Imperial College London

MEng Individual Project

Department of Computing

Imperial College of Science, Technology and Medicine

# Voice Recognition Escape Room

*Author:*
Syretta Man Nga Yin

*Supervisor:*
Dr. Mark Wheelhouse

*Second Marker:*
Dr. Arthur Gervais

June 20, 2022

Submitted in partial fulfillment of the requirements for the MEng Computing of Imperial College London

**Abstract**

Speech recognition technology development could be a game-changer in the video game industry, taking the experience of voice-activated gaming to a whole new level. The existing approaches, however, fail to generalise the words or attain the level of efficiency demanded by video games.

The project investigates the state-of-the-art NLP techniques such as Coreference Resolution and WordNet, as well as the prospects of incorporating them into video game production and gameplay. To process user speech input, an intent extraction interface has been developed and is embedded in the Room Generator and Game Processor. The Room Generator creates a text-based escape room game by conversing with the designers, while the Game Processor extracts intent from the players' input. The engines are deemed to be a quick and effective illustration of the strength of intent extraction. The project also analyses limitations and potential future improvement.

**Acknowledgements**

I would like to express my deep and sincere gratitude to my supervisor, Dr Mark Wheelhouse, for providing their invaluable guidance, comments and support throughout the project. It was a great privilege and honour to work on this project under his guidance.

I would also like to thank my second marker, Dr Arthur Gervais, for taking the time to review my interim report.

I am incredibly grateful to my parents for their love, understanding, and continuous support throughout the past four years of studies, which would have been impossible without them.

Finally, I would like to extend a heartfelt thank you to all my friends who provided me with emotional support and countless joyful times.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

An escape room is a puzzle game where a group of players are locked in a room. Their goal is to escape the room by finding clues and solving puzzles. Escape room games are available both in physical and virtual forms. During the COVID-19 epidemic, however, physical escape rooms have lost popularity due to safety precautions. Even though players can still access a virtual online escape room, the sensation of engagement is diminished because participation in the game is limited to button presses and mouse clicks.

The gaming industry is rapidly expanding and becoming increasingly immersive daily by incorporating virtual reality and motion-sensing into games. Voice recognition has been mainly used for virtual assistants such as Apple's Siri, Amazon's Alexa, and Microsoft's Cortana, which will be activated when a "hot word" like "Hey Siri" is said. Other voice recognition applications include voice biometry, transcription of a conversation, or learning a new language. The rise of speech recognition technology may be a game-changer in this market, making voice-activated gaming even more pleasurable.

In previous years, numerous attempts have been made to integrate voice recognition into video games. However, game production is a complex and time-consuming process. It requires the creation of the plot, characters, animation, and much more. Incorporating voice recognition into the game increases the difficulty of its production. If the game is to be released globally, not only do they need an extensive library of voices to train the algorithm, but they must also consider different ways for different languages, dialects, and accents.

Consequently, most games use a rule-based approach to natural language processing, such as regular expression (REGEX) or context-free grammar (CFG). These methods correlate the user's input with a specific command in the game via pattern matching. Since they produce a high recall but low precision, they can only be employed in specific circumstances and cannot be generalised. Rule-based natural language pro-

cessing is challenging since the programmer must evaluate all possible user inputs and actions. This would include numerous game rules.

Within this project's scope, I would like to investigate using existing natural language processing tools to create a fast yet robust engine that might be integrated into video games to lessen the restrictions on user input compared to the existing rule-based approach.

## 1.2 Objectives

This section explains the key objectives of this project:

1. **Building of Intent Extraction Engine**

   The primary purpose of the project is to compare and select the best tools and algorithms available for various NLP tasks for the intent extraction engine. In the end, we hope to obtain an NLP interface that can comprehend and process the user's intent to provide an accurate response. It seeks to provide as much flexibility as possible in order to reduce the need for complex coded rules for speech understanding.

2. **Extract Relevant Information from Input to Generate Room**

   The second objective is to build an escape room level using the Intent Extraction Engine. Instead of relying on game developers to build levels, anyone without coding knowledge will be able to create a room. With the assistance of the engine, users should be able to draw their own story as expressively and with as many elements as they desire.

3. **Escape Room Gameplay Powered by the Intent Extraction Engine**

   The final objective is to create a playable escape room that can interact with the environment based on user input, demonstrating the power of speech processing of the Intent Extraction engine during gameplay.

# Chapter 2

# Background

The project's essential requirements include dealing with text or speech before extracting intent. The essential technical principle of Natural Language Processing(NLP) and its related tools will be covered in this chapter to provide a general knowledge of the topic before digging into the project implementation.

## 2.1 Existing Voice Recognition Games

Integrating voice recognition technology is not something new in the gaming industry. In this section, we will examine how existing video games, including Albedo, Lifeline, Starship Commander and Façade, utilise voice recognition and understand the challenges of voice recognition in games.

### 2.1.1 Albedo

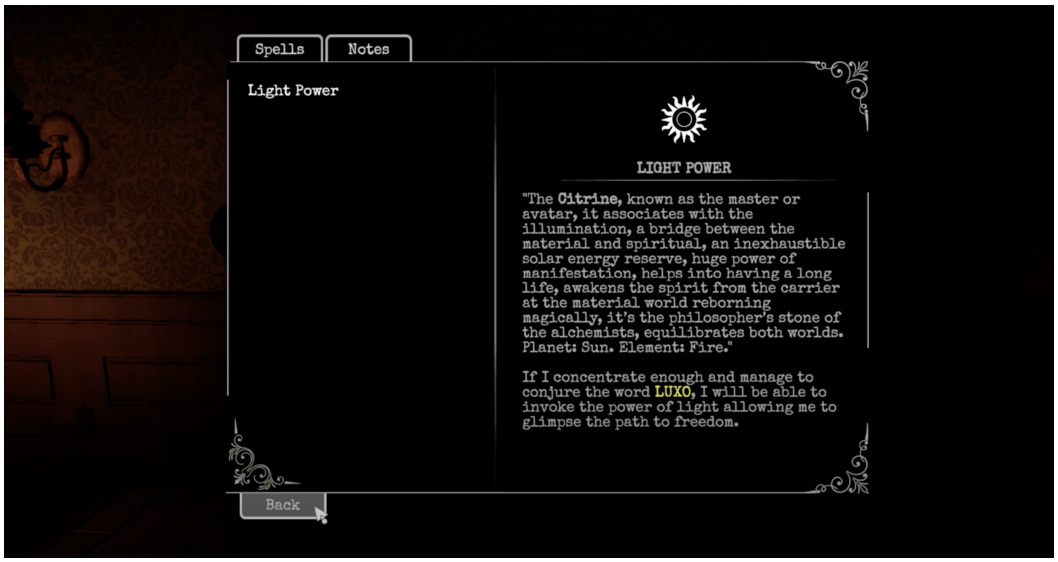



**Figure 2.1:** Screenshot of Spell Instruction in Albedo

Albedo is a first-person perspective survival game developed by APEStudio[1]. Before players can escape from the abandoned manor, they have to explore the place and release the soul that got trapped in the manor. To do so, players are granted special powers that can be activated using their voices. Four different magic spells correspond to different powers. *Luxo* is the power of light, *VERA* is the power to disappear, *ADUL* is the power to kill the evil and *NIMA* is the power to reveal the path of evil.

There was not much NLP technique in the game, as voice recognition is only used to recognise four spells named by the player. Drawing from my experience, the game's voice recognition algorithm is neither sensitive nor accurate. It took a long time for the game to recognise which spell I pronounced, making it insensitive. Also, each spell may have a different way of pronouncing, so even when I tried repeating "ADUL" many times, it did not pick up what I was saying, demonstrating its incapability to handle different pronunciations.

### 2.1.2 Lifeline


**Figure 2.2:** Screenshot from Lifeline when Player is Giving Out Commands to Rio

Lifeline is a video game released by Konami for the PlayStation 2 in 2004[2]. It is the first video game that is solely controlled by voice.

The player joins a party in a space station, which is then attacked by space creatures. The player is imprisoned in the control room but has access to all the Space Station's mechanisms and cameras. The player notices Rio, a cocktail waitress from

the Space Station hotel, is held captive and is trying to call the control room. The player can contact Rio and guide her out of the Space Station by giving voice commands.

As the players cannot physically control Rio, they must give Rio instructions on what to do. The player must push the input mic button on the controller to inform the game that they are issuing a command, after which they can provide commands such as ”hurry,” ”stop,” and ”dodge”. When Rio meets a monster, the player could instruct Rio to attack the monster with a combination of commands such as ”Shoot, Left Eye, Reload” so that Rio could aim at a particular spot of the monster and shoot.

The player will also have a short conversation with Rio throughout the game, guiding them to the ending. During the conversation, we could see that the voice control in the game is not restricted to a list of commands. Rio can respond to whatever the player says.

Lifeline’s ability to facilitate dialogue in a human-like manner is remarkable, but the game’s voice recognition still has some shortcomings. Summarising the reviews on Lifeline, the voice recognition accuracy seems relatively low as Rio cannot comprehend the commands properly during the combat. It is difficult to shout out commands in such a short amount of time during Rio’s combat. Also, specifying certain items when the players explore the rooms seems to be a bit troublesome. For example, if the players tell Rio to ”look at the box”, Rio will likely reply with ”What?” or if they tell her ”The cigar box”, she will reply with something irrelevant like ”I think the chair is totally fine.” [3] Therefore, another weakness of the voice recognition in Lifeline is its inability to generalise terms, so when you say it differently, Rio will not be able to respond correctly.

### 2.1.3 Starship Commander: Arcade

Human Interact released Starship Commander: Arcade in 2018. It is a cinematic virtual reality experience where the players are allowed to speak to the non-player characters (NPC) in the story of a sci-fi setting[4].

The game was released in conjunction with Microsoft[5]. Although the developers from Human Interact have considered developing their NLP technology, they decided to use the widely available tools for their games due to unforeseeable challenges during the development process. They used the Custom Recognition Intelligent Service (CRIS) for their speech recognition engine and Language Understanding Intelligent Service (LUIS) for the translation between spoken phrases and a number of discrete intention actions for the interactive narrative[6].

Starship Commander has two different types of voice interaction. The first is a straightforward speech-to-text problem, in which users are given a command on the screen, and the game responds when they read the exact phrases aloud, such

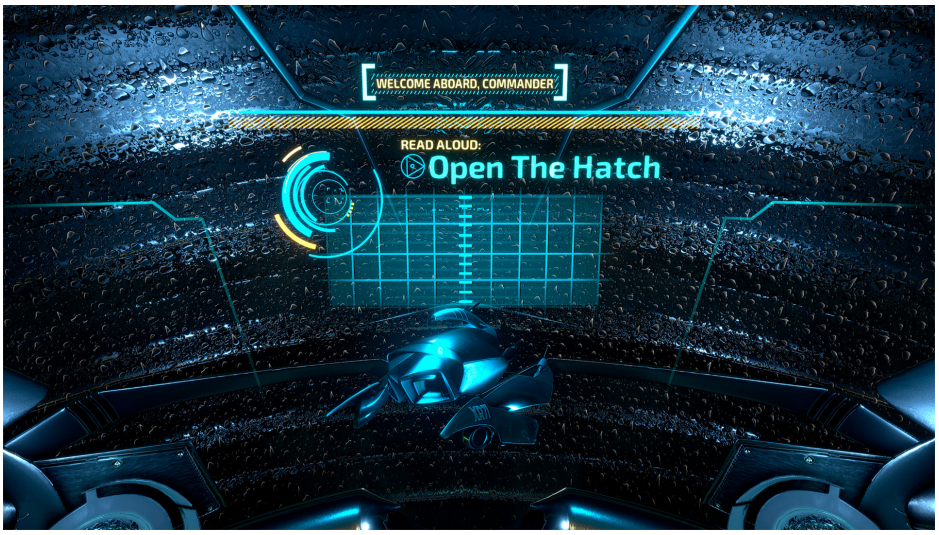



**Figure 2.3:** Screenshot from the Gameplay of Starship Commander:Arcade

as ”Open the Hatch,” as shown in Figure 2.3. However, the second interaction is more engaging when conversing with the spaceship assistant because the players' responses are not restricted. Despite the players' freedom of expression, the assistant is able to respond to the players' input accordingly and promptly without significant delay. Thus, Starship Commander applies a more sophisticated algorithm and performs better than the previous two games.

### 2.1.4 Façade


**Figure 2.4:** Screenshot from the Gameply of Façade

Façade is an artificial-intelligence-based interactive story created by Michael Meteas and Andrew Stern in 2005. The story starts with the player character visiting the married couple, Trip and Grace, at their apartment and becomes entangled in the

high-conflict dissolution of their marriage. The player can interact with the characters by actually speaking to them.

Façade uses natural language processing technology to understand the players' input. The NLP approach can be divided into two phases[7]: Phase 1 maps surface text into discourse acts representing the pragmatic effects of an utterance, while phase 2 maps discourse acts into one or more character responses. To understand the users' utterances, Façade uses surface text rules that map specific patterns directly to an intermediate meaning representation. For example, if the player says "Hello Grace", the word "hello" will assert an iGreet fact, and "Grace" should assert an iCharacter(Grace) fact. The two facts combined will then produce a DAGreet(?x) discourse act fact, as shown in Figure 2.5. To capture more different ways of greeting, the rules for iGreet or DAGreet will have to be updated. Figure 2.6 shows some of the discourse acts represented in Façade as well as their corresponding pragmatic meaning.

```
"hello" → iGreet
"grace" → iCharacter(Grace)
iGreet AND iCharacter(?x) → DAGreet(?x)
```

**Figure 2.5:** Simple greeting rules in Façade

| ***Representation of Discourse Acts*** | ***Pragmatic Meaning of Discourse Acts*** |
|---|---|
| `(DAAgree ?char)` | Agree with a character. (e.g. "certainly", no doubt", "I would love to" ) |
| `(DADisagree ?char)` | Disagree with a character. (e.g. "No way", "Fat chance", "Get real", "Not by a long shot") |
| `(DAPositiveExcl ?char)` | A positive exclamation, potentially directed at a character. ("Yeah", "Wow", "Breath of fresh air") |
| `(DANegExcl ?char)` | A negative exclamation, potentially directed at a character. (e.g. "Damn", "That really sucks", "How awful", "I can't stomach that", "D'oh!") |
| `(DAExpress ?char ?type)` | Express an emotion, potentially directed at a character. The emotion types are *happy* ("I'm thrilled", ":-)"), *sad* ("That bums me out", ":-("), *laughter* ("ha ha"), and angry ("It really pisses me off", "grrrr"). |
| `(DAMaybeUnsure ?char)` | Unsure or indecisive, potentially directed at a character. This discourse act is usually a response to a question. (e.g. "I don't know", "maybe", "I guess so", "You've lost me") |
| `(DAThank ?char)` | Thank a character (e.g. "Thanks a lot") |
| `(DAGreet ?char)` | Greet a character. (e.g. "Hello", "What's up") |
| `(DAAlly ?char)` | Ally with a character. (e.g. "I like you", "You are my friend", "I'm here for you") |
| `(DAOppose ?char)` | Oppose a character. (e.g. "Kiss off", "You're the worst, "I hate you", "Get out of my life") |

**Figure 2.6:** Examples of Discourse Acts in Façade

When testing the NLP engine along with the animation engine, the surface text rules run to completion in 300 milliseconds or less. As a result, the Façade engine's performance appears to be adequate. Even though the set of rules was robust during preliminary play testing, the size of the 800 template rule may still involve substantial human effort when it comes to modification.

