Imperial College London

BEng Individual Project

Imperial College London

Department of Computing

Developing a Natural Language Processing Interface for Voice-Controlled Video Games

Author:
Taariq Pala

Supervisor:
Dr. Mark Wheelhouse

Second Marker:
Dr. Giuliano Casale

June 13, 2021

0 Contents

1 Abstract

Adding effective voice-controls to a video game is a sought-after additional feature that enhances user experience, but currently, significantly limits a player's ability to express themself. Previous research and development have spent a significant amount of effort in hard-coding permutations of phrases or building several grammars to alleviate this constraint.

The primary objective of this project is to reduce the amount of development time required to add voice-controls by creating an easily accessible interface, 'Intent', that allows effective voice recognition to be added to a video game. The interface was developed using a test-driven Rapid Application Development (RAD) approach, which consists of a ten-stage pipeline that optimistically extracts meaning from the user input and matches it to a known in-game command. The interface utilizes several NLP technologies including Semantic Role Labeling, WordNet and Word2vec word embeddings to facilitate user-input matching, and is hosted as a REST API that can be used by games developed on any platform. To illustrate the interface's capabilities, a mobile game named 'Tork' was created, which serves as a template for text-adventure "escape-room styled" video games made for Android.

Scalability, semantic similarity approaches and accuracy were among the performance assessment metrics for the project, which were determined through mock testing and a comprehensive beta programme. The interface was considered to be effective in maximizing a player's freedom of expression by both developers and gamers. The resulting evaluation deemed Intent and Tork to be a fast, effective and scalable solution for adding voice commands to a text-adventure video game. The project also analyses limitations and potential modifications for Intent's wider use.

2 Introduction

2.1 Overview

The world of gaming is continuously expanding its user experience, attempting to make the gaming experience increasingly immersive. In this endeavour, various forms of input have been explored ranging from a traditional gaming controller to input via gestures [1]. However, one form of input has only had a very limited integration into mainstream games, namely, voice controls [2].

Speech is a very natural way for humans to interact and communicate with technology and it was, for many years, viewed as a futuristic interface for interacting with computers. However, the concept of using voice as an input mechanism is not alien to developers or gamers. In voice-controlled video games, the user speaks into a microphone and their speech is converted into text via a speech to text (STT) interface [3]. Many such interfaces have already been developed to a high degree of accuracy. A prominent example of such an interface is Google Cloud Speech-to-Text [4].

However, integrating voice commands into current games is not trivial. With the English language, there are often many phrases with the same meaning. The naive approach would be to hard-code possible statements and their permutations that a user might say and map these to the relevant in-game commands. Any input not recognised by the game engine would be rejected. As such, the flexibility of voice control within video games is vastly restricted. Too few phrases would make for a poor user experience and too many phrases would significantly increase the development time. It could be for this reason that voice control within video games is a limited and often optional feature.

Natural Language Processing (NLP) may be the solution to creating a more intuitive and smooth voice-controlled gaming experience [5]. NLP is a subfield of artificial intelligence aimed at understanding the interactions between computers and human language. It allows machines to derive meaning from human language and accurately respond. For example, if a user was to say, “lift the table”, NLP can be used to determine that the user intends to pick up the table. It is important to note that multiple phrases may have the same meaning and translate to the same command.

“Escape-the-room”-style video games are a subgenre of role-playing video games in which a team of players have the objective of escaping the room they are currently in [6]. This typically involves searching for clues to solve various puzzles within a limited amount of time. It often also features a live game master who provides hints and oversees gameplay during a role-playing game. Online "escape-the-room" style video games have become increasingly popular throughout the world especially during the COVID-19 pandemic which saw a significant increase in online gaming [7].

2.2 Problem Statement

The current predicament for developers is that to map voice input to an in-game command, they would have to hard-code as many variations of a possible input as they can. Too many variations would lead to a significant increase in development time and too few variations would lead to a poor gaming experience. To mitigate this, current implementations limit the format of a user’s input and have a restricted vocabulary, thus reducing the players’ freedom of expression.

This project aims to create a natural language interface that can be used to interpret speech given by a user and can translate these commands into intents within a playable "escape-the-room"-style game. This project will apply natural language processing to input obtained from a user via a microphone. NLP can be used to attempt to understand this input and deduce the user's intent within the game. For example, if the user says "pick up the glass", the interface should be able to infer that the user intends to grab an object within the game room, where grab is an action recognised by the game. Rather than forcing developers to hard-code variations of the same phrase such as "pick up the glass", "lift the glass", "grab the glass", "hold the glass", the NLP will be able to map all of these sentences to the same intent. This will provide the player with more freedom of expression and thus improve the quality of their gaming experience. The NLP interface will avoid the use of a context-free grammar (CFG) where possible to maximise the input flexibility.

2.3 Objectives

There are several key objectives for this project:

1. Create an NLP interface that can extract meaning from user input in the context of "escape-the-room"-style or interactive fiction games.
    - Provide fast processing to ensure a seamless user experience.
    - Enable ease-of-development for game developers wanting to add an effective voice control interface to their game.
    - Be readily available and easily accessible for developers.
    - Apply various NLP techniques to extract meaning from user input.
    - Maximise the freedom of expression a player can have.

2. Create a simple interactive fiction game that makes use of this NLP interface.
    - Use a speech-to-text interface to retrieve user input.
    - Evaluate the effectiveness of the NLP interface by testing its capabilities.

2.4 Motivation

A key motivation behind pursuing this project is the gap in the market for voice-controlled gaming. Due to the tradeoff between user experience and development time, this gap is understandable. However, with the advent of NLP, the technology could provide developers with the opportunity to fill the gap, by providing a seamless user experience without massively increasing the development time thus eliminating the tradeoff. Most importantly, making the interface readily available for game developers can greatly reduce development time.

Another motivation behind this product is the concept of accessibility, particularly for disabled gamers. The ability to use another form of input besides the traditional mouse and keyboard or controller can enable gaming to become more accessible for gamers requiring less conventional methods of input.

This project can be extended to enhance competitive gaming. Having voice-controls that can be trusted and used simultaneously with a traditional controller input can be a key edge in competitive gaming. For this reason, the NLP interface must provide swift and rapid responses to user input.

3 Background

This chapter explores the various research on existing solutions and technologies related to voice recognition and Natural Language Processing that forms the basis of this project.

3.1 Voice Control in Current Games

This section contextualises and examines existing video games that use speech recognition as a primary input method.

3.1.1 There Came An Echo

"There Came An Echo" is an indie real-time strategy game in which a player commands a squadron with the option of using their voice to issue orders. The developers, Iridium Studios, crowdfunded the game via Kickstarter and received funding from Intel to implement Intel's RealSense technology. The voice-control feature was one of the key drivers in marketing this game. The mechanics of the game are fairly simplistic to facilitate the voice commands without overwhelming the player.

The game has predefined commands which are mapped to different actions within the game. The game encourages users to use their voice to issue commands by allowing users to add custom phrases to a mapping. The game is designed for the use of voice-control and so the game can be completed entirely using voice commands [8].

The voice-control implementation in this game is somewhat limited since commands are hard-coded. Whilst there is some flexibility in that users can add custom phrases, this can be a tedious process. It cannot understand the natural language and its many variations. There are alphanumeric designations assigned to everything a player can interact with. It is simple and effective and the voice control is well integrated into the game. However, there are restrictions. Firstly, a player can only move to places that the game has designated with a marker which is particularly restrictive for a tactical game. Another restriction is the time taken to respond to voice commands. Whilst the response time is fast, since the game is real-time, issuing commands under pressure can be difficult [9].

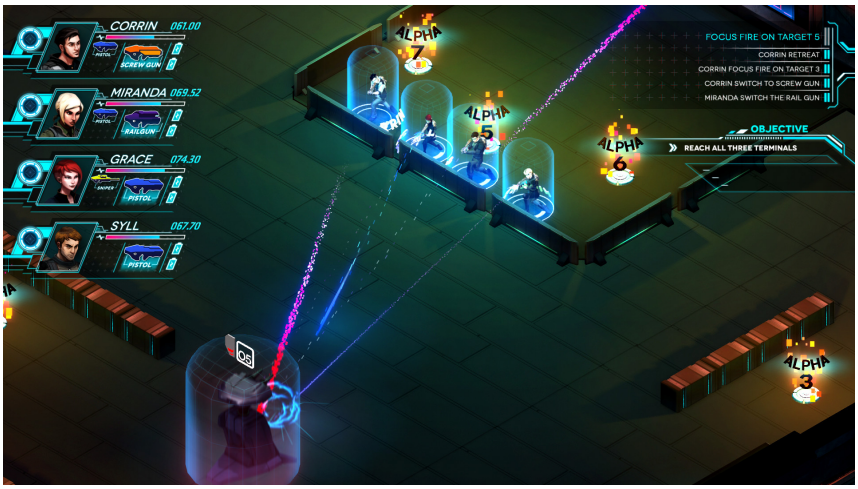


Figure 3.1: A screenshot of "There Came An Echo" in which the player is directing a squad using voice commands.

3.1.2 Tom Clancy's EndWar

"Tom Clancy's Endwar" is a third-person strategy role-playing game in which players control an army and features one of the first attempts at implementing voice-control as a major component into a game [10]. Players may optionally issue commands using a microphone. Ubisoft Shanghai said it was possible to control the game entirely using a microphone. This game was released in early 2009 and as such the speech recognition is not as good as it could be using more recent libraries [11].

The implementation of voice control in this game, known as Overlord, has a clear reliance on a semantic slot-filling grammar. Voice commands follow a specific pattern: usually a who, what and where. Voice commands also trigger a menu that allows the player to see what options are available as seen in Figure 3.2. There are several voice commands associated with several in-game actions such as:

- "Unit <NUMBER> Attack Hostile <NUMBER>".
- "Task group <Number> move to Foxtrot".
- "Deploy Riflemen".

With a real-time strategy game like this, it is often quicker and easier to command troops verbally especially in intense gameplay moments. However, there is the clear use of a grammar to determine what the user intends and these phrases are hardcoded. This greatly limits the freedom of expression a player has, for example saying "Riflemen to Foxtrot" would be an unrecognised command. The voice recognition could have been improved by increasing the number of variations of phrases the player could use for each command [12].

Figure 3.2: A screenshot of "Tom Clancy's EndWar" in which the player is directing troops using on-screen voice commands.

3.1.3 In Verbis Virtus

"In Verbis Virtus" is a first-person adventure video game developed by Indomitus Games [13]. The player begins in a dungeon and is required to overcome obstacles by learning magic through inscriptions they find as they traverse through the chambers. The game was developed using Unreal Engine in conjunction with the PocketSphynx library [14].

Using a microphone, players can use their voice to cast spells by reciting incantations found within the game. For example, saying “freezing beam” produces a spell that projects ice.

This is a very rudimentary implementation of voice recognition and it is clear the necessary spells are hard-coded into the games. The game was also not primarily developed to solely use voice control and a keyboard and mouse are still required for the majority of the game. Rather than recite the spells, users can simply click on the relevant spell or use a key binding which may be more convenient in this fast-paced game [15].

Figure 3.3: A screenshot of "In Verbis Virtus" in which the player is casting a spell using a voice command.

3.1.4 Classic Zork on Alexa

"Zork" is one of the first examples of a text-based interactive fiction video game [16]. Zork is set in an ancient empire and the player is an adventurer searching for wealth and adventure. The player is provided with a scenario via text as seen in Figure 3.4. The user is expected to input via text as to what they want the character to do (e.g. “open the mailbox”). In this game, the player interacts with the “master of the dungeon” which is similar to the game master in an “escape-the-room”-style video game.

Classic Zork on Alexa was developed using Amazon’s Alexa Skills Kit allowing players to issue commands through their Alexa which then responds appropriately [17]. This implementation simply adds a speech-to-text interface whilst maintaining Zork’s limited text parser [18].

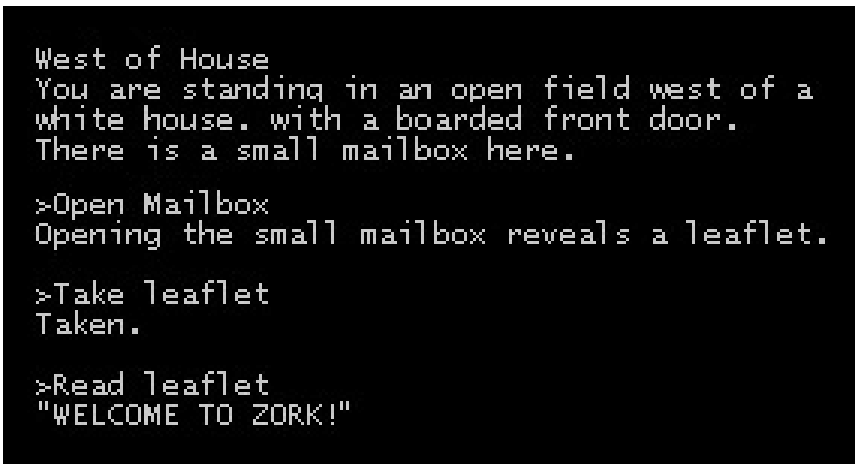


Figure 3.4: A screenshot of the text-based interactive fiction game Zork.

Zork simulates the universe which contains 191 different “rooms” (locations) and 211 “objects” (contexts). The vocabulary it handles is 908 words of which 71 are distinct “actions”. This vastly contrasts conversational vocabulary which far exceeds this number. Whilst Zork’s text parser was considered state of the art at the time it has certain limitations [19].

Zork limits the commands it can be given to a single action accompanied by up to two objects. The parser parses the player’s input into PRSA, PRSI and PRSO. PRSA represents the action, PRSI represents the direct object (target) and PRSO represents the indirect object (context). Initially, the parser will verify that all words in the input sentence are in its vocabulary. It then verifies that any objects mentioned are either global, within the player’s inventory or within the room the player was currently in. Adjectives can be used to distinguish between objects of the same type. If an object remains ambiguous, the game will ask the user for clarification. The next step is syntax analysis in which the parser identifies the verb within the sentence and treats it as an action. Syntax analysis also uses prepositions within the sentence allowing important differentiation, such as between “look at” and “look under” which are distinct actions. If the parser requires additional information, it will use information retrieved from the current game state to determine the missing object. If an object cannot be identified, the user is asked for clarification. All other words are ignored thus reducing the sentence to only what is necessary to process the in-game intent. Once the sentence has been parsed, it searches the syntax table for an entry that matches the verb as well as the PRSO and PRSI combination [20].

As previously mentioned, Zork has a limited vocabulary of 908 words and words that are not recognised are ignored. In addition to this, Zork relies heavily on a context-free grammar, expecting inputs in the form <ACTION> <DIRECT OBJECT> with <INDIRECT OBJECT>. Below are some examples of accepted statements:

- Go north.
- Open the mailbox.
- Close the mailbox.

Here are some examples of rejected statements:

- Close the box.
- Shut the box.
- Check the mailbox.

This project is somewhat inspired by Zork and attempts to recreate the same game mechanics with an improved parser implementing NLP.

3.1.5 Sphinx for Unreal Engine 4

Unreal Engine is a 3D creation platform for developing video games. Sphinx-UE4 is a speech recognition plugin for Unreal Engine 4 [21]. The plugin uses the Pocketsphinx library developed by CMUSphinx to provide offline speech-to-text. The plugin enables the hard-coding of phrases to commands they map to. However, it also supports grammars representing the structure of the phrases that can be accepted as input. A grammar is a set of rules for writing strings, for example, <letter> <digit> <letter> would accept inputs such as "a3b" or "r4t". Sphinx enables developers using the Unreal Engine to conveniently add voice commands to their game. However, the voice commands are governed by hard-coded phrases or grammars which limits the player's freedom of expression.

3.1.6 Unity

There are several built-in methods of adding voice input to a game developed using Unity [22].

1. KeywordRecognizer: provides an app with an array of string commands to listen for.
2. GrammarRecognizer: provides an app with a specific grammar to listen for.
3. DictationRecognizer: allows an app to listen for any word and process the speech.

The speech input can be dealt with directly through the code without requiring any additional plugins [23]. The above methods allow for hard-coding specific phrases which can then be mapped to the relevant in-game commands, as well as creating grammars to restrict inputs. Similarly to Sphinx for Unreal Engine, these methods limit the freedom of expression a player has.

3.2 Natural Language Processing

Natural Language Processing (NLP) is becoming an increasingly investigated area of Artificial Intelligence. NLP is the subfield of Artificial Intelligence associated with deriving meaning from the human language [5]. It can be used to create various systems such as speech recognition, predictive typing and document summarization. Some prominent modern examples of technologies that use NLP at its core include: Microsoft Cortana, Amazon Alexa, Google Home and Apple Siri [24]. Conversing with humans requires an under-

standing of syntax (grammar), semantics (meaning), morphology (tense) and pragmatics (conversation). Many ambiguities appear when dealing with language and this makes the task of determining the meaning of a sentence difficult [25].

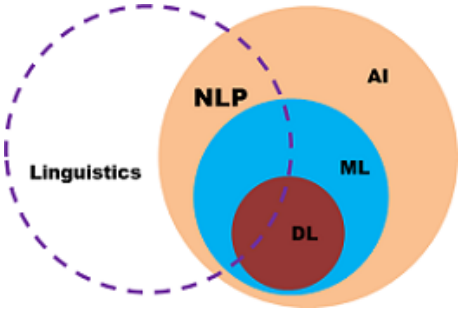


Figure 3.5: A diagram displaying the subfields of Artificial Intelligence and how they overlap.

