Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Otso Ollikainen

Development and technical implementation of an interactive drama for smart speakers

Master's Thesis
Espoo, July, 2020

Supervisor: Associate Professor Tom Bäckström
Advisor: Wesa Aapro

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author: Otso Ollikainen

Title:
Development and technical implementation of an interactive drama for smart speakers

Date:	July, 2020	**Pages:**	89
Major:	Computer Science	**Code:**	SCI3042
Supervisor:	Associate Professor Tom Bäckström		
Advisor:	Wesa Aapro		

Smart speakers have rapidly gained popularity in recent years, with their virtual assistants, such as Google Assistant and Alexa, becoming increasingly familiar in many a household. The capabilities of virtual assistants and smart speakers are expanded by third-party applications, such as Alexa Skills. The functionalities of the different applications range from alarm applications to interactive fiction, but proper interactive drama is few and far between, even though there is demand for interactive content.

In this thesis, we will present different aspects of the development of an interactive drama for smart speakers, focusing on the technical implementation. The implementation consists of the interactive drama "The Dead Are Speaking", created for Google Home smart speakers as a collaboration between Yle and Aalto University. We will discuss different technologies and tools used for the implementation, factors of the format that affect the writing process, and the possibilities the concept brings.

Our evaluation demonstrates that the current methodology and tools are adept in creating interesting and complex drama and interaction structures, and that the users enjoy interacting with such formats. However, there are several limitations that restrict the functionalities and user experience of the format. These shortcomings are together with other improvement opportunities paving future prospects of deepened interaction models and enhanced dramatic qualities of interactive drama for smart speakers.

Keywords: interactive drama, radio drama, smart speakers, virtual assistant, Natural Language Processing, Natural Language Understanding, webhook

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tieto-, tietoliikenne- ja informaatiotekniikan maisteriohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Otso Ollikainen		
Työn nimi:	Interaktiivisen draaman kehitys ja tekninen toteutus älykaiuttimille		
Päiväys:	Heinäkuu 2020	**Sivumäärä:**	89
Pääaine:	Tietotekniikka	**Koodi:**	SCI3042
Valvoja:	Associate Professor Tom Bäckström		
Ohjaaja:	Wesa Aapro		

Älykaiuttimet ovat yleistyneet vauhdilla viime vuosien aikana, tuoden virtuaaliset assistentit, kuten Google Assistantin ja Alexan, yhä useamman kotitalouden tietoisuuteen. Virtuaalisten assistenttien kykyjä lisäävät kolmansien osapuolien kehittämät sovellukset, kuten Alexan Skills:it. Sovellusten toiminnallisuudet vaihtelevat hälytystoiminnoista interaktiiviseen fiktioon, mutta kunnollisen interaktiivisen draaman löytäminen on vaikeaa, vaikka kysyntää interaktiiviselle sisällölle olisi.

Esittelemme tässä työssä interaktiivisen draaman kehityksen eri näkökulmia, kuitenkin keskittyen tekniseen toteutukseen. Itse toteutus koostuu interaktiivisesta draamasta, "The Dead Are Speaking", joka luotiin Google Home -älykaiuttimille Ylen ja Aalto yliopiston yhteistyönä. Käsittelemme toteutukseen hyödynnettyjä työkaluja ja teknologioita, formaatin tuomia tekijöitä, jotka vaikuttavat kirjoitusprosessiin, sekä mahdollisuuksia, joita konsepti luo.

Meidän arviointimme osoittaa, että käytetyt metodit ja työkalut ovat päteviä luomaan mielenkiintoisia sekä monipuolisia draama- ja vuorovaikutusrakenteita, sekä että käyttäjät nauttivat vuorovaikutuksesta kuvatun formaatin kanssa. Formaattiin liittyy kuitenkin useita rajoitteita, jotka heikentävät sen toiminnallisuutta ja käyttäjäkokemusta. Nämä puutteet maalaavat yhdessä kehitysmahdollisuuksien kanssa tulevaisuudennäkymää älykaiuttimien interaktiivisen draaman syvemmistä vuorovaikutusmalleista sekä parannelluista tarinankerrontakyvyistä.

Asiasanat:	interaktiivinen draama, radiokuunnelma, älykaiutin, virtuaalinen assistentti, luonnollisen kielen prosessointi, luonnollisen kielen ymmärtäminen, webhook
Kieli:	Englanti

Acknowledgements

I wish to thank the people at Yle Beta, Wesa Aapro, Satu Keto, Mikael Hindsberg and Anssi Komulainen, and Lecturer Antti Ikonen from Aalto University for giving me this opportunity and supporting me along the whole process. I also want to thank my fellow students and colleagues, Mirka Oinonen and Rauli Valo, for bearing with me through the development process. I wish to express my gratitude for associate professor Tom Bäckström for his support and perseverance, and for pushing me through the writing process. Finally, I would like to thank my family for their support and understanding during these months.