## 2.2 Components of NLP

Natural Language Processing (NLP) is a branch of Artificial Intelligence that enables the computer to understand the text as well as spoken words as much as humans do [8]. This section has outlined the sequence of stages a text or speech must undergo to perform different NLP tasks. The stages are shown in Figure 2.7.

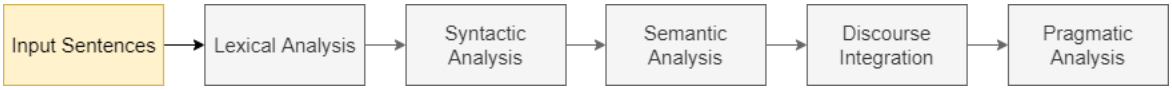



**Figure 2.7:** NLP Pipeline

### 2.2.1 Part-of-Speech Tagging

Part-of-speech (POS) tagging is a crucial prepossessing step for other NLP tasks such as semantic and syntactic analysis. It assigns a special label to each token to indicate the part of speech as well as other grammatical characteristics such as the tense and number[9]. The common English part of speech is noun, verb, adjective, adverb, prepositions etc. The sample of POS tagging is shown in Figure 2.8 with the POS tag highlighted in red.

I_PRP want_VBP to open_VB the_DT box_NN and_CC see_VB what_WP is_VBZ inside_RB

**Figure 2.8:** Example of Part-of-Speech Tagging

There are several predefined tagsets that a tagger can use. The most popular ones are Penn Treebank tagset with 36 tags (Figure 2.9), the British National Corpus Basic (C5) tagset with 61 tags (Figure 2.10), C7 tagset with 146 tags. They vary in level of detail. For example, Penn Treebank only differentiates six kinds of verbs while C5 has 24 different kinds of verbs because it also considers the present, past and continuous forms of the verb.

| | | | | | |
|---|---|---|---|---|---|
| 1 | CC | Coordinating conjunction | 19 | PRP$ | Possessive pronoun |
| 2 | CD | Cardinal number | 20 | RB | Adverb |
| 3 | DT | Determiner | 21 | RBR | Adverb, comparative |
| 4 | EX | Existential *there* | 22 | RBS | Adverb, superlative |
| 5 | FW | Foreign word | 23 | RP | Particle |
| 6 | IN | Preposition or subordinating conjunction | 24 | SYM | Symbol |
| 7 | JJ | Adjective | 25 | TO | *to* |
| 8 | JJR | Adjective, comparative | 26 | UH | Interjection |
| 9 | JJS | Adjective, superlative | 27 | VB | Verb, base form |
| 10 | LS | List item marker | 28 | VBD | Verb, past tense |
| 11 | MD | Modal | 29 | VBG | Verb, gerund or present participle |
| 12 | NN | Noun, singular or mass | 30 | VBN | Verb, past participle |
| 13 | NNS | Noun, plural | 31 | VBP | Verb, non-3rd person singular present |
| 14 | NNP | Proper noun, singular | 32 | VBZ | Verb, 3rd person singular present |
| 15 | NNPS | Proper noun, plural | 33 | WDT | Wh-determiner |
| 16 | PDT | Predeterminer | 34 | WP | Wh-pronoun |
| 17 | POS | Possessive ending | 35 | WP$ | Possessive wh-pronoun |
| 18 | PRP | Personal pronoun | 36 | WRB | Wh-adverb |

**Figure 2.9:** Table showing Penn Treebank Tagset

| | | | | | |
|---|---|---|---|---|---|
| 1 | AJ0 | adjective | 32 | PUQ | punctuation - quotation mark (i.e. ` ' " ) |
| 2 | AJC | comparative adjective | 33 | PUR | punctuation - right bracket (i.e. ) or ] ) |
| 3 | AJS | superlative adjective | 34 | TO0 | infinitive marker TO |
| 4 | AT0 | article | 35 | UNC | "unclassified" items which are not words of the English lexicon |
| 5 | AV0 | adverb (unmarked) | 36 | VBB | the "base forms" of the verb "BE" (except the infinitive) |
| 6 | AVP | adverb particle | 37 | VBD | past form of the verb "BE") |
| 7 | AVQ | wh-adverb | 38 | VBG | -ing form of the verb "BE" |
| 8 | CJC | coordinating conjunction | 39 | VBI | infinitive of the verb "BE" |
| 9 | CJS | subordinating conjunction | 40 | VBN | past participle of the verb "BE" |
| 10 | CJT | the conjunction THAT | 41 | VBZ | -s form of the verb "BE", i.e. IS, 'S |
| 11 | CRD | cardinal numeral | 42 | VDB | base form of the verb "DO" (except the infinitive) |
| 12 | DPS | possessive determiner form | 43 | VDD | past form of the verb "DO" |
| 13 | DT0 | general determiner | 44 | VDG | -ing form of the verb "DO" |
| | DTQ | wh-determiner | 45 | VDI | infinitive of the verb "DO" |
| 15 | EX0 | existential THERE | 46 | VDN | past participle of the verb "DO", |
| 16 | ITJ | interjection or other isolate | 47 | VDZ | -s form of the verb "DO" |
| 17 | NN0 | noun (neutral for number) | 48 | VHB | base form of the verb "HAVE" (except the infinitive) |
| 18 | NN1 | singular noun | 49 | VHD | past tense form of the verb "HAVE" |
| 19 | NN2 | plural noun | 50 | VHG | -ing form of the verb "HAVE" |
| 20 | NP0 | proper noun | 51 | VHI | infinitive of the verb "HAVE" |
| 21 | NULL | the null tag | 52 | VHN | past participle of the verb "HAVE" |
| 22 | ORD | ordinal | 53 | VHZ | -s form of the verb "HAVE" |
| 23 | PNI | indefinite pronoun | 54 | VM0 | modal auxiliary verb |
| 24 | PNP | personal pronoun | 55 | VVB | base form of lexical verb (except the infinitive) |
| 25 | PNQ | wh-pronoun | 56 | VVD | past tense form of lexical verb |
| 26 | PNX | reflexive pronoun | 57 | VVG | -ing form of lexical verb |
| 27 | POS | the possessive (or genitive morpheme) 'S or ' | 58 | VVI | infinitive of lexical verb |
| 28 | PRF | the preposition OF | 59 | VVN | past participle form of lex. verb |
| 29 | PRP | preposition (except for OF) | 60 | VVZ | -s form of lexical verb |
| 30 | PUL | punctuation - left bracket (i.e. ( or [ ) | 61 | XX0 | the negative NOT or N'T |
| 31 | PUN | punctuation - general mark (i.e. . ! , : ; - ? ... ) | 62 | ZZ0 | alphabetical symbol |

**Figure 2.10:** Table showing the C5 Tagset

Although it seems to be simple to indicate if a word is a noun or a verb in a sentence, it becomes complex when there are several possible part-of-speech for that word. For example, the word ”lock” can be a noun that refers to a device that secures something or a verb that refers to the act of securing something with a lock. This is when the problem of ambiguity arises.

There are different approaches to dealing with ambiguity, including rule-based and probabilistic[10]. Rule-based tagger tags the words based on the handcrafting rule that specifies the condition. In contrast, the probabilistic tagger estimates the probability of a specific tag for that given the word under the observed context.

### 2.2.2 Lexical Analysis

Lexical analysis, also known as tokenization, is the process of understanding what words mean and identifying their relationships. It takes a string of characters and turns them into lexical tokens [11]. Tokenization of an English sentence is simple when a specified delimiter is used; it may, however, be challenging for languages that do not have inter-word spaces.

For example, using a space delimiter to tokenize the text ”unlock the door with a key” yields six tokens: ”unlock”, ”the”, ”door”, ”with”, ”a”, ”key”.

### 2.2.3 Syntactic Analysis

Syntax refers to the grammatical structure, determining the role of work in a sentence or phrase. Syntactic analysis, often known as parsing, guarantees that a piece of text is structured correctly. At this stage, techniques like lemmatization, stemming and dependency parsing will be applied. It also uses the Part of Speech(POS) generated to analyze the sentence structure[12]. The details of how the POS can be obtained is illustrated in Section 2.2.1.

- **Lemmatiztion and Stemming**

  The purpose of both lemmatisation and stemming is to reduce the inflectional forms of the words and reduce them to their common base or root, but they act differently. Stemming works by considering a list of common prefixes and suffixes. It then cuts off part of the word, but sometimes it may result in unsuccessful cutting. On the contrary, lemmatisation normalises the words into lemmas, i.e. the word’s root form.

  For example, the result of stemming for the words, *[change, changing, changes, changed]*, will be *”chang”* while lemmatisation will return *”change”*.

- **Dependency Parsing**

  Dependency Parsing is the process of examining the dependencies between words in a sentence to determine its grammatical structure [13]. The dependency relation has one head and a dependent that modifies the head. With dependency parsing, we can understand the subject, the object of the verb, and what is modifying the object.

  Figure 2.11 shows an example of dependency parsing on the sentence ”I open the door with a key.” where ”open” is classified as the verb, ”I” as the subject and ”door” as the object of the verb. As a result, by observing the arrows, we can tell ”door” is the dependent that modifies the head ”open”.

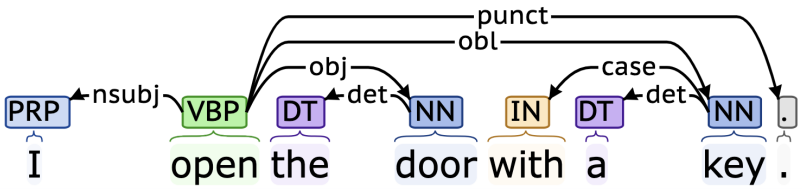



**Figure 2.11:** Example of Dependency Parsing

A fast and effective approach to dependency parsing is transition-based parsing. There is a stack where the parse is constructed, a buffer of tokens to be parsed and an oracle that takes actions on the parse[14]. The parse moves tokens from the buffer onto the stack as it traverses the sentence from left to right. The oracle decides which transition to apply:

- **LEFTARC**: Assign the current words as the head of some previously seen word.
- **RIGHTARC**: Assign some previously seen words as the current word's head.
- **SHIFT**: Store the current word for later.

A complete trace of the transition-based dependency parsing on the example sentence, "book me the morning flight", is shown in Figure 2.12.

| Step | Stack | Word List | Action | Relation Added |
|---|---|---|---|---|
| 0 | [root] | [book, me, the, morning, flight] | SHIFT | |
| 1 | [root, book] | [me, the, morning, flight] | SHIFT | |
| 2 | [root, book, me] | [the, morning, flight] | RIGHTARC | (book → me) |
| 3 | [root, book] | [the, morning, flight] | SHIFT | |
| 4 | [root, book, the] | [morning, flight] | SHIFT | |
| 5 | [root, book, the, morning] | [flight] | SHIFT | |
| 6 | [root, book, the, morning, flight] | [] | LEFTARC | (morning ← flight) |
| 7 | [root, book, the, flight] | [] | LEFTARC | (the ← flight) |
| 8 | [root, book, flight] | [] | RIGHTARC | (book → flight) |
| 9 | [root, book] | [] | RIGHTARC | (root → book) |
| 10 | [root] | [] | Done | |

**Figure 2.12:** Example of Transition-based Dependency Parsing

### 2.2.4 Semantic Analysis

Semantics are the intended meanings of words. Semantic analysis studies the grammatical structure of a sentence or a document and identifies the relationships between individual words in a particular context[15]. This helps the computer understand and interpret the meaning of the documents. The semantic analysis aims to draw meaning from the text and check its meaningfulness. Instead of focusing on smaller tokens like lexical analysis, it focuses on larger chunks like sentences or

phrases. Therefore, it first studies the meaning of individual words, and this component is called lexical semantics. The lexical semantics can then be merged to generate the complete sentence's meaning. For instance, the phrase ”hot ice cream” will be disregarded since the lexical meanings of ”hot” and ”ice cream” are incompatible.

The following are some important elements in semantic analysis:

- **Synonyms**
  When two or more words have a similar meaning, for example, ”happy” and ”delighted”.
- **Antonyms**
  When two or more words have opposite meanings like ”open” and ”close”.
- **Hyponyms**
  When the word has a more specific meaning than a more general word, for example, ”table” is a hyponym of ”furniture”.
- **Homonyms**
  When two or more words share the same spelling or pronunciation but different meanings and origins, such as ”whole” and ”hole”.
- **Meronomy**
  A word that denotes a constituent part or a member of something, like ”arm” is a meronomy of ”body”.

Text classification models and text extractors are the two ways to the semantic analysis[16]. Text classification models assign the text into predefined categories. For example, sentiment analysis is often used for categorising text, like a user's review, into positive, negative, or neutral. Text extractors, on the other hand, take specific information from the text. For instance, the Entity Extractor extracts named entities from text based on established categories, such as a person's or building's name.

The following are examples of NLP tasks that are semantically based:

#### Word Sense Disambiguation

Word sense ambiguation is one of the most significant challenges when applying NLP techniques. A word sense is one of the meanings of a word; for example, WordNet has five possible senses for the word ”pen” [17]:

1. a writing implement with a point from which ink flows
2. an enclosure for confining livestock
3. a portable enclosure in which babies may be left to play
4. a correctional instituion for those convicted of major crimes
5. female swan

We will discuss the most common approaches for word-sense disambiguation (WSD):

- **Dictionary-based Method**

  It is also known as knowledge-based disambiguation. The Lesk method is a seminal dictionary-based method, which is also the most well-known one. We select the word sense that shares the greatest number of definition words with the senses of its neighbouring words. A frequent example of such an algorithm is ”pine cone”. The dictionary definitions are as follows [18]:

  Pine
  1. kinds of evergreen tree with needle-shaped leaves
  2. waste away through sorrow or illness

  Cone
  1. solid body which narrows to a point
  2. something of this shape whether solid or hollow
  3. fruit of certain evergreen trees

  The term ”evergreen tree” appears in the first definition of Pine and the third definition of Cone; therefore, the words are disambiguated.

  However, the limitation of the Lesk Algorithm is that the result is very dependent on the exact wording of the definitions. Besides, dictionary definitions are typically concise, making it challenging to compare similarities.

  Another option would be using a lexical knowledge base such as WordNet. It combines the characteristics of dictionaries and structured semantic networks, grouping synonyms into synsets that represent unique lexical concepts. Therefore, using WordNet makes it easier to consider word-sense relatedness and semantic similarity.

- **Corpus-based Method**

  The corpus-based technique is a supervised machine-learning algorithm designed to address the constraints of the knowledge-based method by learning from sense-annotated corpora. The context is provided as a collection of word features that additionally includes information about the surrounding words [19]. There are many difficulties in the field of supervised WSD, but among all attempts, support vector machine (SVM) and memory-based learning are the most successful approaches to WSD. However, a supervised system involves extensive manual labelling of words. Furthermore, it works under the assumption that the context may resolve ambiguity, which is not always accurate.