There are typically five phases involved in NLP, as seen in Figure 3.6. An initial stage known as tokenization may be added in which the sentence is stripped of unnecessary details and divided into smaller components to make processing easier for the following stages. This typically involves sentence segmentation and tokenization [26].

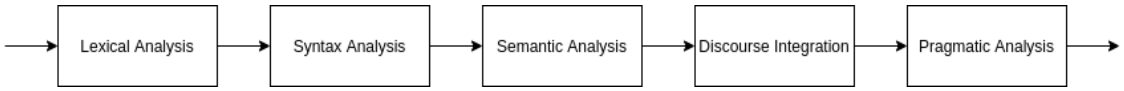


Figure 3.6: A diagram showing the five stage pipeline commonly used in Natural Language Processing.

3.2.1 Stage 1: Lexical Analysis

The first stage of NLP involves analysing the words for their lexical meanings [27]. A lexicon of the language is the collection of words and phrases within the language. The lexicon is used to convert a stream of characters into meaningful lexical units. This stage takes the output of the tokenization stage and attempts to break down the sentence even further [28]. It achieves this through some common lexicon normalization processes to reduce inflectional forms of a word to a common base:

1. Stemming: strip the word of its suffixes (i.e. removing “ing”, “es”, “ly”, “s” etc from a word).
2. Lemmatization: normalises words into their root form, also known as its lemma. This is typically achieved by using a look-up table.

3.2.2 Stage 2: Syntax Analysis

The second stage of NLP is also known as parsing [29]. Syntax refers to the rules that govern the structure in a natural language particularly how different words such as nouns, verbs, adverbs are sequenced in a sentence. The sentence must follow grammatical rules. This involves determining the grammatical structure of the sentence and validating that it makes sense. For example, “I drive a car to my college” makes sense whereas “I a car to my college drive” does not. Typically, this stage involves the use of a syntax tree to show a relationship amongst the words, as shown in Figure 3.7. For this, it uses “part of speech” tagging.

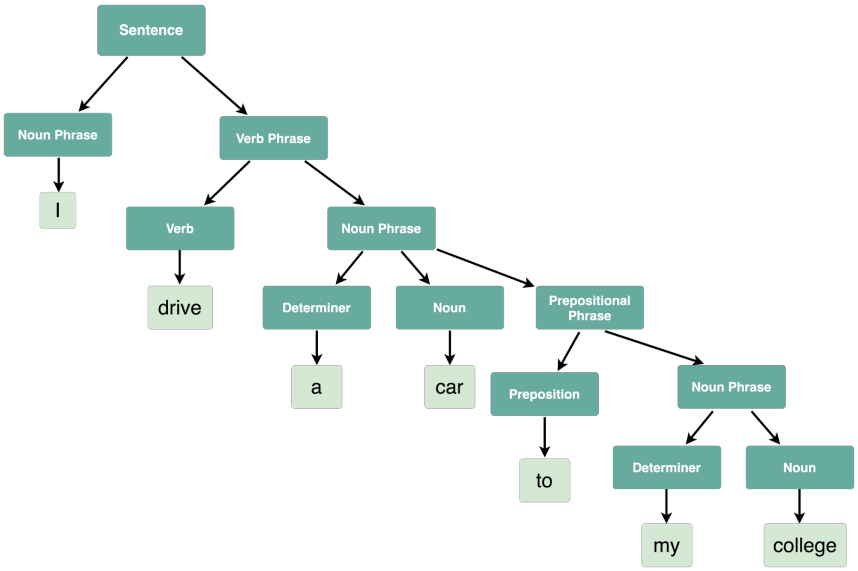


Figure 3.7: A diagram displaying a syntax tree generated from a sentence.

3.2.3 Stage 3: Semantic Analysis

In this stage, the NLP interface attempts to extract meaning from the sentence by focusing on relationships between lexical items (words) [30]. For example, "colourless yellow chocolate" would be rejected as "colourless yellow" has no meaning. Hence, semantically, this sentence would be viewed as incorrect.

3.2.4 Stage 4: Discourse Integration

In this stage, the NLP interface looks at the entire context including previous sentences to help bring meaning to the sentence it is analysing [31]. For example, "Kate is a girl. She goes to school." In this case, "she" is a dependency that points to Kate. The sentence must make sense within the context and so the second sentence is dependent on the prior discourse context. This requires techniques such as anaphora resolution to determine what pronouns refer to.

3.2.5 Stage 5: Pragmatic Analysis

The final stage of NLP and perhaps the most difficult is pragmatic analysis [32]. In this stage, the NLP interface requires world knowledge rather than simply knowledge of the language to discover the true meaning of a sentence. For example,

- The teachers in the school refused the students because they wanted longer breaks.
- The teachers in the school refused the students because they wanted longer lessons.

In this example, the word "they" has different meanings in each sentence. In the first, "they" refers to the students and in the second, "they" refers to the teachers. To understand the difference, some real-world knowledge is required.

3.3 Natural Language Processing Solutions

This section describes available NLP tools and services that game developers may utilise to include speech recognition into their game.

3.3.1 Houndify

Houndify is a platform that uses AI to allow users to add voice input to their applications [33]. It allows for speech recognition and a powerful natural language understanding. Once integrated, the application will be able to understand a variety of inputs and the API can be configured to produce appropriate responses. However, Houndify is a paid service and requires an internet connection. Houndify also supports the mapping of custom voice commands to API responses [22].

Houndify can be used in this project by adding custom voice commands in the form of a grammar with alternative variations of certain words. For instance, the following expression can be added to the API:

("attack" | "strike" | "fight" | "battle") ["the"] "troll"

This expression can be expanded even further to allow for more variations, however, ultimately this will increase the development time and complicate the code whilst not covering as many variations as necessary.

3.3.2 Dialogflow

Dialogflow is an NLP platform developed by Google that allows developers to integrate conversational user interfaces into their applications. Dialogflow filters out the user intent from the input and rejects statements it does not recognise [34].

The main limitation of using Dialogflow is that it requires training phrases to be inputted to allow the intent to be recognised as seen in Figure 3.8. This is similar to hard-coding possible intents. However, there is some flexibility in this approach as variables can be encoded into possible intents. For instance, "attack the troll with the <item>" where Dialogflow searches for the item in a list of possible items. This closely resembles a slot-filling approach to intent recognition [35].

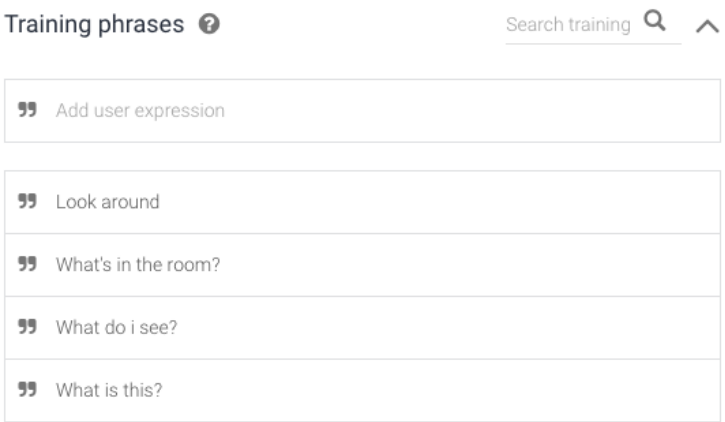


Figure 3.8: A diagram showing how training phrases are added to Google Dialogflow.

3.4 Part-Of-Speech Tagging

Part-of-speech (POS) tagging is the process of annotating each word within a sentence, with a "part-of-speech" tag based on its definition and its context [36]. A part-of-speech tagger is a piece of software that performs this tagging process. Basic tags include nouns, verbs and adjectives however more fine-grained tagging includes tags such as "noun-plural". A single word can have multiple tags [37].

Part of speech tagging is essential in this NLP interface as it needs to determine the action taking place, the indirect object and the direct object. Since the assumption is that the user is issuing commands imperatively, the action is represented by a verb and the objects are represented by nouns. Similarly, the word may contain adverbs, adjectives and prepositions all of which need to be handled accordingly.

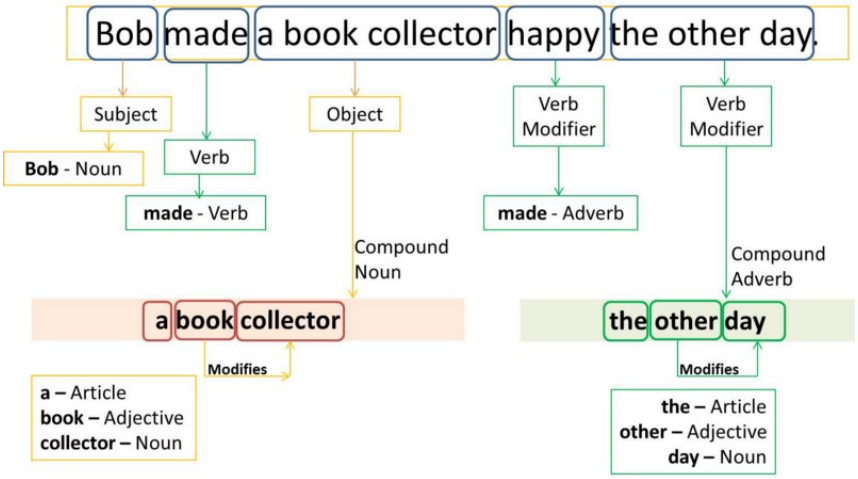


Figure 3.9: A diagram displaying how a sentence is tokenized and tagged using part-of-speech tagging.

3.5 WordNet

A limitation of classical text parsers was that they were only able to handle a select number of commands containing words from a limited selection of keywords. This made for a poor user experience and would often print out error messages to the user due to unrecognised commands. It is complicated to anticipate all the variations of a command a user might use. For example, if a text parser can understand "hit the troll" it may not understand "strike the troll". WordNet is a classic resource used in natural language processing [38]. It was created at Princeton University by Christiane Fellbaum and George Miller. WordNet is effectively a thesaurus designed to help establish connections between nouns, verbs, adjectives and adverbs. It is a massive lexical knowledge base that encodes information about synonyms and antonyms as well as hypernyms/hyponyms. The WordNet tool can be used to provide information about how words are related to each other [39].

A polysemous word is a word with multiple word senses (meanings) [40]. For example, the word arms may refer to the physical limbs protruding from a body (e.g. "my arms bend at the elbow", or it might refer to weaponry (e.g. "a call to arms"). WordNet organises word senses into unordered sets containing synonyms, also known as synsets. Each word may have multiple synsets where each synset represents a different meaning of the

word [41]. In addition to representing synonyms, WordNet can also represent hypernyms and hyponyms. It does this using a hierarchy. For example, a sword is a type of weapon and blue is a type of colour. These can also be referred to as "is-a" relationships. A hyponym is effectively the inverse of a hypernym. For example, if X is-a Y then X is a hyponym of Y and Y is a hypernym of X. Thus, a "sword" is a hyponym of a "weapon" and a "weapon" is a hypernym of a "sword". Each synset has its distinct hypernyms and hyponyms. Figure 3.10 shows the hypernymy hierarchy of the word “cherry” [42].

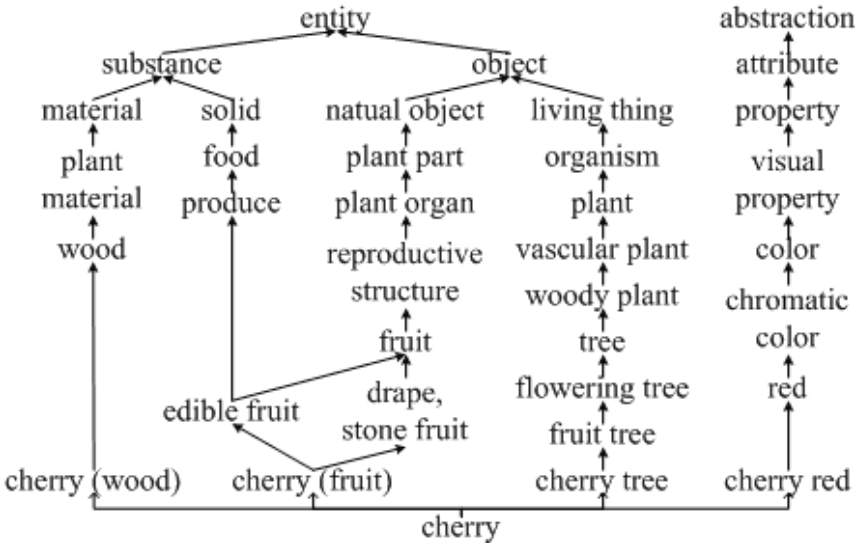


Figure 3.10: A diagram displaying the tree structure containing hyponyms and hypernyms used within WordNet for the word "cherry".

WordNet establishes key relationships between synsets:

- Synonym: words with the same meaning e.g. roof and ceiling.
- Hypernym: a word that is more general than another word e.g. weapon is a hypernym of a sword.
- Hyponym: a word that is more specific than another word e.g. sword is a hyponym of a weapon.
- Meronym: a word that is part of another word e.g. bedroom is a meronym of house.
- Holonym: a word that contains another word e.g. England is a holonym of London because London is in England.

For this project, synonyms, hypernyms and hyponyms will be the primary focus.

Using WordNet, it may be possible to support a wider variety of commands thus allowing for the freedom of expression a player needs for a better user experience. Rather than simply supporting “attack the troll”, many other examples can be supported such as:

- Strike the troll.
- Hit the monster.
- Beat the mythical creature.

WordNet is open-source and will be used within this project via NLTK enabling the NLP interface to interrogate the database [43].

3.6 Word Sense Disambiguation

A word may have multiple meanings or senses that can lead to ambiguity. For example, the word “interest” can refer to interest from a bank or interest in a subject [44]. Word Sense Disambiguation is a technique used to find the most appropriate sense of a word from the context of the sentence.

The Lesk algorithm is an algorithm that uses dictionary definitions to disambiguate words with multiple meanings using the sentence context. Given a sentence and an ambiguous word, the algorithm will provide a synset with the highest number of overlapping words between the sentence and the synset definitions [45].

3.7 Semantic Similarity

Semantic similarity is a metric that represents the distance between two words based on their meaning as opposed to basing it on lexicographical similarity [46]. It can be understood as "how much is one word related to another word?". There are many different ways to define this metric. They extend from simply counting the number of edges in the shortest path between the words within the WordNet hierarchy to more complex methods described below. It is important to note that some metrics do not attempt to distinguish between similarity and semantic relatedness and so whilst two words may not be semantically similar, they may be related which provides a higher score than expected [47]. For instance, love and hate are antonyms and thus have higher relatedness but low similarity. Other semantic similarity methods will be explored such as FASTLEX, Resnik and Cosine amongst others.

3.7.1 Leacock-Chodorow Similarity Algorithm

The Leacock-Chodorow (LCH) method counts the number of edges on the shortest path between two words and uses log smoothing. The similarity is the negative log of the shortest path divided by twice the total depth of the tree.

$$sim_{lch}(S1, S2) = -log(\frac{shortestpath(S1, S2)}{2d_{max}})$$

where d_{max} is defined as the maximum depth of the tree.
For example, the LCH calculation for "fight" and "attack":

$$sim_{lch}(fight, attack) = -log(\frac{3}{2 * 14}) = 2.234$$

3.7.2 Wu-Palmer Similarity Algorithm

The Wu-Palmer method (WUP) is a common semantic similarity measure that builds upon earlier proposed measures [48]. It takes into account the position of each word in the tree and compares it with the least common subsumer. The least common subsumer (LCS) is defined as the lowest node in the hierarchy that is a hypernym of both words. The WUP calculation provides a similarity score between zero and one, where a score closer to one indicates the two words are similar. However, as mentioned this score does not distinguish between semantic similarity and relatedness. For example, the WUP score for dislike and love is 0.77 whereas for romance and love it is 0.25 [49].

$$sim_{wup}(S1, S2) = \frac{2 * depth(LCS(S1, S2))}{depth(S1) + depth(S2)}$$

For example, the WUP calculation for "fight" and "attack":

$$sim_{wup}(fight, attack) = \frac{2 * 2}{3 + 3} = 0.667$$

3.7.3 Lin Similarity Algorithm

The Lin method is similar to the WUP method however it uses Information Content (IC) rather than the least common subsumer. The IC of a concept is calculated by taking the negative log of the probability of a concept which in turn is based on the sense frequency of a word within a synset. IC-based similarity results are considered better than path-based results.

$$sim_{lin}(S1, S2) = \frac{2 * IC(LCS(S1, S2))}{IC(S1) + IC(S2)}$$

For example, the LIN calculation for "fight" and "attack":

$$sim_{lin}(fight, attack) = \frac{2 * 5.666}{6.718 + 7.281} = 0.875$$

3.8 Word Embeddings

Vector space embedding models are becoming increasingly common in natural language processing applications. A word embedding is an n-dimensional vector space representation of words. It is a way of capturing the context of a word within a sentence [50]. Importantly, words that are semantically related or similar based on the training data are closer in the vector space. Rather than manually organizing words into sets and a hierarchy consisting of hyponyms, hypernyms and synonyms, the interface can simply compute the similarity of words using vectors. For example, the most similar words for a 'troll' would be 'goblins', 'ogres', 'elves', 'ghouls', 'hobbits', 'centaurs' etc. The similarity is computed by comparing the vectors and computing the cosine of the angle between them. It assigns a number between 0 and 1 to indicate the distance between words. Thus, vectors with smaller angles between them tend to have similar meanings. An example of this is shown in Figure 3.11. A popular algorithm for this task is Word2vec which was developed by Google in 2013 [51].

