been numerous attempts to use technology to help autistic children develop their social skills, with varying levels of both individual and broader success [3, 4, 5]. This project aims to strike a balance between effectiveness and accessibility not present in the majority of these previous solutions. This will be achieved by using emerging artificial intelligence (AI) technologies to develop a chatbot application that will be able to interact with an autistic child, identify areas within the PNT range of social skills that might be lacking, and tailor subsequent interactions to help the child improve in these areas. While there is still a significant amount of future work that could be done to improve the chatbot, as is likely to always be the case in a field as complex as autism education, the project does succeed in achieving its initial goal and delivers a system capable of making a difference to people's lives.

Keywords: Autism, Autism Intervention, Social Skills, AI, Natural Language Processing, Chatbot

Contents

Introduction **4**
1.1 Motivation 4
1.2 Background 6
Research **8**
2.1 Expert Advice 8
2.2 Existing Interventions 10
2.3 Social Skills Teaching 12
2.4 Ethical Considerations 14
Design **15**
3.1 Social Interaction Breakdown 15
3.1.1 Approach 15
3.1.2 Assumptions 16
3.1.3 Social Skills 16
3.1.4 Teaching Methodology 19
3.2 Requirements 22
3.3 Project Success 25
Project Management **26**
4.1 Project Management Methodology 26
4.2 Organisation 27
4.3 Timeline 30
Implementation **34**
5.1 Software & Tools 34
5.2 Web Application 37
5.3 Initiating the Rasa Chatbot 40
5.4 Connecting the Web Application and the Chatbot 45
5.5 Improving the Chatbot 49
5.6 Implementing Evaluation Points 53
5.7 Second Iteration 67
Evaluation **71**
6.1 Testing 71
6.2 Requirements Evaluation 75
6.3 Project Management Evaluation 77
Conclusion **79**

7.1 Practical Limitations 79
7.2 Future Work & Applications 80

References **81**

Chapter 1

Introduction

1.1 Motivation

Social interaction is something that the general population are likely to take for granted as something that they are not only able to do, but might enjoy on a daily basis. Without social interaction, people are more likely to suffer from issues such as loneliness and depression. Interaction normally takes place through a mutually understood set of rules or guidelines, the understanding and implementation of which amounts to possessing a typical set of 'social skills'. The general population, by definition, have brains which can be classified as being of the "Predominant Neurotype" (PNT) [1]. Therefore, these common social skills understood by the general population can be described as PNT social skills [2].

The National Autistic Society defines autism as a "spectrum condition" that "affects how people perceive the world and interact with others" [6]. This means that people with autism have brains that differ from the PNT, and therefore, generally, do not inherently understand PNT social skills. This lack of PNT social skills can drastically hinder an autistic person's ability to interact with others. This can lead to immense anxiety and stress when undertaking a social interaction, or even in the days or weeks leading up to a social event. Loneliness can then be a big issue as the autistic person might struggle to get any value from social interactions if they don't know how to communicate with the other person, or may decide to avoid social interactions altogether to prevent feelings of anxiety or stress [2].

In fact, many of the issues facing autistic people in society could, at least in part, be attributed to the barriers that exist in communication. For example, the National Autistic Society found in their UK 'Autism Employment Gap Report' in 2016 [7] that "only 32% [of autistic adults surveyed] are in some kind of paid work (full and part-time combined), compared to 47% of disabled people and 80% of non-disabled people" and that "over three quarters (77%) who are unemployed say they want to work." Whilst there are undoubtedly many factors influencing autistic people's drastically disproportionate hardship in finding employment, it is certain that barriers in communication play a part. Unfortunately, the vast chasm between the experiences of autistic people and the general population is not limited to the realm of employment opportunities. People with autism are more likely to suffer from a range of mental health problems, such as anxiety disorders and depression. According to the National Autistic Society's 2012 report 'I Exist: The message from adults with autism in England' [8], "75% [of autistic adults surveyed] do not have any friends or find it hard or very had to make friends" and "72% would like to spend more time in the company of other people".

There is certainly a huge amount to be done to help people with autism enjoy the same quality of life as the general population, but it is clear that helping to bridge the gap between social skill sets is a key issue. To do this we need to help both those with "autistic social skills" [2] better understand PNT social skills, and those with PNT social skills better understand those with autistic social skills. The latter is a huge task, seemingly requiring a societally pervasive approach to better understand people with autism, almost certainly stemming from government policy. The former, however, is a much more approachable task. The first reason for this is that only around 1% of the UK population is estimated to be autistic. This is still a huge number of people, but is significantly less than the 99% without autism. The second reason is that PNT social skills are generally better understood, at least in many cases.

When attempting to devise a solution to many, more specific, kinds of problems in this area, an obvious preference for focussing on autistic children emerges. It is simpler to attempt to help autistic children understand the PNT social skills of their peers than an autistic adult because the range of PNT social skills in children is simpler. It is also preferable to bring about this understanding in an autistic person when they are younger, so that they may develop alongside their peers and enjoy the benefits throughout both their child and adult years.

1.2 Background

There have been numerous attempts to use technology to address this challenge of helping autistic children develop their understanding of PNT social skills, ranging from simple game-based apps [9] to walking, talking robots [4, 10]. Mobile and tablet apps tend to focus on one or two more specific areas of social interaction. Common focuses include emotion recognition and production, and promoting collaboration with others in the context of playing games [11]. Unfortunately, whilst some of these games have seen genuine success, many fall short. Some suffer from issues with design and some with the core implementation that mean autistic children struggle to receive any benefit from them. Others might achieve improvement within the context of the game, but fail to transition that success through to the real world. In addition, a drawback present in the vast majority of apps is the focussed nature of them. Even if they enable autistic children to improve in one area, they will still be lacking in many others.

The use of robots has been found to do a much better job of overcoming many of these issues. Robots have been used to help autistic children develop a much wider range of PNT social skills, including emotion recognition, emotion production, eye contact [12], and many more. Significantly, therapy with robots has been found to improve the autistic child's social skills when interacting with other humans [12], not just with the robot, and can be useful for children on both the high and low [13] functioning ends of the spectrum. Whilst there may be many nuances that need to be better understood in order to develop robots that can help autistic children understand PNT social skills even better and transition that understanding into interactions with other children more effectively, it is clear that the use of robots is tremendously promising. The biggest drawback currently with this approach is severely restricted accessibility. Robots designed to help children with autism are currently either far too expensive for the average family with an autistic child or simply not available outside of the lab.

It has also been found that technologies not originally intended for this specific purpose have had a positive impact in this area. In fact, it was one such example, described in a New York Times article [14], that initially inspired this project. In the article, a mother describes the relationship between her autistic 13 year-old son and Siri, Apple's 'virtual assistant'. The child is able to interact with Siri in a way that he is not able to with other people and gain confidence and a greater understanding of some PNT social skills. This is an example of the potential of conversational agents. Alongside the previous work done in the area, particularly that which utilises robots, this suggests that a conversational 'chatbot' developed specifically to help autistic children improve their understanding of PNT social skills could be effective. Such a chatbot could have the potential to bridge the gap between the two key approaches described previously to deliver a solution that can be both genuinely effective and incredibly accessible.

Chapter 2

Research

2.1 Expert Advice

In order to ensure the approach towards educating autistic children's PNT social skills was appropriate, several experts in the field of autism research were contacted. They were asked if they would be kind enough to offer their advice on the areas of social interaction that autistic children struggle with relative to the PNT and which 'game'-type approaches might be particularly effective for addressing these areas. The responses received are summarised in the table below, broken down into formalised pieces of advice so that they can be referred back to later on:

Dr Louise Denne, Senior Research Fellow, Centre for Educational Development, Appraisal and Research (CEDAR), Faculty of Social Sciences, University of Warwick	
LD1	"Typical child development shows that skills are by and large learned sequentially. The skills associated for example with 'theory of mind' become established between the ages of 18 months and 5 years from the relatively simple concept of perspective taking through to the more complex understanding of false belief."
LD2	"My suggestion is that you chose the skills that you wish to focus on from an established 'curriculum' of social skills teaching"
LD3	Examples including the 'Early Start Denver Model for young people with autism (Rogers and Dawson, 2010)', 'The Assessment of Basic Language

	and Learning Skills (Partington, 2010)', and 'Verbal Behaviour Milestones Assessment and Placement Programme (Sundberg 2008)'
LD4	"Have a look at teaching principles that have been tried and tested when teaching concepts to children with autism. Principles like breaking targets into small steps so that complex skills are taught gradually, respecting the 'typical' development sequence, developing scenarios that are as naturalistic as possible so that they can more easily be generalised, building in reinforcement, providing many opportunities for repetition and limiting the opportunities for making mistakes (i.e. early on providing prompts that make it easier to spot the right answer)"
Richard P. Hastings PhD, Deputy Director, Professor, and Cerebra Chair of Family Research, CEDAR, Faculty of Social Sciences, University of Warwick	
RH1	"The context for something like a chatbot would ideally be to augment an existing intervention (as opposed to standing alone)"
RH2	"You could select a target behaviour or two from an existing intervention and design something that helps a child practice those skills in the context of an ongoing intervention for them"
RH3	"The general idea might be then to use the chatbot to potentially improve outcomes within the context of an already evidence-based programme"
Kylie M. Gray PhD, Professor, CEDAR, Faculty of Social Sciences, University of Warwick	

KG1	Reiterated Richard Hastings' suggestion regarding how to approach developing some kind of intervention
KG2	Suggested four research papers that could be relevant
KG3	"It will also be important to think about who the intervention is for? What age range? What developmental age - i.e. people with/without co-occurring learning disabilities? People with communication impairments or people with autism who are verbally fluent?"

Table 2.1

2.2 Existing Interventions

These responses sparked a significant amount of further research into the area of existing social skills interventions for children with autism. The first step was to look into existing curriculums of social skills teaching, particularly those mentioned by Dr Louise Denne. The Early Start Denver Model is "a behavioral therapy for children with autism between the ages of 12-48 months" [15]. Administered by parents and trained therapists, this intervention focuses on building language and social skills through play and building relationships. Teaching follows the PNT development sequence for young children and is delivered in a highly natural and fun way in a variety of settings. These are very key points as they enable the skills developed in sessions to be generalized relatively well and the intervention to be effective with a relatively high percentage of autistic children. The two other 'curriculums' recommended are more like skills assessment protocols than an actual programme for therapy. Whilst theoretically it could be beneficial for the chatbot to follow a formal assessment such as these when identifying the areas of social interaction in which a child might be lacking, many of the skills mentioned in these assessments are not likely to be easily measured with a chatbot. This fact along with the tremendous availability of other material on how children with autism struggle with interaction means that purchasing one of these formal assessments for use in this chatbot would not be worth the relatively high cost. Other research papers on the topic that were recommended varied in their applicability for this project, but some did give a good overview of some of the areas of social interaction that autistic children commonly struggle with, as well as some of the common methods of teaching and improving them. These insights were used to help shape the design of the chatbot and will be discussed in more depth later on in this report as part of a broader look at the areas of social interaction children with autism commonly struggle with.

The next step was to look into some of the common social skills interventions available for autistic children. There are countless research papers investigating interventions for children with autism in every way imaginable: reporting on commonly used interventions, analysing a particular intervention and the degree of scientific rigour used to develop it, proposing a new intervention, and more. One particularly helpful paper identified a number of the most commonly used interventions in the United Kingdom, and was actually co-authored by two of the researchers I spoke with: Louise Denne and Richard Hastings. Motivated by the desire to address what they described as "a distinct lack of research identifying the interventions used by parents for the support and education of children with autism in the UK" [16], the paper gives a valuable platform from which to determine where this chatbot might be best suited for use as part of an existing intervention. Among the most common interventions (with the percentage of survey respondents who are currently using or have previously used the intervention in parentheses) are Visual Schedules (78.1%), Speech and Language Therapy (75.6%), Applied Behaviour Analysis (45.0%), and PECS (42.5%). Interventions are commonly used together in various combinations to best suit the individual. Despite being the most commonly used intervention in that survey, a Visual Schedules intervention would not be an appropriate home for a chatbot as it is simply "a type of visual prompt used to help individuals predict or understand upcoming events." [17] PECS, or the Picture

Exchange Communication System, is designed to enable people with "little or no communication abilities to communicate using pictures." [18] Whilst a chatbot could be designed or adapted to fit in with this intervention at a later date, it is not in keeping with the idea for this project of a natural language conversational agent. Speech and Language Therapy and Applied Behaviour Analysis (ABA) on the other hand are much more appropriate for the use of a chatbot. Speech and Language Therapy naturally focuses on addressing "challenges with language and communication" [19], whilst ABA additionally addresses other aspects of behaviour such as "attention, focus" and more general "problem behaviours." [20] Both involve the child working with a trained therapist, with help from the parents, in a variety of settings in a way that best suits the individual and emphasising teaching strategies such as positive reinforcement. A chatbot, therefore, could fit nicely in line with either of these interventions to potentially work as a method of additional conversational practice or as a form of Alternative Augmentative Communication (AAC) for those that struggle more with typical spoken interactions. There is then the potential for the chatbot to then be used as more of a stand-alone intervention as it increases in scope and robustness.

2.3 Social Skills Teaching

There are a huge number of PNT social skills that are mentioned in the various intervention materials, 'curriculums' and in other tremendously helpful resources such as the National Autistic Society website [21]. The most common of these skills can be grouped into different categories and ordered according to where they fit on the 'typical' development sequence. At the most basic level, many children with autism struggle to interact with others and initiate conversations themselves. Children with autism can often struggle with 'Theory of Mind', a skill that is typically developed before the age of 5. Theory of Mind is the ability to "attribute mental states to oneself and others", to "think about thoughts" [22]. Other fundamental aspects of interaction that the PNT would usually take for granted that can be a struggle for an autistic child include understanding body language and recognising emotions in one's conversational counterpart. Understanding societal norms of the flow of a basic conversation can also be challenging. The standards for greeting a stranger or a friend, the expectation for questions to be asked or answered and taking turns, and the use or understanding of certain, more advanced, types of language can be a mystery. Repetition of sounds, words or phrases that they've heard but do not necessarily understand, known as Echolalia, can also be common. Whilst there are certainly many more areas of 'typical' social interaction that autistic children struggle with, those described above are an important combination of some of the most fundamental skills and also some of the most commonly challenging.

In addition to identifying the social skills themselves, many of these educational sources address common methods, techniques, and tips for teaching autistic children new skills. These include the ideas suggested to me by Dr Louise Denne in LD4:

- "Breaking targets into small steps so that complex skills are taught gradually, respecting the 'typical' development sequence"
- "Developing scenarios that are as naturalistic as possible so that they can more easily be generalised"
- "Building in reinforcement - always more successful if it's an intrinsic part of the task"
- "Providing many opportunities for repetition"
- "Limiting opportunities for making mistakes (i.e. early on providing prompts that make it easier to spot the right answer"

Other common ideas are "practicing new skills in different places to help with generalisation" and "carefully picking the time to introduce new social skills." There are also numerous ways suggested to teach the skills themselves. Some of the most common methods include 'Social Scripts', emotion recognition games, modelling and role play, and other interactive 'games'. 'Games' are a very common way to teach new skills because of the naturally fun nature of them and the opportunities for positive reinforcement, repetition and incorporating a variety of scenarios.

2.4 Ethical Considerations

While there are no direct ethical issues that will arise from the project as it's not actually going to be given to actual users to test, there are broader ethical questions that should be considered when designing the chatbot. Since the chatbot is designed to be used in the development of young children, the efficacy of the system is vital. Plans must therefore be put into place when considering using the chatbot in the real world to ensure the wellbeing of the child. Whilst considerable training data can be provided and endless end-to-end and user testing can be carried out, the nature of the chatbot as an artificial intelligence means that its behaviour can never be guaranteed. Therefore, given the importance of the efficacy of the chatbot, any real-world intervention with a child should be closely managed by a trained therapist or a parent. This will help to mitigate against any potentially negative behaviours from the chatbot or interactions generally.

Chapter 3

Design

3.1 Social Interaction Breakdown

Following all the research, the first thing to do was to outline how the chatbot would approach helping an autistic child improve their PNT social skills. By formally stating what the chatbot would achieve and how it would achieve it, the incredibly vast task of dealing with social interaction could be narrowed down enough to be achievable for this project whilst still functioning to solve the fundamental issue. This process of formalising what exactly the chatbot could be capable of in terms of social interaction resulted in the creation of a 'Social Interaction Breakdown' (SIB). The SIB is split into four areas: the general approach for the intervention, assumptions about the user, the social skills to be addressed, and the method of teaching.

3.1.1 Approach

The general approach for the intervention is closely tied to the ethical considerations described above, and is therefore quite simple but very important. The chatbot should be used to augment an existing intervention (as per RH1 & RH3) such as Speech and Language Therapy or Applied Behaviour Analysis (ABA). The chatbot could function as a form of Alternative Augmentative Communication (AAC) for those that struggle to speak to other people. It could also be used with children who are more comfortable with spoken interaction as a different partner with which to practice interacting. This would be useful for two main reasons: 1) Opportunities for practice and repetition of a newly learned skill is always useful, and 2) practicing with a conversation partner other than the therapist or parent helps to avoid the child getting too used to always talking to the same person, meaning the social skills learned can better generalise to different people and situations. Whilst the chatbot itself will provide positive reinforcement where possible and appropriate (in line with LD4), interactions between the child and the chatbot should also be reviewed by the therapist or parent to assess how the child did and provide appropriate reinforcement. It should also be noted that the chatbot will target a mixture of more basic and more advanced skills. These skills should be considered by the therapist or parent when deciding whether to use the chatbot to augment an intervention.

3.1.2 Assumptions

The assumptions made about the user in the design of the chatbot are also important considerations when deciding whether to use it or not (KG3). The nature of the chatbot as it will be designed demands that the user have an understanding of 'Theory of Mind' (as mentioned in LD1), is able to read and can type at a computer, all at least a relatively basic level. The target developmental age range for the chatbot is approximately between 6 and 12 years. It is important to distinguish this as the 'developmental' age according to the 'typical' developmental sequence rather than the biological age of the child. This age range should ensure the user will have the requisite skills to use chatbot, but is not yet too advanced to gain any benefit from it.

3.1.3 Social Skills

Based on the research done into the education of PNT social skills for children with autism, there are a huge number of 'social skills' that are commonly found to be challenging. In order to successfully develop a chatbot that can help an autistic child to develop their 'social skills', it is essential to formally lay out exactly what skills will be targeted. When perusing the literature on the subject of social skills in autism, it quickly becomes clear that some skills could quite naturally be targeted through a chatbot intervention, but others would be outside the realm of possibility to incorporate (at least in a solution without some combination of truly cutting-edge computer vision, natural language processing and robotics). For the purposes of this project the social skills that can be addressed in the chatbot can be divided into the following areas of social interaction:

1. Interaction
2. Greeting
3. Questions
4. Manners
5. Figurative Language

Within these areas are the specific social skills that can be addressed by the chatbot, which are described as 'Evaluation Points' (EPs) as they are the specific aspects of interaction that the chatbot will be evaluating. The naming convention for the EPs is such that EP2.1 describes the first EP in the second area of social interaction: 'Greeting'. The table below outlines the the evaluation points to be considered in this project with corresponding MSCW values to indicate the importance of each EP to the success of the project:

Evaluation Points

1. Interaction	
Name	**MSCW**
EP1.1 Does the user interact with the chatbot	MUST
EP1.2 Does the user initiate conversation	MUST
EP1.3 Does the user interact with the chatbot by typing	COULD
EP1.4 Does the user interact with the chatbot by speaking aloud	COULD
EP1.5 Does the user interact a similar amount during ‘games’ compared with general conversation	COULD
EP1.6 Does the user interact with similar sentiment during ‘games’ compared with general conversation	WON’T

EP1.7 Is the user able to identify different emotions	SHOULD
EP1.8 Does the user exhibit echolalia in the form of stimming during general conversation or ‘games’	WON’T
EP1.9 Does the user understand ‘turn-taking’	SHOULD
2. Greeting	
Name	**MSCW**
EP2.1 Does the user say ‘hello’, or some acceptable alternative, at the beginning of the conversation	MUST
EP2.2 Does the user introduce themselves properly by giving their name in their first interaction with the chatbot	MUST
EP2.3 Does the user use the chatbot’s name when greeting the chatbot	COULD
EP2.4 Does the user ask ‘how are you?’, or some acceptable variation, at the beginning of the conversation	SHOULD
EP2.5 Does the user answer questions like ‘how are you?’ during greetings with an appropriate level of detail	SHOULD
3. Questions	
Name	**MSCW**
EP3.1 Does the user ask questions	MUST
EP3.2 Does the user wait for a response after asking a question	SHOULD
EP3.3 Does the user answer questions	MUST
4. Manners	
Name	**MSCW**
EP4.1 Does the user say ‘please’ appropriately in conversation	COULD

EP4.2 Does the user say ‘thank you’ appropriately in conversation	COULD
5. Figurative Language	
Name	**MSCW**
EP5.1 Does the user understand similes	WON’T
EP5.2 Does the user understand metaphors	WON’T
EP5.3 Does the user understand idioms	WON’T
EP5.4 Can the user use similes	WON’T
EP5.5 Can the user use metaphor	WON’T
EP5.6 Can the user use idioms	WON’T

Table 3.1.3

3.1.4 Teaching Methodology

Based on the research described in the previous section, the methodology employed to teach these social skills can be things that are directly actionable and will therefore be more 'visible' in the finished product, and things that will be used to shape the design of the chatbot, but don't necessarily amount to a tangible feature. One of the key aspects of the former is the implementation of more focused 'games' to target specific skills alongside the general conversation functionality. This is a very common idea in existing ‘curriculums’ of social skills teaching (LD3). It is important to note that these 'games' are not traditional games in the sense that they will have a winner and loser, and so the user will not require an understanding of that concept to use the chatbot. These games are referred to as such simply because they sit outside of the primary functionality of the chatbot to provide a more targeted and fun experience where necessary. Naturally, of the most common of these games, some are more applicable to the chatbot environment than others. As with all other aspects of the project, games will be developed iteratively according to the roadmap of development that will be described in a later section. Those that could be included in this project (listed from highest to lowest priority) are as follows:

- Social Scripts
- Emotion Recognition
- 'Never Ending Story'
- Figurative Language games

The Social Scripts games will take a number of forms. In addition to a number of literal 'scripts' which will walk the user through a variety of scenarios to educate them on aspects of the PNT standard format for many conversations (greeting, questions, turn-taking, etc.), the Social Scripts games will also include a feature to suggest example phrases for different situations that they find particularly challenging. This feature will help improve the user's confidence in the relevant scenarios as they will have an arsenal of appropriate phrases to draw upon where previously they would not have known what to say. Social Scripts and Social Stories is a very broad area meaning a virtually endless variety of games could be The Social Scripts games will target the following Evaluation Points: EP1.1, EP1.2, EP1.9, EP2.1, EP2.2, EP2.3, EP2.4, EP2.5, EP3.1, EP3.2, EP3.3, EP4.1, EP4.2. Since Social Scripts can be used to target so many of the EPs, including the most fundamental EPs and those classified as 'Must' EPs, this type of game will be the first to be implemented.