#### Named Entity Extractions and Relationship Extractions

Named Entity Extractions(NER) is a subtask of Information Extraction that identify the entities in an unstructured text into pre-defined categories like a person, organisation, location etc. It may not be relevant to this project as Entities are less likely to be named in the escape room. However, the Relationship Extractions(RE) that usually follow Named Entity Extractions will be closely related to our project. It extracts the relationship between two or more entities. For example, "The scissors are on the table." The "is on" is a relation between the scissors and the table [20].

The extraction can be accomplished in several ways, including a rule-based method that could be customised for our project. However, this would require substantial manual work to create all the required rules. Other methods, such as supervised RE or even unsupervised RE, can solve the hardcoding problem but would require a huge training library.

### 2.2.5 Discourse Integration

A discourse is a collection of statements that, when read together, offer a coherent understanding. Discourse integration looks at how the previous sentences affect the sentences afterwards; it also brings about the meaning of the immediately succeeding sentence[21]. It is suggested that implicit references of the speaker, time or place of utterance in expressions such as pronouns (I, you) and demonstratives (this, that) are an inherent and unavoidable property of natural language[22]. Therefore, discourse integration is vital in resolving coreference in speech.

#### Coreference Resolution

Coreference resolution plays an important part in discourse integration as it aims to resolve repeated references to an object in a context[21]. If two or more expressions refer to the same discourse entity, they are said to be *corefer*[23]. In the example, "Put the key into my inventory. Use it to open the lock.", "it" refers to "key", and they are said to *corefer*.

Coreference resolution can be divided into the following subtasks[24]:

- **Mention Detection**
  It aims to find all candidates span referring to some entities, including pronouns, noun phrases, and named entities. The following entities will be detected from the above example: {*key, inventory, it, lock*}, they are called the *mentions*.
- **Mention Clustering**
  The second sub-task is to figure out which ones refer to the same entity, and then the mentions will be grouped into a cluster that corresponds to the entities in the context. The result from mention clustering for the previous example will be {*it, key*}, {*inventory*}, {*lock*}.

### 2.2.6 Pragmatic Analysis

The pragmatic analysis deals with outside word knowledge, which means it is not limited to the context. It finds out what the word meant [25]. For example, without pragmatics, a response to the sentence ”Can you pass the salt?” will be a simple ”yes” or ”no.” However, with pragmatism, we should be able to recognise that it is a request to pass the salt to the speaker [26].

## 2.3 Word Embedding

In order to perform text analysis, the raw text data should be represented in a certain way so that information can be extracted from them. A vector representation of a word is called word embedding. It is one of the most popular representations of raw text since it captures the context of a word, semantic and syntactic similarity and relation with other words[27].

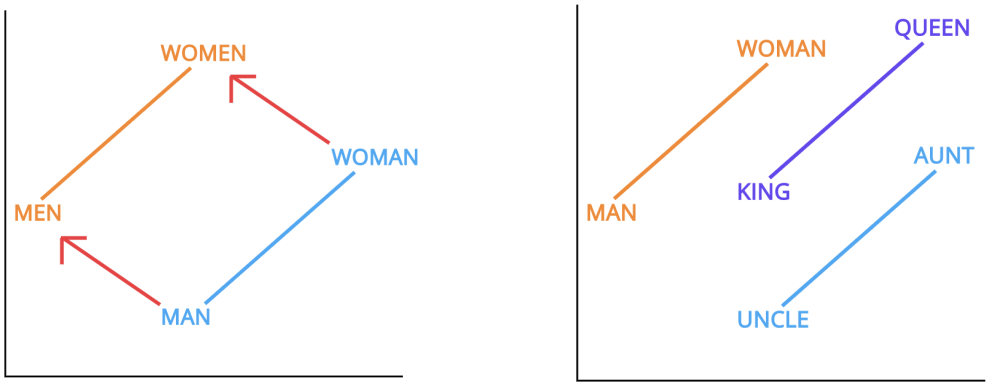
There are various ways of doing this, such as Bag-of-Words (BOW), using features extracted from words, etc. Take BOW as an example; the tokens from the text data will be used to create a vocabulary. Then a vector with values representing the frequency of tokens appearing in the text data is formed with a length equivalent to the vocabulary size. This poses a challenge to the computation since the vectors are likely to contain many null values, resulting in sparse vectors[28]. It also ignores the semantic relationship between tokens, resulting in a poor understanding of the words.

This section will look at three unsupervised models for learning word embeddings that potentially tackle the concerns mentioned above.

### 2.3.1 Word2Vec

Word2Vec is a predictive model that creates word embedding using a 2-layer neural network. It takes in a large corpus of text and outputs a vector space with the words that share a common context in the corpus located close to each another[29]. In this way, the vectors can capture relationships such as plurals, as shown in Figure 2.13(a). As demonstrated in Figure 2.13(b), if the embedding vector of ”MAN” is minus from the one of ”KING”, adding the embedding vector of ”WOMAN” will then yield the embedding vector of ”QUEEN”. This illustrates how Word2Vec can be used to deduce the relationship between words.

Word2Vec is a combination of the Continuous Bag-of-Words model (CBOW) and Skip-gram model, illustrated in Figure 2.14[30]. CBOW predicts the target word given the context words, i.e. the words surrounding the target word, whereas the Skip-gram model instead predicts the context words given the target word. Both algorithms are trained similarly. By inputting the word and taking the hidden layer after training, we can retrieve the embedding layer for that specific word.




**((a))** Plurals of the word

**((b))** Relationship between words

**Figure 2.13:** Vector Space from Word2Vec

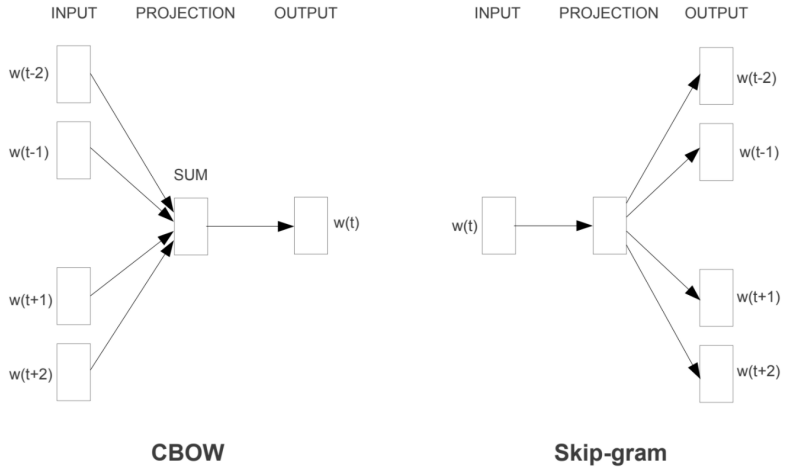



**Figure 2.14:** Word2Vec - CBOW and Skip-gram Model

The limitation of Word2Vec is that summing across the entire corpus vocabulary will lead to a costly computation. Also, Word2Vec cannot represent words that are not in the vocabulary.

### 2.3.2 BERT

BERT (Bidirectional Encoder Representations from Transformers) is published by Google AI Language. It applies the bidirectional training of a Transformer to language modelling [31]. The following are essential for understanding how BERT works:

**Sequence-to-sequence (Seq2Seq)**
Seq2Seq is a model that employs a recurrent neural network (RNN), composed of

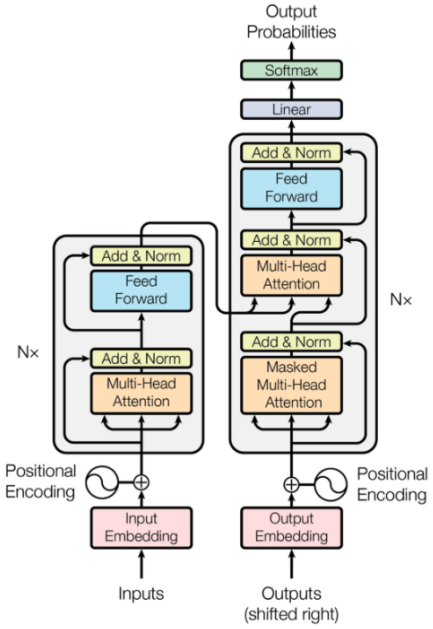



**Figure 2.15:** Architecture of Transformer

an Encoder and a Decoder. The former takes the input sequence and maps it into an n-dimensional vector, then it is fed into the Decoder and turned into an output sequence[32]. However, the dependency of token computation on previous results makes it inefficient to train.

**Attention Mechanism**
With the help of the attention mechanism, the decoder takes in a weighted combination of all encoded input vectors, with the most relevant vectors being assigned the highest weights, thus utilising the most relevant part of the input sequence[33].

**Transformer**
Transformer is a model that adopts self-attention mechanism as well as the encoder-decoder architecture[34], shown in Figure 2.15. Instead of sequentially processing the input, it processes the full input at once, offering greater parallelisation than RNN models. Bidirectionality is one of the critical features of the Transformer because it enables the model to process input from start to finish or vice versa.

Although BERT adopts the bidirectional training of a Transformer, it only uses the encoder but not decoder[31], because it only need the encoder for the semantic and syntactic information in the embeddings for a wide range of tasks.

From Figure 2.16, there are several embedding layers in BERT's text processing stage[35]. In the Segment Embeddings layer, BERT learns unique embeddings for different sentences to distinguish between them, which will benefit tasks like question-answering. The Position Embeddings layer expresses the position of the words in a

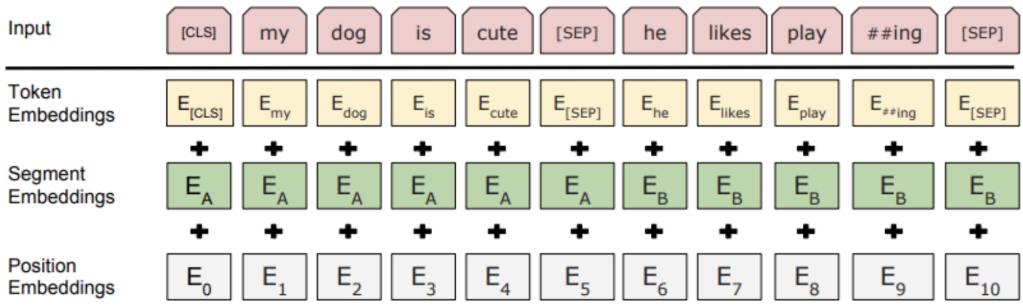



**Figure 2.16:** BERT's Text Processing

sentence, which helps capture information about the words' order. Summing all three embeddings will result in the representation of a particular word.

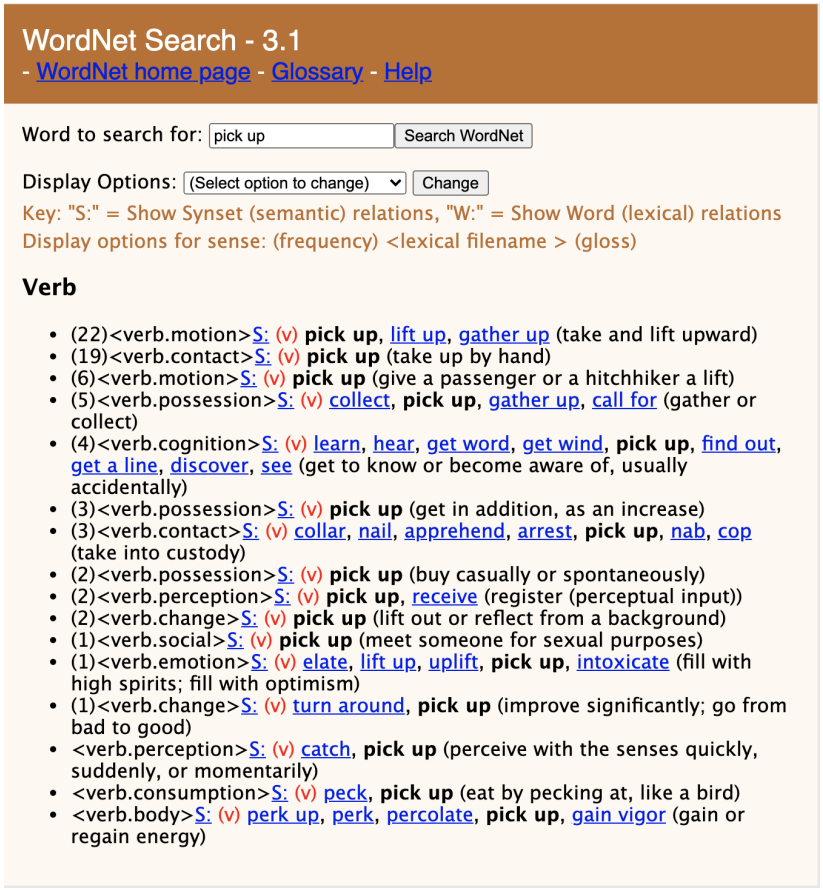
The text processing layers of BERT make it easier to train a BERT model for different NLP without any significant changes in the model. The biggest limitation of BERT is being compute-intensive at inference time. However, it should not be a major problem in this project since it only becomes costly when it is used in production at scale.

## 2.4 Tools

### 2.4.1 WordNet

WordNet is an extensive lexical database of English that is very useful for text analysis or natural language processing[36]. Each word is displayed with short definitions and examples, as shown in Figure 2.17. Synonyms like ”close” and ”shut” are grouped into synsets, each expressing a distinct concept and are interlinked by conceptual-semantic and lexical relations. WordNet is not like an ordinary thesaurus. It does not only find synonyms or antonyms of the word; it finds other relations such as hyperonymy, hyponymy etc. WordNet is a handy tool in NLP tasks, such as word-sense disambiguation, information extraction, machine translation etc. It is also helpful in determining the similarity between words.

As shown in Figure 2.18, the word-form ”funds” is recognized by WordNet as two lexical entries: ”fund” and ”funds”. The former belongs to three synsets: store, fund, fund, monetary fund and investment company, fund while the latter belongs to one synset: monetary resource, funds. [37] The example also shows us how WordNet connects synset as mentioned above. The synset money is related to another synset medium of exchange because ”money” is a hyponym of ”medium of exchange”.