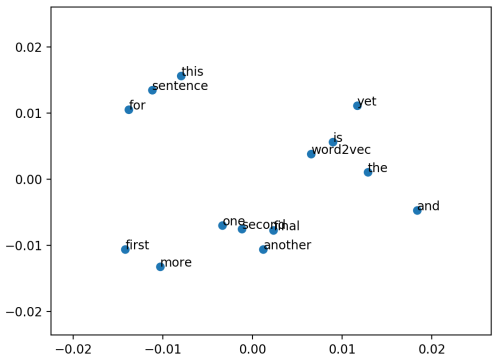


Figure 3.11: A simplified diagram displaying how words are represented by vectors using Word2vec.

3.8.1 Word2vec

Word2vec is a predictive embedding model used to produce a distributed representation of words. It was developed by Google in 2013 [52]. Word vectors are stored within a dictionary which can be interrogated. Word2vec can also predict the next probable word in a sentence. However, these vector sets based on corpora are usually extremely large and interrogating them can be a tedious process.

There are two main categories of word vectors:

- Frequency-based: Compute probability of word co-occurring with respect to its neighbouring words based on statistics.
- Prediction-based: Use predictive analysis to make a weighted guess of a word co-occurring with its neighbouring words.

There are two variants for Word2vec:

- Continuous bag of words: the model predicts a word based on the neighbours in a specific context.
- Continuous Skip Gram Model: this model tries to predict the neighbours of a word in each context separately.

3.8.2 Limitations of Word Embeddings

The main disadvantage of Google's Word2Vec and Stanford's GloVe embeddings are that they are unable to encode unknown or out-of-vocabulary words. Facebook proposed FastText, an extension to Word2Vec which initially breaks words into several sub words and feeds them into the neural network model. This methodology allows for unknown words to be represented in vector form [52].

Whilst word embedding techniques are fast and efficient they do not always provide an accurate result as they do not take into consideration the order of words in which they appear. This results in a loss of syntactic and semantic understanding of the sentence. For example, "you are going there to learn not to fight" and "you are going there to fight not to learn" have different meanings however they have the same vector representations.

To mitigate this, AllenNLP produced the Embedding from Language Model (ELMo). ELM is a deep contextualized word representation that models both syntax and semantic as well as linguistic contexts. They effectively consider words within the context they have been used in rather than creating a dictionary of words.

3.8.3 BERT Contextual Word Embeddings

Another related technology is Google's BERT Contextual Word Embeddings. Since word embeddings don't account for context, they do not handle ambiguity well [53]. A good example of this is using the word "bat" which can either refer to a cricket bat or the winged mammal. This would depend on the context. The regular Word2vec method would pick a vector between the two. Contextual word embeddings are word embeddings that account for the context of the sentence. Google created Bidirectional Encoder Representations from Transformer (BERT), a neural network designed to produce a single embedding per token from its input. Similarity tasks become accurate and contextual, however, the computation complexity is increased significantly.

3.8.4 Magnitude

Magnitude is a lightweight Python package that acts as a simple vector embedding utility library that allows for fast, efficient interrogations of vector embeddings in machine learning models. The Magnitude file format for vector embeddings is intended to optimise interrogations. It does this in several ways including lazy loading for faster cold starts, SQLite data stores, indexing for fast key lookups, spatial indexing for fast similarity searching as well as caching. The benchmarks make it an ideal package for a video game context where response time is crucial.

It requires the download of a set of vectors from a source in a format that Magnitude can interpret. Magnitude offers pre-converted magnitude formats of popular embedding models such as Word2vec, GloVe, FastText and Elmo. It offers various versions to allow for vectors with more dimensions and additional support for out-of-vocabulary keys. A compatible BERT embedding model is currently under development [54].

3.9 Dependency Parsing

A dependency parser analyses the grammatical structure of a sentence to determine how each word relates to its parent. It creates a graph in which each word has a single parent and each edge in the graph has a label that describes its grammatical role, such as whether it is a determinant or a direct object [55].

For this project, rather than hard-coding the verb object, a dependency parser can be used to automatically extract the verbs and their corresponding direct objects. This can even allow the interface to support a string of commands such as “pick up the sword and attack the troll.”

AllenNLP provides a visualisation of how the dependency parser works. This can be seen in Figure 3.12. As shown, the sentence, “attack the troll with the sword” is broken down into a verb, prepositions, prepositional objects and dependencies.

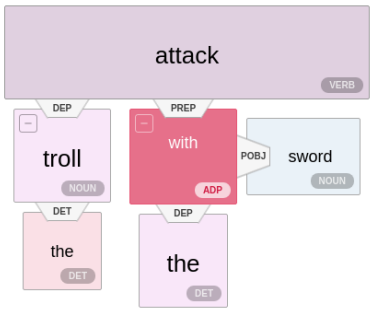


Figure 3.12: A dependency parsing graph generated using the AllenNLP demo [56].

3.10 Semantic Role Labelling

Semantic Role Labeling (SRL) is an NLP technology that aims to recover the verb predicate-argument structure of a sentence such as "who did what to whom, when, why, where and how". It enables the extraction of the main verbs and their corresponding objects [57]. To achieve this, it detects arguments associated with the verb of the sentence and classifies

them into roles, such as an agent, predicate, theme or location. The example in Figure 3.13 shows how the sentence "get the sword and attack the troll with the sword" is broken down.

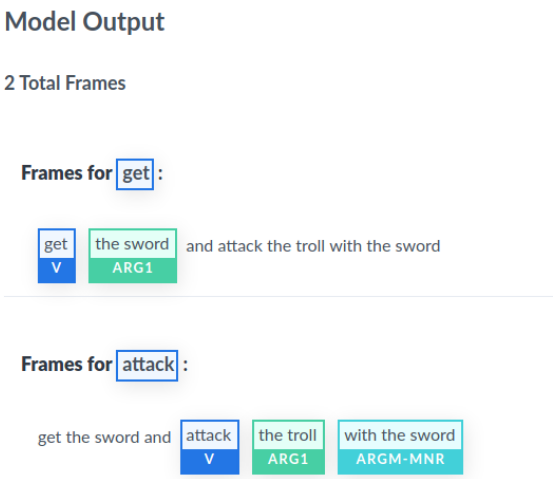


Figure 3.13: A screenshot taken from the AllenNLP demo for Semantic Role Labeling, displaying how SRL can be applied.

3.11 Open Information Extraction

Open Information Extraction (Open IE) takes an input sentence and extracts a list of propositions, each of which is composed of a single predicate and an arbitrary number of arguments. Open IE produces a set of relational tuples which attempt to capture the relationships within the sentence [58]. For instance, given the input "In the Battle of Agincourt, the French fought the English but lost the battle", the Open IE system extracts tuples typically in the form <first argument, predicate, second argument>. Here it extracts: <the French, fought, the English> and <the French, lost, the battle>.

3.12 Coreference Resolution

Coreference resolution is an algorithm that has the role of finding all expressions that refer to the same entity in a text [59]. For example, in the example "I went to the shop for Jack because he was hungry", the algorithm produces clusters and associates "Jack" and "he" to the same cluster. This can be incredibly useful for this project where the player may not refer to an object by its name and rather as "it" or they may refer to an in-game entity as "him" or "her".

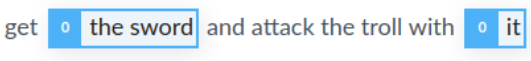


Figure 3.14: A screenshot taken from the AllenNLP demo for Coreference Resolution, displaying how CR can be applied.

3.13 Sentence Segmentation

Sentence segmentation or sentence boundary detection is a common NLP task where the text is divided into sentences [60]. This task typically occurs during the tokenization stage.

The sentence segmentation algorithm takes the sentence as input, splits it into tokens as part of the tokenization process and then separates the sentences by special tokens such as conjunctions or punctuation. For example, the sentence "Pick up the dog and pet it" can be divided into two sentences by removing the conjunction "and". It becomes "Pick up the dog. Pet it". In this project, sentence segmentation in conjunction with coreference resolution can be used to chain commands together.

3.14 Sentiment Analysis

For completeness, sentiment analysis will be briefly discussed. Sentiment analysis is an NLP technique used to determine whether a sentence is positive, negative or neutral [61]. This typically involves taking a piece of text and returning a sentiment score using machine learning techniques. The applications of this are limited within this project, however, it could be useful to demonstrate the power of NLP and add another dimension to this project. For example, a player can be required to insult an entity to progress to the next room. Whilst this is not an essential feature, it expands the freedom of expression a player can have.

4 Project Design

The project is split into two distinct sub-projects:

- Intent: an NLP interface capable of parsing user input for an escape-room style video game.
- Tork: an escape-room styled mobile game that can demonstrate the power and complete functionality of the NLP interface.

The following section outlines the plan for the project, explaining the process in which the NLP interface extracts meaning from an input and how a game is expected to interact with the interface.

4.1 Target Platforms

This section explains how the platforms of focus were chosen.

4.1.1 Intent

As discussed in Section 2, accessibility is a key requirement for this project. To facilitate this, the interface was chosen to be developed as a REST API. REST APIs are widely regarded as the industry standard for web APIS, with the ability to handle a variety of HTTP requests with the appropriate parameters in the body. Furthermore, a RESTful API can be easily and securely hosted online, making it a pertinent and accessible choice for developers on all gaming platforms, including Android, iOS and web-based video games.

Whilst its primary advantage is accessibility, a limitation to be noted for REST APIs is the response time. Although HTTP REST API's provide fast responses, they are not instantaneous as the response time includes the latency between the game and the web server. A solution native to the target platform may be more appropriate in the context of gaming as a faster response can be provided. However, this would limit the interface to the target platform, thus the accessibility requirement would no longer be fulfilled.

4.1.2 Tork

Ovum's Mobile Games Market forecast for 2017-2022 shows that by 2022, approximately 50% of video game revenue generated will come from mobile games as opposed to its counterparts [62]. Furthermore, all mobile phones have a built-in microphone which makes voice recognition possible for all users. For these reasons, Tork will be designed as a mobile application.

According to the GlobalStats Statcounter for the worldwide mobile operating system market share, as of March 2021, Android is by far the most popular mobile OS, with 71.83% of all mobile phones using Android as its operating system. In contrast, iOS devices make up 27.41% of all devices [63]. Moreover, Android is more accessible for development and the presence of Google's Speech-To-Text API makes it an ideal choice for a voice recognition application. For these reasons, Tork will be developed for Android.

4.2 Technologies

This section describes why particular development tools were chosen.

4.2.1 Intent - Python / Django

Python is a high-level general-purpose programming language that is increasingly used for machine learning and NLP applications. Python offers simple, concise and reasonable code. It greatly resembles the English language and is relatively lightweight. The project will require the use of WordNet which is a popular NLP tool and the database can easily be interrogated using the Natural Language Toolkit (NLTK) module which is exclusive to Python. In addition to this, various other Python libraries can be used to reduce the amount of code that needs to be written. Other languages such as Java and JavaScript were considered but due to the availability of NLP libraries, Python was deemed the best fit for this project.

Django is a high-level Python web framework that enables the rapid development of websites. Django REST framework extends Django by providing a flexible toolkit for building Web APIs that can handle various HTTP requests. Flask is another Python web framework that was considered; however, Django provides a full-stack web framework whilst allowing for rapid development whereas Flask is a lightweight extensible framework. Django was chosen due to the availability and accessibility of the Django REST framework.

4.2.2 Tork - Java / Android Studio

Java is a high-level object-oriented programming language that is commonly used in creating video games. As an OOP language, it enables the creation of maintainable code that is easily understandable, adaptable and extensible. The use of an inheritance hierarchy makes it easy to create an easily expandable game. For example, a class Dog and a class Troll can extend a class Entity. However, Java does not support multiple inheritance unlike other languages such as C++. Multiple inheritance can allow a class to inherit from multiple parent classes, which can be a major advantage when developing a game as it allows for more complex relationships between classes and greater flexibility when casting.

Android Studio is a software that provides a unified environment and necessary utilities for enabling easy application development for Android devices. It is an ideal choice for this project as the primary development language is Java which is coincidentally the same underlying language for Android development.

Flutter was an alternative to Android Studio that was considered as it provides an easy way to create cross-platform applications for both iOS and Android. It uses a language known as Dart which has some similarities to Java, yet it is fundamentally different. This application's primary target platform is Android, and development should be concentrated on a single platform for ease of development. As a result, Android Studio was selected.

4.3 System Architecture

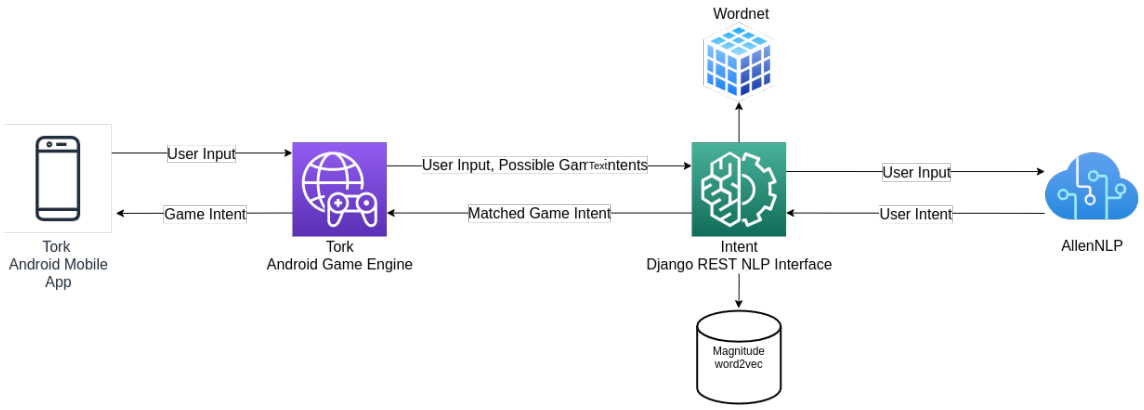


Figure 4.1: A diagram displaying the interaction of project components.

4.4 System Overview

This section describes the system's overall design as well as some of the design decisions taken.

4.4.1 Intent

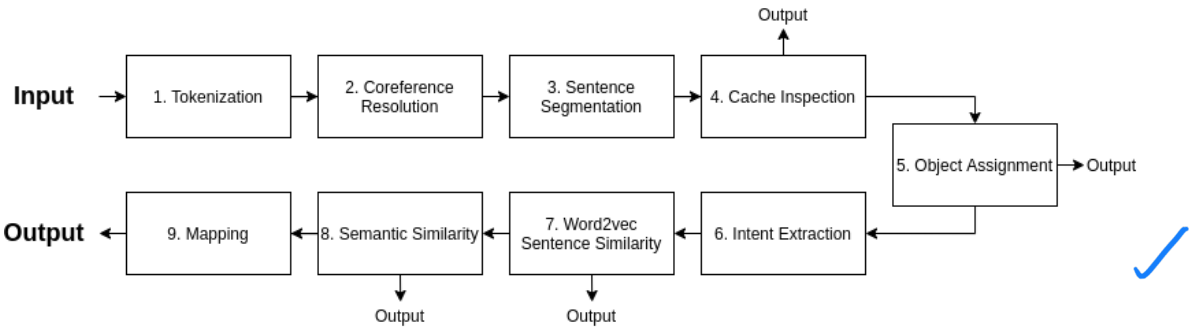


Figure 4.2: A diagram displaying the pipeline used by Intent.

Intent aims to initially discard any obviously incorrect matches and then choose the most similar matches and assign a probability to each one. The highest-ranking match will be considered the correct match.

1. **Input** - Accepting input consisting of the user sentence, in-game commands, inventory objects and room objects.
2. **Tokenization** - Validating the input and normalising the sentence.
3. **Coreference Resolution** - Replacing references with their respective objects.
4. **Sentence Segmentation** - Segmenting the sentences by splitting it on conjunctions.
5. **Cache Inspection** - Searching for a mapping within the cache.
6. **Object Assignment** - Determining which of the nouns are inventory objects and which are room objects. Handling common and special commands such as fetch, drop and examine.
7. **Word2vec Sentence Similarity** - Calculating a vector for the sentence to compare how similar the sentences are.

8. **Intent Extraction** - Extracting a verb, direct object and indirect object from the user-inputted sentence and the sentences representing in-game actions.
9. **Semantic Similarity** - Calculating the semantic similarity between the user-inputted components and the components of the game command.
10. **Mapping and Output** - Determining if there is no match, an ambiguous match or a successful match.

Intent will consist of this ten-stage pipeline and will utilise several NLP technologies provided by external modules:

- AllenNLP's models for coreference resolution and semantic role labelling
- NLTK's WordNet module for accessing the WordNet database and calculating semantic similarities
- Magnitude's word embedding module to provide sentence similarity methods

Using these external modules reduces the amount of development labour significantly, thus streamlining the process; nevertheless, adaptors/wrappers must be written to develop the application of these technologies to the sentence. This latter exercise constitutes a large portion of the code and must be developed independently of any external modules.

4.4.2 Tork

This mobile game aims to display the processing power of the NLP library. For this reason, the game will be simplistic in terms of gameplay and will focus on testing the capabilities of the NLP library by encouraging the players to use maximise their freedom of expression. The design for Tork is expected to be simplistic and minimalistic; however, importantly there should be room to expand the game. For this reason, the game is designed with a hierarchical structure in mind.