The Emotion Recognition games will be just that: games specifically designed to help the user improve their ability to recognise emotion in their conversational partners. Identifying emotions in facial expressions will be the first thing trained, with the emotion conveyed in the language itself addressed second. EP1.7 will be the only evaluation point targeted by this type of game.

The 'Never Ending Story' and 'Figurative Language' games are less common in existing interventions, but suitably meet the criteria for this project of both being able to target the EPs and being possible to implement naturally in a chatbot system. The 'Never Ending Story' game would primarily address EP1.9 - 'Turn Taking'. The game works as follows: "One person starts and says one word only such as 'one' the next person says another word such as 'day', and you keep going between yourselves adding one word at a time. This requires the both of you to listen to what the other person has said and tailor your response to keep the story going. These stories can end up being very silly and fun – but they can help to develop listening skills" and an understanding of turn taking. The Figurative Language games would be used to help the child develop their understanding and ability to use figurative language. Therefore they would target EP5.1, EP5.2, EP5.3, EP5.4, EP5.5, and EP5.6. The games will test the users understanding of similes, metaphors and idioms and provide prompts, information and opportunities for practice both recognition and use themselves where understanding is lacking.

Other key teaching methodologies that will be present throughout the chatbot, both in games and in general conversation, are "Breaking targets into small steps so that complex skills are taught gradually, respecting the 'typical' development sequence", "Building in reinforcement", and "Limiting opportunities for making mistakes (i.e. early on providing prompts that make it easier to spot the right answer," as suggested in LD4. The design of the Evaluation Points framework organises skills according to how fundamental they are and where they sit in the 'typical' development sequence. Largely, the smaller the number the more fundamental the EP, with the exception being those classified as 'Must' EPs are more fundamental. The more fundamental EPs will be chosen to be targeted first by the system. The games will also be developed with this idea of breaking things down and building skills up gradually in mind. Both general conversation and the games will utilise the idea of positive reinforcement and will give prompts and options where appropriate to help limit mistakes and encourage the correct behaviour. A couple of other important ideas won't be considered by the chatbot itself, but should be considered by the therapist or parent when deciding to use the chatbot intervention. These ideas are "practicing new skills in different places to help with generalisation" and "carefully picking the time to introduce new social skills."

3.2 Requirements

The main objective of this project is to develop a chatbot that can interact with an autistic child using some basic PNT social skills, analyse the responses to identify areas that need improvement, and tailor subsequent conversation to focus on improving these weak points. Now that these social skills have been formalised as the Evaluation Points and the teaching methodology has been decided upon, the requirements of the system to be able to achieve this task can be decided. The functional and non-functional requirements of the system are as follows:

Functional Requirements		
Requirement	**MSCW**	**Iteration**
F1: The system must accept a user message and respond with an appropriate message of its own through a web application user interface.	MUST	MVP
F1.1: The system must accept a user message as a string of text and parse it into structured data that includes the intent of the message and any useful entities the message contains such as a name or place.	MUST	MVP
F1.2: The system must use the structured data it generates to determine an appropriate natural language response	MUST	MVP
F1.3: The system must output it's response to the user through the web application user interface	MUST	MVP

F2: The system must be able to hold a very basic conversation with the user	MUST	MVP
F3: The system must learn from the conversations with the user to identify how competent they are with each of the Evaluation Points	MUST	MVP
F4: The system should tailor interactions with the user to target the Evaluation Points (Section 3.1.3) it determines the user's ability could be improved	SHOULD	2
F4.1: The system should use prompts, reinforcement and examples in general conversation to target specific Evaluation Points that need improvement	SHOULD	2
F4.2: The system should be able to play structured 'games' with the user to target specific Evaluation Points that need improvement	SHOULD	2
F4.3: The system should consider both the user's ability with each Evaluation Point and how fundamental that Evaluation Point is to determine which Evaluation Point to target first	SHOULD	2
F5: The system should be able to take speech from the user as input as an alternative to the text received in order to accomplish F1, also known as 'Speech Recognition'	SHOULD	3

F6: The system could output it's own messages as audio: also known as 'Speech Synthesis'	COULD	7
F7: The system could include a 'dashboard' in it's UI for the user's therapist or parent to view the information gathered as part of F3 regarding the user's ability with each of the Evaluation Points	COULD	8
F8: The system won't answer more general questions from the user by searching the internet and outputting the result	WON'T	9
Non-Functional Requirements		
Requirement	**MSCW**	
NF1: The system could ensure it always responds to the user in under 1 second in order to make the conversation seem more natural	COULD	
NF2: The system won't employ a Speech Synthesis solution that sounds to the user like a realistic human voice	WON'T	

Table 3.2

3.3 Project Success

The project will be considered to be a success at the most basic level if all the 'Must' requirements are satisfied. If all 'Should' requirements are satisfied then the project will be considered to be a great success. The satisfaction of the remaining requirements will not be a consideration when evaluating the success of this project, but are still important requirements for a complete, well-rounded solution to the task at hand. The effectiveness of this project as a solution to the fundamental task of using a chatbot to help improve an autistic child's social skills will be shown by the increase of the metrics used to describe the child's ability with the different social skills as part of F3 with further use of this chatbot intervention. It could also be determined by the therapist by administering one of many existing social skills assessments before and after use of this chatbot intervention, although that is certainly beyond the scope of this project.

Chapter 4

Project Management

4.1 Project Management Methodology

The system will be developed using a combination of Lean, Agile and Kanban ideas that best suits the nature of the project:

Kanban

- Work In Progress
  - Limited number of tasks - one 'In Progress' at a time
  - Better (Leaner) to finish one task than do 50% of two
  - Less waste as a result of task-switching (Lean)
- Pull-System
  - Pull highest priority tasks from backlog
- Kanban Board
  - Maximum of 4 'To-Do' items and 2 'In Progress' items (1 ideal)

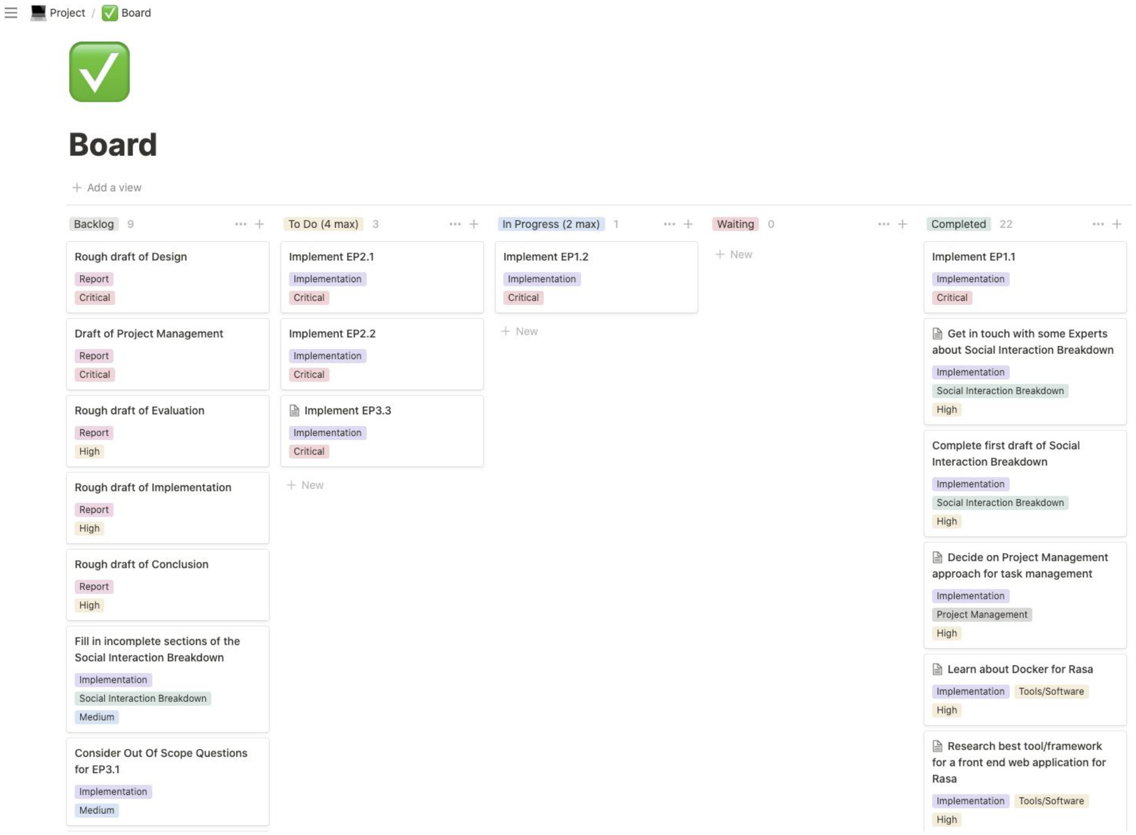
    - When fewer than 2 tasks in To-Do column, stop to pull more tasks in

Lean & Agile

- Focus on Minimum Viable Product (MVP)
    - Get feedback and then improve on that
    - Early and continuous delivery
- Don't estimate the time/effort to complete a task
    - Not adding any value
    - Not Lean

4.2 Organisation

To avoid having a slightly messy collection of documents and files in different locations, the project was managed exclusively through the highly flexible application, Notion. The first component of the Notion project workspace is the 'Board'. Tasks were managed with a Kanban board that exhibits the methodology described above:



Project / Board

Board

+ Add a view

Backlog 9

Rough draft of Design
Report
Critical

Draft of Project Management
Report
Critical

Rough draft of Evaluation
Report
High

Rough draft of Implementation
Report
High

Rough draft of Conclusion
Report
High

Fill in incomplete sections of the Social Interaction Breakdown
Implementation
Social Interaction Breakdown
Medium

Consider Out Of Scope Questions for EP3.1
Implementation
Medium

To Do (4 max) 3

Implement EP2.1
Implementation
Critical

Implement EP2.2
Implementation
Critical

Implement EP3.3
Implementation
Critical

+ New

In Progress (2 max) 1

Implement EP1.2
Implementation
Critical

+ New

Waiting 0

+ New

Completed 22

Implement EP1.1
Implementation
Critical

Get in touch with some Experts about Social Interaction Breakdown
Implementation
Social Interaction Breakdown
High

Complete first draft of Social Interaction Breakdown
Implementation
Social Interaction Breakdown
High

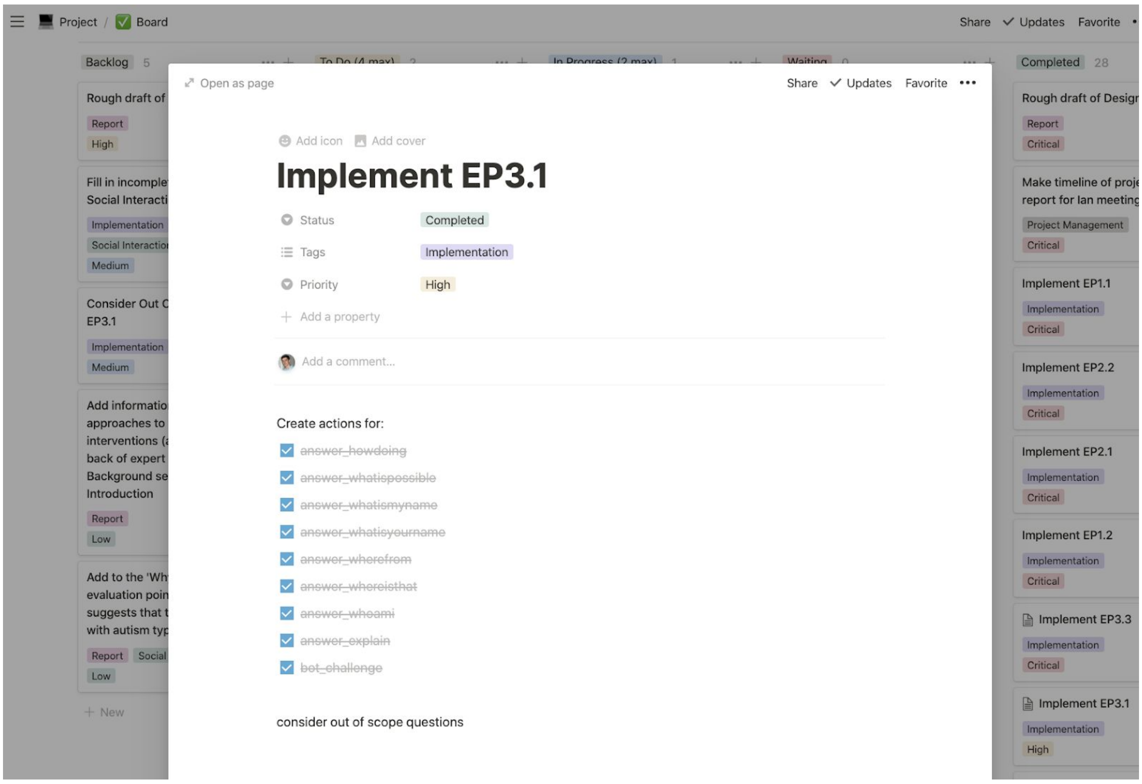
Decide on Project Management approach for task management
Implementation
Project Management
High

Learn about Docker for Rasa
Implementation Tools/Software
High

Research best tool/framework for a front end web application for Rasa
Implementation Tools/Software
High

Figure 1: The project Kanban board

Each element of the Kanban board can be expanded to include more information necessary to complete the task:



Implement EP3.1

Status Completed

Tags Implementation

Priority High

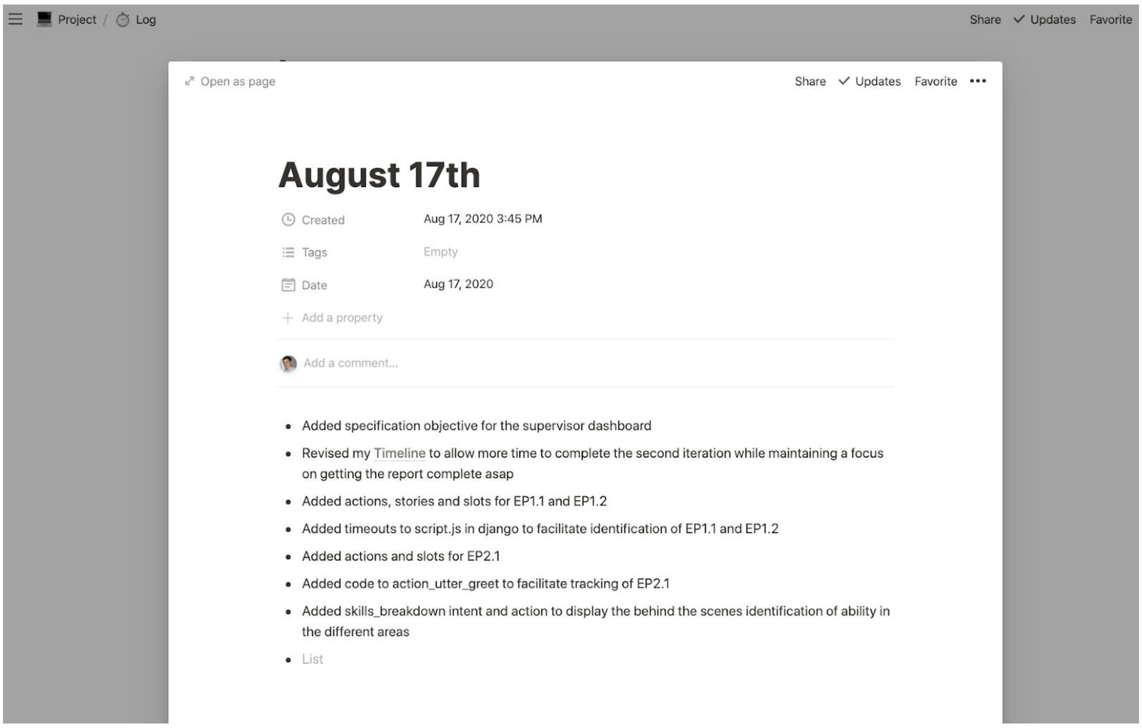
Create actions for:

- ☑ ~~answer_howdoing~~
- ☑ ~~answer_whatispossible~~
- ☑ ~~answer_whatismyname~~
- ☑ ~~answer_whatisyourname~~
- ☑ ~~answer_wherefrom~~
- ☑ ~~answer_whereisthat~~
- ☑ ~~answer_whoami~~
- ☑ ~~answer_explain~~
- ☑ ~~bot_challenge~~

consider out of scope questions

Figure 2: A card from one of the tasks on the Kanban board

The second component of the workspace is a daily log of each day's work on the project:



August 17th

Created Aug 17, 2020 3:45 PM

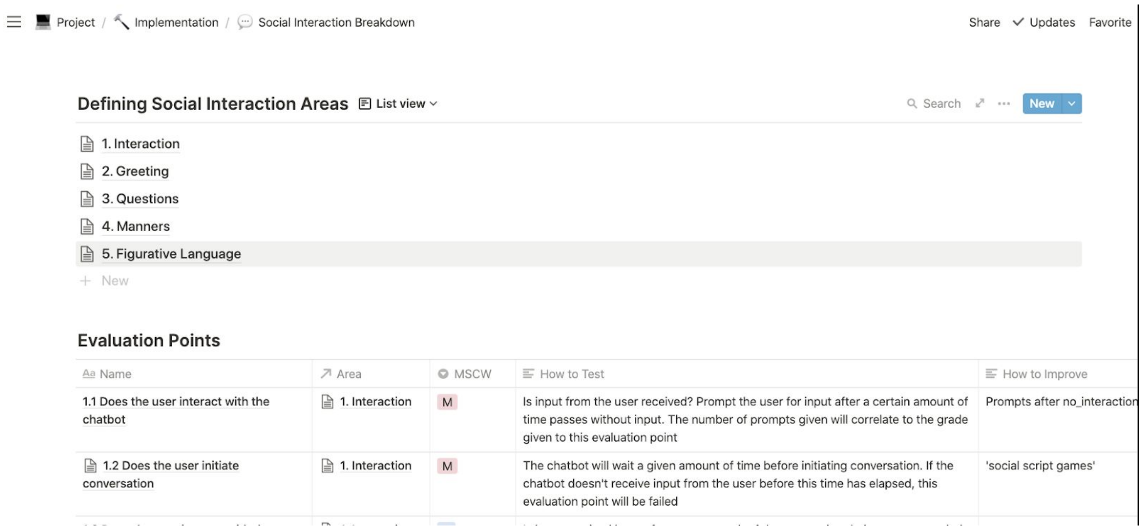
Tags Empty

Date Aug 17, 2020

- Added specification objective for the supervisor dashboard
- Revised my Timeline to allow more time to complete the second iteration while maintaining a focus on getting the report complete asap
- Added actions, stories and slots for EP1.1 and EP1.2
- Added timeouts to script.js in django to facilitate identification of EP1.1 and EP1.2
- Added actions and slots for EP2.1
- Added code to action_utter_greet to facilitate tracking of EP2.1
- Added skills_breakdown intent and action to display the behind the scenes identification of ability in the different areas
- List

Figure 3: A log entry outlining the work done that day

The workspace also includes all the information, ideas and plans for both the implementation of the project and the final report. The built in components such as lists and tables make it very easy to manage everything, and the opportunity to easily link to different parts of the workspace or pages elsewhere on the internet is tremendously helpful.



Defining Social Interaction Areas

- 1. Interaction
- 2. Greeting
- 3. Questions
- 4. Manners
- 5. Figurative Language

Evaluation Points

Name	Area	MSCW	How to Test	How to Improve
1.1 Does the user interact with the chatbot	1. Interaction	M	Is input from the user received? Prompt the user for input after a certain amount of time passes without input. The number of prompts given will correlate to the grade given to this evaluation point	Prompts after no_interaction
1.2 Does the user initiate conversation	1. Interaction	M	The chatbot will wait a given amount of time before initiating conversation. If the chatbot doesn't receive input from the user before this time has elapsed, this evaluation point will be failed	'social script games'

Figure 4: An example of some of the useful components in Notion taken from the Social Interaction Breakdown page

4.3 Timeline

In order to manage the delivery of the system in the timeframe available and according to the project management principles outlined above, it is necessary to define the Minimum Viable Product (MVP) and a roadmap of planned development thereafter. These 'iterations' of development are also outlined in table 3.2.

The MVP will contain the following requirements **F1**, **F2** and **F3** (**F3.1**, **F3.2**, and **F3.3**). The implementation of these requirements (the 'Must' requirements) will be minimally sufficient for a functioning product that could be delivered to a user to test. It is not necessary for all of the Evaluation Points defined above to be included in the MVP for it to function. The Evaluation Points that will be considered as part of F3 that will be necessary for the MVP are:

- EP1.1 Does the user interact with the chatbot
- EP1.2 Does the user initiate conversation

- EP2.1 Does the user say 'hello', or some acceptable variation, at the beginning of of the conversation
- EP2.2 Does the user introduce themselves properly by giving their name in their first interaction with the chatbot
- EP3.1 Does the user ask questions
- EP3.3 Does the user answer questions

The MVP will not include Tailoring (F4) for the included EPS, only Identification (F3). Once F4 is implemented, all EPs implemented will include both Identification and Tailoring.

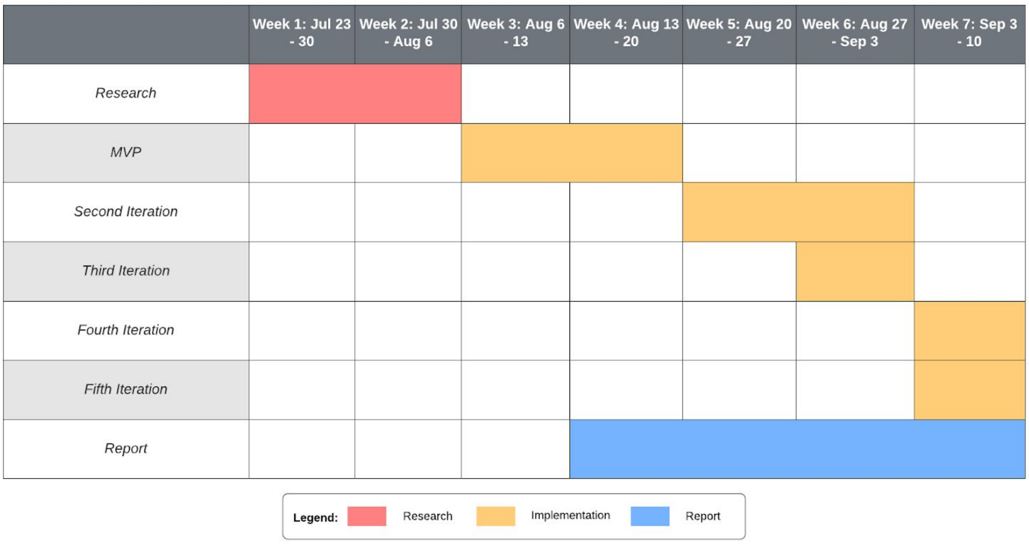
Following the delivery of the MVP, the Second Iteration of development will implement **F4** (**F4.1**, **F4.2**, and **F4.3**). As with the MVP, not all possible Evaluation Points and games will be included in this iteration, only those that will be minimally sufficient for delivery. Only the six Evaluation Points included in the MVP will be addressed in this iteration, and only the first game, 'Social Scripts' will be delivered. The remaining EPs and games will be delivered in later iterations.