WordNet Search - 3.1
- WordNet home page - Glossary - Help

Word to search for: pick up Search WordNet

Display Options: (Select option to change) Change

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations
Display options for sense: (frequency) <lexical filename > (gloss)

**Verb**

- (22)<verb.motion>S: (v) **pick up**, lift up, gather up (take and lift upward)
- (19)<verb.contact>S: (v) **pick up** (take up by hand)
- (6)<verb.motion>S: (v) **pick up** (give a passenger or a hitchhiker a lift)
- (5)<verb.possession>S: (v) collect, **pick up**, gather up, call for (gather or collect)
- (4)<verb.cognition>S: (v) learn, hear, get word, get wind, **pick up**, find out, get a line, discover, see (get to know or become aware of, usually accidentally)
- (3)<verb.possession>S: (v) **pick up** (get in addition, as an increase)
- (3)<verb.contact>S: (v) collar, nail, apprehend, arrest, **pick up**, nab, cop (take into custody)
- (2)<verb.possession>S: (v) **pick up** (buy casually or spontaneously)
- (2)<verb.perception>S: (v) **pick up**, receive (register (perceptual input))
- (2)<verb.change>S: (v) **pick up** (lift out or reflect from a background)
- (1)<verb.social>S: (v) **pick up** (meet someone for sexual purposes)
- (1)<verb.emotion>S: (v) elate, lift up, uplift, **pick up**, intoxicate (fill with high spirits; fill with optimism)
- (1)<verb.change>S: (v) turn around, **pick up** (improve significantly; go from bad to good)
- <verb.perception>S: (v) catch, **pick up** (perceive with the senses quickly, suddenly, or momentarily)
- <verb.consumption>S: (v) peck, **pick up** (eat by pecking at, like a bird)
- <verb.body>S: (v) perk up, perk, percolate, **pick up**, gain vigor (gain or regain energy)

**Figure 2.17:** Example Query Result in WordNET

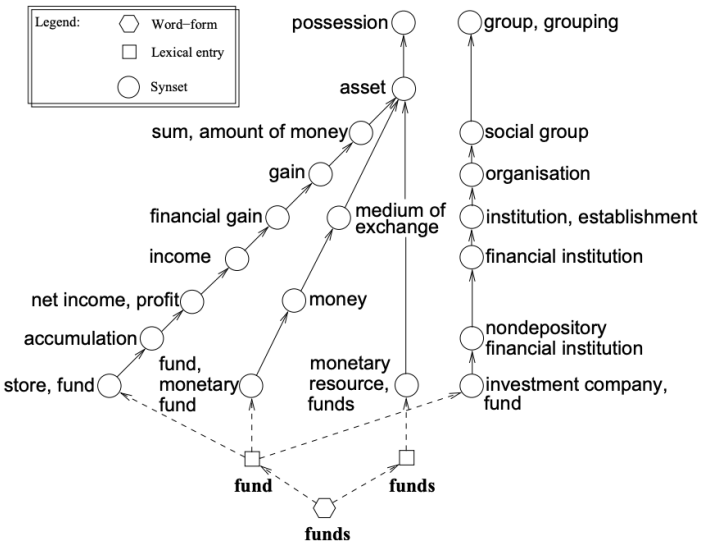



**Figure 2.18:** Mapping from word-forms and lexical entries to synsets and their hypernyms in WordNet

### 2.4.2 NLTK

NLTK is one of the most popular libraries for building programs with language data in Python. The interface is easy to use, so it may be a good choice for a project that has a limited time. It also has rich corpora and lexical resources like WordNet, along with pre-models of text processing tasks like tokenization, stemming, tagging, parsing, etc., which will be very convenient when dealing with speech data[38].

However, it does not analyze the semantic structure before splitting the text into sentences[39]. Also, another con of using NLTK is that they do not provide a neural network model.

### 2.4.3 spaCy

spaCy[40] is another NLP library that is available in Python. It provides custom models for deep learning frameworks like TensorFlow and Pytorch. It also has pre-trained transformers like BERT and pre-trained word vectors, which on the contrary, are not provided by NLTK. It also provides models for named entity recognition, POS tagging, dependency parsing, sentence segmentation, text classification, lemmatization, morphological analysis etc. Therefore, spaCy might be a better choice when compared to NLTK as it contains all the tools we might need in our project.

# Chapter 3

# Project Design

The voice recognition escape room is divided into two sub-projects, including the game creation and the gameplay. This chapter will discuss the selection of NLP technologies and provide an overview of the components.

## 3.1 Overview

This section presents a brief explanation of the purpose of each component. In the future sections, the Room Generator and Game Processor shall be referred to as the *engines*. Figure 3.1 shows the system architecture of the voice recognition escape room.

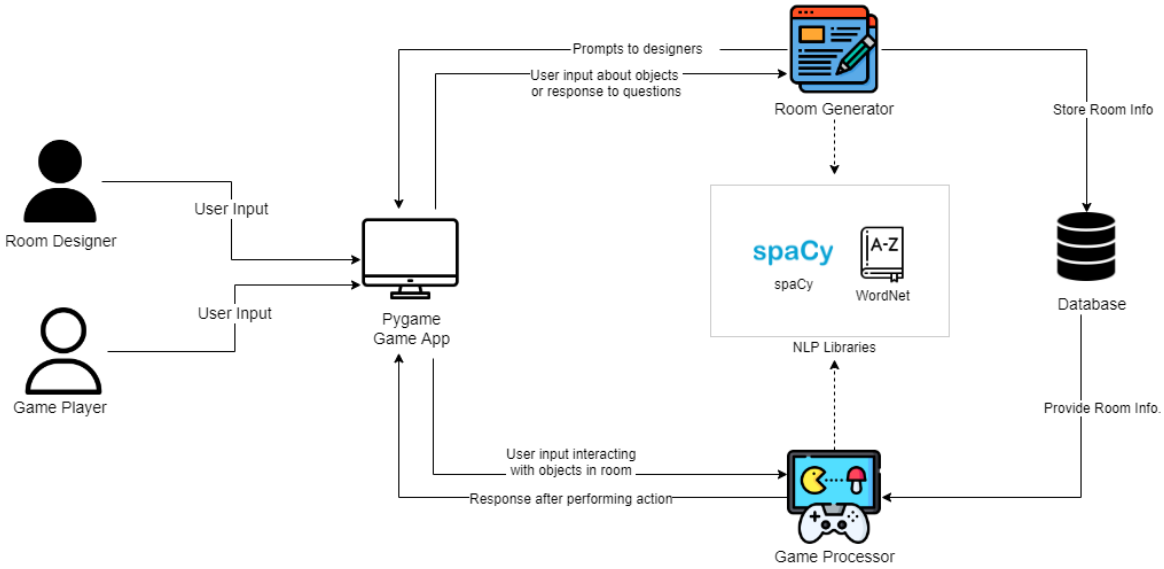



**Figure 3.1:** Diagram showing the Interaction between Project Components

The *engines* are developed using Python, a simple but powerful language that offers some of the most useful NLP libraries. The Room Generator will return a game written in Pygame and, therefore, will be used as the game console when showcasing the *engines* functionality. Web development framework like Flask is not used for

the game console because the power of NLP tools during the room creation and gameplay is more important and should be prioritised in the development stage. Therefore, Pygame is used as it is sufficient in demonstrating the functionality of the escape room games generated by the *engines*.

### 3.1.1 Room Generator

The Room Generator is responsible for the creation of the room to escape. The user who runs the Room Generator will be referenced as the *designer* in the following sections. The Room Generator does not require the *designer* to know how to code an escape room game. With the help of the NLP technologies, the *designer* can construct their own escape room game with their unique storyline by simply conversing with the Room Generator. Upon completion, a playable escape room should be created.

### 3.1.2 Game Processor

The Game Processor manages the escape room game after it has been constructed by the Room Generator. The users playing the game will be referred to as the *player* in the following parts. The *player* can interact with the objects in the room using either voice input or normal text input. The Game Processor will then process the user input, extract the intent from it and perform the corresponding action predefined by the *designer* if a match is found.

## 3.2 System Overview

Figure 3.2 and Figure 3.3 show the architecture of the Room Generator and the Escape Room respectively. Both processes involve the processing of user input before performing their own duties; therefore, they share some common components like speech-to-text, coreference resolution and identification of object, action and tool. Each of the components will be discussed further in its corresponding chapter.

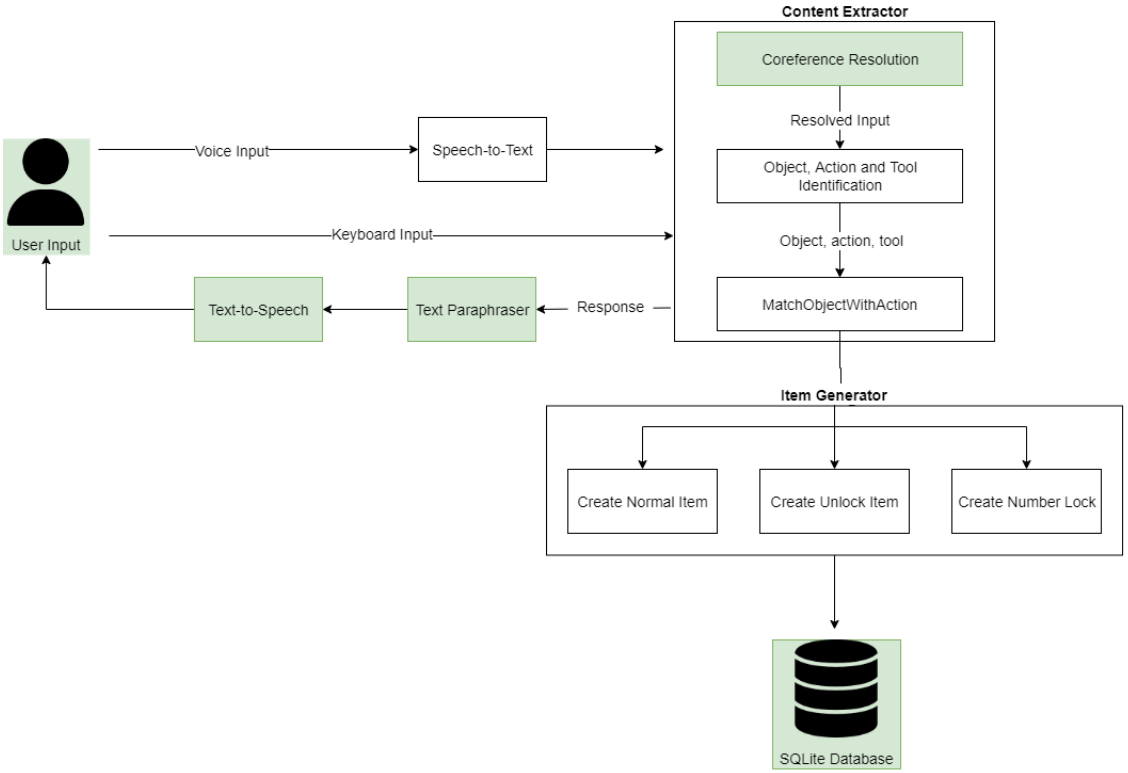



**Figure 3.2:** System Overview of the Room Generator

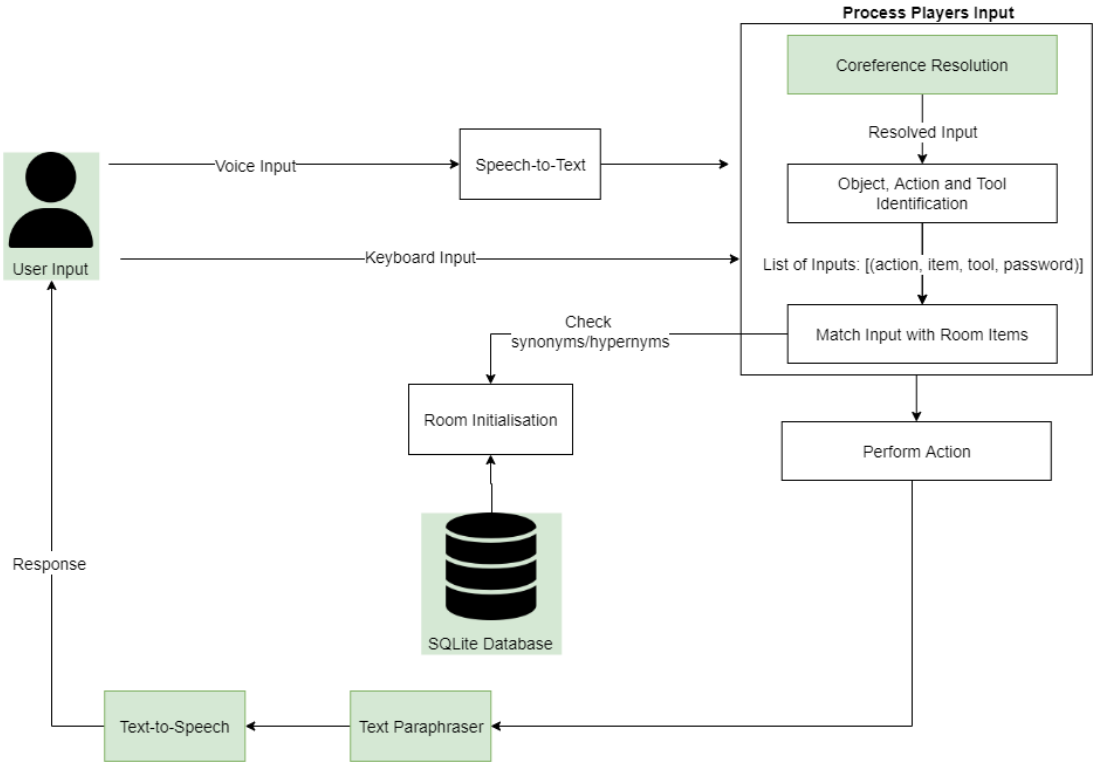



**Figure 3.3:** System Overview of the Game Processor

## 3.3 Technologies

There are many available NLP libraries and software that we could use to process the user input, each having its pros and cons. This section will discuss the rationale behind the chosen tools.

### 3.3.1 NLTK

**NLTK** [38] is a platform to work with human language data. It provides user-friendly interfaces to many corpora and lexical resources such as WordNet. It provides a suite of text processing libraries such as tokenization, stemming, tagging etc. However, NLTK is only used for accessing the **WordNet** corpus reader, enabling us to identify objects using their synonyms and hypernyms.

### 3.3.2 spaCy

**spaCy**[40] is the main software library for NLP that I used in the project. Compared to NLTK, spaCy's object-oriented approach returns an object rather than just a string from the function, making investigation and access to information easier.

To get started with spaCy, a trained pipeline is loaded via `spacy.load()`. In this project, the pipeline `en_core_web_sm` is used. It is an English pipeline optimised for CPU that includes the following components: Tok2vec, Tagger, DependencyParser, SentenceRecognizer, EntityRecognizer, AttributeRuler, and Lemmatizer. It is the smallest one with a size of 12MB but will be enough for the current project. After loading the pipeline, a Language object that contains all components and data needed to process text will be returned. Then we can access the information of the processed document, sentences or tokens by calling the functions of that objects. For example, for a given `token`, we can easily access the part-of-speech and dependency tag by `token.pos_` and `token.dep_`. The all-in-one property, alongside the outstanding performance of spaCy, has made it a good choice for the project.

Moreover, it has a powerful component, **Matchers**, that enables us to match sequences of tokens based on pattern rules we define. An example of pattern rules that we can apply is like:

```
[LOWER: "i", LEMMA: "IN":  ["like", "love"], POS: "NOUN"]
```

This means if the text contains a word equal to "i" after lowering the case, and a word with its lemma be either "like" or "love", followed by a noun, then it will be returned as a match, e.g. "I like Imperial" or "I love Computing". Matchers are very useful when it comes to recognising the objects and actions from user input.

Furthermore, **NeuralCoref** [41] is pipeline extension for spaCy that performs coreference resolution using a neural network. It consists of two modules; one is a rule-

based module identifying the potential coreference mentions with the help of spaCy's parser, tagger and NER annotations. Another is a Feed-Forward neural network to compute the coreference score for each potential pair.

**spacy-wordnet** is another spaCy's pipeline component that allow us to use **WordNet Domains**. It extends the functionality of WordNet from synonyms and hypernyms to showing the list of domains that one particular synset is in. Searching through the domain would be useful in shortlisting the synsets because we could skip the computation for those synsets that are out of the domain.