Espoo, July, 2020

Otso Ollikainen

Abbreviations and Acronyms

API	Application Programming Interface
CLI	Command-Line Interface
CSS	Cascading Style Sheets
FLAC	Free Lossless Audio Codec
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IDE	Integrated Development Environment
IPA	Intelligent Personal Assistant
IVA	Intelligent Virtual Assistant
JSON	JavaScript Object Notation
MP3	MPEG-1 Audio Layer 3
NLP	Natural Language Processing
NLU	Natural Language Understanding
SDK	Software Development Kit
SSML	Speech Synthesis Markup Language
TTS	Text-To-Speech
WAV	Waveform Audio File Format

Contents

Chapter 1

Introduction

In recent years, brands like *Amazon Echo*, *Google Home* and *Sonos One*, have become familiar to many households, with virtual assistants, such as *Siri* or *Alexa*, becoming available in increasingly many devices. Smart speakers and their virtual assistants are offering a range of different operations with their capabilities expanding all the time — all with voice-activated commands or even natural conversation-like interaction. The natural language processing skills and built-in machine learning capabilities of the smart speakers offer a conversational platform for an abundant variety of possible functionalities ranging from private alarms to personalized music suggestions and to creating interactive fiction. But are their conversational capabilities properly utilised?

The popularity of smart speakers and other smart devices have been growing rapidly in the short span of time that they have been on the market. In the United States 34.4% of adults own at least one smart speaker, making it the leading country of smart speaker adoption. The expansion of the linguistic capabilities of the virtual assistants to support multiple languages has allowed the smart speaker market to spread into non-English speaking countries resulting in an seemingly ever-increasing user base. For instance, the estimated yearly growth in the number of installed smart speakers from 2018 to 2019 was 166% growth for China, 131% growth for Japan with a worldwide growth of 82.4%, increasing from 114 million to 207.9 million [1]. Therefore, smart speakers have been estimated to be one of the fastest growing device markets [2].

1.1 Problem statement

Despite the trends, in countries with no language support, the adoption rate is naturally slower. For instance, in Finland only 1% of the population uses

smart speakers [3]. However, smart speaker shipments in Nordic countries reached 900,000 in 2019, and there are 3.4 million daily listeners of radio in Finland, which is over 60% of the population, indicating interest and demand for audio content and voice-controlled interfaces [3, 4]. Despite the lacking support, there are hopes that the smart speakers expand their linguistic capabilities into Finnish in the future, as for instance Google has already added language support for other Nordic countries in its products, and Siri even talks Finnish already; however, Apple's *HomePod* smart speakers has yet to reach the Finnish market.

To address this opportunity, the Finnish radio company, Yle, wanted to try new kind of interactive content for smart speakers so that there would be an existing proof of concept when Finnish language support would be added. The development started in the end of 2018 as a collaboration between Yle Beta, the department of new media within Yle, and Aalto University. The supervision committee consisting of Yle and Aalto personnel assigned the execution of the project to an implementation team, consisting of three students from Aalto University. The main responsibilities were divided in three roles: scriptwriter, producer and sound designer, and head of the technical implementation and composer, the last two being my contribution. Despite the assigned roles, each member attended and contributed to many other tasks outside their main responsibility. The task was to create interactive content for smart speakers, which in the course of spring 2019 was refined into an interactive radio drama based on the infamous *Tattarisuo case* that happened in the 1930s. In November 2019, after six months of active development, the interactive drama called *The Dead are Speaking* was published as a public *Action*, an application for *Google Assistant* [5].

The Dead Are Speaking is an interactive story that runs on Google Home smart speakers and other Google Assistant devices. It employs voice actors rather than speech synthesis and has a relatively long running time compared to other Actions. Consisting of over three hours of content, the Action offers an average playtime of 90 minutes, making it presumably the lengthiest smart speaker drama yet. The interactive drama was created with tools that are suitable or intended for making Actions for Google Assistant. Consequently, this thesis covers the development and implementation of the project The Dead Are Speaking, discussing the methodology, technologies, tools, and other factors that affect the development of a smart speaker application, focusing on the technical implementation.

1.2 Structure of the Thesis

To understand the background of interactive drama and smart speakers, the background of audio based interactive drama and different smart speakers and their content is discussed in chapter 2. Chapter 3 delves into technologies and tools of smart speaker application development, establishing the basis for the implementation. To understand how the script is affected by the format, and how the script can affect the development, different aspects of the story and the script are reviewed in chapter 4, along with the setting of the story. The architecture and the significant traits of the implementation are discussed in chapter 5. The test methods and evaluation of the implementation are reviewed in chapter 6. Finally, chapter 7 concludes the main points of the thesis.

Chapter 2

Background

Humankind has always been drawn to stories. At first stories were told as an oral tradition, but since the invention of the written word, the stories have been able to grow more intricate in nature. Nowadays there are several popular story formats, such as movies, books, theatre and games — and a huge market for them. For instance, annual global box office revenue has climbed to over 40 billion US dollars in the recent years, and the gaming industry had a revenue of 120 billion in 2019 [6, 7].

The story-flow can be predetermined or changing as the story progresses according to certain factors. Most movies and books are predetermined. They always progress the same path from start to finish; even though the story itself might be told in a nonlinear manner, the story unfolds always in the same way and order. In turn, games utilise naturally an interactive approach, where the actions of the player evoke responses to the specific actions. This makes the player a proactive part of the experience — being of course natural due to the principal idea of a game. However, there are differences in the degree of interactivity. Some games might let the player progress in the story only within precisely predetermined frames and environments, whereas in others, the actions of the player might even drastically affect the outcome of the story.