Iterations 3-9 will implement the following:

- Third Iteration: **F5**
- Fourth Iteration:
    - The second game for F4.2: Emotion Recognition
    - EP1.7 Is the user able to identify different emotions
- Fifth Iteration: The remaining ‘Should’ EPs:
    - EP2.4 Does the user ask 'how are you?', or some acceptable variation, at the beginning of the conversation
    - EP2.5 Does the user answer questions like 'how are you?' during greetings with an appropriate level of detail
    - EP3.2 Does the user wait for a response after asking a question
- Sixth Iteration: The remaining ‘Could’ EPs:
    - EP1.3 Does the user interact with the chatbot by typing
    - EP1.4 Does the user interact with the chatbot by speaking aloud

- EP1.5 Does the user interact a similar amount during 'games' compared with general conversation
  - EP2.3 Does the user use the chatbot's name when greeting the chatbot
  - EP4.1 Does the user say 'please' appropriately in conversation
  - EP4.2 Does the user say 'thank you' appropriately in conversation
- Seventh Iteration: **F6**
- Eight Iteration: **F7**
- Ninth Iteration: **F8**

Whilst all functional requirements (excluding the Won’t requirement: F9) have been included in this roadmap, given the timeframe it is not going to be possible to deliver all of them. In order to effectively manage the time available to deliver as many of these iterations as possible, whilst also ensuring this report can be delivered by the deadline, it is helpful to develop a timeline of the work to be carried out. Figure 5 below shows a simple Gantt chart of the work to be done in this shorter, 'resubmit' timeframe.



	Week 1: Jul 23 - 30	Week 2: Jul 30 - Aug 6	Week 3: Aug 6 - 13	Week 4: Aug 13 - 20	Week 5: Aug 20 - 27	Week 6: Aug 27 - Sep 3	Week 7: Sep 3 - 10
Research	🟥	🟥					
MVP			🟨	🟨			
Second Iteration					🟨	🟨	
Third Iteration						🟨	
Fourth Iteration							🟨
Fifth Iteration							🟨
Report				🟦	🟦	🟦	🟦

Legend: 🟥 Research 🟨 Implementation 🟦 Report

Figure 5: A Gantt chart of the expected timeline of work

Following the first meeting with my project supervisor, the following plan of action on a weekly basis was developed for the five weeks before the deadline to outline when work on both the system and the report would be carried out:

Week 1 - August 6th-13th

- F1.1
- F1.2
- F1.3
- F2

Week 2 - August 13th-20th

- F3
- MVP complete
- Rough Draft of design
- Rough draft of Implementation

Week 3 - August 20th-27th

- Rough draft of Evaluation
- Rough draft of Future Work
- Rough draft complete
- Begin Second Iteration:
    - F4.1

Week 4 - August 27th-September 3rd

- Complete Second Iteration
    - F4.2
    - F4.3
- Refine rough draft into complete First Draft
- Third Iteration

Week 5 - September 3rd-10th

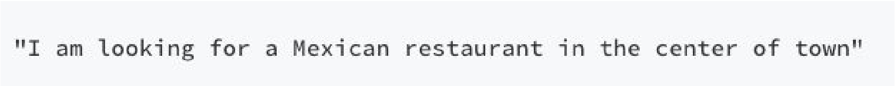
- Make revisions to Report
- Fourth Iteration
- Fifth Iteration

Chapter 5

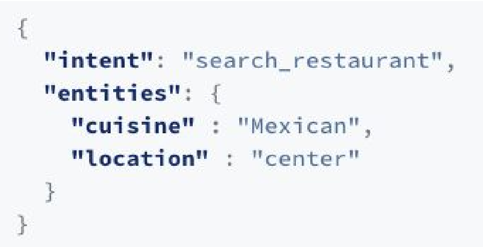
Implementation

5.1 Software & Tools

The first step in implementing the proposed chatbot is to decide on the software, libraries and other tools that will be used in order to best execute the given requirements. The two primary question marks are what technologies will be used to create the chatbot and the web application. As chatbots have rapidly become more common over the past few years, an increasing number of viable chatbot engines have been created for developers. These range from the more rigid platforms intended for use in customer service in specific industries, to more powerful software that allows developers to deliver a much more flexible and natural AI assistant. One of the most advanced, powerful and ambitious examples of the latter is Rasa. Rather than relying solely on premade scripts or state machines to navigate conversation, Rasa uses cutting edge natural language understanding (NLU) and machine learning techniques to "determine intent and capture key contextual information." [23] Rasa "learns interactively from real conversations" and powers meaningful and natural-sounding conversations. Given this potential for cutting-edge, natural, and flexible chatbots, Rasa is the software that will be used for this project. Rasa consists of two parts: Rasa NLU and Rasa Core. Rasa NLU is an "open-source natural language processing tool for intent classification, response retrieval and entity extraction in chatbots." It takes a user message as a string and returns the structured data associated, i.e. the 'intent' of the message and any useful entities. For example:



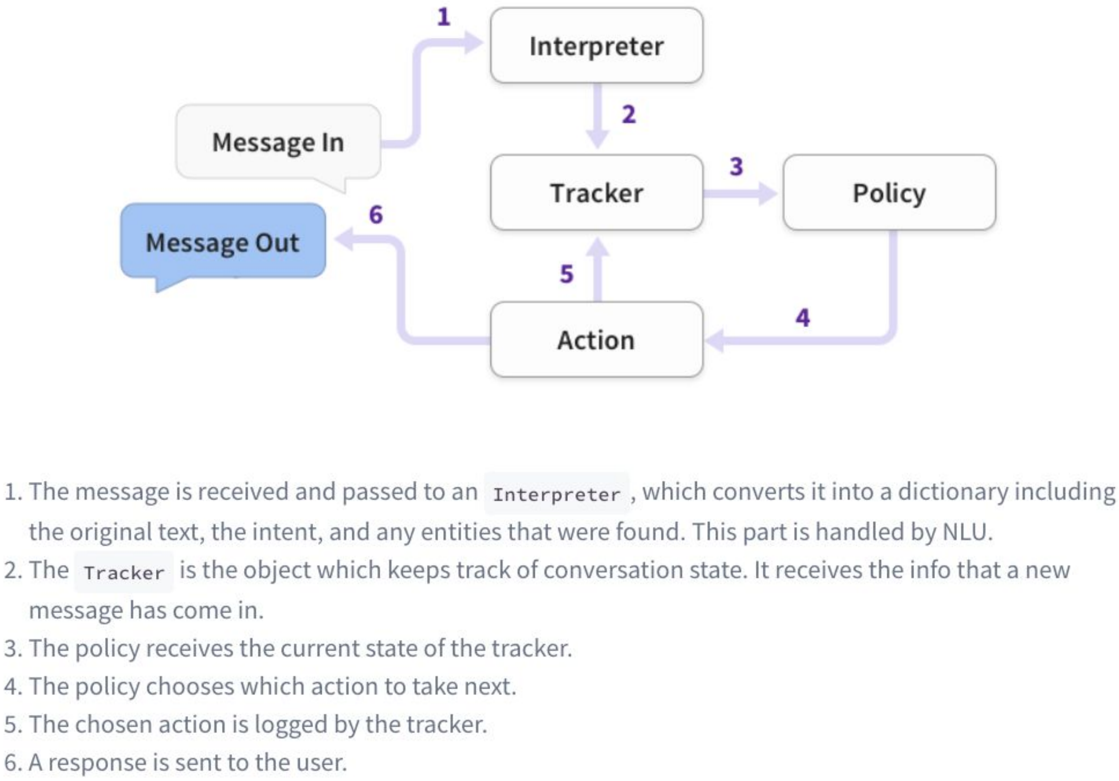
```
"I am looking for a Mexican restaurant in the center of town"
```



```
{
  "intent": "search_restaurant",
  "entities": {
    "cuisine" : "Mexican",
    "location" : "center"
  }
}
```

Figure 6: An example input and output of Rasa NLU - Taken from the Rasa Docs [24]

Rasa Core is the dialogue management engine that essentially decides what to do or say next based on factors such as the intent of the previous user message and the contextual history of the conversation. Rasa also contains an Actions server that allows the developer to create and execute any custom actions to be run during a chatbot's conversation with a user. This allows for much more control and flexibility than if absolutely everything was left to the dialogue engine. Rasa NLU and Rasa Core work together in the overall Rasa stack to take a user message as input and react accordingly. Figure 7 below outlines this process:



1. The message is received and passed to an `Interpreter`, which converts it into a dictionary including the original text, the intent, and any entities that were found. This part is handled by NLU.
2. The `Tracker` is the object which keeps track of conversation state. It receives the info that a new message has come in.
3. The policy receives the current state of the tracker.
4. The policy chooses which action to take next.
5. The chosen action is logged by the tracker.
6. A response is sent to the user.

Figure 7: Rasa Architecture - Taken from the Rasa Docs [25]

Rasa allows deployment in a variety of ways to a variety of platforms, including Facebook Messenger and Slack, and includes an HTTP API to connect the chatbot to a custom website. One of the most common frameworks for web application development with Rasa is Django. Django is a "high-level Python Web framework that encourages rapid development and clean, pragmatic design." It is both powerful and easy to use, and has the added benefit as using the same programming language as much of Rasa. It is very widely used so enjoys comprehensive documentation and a fantastic development community. These qualities, along with the fact that I have a little prior experience with it, make Django the ideal choice for developing the web application in this project. Django 3.1, the latest version of Django as of the start of development, will be used for this project, powered by Python 3.8.5. A CSS framework will be used to aid in the swift development of a clean and responsive user interface. Bulma is a modern, easy to learn, open source CSS framework that enables the delivery of a good looking, flexbox based UI in relatively minimal time. Since the chatbot is mostly developed in Python, the PyCharm IDE will be the development environment of choice. Git will be used for version control, with a private repository hosted on GitHub. Changes will be pushed to the repository from the command line upon the initial completion of the different components that will be added to the chatbot.

The chatbot will require three servers to be running simultaneously: the Django application server, the Rasa server, and the Rasa Action server. The architecture of the system will look as follows:

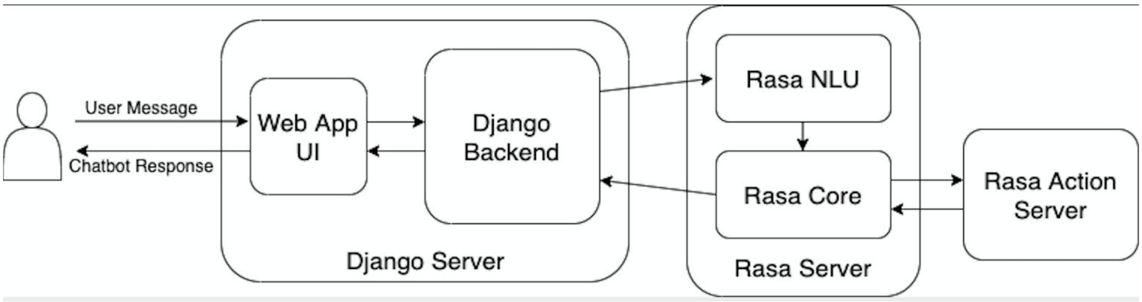


Figure 8: Chatbot System Architecture

5.2 Web Application

The project is organised such that the Rasa chatbot and the Django web app are separate directories within the main project directory. First, the Django environment was initiated. The Django project, named *djangoproject* was created by running the following command in the terminal:

```
django-admin startproject djangoproject
```

Amongst other things, this project contains the *manage.py* file which facilitates a number of command line controls including launching the Django server, *djangoproject/settings.py* where the configuration of the project can be altered, and *djangoproject/urls.py* which contains the URLs for the project as a whole. Within this project, the Django application, named *webapp* in this case, was created next by running the following command:

```
python3 manage.py startapp webapp
```

This creates the *webapp* directory containing the default files and directories that follow the Django convention for a web application.

With the Django environment set up, the next task is to create a basic webpage that will serve as the chatbot user interface. An *index.html* file is created following the Django convention, meaning it can be found in the *djangoproject/webapp/templates/webapp* directory. This specific file structure allows Django to find the file when it is rendered in the *views.py* file in the *webapp* directory. The *webapp* directory has its own *urls.py* file that contains the urls that can be found in that specific app, so the index url needs to be added there. This *webapp/urls.py* file should then be added to the urls in *djangoproject/urls.py* so it can be found by the Django project. With these additions to the base Django project, the index webpage is now ready for viewing on the internal IP at port 8000 by default by starting the Django server:

```
python3 manage.py runserver
```

The structure is now in place to dive into creating the chatbot UI by adding to the *index.html* file. In order to achieve the basic, messenger-style UI for the chatbot, it is necessary to design the html using the CSS framework. Bulma is imported using a CDN service, followed by the custom CSS file created in *webapp/static/css/style.css*:

```
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bulma@0.9.0/css/bulma.min.css">
<link rel="stylesheet" type="text/css" href="{% static
'css/style.css' %}">
```

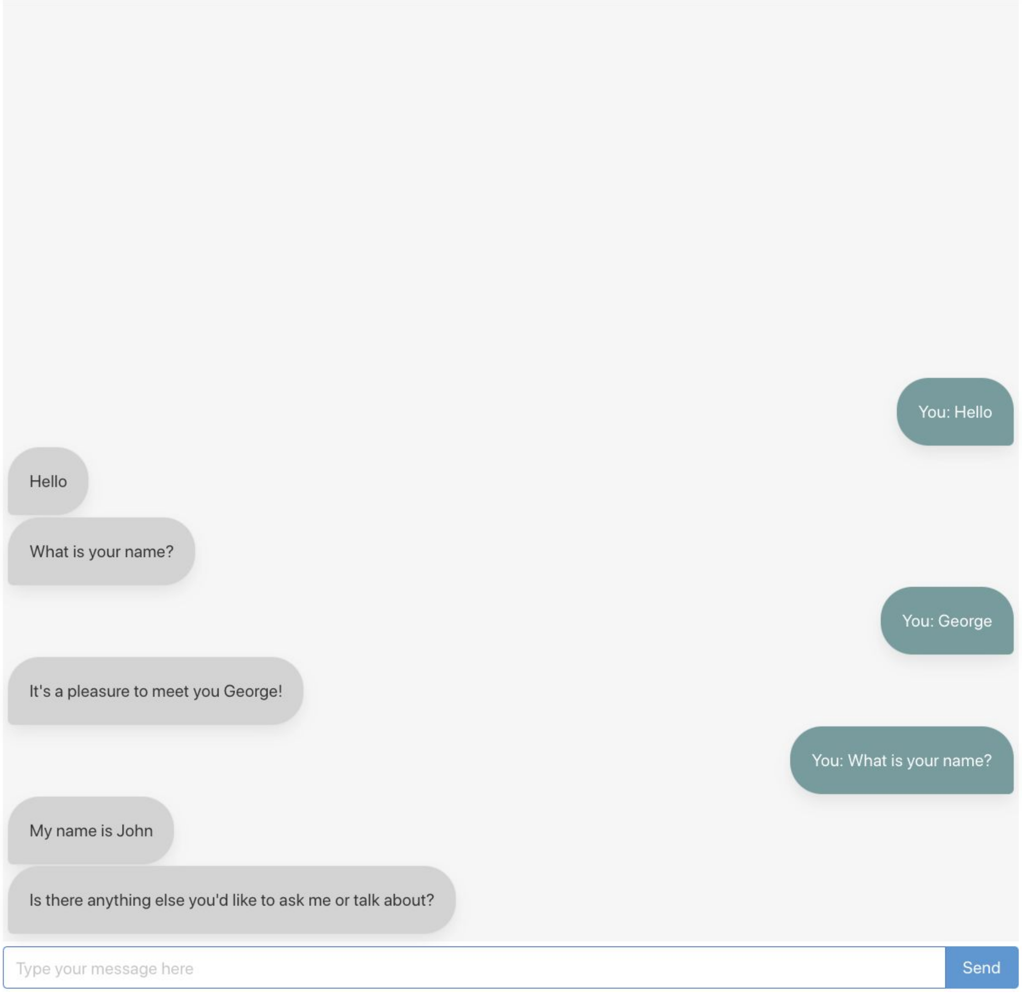
Rather than relying on hard-coded file paths, Django allows static files to be loaded into the views using this {% static %} notation. Simply by creating the specific directory structure to follow the Django convention, and by adding the code below to the *djangoprojects/settings.py* file, static CSS and JavaScript files can be loaded easily and consistently.

```
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

By making use of Bulma's container, navbar and input classes, the html for the chatbot UI can be made in relatively few lines of code:

```
<div class="container chatbox">
    <ul id="messagebox"></ul>
</div>
<nav class="navbar is-fixed-bottom">
    <div class="container">
        <div class="field has-addons">
            <div class="control is-expanded">
                <input class="input" type="text" placeholder="Type
your message here" autofocus>
            </div>
            <div class="control">
                <a class="button is-info send">Send</a>
            </div>
        </div>
    </div>
</nav>
```

With this structure, whenever messages are sent, by either the user or the chatbot, the message can be simply be added as a list item (<li>*)* to the #messagebox unordered list (<ul>). The '*column-reverse'* flex-direction assigned to the *chatbox* class ensures that new messages are added to the bottom of the UI, just like other messaging services that the user may be familiar with, which will help make conversing with the chatbot feel more natural to the user. Messages were given rounded corners to look like speech bubbles, and user messages and chatbot messages appear in different colours on opposite sides of the UI in further examples of the UI being designed to look more similar to other common messaging services.



You: Hello

Hello

What is your name?

You: George

It's a pleasure to meet you George!

You: What is your name?

My name is John

Is there anything else you'd like to ask me or talk about?

Figure 9: Chatbot UI

5.3 Initiating the Rasa Chatbot

With the UI complete, the next step in development is to set up the Rasa chatbot environment. After installation using Pip and the initialisation of the Rasa project, the core files of a Rasa chatbot containing a very basic demo bot were created:

- *init.py*
- *actions.py*
- *config.yml*
- *credentials.yml*
- *data/nlu.md*
- *data/stories.md*
- *domain.yml*
- *endpoints.yml*
- *models/20200810-211133.tar.gz*

Actions.py is where any custom actions will go. Custom actions can do anything you can write in Python, with the most useful applications for this project being to store specific information about the conversation and send messages to the user.

Config.yml contains the information necessary to configure Rasa Core and Rasa NLU. This information includes the NLU pipeline: the sequence of components that process messages received. The pipeline used for this project is as follows:

```
pipeline:
  - name: SpacyNLP
    model: "en_core_web_md"
    case_sensitive: false
  - name: SpacyTokenizer
    intent_tokenization_flag: True
    intent_split_symbol: "+"
  - name: SpacyFeaturizer
  - name: RegexFeaturizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: "char_wb"
    min_ngram: 1
    max_ngram: 4
  - name: DIETClassifier
    epochs: 100
  - name: EntitySynonymMapper
  - name: ResponseSelector
    epochs: 100
```

To breakdown what this all means:

- The project uses a medium-size SpacyNLP model with English language pre-trained word embeddings.
- The Tokenizer splits the text up into separate tokens, which is useful for predicting more than one intent per message (multiple intents separated by the "+" character in this case).
- The Featurizers create feature vectors of the text using pre-trained word embeddings, which will be used to make predictions about things like the intent of the message and any named entities present.
- The DIETClassifier is a cutting-edge combined intent classifier and entity extractor developed by Rasa. "DIET is a multi-task transformer architecture that handles both intent classification and entity recognition" which "improves upon the current state of the art, outperforms fine-tuning BERT, and is six times faster to train." [26]
- The EntitySynonymMapper allows you to define synonyms in your training data to increase the chatbot's ability to handle certain inputs.
- The ResponseSelector chooses which response to go with from a set of candidate responses.

Also included in this file are the policies that make up the configuration for Rasa Core:

```
policies:
  - name: MemoizationPolicy
  - name: TEDPolicy
```

```
    max_history: 5
    epochs: 100
  - name: MappingPolicy
  - name: "FallbackPolicy"
    nlu_threshold: 0.3
    core_threshold: 0.3
    fallback_action_name: "action_default_fallback"
```

- The MemoizationPolicy simply memorises conversations in the training data and predicts the next action with confidence 1.0 if the conversation exists and 0.0 if it does not.
- The TEDPolicy is Rasa's cutting edge Transformer Embedding Dialogue (TED) policy which is another policy used to predict the appropriate next action.
- The MappingPolicy allows actions to be mapped explicitly to intents in the *domain* file.
- The FallbackPolicy tells the chatbot which action to take in the event that no other actions are predicted with confidence greater than the given threshold values. '*Action_default_fallback'* is a default action in Rasa and is mapped to the *'utter_default'* response template that should be defined in the *domain* file.

Policies in Rasa Core are intended to be used together, and there are several other policies that could be used. It is therefore necessary to define a policy hierarchy to identify which policy should take precedence in a given situation. This hierarchy is defined in Rasa itself. The policies used in this project are ordered with the FallbackPolicy taking highest priority, then the MemoizationPolicy, then the MappingPolicy, then the TedPolicy taking lowest priority.

Credentials.yml is where the information for connecting to other services can be found, including the *rest* channel, which this project will use to connect to the web application, as opposed to the socketio channel. This is because it is simpler and easier to simply use JavaScript in the web application than to attempt to send and receive messages via websockets. Similarly, *Endpoints.yml* contains information for connecting to other channels. For this project, the only thing present in this file will be the url to which information from the web application will be sent:

```
action_endpoint:
  url: "http://localhost:5055/webhook"
```

The *data* directory contains the training data used to create the chatbot model. The *nlu.md* file contains the 'intents' used in the system and the example user inputs for each intent that the NLU model will be trained on. For example:

```
## intent:goodtoseeyou
- good to see you
- good to see you again
- good to talk to you again
- great to see you
- great to see you again
- great to talk to you again
- great seeing you
- glad to see you again
- glad to talk to you again
- happy to see you
- Nice to talk to you today John!
```

Training data taken from *data/nlu.md*

Intents can also be used to give examples of different named entities. This helps Rasa NLU detect a named entity and better identify the associated intent. For example, in this project the basic named entities include the user's name and their location, which would be expressed in *nlu.md* as follows:

```
## intent:inform
- my name is [george](name)
- I'm called [toby](name)
- I am called [james smith](name)
- I'm from [Redditch](location)
- I live in [Birmingham](location)
- [Warwick](location)
- [London, England](location)
```

More advanced training data taken from *data/nlu.md* that includes entities

The *stories.md* file contains the example stories used to train the Rasa Core dialogue management model. Stories are example conversations between the user and the chatbot that outline what the chatbot might do in response to different user intents, according to the context of the conversation. Stories can either be hard coded as initial training data, or generated by the *rasa interactive* function that will be discussed in more detail later on. For example:

```
## greet + name + howdoing + happy
* greet
  - action_utter_greet
  - action_ask_whatisyourname
* inform{"name": "adam"}
  - slot{"name": "adam"}
  - action_user_inform
  - action_utter_goodtomeetyou
  - utter_ask_howdoing
* mood_great
  - action_user_mood_great
```

Training data taken from *data/stories.md*

Received intents are denoted by a "*" and the chatbot's actions by a "-". The names of stories are only important for debugging; they have no bearing on the model itself.

When named entities are detected by the NLU, the values are stored in the appropriate 'slot'. Slots are essentially the chatbot's memory. They can be a number of different types such as Text, Boolean, Categorical, Float, and Unfeaturized. Contrary to the developer's likely intuition, slots of type Text do not actually store the value of the entity, only whether or not they have been set already. Slots of most types will have an influence on the conversation if they have not set. Unfeaturized slots are the exception to this, and can hold any type of value. It is not necessary to have the slots influence the conversation in this project, so the slots used will all be Unfeaturized slots. Slots can be set automatically by the NLU when a named entity is detected, or they can be set explicitly by a custom action.

Domain.yml lists all of the actions, entities, intents, slots, and template responses in the chatbot. Whilst most things are just listed, some contain additional information. For example, intents can specify an action that would automatically be triggered according to the MappingPolicy. Template responses can consist of multiple responses, one of which will be chosen randomly by Rasa Core. This increases the natural feeling of conversing with the chatbot as the user won't always receive the same template response to their messages. For example:

```
utter_goodbye:
  - text: Goodbye
  - text: Goodbye! Nice talking to you
  - text: See you soon!
  - text: Bye!
  - text: Talk to you soon! Bye
```

A template response taken from *domain.yml* with multiple options

Slots are initialised simply by giving the name of the slot and the type. Where appropriate, an initial value can also be set.

Finally, the *models* directory contains the models of the chatbot that are created each time it is trained. The most recent of these models is used by default when the Rasa server is run.

5.4 Connecting the Web Application and the Chatbot

With this chatbot environment initiated and the basic demo chatbot trained, the chatbot can be interacted with via the command line by running 'rasa shell', and the Rasa server can be run. To run the Rasa server, the following command is executed in the chatbot directory in the terminal:

```
rasa run -m models --enable-api --cors "*" --debug
```

The '-m' argument specifies the path to the trained models. The '—enable-api' argument starts the web server API so that the server can communicate with another service (the Django web application in this case). The '—cors "*"' argument enables Cross-Origin Resource Sharing (CORS) for all origins. This is acceptable for this project as the chatbot will only be used on the local machine, but in production it would be necessary to consider the security implications of this. The '—debug' argument sets the logging level to 'DEBUG' to print information to the terminal for debugging during use.

When the server is running, Rasa handles incoming messages and responds appropriately so the next job is to make the necessary additions to the web application to communicate with the Rasa server. As mentioned earlier, the web application will communicate with the chatbot via the chatbot's HTTP API. JavaScript and jQuery will be used to asynchronously send and receive messages, and so at the end of the HTML <body> jQuery is imported from the jQuery CDN and the *js/script.js* file is imported using the same {% static %} notation used for the CSS.

In *script.js,* the first step is to send the message entered in the input box when the user clicks send. This will be achieved in two parts: 1) get the message from the input and ensure it's a valid message, 2) send an Ajax request to the Rasa webhook defined in the chatbot's *endpoints.yml.* For the first part, basic jQuery is used to get the message from the input and check it isn't empty:

```
$('.send').click(function() {
       var message = $('.input').val();

       if (message == "" || message.trim() == "") {
           return false;
```

The user's message is then displayed in the UI using the Bulma 'card' class along with the custom 'user-message' class:

```
var userMessage = '<li class="card user-message">You: ' + message +
'</li>';
 $(userMessage).appendTo('#messagebox');
```

```
scrollToBottom();
```

The *scrollToBottom* function ensures the UI always displays the most recent message by scrolling down to the bottom of the page:

```
function scrollToBottom() {
    $(document).scrollTop($(document).height());
}
```

For the sake of increased usability, a simple function has also been added that executes this process in response to the enter button being pressed in addition to the 'send' button being clicked.

For the second task, the message is trimmed of any excess whitespace, and sent to the *sendToRasa* function. This function uses the jQuery *ajax* function to send the message to the Rasa webhook using the default HTTP template defined in the API:

```
$.ajax({
        url: 'http://localhost:5005/webhooks/rest/webhook',
        type: 'POST',
        data: JSON.stringify({
            "message": message,
            "sender": "user"
        }),
        success: function(message, status) {
            postChatbotMessage(message);
            console.log('Rasa Response: ' + message + '\n Status: '
+ status);
        },
        error: function(errorMessage) {
            postChatbotMessage("");
            console.log('Error: ' + errorMessage);
        }
});
```

In this format, the Rasa server can receive and handle the message appropriately, processing it with the Rasa NLU and then decide what to do next with Rasa Core, replying asynchronously through the Ajax request. The POST request will then return either a success or error. In the case of a success, the chatbot's message will be received along with a status. Once the chatbot's response is received it will be added to the UI through the *postChatbotMessage* function. The message received from the Ajax request is more than just a simple string of text; it can include other elements such as images and buttons. For example:

```
[
  {"text": "Hey!"}, {"image": "http://example.com/image.jpg"}
]
```

Example json object taken from the Rasa documentation

It is therefore necessary to traverse the message object, looking for the *text* elements, which can then be added to the UI in a similar way to the user messages:

```
for (i = 0; i < message.length; i++){
    if (message[i].hasOwnProperty("text")) {
        var chatbotMessage = '<li class="card chatbot-message">' +
message[i].text + '</li>';

$(chatbotMessage).appendTo('#messagebox').hide().fadeIn(100);
    }
}
```

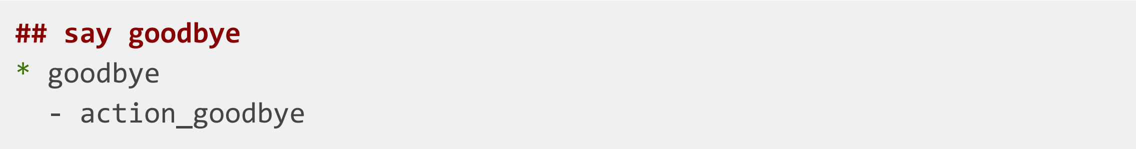
In the case that an error message is received from the Ajax request, the *postChatbotMessage function* will simply be sent the empty string. If the empty string is received, the *postChatbotMessage* function will add some kind of error message to the UI to explain the error.

The web application and the Rasa chatbot are now able to communicate, and the user can interact with the chatbot via the UI by sending and receiving messages. The only other thing to note before concluding this stage of development is that, unlike many other chatbot software platforms which can take several seconds to reply to a user message, Rasa replies virtually instantly. It is therefore necessary to delay the posting of the chatbot messages slightly with the JavaScript *setTimeout* function, to make the replies seem a little more natural.

5.5 Improving the Chatbot

With requirement F1 satisfied, the next step is to expand the chatbot's capability to the point that it can hold a very basic conversation in a relatively natural way. This will be achieved firstly by adding a significant amount of training data to *data/nlu.md* and *data/stories.md* manually, and secondly by interacting with the chatbot and correcting any mistakes through *rasa interactive*. As the MVP only includes six very basic evaluation points, only a relatively small selection of intents are necessary. For F2, 25 basic intents were added. Each intent was given a number of examples in *data/nlu.md,* like the few examples given above for 'intent:goodtoseeyou' and 'intent:inform'. The total number of examples in *data/nlu.md* is 744, the mean number of examples was 29.76, the smallest number of examples was 5 and the largest number was 104, with the range therefore being 99. This large number of examples allows the NLU model to better predict the correct intent from a wide range of potential user inputs, and generalise better to user messages not given in the training data.

Stories are then created for example conversations that include these intents. Over 30 example stories were given initially, with more generated later on through *rasa interactive*. Some intents don't require the context of the conversation history to correctly predict the next action, so feature in the simplest possible stories. For example:



```
## say goodbye
* goodbye
  - action_goodbye
```

Minimal story taken from the *data/stories.md* training data

Other intents rely on this kind of context so feature in much more elaborate stories, such as the story given in the earlier "Initiating the Rasa Chatbot" section.

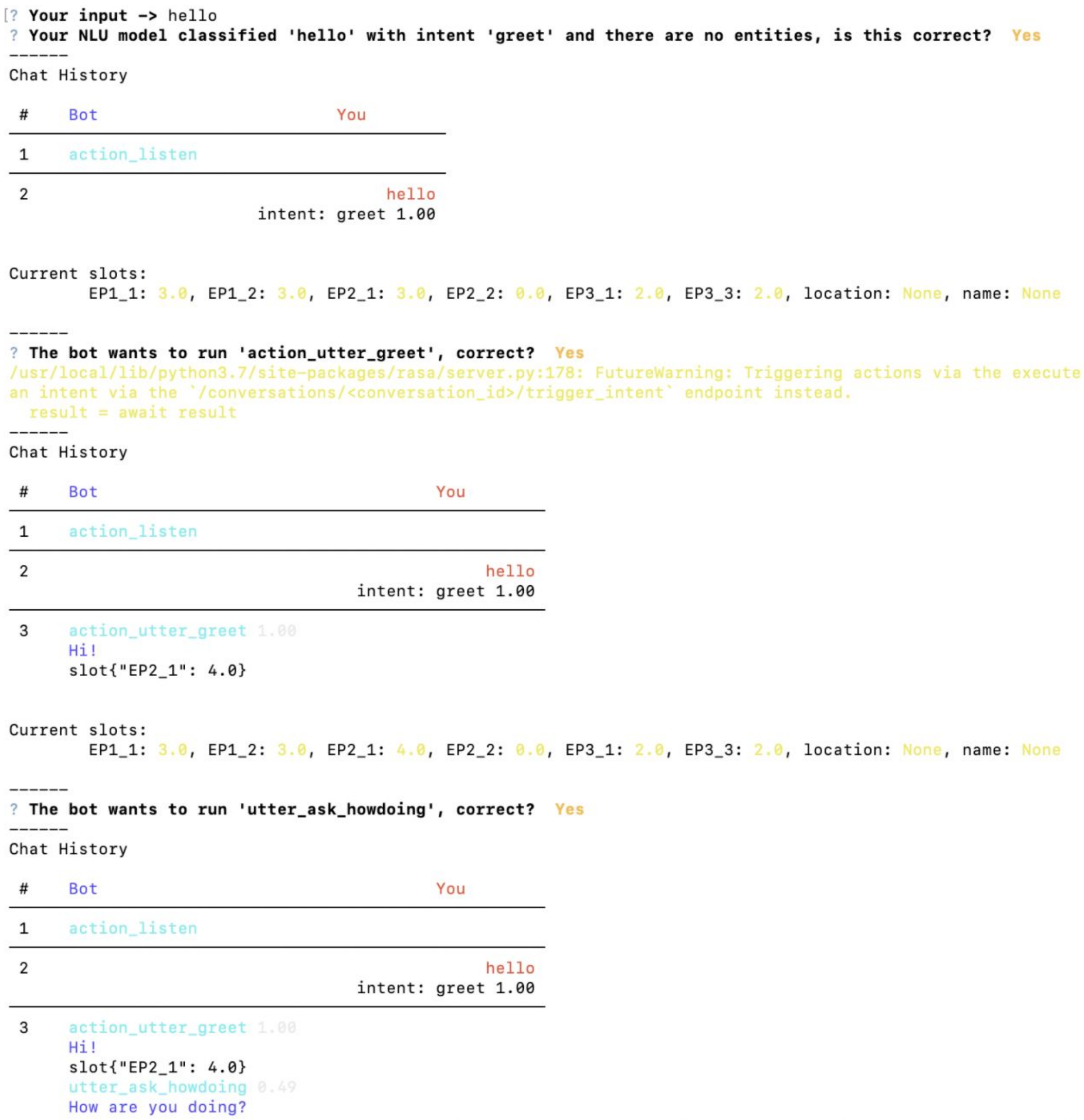
These stories outline the appropriate chatbot action in response to the given intents that is used in the MemoizationPolicy, so it is necessary to define these actions in the chatbot. Simple utterance actions (those that begin with "utter") are defined directly in the *domain* file. For example, the '*utter_goodbye'* is given in the earlier "Initiating the Rasa Chatbot" section. Some stories may include custom actions which are given in the *actions.py* file. A custom action can send a template response to the user, just like the utterance actions, but can do other, more powerful things in addition. For this stage of development, it is not necessary to create many custom actions. One example is '*action_utter_greet'* which sends a personalised greeting if the name of the user is known, or sends a template greeting if not.

```
class ActionUtterGreet(Action):

    def name(self) -> Text:
        return "action_utter_greet"

    def run(self, dispatcher, tracker, domain):

        name = tracker.get_slot("name")

        if name is not None:
            dispatcher.utter_message(f'Hi {name}!')
        else:
            dispatcher.utter_message(template="utter_greet")

            return []
```

As you can see, actions are defined as Python classes, with a 'name' function which gives the name of the action to link it to the chatbot domain, and a 'run' function which contains the custom code to be executed. The *tracker* contains useful information about the current state of the chatbot such as previous actions, slots and more that can be used in the custom actions. The *dispatcher* allows the action to send a message to the user. This can be a response template defined in the domain (as in the above example), or can be hard-coded in the action. Custom actions are housed on the Rasa Actions server, which is required to be running in a separate terminal alongside the Rasa server. When custom actions are called in Rasa Core, the Rasa server communicates with the Actions server to execute the appropriate action.

Once the model is trained, the user is able to hold a very basic conversation with the chatbot, such as the one shown in Figure 9. Rasa Core will adhere to the policy hierarchy to decide what action to take in response to each user input. It will first check to see if the confidence of the predicted actions from the other policies is below the threshold of the FallbackPolicy. If an action has confidence greater than that threshold it will move onto the MemoizationPolicy to see if the exact conversation exists in the stories. If it does, the chatbot will choose the next action given in the story. If it doesn't, the chatbot will move on to the MappingPolicy which looks for actions triggered like the '*insult'* intent above triggers the '*action_respond_insult'.* If that also fails to provide the chatbot with an appropriate response, the TEDPolicy will attempt to calculate the most appropriate action using it's predefined architecture (outlined in the Rasa documentation).

With sufficient training data added for the chatbot to hold a very basic conversation, the chatbot was run with *rasa interactive* to identify flaws in the system and add more training data to fix them. *Rasa interactive* allows the developer to interact with the chatbot via the command line, but gives the developer the chance to approve the predicted action or flag up a mistake. Upon completion of the interaction, new training data can be saved to *data/nlu.md* and *data/stories.md* to fix those mistakes and further improve the robustness of the chatbot.



```
[? Your input -> hello
 ? Your NLU model classified 'hello' with intent 'greet' and there are no entities, is this correct?  Yes
------
Chat History

 #    Bot                           You
──────────────────────────────────────────
 1    action_listen
──────────────────────────────────────────
 2                                 hello
                         intent: greet 1.00


Current slots:
        EP1_1: 3.0, EP1_2: 3.0, EP2_1: 3.0, EP2_2: 0.0, EP3_1: 2.0, EP3_3: 2.0, location: None, name: None

------
? The bot wants to run 'action_utter_greet', correct?  Yes
/usr/local/lib/python3.7/site-packages/rasa/server.py:178: FutureWarning: Triggering actions via the execute
an intent via the `/conversations/<conversation_id>/trigger_intent` endpoint instead.
  result = await result
------
Chat History

 #    Bot                                     You
────────────────────────────────────────────────────
 1    action_listen
────────────────────────────────────────────────────
 2                                           hello
                                   intent: greet 1.00
────────────────────────────────────────────────────
 3    action_utter_greet 1.00
      Hi!
      slot{"EP2_1": 4.0}


Current slots:
        EP1_1: 3.0, EP1_2: 3.0, EP2_1: 4.0, EP2_2: 0.0, EP3_1: 2.0, EP3_3: 2.0, location: None, name: None

------
? The bot wants to run 'utter_ask_howdoing', correct?  Yes
------
Chat History

 #    Bot                                     You
────────────────────────────────────────────────────
 1    action_listen
────────────────────────────────────────────────────
 2                                           hello
                                   intent: greet 1.00
────────────────────────────────────────────────────
 3    action_utter_greet 1.00
      Hi!
      slot{"EP2_1": 4.0}
      utter_ask_howdoing 0.49
      How are you doing?
```

Figure 10: An example interaction with the chatbot through *Rasa interactive*

In the event that an incorrect intent is predicted or an entity is not correctly identified, responding “no”, rather than “yes” as in the example above, allows the developer to choose from the entire list of intents (ordered by prediction confidence) or highlight exactly which characters in the user’s message make up the entity.

5.6 Implementing Evaluation Points

With the chatbot now able to hold a basic conversation with the user, the next stage of development is to incorporate the EPs that make up F3 for the MVP. In order to keep track of the user's 'performance' with respect to each EP, a value between 0 and 5 will be stored in a slot for each EP, where 0 indicates a complete lack of understanding of that social skill and 5 indicates a PNT level understanding. Each EP will be given an initial value which can vary from EP to EP to best reflect the nature of that skill within the 0-5 scale. EPs will be increased/decreased in different increments (such as 0.5, 1, or 2) for the same reason. While previous parts of this project follow the intended use of the software more closely, implementing these Evaluation Points moves significantly further away from this. This stage of development therefore proved to require significantly more creative thinking and problem solving to use the tools available to solve this more novel problem.

The first EP to implement is EP1.1: Does the user interact with the chatbot. First, the slot for this EP is set in the domain (*domain.yml)* as follows:

```
EP1_1:
  initial_value: 3.0
  type: unfeaturized
```

Unlike most other EPs, EP1.1 will be tested in the Django web application rather than by the Rasa chatbot itself. 'Interaction' with the chatbot will be split into three categories: interaction, slow interaction, no interaction. Once the web application has displayed the chatbot's message in the *postChatbotMessage* function, it will listen out for the user's response. Two 'timeouts' will be set using the JavaScript *setTimeout* function. When the user begins typing in the input box the timeouts get cleared and Django lets Rasa know that the user has interacted by sending the message "/interaction" through the *sendToRasa* function. If there is no response after 12 seconds, the web application will send the message "/slow_interaction" to Rasa. If there is still no response a total of 30 seconds after the chatbot's message is posted, the web application will send the message "/no_interaction".

```
var slow_interaction = setTimeout(function()
{sendToRasa('/slow_interaction')}, 12000);
var no_interaction = setTimeout(function()
{sendToRasa('/no_interaction')}, 30000);
var interacted = false;
$('.input').keypress(function() {
   if (!interacted) {
       clearTimeout(slow_interaction);
       clearTimeout(no_interaction);
       sendToRasa(('/interaction'));
       interacted = true;
  }
})
```

Extract from *djangoproject/webapp/static/js/script.js*

Preceding a message with a "/" character tells Rasa to look for an intent matching the message exactly, rather than using the NLU to predict the intent as normal. The three intents, *'interaction', 'slow_interaction', no_interaction'*, are added to the domain file and the appropriate action is assigned to each intent in *stories.md*. For example:

```
## no interaction EP1_1
* no_interaction
 - action_no_interaction
 - slot{"EP1_1": 0}
```

Training data using slots taken from *data/nlu.md*

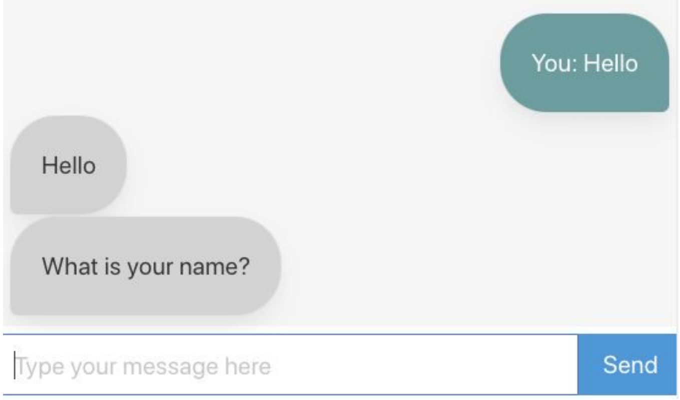
The line below the action in this example simply notes that the action should set the following slot, and gives an example value, rather than explicitly telling the chatbot what to do next like most lines in *stories.md*.

As a result of these stories, the chatbot calls the corresponding custom action when the intent is received, so the next step is to create actions in *actions.py* that will alter the EP1.1 slot value appropriately.