### 3.3.3 pyttsx3

It is a text-to-speech conversion library in Python. This is chosen over other alternative libraries like Google Text to Speech (GTTs) because it works offline and provides a series of configurations, thus providing greater flexibility. For example, we can use different voices installed on the system or control the speed of speech as well as adjust the volume.

# Chapter 4

# Implementation

This chapter will investigate the implementation of two main components in this project, the Room Generator and the Game Processor.

## 4.1 Overview

Before diving into the implementation of the *engines*, we should look into assumptions that the *engines* hold and the `Item` class to understand the type of input accepted by the *engines*.

### 4.1.1 Assumptions

To ensure that the escape room game can be played effectively, the *engines* simply hold two assumptions:

- The *designers* are expected to have a solvable design of the room.

### 4.1.2 `Item` Class

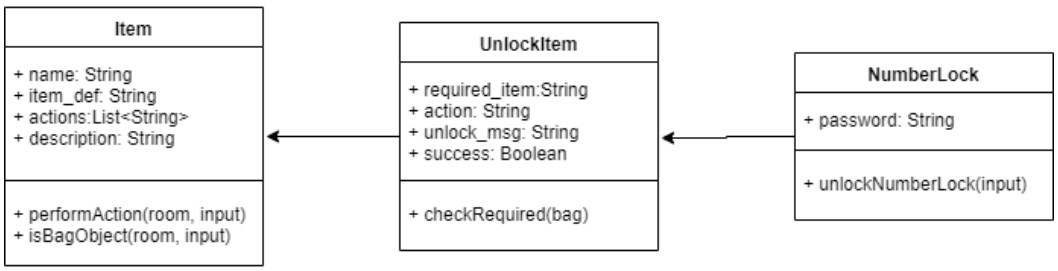



**Figure 4.1:** UML Class Diagram of Item Class

Figure 4.1 shows the three different types of items that are currently supported by the *engine*.

1. **Item**: These are the normal items that are allowed to be picked up or investigated by default, but it is not limited to these. The *designer* is free to associate different actions with the item.
    - name: The object's name.
    - item_def: The name of associated Wordnet synsets, e.g. *key.n.01*.
    - action: A list of actions associated with this item.
    - description: The description of the object.
    - performAction(): Takes in input that contains the object, action and tools and performs the action. For example, acting "investigate" is set to return the item's description.
    - isBagObject(): Check if the input object is in the bag.
2. **UnlockItem**: These item needs to be unlocked by other items, e.g. a door will be an unlock item if it is unlocked with a key. The unlock method is not restricted to only using a key. It depends on the creativity of the *designer*.
    - required_item: The name of the item that will unlock this item.
    - unlock_action: The action that will unlock the item is "unlock" by default.
    - unlock_msg: Hint giving to the players who unlocked this item.
    - success: Indicate if the item is unlocked.
    - checkRequired(): Check if the required_item is in the bag.
3. **NumberLock**: This is a special case of the unlock items. Instead of using another item, it is unlocked using a numeric password. The length of the numeric password is not restricted and depends on the *designer*.
    - password: The numeric string used to unlock the item.
    - unlockNumberLock(): Takes in the input string and checks if it matches the password.

## 4.2 Shared Components

The stages highlighted in green in system diagrams of the Room Generator and Game Procesor, Figure 3.2 and Figure 3.3, refers to the common stages that both Room Generator and Game Processor will go through whenever taking in user input.

### 4.2.1 User Input

In order to give the user the freedom to express themselves in the *engines*, the users are not required to input in a standard format or with specific phrases. The Room Generator and Game Processor are expected to handle any kind of user input without any restrictions and react accordingly to provide a more favourable user experience. The followings are acceptable from the *designer* and the *player*:

- Input can consist of multiple sentences.
- Input can consist of more than one object and action.
- Input can consist of adjectives and adverbs that describe the objects or actions.

The *engines* takes voice as the primary means of input, although there is also flexibility in switching between voice and text input. Figure 4.2 shows the interface of the Room Generator, which is similar to the one of the Game Processor. If the users wish to create the room or play the game using voice input, they can press on the microphone icon at the bottom to start recording their voice. Otherwise, they can simply type into the yellow input box.

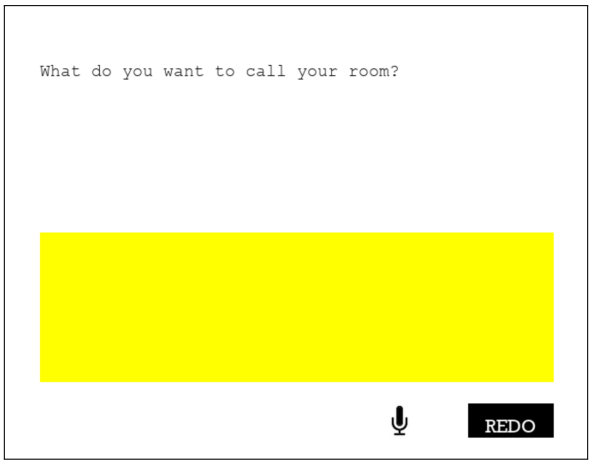



**Figure 4.2:** Screenshot of Room Generator

Google Speech Recognition in PyPI library is chosen among all the available engines to perform the speech-to-text processing. Although the Google Speech Recongition is deemed to be powerful and accurate, there is possibility that the result could be wrong. As a result, the users can click on the ”REDO” button at the bottom right corner, which will reset the input in the box, or they can amend the result by pressing backspace on their keyboard.

The *player* can interact with the room object freely, such as ”pick the key up” and ”unlock the door with the key” as long as their input satisfies the conditions above. However, in the Room Generator, the expected inputs from the *designer* depend on the state of the game creation.

### States of Room Creation

The room generating stage is divided into different states as shown in Table 4.1. The program will prompt the users to answer the questions accordingly to create a room that meets the requirement of having at least one possible solution.

| State | Meaning of the state |
|---|---|
| NAME_ROOM | To provide a name for the current room. |
| OVERALL_DESCRIPTION | To briefly describe the room, telling what the players will see when they first enter the room. |
| INPUT_PROCESS | To identify the object and action to add to the room. |
| UPDATE_STORED_ITEM | To decide if a new item is added or an existing item is updated. |
| CREATE_NEW_ITEMS | To create a new item in the room with the provided description. |
| ASK_FOR_UNLOCK_ITEM | To decide if an item is unlocked with another object or a numerical password. |
| FILL_IN_UNLOCK_ITEM | To provide the tool and action to unlock the object. |
| FILL_IN_PASSWORD | To provide the password of the number lock. |
| CONGRATS_MSG | To provide a hint or congratulation message after the players unlock the object. |
| ADD_MORE | To decide if more objects are added. |
| FINISHED | To decide if the room creation has finished. |
| EXTRA_ITEM | To provide more items to create. |

**Table 4.1:** Table showing States in Room Creation

During the room creation, the *designers* should expect follow-up questions. Those questions may ask for further information about particular objects, such as the password of the `numberlock` object or how to unlock it. It may also be a simple yes/no question where the users can click on the screen when the options appear, like in Figure 4.3.

### 4.2.2 Coreference Resolution

The *engines* will then handle any coreference existing in the current text. As a result, applying coreference resolution on the current input will not be useful if the *designer* or *player* is referring to something from the previous input, like the situation shown in Figure 4.4. Considering only the current input will result in an unsuccessful resolution since it has no information about what ”It” refers to. As a result, a text file called `room_generator_log.txt` is created to log the *designer* input and `user_input_log.txt` for the *player*'s. NeuralCoref, as described in the preceding section, performs coreference resolution based on the whole log file and returns the


**Figure 4.3:** Screenshot of Yes/No Option

last phrase as the resolved current input.

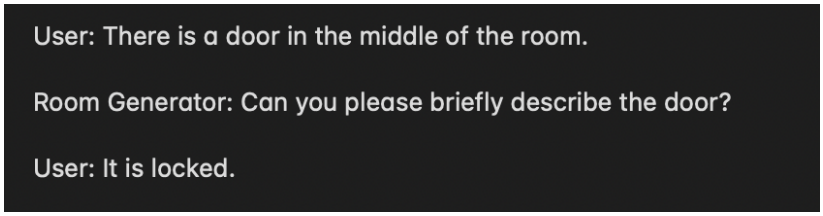



**Figure 4.4:** Example of Coreference Resolution Usage

With the help of the log file, coreference resolution is not restricted to the current input. The resolved text in the log file from the above example will then be *"There is a door in the middle of the room. The door is locked."* Thus returning *The door is locked.* as the resolved current input.

### 4.2.3 Text Paraphrasing and Text-to-Speech Processing

The text paraphrasing stage is added to the end of the pipeline to diversify the responses from the *engines* so that they do not repeat the same phrases all the time; otherwise, it would sound like speaking to a machine. Among all the available paraphrasing tools, Pegasus, T5 and Parrot paraphrasers are experimented with and compared, which will be discussed in Section 5.1.2. After comparing all three models, Pegasus is selected and a pretrained model `"tuner007/pegasus_paraphrase"` is used to paraphrase for the *engines*. The input is first put into the Pegasus Tokenizer and then into the model to generate paraphrased text.

Truncation is activated in the tokeniser, and the maximum length of the text is set to be 60 characters; any text over the maximum will be truncated. The model is configured to return only five sequences, and one will be picked randomly as the response text. Table 4.2 shows the possible response texts for the input, *"It is written 9797 on the painting."* resulted from the Pegasus Paraphraser.

| **It is written 9797 on the painting.** |
|---|
| The painting has a writing on it. |
| It is written on the painting. |
| There is a writing on the painting. |
| It is written on a painting. |
| On the painting is written 9797. |

**Table 4.2:** Example of Text Paraphrasing using Pegasus

Instead of only displaying the paraphrased text on the game console, they are passed into the pyttsx3 engine. The Room Generator or the Game Processor will then read the text aloud. This is believed to create a more engaging experience for the users as it feels like a two-way conversation between the game console and the users.

### 4.2.4 SQLite Database

The Room Generator stores created items into the SQLite Database. Every table in the database represents one created room in the escape room game, and the room name is stored as the table name in the database. In order to recognise the object that leads to the success of the escape, the first row of every table is reserved for storing information about the entire room. As shown in Table 4.3, the *item* on the first row is always set to be "room", and the description will be the overall description of the room and is the first hint that the game players will see when entering the room. The object that is classified as the success item of the room will be stored as the *required_item*. For example, unlocking the door in the room shown in the following table will lead to a successful escape.

The next three rows show how different type of objects is stored in the database. The three different types of items that a `designer` can create have their special fields and are explained in the previous sections. If the items do not have information about certain fields, like normal objects do not have information about *unlock_msg*, they will be left blank.

| type | item | item_def | action | description | unlock_msg | required_items | unlock_action |
|---|---|---|---|---|---|---|---|
| | room | | | There is a door and a box. | | door | |
| normal | key | key.n.01 | [investigate] | It's a golden key. | | | |
| unlock | door | door.n.01 | [investigate, unlock] | The door is locked. | You have escaped | key | unlock |
| numberlock | box | box.n.01 | [investigaet, open] | The box is locked | You saw a key. | 9977 | |

**Table 4.3:** Structure of Database

## 4.3 Stages in Room Generator

The stages that is only involved in the Room Generator is explained in this sections.

### 4.3.1 Object, Action and Tool Identification

In this stage, the Room Generator will extract information from the designer's input. It uses Matcher, a component from spaCy, to extract the nouns and verbs from the resolved input by matching patterns. The words are classified into noun phrases and verb phrases according to the rules shown in Figure 4.5 and Figure 4.6.

```
pattern1 = [{"POS": "PROPN"} ]
pattern2 = [{"POS": "NOUN"},{"POS": "NOUN", "OP":"?"}]
pattern3 = [{"POS": "NOUN"}, {"LOWER": "of"},{"POS": "DET", "OP":"?"},{"POS":"NOUN"}]
```

**Figure 4.5:** Noun Matcher

```
pattern1 = [{"POS":"VERB"}, {"POS":"PART", "OP":"*"}, {"POS":"ADV", "OP":"*"}]
```

**Figure 4.6:** Verb Matcher

```
pattern1 = [{"DEP":"pobj"}]
```

**Figure 4.7:** Tool Matcher

A noun phrase is extracted, for instance, if there is a word with POS of "NOUN," such as "key." Additionally, two successive words with POS of "NOUN" might combine to make a noun phrase, such as "axe head." The phrase "head of the axe" reflects the final criteria for matching noun phrases. On the other hand, a phrase is defined as a verb phrase if a POS "VERB" word is followed by an optional participle, as in "pick" and "pick up." Lastly, Figure 4.7 shows that a noun will be identified as a tool when the dependency tag is "pobj".

However, Matcher will return every possible match from the text. For example, the input, *"There is an ax head"*, will return a list of matches: *["ax", "head", "ax head"]*. Only the longest match returned by the Algorithm 1 will be chosen to proceed to the next stage. The lines in Algorithm 1 have covered the following cases:

- **Line 4**: If the current match has included the previous match entirely, then keep the current match and discard the previous one.

  *previous match = "pick", current match = "pick up", longest span = "pick up".*

- **Line 6**: If the current match follows the previous match immediately, they will be combined into one single phrase.

  *previous match = "ax", current match = "head", longest span = "ax head"*

- **Line 8**: If the current match begins within the prior match, we shall use the previous match's start index but the current match's end index.

  *previous match = "wooden table", current match = "table top", longest span = "wooden table top"*

**Algorithm 1:** Finding the longest span

**Input:** matches, $length > 0$
**Output:** new_matches

```
1  new_matches = [ ]
2  (prev_start, prev_end) = longest_span ← matches[0]
3  for (start, end) in matches do
4      if prev_start ≥ start + 1 and prev_end ≤ end then
5          longest_span ← (start, end)
6      else if prev_end == start then
7          longest_span ← (prev_start, end)
8      else if start ≥ prev_start and start ≤ prev_end then
9          longest_span ← (prev_start, end)
10     else
11         Append to new_matches
12         longest_span ← (start, end)
13 end
14 Append last span text to new_matches
15 return new_matches
```

### 4.3.2 Object, Action and Tools Matching

After identifying the verbs as actions, and the nouns as objects and tools, the next step is to match the corresponding action and tool to the correct object. Take the sentence, "take the key and open the door with the key" as an example; the followings should be identified:

- **Object:** key, door
- **Action:** take, open

- **Tools:** key

However, it is important to know that the action ”take” is for the key, ”open” is for the door, and the key is a tool to open the door. This will require the help of dependency parsing, where we can deduce that the head of the noun will be the associated action.

For example, Figure 4.8 shows the result of dependency parsing on the example text. ”take” is the head of the noun ”key” while ”open” is the head of the ”door”. Then, the head of its own head token needs to be checked to link the tools to the noun. For instance, the head of ”key” is ”with”, while the head of ”with” is ”open”, which is associated with ”door”. As a result, we can deduce that the ”key” is a tool for the ”door”.