Speech-To-Text

Tork will use Android's built-in speech recognition service which enables speech to be converted to text easily. Developers can use the SpeechRecognizer class to provide access to the speech recognition service. The speech-to-text API works by streaming the audio to remote servers to perform speech recognition. Despite this, speech recognition is almost instantaneous and since internet access is already required by the game to communicate with the NLP interface, there are no additional requirements for the game to play. In the case of a phone with limited capabilities or users that are unable to use the microphone, there is also the option to input text via the Android Keyboard.

Communicating with Intent

Once the user's input has been received, the game must communicate with Intent. To communicate with a REST API, Android provides a useful library known as Volley which allows for HTTP requests to be sent to a URL. Volley enables this communication to be completed quickly which is crucial for the game. The user input, as well as the current possible in-game commands, will be placed within a JSON object which will constitute the body of a POST request to be sent to Intent.

The game should expect to receive a JSON payload back, with its body containing the parsed game intent. It will have a status field that indicates whether the interface was successful in matching the user input to an intent. If the interface successfully finds a match, it will also contain a verb, a direct object and an indirect object. These will be

used by the game to carry out the in-game action.

4.5 Deployment and CI/CD

The Intent platform will be a deployed web application hosted on Heroku, a free web hosting service. For continuous integration and continuous deployment, the project utilises CircleCI, an online CI/CD tool that automatically builds and tests before deploying to Heroku. Ngrok is a useful tool that can be used to tunnel from a public URL to an application running locally. Development is carried out locally using Ngrok before commits are pushed to the repository and subsequently to the live production version of the application.

However, Tork is not developed using continuous integration or deployment as it is a standalone mobile application. The Android project will be developed locally and tested using a Samsung Galaxy S9. This is because the primary focus of the mobile application is simply to demonstrate the capabilities of the Intent interface.

4.6 Testing

To ensure the project satisfies requirements and functions correctly, a test-driven development approach was utilised throughout the software development iterations. All features of the NLP interface are covered by integration and unit tests developed using the Django Test Framework. The testing is explored in more detail in Section 6 of this report.

5 Implementation

This section explores the various components of each of the sub-projects in greater detail.

5.1 Intent - The NLP Interface

Intent is implemented in Python using the Django Web framework to encourage rapid development and pragmatic design. It focuses on taking a user input and mapping it to a meaningful intent within the game. Intent disregards any knowledge of the game other than that supplied by the developer. Intent is designed as an optimistic "best-fit" matching algorithm that assigns probabilities to each of the game intents provided by the developer. For this to function correctly, Intent operates under the assumption that the user has inputted something meaningful and relevant into the game.

5.1.1 Django Implementation

Intent has been created as a Django project and thus follows the standard Django structure. A RESTful API includes numerous endpoints (or URLs) that define the structure of the API and how users can access its functionality using HTTP methods such as: GET, POST, PUT and DELETE. Intent has three endpoints:

- The root endpoint (/): all requests for intent processing occur here in the form of a POST request.
- The ambiguity endpoint (/ambiguity): requests to add a mapping to the database can be sent here in the form of a POST request.
- The feedback endpoint (/feedback): players can send feedback to Intent to inform the interface if the matching result was correct or not.

These URLs are stored in a file called *urls.py* which governs the routes that exist for the API. The file *settings.py* contains crucial information about the applications' configuration.

5.1.2 Views, Models and Serializers

Models are effectively the Python representation of tables within the database. A model contains the essential fields and behaviours of the data being stored. Each model maps to a single database table and each field within the model corresponds to a column. The fields can have various types such as CharField or IntegerField etc. Django automatically creates these tables when a model is created.

Views are used to help structure the backend code. The view is a callable which receives the request and returns a response. As such, it is effectively the entry point for HTTP requests. The Django REST framework provides various types of views such as the APIView and the class-based view. The class-based view is a powerful pattern that allows the reuse of common functionalities. Class-based views allow the interface to respond to various HTTP request methods using functions rather than conditional branching.

Serializers are a component of the Django REST framework that essentially serializes complex model instances into native Python data types that may subsequently be easily rendered into JSON. Serializers also enable deserialization to convert parsed data back into complex types after validating the data.

For this project, there are two main models, serializers and views: the intent processor and the ambiguity processor. The intent processor model consists of multiple fields including the sentence, game intents, inventory objects, room objects and entities. The ambiguity

processor simply consists of the two fields sentence and mapping.
An additional model called "Metrics" was created to store useful metrics and feedback about the responses made by Intent. This is used in Section 6.

5.1.3 Database

Django offers support for various types of databases but for this project, SQLite is used to store mappings. SQLite is used due to its simplicity and speed. PostgreSQL was also considered, however SQLite was determined to be more appropriate due to its speed advantage in a game context. The database stores a table containing ambiguous mappings. For efficiency, the database is only interrogated during the Cache Inspection stage of the pipeline. Additional interrogations of the database would cause an increase in response time.

5.1.4 Pipeline

The pipeline shown in Figure 3.6 is intended to give high accuracy without sacrificing performance. It accomplishes this by deferring the most performance-demanding operations to the end of the pipeline, allowing preceding stages to work to lower the amount of processing required. Several steps of the pipeline seek to short-circuit, including cache inspection, phrase similarity, and object assignment.

Stage 1: Input

The input to the NLP interface consists of:

- A user-inputted sentence.
- A list of possible game intents in sentence form.
- A list of items currently stored within the inventory.
- A list of items currently within the room.
- A list of entities currently within the room.

The game and user sentences are subject to the following assumptions to simplify game mechanics and the NLP task:

Assumptions

- The user will attempt to input something valid and issue a pragmatic and relevant command to the game.
- The subject of any input will be the player (e.g. I).
- The user will provide some form of an imperative command (e.g. attack the troll).
- The sentence will contain only one verb and a maximum of two nouns (e.g. attack, troll and sword).
- The developer will have provided a list of valid and unique commands also containing a single verb and a maximum of two nouns.
- The developer will have provided a list of inventory objects, room objects and entities in which any single object only works once.
- If the user wants to use an object, the object must already be within its inventory.
- If the user-inputted command contains two objects, at least one of those objects must be within the user's inventory.
- If the user-inputted command contains a single object, it can be a room object or an object within the inventory.
- The user-inputted sentence contains at most a singular object pronoun such as "it",

"him" or "her".
- The user-inputted sentence contains at most a single conjunction such as "and" or "then"

Since Intent is being designed as a REST API, it is expected to receive an HTTP request with a JSON body. Below is an example of a POST request made to the server. It consists of four JSON arrays: sentence, game_intents, inventory_objects and room_objects. At each phase of the gameplay, the developer is required to keep track of the inventory objects, room objects, possible game actions, entities as well as the inputted sentence from the user. This can then be easily provided to the interface as shown in Figure 5.1.


```
{
    "sentence": "strike the hound with the blade",
    "game_intents": [
        "eat the apple",
        "examine the apple",
        "raise the apple",
        "attack the dog with the sword"
    ],
    "inventory_objects": [
        "apple",
        "sword",
        "cloak"
    ],
    "room_objects": [
        "hook",
        "door"
    ],
    "entities": [
        "dog"
    ]
}
```


Figure 5.1: An example of a POST request sent to Intent using Postman.

Stage 2: Tokenization

This initial stage aims to standardise user input by reducing it to its essential and validating the input assumptions stated above. This is a crucial phase since unexpected inputs might cause the pipeline to fail. Several steps are involved in the standardising process:

1. Expanding contractions (e.g. we're becomes we are).
2. Removing punctuation such as commas.
3. Tokenizing the sentence by splitting it into words.
4. Part-of-speech tagging on the tokenized sentence.
5. Validating the sentence to ensure it matches the assumptions made about the input.

Stage 3: Coreference Resolution

In this stage, if the sentence contains a coreference, the interface works to resolve it. For example, with the sentence "fetch the sword from the wall and attack the troll with it", the reference "it" would resolve to "the sword". The interface replaces the occurrence of "it" with its respective object.

AllenNLP provides a model for coreference resolution that can be used to get an embedded representation of each span in the document. The span representations are scored based on how likely they are to occur in a coreference cluster. Using this score, references

and their respective objects are determined. The model returns a list of tuples containing the indexes of the reference and the object it refers to. Intent then locates the indexes of the references and replaces the elements with the relevant noun.

For the purposes of this project, the interface assumes there is only one reference within the sentence. In the current implementation, the list of supported references contains "it", "him" or "her". However, the capabilities of coreference resolution can easily be extended and this is discussed further in Section 6. Once the reference and its corresponding object have been detected, the interface must replace the reference accordingly.

Stage 4: Sentence Segmentation

This stage is intended to process multiple commands. Users can sequence multiple instructions using conjunctive words such as "and" or "then. For efficiency, this stage is only executed if the sentence contains a conjunction. The function takes in a tokenized sentence and splits it by the tokens provided. Each of these sentences can then be processed separately. For example, "take the apple and eat the apple" will be divided into two independent sentences: "take the apple" and "eat the apple". The algorithm designed for this is as follows:

```
def segment_sentence(tokenized_sentence, conjunctions):
    sentences = []
    for conjunction in conjunctions:
        if conjunction in tokenized_sentence:
            index = tokenized_sentence.index(conjunction)
            sentences.append
                        (" ".join(tokenized_sentence[:index]))
            sentences.append
                        (" ".join(tokenized_sentence[index + 1:]))
    return sentences
```

Stage 5: Cache Inspection

This stage is implemented to enable the fastest possible response and allow for ambiguity handling. Given a text-adventure game with a limited room it is likely that different users will input similar sentences. Thus, a cache is a useful way of short-circuiting the pipeline if a mapping has already been created previously. The interface interrogates a database of previously generated mappings and if a match is found this can be returned by Intent. The matching uses the Word2vec sentence similarity method to determine if a match is found. The algorithm generates a vector for each of the sentences within the database and a vector for the user input. The cosine angle between each sentence from the database and the user input is calculated and the highest scoring sentence is returned. This is far quicker and more efficient than running the full pipeline.

Cache Inspection is also useful for ambiguity handling. If the player inputs something ambiguous and the interface is unable to determine its in-game meaning, the game can ask the user for clarification. Upon receiving clarification, the interface can store the new information in its appropriate mapping within its database.

Stage 6: Object Assignment

This stage is required in order to determine which of the indirect and direct nouns is an inventory object and which is the room object. The direct object is the receiver of the action mentioned in the sentence whereas the indirect object identifies the object for whom the action is performed. This can vary from statement to statement for example:

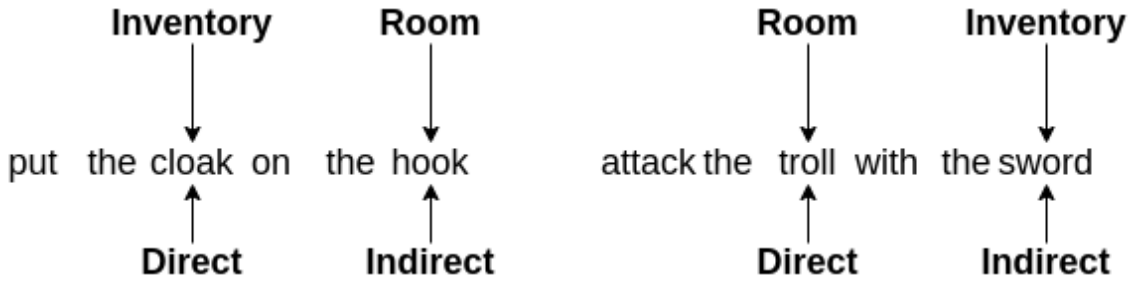


Figure 5.2: A diagram showing two examples in which the direct and indirect objects do not necessarily correlate to their corresponding inventory objects.

Determining what an object represents allows the number of commands to be further restricted without compromising the freedom of expression a player can have. With these assumptions, the interface can further limit what the user can do directly through the NLP interface. This greatly reduces the amount of development required by the developer. The assumptions are as follows:

- The room objects and inventory objects are supplied to the NLP interface.
- If the user wishes to use an item in any way, it must be within their inventory.
- If the user uses two nouns in a command, one noun should be an inventory object and the other should be a room object.
- If the user uses a single noun in a command, the interface determines if the object belongs to the room or the inventory. It can then check if the verb is semantically similar to "fetch", "examine" or "drop".

In addition to designating objects, the interface handles "fetch", "drop" and "examine" commands. These commands can occur at any stage in the gameplay and thus to reduce the amount of development work significantly, the interface seeks to handle this set of commands. It achieves this by detecting the relevant command and objects and responding to the game. During development, care has been taken to ensure that objects do not appear in multiple lists.

This is achieved by calculating a score for "fetch", "examine" and "drop" for the verb provided by the user. If the verb is similar to fetch, the interface expects the noun to belong to the room. If the verb is similar to "drop" then the interface expects the object in the sentence to be an inventory object. If the verb is similar to "examine" then the noun can correspond to a room object, inventory object or entity. The developer is expected to handle "examine" commands by outputting a description of the object. "Fetch" and "drop" commands should add and remove the respective items from the inventory whilst simultaneously removing the items from the room.

If the interface successfully detects a "fetch", "examine" or "drop" command then the responses will indicate this. The interface provides useful error messages if the assump-

tions fail. For example, if the player already possesses an item then they cannot fetch that item. Similarly, they cannot use or drop an item they do not possess. The various types of outputs are discussed in Stage 10: Mapping and Output.

Stage 7: Sentence Similarity

In this stage, a similarity measure between the user-inputted command and the in-game commands is calculated in order to eliminate incorrect possibilities whilst retaining possible matches. The primary purpose of this stage is to eliminate improbable matches earlier, thus reducing computational complexity and reducing the processing time in the proceeding stages. This stage was implemented for efficiency which is especially important when the interface must deal with many in-game intents at any given point in the gameplay (i.e. in larger, more complex games). There are several methods to measure similarities and the chosen metric chosen must o provide a fast but accurate measure. Some examples include:

- Jaccard Similarity
- K-means Embeddings
- Cosine Similarity
- Latent Semantic Indexing
- Word2Vec
- BERT Embeddings

Initially, BERT and Cosine Similarity were explored; however, whilst they provided accurate results, both methods required the use of an additional external API, which caused a detriment to performance. Therefore, Magnitude was used. Magnitude provides a compressed version of Word2vec, designed for speed and efficiency.

A threshold value must be determined to govern what game intents can be considered similar enough and care must be taken to ensure the correct intent is not discarded. This threshold value was initially determined by trial and error and later refined using mock testing and the beta program. By eliminating improbable matches, the interface no longer has to apply SRL or semantic similarity to these intents at future stages. To enhance the performance, the interface only considers the top three highest-ranking sentences. This value can be modified by the developer. In addition to this, the calculated similarity score is also used to inform the probability of a match for future stages.

Magnitude requires an embedding model for vectors to be generated (based on a corpus). The following embedded models that were supported by Magnitude were explored:

- Google - Word2vec
- Stanford - GloVe
- Facebook - FastText
- AllenNLP - ELMo
- Google - Bert

For its simplicity, the 300-dimensional version of Google Word2vec was chosen. The algorithm first generates a vector word embedding for each sentence in the known game commands using the corpus provided and also a vector for the sentence provided by the user. The sentence embedding for a command is generated by taking the component-wide average of words in the sentence. A vector is generated for each word within the sentence and they are summed and averaged. Thus, the sentence embedding will be the same size as a word embedding.

The algorithm then calculates the cosine similarity between each game vector and the user input using the following equation:

$$similarity(a, b) = \cos\theta = \frac{a \cdot b}{|a| \cdot |b|}$$

This generates a similarity score between zero and one. Values closer to one indicate that the two vectors are similar, whereas values close to zero indicate the vectors are dissimilar. These similarities are validated against the minimum similarity threshold and then they are sorted. The most similar commands are returned. The number of commands returned at this stage can also be calibrated by the developer.

As discussed in Section 6, the performance of the pipeline is compared with and without this stage to gauge the performance-accuracy tradeoff when the interface is used for a larger game with more in-game commands.

Stage 8: Intent Extraction

This stage attempts to discover the key components of a sentence, namely the verb, the subject noun and the object nouns. It receives both the user-inputted intent and the in-game intents. The interface could simply detect the verb and the two nouns using a part-of-speech tagger; however, this would not allow the interface to distinguish between a direct and indirect object. There are several methods that can be used to achieve this:

- POS-tagging
- Dependency Parsing
- Semantic Role Labeling
- Open Information Extraction

For this project, while there are viable alternatives, Semantic Role Labelling is used. Whilst using Open Information Extraction and Dependency Parsing would achieve the same goal, they also have additional functionalities to provide further information that is not used for this project. This is inefficient and can increase the average response time. For this reason, SRL was determined to be most suited to this project.

AllenNLP's Semantic Role Labelling is an external API to detect the verbs and their corresponding direct and indirect objects. This is applied to both the user-inputted intent and later the in-game intents provided by the developer. SRL is applied to the game intents at this later stage because there may be multiple game intents and therefore the interface will attempt to short-circuit the pipeline as much as possible for efficiency.

Stage 9: Semantic Similarity

In this stage, the interface takes the components from the previous stage and calculates the semantic similarity between the user-inputted component and components of known commands. The semantic similarity measure effectively calculates the relatedness by considering the depths of the two synsets within the WordNet database. Thus, it accounts for hyponyms, hypernyms and synonyms as required. Synonyms will have a score of 1.0 and hypernyms and hyponyms will have scores ranging from zero to one depending on their depths.

There are several semantic similarity methods to choose from:

- Lin Similarity
- Wu-Palmer Similarity
- Resnik Similarity
- Lesk Similarity
- LCH Similarity

The Lin and Wu-Palmer methods were evaluated to see which delivers the fastest and most accurate response. For speed, Wu-Palmer is determined to be the most optimal as outlined in Section 6.