```
class ActionSlowInteraction(Action):
   """User interaction slow so decrease EP1_1 value towards the
middle (3)"""

   def name(self) -> Text:
       return "action_slow_interaction"

   def run(self, dispatcher, tracker, domain):
       ep1_1 = tracker.get_slot("EP1_1")

       if in_session:

           if ep1_1 > 3.5:
               ep1_1 -= 1
           elif ep1_1 > 3:
               ep1_1 -= 0.5

       return [SlotSet("EP1_1", ep1_1)]
```

The *‘in_session’* flag is used in the action to ensure that the web application doesn’t continue to tell the chatbot that there has been no interaction or a slow interaction after the conversation has concluded, which would typically be after the user has said ‘goodbye’.

To see these actions working in the chatbot:

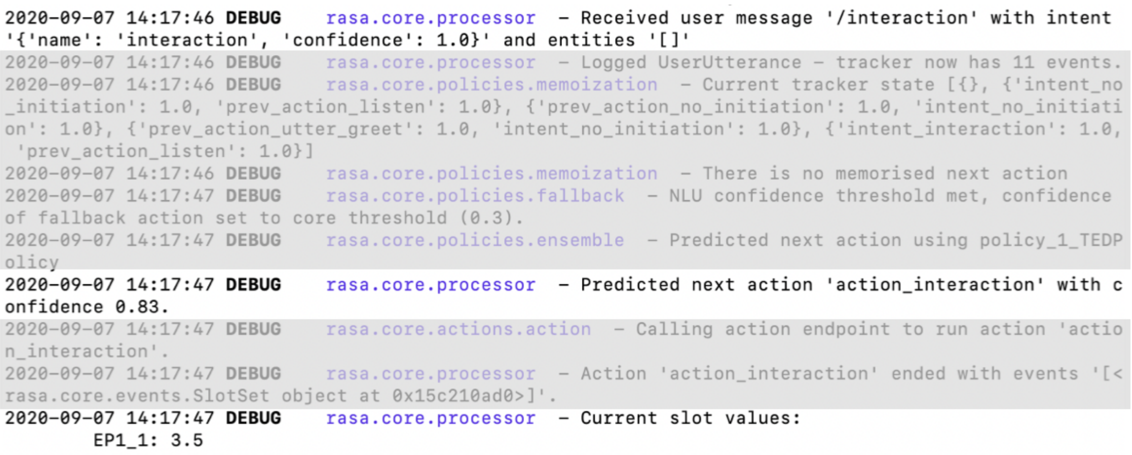


You: Hello

Hello

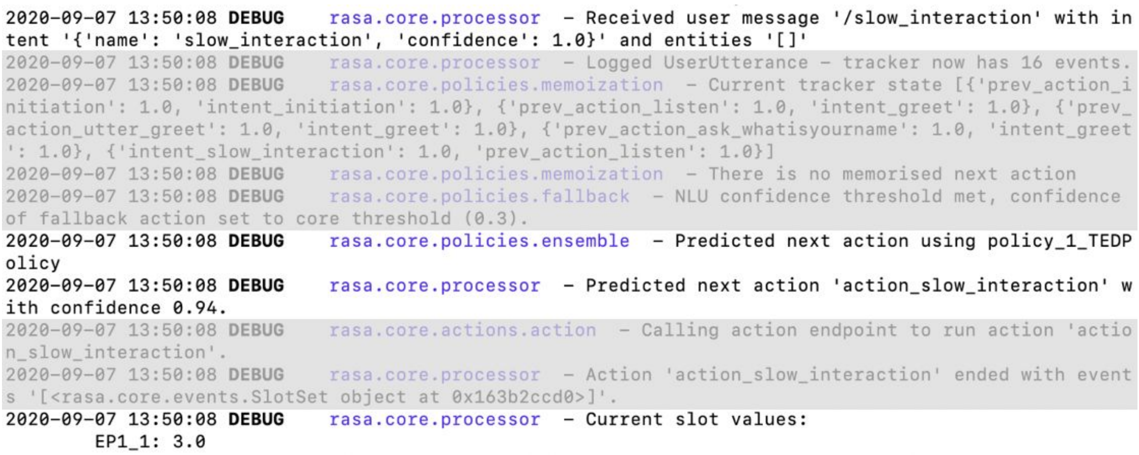
What is your name?

When the user says “hello”, the web application sends the “/interaction” message and the EP1.1 slot is increased, which can be seen in the terminal debugging output below:



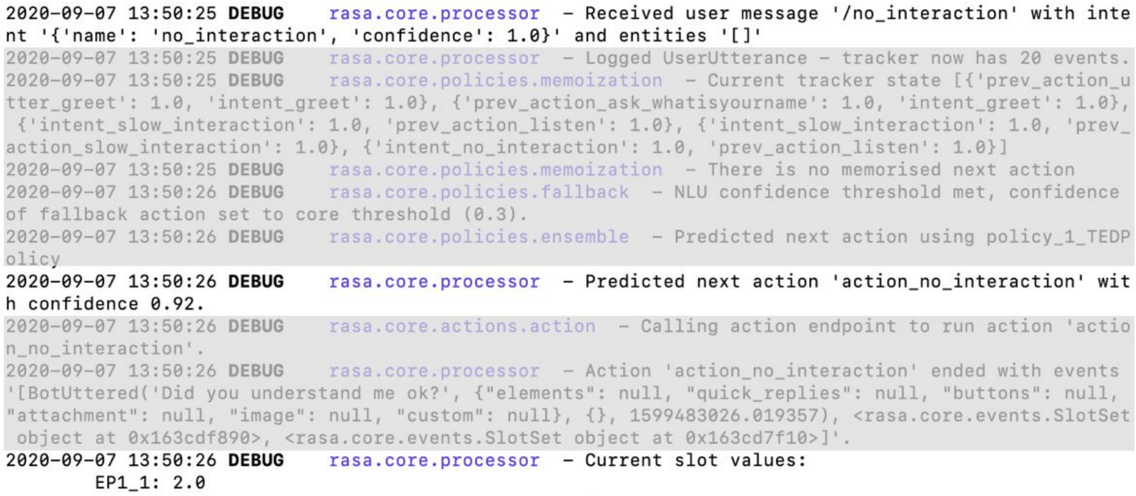
```
2020-09-07 14:17:46 DEBUG    rasa.core.processor  - Received user message '/interaction' with intent
'{'name': 'interaction', 'confidence': 1.0}' and entities '[]'
2020-09-07 14:17:46 DEBUG    rasa.core.processor  - Logged UserUtterance - tracker now has 11 events.
2020-09-07 14:17:46 DEBUG    rasa.core.policies.memoization  - Current tracker state [{}, {'intent_no
_initiation': 1.0, 'prev_action_listen': 1.0}, {'prev_action_no_initiation': 1.0, 'intent_no_initiati
on': 1.0}, {'prev_action_utter_greet': 1.0, 'intent_no_initiation': 1.0}, {'intent_interaction': 1.0,
 'prev_action_listen': 1.0}]
2020-09-07 14:17:46 DEBUG    rasa.core.policies.memoization  - There is no memorised next action
2020-09-07 14:17:47 DEBUG    rasa.core.policies.fallback  - NLU confidence threshold met, confidence
of fallback action set to core threshold (0.3).
2020-09-07 14:17:47 DEBUG    rasa.core.policies.ensemble  - Predicted next action using policy_1_TEDP
olicy
2020-09-07 14:17:47 DEBUG    rasa.core.processor  - Predicted next action 'action_interaction' with c
onfidence 0.83.
2020-09-07 14:17:47 DEBUG    rasa.core.actions.action  - Calling action endpoint to run action 'actio
n_interaction'.
2020-09-07 14:17:47 DEBUG    rasa.core.processor  - Action 'action_interaction' ended with events '[<
rasa.core.events.SlotSet object at 0x15c210ad0>]'.
2020-09-07 14:17:47 DEBUG    rasa.core.processor  - Current slot values:
        EP1_1: 3.5
```

The user's message is then sent to the chatbot, and the intent and predicted next action is determined in the same way and the chatbot sends it's reply. When there is no response to the chatbot's reply after 12 seconds, the web application sends the "/slow_interaction" and the EP1.1 slot value is decreased:



```
2020-09-07 13:50:08 DEBUG    rasa.core.processor  - Received user message '/slow_interaction' with in
tent '{'name': 'slow_interaction', 'confidence': 1.0}' and entities '[]'
2020-09-07 13:50:08 DEBUG    rasa.core.processor  - Logged UserUtterance - tracker now has 16 events.
2020-09-07 13:50:08 DEBUG    rasa.core.policies.memoization  - Current tracker state [{'prev_action_i
nitiation': 1.0, 'intent_initiation': 1.0}, {'prev_action_listen': 1.0, 'intent_greet': 1.0}, {'prev_
action_utter_greet': 1.0, 'intent_greet': 1.0}, {'prev_action_ask_whatisyourname': 1.0, 'intent_greet
': 1.0}, {'intent_slow_interaction': 1.0, 'prev_action_listen': 1.0}]
2020-09-07 13:50:08 DEBUG    rasa.core.policies.memoization  - There is no memorised next action
2020-09-07 13:50:08 DEBUG    rasa.core.policies.fallback  - NLU confidence threshold met, confidence
of fallback action set to core threshold (0.3).
2020-09-07 13:50:08 DEBUG    rasa.core.policies.ensemble  - Predicted next action using policy_1_TEDP
olicy
2020-09-07 13:50:08 DEBUG    rasa.core.processor  - Predicted next action 'action_slow_interaction' w
ith confidence 0.94.
2020-09-07 13:50:08 DEBUG    rasa.core.actions.action  - Calling action endpoint to run action 'actio
n_slow_interaction'.
2020-09-07 13:50:08 DEBUG    rasa.core.processor  - Action 'action_slow_interaction' ended with event
s '[<rasa.core.events.SlotSet object at 0x163b2ccd0>]'.
2020-09-07 13:50:08 DEBUG    rasa.core.processor  - Current slot values:
        EP1_1: 3.0
```

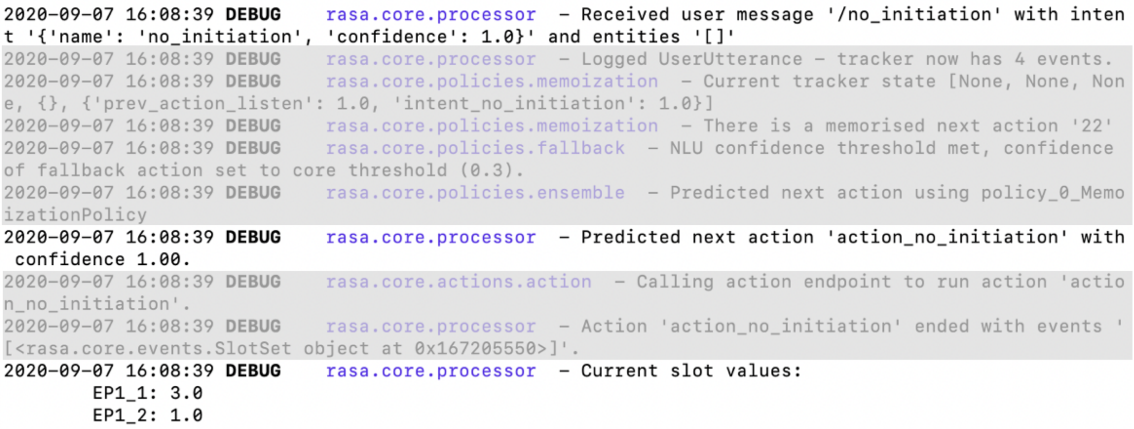
Then after 30 seconds in total without a response, the web application sends the "no_interaction" message and the EP1.1 slot value is decreased further:



```
2020-09-07 13:50:25 DEBUG    rasa.core.processor  - Received user message '/no_interaction' with inte
nt '{'name': 'no_interaction', 'confidence': 1.0}' and entities '[]'
2020-09-07 13:50:25 DEBUG    rasa.core.processor  - Logged UserUtterance - tracker now has 20 events.
2020-09-07 13:50:25 DEBUG    rasa.core.policies.memoization  - Current tracker state [{'prev_action_u
tter_greet': 1.0, 'intent_greet': 1.0}, {'prev_action_ask_whatisyourname': 1.0, 'intent_greet': 1.0},
 {'intent_slow_interaction': 1.0, 'prev_action_listen': 1.0}, {'intent_slow_interaction': 1.0, 'prev_
action_slow_interaction': 1.0}, {'intent_no_interaction': 1.0, 'prev_action_listen': 1.0}]
2020-09-07 13:50:25 DEBUG    rasa.core.policies.memoization  - There is no memorised next action
2020-09-07 13:50:26 DEBUG    rasa.core.policies.fallback  - NLU confidence threshold met, confidence
of fallback action set to core threshold (0.3).
2020-09-07 13:50:26 DEBUG    rasa.core.policies.ensemble  - Predicted next action using policy_1_TEDP
olicy
2020-09-07 13:50:26 DEBUG    rasa.core.processor  - Predicted next action 'action_no_interaction' wit
h confidence 0.92.
2020-09-07 13:50:26 DEBUG    rasa.core.actions.action  - Calling action endpoint to run action 'actio
n_no_interaction'.
2020-09-07 13:50:26 DEBUG    rasa.core.processor  - Action 'action_no_interaction' ended with events
'[BotUttered('Did you understand me ok?', {"elements": null, "quick_replies": null, "buttons": null,
"attachment": null, "image": null, "custom": null}, {}, 1599483026.019357), <rasa.core.events.SlotSet
 object at 0x163cdf890>, <rasa.core.events.SlotSet object at 0x163cd7f10>]'.
2020-09-07 13:50:26 DEBUG    rasa.core.processor  - Current slot values:
        EP1_1: 2.0
```

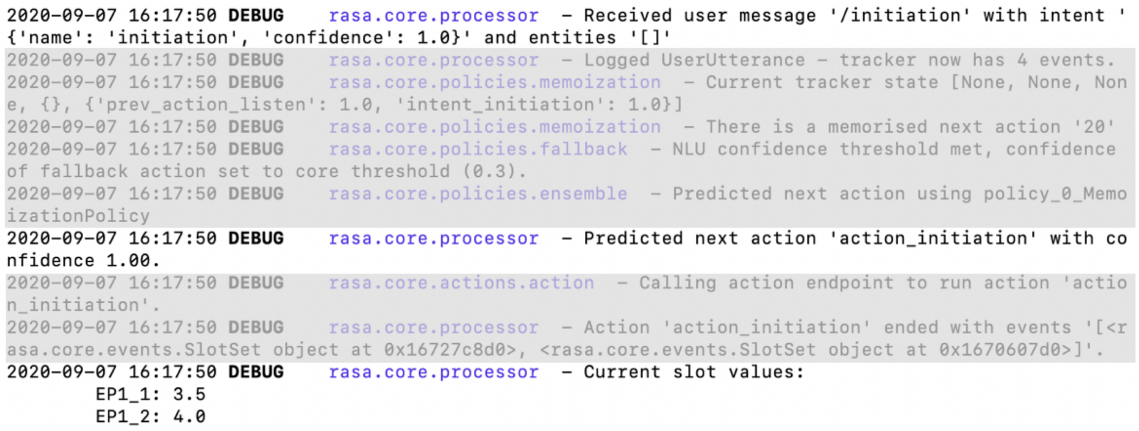
At this point, nothing is displayed in the chatbot UI to reflect these changes that are taking place behind-the-scenes on the Rasa server.

The next EP to implement is EP1.2: Does the user initiate conversation. This will be implemented in a similar way to EP1.1, with the web application sending the chatbot a message if no response is received after a certain period of time at the beginning of the conversation. The slot is set in the domain file in the same way as EP1.1, with an initial value of 3.0 and type of ‘unfeaturized’. As soon as the web page is loaded, the JavaScript listens out for the user typing anything in the input box, sending the message “/initiation” if there is input. If there is no input after 15 seconds, the web application sends the message “/no_initiation”. Both “*initiation*” and “*no_initiation*” are added as intents in the domain file, are given custom actions in *actions.py* to alter the EP1.2 slot value appropriately, and are entered in *stories.md* to call the corresponding action. The “*no_initiation”* story also includes calling “*action_utter_greet*” to ensure the conversation can go ahead without relying on the user to initiate it. The custom actions work in exactly the same way as the actions for EP1.1; the slot value is increased towards 5 if there is user initiation, and decreased towards 0 if not. This works in the chatbot as follows:



```
2020-09-07 16:08:39 DEBUG    rasa.core.processor  - Received user message '/no_initiation' with inten
t '{'name': 'no_initiation', 'confidence': 1.0}' and entities '[]'
2020-09-07 16:08:39 DEBUG    rasa.core.processor  - Logged UserUtterance - tracker now has 4 events.
2020-09-07 16:08:39 DEBUG    rasa.core.policies.memoization  - Current tracker state [None, None, Non
e, {}, {'prev_action_listen': 1.0, 'intent_no_initiation': 1.0}]
2020-09-07 16:08:39 DEBUG    rasa.core.policies.memoization  - There is a memorised next action '22'
2020-09-07 16:08:39 DEBUG    rasa.core.policies.fallback  - NLU confidence threshold met, confidence
of fallback action set to core threshold (0.3).
2020-09-07 16:08:39 DEBUG    rasa.core.policies.ensemble  - Predicted next action using policy_0_Memo
izationPolicy
2020-09-07 16:08:39 DEBUG    rasa.core.processor  - Predicted next action 'action_no_initiation' with
 confidence 1.00.
2020-09-07 16:08:39 DEBUG    rasa.core.actions.action  - Calling action endpoint to run action 'actio
n_no_initiation'.
2020-09-07 16:08:39 DEBUG    rasa.core.processor  - Action 'action_no_initiation' ended with events '
[<rasa.core.events.SlotSet object at 0x167205550>]'.
2020-09-07 16:08:39 DEBUG    rasa.core.processor  - Current slot values:
        EP1_1: 3.0
        EP1_2: 1.0
```

The EP1.2 value is decreased from it’s initial value of 3.0 to 1.0 when the chatbot receives the “/no_initiation” message. The EP1.1 value is not decreased in the case of no initiation since the two evaluation points are treated separately for this specific case. When the user does initiate the conversation, both the EP1.1 and EP1.2 slot values are increased to reflect that the user has demonstrated an ability to both initiate and interact:

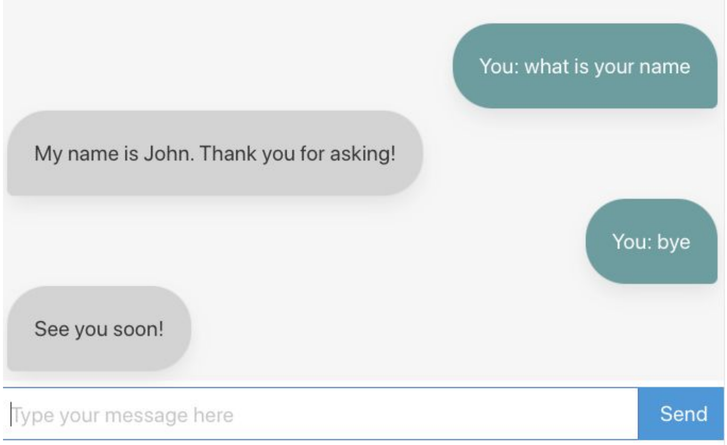


```
2020-09-07 16:17:50 DEBUG    rasa.core.processor  - Received user message '/initiation' with intent '
{'name': 'initiation', 'confidence': 1.0}' and entities '[]'
2020-09-07 16:17:50 DEBUG    rasa.core.processor  - Logged UserUtterance - tracker now has 4 events.
2020-09-07 16:17:50 DEBUG    rasa.core.policies.memoization  - Current tracker state [None, None, Non
e, {}, {'prev_action_listen': 1.0, 'intent_initiation': 1.0}]
2020-09-07 16:17:50 DEBUG    rasa.core.policies.memoization  - There is a memorised next action '20'
2020-09-07 16:17:50 DEBUG    rasa.core.policies.fallback  - NLU confidence threshold met, confidence
of fallback action set to core threshold (0.3).
2020-09-07 16:17:50 DEBUG    rasa.core.policies.ensemble  - Predicted next action using policy_0_Memo
izationPolicy
2020-09-07 16:17:50 DEBUG    rasa.core.processor  - Predicted next action 'action_initiation' with co
nfidence 1.00.
2020-09-07 16:17:50 DEBUG    rasa.core.actions.action  - Calling action endpoint to run action 'actio
n_initiation'.
2020-09-07 16:17:50 DEBUG    rasa.core.processor  - Action 'action_initiation' ended with events '[<r
asa.core.events.SlotSet object at 0x16727c8d0>, <rasa.core.events.SlotSet object at 0x1670607d0>]'.
2020-09-07 16:17:50 DEBUG    rasa.core.processor  - Current slot values:
        EP1_1: 3.5
        EP1_2: 4.0
```

EP2.1 checks to see if the user begins the conversation by saying ‘hello’ or some acceptable variation. For this project, acceptable variations of ‘hello’ are given as examples of the “*greet*” intent in the training data, so any user intent classified as “*greet*” will be considered sufficient for EP2.1. In the training data, the greet intent is always followed by “*action_utter_greet*”, and in the case that the user does not greet the chatbot first, the chatbot will execute “*action_utter_greet*”. This action is therefore an appropriate place for the EP2.1 logic. In order to check whether the action was called as a result of the user greeting the chatbot or a result of the user not initiating conversation, the action has to check to see if the most recent intent received was “*no_initiation*” or not. Since Rasa is a relatively new piece of software (only about 4 years old), the documentation in some areas is still lacking, and as a result it was not a straightforward process to figure out how to implement this kind of check. Since the tracker contains pretty much all the information about the chatbot, it was likely that the answer could be found somewhere there, but there is very little documentation on what the tracker can actually give you. A bit of trial and error was needed to determine that the most recent message received can be accessed in the tracker with the following way:

```
if tracker.latest_message['intent'].get('name') == greet:
```

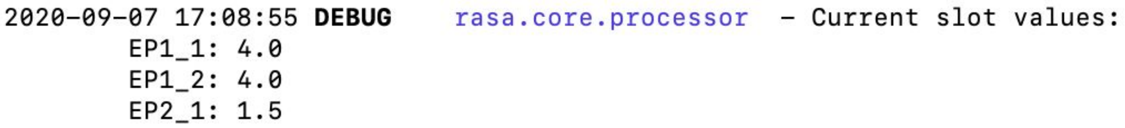
This ‘if’ statement allows the action to determine whether or not EP2.1 has been satisfied and so from there the slot value can simply be increased or decreased appropriately. For example:


You: what is your name

My name is John. Thank you for asking!

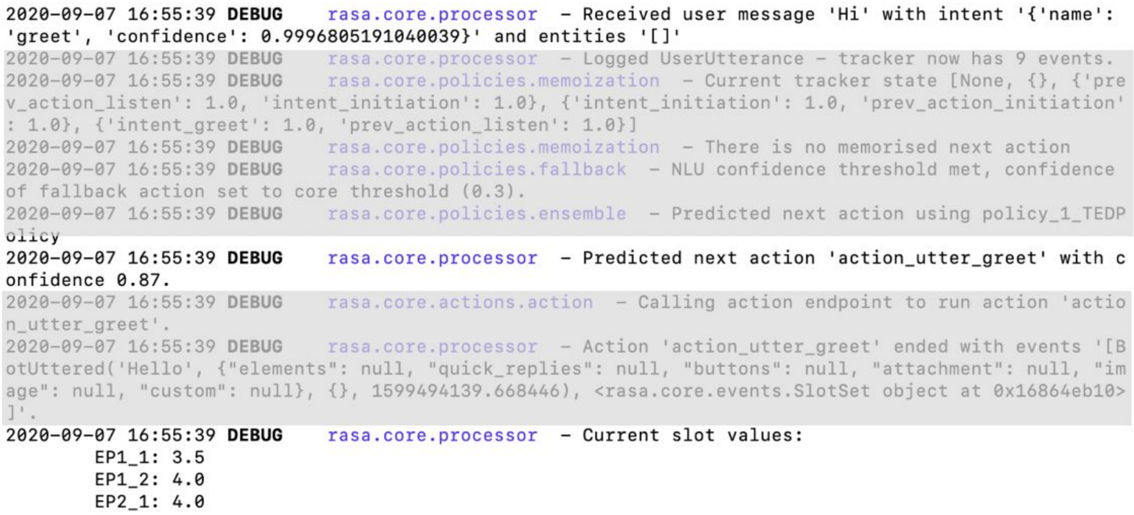
You: bye

See you soon!