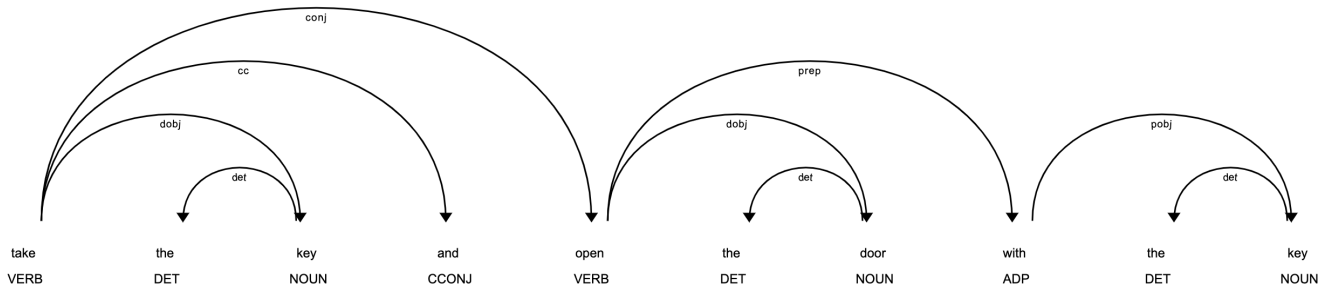



**Figure 4.8:** Example of Dependency Parsing

At this point, the Content Extractor within the Room Generator has completed its job. It should return a queue to the Room Generator that consists of all the objects, the actions and tools specified by the *designer*.

### 4.3.3 Item Generator

The flow of the Item Generator is shown in Figure 4.9. The Item Generator will take the queue from the Content Extractor and process the items one by one. It will first search for a synset for that object and store the 3-part name of the synset of the form: `word.pos.nn`, which will be useful when processing user input during gameplay. If we query the synsets for the word ”table” as a noun, WordNet will return the results listed in Figure 4.10. There may be more than one possible definition, so the Item Generator will choose according to Algorithm 2.

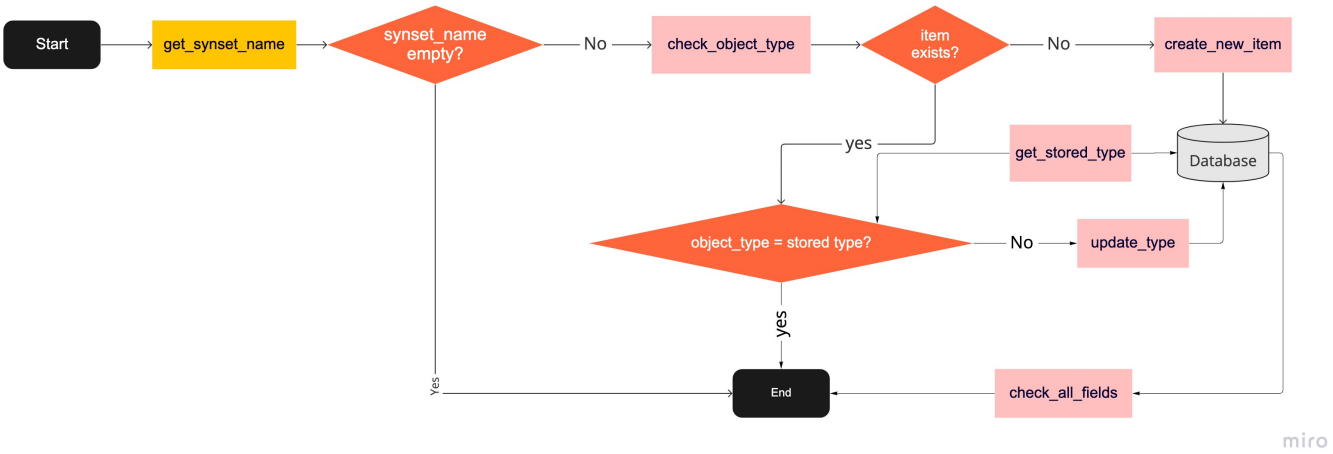



**Figure 4.9:** Flow of Item Generator

| Name | Definition |
|---|---|
| table.n.01 | a set of data arranged in rows and columns |
| table.n.02 | a piece of furniture having a smooth flat top that is usually supported by one or more vertical legs |
| table.n.03 | a piece of furniture with tableware for a meal laid out on it |
| mesa.n.01 | flat tableland with steep edges |
| table.n.05 | a company of people assembled at a table for a meal or game |
| board.n.04 | food or meals in general |

**Figure 4.10:** Synset from WordNet for ”table”

**Algorithm 2:** Finding synset for object

**Input:** $O$ object name
**Output:** $S$ 3-part name of synset
**Data:** $D$ Domains

```
in_domain = synset ∈ D
if in_domain is not empty then
    return in_domain[0]
else
    H_o = unique words that are hypernyms of O
    for h in H_o do
        if ”physical_object” ∈ h then
            return h
    end
end
```

Algorithm 2 starts with getting the synsets in the domain with the help of WordNet Domains. The domain is set to be ”furniture”, considering that most of the objects to be created are likely to be furniture. If the list of synsets is not empty, we can return any synsets from that list. Otherwise, we will need to compute all the hypernyms of

the objects and check if it has a ”physical object” as a hypernym and returns that if it does.

After obtaining the 3-part name of the synset, it determines the type of the item. The object is determined to be an **Unlock** item if the action includes any word with similar meaning as ”unlock” or ”lock”, indicating that the object requires something to unlock it. Here, having a similar meaning means having ”unlock” or ”lock” in their synonyms or hypernyms. If the *designer* claims this item is unlocked using a password, it will be classified as an **NumberLock** item. Otherwise, the item will be classified as an **Normal** item.

The Generator will then query the database to see if the object exists. It will proceed to complete all other required fields of that particular type of object if the object is not found in the database. On the other hand, if the object exists already and the stored type does not match the current identified type, the Generator will update the type and fill in the extra required field. This brings to the end of the Item Generator.

## 4.4 Stages in Game Processor

The stages only involved in the Game Processor are then illustrated in the following section.

### 4.4.1 Room Initialisation

Before starting the game, an empty `Room` object will be created. A `Room` object contains the following variables and function:

**Vairables**

- **name:** The name of the room. It is stored as the table name in the Database.
- **level:** An integer indicating the index of the room when there is more than one room created.
- **description:** It is the description of the entire room that is the first message that the player will see when entering the room.
- **items_in_room:** This is a list of all items created in this room.
- **current_items:** This is a list of items currently existing in the room. It does not include the items that need to be created by players by combining two objects.
- **bag:** The list of items that the players picked up.
- **success:** A boolean value indicating if the player has successfully escaped the room or not.
- **succeed_item:** The name of the item that acts as the exit of the room.

**Functions**

- **obtainedItems:** Check if the item is in the player's bag.
- **getItem:** Take the `Item` object that match with the given name.
- **addItem:** Add a new item into the room by appending the given item into `items_in_room` and `current_items`.
- **initialiseRoom:** It takes in a list of item information and create the corresponding `Item` object according to the type, such as *normal, unlock, combine*. Add all the created `Item` into the room.
- **succeedCondition:** Updating the `success` state of the room by checking the the success state of the `succeed_item`.

The Game Processor will query the database to grab all the items' information for that room and call `initialiseRoom` to fill in the room with the queried items.

### 4.4.2 Object, Action, Tool Identification

The Game Processor uses a different approach to identify the intent from the *player*'s input. Identifying the action and password from the sentence is simply checking if the POS of the word is "VERB" and "NUM", respectively. However, determining the subject and the tool from the sentence requires more care. A noun can be either the object or the tool; therefore, the dependency tags need to be checked. If the noun has a dependency tag of "dobj", it will be classified as the direct object. If the noun has a dependency tag of "pobj", it will be identified as the tool. The example in Figure 4.11(a) shows that "open"," door", and "key" will be classified as action, direct object and tool, respectively, according to the above rules.

open the door with a key

| Token | POS | Dependency Tag | Head |
|---|---|---|---|
| open | VERB | ROOT | open |
| the | DET | det | door |
| door | NOUN | dobj | open |
| with | ADP | prep | open |
| a | DET | det | key |
| key | NOUN | pobj | with |

**((a))** "Open the door with the key"

get the head of ax

| Token | POS | Dependency Tag | Head |
|---|---|---|---|
| get | AUX | ROOT | get |
| the | DET | det | head |
| head | NOUN | dobj | get |
| of | ADP | prep | head |
| ax | NOUN | pobj | of |

**((b))** "Get the head of ax"

**Figure 4.11:** Examples of Part-of-Speech and Dependency Parsing

However, the above does not hold in the example in Figure 4.11(b). If we apply the same rule to this sentence, we will have "get", "head", and "ax" as the action, object and tool, respectively, while the correct result should be "get" as the action and "head of ax" as the object. Therefore, we need the rule to take care of the objects with the form of *"__ of __"*. As a result, the head of the token will play an important part. When we encounter a "pobj" token, we check if its head is an "of" token. If it

is, and if the head of the "of" token is a "dobj", then we know it is an object in the form of *"__ of __"*. The above can be demonstrated by the example in Figure 4.11(b).

Moreover, we should take care of the case where the player mentions more than one object in the input and uses "and" as the conjunction. There can be several meanings for the use of "and":

1. Pick up the key and the box.

2. Pick up the key and open the box.

The "and" in the first sentence means we pick up both the key and the box, while in the second sentence, it works as a delimiter that separates the sentence into two. Ideally, the game processor can identify the different meanings of "and" in the sentence. However, this requires further investigation and time. Therefore, this issue is mitigated by assuming all the "and" act as a sentence separator.

### 4.4.3 Match Input with Room Items

After extracting the object, action and tool from the *player*'s input, the Game Processor will then match the objects and actions with the existing room items and the available actions of that particular item. The process of matching objects and actions is identical, they both involve a cache search, synonyms and hypernyms search, as well as similarity calculation. Algorithm 3 illustrates the general approach of the match.

**Algorithm 3:** Matching input with room items

```
   Input: I item to match, R room
   Output: new_matches
 1 if I ∈ C cache then
 2 |   return result from C
 3 for I_R items ∈ R do
 4 |   for ss ∈ synsets of I_R do
 5 |   |   if ss name matches I then
 6 |   |   |   S = synonyms of ss
 7 |   |   |   H = hypernyms of ss
 8 |   |   |   if I ∈ S or I ∈ H then
 9 |   |   |   |   return I_R
10 |   |   |   else
11 |   |   |   |   p = compute similarity between ss and synsets of I
12 |   |   |   |   return I_R if p > threshold
13 |   end
14 end
15 return nomatchfound
16
```

The cache is a dictionary structure that stores all the previous matches so that we can skip all the other searches and return the match if the same word is reencountered.

If the input is not found in the cache, it will proceed to the synonyms and hypernyms search. It will iterate through all the room items. If a synset is found with a name the same as the input item's name, the synonyms and hypernyms of the synset will be computed and return that room object if the input is found within the comprehensive list of synonyms and hypernyms.

If it is still not found in the synonyms and hypernyms search, we can calculate the similarity between the synsets of all the room objects and the synsets of the input. Leacock Chordorow(LCH) is used to calculate similarity.

$$LCHSimilarity = -log\frac{shortest\_path(synset1, synset2)}{2 * D}$$

where $D$ represent the depth of taxonomy, i.e. the depth of the synset in WordNet. If $LCHSimilarity$ is greater than the threshold, a hyperparameter that requires calibration, the room items will be returned.

If there is no match for the input after all these steps, meaning that the item does not exist in the room. The *player* will then be prompt to try again.

### 4.4.4 Perform Action

After identifying the object and the corresponding action, the Game Processor will perform the action on the room. There are some default responses for standard actions for a different kind of `Item` class. For the **Normal** items, the default actions will be "investigate" and "get", which will allow them to read the description of the items and put the item into their bags. The **Unlock** and **NumberLock** items also have "unlock" as their default action so that the player can open the locked items.

# Chapter 5

# Evaluation

In this section, we will be evaluating the program against the objectives we set in Section 1.2.

1. Create engine that could understand the intent from the input.

2. Build a Room Generator that employs an intent extraction engine to construct an escape room game.

3. Build a Game Processing that uses the intent extraction engine to process *player* input during gameplay.

We will investigate the effectiveness and accuracy of the intent extractor, as well as the performance of the Room Generator and Game Processor. In addition, user feedback will be collected to assess the performance of the *engines*.

## 5.1 User Experience

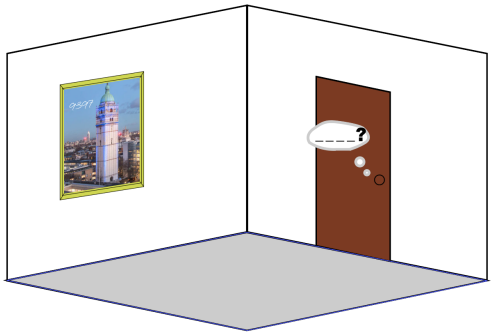
Ten users were invited to test the program; five of them tested the Room Generator, while the other five tested the Game Processor, so they did not know what they would expect in the game. The users are given three different designs that corresponds to different sizes of room. There is a small room with only two objects, a medium room with six objects and a large room with 12 objects as shown in Figure 5.1. The users are given the images and are asked to either create the room according to the images or play with the rooms created by other users.

In this section, we are trying to investigate the following:

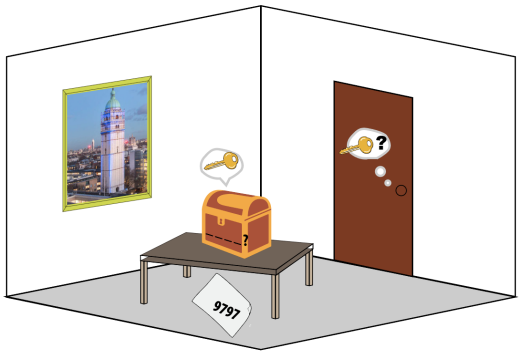
- How does the Room Generator or Game Processor react to the scaling of input?

- Which component is the most inefficient throughout the input processing?

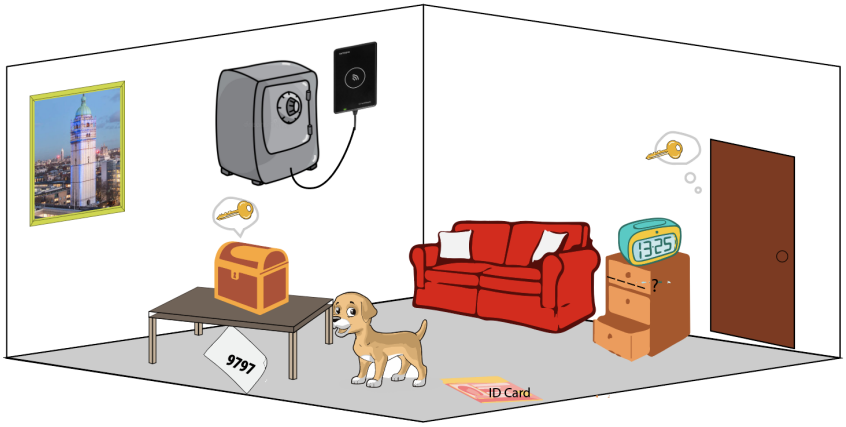
- How can the Room Generator or Game Processor improve?