To calculate the similarity between intents, the interface calculates the semantic similarity between each of the three components: verb, direct noun and indirect noun. Calculating the semantic similarity between two words requires the use of NLTK's pre-built functions for calculating similarity using each of the methods. However, the pre-built functions can only compare word senses. Thus, the algorithm retrieves all the word senses associated with a word using the part-of-speech (e.g. noun or verb) as a filter. It then calculates the similarity between each of these senses and picks the highest-ranking score. This has a detrimental effect on performance and so this function is used minimally.

Stage 10: Mapping and Output

The interface now possesses a short list of known commands, each assigned a probability representing the likelihood that this is the command the user intended. In stage 10, the interface must determine if there is a successful match or if there is a potential ambiguity. This is done by sorting the list of commands by their probabilities. The interface takes the two highest-ranking commands and calculates the difference between their probabilities. If the difference is less than a certain value then the interface considers both sentences to be potential matches and thus sends a response with the status set to "ambiguous".

If there is no ambiguity, then the highest-ranking command is returned to the game in the form of a JSON response. A response typically contains: a status code, verb, direct object, indirect object and an error message. Figure 5.6 shows a typical successful response made by Intent.

5.1.5 Response Codes

The following table displays a list of status codes, their definitions and their meanings with respect to the game. Status codes between 0 and 4 are considered successes and status codes between 5 and 14 are considered failed mappings.

Code	*Status*	*Meaning*
0	SUCCESS_GENERAL	A single match was successfully found within the list of game intents provided.
1	SUCCESS_CACHE	A single match was successfully found within the cache.
2	SUCCESS_FETCH	The interface successfully detected a fetch command.
3	SUCCESS_DROP	The interface successfully detected a drop command.
4	SUCCESS_EXAMINE	The interface successfully detected an examine command.
5	FAIL_INVALID_USER_INPUT	The user inputted sentence is empty or invalid.
6	FAIL_INVALID_COMMAND_LIST	The command list is empty.
7	FAIL_SENTENCE_SIMILARITY	The user input was not considered similar enough to any of the game intents provided.
8	FAIL_COMPONENT_SIMILARITY	One or more components provided by the user were not semantically similar enough.
9	FAIL_TOTAL_SIMILARITY	The maximal total probability calculated was not considered high enough to be a match.
10	FAIL_AMBIGUITY	Multiple matches were detected thus there is ambiguity.
11	FAIL_ITEM_ALREADY_IN_INVENTORY	The user has tried to fetch an item that is already within its current inventory.
12	FAIL_ITEM_NOT_IN_INVENTORY	The user has tried to utilise an item they do not currently possess.
13	FAIL_OBJECT_UNKNOWN	The object provided was not within the inventory or the room.
14	FAIL_ENTITY_DETECTED	The user tried to interact with an entity in a way that isn't possible (e.g. fetch, drop)

Table 5.1: A table displaying the Intent response codes and their definitions.

5.1.6 Thresholds and Calibration

Several threshold values must be determined and fine-tuned to enable the best possible matching as well as reduce the possibility of incorrectly matching and not matching at all. The *constant.py* module contains a list of constants and their values. Each of these constants can be calibrated to provide the best possible performance. The default parameters

are those that were calibrated and found to be optimal during the beta program.

Constant	*Definition*	*Value*
Number of components	The number of components expected within a single command (e.g verb, direct noun, indirect noun).	3
Minimum Component Similarity	The minimum similarity score required between a game component and the component. If the semantic similarity score calculated is greater than this value the interface can consider two words semantically similar	0.8
Minimum Total Similarity	The minimum overall similarity score required between a game intent and the user input. If the total similarity exceeds this value the interface can consider this intent as a potential match.	0.7
Word2vec Minimum Similarity	The minimum similarity score between the user input and the in-game intent for the intent to not be discarded. Any known commands which have a similarity lower than this value will no longer be considered by the algorithm.	0.85
Word2vec Similar Enough	If the in-game intent scores a Word2vec similarity score higher than this value then it can be considered a suitable match. This enables short-circuiting of the pipeline.	0.98
Ambiguity Threshold	If the two highest-ranking in-game intents are within this value of each other, the input can be considered ambiguous.	0.05
Object Similarity Threshold	How similar two objects need to be considered the same.	0.9
Similarity Method	The similarity method (WUP, JCN, LIN) chosen by the developer.	WUP

Table 5.2: A table displaying the Intent constants and their default definitions.

5.1.7 Handling Ambiguity

As described in Stage 10, the interface attempts to detect an ambiguity. If an ambiguity is detected it sends a JSON response with the status code corresponding to an ambiguous input. The JSON response also contains fields with the two most probable matches.

```
1  {
2      "sentence": "fight the animal with a weapon",
3      "mapping": "attack the dog with the sword"
4  }
```

Figure 5.3: A POST request containing a sentence and a mapping made to the ambiguity route of the API.

The developer is required to implement code to allow the user to select their intended meaning from the two options. From here, Intent has another REST API route which allows a mapping to be added to the database via a POST request. This mapping is added to the database and for future requests, after sentence segmentation, the interface will

interrogate the database to determine if a mapping already exists. This should allow for short-circuiting as well as preventing the pipeline from failing.

5.1.8 Handling Feedback

Intent also allows for the submission of binary feedback and stores metrics for the responses it sends out in a table. The feedback model accepts a boolean value for whether the match was correct or incorrect. This is useful for calibration as well as generating user metrics for the game. For example, if the number of incorrect matches is too high, the developer might want to increase the similarity thresholds. Another example is if the number of unknown object responses is too high, the developer might want to reduce the object similarity threshold. This can also open up the opportunity to automate calibration by incrementing/decrementing threshold values according to the feedback data collected.

5.1.9 Efficiency Considerations

Intent is required to provide a reasonably fast response time. For this to occur, processing must complete as fast as possible. Thus, it is important to short-circuit the pipeline when possible. Several methods are utilised to achieve this:

- The input is validated at the start to short-circuit the pipeline as soon as possible if there is an invalid input.
- Sentence Segmentation and Coreference Resolution is only applied if certain keywords are found within the user-inputted sentence.
- Cache Inspection is added specifically for short-circuiting. Since there are a limited number of possible commands within a game, it is likely users will attempt to input similar statements. Rather than spend time progressing through the entire pipeline repeatedly, if the statement and its match is stored in the cache then this can reduce complexity.
- It is common within text adventure games for the user to perform the fetch, drop and examine commands. Operating under this knowledge, the object assignment stage attempts to handle these common commands first as well as handle invalid commands. This allows for short-circuiting by handling common commands first.
- Sentence Similarity is a method of reducing the number of commands by discarding the most dissimilar results. Whilst it is not the best metric, it acts as a filter to reduce the number of game intents to be parsed. Section 6 outlines the importance of this stage as the application is scaled.
- If a sentence has a high probability of being a match according to the sentence similarity stage then the pipeline is short-circuited.
- By stage 8, the number of game intents that will need to go through the more time-consuming processes such as semantic role labelling and semantic similarity calculation, will have been reduced significantly.

nice

5.2 Tork - The Mobile Game

The following section describes the implementation of the Tork game in greater detail.

5.2.1 Graphical User Interface

The user interface is designed simplistically as can be seen in Figure 5.4. The main components are a text interface that lists all messages from the game and the user in chronological order. This list is scrollable.

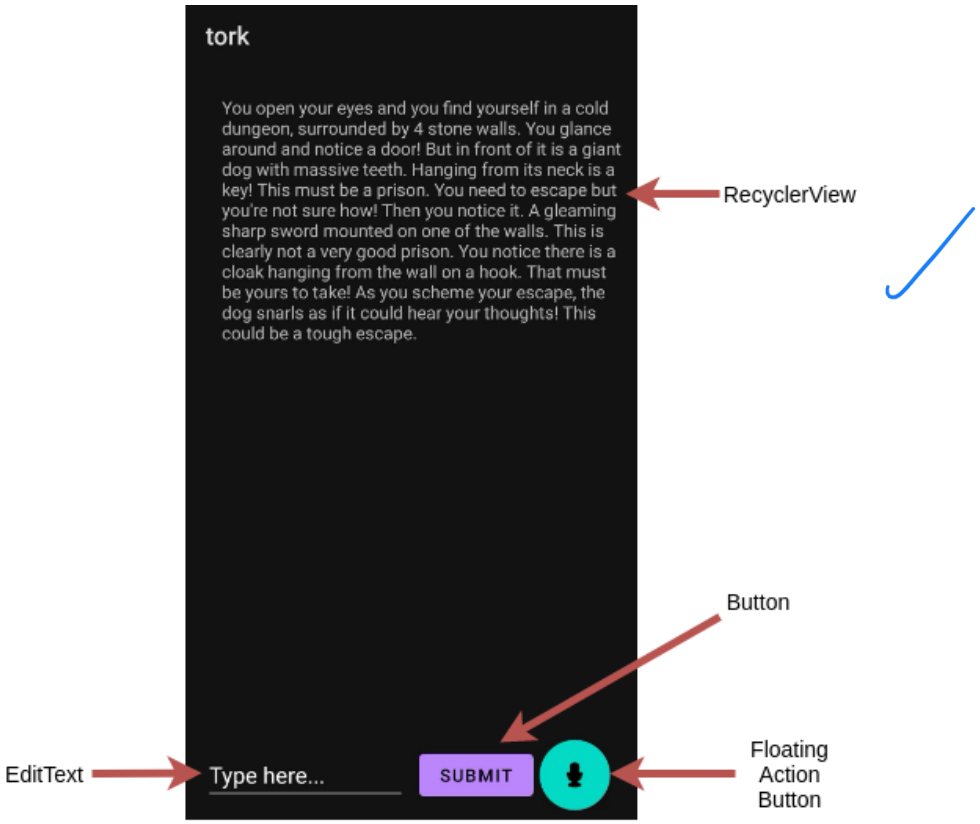


Figure 5.4: The initial graphical user interface designed for Tork.

Some of the main components include:

1. EditText element enabling the user to provide textual keyboard using the keyboard
2. A "Submit" button which sends the textual input to Intent for processing
3. Floating Action Button which enables the microphone for speech recognition
4. RecyclerView which contains TextView elements which acts as a list of messages from the game to the user.

5.2.2 Main Activity

The MainActivity class contains code for handling interaction with the user interface. It contains code to handle speech recognition as well as user input via the text box. It also issues requests to Intent via the IntentRequests class. There are several major components to this class.

Recycler View

The list is implemented as a RecyclerView. The RecyclerView makes it efficient to display large sets of data. The view creates elements when they are needed, recycling the individual elements. Reusing the elements vastly improves performance. In the context of the text-based game, the RecyclerView is the primary interface for the game. All messages sent from the game are presented to the user for example when the player performs an action or if the game changes state. This also includes displaying the items within the inventory to the user.

Input Options

The text-based input is simply implemented by creating an EditText object and linking it to a button which when clicked, sends a request to Intent.

The speech input is implemented using the Android SpeechRecognizer class. The API streams the audio to remote servers to perform the speech recognition. When the floating action button is clicked, the Google Voice Input interface appears on the screen as shown in 5.5. The user can then speak into the microphone and the audio is instantaneously sent to the Google API to create a transcript. The user can easily cancel the recording by clicking off the screen and this action automatically discards any previously recorded result.

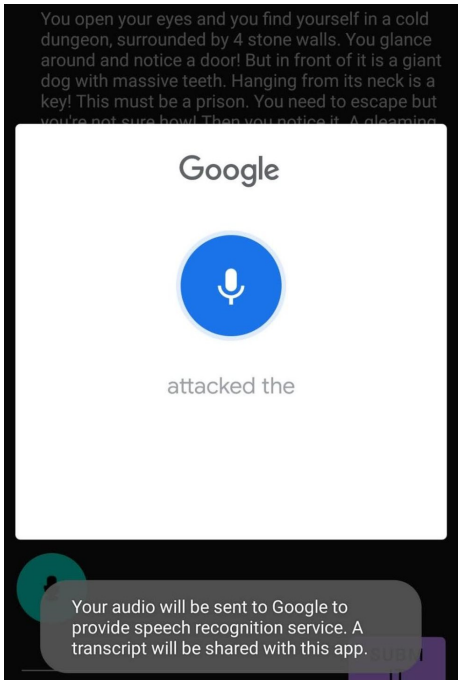


Figure 5.5: A screenshot taken from the Tork application displaying the Google Speech To Text interface.

Intent Requests using Volley

The most crucial component of the mobile application is the way it communicates with Intent. Volley provides a simplistic way to make HTTP requests. This HTTP library makes networking faster which is crucial within a game context. There are three primary functions within the MainActivity class which utilise Volley: sendMapping, sendFeedback and getUserIntent. All of these functions send POST requests to Intent. The function

"sendMapping" is used to send a mapping to be added to the Intent database whereas the function "getUserIntent" is sent to Intent to be processed using NLP. The "sendFeedback" function is utilised for evaluation purposes enabling the user to inform Intent whether the matching was correct or incorrect.

5.2.3 Game Implementation

The implementation of the game can be found within the game package of the repository.

Handling Responses

As previously mentioned, the JSON response from Intent contains the "status" field. This field is an integer field with integers corresponding to the type of response. This information can be found in 5.1. Crucially, each of these status codes must be handled by the developer of the game. Figure 5.6 shows a typical output for a successful command.

Figure 5.6: A response from Intent indicating a successful matching.

If the status code corresponds to a success, the developer should write code to carry out an action within the game. For example, a fetch command would remove an object from the room and add it to the player's inventory whilst simultaneously adding intents that are possible with that object to the list of possible intents. The drop command should do the opposite.
If the status corresponds to a failure, Intent automatically provides an error message for the developer to print to reduce development time. The developer is then required to request a new input from the user if a failure occurs.

Handling Ambiguity

If the status code returned corresponds to "FAILED_AMBIGUITY", the game requests clarification from the user. Tork presents the user with two buttons each representing one of the ambiguous results. The user simply has to clarify what they meant by selecting one of the buttons. The button click triggers a POST request to be sent to Intent. The request will contain the last user input and the selected mapping. The clarification is stored as a mapping within the database for future cache inspections. Once clicked, the button options are then removed.

Handling Feedback

With every response provided by Intent, the game asks the user whether the interface provided a correct match or an incorrect match. It does this in a similar way to how ambiguity is handled. It provides two buttons: correct or incorrect. Upon clicking a

button, the respective boolean value is sent to Intent to increment the relevant metrics.

Game Design Overview

As previously mentioned, the game is very simplistic, focusing on having a robust game design that encourages users to vary their inputs. Text-based adventure gaming offers an unrestricted, free-form interface to allow for this. A story is presented to the user at the start describing the environment and the user must respond with an imperative command. Tork enables both voice and text input for this. The player will interact with objects in its environment. It is important to note that the gameplay requires the user to pick up any object they intend to use. This reduces the number of possible commands in each state. Figure 10.1 shows a UML diagram that was created to display some of the major classes within the game design.

Game Engine

The game engine acts as the focal point for the game. It contains logic to create major components such as the room, the player, the inventory as well as the ability to send game messages to the user. The logic for changing the state of these components is implemented here. For example, the methods "postGameMessage" and "addToInventory" are implemented in this class.

Game Intents

The GameIntent class is the way that the game represents an action within the game. It can also be used to represent the response from Intent. It typically contains fields for the status, verb, direct object, indirect object, error message as well as a sentence. The status and error message fields can be blank as these are used only when the GameIntent represents a response from Intent. The developer is expected to create any initial game intents within the room and GameObjects.

Since one of the primary assumptions of the game is that any action involving two objects will include a single inventory object and a single room object, the interface can simply associate intents with objects. For example, a sword can be used to "attack" an entity. Thus, when the sword is added to the inventory, any intents that are possible with the sword are added to the list of possible intents within the room. It is important to note that at this stage the "dog" in "attack the dog with the sword" may not exist and so this will need to be checked within the action handler for the "attack" command. Note that there is also the opportunity to create intents within the action handler if necessary.

Room, Player and Game Objects

The developer is responsible for initialising the state of the room. This can be done easily by creating the necessary object classes all of which inherit from a superclass called GameObject. GameObjects will also contain a list of intents that are made possible if the player owns the object. For the given room, there are several GameObjects to be created initially:

- Entity extends GameObject
- Weapon extends Sword
- Dog extends Entity (which extends GameObject)
- Sword extends Weapon (which extends GameObject)
- Cloak extends GameObject
- Key extends GameObject

The room must also initialise the initial possible commands such as:

- Fetch the sword from the wall
- Fetch the cloak from the wall.
- Take the apple.

The room contains a set of commands that are currently possible within its current state. The game engine sends processed user intents directly to the room. The room has to validate the intent and pass it to the action handler for gameplay to take place. The player will act as a standalone object and have attributes such as health, an inventory, and a name.

Action Handlers

The room is also associated with an action handler which extends a super action handler. The purpose of the super action handler is to handle actions that can occur at any stage of the game. These include:

- Examine [item]
- Fetch [item]
- Drop [item]

The super action handler is made abstract so as not to be instantiated. The super action handler is only called if Intent responds with a status that corresponds to any of the above commands.