```
2020-09-07 17:08:55 DEBUG    rasa.core.processor  - Current slot values:
        EP1_1: 4.0
        EP1_2: 4.0
        EP2_1: 1.5
```

If the user doesn't greet the chatbot appropriately, the EP2.1 slot value will be decreased. If the user does greet the chatbot, the value will be increased:

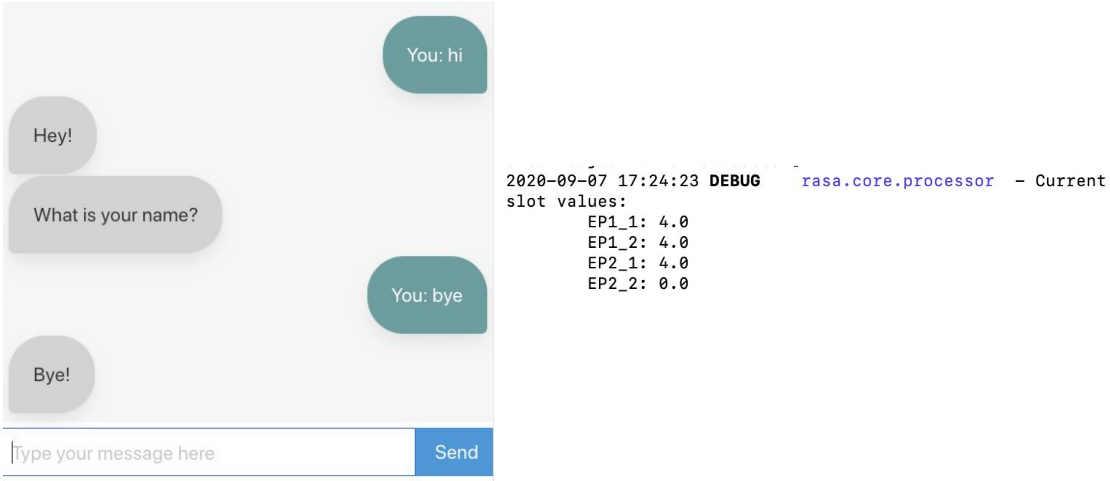


```
2020-09-07 16:55:39 DEBUG    rasa.core.processor  - Received user message 'Hi' with intent '{'name':
'greet', 'confidence': 0.9996805191040039}' and entities '[]'
2020-09-07 16:55:39 DEBUG    rasa.core.processor  - Logged UserUtterance - tracker now has 9 events.
2020-09-07 16:55:39 DEBUG    rasa.core.policies.memoization  - Current tracker state [None, {}, {'pre
v_action_listen': 1.0, 'intent_initiation': 1.0}, {'intent_initiation': 1.0, 'prev_action_initiation'
: 1.0}, {'intent_greet': 1.0, 'prev_action_listen': 1.0}]
2020-09-07 16:55:39 DEBUG    rasa.core.policies.memoization  - There is no memorised next action
2020-09-07 16:55:39 DEBUG    rasa.core.policies.fallback  - NLU confidence threshold met, confidence
of fallback action set to core threshold (0.3).
2020-09-07 16:55:39 DEBUG    rasa.core.policies.ensemble  - Predicted next action using policy_1_TEDP
olicy
2020-09-07 16:55:39 DEBUG    rasa.core.processor  - Predicted next action 'action_utter_greet' with c
onfidence 0.87.
2020-09-07 16:55:39 DEBUG    rasa.core.actions.action  - Calling action endpoint to run action 'actio
n_utter_greet'.
2020-09-07 16:55:39 DEBUG    rasa.core.processor  - Action 'action_utter_greet' ended with events '[B
otUttered('Hello', {"elements": null, "quick_replies": null, "buttons": null, "attachment": null, "im
age": null, "custom": null}, {}, 1599494139.668446), <rasa.core.events.SlotSet object at 0x16864eb10>
]'.
2020-09-07 16:55:39 DEBUG    rasa.core.processor  - Current slot values:
        EP1_1: 3.5
        EP1_2: 4.0
        EP2_1: 4.0
```

The next evaluation point is more of a binary, does the user do it or not, type EP than the previous EPs. EP2.2 deals with whether or not the user introduces themselves by giving their name in the first interaction with the chatbot. Since, information the chatbot holds on the user persists between web application sessions, the implementation of this requires tracking whether or not an interaction is the first interaction with the chatbot. This is achieved by storing a global variable in the *actions.py* file that is initially true and then set to false when the conversation is concluded. The EP2.2 slot value is initially set to 0, and then at the end of the first interaction the slot is set to 0 if the user didn't give their name, or 5 if they did.

```
# decrease EP2.2 if name hasn't been given in the first interaction
ep2_2 = tracker.get_slot("EP2_2")

global first_interaction
if first_interaction:
   name = tracker.get_slot("name")
   if name is None:
       ep2_2 = 0.0
   else:
       ep2_2 = 5.0

first_interaction = False
```

For example:



You: hi

Hey!

What is your name?

You: bye

Bye!

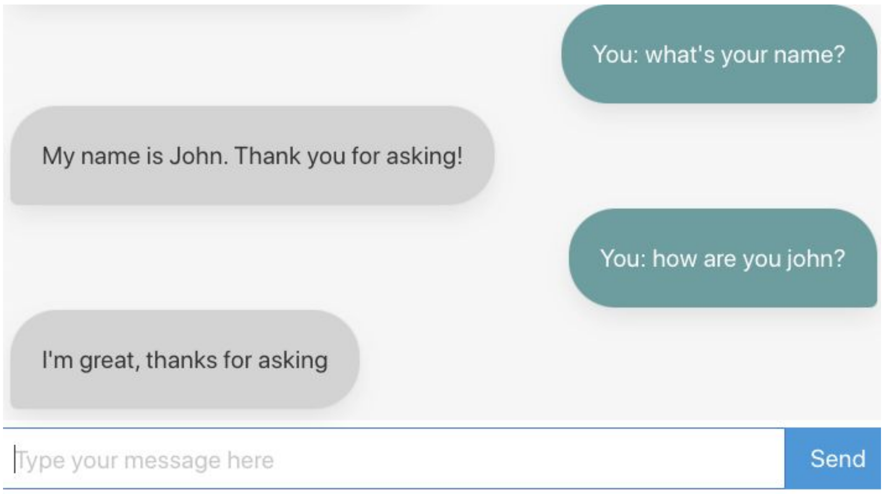
```
2020-09-07 17:24:23 DEBUG     rasa.core.processor  - Current
slot values:
        EP1_1: 4.0
        EP1_2: 4.0
        EP2_1: 4.0
        EP2_2: 0.0
```

When the conversation is concluded, the EP2.2 slot value is set to 0 as the user did not give their name properly

The final two EPs to be implemented in the MVP are in the area of 'Questions'. The first is EP3.1: Does the user ask questions. Unfortunately, Rasa does not include a straightforward feature to track if user messages are questions or not. So, in order to implement this EP, the response for each intent that is asking the chatbot a question needs to be changed from simply replying with the template response to a custom action that sends that response, but also adjusts the EP3.1 slot value appropriately. These actions can also set a global "question_asked" flag to true so that if no questions are asked during the interaction the system will know to decrease the EP3.1 value. These actions typically follow the naming convention of beginning with "*action_answer_*" as they are answering the question the user has asked. An example of one of these custom actions is "*action_answer_whatisyourname*" which is executed when the chatbot predicts the intent "*ask_whatisyourname*":

```
class ActionAnswerWhatIsYourName(Action):

   def name(self) -> Text:
       return "action_answer_whatisyourname"

   def run(self, dispatcher, tracker, domain):

dispatcher.utter_message(template="utter_answer_whatisyourname")

       global question_asked
       question_asked = True

       ep3_1 = tracker.get_slot("EP3_1")

       if ep3_1 < 5:
           ep3_1 += 0.5

       return [SlotSet("EP3_1", ep3_1)]
```

In the example conversation below, you'll see how the EP3.1 slot value is increased by just 0.5 from its initial value of 2.0 each time the user asks a question:

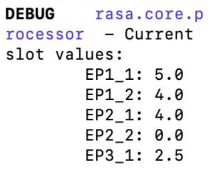


You: what's your name?

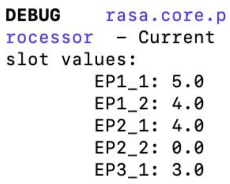
My name is John. Thank you for asking!

You: how are you john?

I'm great, thanks for asking

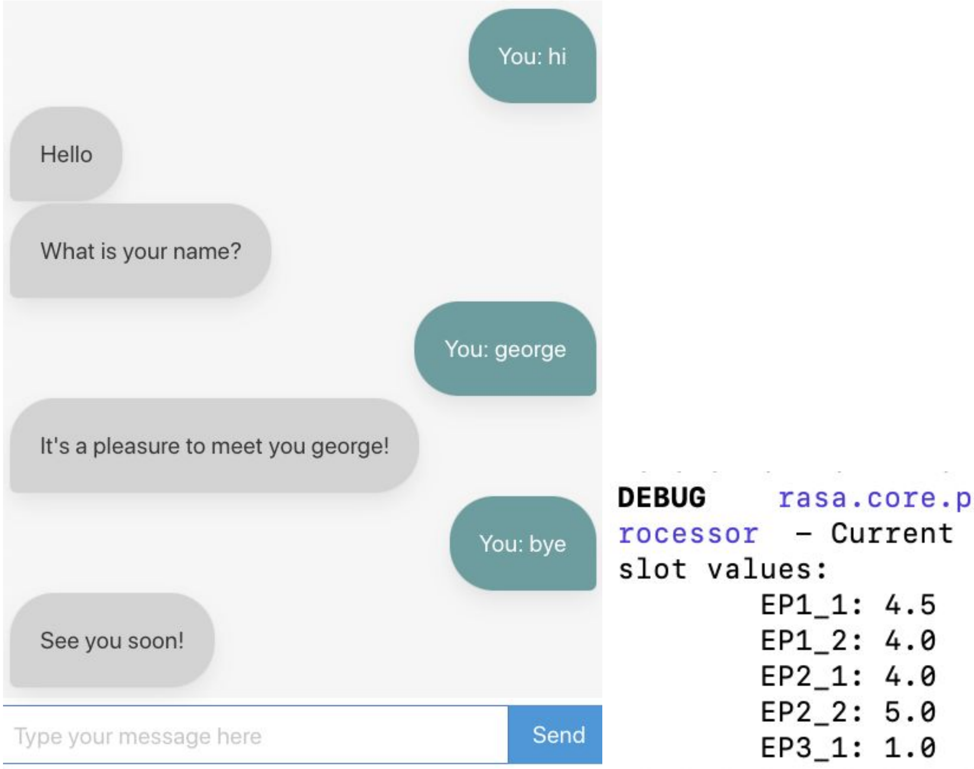


```
DEBUG    rasa.core.p
rocessor  - Current
slot values:
        EP1_1: 5.0
        EP1_2: 4.0
        EP2_1: 4.0
        EP2_2: 0.0
        EP3_1: 2.5
```



```
DEBUG    rasa.core.p
rocessor  - Current
slot values:
        EP1_1: 5.0
        EP1_2: 4.0
        EP2_1: 4.0
        EP2_2: 0.0
        EP3_1: 3.0
```

When an interaction is concluded without the user having asked any questions, the EP3.1 slot value is decreased by a more significant value of 1:



You: hi

Hello

What is your name?

You: george

It's a pleasure to meet you george!

You: bye

See you soon!

```
DEBUG     rasa.core.p
rocessor  - Current
slot values:
        EP1_1: 4.5
        EP1_2: 4.0
        EP2_1: 4.0
        EP2_2: 5.0
        EP3_1: 1.0
```

The EP3.1 slot value is decreased when the interaction concludes

The final EP to be implemented for the MVP is EP3.3: Does the user answer questions. As with EP3.1, the initial slot value for EP3.3 is set to 2.0 and there is no easy solution built into Rasa to accomplish this task, so it is necessary to think a bit harder to come up with a solution. A way of solving this problem is by turning each intent that could be an answer to a question into a custom action that checks to see if it is indeed a response to a question and increases the EP3.3 slot value if it is. This is not as straightforward as the similar sounding approach used to implement EP3.1 as it is not guaranteed that the intent is actually an answer to a question. Rasa does not feature a straightforward way of checking if the last chatbot message was a question and, again, the documentation in this area is lacking. Again the answer was found in the *tracker*, and after significant trial and error that involved printing the entire status of the tracker in order to understand how it worked, it was discovered that the '*get_last_event_for*' function was the solution. For most intents, a regex check of the last chatbot message, which is designated as a ‘bot’ object in the tracker, for a question mark was sufficient to determine if the user was answering a question.

```
if re.search("[?]", tracker.get_last_event_for('bot')['text']) is
not None:
```

The Python regex function ‘*search’* looks for a match anywhere in the string, and is appropriate because a question could appear in the message, not just at the end. This check was added to actions for most of the ‘answering’ intents followed by code to increase the EP3.3 slot value and then set a global “*question_answered*” flag to true. For example:

```
class ActionUserAffirm(Action):

   def name(self) -> Text:
       return "action_user_affirm"

   def run(self, dispatcher, tracker, domain):
       ep3_3 = tracker.get_slot("EP3_3")

       if re.search("[?]",
tracker.get_last_event_for('bot')['text']) is not None:
           if ep3_3 < 4.5:
               ep3_3 += 1.0
           elif ep3_3 < 5:
               ep3_3 += 0.5

           global question_answered
           question_answered = True

       dispatcher.utter_message(template="utter_happy")

       return [SlotSet("EP3_3", ep3_3)]
```

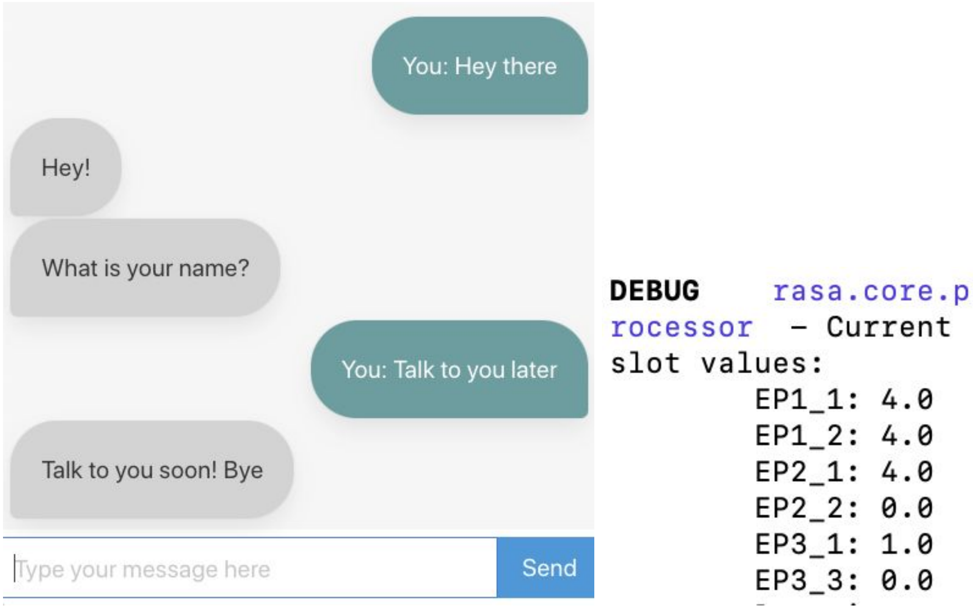
Whilst this was the most common approach to implementing EP3.3, there were two other types of intent that required slightly different approaches. The first of these is intents indicating a response to the chatbot asking “how are you”: ‘*action_user_mood_great*’ and ‘*action_user_mood_unhappy*’. Rather than just checking to see if a question was asked like the previous example, these actions check to see if the response if appropriate by checking to see if the previous chatbot question was one of the template responses for asking the user “how are you”:

```
if tracker.get_last_event_for('bot')['text'] in ['How are you?',
'How are you today?', 'How are you doing?']:
```

The third approach to implementing EP3.3 was for the instance where the user answers a question and then asks a question of their own in the same message, which would be classified as a different intent. In the MVP, the only instance where this could happen is in the action '*action_answer_howdoing*'. For example, the user could reply to the chatbot asking "how are you?" with the message "good thanks, and you?" which would be classified as the '*ask_howdoing*' intent. Another regex check was added to the action to account for this scenario:

```
if re.search("ok|good|great",
tracker.get_last_event_for('user')['text']) is not None:
```

Each of these actions sets the global variable '*question_answered*' if the question check is passed. If the interaction is concluded and the '*question_answered*' is still set to false, the EP3.3 slot value is decreased significantly:



You: Hey there

Hey!

What is your name?

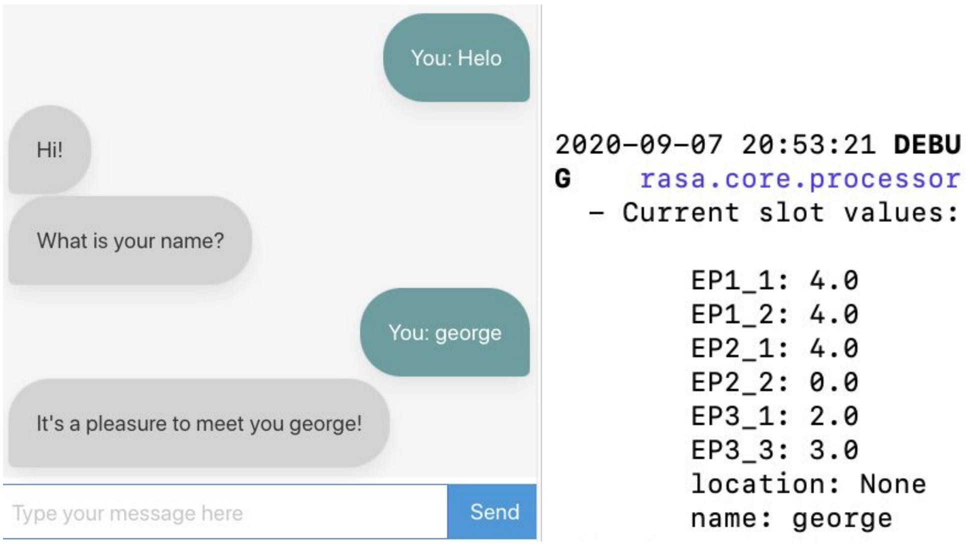
You: Talk to you later

Talk to you soon! Bye

```
DEBUG    rasa.core.p
rocessor  - Current
slot values:
        EP1_1: 4.0
        EP1_2: 4.0
        EP2_1: 4.0
        EP2_2: 0.0
        EP3_1: 1.0
        EP3_3: 0.0
```

The EP3.3 slot value is decreased when the interaction concludes

If questions are answered however, the EP3.3 slot value will be increased by 1 each time:



You: Helo

Hi!

What is your name?

You: george

It's a pleasure to meet you george!

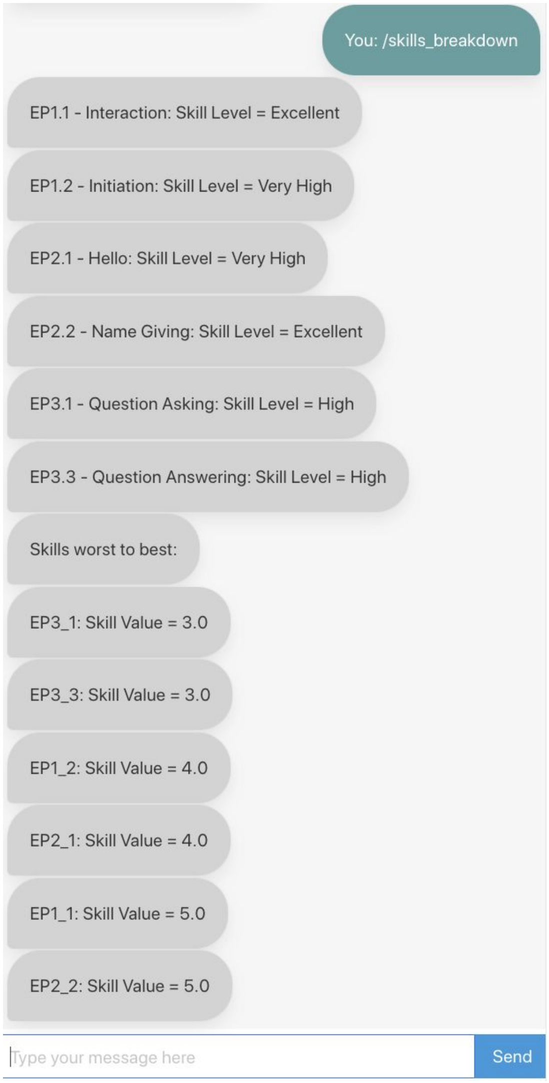
```
2020-09-07 20:53:21 DEBU
G     rasa.core.processor
  - Current slot values:

        EP1_1: 4.0
        EP1_2: 4.0
        EP2_1: 4.0
        EP2_2: 0.0
        EP3_1: 2.0
        EP3_3: 3.0
        location: None
        name: george
```

The EP3.3 slot value is increased when the user answers a question

With all six EPs required for the MVP now implemented, the only thing left to do is add a simple command to allow the user's therapist or parent to view the EP values, in lieu of a more formal 'dashboard' (F7). At any point in the conversation, but typically after the interaction has concluded, entering the message "/skills_breakdown" will trigger "*action_skills_breakdown*", as defined in *domain.yml*. This action will collect all of the current EP slot values and perform a little processing to present the values to the user:

```
eps = tracker.current_slot_values()
eps.pop('name', None)
eps.pop('location', None)

skill_values = ['Very Low', 'Low', 'Medium', 'High', 'Very High',
'Excellent']

ordered_skills = [f"{key}: Skill Value = {value}" for key, value in
sorted(eps.items(), key=lambda item: item[1])]
ordered_skills = '\n\n'.join(ordered_skills)

dispatcher.utter_message(f"EP1.1 - Interaction: Skill Level =
{skill_values[math.floor(eps.get('EP1_1'))]}\n\n"
                        f"EP1.2 - Initiation: Skill Level =
{skill_values[math.floor(eps.get('EP1_2'))]}\n\n"
                        f"EP2.1 - Hello: Skill Level =
{skill_values[math.floor(eps.get('EP2_1'))]}\n\n"
                        f"EP2.2 - Name Giving: Skill Level =
{skill_values[math.floor(eps.get('EP2_2'))]}\n\n"
                        f"EP3.1 - Question Asking: Skill Level =
{skill_values[math.floor(eps.get('EP3_1'))]}\n\n"
                        f"EP3.3 - Question Answering: Skill Level = 
{skill_values[math.floor(eps.get('EP3_3'))]}\n\n"
                        f"Skills worst to best: 
\n\n{ordered_skills}")
```

The action prints out a list of all EPs in numerical order with the skill value expressed as a word from the ‘*skills_values*’ list by taking the floor of the numerical value and using that as an index. The action also prints a list of all EPs with their numerical value ordered from worst to best so the supervisor can easily see which skills need the most work. EPs are stored as key-value pairs, so are sorted on the value using a python lambda function. The output of the function after a very short conversation looks as follows:



You: /skills_breakdown

EP1.1 - Interaction: Skill Level = Excellent

EP1.2 - Initiation: Skill Level = Very High

EP2.1 - Hello: Skill Level = Very High

EP2.2 - Name Giving: Skill Level = Excellent

EP3.1 - Question Asking: Skill Level = High

EP3.3 - Question Answering: Skill Level = High

Skills worst to best:

EP3_1: Skill Value = 3.0

EP3_3: Skill Value = 3.0

EP1_2: Skill Value = 4.0

EP2_1: Skill Value = 4.0

EP1_1: Skill Value = 5.0

EP2_2: Skill Value = 5.0

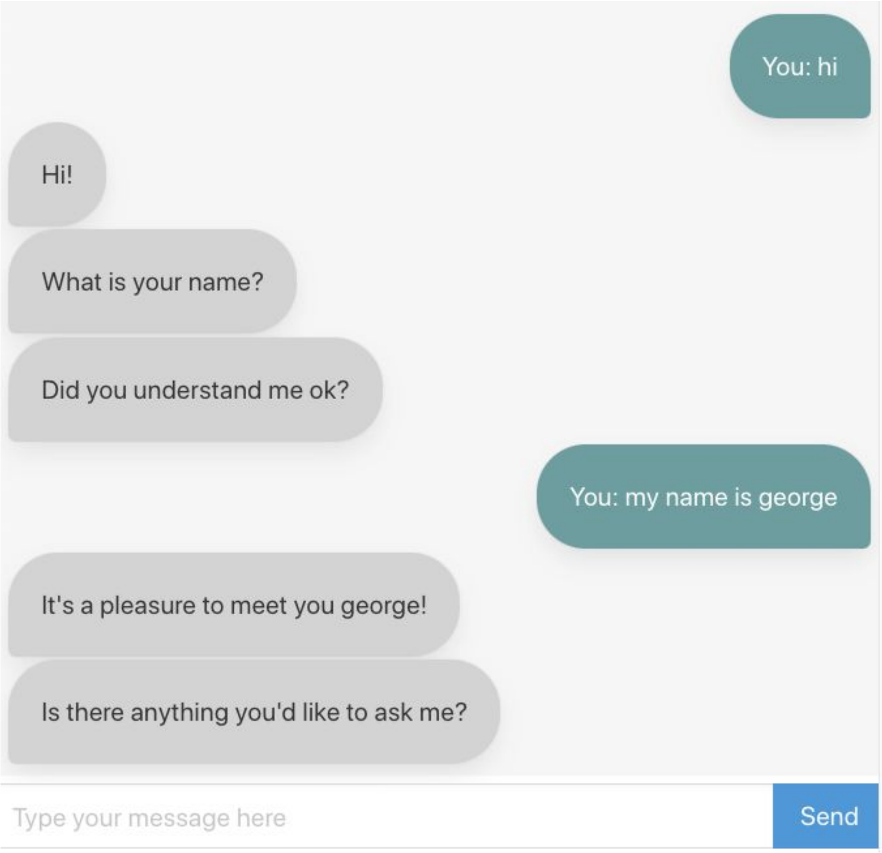
With the MVP now complete, the system is capable of being used by an autistic child to practice interacting with an alternative communication partner, and the therapist or parent is able to review how well the child performed the different skills.

5.7 Second Iteration

Now that the chatbot is able to evaluate the user's ability with the different social skills, it can begin to tailor the conversation to help improve them (F4). This will be achieved in two main parts: 1) The chatbot will provide prompts, reinforcement and examples in general conversation (F4.1) , and 2) the chatbot will play structured 'games' to target specific social skills (F4.2). Given the very fundamental nature of the EPs implemented in the MVP, there are only a few things that can be done to address them in general conversation. For EP1.1, the chatbot can send a follow up message to prompt the user if they don't interact. This simply requires adding some kind of message to be sent to the user in "*action_no_interaction*":