**((a))** Design of a Small Room




**((b))** Design of a Medium Room




**((c))** "Design of a Large Room"

**Figure 5.1:** Room Designs Provided to Test Users

### 5.1.1 Efficiency of Input Processing

Although an escape room game is not a kind of action game that the player might lose if an immediate response is not given, the efficiency of the input processing is still crucial. A slight delay between the user's input and the program's response could largely affect the user's experience. Therefore, the duration of each component is measured so that the most time-consuming component can be spotted and optimised.

#### Efficiency of Room Generator

The efficiency of the Room Generator is first evaluated. The data is collected during the users' trials and they are illustrated in Figure 5.2 and Figure 5.3. It is clearly shown that the time spent on creating different rooms varies with the number of objects in the room. However, splitting the room generation process into individual tasks shows that the average time spent on each task is approximately the same no matter the room size. The average time of creating one single item is around 60 seconds in every room. As a result, the Room Generator does not seem to be

influenced by a scaling input.

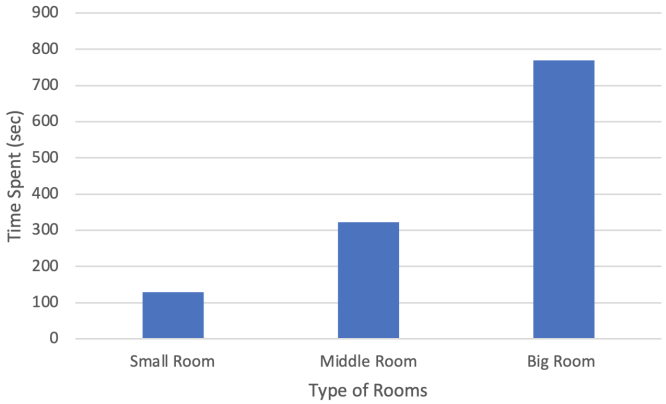



**Figure 5.2:** Comparing Time Spent in Creating Rooms

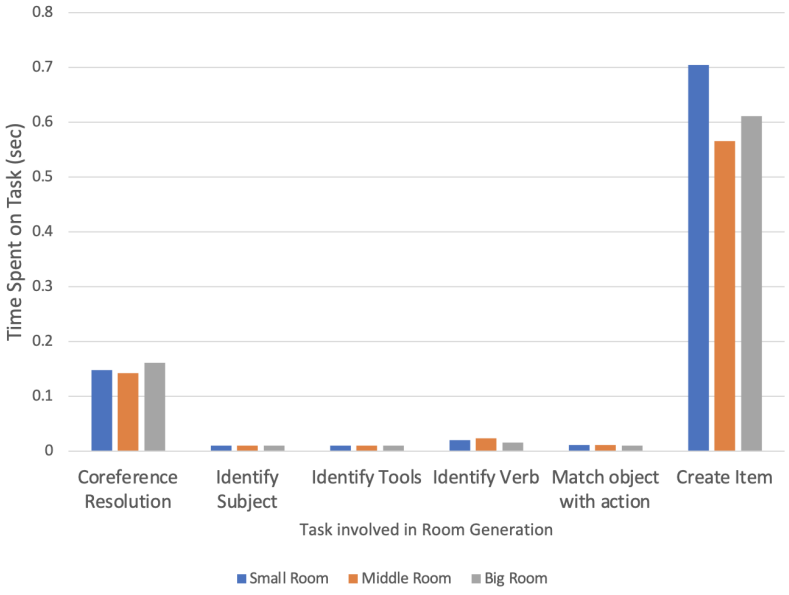



**Figure 5.3:** Comparing Room Generator Efficiency in Different Rooms

### Efficiency of Game Processor

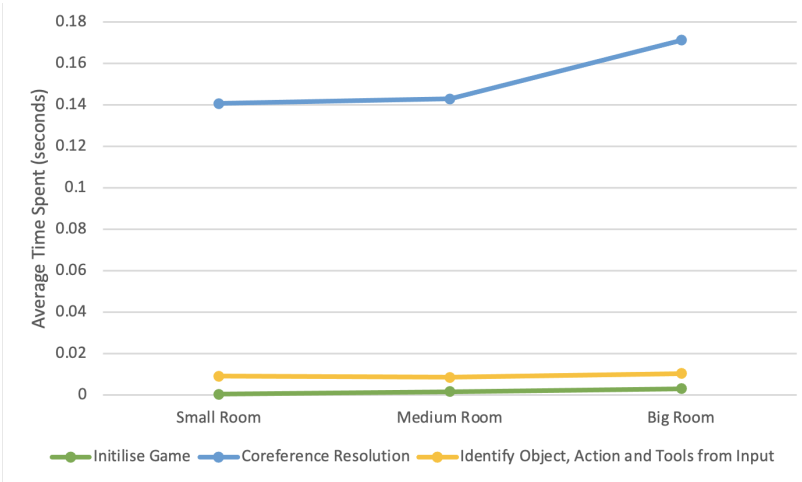
Then, the performance of the Game Processor is also evaluated component by component. The average processing time has been recorded while the test users were playing the game and are summarized in Figure 5.4. The data is split into to diagram to show a clearer comparison due to the difference in range.

The average total time of identifying actions and paraphrasing response are the only two components that do not seem to be affect by the number of items in room. The diagrams also shows some correlation between the room sizes and the time spent on

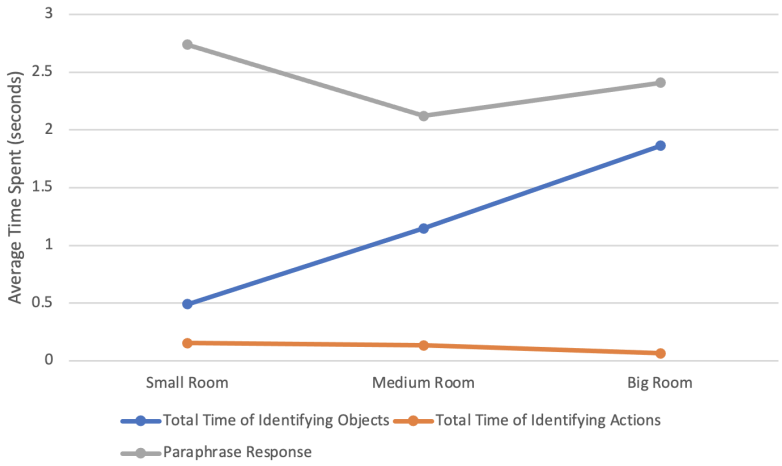
initialising room and identifying object-action pairs from input, though not significant.

Coreference resolution spent approximately 0.15 seconds across all rooms, but it still shows a slightly increasing trend when the number of items increases. This may be caused by the increasing size of the log files when the user input increases as there are more items to interact with. However, this is easy to solve by storing only a specific number of lines in the log file to keep the size small.

One of the apparent findings from Figure 5.4 is that the text paraphrasing stage is the most time-consuming part of the entire procedure, which could take up to 2 seconds regardless of the room size. Additionally, the time used to match the input objects to the room objects is sizable. The average time spent matching the input object to the room object increases drastically. This would be a serious problem when the room scales further. The subsequent sections will look into the causes of such inefficient computation and see how it can be improved.




((a))




((b))

**Figure 5.4:** Comparing Game Processor's Efficiency in Different Room

| **Original Text** | **Pegasus** | **T5** | **Parrot** |
|---|---|---|---|
| It is locked | It is locked up. | It is locked in place. | he's locked |
| There is a painitng hanging on the wall of the room. | The wall of the room has a painitng hanging on it | There is a painitng hanging on the wall. | it is on the wall in the room. |
| It is written 9797 on the painting | It is written on the painting. | On the painting it is written 9797. | It is written 9797 on the painting. |
| I am not sure which item are you referring to, can you try again? | Can you try again, I don't know which item you're referring to. | I am not sure which item are you referring to, can you try again? | can you try again? |
| What do you want to say to them after unlocking this item? Perhaps congratulating them for escaping from this hell or maybe give them some tips? | What should you say to them after you unlocked this item? | What do you want to say to them after unlocking this item? Maybe congratulating them | what do you want to say to the characters after they unlock this item? maybe congratulations for escaping from hell or maybe give them tips? |

**Table 5.1:** Table showing example result of paraphrasing and the time spent

### 5.1.2 Efficiency Issue of Text Paraphrasing

A text paraphrasing stage is added to the end of the pipeline. It is utilised whenever the program returns a response to the user. The objective of paraphrasing text was to create a more realistic interaction between the game and the users by avoiding repetitive sentences.

Numerous pre-trained models were available, and three of them were compared: Pegasus, T5 and Parrot paraphraser. Table 5.1 presents five texts that will be encountered in the Escape Room Game and the result from each model. Comparing these models under identical configurations.

- **num_of_beam**: This refers to the number of beams in the beam search, i.e. the number of active candidates at each timestep. The higher number will likely lead to better performance as this considers more possibilities. This is set to 5.
- **num_of_return_sequences**: This is the number of sequences to be returned by the paraphraser. The current implementation is set to 5, so we can randomly choose one from it.

Figure 5.5 shows how much time different models need for paraphrasing text of varying lengths. The longer the text, the more time they need to paraphrase it, especially with Pegasus and Parrot Paraphraser. Integrating into the game could be

a problem because it will slow down the gaming process and diminish the user experience. On the other hand, T5 is the quickest one that processes a text within 2 seconds regardless of its input length. Therefore, T5 may have an advantage if we only consider the processing speed. However, a good paraphraser is not only

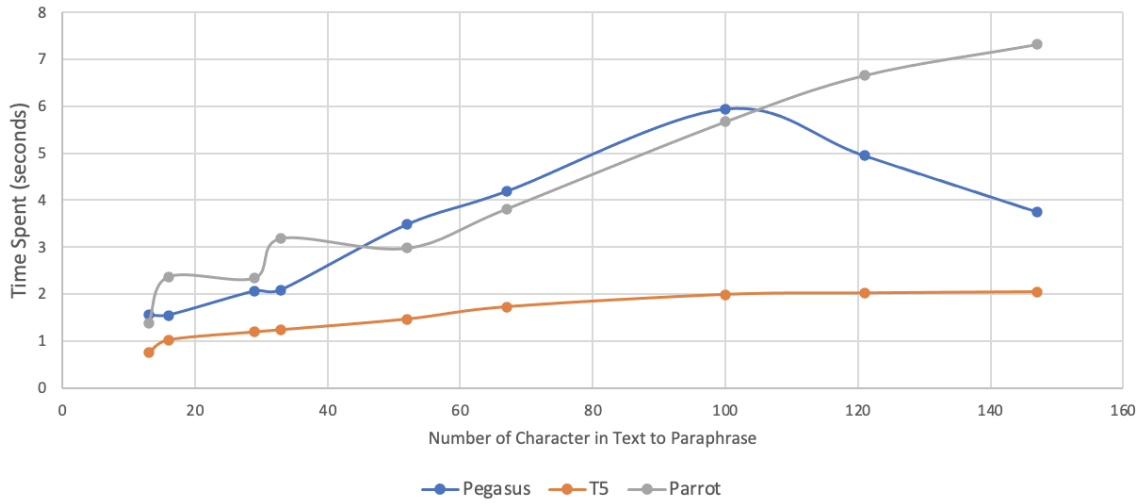



**Figure 5.5:** Comparing Time Spent of Different Paraphrasing Models

determined by speed. The adequacy of the meaning preserved, the fluency, and the diversity of the paraphrase should all be taken into account. Referring to Table 5.1, the paraphrase from the T5 model is less diverse from the original text than the other two models. For a better paraphrase performance, we may need to increase the number of beams to a larger value, such as20, to get a similar diversity score to the other two models. However, this will increase the processing time.

Therefore, it suggests that there may be a trade-off between the speed of paraphrasing a text and the resulting diversity. Given that the arguments in the T5 model must be modified to match the performance of other models and that the text from the Game Processor is typically less than 100 characters, Pegasus is used in the final implementation since it strikes a compromise between speed and text diversity.

### 5.1.3 Efficiency Issue of Object Identification

Object Identification is another element that could consume considerable time during the procedure. On average, it could take up to 0.5 seconds in a small room with only two objects. When it gets to a bigger room containing dozens of objects, it could take approximately 2 seconds to match the input object to the stored room object. Possible causes for the prolonged computations at this step include searching for synonyms and hypernyms and calculating the similarity score.

In the synonyms and hypernyms searches, each word's WordNet synsets are iterated to generate a list of synonyms. Each synset's hypernyms are then iterated, resulting

in a comprehensive list of synonyms and hypernyms for the word. The time complexity of computing the list of synonyms and hypernyms is, therefore, $O(k \times h)$, where $k$ is the number of synsets and $h$ is the number of hypernyms of the synsets.

If the word is not found as a synonym or a hypernym, it will proceed to the similarity calculation. Using the Leacock Chordorow (LCH) similarity, we will compare each room object to all potential synsets of the input item to see if the input object is classed as one of the room objects. This also leads to the time complexity of $O(m \times k)$, where $m$ and $k$ are the numbers of room items and the number of synsets of the input, respectively.

Figure 5.6 has indicated the total number of seconds spent searching synonyms and hypernyms and computing similarity. When the room size scales, the amount of time spent increases significantly. The time required to calculate similarities for objects in a large room could be nearly double that of a small room, indicating that object recognition may struggle with scaling.

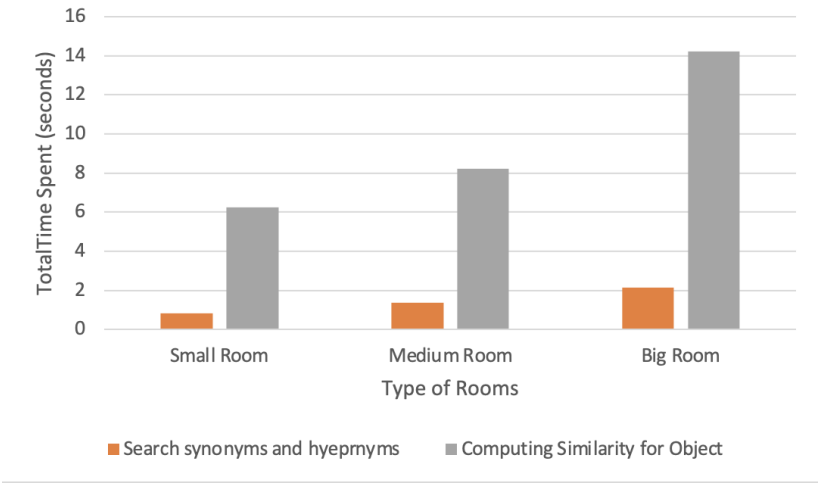



**Figure 5.6:** Diagram showing Total Time Spent in Object Identification in Different Room Sizes

## 5.2 User Feedback

The users who participated in the testing are interviewed, and their feedback is gathered and summarised in the following categories in order to understand the project's goal better.

### 5.2.1 Accuracy of Voice Recognition

The primary objective of the project is that the users could use voice as input for no matter room creating or game playing. Before getting into the performance of the input processing, we asked the users how accurately their voice input was processed.