On the other hand, the action handler only handles actions specific to the current room or the object. If Intent responds with a successful match the game receives the verb, indirect and direct objects. This is uniquely matched to a specific in-game command. This can be used to appropriately carry out the action within the game and change the game state accordingly. The developer is required to write the functions corresponding to various actions. This may involve creating new objects, inflicting damage upon entities, removing and adding objects from the inventory and outputting blocks of text to the player. For example, the attack function might:

1. Check if the entity is dead.
2. Check if the entity is confused
3. Reduce the entities health
4. Reduce the player's health

Otherwise, the game knows that there are two nouns within the sentence and so it extracts the names of these objects. Then the action handler attempts to delegate to the correct function based on the verb. Once the verb has been detected, the game has more knowledge about the GameObjects that are required. For example, "attack" accepts an Entity and a Weapon and so the action handler will attempt to retrieve these objects from the room/inventory.

Game Flow

To begin with, the game is designed as a simple room with very few objects and a singular entity. However, it utilises Java's inheritance capabilities to make the game easily extensible as more voice recognition features are added. The following paragraph is the block of text that will be presented to the user to introduce the game in the Dungeon game mode.

"You open your eyes and you find yourself in a cold dungeon, surrounded by 4 stone walls. You glance around and notice a door! But in front of it is a giant dog with massive teeth. Hanging from its neck is a key! This must be a prison. You need to escape but you're not sure how! Then you notice it. A gleaming sharp sword mounted on one of the walls. This is clearly not a very good prison. You notice there is a cloak hanging from the wall right next to the sword. That must be yours to take! As you scheme your escape, the dog snarls as if it can hear your thoughts! This could be a tough escape."

This simple environment allows for a whole host of possible actions. Some obvious examples include:

- Fetch the sword from the wall
- Fetch the cloak from the hook
- Examine the dog

Assuming the user now chooses to "fetch the sword from the wall", this is sent to Intent which parses it and responds, informing the game that this is a fetch command. The action is passed to the super action handler which carries out the fetch command sequence. The sword has intents associated with it and these are added to the list of possible game intents. An example of a possible game intent that is unlocked is to "attack the dog with the sword."

Note that such an action is repeatable since the player has this object within their possession. However, this sequence of commands leads to a state in which the player will take damage. The correct sequence of commands to escape the room is to:

1. Fetch the cloak from the wall.
2. Fetch the sword from the wall.
3. Throw the cloak at the dog.
4. Attack the dog with the sword.
5. Fetch the key from the dog.
6. Walk through the door.

Other intents are possible; however, they will not be encoded as they do not result in any meaningful gameplay. The level of complexity is left to the developer.

5.2.4 Gameplay

Escape The Dungeon

In this game mode, the player is required to escape a dungeon. This is a simple room as described in the above section regarding the Game Implementation. Its primary purpose is to demonstrate the capabilities of the interface in an unfamiliar setting. This is useful because users are likely to input unanticipated commands that may or may not be handled correctly. This game mode will be used as part of the evaluation.

Make The Tea

In this game mode, the player is required to make a cup of tea. This game mode aims to provide a player with a familiar context. This is useful because commands are more predictable and players may be able to think of different ways to phrase common actions. This game mode will also be used as part of the evaluation in determining the success of the project.

Extending The Game

By using notions of inheritance and abstract classes, the game is designed for extensibility. If a developer chose to create a new escape room this can easily be accomplished by creating the room, the necessary objects and an action handler. The Room class contains the necessary functions to easily add, remove and retrieve room objects, entities and other important details. Overall, Tork acts as a model for escape room games that interact with the Intent interface.

6 Evaluation

The project was designed using a rapid application development approach. This consisted of two-week development cycles in which new features were added and evaluated. The initial pipeline consisted of only four stages as seen in Figure 6.1, however, as features were added this was expanded into a ten stage pipeline.

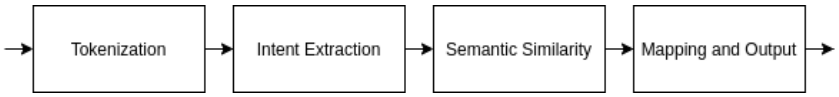


Figure 6.1: A diagram displaying one of the initial pipelines.

6.1 Mock Testing

The primary method of evaluating how well the interface performs will be to utilise the Django Test Framework. As mentioned, the development was driven by tests. Some of the main test cases and the order in which features were added are displayed within Table 6.1. Most iterations have been omitted for purposes of keeping the report succinct.

#	*Example Test Case*	*Feature added*	*Implementation*
1	attack the dog with the sword	Basic validation and exact matching	Aims to parse an exact match.
2	assault the hound with the blade	Generating synonyms from WordNet	Aims to find synonyms of the verb, direct and indirect objects.
3	Charge the puppy with the falchion	Generating hyponyms from WordNet	Aims to find hyponyms of the verb, direct and indirect object.
4	Attack the canine with the weapon	Generating hypernyms from WordNet	Aims hypernyms of the verb, direct and indirect object.
5	Take the apple and eat the apple	Sentence Segmentation	Attempts to segment the sentence and process each one.
6	Take the apple and eat it	Coreference Resolution	Replaces references with the correct object.
7	Fight the dog	Sentence Similarity	Magnitude was a lot faster. Sentence similarity is applied to any other stages to reduce the number of intents significantly.
8	Throw the sword at the dog	Ambiguous command requiring Cache Inspection	The database was created enabling ambiguous commands to be clarified.
9	Fetch the sword	Object Assignment with a single object	Reduces the amount of work the developer is required to do.

Table 6.1: A table displaying the additions to the pipeline used in the iterative development process.

6.2 Evaluating Semantic Similarity Methods

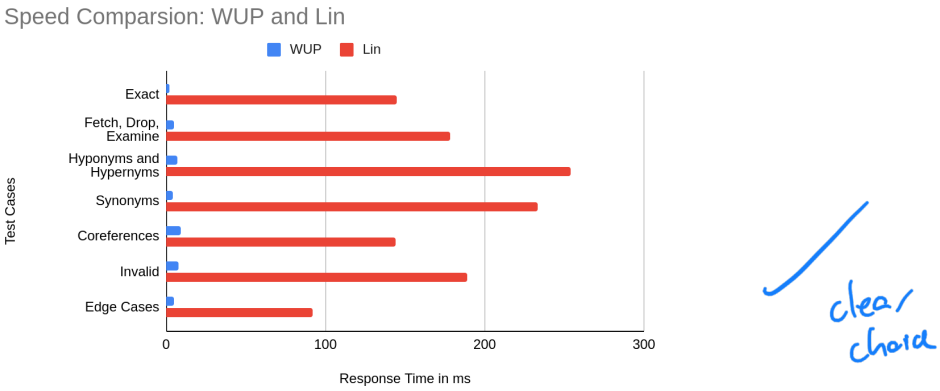


Figure 6.2: A bar chart displaying the response time for each set of test cases using the WUP and LIN similarity methods.

Figure 6.2 shows that the WUP calculation approach is substantially faster than the Lin technique, to the point where Lin would not be a feasible option in a game where a quick response time is critical. In comparison to the WUP approach, the Lin approach performed poorly in terms of response time.

6.3 Beta Testing

The application was beta tested over a period of four weeks with eight participants who acted as players of the Tork game that utilises Intent. The software archives were also provided to a group of eight Android developers. During the beta program all user input was recorded, user metrics were generated and qualitative feedback was collected from players and developers reviewing the interface and the game.

6.3.1 User Metrics

Correct and Incorrect Matches

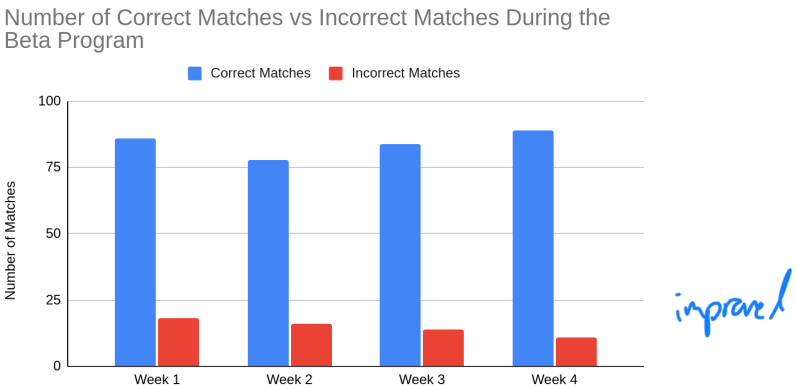


Figure 6.3: A bar chart displaying the proportion of correct to incorrect matches during the beta program.

To collect this metric, a feature was developed for the Tork application where, in addition to every response provided by Intent, the user is asked to provide feedback on whether the detected match was correct or not. Figure 6.4 shows how two buttons are presented to the user indicating correct and incorrect choices. Once clicked the button disappears and the feedback is recorded and saved in a database by Intent. This metric can be used to inform the threshold values of the interface. Throughout the beta program, the calibration values were manually tweaked to minimise the number of incorrect matches. As can be seen from the bar chart, the general trend is an increase in correct matches and a decrease in incorrect matches. The number of correct matches far exceeds the number of incorrect matches by Week 4, indicating that the Intent matches correspond with the input.

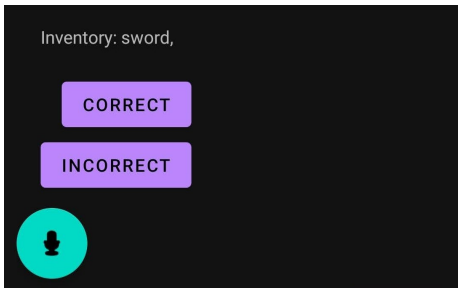


Figure 6.4: A screenshot taken from Tork application displaying the feedback mechanism.

Response Types

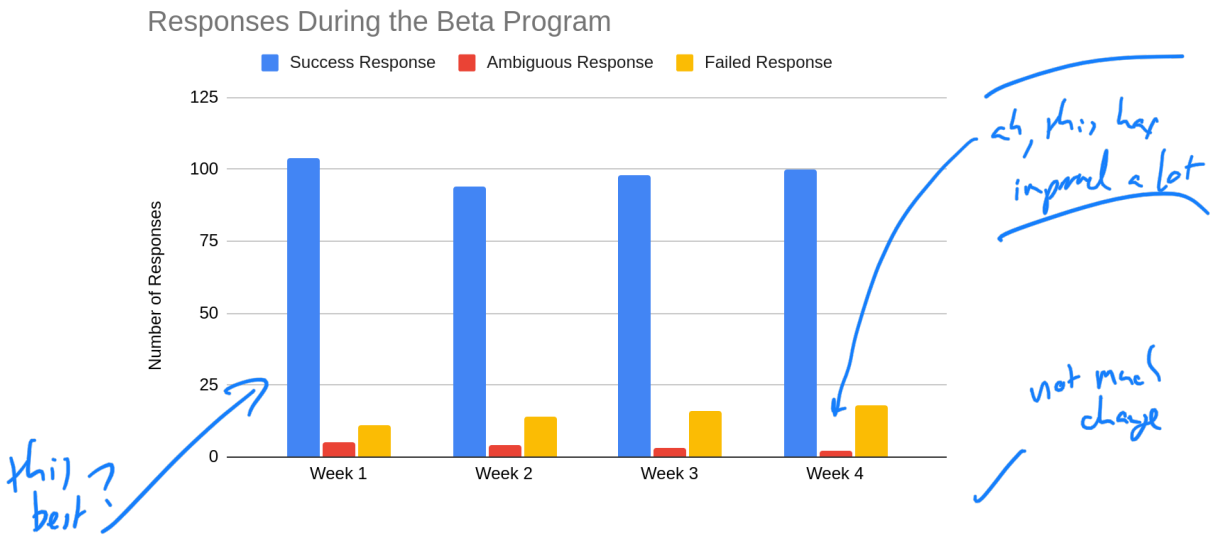


Figure 6.5: A bar chart showing the various responses made by Intent during the program.

Another metric that was collected was the various responses made during the beta program (as seen in Figure 6.5). This metric helps inform the threshold values as well as provide an overall evaluation of the interface. If there are too many failed responses, this indicates that threshold values are too high. If there are too many ambiguous responses, this indicates that the ambiguity threshold is too high. The calibration values were manually corrected so that by Week 4, the number of ambiguous responses was lowered significantly. This also indicates that the cache inspection stage was useful in reducing the number of ambiguities.

How Scaling Affects The Response Time

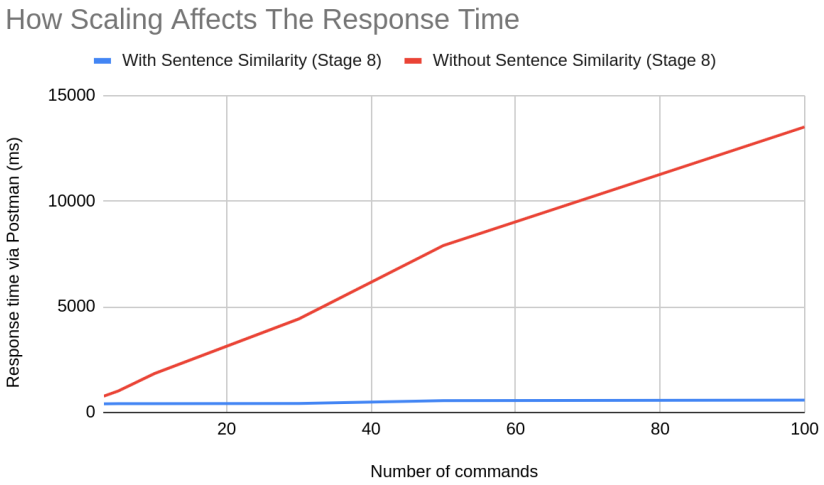


Figure 6.6: A line graph showing the response time for various number of commands with and without Stage 8.

An important metric that was collected was how the application scaled to larger games with larger rooms (as seen in Figure 6.6). The number of commands ranged from 3 to 100 and the response time was recorded via Postman. Postman is an API development tool that can be used to send HTTP requests and record response times. The response time was tested with and without the sentence similarity stage to underline its importance in making the application scalable. Without the sentence similarity stage, the response time grows linearly as the number of possible in-game commands increases. However, when sentence similarity is applied, a constant response time can be expected. There were slight increases in response time as sentence similarity has to be calculated for each in-game command. However, overall the sentence similarity stage proves to be crucial for scaling. The response time should be under one second for the user to not feel an excessive delay in response from the game. Ideally, the system should provide a response time under 0.1s for the user to feel the game responds instantaneously [64]. The project fulfils the initial objective of having a fast processing time.

6.3.2 Freedom of Expression

One of the primary objectives of this project is to maximise the freedom of expression a player has. Players were asked about how well Intent was able to handle their commands.

"Overall the system handled my commands very well. One of the main restrictions was the size of the room and what commands had been encoded within the game. For a small room, Intent recognised nearly all my commands, however, for larger rooms, not all possible actions had been covered by the developer and so the game felt restricted." - Player 1

As a result, it may be concluded that Intent maximises freedom of speech, although this is severely constrained by the behaviours that the game's developer has programmed into the game.

6.3.3 Developer Feedback

In addition to this quantitative analysis, eight Android developers were asked to create an application that utilised the Intent interface and their feedback was recorded in a survey.

"The interface works quite well when used in a medium-sized room with multiple entities and up to fifteen objects. However, for our use case, we had to spend a long time calibrating and recalibrating the constants to optimise the performance of the interface and reduce false positives. Ideally, this calibration process should be automated. Overall, it greatly reduced development time." - Android Developer 1

"Development was incredibly simple following the pattern provided by Tork and I was able to easily extend the game to a different context. The game scales well and provides fast responses; however, the performance became slower when more than twenty objects were added to a room." - Android Developer 2

The feedback received is positive and users did not find any issues with the way that Intent finds matches. Importantly, the developers found that Intent greatly reduced the development time, thus fulfilling the primary objective of this project. Several weaknesses were identified by the developers including manual calibration being time-consuming and performance slowing down for large sets of objects; these are explored further in the following section.

6.4 Limitations

Whilst the NLP interface has many features and meets expectations, several limitations and their possible effects on the use of Intent and Tork are identified and explored in this section. Solutions to these limitations are explored in Section 7.

6.4.1 Performance and Bottlenecks

As can be seen, by the metrics, performance can be improved. Whilst steps were taken to ensure that the interface processed sentences as efficiently as possible with several short-circuiting opportunities utilised, the overall system takes an average of 0.15 seconds for a response in the Dungeon game mode and an average of 0.2 seconds in the Kitchen game mode. The overall speed is below the expected standard for a real-time video game; however, for a text-adventure game, the speed is sufficient to not hinder gameplay [64]. Whilst the performance speed is sufficient and timely for a mobile phone game, to extend the application to real-time video gameplay, the speed would have to be improved further.

As with any linear pipeline, there are bottlenecks to performance. Efforts were made to ensure time-consuming processes occurred infrequently and towards the end of the pipeline to allow for the short-circuiting approach. The main bottleneck for the pipeline is the function calls to calculate the semantic similarity between two words. This occurs in the object assignment stage as well as the semantic similarity stage. The reason this is a bottleneck is because a single word can have multiple word senses within the WordNet database. The interface is unable to distinguish the sense of a particular word and so, as part of calculating the semantic similarity, it generates all senses of both words and compares each pair. This is an $O(N^2)$ algorithm that is detrimental to performance. For example, the word "bat" can refer to a winged animal or an implement with a handle. Whilst WordNet provides word sense disambiguation which accounts for the context of the

sentence, this is often slow and inaccurate and thus was deemed unsuitable for the task.

6.4.2 Manual Calibration

Currently, the interface is heavily dependent on accurate calibration. There are several "constants" that must be manually calibrated. These may differ between themes within the game and may not always lead to the intended result. Whilst efforts were made to pick good initial values, the developer is required to adjust the various thresholds which requires trial and error. A viable solution to this is to automate this calibration; this is discussed further in Section 7.