```
if re.search("[?]", tracker.get_last_event_for('bot')['text']) is
not None:
   if ep3_3 > 0:
       ep3_3 -= 0.5
   dispatcher.utter_message("Did you understand me ok?")
elif not question_asked:
   dispatcher.utter_message(template="utter_doyouhaveaquestion")
else:
   dispatcher.utter_message("Is there anything else you'd like to
ask me or talk about?")
```

This action also works as a prompt for EP3.1 as it encourages the user to ask a question if they haven't already, and as a prompt for EP3.3 as it encourages the user to respond if they didn't answer a question. For example:



You: hi

Hi!

What is your name?

Did you understand me ok?

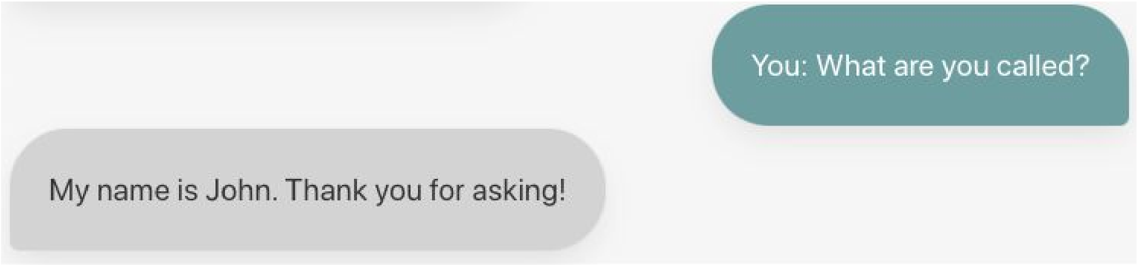
You: my name is george

It's a pleasure to meet you george!

Is there anything you'd like to ask me?

When the chatbot receives the '*/no_interaction'* intent, it prompts the user to interact

The chatbot initiating conversation if the user does not and asking questions of its own serves as an example for the user to encourage them to do the same in the future. Positive reinforcement was also added to the chatbot's responses to questions asked by the user:



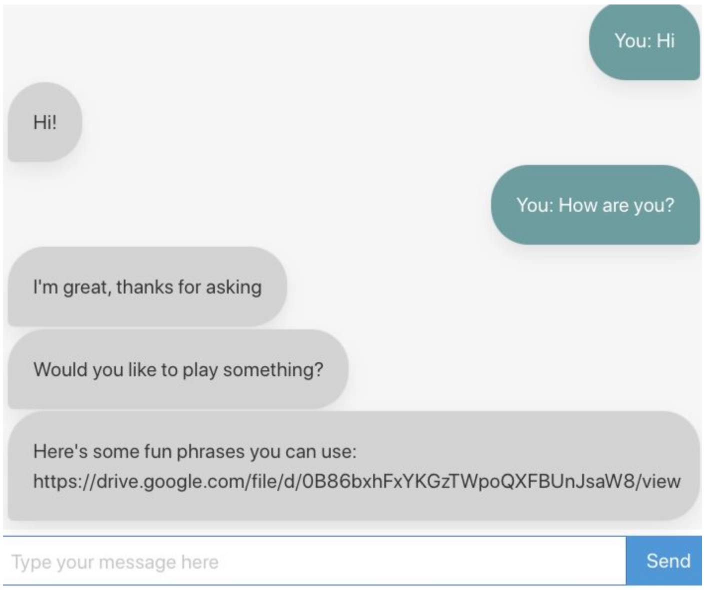
You: What are you called?

My name is John. Thank you for asking!

The other way that these skills will be improved is through the second part of the Second Iteration: the implementation of the Social Scripts games. The minimum Social Scripts games implementation would choose from a selection of online Social Scripts resources based on the scores the system has for each EP. Further stages of development would have the games built into the chatbot itself, but a minimum implementation in line with the project management methodologies described in section 4.1 would simply send the user a link to an existing online Social Scripts resource. For this iteration, the chatbot will suggest playing one of these games when the user asks what the chatbot can do, or when the user doesn't interact. A global function was added to *actions.py* that takes the current tracker as an input and returns a link to the appropriate game:

```
def pick_game(tracker):
   eps = tracker.current_slot_values()

   game = "Here's a fun story you can follow along with:
https://starautismsupport.com/sites/default/files/STAR%20-%20Social%
20Script%20-%20Sharing%20and%20Turn%20Taking_08.29.18.pdf"

   if eps.get('EP1_1') < 3 or eps.get('EP1_2') < 3:
       game = "Here's some fun phrases you can use:
https://drive.google.com/file/d/0B86bxhFxYKGzWjlSdWhzVnBpSWc/view"
   elif eps.get('EP2_1') < 3 or eps.get('EP2_2') < 3:
       game = "Here's some fun phrases you can use:
https://drive.google.com/file/d/0B86bxhFxYKGzTWpoQXFBUnJsaW8/view"

   return game
```

EPs are considered in order of the most fundamental skills to the less fundamental skills (in line with LD4). If the EPs in the 'Interaction' area are low, the chatbot will suggest a game that provides some useful phrases the user could use when they're not sure what to say or need some help [27]. If 'Interaction' is ok, but the user struggles with 'Greetings', the chatbot will suggest a game that gives some example phrases to use at the start of a conversation [28]. Otherwise, the chatbot will suggest a more general Social Script game to help the user develop their understanding of friendships and turn-taking [29].



You: Hi

Hi!

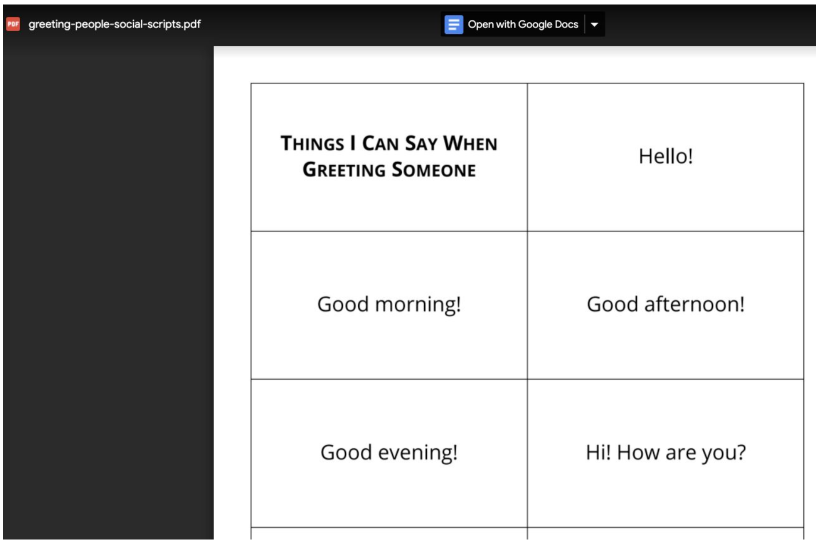
You: How are you?

I'm great, thanks for asking

Would you like to play something?

Here's some fun phrases you can use: https://drive.google.com/file/d/0B86bxhFxYKGzTWpoQXFBUnJsaW8/view

```
2020-09-09 19:25:01 DEBUG
   rasa.core.processor  -
Current slot values:
        EP1_1: 2.0
        EP1_2: 4.0
        EP2_1: 4.0
        EP2_2: 0.0
        EP3_1: 2.5
        EP3_3: 2.0
        location: None
        name: None
```



THINGS I CAN SAY WHEN GREETING SOMEONE	Hello!
Good morning!	Good afternoon!
Good evening!	Hi! How are you?

When the users ‘Greeting’ EPs are low, the chatbot suggests a game to help

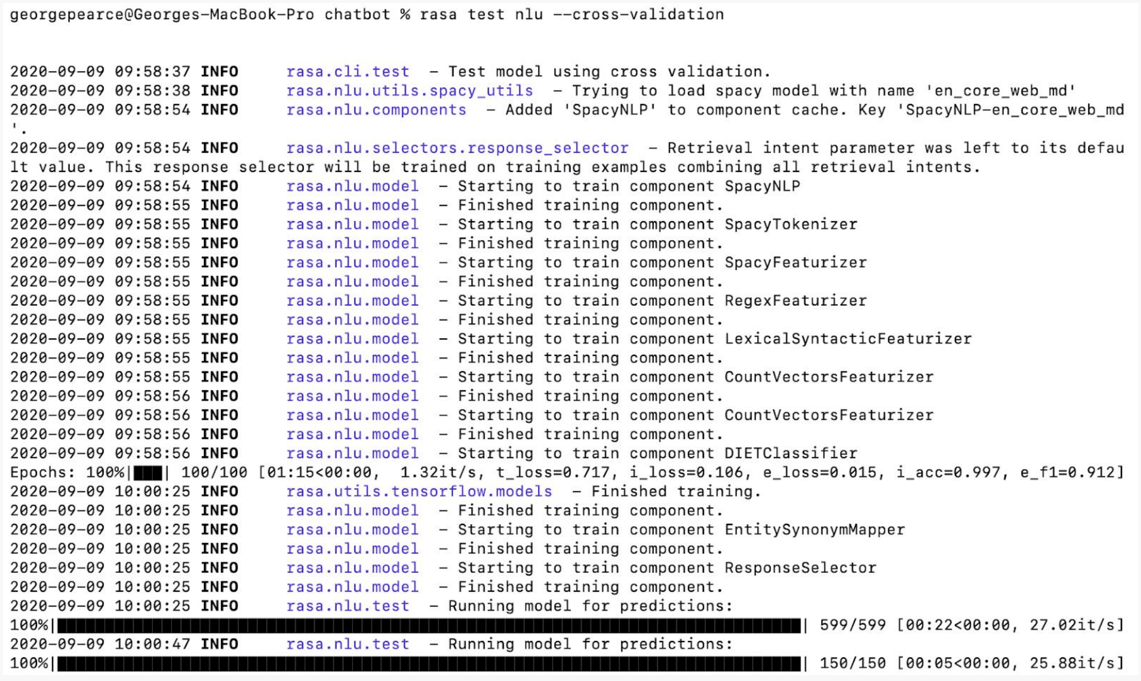
Chapter 6

Evaluation

6.1 Testing

A common issue that can arise in any machine learning model is an inability to generalise well to new data, not in the training data. A cross validation test can be used to see how well the model generalises. In such a test, the training data is split into 5 groups, one group is used as the testing data, whilst the other four are used as the training data. A model is trained on the training data and then applied to the testing data to see how well it performs. This process is then repeated four more times so each group is used as the testing data once. The overall performance of the model is then calculated by combining the individual scores.

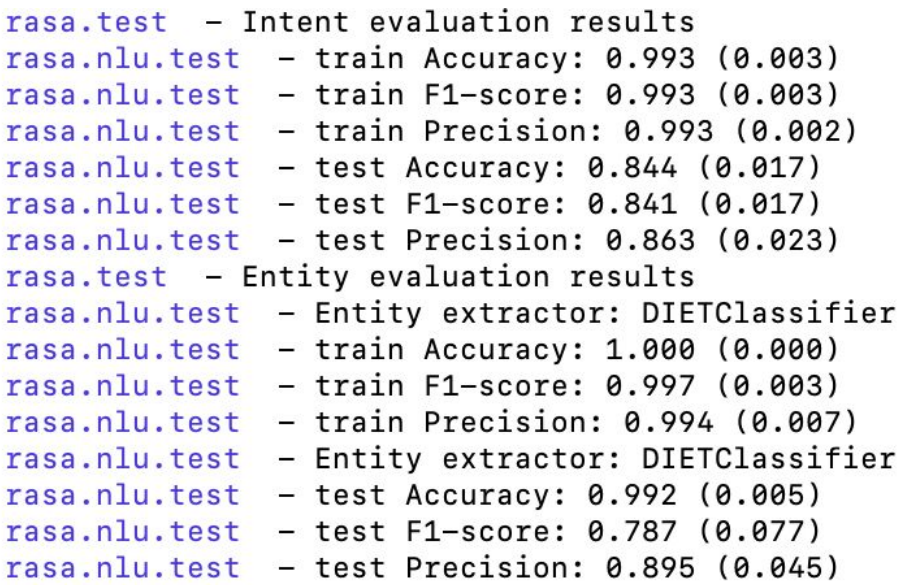
```
rasa test nlu --cross-validation
```



```
georgepearce@Georges-MacBook-Pro chatbot % rasa test nlu --cross-validation


2020-09-09 09:58:37 INFO     rasa.cli.test  - Test model using cross validation.
2020-09-09 09:58:38 INFO     rasa.nlu.utils.spacy_utils  - Trying to load spacy model with name 'en_core_web_md'
2020-09-09 09:58:54 INFO     rasa.nlu.components  - Added 'SpacyNLP' to component cache. Key 'SpacyNLP-en_core_web_md
'.
2020-09-09 09:58:54 INFO     rasa.nlu.selectors.response_selector  - Retrieval intent parameter was left to its defau
lt value. This response selector will be trained on training examples combining all retrieval intents.
2020-09-09 09:58:54 INFO     rasa.nlu.model  - Starting to train component SpacyNLP
2020-09-09 09:58:55 INFO     rasa.nlu.model  - Finished training component.
2020-09-09 09:58:55 INFO     rasa.nlu.model  - Starting to train component SpacyTokenizer
2020-09-09 09:58:55 INFO     rasa.nlu.model  - Finished training component.
2020-09-09 09:58:55 INFO     rasa.nlu.model  - Starting to train component SpacyFeaturizer
2020-09-09 09:58:55 INFO     rasa.nlu.model  - Finished training component.
2020-09-09 09:58:55 INFO     rasa.nlu.model  - Starting to train component RegexFeaturizer
2020-09-09 09:58:55 INFO     rasa.nlu.model  - Finished training component.
2020-09-09 09:58:55 INFO     rasa.nlu.model  - Starting to train component LexicalSyntacticFeaturizer
2020-09-09 09:58:55 INFO     rasa.nlu.model  - Finished training component.
2020-09-09 09:58:55 INFO     rasa.nlu.model  - Starting to train component CountVectorsFeaturizer
2020-09-09 09:58:56 INFO     rasa.nlu.model  - Finished training component.
2020-09-09 09:58:56 INFO     rasa.nlu.model  - Starting to train component CountVectorsFeaturizer
2020-09-09 09:58:56 INFO     rasa.nlu.model  - Finished training component.
2020-09-09 09:58:56 INFO     rasa.nlu.model  - Starting to train component DIETClassifier
Epochs: 100%|███| 100/100 [01:15<00:00,  1.32it/s, t_loss=0.717, i_loss=0.106, e_loss=0.015, i_acc=0.997, e_f1=0.912]
2020-09-09 10:00:25 INFO     rasa.utils.tensorflow.models  - Finished training.
2020-09-09 10:00:25 INFO     rasa.nlu.model  - Finished training component.
2020-09-09 10:00:25 INFO     rasa.nlu.model  - Starting to train component EntitySynonymMapper
2020-09-09 10:00:25 INFO     rasa.nlu.model  - Finished training component.
2020-09-09 10:00:25 INFO     rasa.nlu.model  - Starting to train component ResponseSelector
2020-09-09 10:00:25 INFO     rasa.nlu.model  - Finished training component.
2020-09-09 10:00:25 INFO     rasa.nlu.test  - Running model for predictions:
100%|████████████████████████████████████████████████████████████████████████████| 599/599 [00:22<00:00, 27.02it/s]
2020-09-09 10:00:47 INFO     rasa.nlu.test  - Running model for predictions:
100%|████████████████████████████████████████████████████████████████████████████| 150/150 [00:05<00:00, 25.88it/s]
```

Figure 11: Terminal output of cross validation in progress

This evaluation produces overall results for intent and evaluation prediction:



```
rasa.test  - Intent evaluation results
rasa.nlu.test  - train Accuracy: 0.993 (0.003)
rasa.nlu.test  - train F1-score: 0.993 (0.003)
rasa.nlu.test  - train Precision: 0.993 (0.002)
rasa.nlu.test  - test Accuracy: 0.844 (0.017)
rasa.nlu.test  - test F1-score: 0.841 (0.017)
rasa.nlu.test  - test Precision: 0.863 (0.023)
rasa.test  - Entity evaluation results
rasa.nlu.test  - Entity extractor: DIETClassifier
rasa.nlu.test  - train Accuracy: 1.000 (0.000)
rasa.nlu.test  - train F1-score: 0.997 (0.003)
rasa.nlu.test  - train Precision: 0.994 (0.007)
rasa.nlu.test  - Entity extractor: DIETClassifier
rasa.nlu.test  - test Accuracy: 0.992 (0.005)
rasa.nlu.test  - test F1-score: 0.787 (0.077)
rasa.nlu.test  - test Precision: 0.895 (0.045)
```

The F1-score takes into account both the precision value and the recall value, where precision is equal to the number of true positive predictions divided by the total number of true positives and false positives, and the recall is equal to the number of true positive predictions divided by the number of true positives and false negatives. Whilst the scores for the testing data are noticeably lower than the training data for both intents and entities, they are still acceptably high scores for a machine learning model of this nature. With further fine-tuning of the configuration of the chatbot and the parameters of the neural networks used to train the model these scores could be increased, meaning the chatbot could generalise better.

The cross validation test also determines these scores for each individual intent and named entity. For example:

```
"inform": {
 "precision": 1.0,
 "recall": 0.88,
 "f1-score": 0.9361702127659575,
 "support": 50,
 "confused_with": {
   "mood_great": 2,
   "affirm": 1
 }
```