*”My voice input was recognised most of the time accurately. There are only a few times where it misunderstood my speech, such as when it mistook ”dog” for ”dock,” but this was okay because I could simply try again or change it directly with a keyboard.” - Designer 1*

It may seem that Google’s speech recognition tools are not always perfect, which sometimes could misinterpret user input. However, the problem is mitigated by the option of correcting the result or re-recording by pressing the ”redo” button.

### 5.2.2 Room Creation

One of the objects is to build a Room Generation with the embedded intent extractor and make it easy for people to build an escape room game. Other than input processing accuracy, the users were also asked about any significant problems that impacted their experience.

*I was surprised that the Generator was able to pick up every single item in my very expressive description, associating the correct actions as well. Although it sometimes needs extra information about the special items, it already saved a lot of effort than coding the room, which I doubt I will be able to do.” - Designer 1*

*”The experience was on the small and medium room was nice. It was quick and easy. It starts to get repetitive and boring when creating the big room.” - Designer 2*

The feedback gathered for the Room Generator was mainly positive, as the users agreed that their input was mainly processed correctly. However, it may be improved if the process of the room creation engine can extract more information from the user input and thus require less manual input for the items.

### 5.2.3 Freedom of Expression

Another major goal is to incorporate speech recognition into the gameplay without compromising the game’s performance while allowing players to express themselves.

*”The engine understood what I meant even though I did not refer to the object by its exact name. I was attempting to perform insane things, such as ”throw the dog,” but the engine responded with something reasonable to stop me. I believe the Game Processor handles these kinds of cases properly; therefore, I don’t feel constrained by the programme. However, the Game Processor will be more engaging if it can respond to questions such as ”What is in my bag?”” - Player 1*

To summarise, the users are satisfied with the level of freedom provided by both engines. Although their actions are still restricted by the available actions decided by the game designers, their input structure is not constrained, and the response

from the engine also makes the process feel humanised.

### 5.2.4 User Interface

As part of the user experience, although the user interface is not the top priority in the project, it may still play an important role in the users' impression of the engine.

*"The user interface is simple and easy to use because I can freely switch between the different input methods. However, the screen only shows the item's description but not the content of my bag. I was confused sometimes because I didn't know if I had picked up the thing successfully." - Player 1*

Summarising the feedback for the user interface, the lack of information about the result of the user actions on the room objects leads to a feeling of uncertainty for the users. This applies not only to the Game Processor but also to the Room Generator, as the information of the newly created item is not presented to the user. Therefore, they are unsure if the right items with the right type or fields are inserted into the room.

## 5.3 Limitation

The current implementations of Room Generator and Game Processor have granted users greater freedom and a novel gameplay experience. However, the existing program's restrictions prohibit it from attaining its full potential.

### 5.3.1 Manual Input in Room Creation

The Room Generator was capable of accurately identifying the object and its actions and could sometimes identify the thing as an unlock item automatically. However, there are times when users must manually enter the information by answering the subsequent questions. Although dividing the procedure into smaller parts can guarantee that the correct information is entered, this may become boring for users when many objects need to be created.

### 5.3.2 Fixed Response from Engines

The *engines* are only able to respond with the predefined responses whenever triggered. Although attempts have been made to diversity the engines' responses by returning a paraphrased text, the engines will be unable to provide an appropriate response if the user says something irrelevant or asks a question. The engines will only return "I am not sure what you want to do." or "Can you try again?" if no objects are detected.

### 5.3.3 No Duplicated Objects

The items in the *engines* are matched using the name of the synset. It does not store the characteristic of that items. For example, even if the *designer* put in ”There is a big golden key in the room”, the adjectives ”big” and ”golden” will be entirely neglected by the *engines* because they only search for nouns. Also, if an object of the same type but a different colour is introduced to the room, the *engines* will be unable to distinguish between them and treat them as a single entity.

# Chapter 6

# Conclusion and Future Work

The overall construction of the project has been successful and has exceeded my expectations. Given the positive user response, it can be said that an interface for extracting intent has been established successfully with the incorporation of advanced NLP techniques.

The *engines* have given users a great deal of flexibility of expression. Even when the input from the user is not recognised or comprehended by the *engines*, they respond appropriately to maintain a natural conversation with the user.

The application of intent extraction in the Game Processor has demonstrated the viability of incorporating speech recognition technology into games, decreasing the need for manual coding to match the user's input. Notably, the intent extracting capability not only applies to the escape room's gameplay but also enables the creation of games without the requirement for coding by the room designer, utilising voice input instead.

## 6.1 Future Work

The primary objectives are met by the end of the project, where we have an intent extract embedded in both the Room Generator that creates rooms and the Game Processor that keeps the user engaged in the game. This section suggests ways to maximise the capabilities of the *engines* and reach their full potential.

### 6.1.1 Automatic Room Content Extraction

The present Room Generator requires substantial supplementary user input to generate a solvable room. For special items, such as a locked box with a password of 1234, it would need further information, like how it is locked and the password. As a result, the Room Generator's item creation remains manual. Training a machine-learning model allows for automatic content extraction. Instead of providing brief responses to the Room Generator's prompts, the user might provide a whole paragraph describing the room. The model will automatically extract the name and

description of the item from the paragraph and classify the item's category.

### 6.1.2 Extend Object Types

The current *engines* only handle the three types of object, which is very limited. It can be improved by extending the accepted object type. For instance, the *player* may form new objects by merging the existing ones. Moreover, the *engines* can consider the objects' properties, e.g. colour and size, so that the same object with different properties can be added to the room, such as blue and red keys. This could lead to more diverse escape routes and a more engaging game.

### 6.1.3 Learning User Inputs for Room Improvements

The current implementation accepts only one object as the "escape object" that leads to a successful escape if unlocked. However, the players may have more creative ways of escaping the room, such as breaking the window instead of opening the door with a key. It could still be a valid escape method, but the room's designer did not program it. The solution path can be extended with different methods that the users tried, the escape game could be even more dynamic and interactive, as the solution would be less rigid.

### 6.1.4 Improvement in User Interaction

The Room Generator and Game Processor are only sensitive to verbs and nouns in the sentences. The users will receive an error message and be prompted to try again if any of these are missing. Although any response will be paraphrased before being sent back to the user, it may still be somewhat monotonous or repetitive. The interaction with the users would be significantly enhanced if the program could generate a response according to a different type of user input. In such situations, sentiment analysis might be helpful because it can interpret the users' feelings and help the *engines* generate a relevant response. For instance, if the players sound puzzled, the algorithm could provide tips or calm them down. The additional chatting feature could result in a more realistic user experience during the room creation and gameplay.

### 6.1.5 Application in Non-text-based Game

The power of the Room Generator and Game Processor is demonstrated as a text-based escape room game. It is possible to implement them in a visual escape room game, but it will require increased component efficiency optimisation.

# Chapter 7

# Ethical Issues

There are several ethical issues with this project that should be considered.

There could be an ethical risk when the Room Generator or the Game Processor takes users' voice input by recording their voices. The distinctive characteristics of a person's speech make voice recordings a type of personal data according to GDPR guidelines[42], as they may be used to identify an individual easily[43]. Therefore, consent must be obtained before any data collection, like voice recording during the gameplay or room creation. Besides, user responses have been collected anonymously, so the data collected cannot be used to identify the person who partook in the user feedback.

The *engines* are developed with the English model and, therefore, will only work for English speakers. The development of the engines has a risk of creating cultural bias, including race, gender, age and accent[44]. Google's speech recognition which is what the *engines* are using for the speech-to-text process, is found to be 13% more accurate for men than for that of women[45]. Also, it is suggested that Indian English has an accuracy rate of 78% while Scottish English has a 53% accuracy rate. This suggests that the *engines* may be susceptible to cultural and sex bias.

One of the ethical issues is that the users create the levels in the escape room games, and we have no control over their design of the room. The room's description may contain abusive or offensive language, such as swearing. This could cause distress in the users who are playing the level. However, this issue can be mitigated since Google's speech recognition service will block such offensive languages.

# Bibliography


[1] APEStudio. URL https://apestudio.itch.io/albedo. pages 5

[2] Wikipedia. Lifeline (video game), 2021. URL https://en.wikipedia.org/wiki/Lifeline_(video_game). pages 5

[3] Miguel Lopez. 'lifeline' (ps2) review, 2004. URL https://web.archive.org/web/20041029012230/http://www.g4techtv.com/xplay/features/493/LifeLine_PS2_Review.html. pages 6

[4] Human Interact. Starship commander: Arcade, 2020. URL https://human-interact.com/starshipcommanderarcade/. pages 6

[5] Wikipedia. Starship commander: Arcade, 2021. URL https://en.wikipedia.org/wiki/Starship_Commander:_Arcade. pages 6

[6] Kent Bye. Bringing conversational gameplay interactive narrative to 'starship commander', 2017. URL https://www.roadtovr.com/conversational-gameplay-interactive-narrative-starship-commander/. pages 6

[7] Michael Mateas and Andrew Stern. Natural language understanding in façade: Surface-text processing. *Technologies for Interactive Digital Storytelling and Entertainment*, page 3–13, 2004. doi: 10.1007/978-3-540-27797-2_2. pages 8

[8] IBM Cloud Education. Natural language processing (nlp), 2020. URL https://www.ibm.com/cloud/learn/natural-language-processing. pages 9

[9] Pos tags and part-of-speech tagging, Sep 2020. URL https://www.sketchengine.eu/blog/pos-tags/. pages 9

[10] Marco Maggini. Natural language processing, part 2: Part of speech tagging. URL https://www3.diism.unisi.it/~maggini/Teaching/TEL/slides%20EN/06%20-%20NLP%20-%20PoS%20Tagging.pdf. pages 10

[11] Data science: Natural language processing (nlp). URL https://www.oak-tree.tech/blog/data-science-nlp. pages 11

[12] Deep Mehta. Stages of natural language processing, 2020. URL https://byteiota.com/stages-of-nlp/. pages 11

[13] Dependency parsing in nlp (natural language processing), Dec 2021. URL https://www.analyticsvidhya.com/blog/2021/12/dependency-parsing-in-natural-language-processing-with-examples/. pages 11

[14] Dan Jurafsky and James H. Martin. *Transition-Based Dependency Parsing*. Prentice Hall, Pearson Education International, 2014. pages 12

[15] Semantic analysis: Guide to master natural language processing (part 9), Jun 2021. URL https://www.analyticsvidhya.com/blog/2021/06/part-9-step-by-step-guide-to-master-nlp-semantic-analysis/. pages 12

[16] Bhumika Dutta. URL https://www.analyticssteps.com/blogs/semantic-analysis-working-and-techniques. pages 13

[17] Eneko Agirre Philip Edmonds. Word sense disambiguation, 2008. URL http://www.scholarpedia.org/article/Word_sense_disambiguation. pages 13

[18] Ruslan Mitkov. *The Oxford Handbook of Computational Linguistics*. Oxford University Press, 2003. ISBN 9780198238829. pages 14

[19] Wikipedia. Word sense disambiguation, 2021. URL http://www.scholarpedia.org/article/Word_sense_disambiguation#Methods. pages 14

[20] Andreas Herman. Different ways of doing relation extraction from text, 2019. URL https://medium.com/@andreasherman/different-ways-of-doing-relation-extraction-from-text-7362b4c3169e. pages 15

[21] Rhea Sukthanker, Soujanya Poria, Erik Cambria, and Ramkumar Thirunavukarasu. Anaphora and coreference resolution: A review. *Information Fusion*, 59:139–162, 2020. ISSN 1566-2535. doi: https://doi.org/10.1016/j.inffus.2020.01.010. URL https://www.sciencedirect.com/science/article/pii/S1566253519303677. pages 15

[22] Paul Mc Kevitt, Derek Partridge, and Yorick Wilks. Approaches to natural language discourse processing. *Artificial Intelligence Review*, 6:333–364, 12 1992. doi: 10.1007/BF00123689. pages 15

[23] Dan Jurafsky and James H. Martin. *Coreference Resolution*. Pearson Prentice Hall, 2009. pages 15

[24] Eraldo Fernandes, Cicero Dos Santos, and Ruy Milidiú. Latent trees for coreference resolution. *Computational Linguistics*, 40:801–835, 12 2014. doi: 10.1162/COLI_a_00200. pages 15

[25] Jalaj Thanaki. *Python Natural Language Processing*. Packt, 2017. ISBN 9781787121423. pages 16

[26] School of English. What is pragmatics? URL https://all-about-linguistics.group.shef.ac.uk/branches-of-linguistics/pragmatics/what-is-pragmatics/. pages 16

[27] Dhruvil Karani. Introduction to word embedding and word2vec, Sep 2020. URL https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa. pages 16

[28] Bag of words: Approach, python code, limitations, Aug 2020. URL https://blog.quantinsti.com/bag-of-words/#:~:text=Limitations%20of%20Bag%20of%20Words,-Consider%20deploying%20the&text=The%20resultant%20vectors%20will%20be,making%20sense%20of%20text%20data. pages 16

[29] Long Ma and Yanqing Zhang. Using word2vec to process big text data. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 2895–2897, 2015. doi: 10.1109/BigData.2015.7364114. pages 16

[30] Vatsal. Word2vec explained, May 2022. URL https://towardsdatascience.com/word2vec-explained-49c52b4ccb71. pages 16

[31] Rani Horev. Bert explained: State of the art language model for nlp, 2018. URL https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270. pages 17, 18

[32] Maxime. What is a transformer?, Mar 2020. URL https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04. pages 18

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 06 2017. pages 18

[34] Author Cathal Horan. 10 things to know about bert and the transformer architecture, Jun 2022. URL https://neptune.ai/blog/bert-and-the-transformer-architecture. pages 18

[35] Mohd Sanad Zaki Rizvi. Demystifying bert: A comprehensive guide to the groundbreaking nlp framework, 2019. URL https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/. pages 18

[36] Princeton University. About wordnet., 2010. URL https://wordnet.princeton.edu/citing-wordnet. pages 19

[37] Dimitar Kazakov and Simon Dobnik. Inductive learning of lexical semantics with typed unification grammars. 01 2022. pages 19

[38] NLTK Project. Python nlp libraries: Features, use cases, pros and cons, 2021. URL https://www.nltk.org/. pages 21, 25

[39] Tomasz Bak. Python nlp libraries: Features, use cases, pros and cons, 2019. URL https://medium.com/@tomaszbak/python-nlp-libraries-features-use-cases-pros-and-cons-da36a0cc6adb. pages 21

[40] Industrial-strength natural language processing, 2022. URL https://spacy.io/. pages 21, 25

[41] Huggingface. Fast coreference resolution in spacy with neural networks. URL https://github.com/huggingface/neuralcoref. pages 25

[42] Gdpr : Why is voice considered a personal data ?, Jun 2021. URL https://mediartis.com/blog/gdpr-voice-is-personal-data/. pages 53

[43] Capital Flows. Voice recognition: Risks to our privacy, Oct 2016. URL https://www.forbes.com/sites/realspin/2016/10/06/voice-recognition-every-single-day-every-word-you-say/?sh=a16647b786d0. pages 53

[44] Posted byKristen Stephens. The role of ethics in voice assistant design, Nov 2021. URL https://voices.soundhound.com/the-role-of-ethics-in-voice-assistant-design/. pages 53

[45] Voice recognition still has significant race and gender biases, May 2019. URL https://hbr.org/2019/05/voice-recognition-still-has-significant-race-and-gender-biases. pages 53