6.4.3 Irregular Words

In its current state, the interface is unable to detect compound words such as "lash out" and "ice cream. These words exist within the WordNet database; however, they are detected as separate words rather than a single compound word. Semantic Role Labeling will not understand the verb "lash out" as a single word. This limits the user input to non-compound words.

6.4.4 Input Assumptions

The overall interface is greatly constrained by its assumptions. For example, the assumption that a sentence will only contain at most two nouns is not ideal. A user may input "attack the dog with the sword and shield" which contains 3 different game objects. Intent is not currently able to handle this command.

7 Conclusion

This section summarises the report's conclusions and highlights the project's deliverables and contributions.

7.1 Deliverables

As previously mentioned, the project is split into two distinct subprojects; Tork, the mobile game and Intent, the NLP interface. Both have been developed successfully.

A prototype for a text-based escape-room game was developed for Android with a primary form of input as voice recognition. The game provides an easy-to-follow format of a classical text-adventure game which makes it easily extensible for the average developer with basic Java experience. This template directly interacts with the Intent interface.
The Intent interface is a readily accessible REST API that gathers several state-of-the-art NLP technologies and unites them seamlessly to serve a text-adventure game designed on any platform. As evident from the development of Tork, the interface greatly reduces the development efforts. The interface eliminates the need for any hard-coding or grammars. The developer is only required to track entities, room objects, inventory objects, possible game commands and the user input at any given stage.

The interface can handle meaningful user input using optimistic matching to help determine the in-game command. Users can use synonyms, hyponyms, hypernyms as well as similar words in their inputs. The interface is also capable of handling common commands reducing development time for the game. Some key achievements of the Intent interface are:

- Intent's unique feature is the use of an optimistic matching algorithm to determine the most probable match.
- Synonyms, hypernyms and hyponyms from user input are individually dealt with, along with similar words and phrases.
- Intent also resolves pronoun references such as 'it', 'him' and 'her' within the sentence.
- The interface orders and handles multiple commands in the appropriate sequence.
- Intent handles ambiguity between multiple in-game commands by requesting clarification and caching the response for future reference.
- The interface can short-circuit common commands such as "fetch", "drop" and "examine" to reduce development time.
- Intent handles violations of the assumptions (such as if a player does not possess an object) as invalid commands and provides a prompt to the user for clarification.
- The interface receives feedback that enables developers to produce a more granular calibration.
- The interface utilises several short-circuiting methods to provide the fastest possible response.
- Intent scales to larger rooms with a greater number of in-game commands.

Despite some limitations to the system as outlined in Section 6, this provides a strong foundation to further the development of the NLP interface, expanding it to handle a greater set of commands from users. The NLP interface is focused on text-adventure games however, the concept can easily be generalised to other genres of video games as well as other applications such as virtual personal assistants like Siri.

7.2 Contributions

As mentioned, there is a niche within the market for intelligent text-adventure games. Games are often restricted to exact commands or grammars. The Intent interface proves that input to text-adventure games can be more freely formed rather than having to conform to set rules. It expands the very small set of commands that are understood by the game into a much larger set of commands. Further to this, the Intent interface has been made scalable to larger text-adventure games. Importantly it has been made extensible so that the pipeline can be adjusted and additional stages can be implemented.

In addition to this, Tork provides a meaningful template for implementing the Intent interface. The game acts as a use case for Intent to prove its functionality in a text-adventure game context along with voice recognition capabilities.

7.3 Future Development

There are many ways that the NLP interface can be improved by utilising different NLP technologies. These are listed from the most easily achievable to the least easily achievable.

7.3.1 Automated Calibration

As mentioned, one of the limitations of the project is the requirement for the developer to manually calibrate the thresholds which may change in various rooms. However, there is the opportunity for automating the calibration by adjusting the "constants" with every response or with feedback from the user. For example, if the user responded with an incorrect matching, the threshold values can be lowered by a small value. With a large number of inputs, the values will converge to their optimal values.

7.3.2 Compound Words

Intent currently does not support compound words such as "pick up" or "lash out" and similarly for nouns, such as "ice cream". Compound words are supported by WordNet however they are not currently detected by Intent. For example "lash out" is a synonym for "attack".

7.3.3 Named Entity Recognition

Another NLP technology that could be useful is Named Entity Recognition. This will allow the game to give names to entities. For example, a troll could be named "Frank" and when the user inputs "attack Frank", this would translate to "attack the troll". Named Entity Recognition provided as an API by AllenNLP.

7.3.4 Sentiment Analysis

Another useful NLP technology that can be explored is the use of sentiment analysis. This allows developers to add more features to their game. For example, the game might ask the user to say something pleasant to the guard to escape the room. This requires sentiment analysis to detect the sentiment of the user input.

7.3.5 Improving Performance with Contextual Word Embeddings

As previously mentioned, a major bottleneck for the interface is not being able to disambiguate the sense of a word within the WordNet database. A way of dealing with this is to use contextual word embeddings before searching the WordNet database. Google provides

BERT which allows for the context of the sentence to be detected. Whilst the interface attempted to implement this, it caused a major detriment in performance. A viable alternative would be to use AllenNLP's ELMo embeddings which also allow for contextual word embeddings. This can help find the correct sense of the word and thus reduce the time spent generating and handling all the senses of a word.

7.3.6 Affordant Extraction

The interface could be improved by providing more support for developers. In particular, there may be actions that the developer has not encoded. The interface may be able to inform the developer of this using a method called Affordant Extraction [65]. A list of affordant verbs for a noun is the subset of possible actions that are reasonable in the given situation. For example, if the player owns a sword then "attack" would be a suggested action for the developer to encode.

7.3.7 Visualisation of a text-adventure game

A text-adventure game can be represented as a graph consisting of nodes and edges which represent states and actions respectively. It could be extremely useful to developers to help them consider various state transitions they may not have considered.

7.3.8 Hint Generation

As part of a text-adventure game, players are often provided with hints to help them successfully escape the room. Currently, hints have to be outputted to the player manually by the developer. However, there is the possibility of providing a hint generation system with the correct sequence of commands to escape the room. This can then be used to generate hints at various stages of the gameplay.

8 Ethical Considerations

There are several ethical issues associated with this project that have been considered. Crucially this application ensures compliance with GDPR since the application target market is within the UK [66].

GDPR states that users should have control over how their data is used and thus before installation the user must provide explicit permission for the application to access the mobile's microphone. The application also explicitly states how the data will be used. GDPR states that data should not be stored for longer than is necessary. For this reason, after Intent has processed the user input, the result is discarded and not stored. The speech recognition API is provided by Google which states that Google "collects and stores a copy of the voice input temporarily and securely". The Google API also follows guidelines set out by the GDPR. Moreover, when recording their voice, the user has the option to cancel the recording so as not to send data to Intent. As part of the pipeline, an AllenNLP API is used. AllenNLP also conforms to GDPR as outlined in their privacy policy [67].

The security of users data is paramount as the project is processing anything that the user says. This may unintentionally include personal or sensitive information. Data must be encrypted so criminals cannot sniff the packets and the REST API must process it confidentially. Volley encrypts data before it is sent to Intent. The project ensured the use of technologies that hold a high level of security, to counteract any potential malicious use or unauthenticated access to user data. Each of these API's enforce strong security policies in line with GDPR. The software will rely on some open-source libraries such as WordNet, NLTK and AllenNLP. These libraries have open-source copyright licenses for commercial and research purposes which are provided to developers for usage.

Accessibility is an important factor to consider when creating a game dependent on voice recognition. This may not be accessible for people with speech impediments. For this reason, in addition to voice recognition, the app will allow for textual user input. The technology currently only works for English-speaking users. Moreover, voice recognition will not be as accurate for people with various accents and so they will have less success using the application.

Another ethical issue is associated with developing and testing software using real users. As part of the evaluation process, a beta program was created to collect metrics and gain feedback. To ensure the beta program was run ethically, there is a strong emphasis on fairness, anonymity and data security. Users were picked at random and anonymity was preserved.

9 Bibliography

[1] Wikipedia. Kinect; 2021. Available from: https://en.wikipedia.org/w/index.php?title=Kinect&oldid=998431358.

[2] van der Velde N. Voice controlled games: The rise of speech technology in gaming; 2017. Available from: https://www.globalme.net/blog/voice-controlled-games/.

[3] Wikipedia. Speech recognition; 2021. Available from: https://en.wikipedia.org/w/index.php?title=Speech_recognition&oldid=998560293.

[4] Google Speech To Text;. Available from: https://cloud.google.com/speech-to-text.

[5] Wikipedia. Natural language processing; 2021. Available from: https://en.wikipedia.org/w/index.php?title=Natural_language_processing&oldid=1004005071.

[6] Wikipedia. Escape the room; 2021. Available from: https://en.wikipedia.org/w/index.php?title=Escape_the_room&oldid=999296224.

[7] Wikipedia. Impact of the COVID-19 pandemic on the video game industry; 2021. Available from: https://en.wikipedia.org/w/index.php?title=Impact_of_the_COVID-19_pandemic_on_the_video_game_industry&oldid=1003155134.

[8] There Came An Echo on Steam;. Available from: https://store.steampowered.com/app/319740/There_Came_an_Echo/.

[9] Review by Dan Whitehead C. There Came an Echo review; 2015. Available from: https://www.eurogamer.net/articles/2015-03-02-there-came-an-echo-review.

[10] Wikipedia. Tom Clancy's EndWar; 2021. Available from: https://en.wikipedia.org/w/index.php?title=Tom_Clancy%27s_EndWar&oldid=999125521.

[11] Tom Clancy's EndWar Review;. Available from: https://www.wargamer.com/articles/tom-clancys-endwar/.

[12] Michaes D. Tom Clancy's Endwar. Berkley Books; 2008. Available from: https://www.metacritic.com/game/pc/tom-clancys-endwar.

[13] Wikipedia. In Verbis Virtus; 2020. Available from: https://en.wikipedia.org/w/index.php?title=In_Verbis_Virtus&oldid=977711751.

[14] pocketsphinx 0.1.15;. Available from: https://pypi.org/project/pocketsphinx/.

[15] Ferrari M. In: In Verbis Virtus. Springer International Publishing; 2015. p. 141–144.

[16] Wikipedia. Zork; 2021. Available from: https://en.wikipedia.org/w/index.php?title=Zork&oldid=1003343845.

[17] Zork - Text Adventure Online;. Available from: https://www.hackster.io/devops-dungeoneers/classic-zork-494ff1.

[18] Ajdnik R. Zork: The great inner workings - the Startup - medium; 2020. Available from: https://medium.com/swlh/zork-the-great-inner-workings-b68012952bdc.

[19] Infocom Type Parsers;. Available from: http://ifwiki.org/index.php/Infocom-type_parser.

[20] Zork - The Inner Workings;. Available from: https://mud.co.uk/richard/zork.htm.

[21] Sphynx for Unreal Engine 4;. Available from: https://michaeljcole.github.io/wiki.unrealengine.com/Speech_Recognition_Plugin/.

[22] Core D. How to bring voice to your unity games with houndify voice AI; 2020. Available from: https://medium.com/houndify/bringing-voice-to-your-unity-games-through-houndify-19144570afd7.

[23] thetuvix. Voice input in Unity;. Available from: https://docs.microsoft.com/en-us/windows/mixed-reality/develop/unity/voice-input-in-unity.

[24] Introduction To NLP;. Available from: https://builtin.com/data-science/introduction-nlp.

[25] Garbade MJ, Garbade MJ. A simple introduction to natural Language Processing; 2018. Available from: https://becominghuman.ai/a-simple-introduction-to-natural-language-processing-ea66a1747b32.

[26] Natural Language Processing Simplified;. Available from: https://datascience.foundation/sciencewhitepaper/natural-language-processing-nlp-simplified-a-step-by-step-guide.

[27] Wikipedia. Lexical analysis; 2021. Available from: https://en.wikipedia.org/w/index.php?title=Lexical_analysis&oldid=998644479.

[28] Stemming and Lemmatization;. Available from: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html.

[29] Singh I. Syntactic processing for NLP - towards data science; 2020. Available from: https://towardsdatascience.com/syntactic-processing-for-nlp-e88e2eb4fa35.

[30] Español CK. What are the different levels of NLP? - CK Español - Medium; 2017. Available from: https://medium.com/@CKEspanol/what-are-the-different-levels-of-nlp-how-do-these-integrate-with-information-retrieval-c0de6b9ebf61.

[31] Discourse Integration;. Available from: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781787121423/3/ch03lvl1sec30/discourse-integration.

[32] Pragmatic Analysis;. Available from: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781787121423/3/ch03lvl1sec31/pragmatic-analysis.

[33] Houndify;. Available from: https://www.houndify.com/.

[34] Google Dialogflow;. Available from: https://cloud.google.com/dialogflow/docs.

[35] van der Pluijm E. How to build a Dialogflow powered Escape Room with Google AIY Kit; 2018. Available from: https://medium.com/@erik_81851/how-to-build-a-dialogflow-powered-escape-room-with-google-aiy-kit-1542df2e8203.

[36] Wikipedia. Part-of-speech tagging; 2020. Available from: https://en.wikipedia.org/w/index.php?title=Part-of-speech_tagging&oldid=992379990.

[37] Introduction to POS tagging. freeCodeCamp.org; 2018. Available from: https://www.freecodecamp.org/news/an-introduction-to-part-of-speech-tagging-and-the-hidden-markov-model-953d45338f24/.

[38] Green R. In: WordNet. CRC Press; 2009. p. 5659–5664.

[39] Wikipedia. WordNet; 2020. Available from: https://en.wikipedia.org/w/index.php?title=WordNet&oldid=996474412.

[40] Senses W, Rela-ons W. Word meaning and similarity;. Available from: https://web.stanford.edu/class/cs124/lec/semlec.pdf.

[41] Miller GA. WordNet: A lexical database for English. Communications of the ACM. 1995;38(11):39–41.

[42] Strapparava C, Valitutti A, et al. Wordnet affect: an affective extension of wordnet. In: Lrec. vol. 4. Citeseer; 2004. p. 40.

[43] NLTK WordNet;. Available from: https://www.nltk.org/howto/wordnet.html.

[44] Edmonds P, Agirre E. Word sense disambiguation. Scholarpedia journal. 2008;3(7):4358.

[45] Wikipedia. Lesk algorithm; 2020. Available from: https://en.wikipedia.org/w/index.php?title=Lesk_algorithm&oldid=973918125.

[46] Thanh Dao TS. WordNet-based semantic similarity measurement;. Available from: https://www.codeproject.com/Articles/11835/WordNet-based-semantic-similarity-measurement.

[47] Semantic Similarity Using WordNet Ontology;. Available from: https://medium.com/@pragadesh/semantic-similarity-using-wordnet-ontology-b12219943f23.

[48] NLP Wu-Palmer Similarity using WordNet; 2019. Available from: https://www.geeksforgeeks.org/nlp-wupalmer-wordnet-similarity/.

[49] Budanitsky A, Hirst G. Evaluating WordNet-based measures of lexical semantic relatedness. Computational linguistics (Association for Computational Linguistics). 2006;32(1):13–47.

[50] Brownlee J. What are word embeddings for text?; 2017. Available from: https://machinelearningmastery.com/what-are-word-embeddings/.

[51] Levy O, Goldberg Y. Dependency-based word embeddings. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers); 2014. p. 302–308.

[52] Alammar J. The Illustrated Word2vec;. Available from: https://jalammar.github.io/illustrated-word2vec/.

[53] Devlin J, Chang MW, Lee K, Toutanova K. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:181004805. 2018.

[54] Magnitude by Plasticity;. Available from: https://www.plasticity.ai/.

[55] Kübler S, McDonald R, Nivre J. Dependency parsing. Synthesis lectures on human language technologies. 2009;1(1):1–127.

[56] AllenNLP;. Available from: https://allennlp.org/.

[57] Palmer M, Gildea D, Xue N. Semantic role labeling. Synthesis Lectures on Human Language Technologies. 2010;3(1):1–103.

[58] Fader A, Soderland S, Etzioni O. Identifying relations for open information extraction. In: Proceedings of the 2011 conference on empirical methods in natural language processing; 2011. p. 1535–1545.

[59] Bengtson E, Roth D. Understanding the value of features for coreference resolution. In: Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing; 2008. p. 294–303.

[60] Palmer DD. Tokenisation and sentence segmentation. Handbook of natural language processing. 2000:11–35.

[61] Feldman R. Techniques and applications for sentiment analysis. Communications of the ACM. 2013;56(4):82–89.

[62] ;. Available from: https://omdia.tech.informa.com/.

[63] ;. Available from: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[64] Response time limits: Article by Jakob Nielsen;. Available from: https://www.nngroup.com/articles/response-times-3-important-limits/.

[65] Fulda N, Ricks D, Murdoch B, Wingate D. What can you do with a rock? Affordance extraction via word embeddings. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence. International Joint Conferences on Artificial Intelligence Organization; 2017. .

[66] Goddard M. The EU General Data Protection Regulation (GDPR): European regulation that has a global impact. International Journal of Market Research. 2017;59(6):703–705.

[67] AllenNLP Privacy Policy;. Available from: https://allenai.org/privacy-policy.