```
"name": {
 "precision": 0.9310344827586207,
 "recall": 0.8181818181818182,
 "f1-score": 0.8709677419354839,
 "support": 33
}
```

It also produces an Intent Prediction Confidence Distribution and an Intent Confusion Matrix to help visualise how the model performs and where it is going wrong:

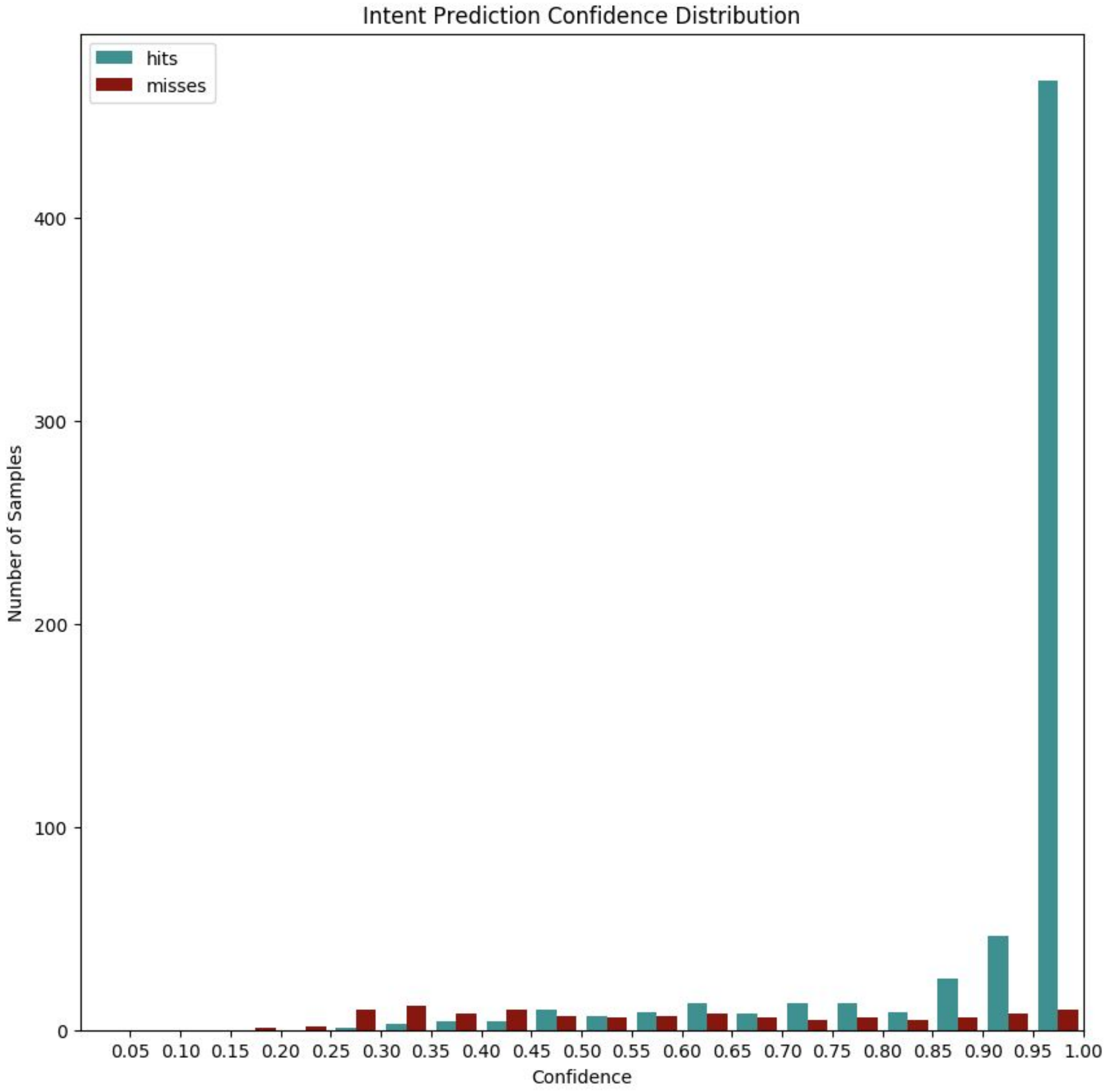
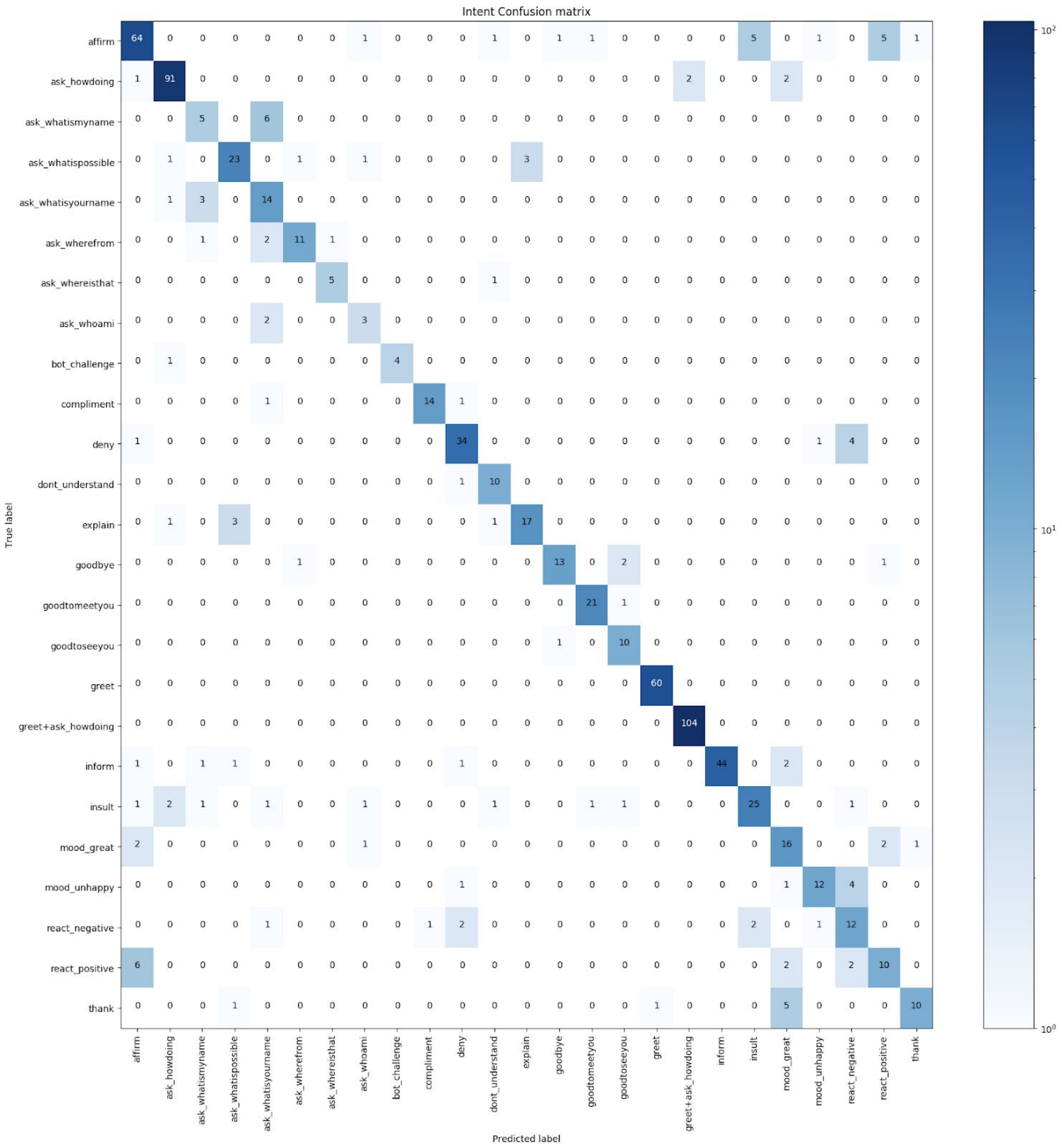


Figure 12: Intent Prediction Confidence Distribution histogram



	affirm	ask_howdoing	ask_whatismyname	ask_whatispossible	ask_whatisyourname	ask_wherefrom	ask_whereisthat	ask_whoami	bot_challenge	compliment	deny	dont_understand	explain	goodbye	goodtomeetyou	goodtoseeyou	greet	greet+ask_howdoing	inform	insult	mood_great	mood_unhappy	react_negative	react_positive	thank
affirm	64	0	0	0	0	0	0	1	0	0	0	1	0	1	1	0	0	0	0	5	0	1	0	5	1
ask_howdoing	1	91	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	2	0	0	0	0
ask_whatismyname	0	0	5	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ask_whatispossible	0	1	0	23	0	1	0	1	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0
ask_whatisyourname	0	1	3	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ask_wherefrom	0	0	1	0	2	11	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ask_whereisthat	0	0	0	0	0	0	5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
ask_whoami	0	0	0	0	2	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bot_challenge	0	1	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
compliment	0	0	0	0	1	0	0	0	0	14	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
deny	1	0	0	0	0	0	0	0	0	0	34	0	0	0	0	0	0	0	0	0	0	1	4	0	0
dont_understand	0	0	0	0	0	0	0	0	0	0	1	10	0	0	0	0	0	0	0	0	0	0	0	0	0
explain	0	1	0	3	0	0	0	0	0	0	0	1	17	0	0	0	0	0	0	0	0	0	0	0	0
goodbye	0	0	0	0	0	1	0	0	0	0	0	0	0	13	0	2	0	0	0	0	0	0	0	1	0
goodtomeetyou	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21	1	0	0	0	0	0	0	0	0	0
goodtoseeyou	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	10	0	0	0	0	0	0	0	0	0
greet	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0
greet+ask_howdoing	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	104	0	0	0	0	0	0	0
inform	1	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	44	0	2	0	0	0	0
insult	1	2	1	0	1	0	0	1	0	0	0	1	0	0	1	1	0	0	0	25	0	0	1	0	0
mood_great	2	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0	2	1
mood_unhappy	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	12	4	0	0
react_negative	0	0	0	0	1	0	0	0	0	1	2	0	0	0	0	0	0	0	0	2	0	1	12	0	0
react_positive	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	2	10	0
thank	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	5	0	0	0	10

Figure 13: Intent prediction confusion matrix

The confusion matrix indicates that the vast majority of intents are classified correctly, but there are a number intent-pairs that are confused occasionally. The most common of these intent-pairs is '*affirm*' and *'react_positive*', with the former being wrongly classified as the latter 5 times, and the former wrongly classified as the latter 6 times. An example of this can be found in the generated *intent_errors.json* file:

```
{
 "text": "that is great",
 "intent": "affirm",
 "intent_prediction": {
   "name": "react_positive",
   "confidence": 0.9206823706626892
 }
},
```

This is likely the most commonly confused pair due to the usually similarly positive nature of the language. Adding more varied training data could help to reduce this confusion, and confusion between other pairs of intents. Whilst this cross validation method does give an indication of how well the model might generalise, since it doesn't use truly unseen data it may not be a completely accurate representation of the well the model could handle more predictable user input. Setting aside truly separate, unseen data solely for the use as testing data would help with this.

6.2 Requirements Evaluation

The following table gives a breakdown of the implementation of the requirements outlined in section 3.2, indicating whether the requirement was successfully implemented, partially implemented, or not implemented:

Functional Requirements		
Requirement	**MSCW**	**Implemented?**
F1	MUST	YES
F1.1	MUST	YES
F1.2	MUST	YES
F1.3	MUST	YES
F2	MUST	YES
F3	MUST	YES
F4	SHOULD	YES
F4.1	SHOULD	YES
F4.2	SHOULD	YES

F4.3	SHOULD	YES
F5	SHOULD	NO
F6	COULD	NO
F7	COULD	NO
F8	WON’T	NO
Non-Functional Requirements		
Requirement	**MSCW**	**Implemented?**
NF1	COULD	YES
NF2	WON’T	NO

Table 6.2

As you can see from the table above, all of the requirements classed as ‘Must’ requirements have been successfully implemented. Therefore, in line with the criteria for success outlined in section 3.3, the project is a success. Additionally, most ‘Should’ requirements have also been implemented. It is therefore reasonable to consider the project a significant success. A working solution that satisfies the minimum requirements and some other key requirements has been delivered.

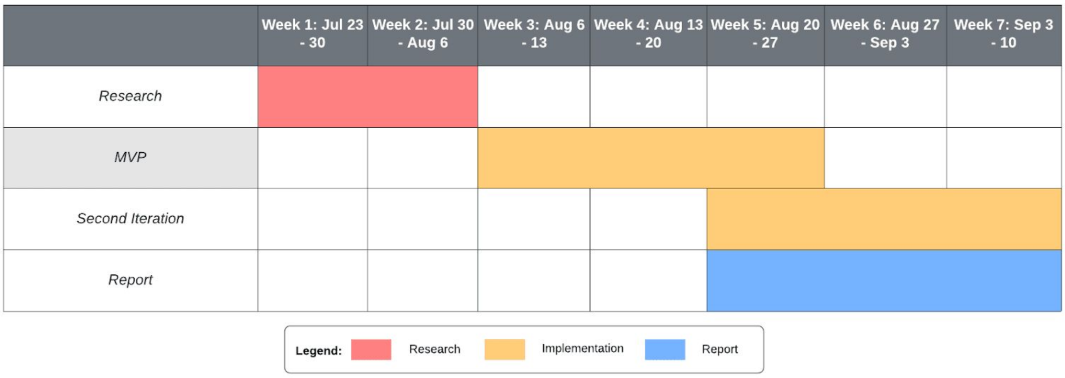
The first couple of requirements proved fairly straightforward to implement once sufficient research had been done into how to use the chosen tools. F3 however proved more challenging to implement as it required more outside-the-box type thinking to achieve something that the tools weren’t strictly designed to achieve. It should be noted that this is not a criticism of the selection of the tools. Since this is a task that has not really been attempted before, there are no tools designed specifically for this task. Given that a working system was delivered despite this obstacle means that the correct tools for the job were indeed selected. Despite these challenges, F3 was implemented successfully.

The primary tool used, Rasa, was largely appropriate for the task, despite the slight limitations described above. It was slightly less appropriate for the implementation of games however, although given the more novel nature of the task it would be unlikely that a more suitable alternative could be found. With more experimentation with the less well documented features of Rasa, a better approach could potentially have been found. Alternatively, it might have been better to approach the games through the Django web application, although this would have sacrificed the more flexible nature of the system's deployment. Fundamentally however, this fine-tuning of the effectiveness of these games is out of the scope of this project, so it can be concluded that Rasa was a very good choice. Django proved to be a wholly successful choice for the implementation of the web application, providing a robust platform for the UI to be implemented relatively quickly, allowing more time to focus on the more challenging implementation of the Rasa chatbot.

With these requirements satisfied as outlined in the 'Design' section, the system successfully addresses many of the suggestions and principles highlighted in the 'Research' section. The design of the system implements Richard Hastings suggestion (RH1, RH2, RH3) of using the chatbot to improve outcomes in an existing intervention. Louise Denne's advice regarding the teaching of the skills (LD4), and other teaching principles researched, are incorporated in F3 and F4 as described in the design section. Kylie Gray's suggestion of deliberately considering the potential end-user (KG3) was also addressed in the implementation of the proposed design.

6.3 Project Management Evaluation

The methodology used was successful as it ensured the must requirements were executed first and were delivered in a complete, usable system. Despite this overall success, there were times when my natural instinct as a perfectionist got the better of me and slowed down development as I lost sight of what was strictly necessary for a 'minimum' implementation. These instances were only minor however and did not drastically hinder progress of the project. Additionally, the minimum implementation of the final EP in F3 took a little more time than expected as the challenge of determining whether the previous chatbot message was a question was overcome. The solution was found and the EP was successfully implemented, as described in section 5.6, but it did push the work on the MVP past the anticipated deadline of week 4 into week 5. Overall, the proposed timeline of work was successfully adhered to for the first few weeks, but drifted from towards the end of the project as the work necessary to write the report took more time to complete than expected. This limited the amount of implementation work that could be carried out during the final weeks, meaning the third, fourth and fifth iterations of development were not worked on as expected. The below gantt chart shows the actual timeline of work:



	Week 1: Jul 23 - 30	Week 2: Jul 30 - Aug 6	Week 3: Aug 6 - 13	Week 4: Aug 13 - 20	Week 5: Aug 20 - 27	Week 6: Aug 27 - Sep 3	Week 7: Sep 3 - 10
Research	🟥	🟥					
MVP			🟨	🟨	🟨		
Second Iteration					🟨	🟨	🟨
Report					🟦	🟦	🟦

Legend: 🟥 Research 🟨 Implementation 🟦 Report

Figure 14: Gantt chart of actual work done

The failure to reach the later iterations by no means makes the project overall or the planning itself a failure. The iterations not reached were also merely ‘Could’ iterations, with the exception of the some ‘Should’ EPs and F5: Speech Recognition, and so there was no solid expectation in the planning of the project that they would be achieved. Most importantly, the agile methodologies employed ensured that a minimum working solution was delivered as soon as possible, minimising the risk of overrunning work hindering the success of the project.

Chapter 7

Conclusion

Overall, the project was a significant success. The basic goal set out of the start of the project, to deliver a system that could be used as part of a Speech Therapy or ABA intervention to provide an autistic child with an AI conversation partner that could identify how well the child performs in different areas and then tailor future interactions to address them, has been achieved. Whilst more work could be done to increase the potential usefulness of the chatbot in a real-world intervention, the end result of this project is indeed a chatbot that could be used to help autistic children develop their social skills. With the chatbot developed as a standalone part of the project, it has the potential to be accessed through many other mediums in addition to the web application developed in this project. These achievements take very important and meaningful first steps towards achieving the potentially revolutionary intervention proposed in the introduction and design sections of this report.

7.1 Practical Limitations

Comparing the system delivered at the end of this project to the potential solution described as “both genuinely effective and incredibly accessible” in section 1.2, the system falls short in some areas. Whilst the chatbot could be used by some autistic children, the assumptions made about the user limit the potential user base to those that possess the necessary skills. Incorporating predefined, selectable options as well as a speech interface in addition to the text interface implemented would open the chatbot up to use with a much larger user base. The chatbot does fit the description of potentially being “genuinely effective” in helping autistic children improve social skills, but only for a relatively small number of social skills. The addition of more social skills, including those described in table 3.1.3 such as emotion recognition (EP1.7), turn-taking (EP1.9), other ‘Greeting’ skills, ‘Manners’ and ‘Figurative Language’ in particular, would help make the chatbot even more useful in a real-world intervention. Some of the social skills that are less appropriate for targeting with a chatbot such as understanding body language could also potentially be addressed by pairing the chatbot with some advanced computer vision technology in order to truly push the boundaries of how technology can be used to help children with autism. Despite these practical limitations, the solution delivered remains a working system and the potential improvements discussed above and in the next section would simply serve to increase the usefulness of an already successful chatbot.

7.2 Future Work & Applications

The obvious next steps for development would be the completion of the remaining iterations proposed in section 4.3. With that functionality the chatbot would be an incredibly powerful tool capable of helping a large percentage of the young autistic population improve their social skills, and other aspects of their lives by extension. The implementation of speech recognition and synthesis (F5 and F6) in particular have the potential to greatly increase the effectiveness of the chatbot for practical use. The chatbot acts as a solid platform for more and more functionality to be added. In addition to the EPs included in this project's Social Interaction Breakdown, an almost endless list of social skills could be incorporated over time to bring the capabilities of the chatbot closer and closer to that of a trained therapist.

The chatbot could also be used through other mediums such as a mobile application or even a robot. This flexibility of deployment has the potential to increase the effectiveness of the chatbot as a social skills intervention. It increases accessibility as different children may be more comfortable using different technologies or mediums to interact with the chatbot; some may prefer a web application whilst some may prefer talking to a robot equipped with the chatbot. It will also help the child generalise the skills they learn as they will have the chance to practice with a wider range of conversation partners.

References

1. Beardon, Luke. *Asperger syndrome and perceived offending conduct: a qualitative study*. Diss. Sheffield Hallam University, 2008.
2. Beardon, Luke. "3 Social Relationships." *Autism and Asperger Syndrome in Adults*. N.p.: Sheldon, 2017.
3. Mintz, Joseph, et al. "Key factors mediating the use of a mobile technology tool designed to develop social and life skills in children with Autistic Spectrum Disorders." *Computers & Education* 58.1 (2012): 53-62.
4. Wood, Luke J., Abolfazl Zaraki, Ben Robins, and Kerstin Dautenhahn. "Developing Kaspar: A Humanoid Robot for Children with Autism." *International Journal of Social Robotics* (2019).
5. Daniels, Jena, Jessey N. Schwartz, Catalin Voss, Nick Haber, Azar Fazel, Aaron Kline, Peter Washington, Carl Feinstein, Terry Winograd, and Dennis P. Wall. "Exploratory Study Examining the At-home Feasibility of a Wearable Tool for Social-affective Learning in Children with Autism." *Npj Digital Medicine* 1.1 (2018).
6. "Autism - What Is It?" *Autism Support - Leading UK Charity - National Autistic Society*. N.p., n.d. Web. 04 June 2020. <https://www.autism.org.uk/about/what-is/asd.aspx>.
7. "2016-10-27 - Government Must Tackle the Autism Employment Gap." *Autism Support - Leading UK Charity - National Autistic Society*. N.p., n.d. Web. 04 June 2020. <https://www.autism.org.uk/get-involved/media-centre/news/2016-10-27-employment-gap.aspx>.
8. "I Exist - the Message from Adults with Autism." *I Exist - the Message from Adults with Autism | Network Autism*. N.p., n.d. Web. 04 June 2020. <https://network.autism.org.uk/file/i-exist-message-adults-autism>.
9. Hourcade, Juan Pablo, Natasha E. Bullock-Rest, and Thomas E. Hansen. "Multitouch Tablet Applications and Activities to Enhance the Social Skills of Children with Autism Spectrum Disorders." *Personal and Ubiquitous Computing* 16.2 (2011): 157-68. Print.
10. Stanton, Cady M., Peter H. Kahn Jr., Rachel L. Severson, Jolina H. Ruckert, and Brian T. Gill. "Robotic Animals Might Aid in the Social Development of Children with Autism." *Proceedings of the 3rd International Conference on Human Robot Interaction - HRI '08* (2008): n. pag. Print.
11. Grossard, Charline, Ouriel Grynspan, Sylvie Serret, Anne-Lise Jouen, Kevin Bailly, and David Cohen. "Serious Games to Teach Social Interactions and

Emotions to Individuals with Autism Spectrum Disorders (ASD)." *Computers & Education* 113 (2017): 195-211. Print.

12. Boccanfuso, Laura, Sarah Scarborough, Ruth K. Abramson, Alicia V. Hall, Harry H. Wright, and Jason M. O'Kane. "A Low-cost Socially Assistive Robot and Robot-assisted Intervention for Children with Autism Spectrum Disorder: Field Trials and Lessons Learned." *Autonomous Robots* 41.3 (2016): 637-55. Print.
13. Duquette, Audrey, François Michaud, and Henri Mercier. "Exploring the Use of a Mobile Robot as an Imitation Agent With children with Low-functioning Autism." *Autonomous Robots* 24.2 (2007): 147-57. Print.
14. Newman, Judith. "To Siri, With Love." *The New York Times*. The New York Times, 17 Oct. 2014. Web. 04 June 2020. <https://www.nytimes.com/2014/10/19/fashion/how-apples-siri-became-one-autistic-boys-bff.html>.
15. "Early Start Denver Model (ESDM)." Autism Speaks. N.p., n.d. Web. 26 Aug. 2020. https://www.autismspeaks.org/early-start-denver-model-esdm
16. Denne, Louise D., Richard P. Hastings, and Carl J. Hughes. "Common Approaches to Intervention for the Support and Education of Children with Autism in the UK: An Internet-based Parent Survey." International Journal of Developmental Disabilities 64.2 (2017): 105-12. Print.
17. UK, Research Autism. "Visual Schedules and Autism Ranking:." Visual Schedules and Autism - Interventions - Research Autism. N.p., n.d. Web. 27 Aug. 2020. http://www.researchautism.net/interventions/57/visual-schedules-and-autism.
18. "The Picture Exchange Communication System (PECS)." The Picture Exchange Communication System (PECS). N.p., n.d. Web. 27 Aug. 2020. https://www.nationalautismresources.com/the-picture-exchange-communication-system-pecs.
19. "Speech Therapy." Autism Speaks. N.p., n.d. Web. 27 Aug. 2020. https://www.autismspeaks.org/speech-therapy.
20. "Applied Behavior Analysis (ABA)." Autism Speaks. N.p., n.d. Web. 27 Aug. 2020. https://www.autismspeaks.org/applied-behavior-analysis-aba-0.
21. "Understanding and Developing Communication." Autism Support - Leading UK Charity - National Autistic Society. N.p., n.d. Web. 27 Aug. 2020. https://www.autism.org.uk/advice-and-guidance/topics/communication/understanding-and-developing-communication.
22. Happé, Francesca, and Uta Frith. "Theory of Mind in Autism." Learning and Cognition in Autism (1995): 177-97. Print.
23. "AI Assistants That Go beyond Basic FAQs." Rasa. N.p., n.d. Web. 05 Sept. 2020. https://rasa.com/product/why-rasa/.

24. "Rasa NLU: Language Understanding for Chatbots and AI Assistants." Rasa NLU: Language Understanding for Chatbots and AI Assistants. N.p., n.d. Web. 05 Sept. 2020. https://rasa.com/docs/rasa/nlu/about/.
25. "Architecture." Architecture. N.p., n.d. Web. 05 Sept. 2020. https://rasa.com/docs/rasa/user-guide/architecture/.
26. Mantha, Mady. "Introducing DIET: State-of-the-art Architecture That Outperforms Fine-tuning BERT and Is 6X Faster to Train." The Rasa Blog: Machine Learning Powered by Open Source. The Rasa Blog: Machine Learning Powered by Open Source, 11 Mar. 2020. Web. 06 Sept. 2020. https://blog.rasa.com/introducing-dual-intent-and-entity-transformer-diet-state-of-the-art-performance-on-a-lightweight-architecture/.
27. "Free Printable Asking For Help Social Scripts For Kids". And Next Comes L. N.p., n.d. Web. 09 Sept. 2020. https://www.andnextcomesl.com/2016/12/asking-for-help-social-scripts.html.
28. "Free Printable Greeting People Social Scripts For Kids". And Next Comes L. N.p., n.d. Web. 09 Sept. 2020. https://www.andnextcomesl.com/2016/11/greeting-people-social-scripts.html.
29. "Simple Social Scripts That Work". STAR Autism Support. N.p., n.d. Web. 09 Sept. 2020. https://starautismsupport.com/simple-social-scripts-work.