10 Appendix

10.1 UML Diagram for Tork

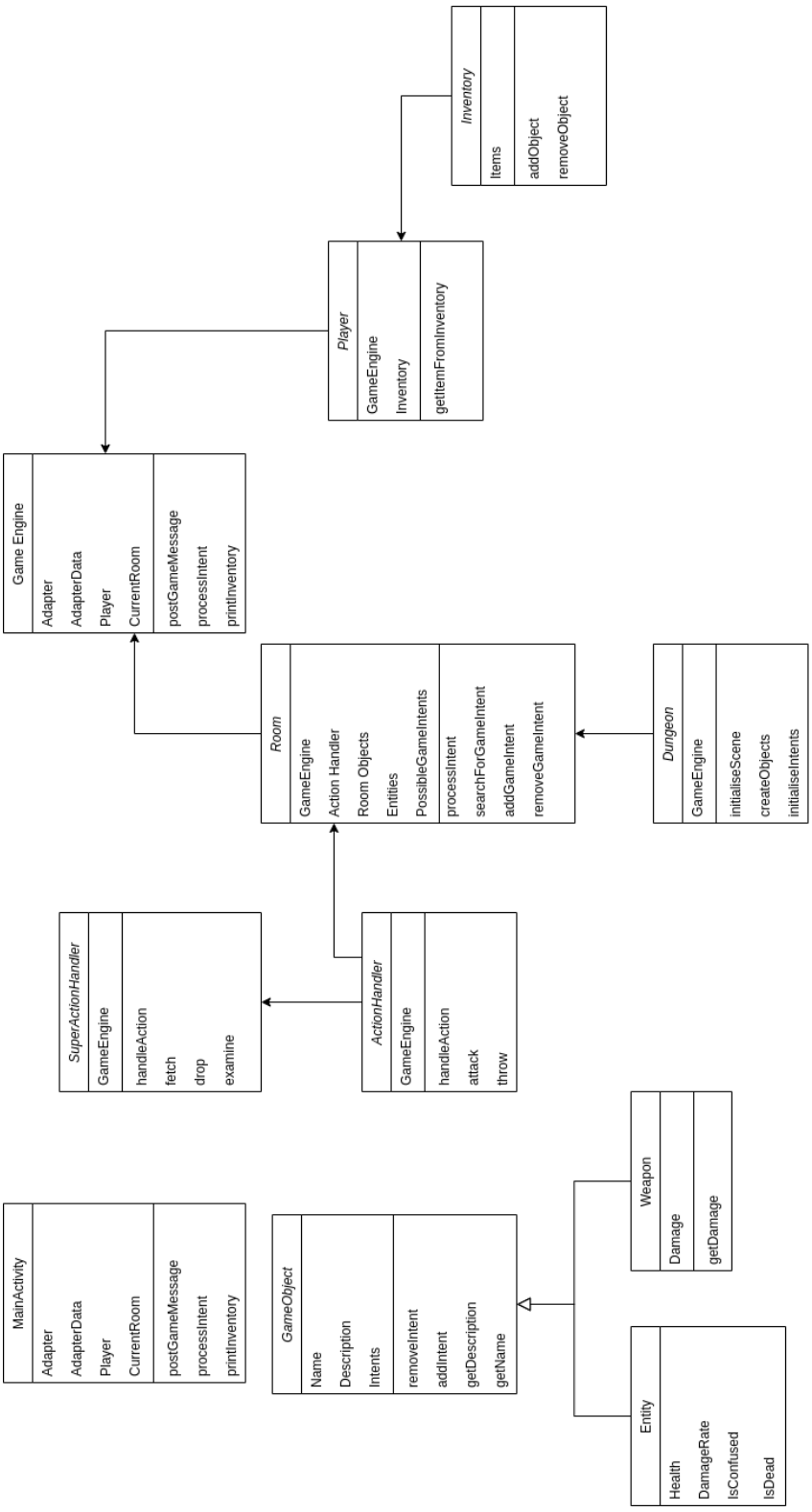
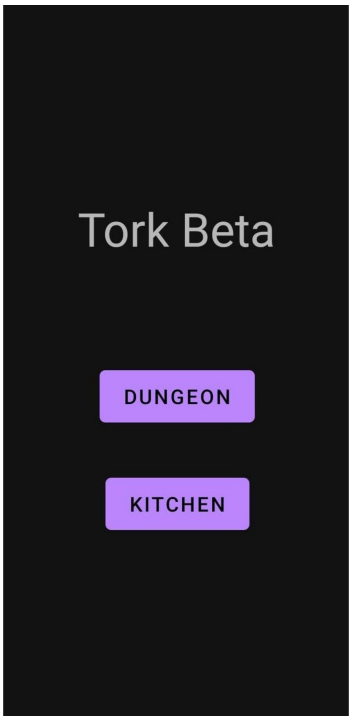


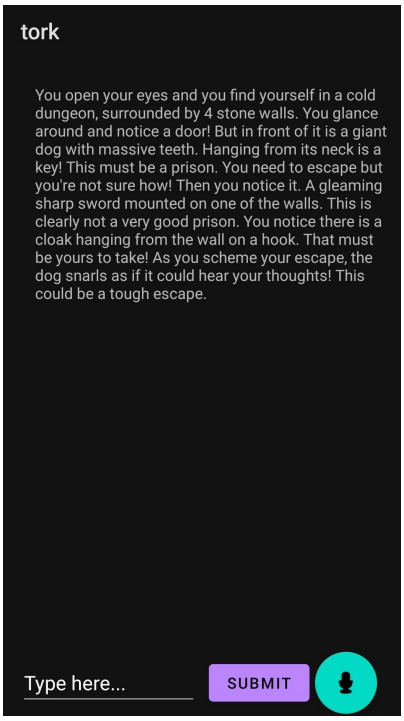
Figure 10.1: A UML Diagram displaying the main components and attributes of the Tork game.

10.2 Screenshots from Tork

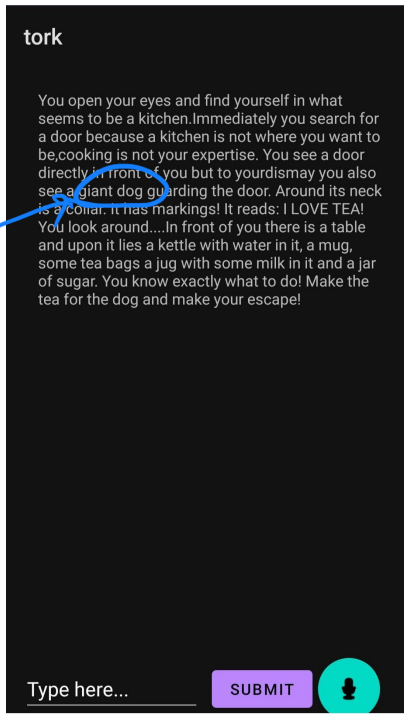
This section contains screenshots taken from the Tork Android Development Application during gameplay.



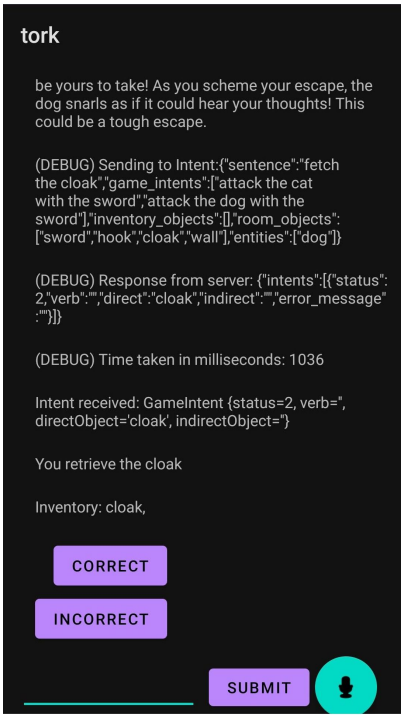
(a) The main menu with the game mode selection.



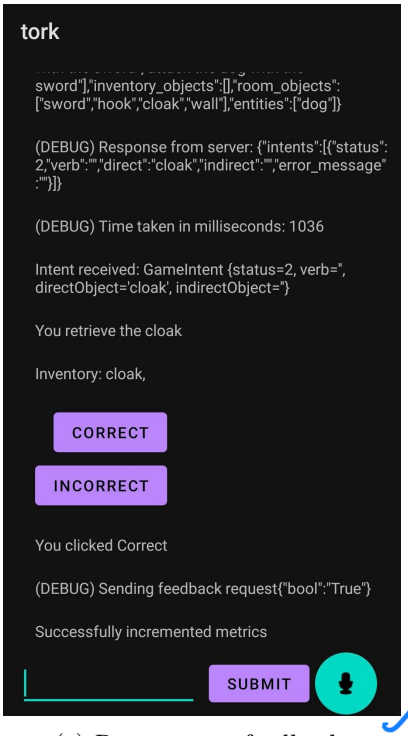
(b) The opening for the Dungeon game mode.



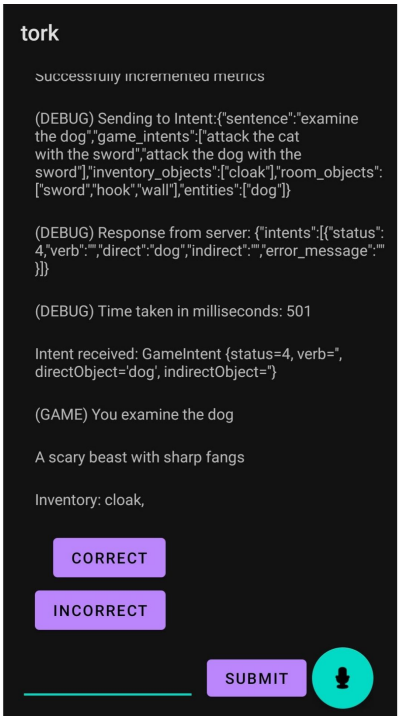
(c) The opening for the Kitchen game mode.



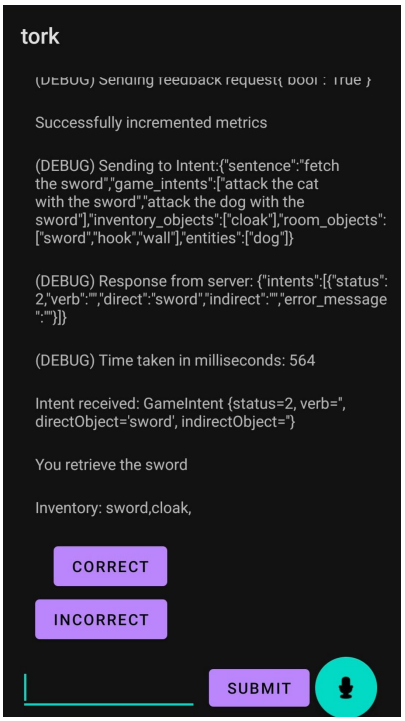
(d) Response to "fetch the cloak."



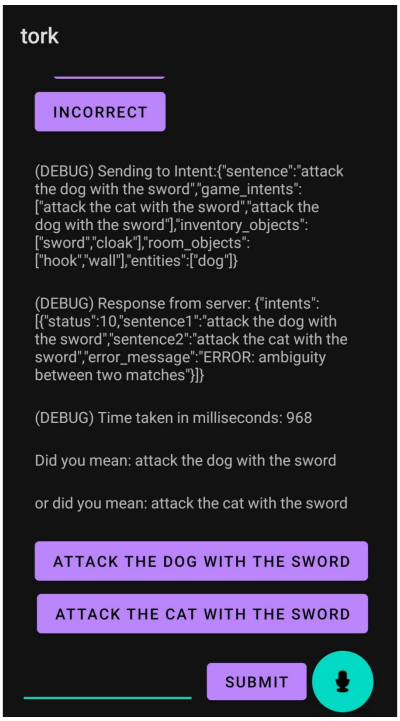
(e) Response to feedback.



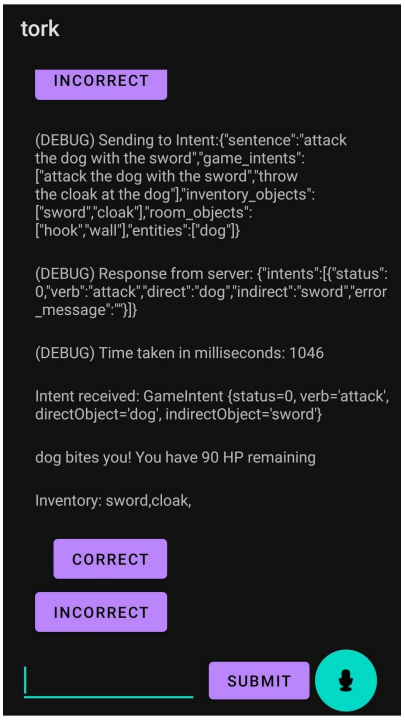
(f) Response to "examine the dog."



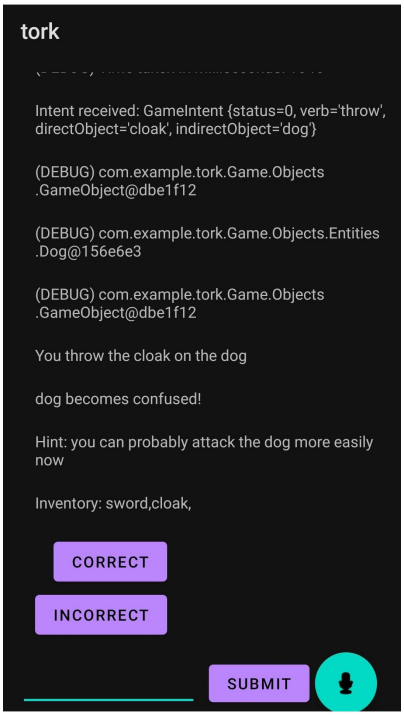
(g) Response to "fetch the sword."



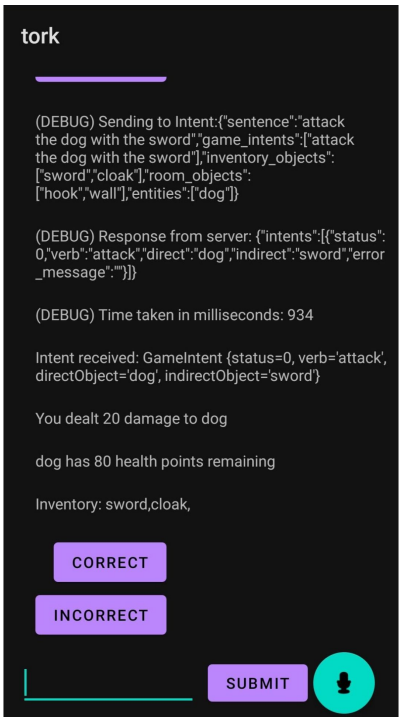
(h) Request for ambiguity clarification.



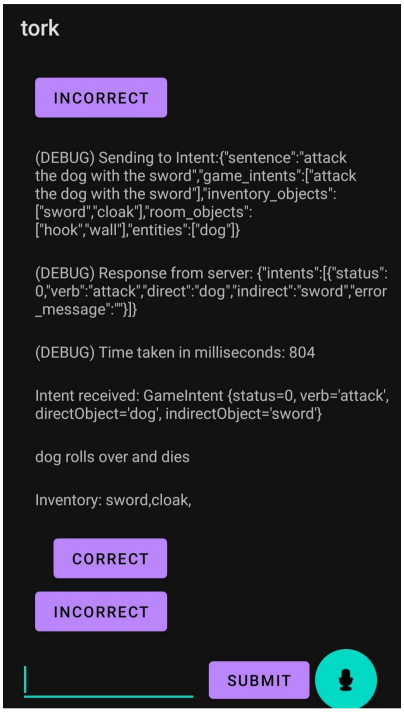
(i) Response to "strike the hound with the blade."



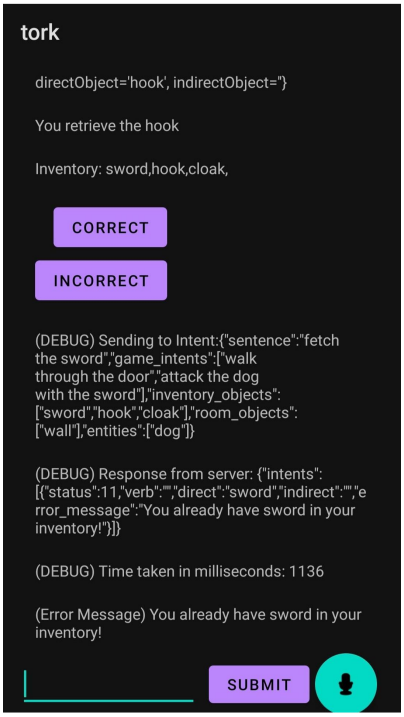
(j) Response to "throw the cloak at the dog."



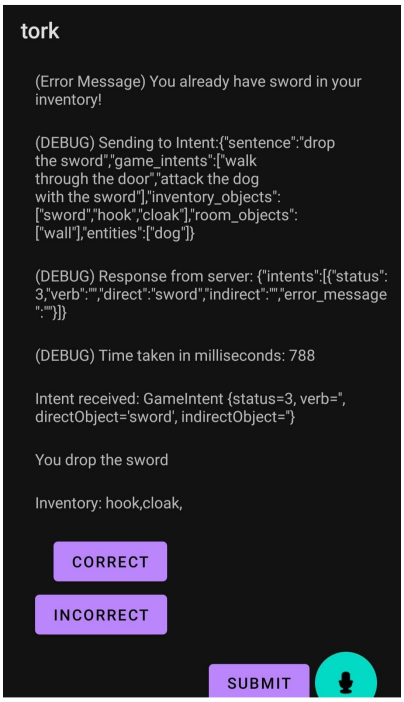
(k) Dealing damage to the dog.



(l) The dog reaches zero health.



(m) Response to "fetch the sword."



(n) Response to "drop the sword."