Even though there is a demand for interactive drama and there has been implementations on different formats, such as the interactive film, *Black Mirror: Bandersnatch*, there is little proper interactive drama for smart speakers. This is despite the fact that Google has revealed that there are 500 million monthly Google Assistant users, and that the number of smart speakers has been increasing rapidly [8]. There is a lot of content for smart speakers, of which some are story based, but the only considerable attempt in interactive drama so far has been *The Inspection Chamber* by BBC. The Inspection Chamber is a science fiction comedy built initially for Alexa, which used real

actors and branching dialogue options [9].

2.1 Drama and interaction

Interactive drama for smart speakers has its roots in radio drama and *Choose your own adventure* type of interactive books. It is a new developing acoustic form of entertainment that shares many characteristics with radio plays, especially in the terms of sound design and aesthetics. Yet the story unfolds based on the choices and actions of the user, similarly to the interactive literature popularized in the late 1970s [10]. To understand the background for audio based interactive drama, we will now briefly discuss the history and elements related to radio dramas and interactive drama.

2.1.1 Radio drama

Radio drama or radio play is an acoustic dramatised performance for radio, established in the 1920s [11]. The popularity of radio plays was at its peak during the golden age of radio which marks the time when radio was the dominant form of home entertainment. Being one of the leading forms of entertainment in the 1940s, the popularity of radio plays declined in the 1950s, due to the arrival of television. However, radio plays still retained their position as an art form and experienced a new revival around 2010, when podcasting, streaming or downloading episodic digital audio content consisting of spoken words, took its form [12].

Radio drama was initially perceived differently in different countries and even between different radio stations. In most countries, radio plays were regarded as a literary art form opposed to being acoustic art. The focus of radio plays was placed on the word and the technical side was concealed, as to prevent the listener from getting distracted. Further division can be drawn whether to regard the radio drama to be a type of prose literature, as was the notion in Germany, or as a form of dramatic literary, which was the case for English speaking countries. Nowadays, the framework of the radio drama has shifted more into the direction of being acoustic art, and other aural elements are taken into account. Radio productions use a multitude of acoustic features, such as noises, music, or electro-acoustic manipulation in order to enhance the effect of storytelling [13].

Radio dramas are characterized by a relatively short running-time, rarely exceeding an hour and a half. There are usually also less characters compared to traditional literature, since distinguishing the characters can be hard for the listener without any visual cue. Minor characters can even be omitted.

The sense of environment can be created with narration, incorporating the descriptions in the dialogue, or with sound-effects. Sound-effects constitute a prominent part of radio drama. While they can have a potent effect, they can in some situations remain disconnected and unidentified if the listener is not given a proper context of what they present. Additionally, exaggerated effects are often employed to emphasize the action — it is not necessarily about how objectively realistic the effect sounds like, but rather how the listener or the character subjectively experiences it [14].

There have been implementations of interactive radio drama, such as the detective series Der Ohrenzeuge, broadcasted between 1993 and 2015 on Radio Fritz [15]. The play presented a fictitious detective team upon unsolved mystery cases, and the listeners could call the radio station to suggest what to do next. However, the story was fixed and the role of the listeners was to find the correct clue in order to progress in the story. The current state of technology enables creation and easier access to more personalised interactive radio experiences, such as BBC:s The Inspection Chamber.

2.1.2 Interactive drama

Interactive drama is a form of drama which uses interactive narrative in unraveling a non-predetermined story-line. In other words, the interaction between the user or the player and the story creates the story progression. While the term *interactive drama* can be ambiguous and is sometimes differentiated from concepts such as interactive fiction, we will consider interactive drama in this context to be a unique non-predetermined story that unfolds based on the user's interaction with the story. Typical elements of interactive drama include choices presented to the user that affect the outcome of the situations, branching story-lines, and multiple endings. Additionally, the interaction between the player and the story or the environment happens within the narrative boundaries, or the story space, consisting of the set of different possible story experiences [16].

Interactive game-books can be seen as early predecessors of interactive drama. Perhaps the most famous example, Choose Your Own Adventure series has become almost synonymous with game-books. In interactive literature, the narrative branches based on the choices of the reader, usually by utilising numbered pages or paragraphs. Choose Your Own Adventure series is also referenced in the interactive movie Bandersnatch. Another early form of interactive narrative can be found from interactive fiction, also called text adventures. In text adventures, textual commands are used to control the characters and influence the environment, resulting in a textual output of the game-state. One approach to an interactive story-telling system, that utilised

natural language understanding (NLU) and artificial intelligence (AI), was the interactive story Façade, released in 2005. Façade combined procedural simulation and structured narrative, in order to bind hand-crafted pre-made events and the unexpectedness of the user input [17]. It won the Grand Jury prize in 2006 Slamdance Independent Game Festivals and some consider it as the first true interactive story-telling software [18]. Subsequently, interactive narrative has been employed in a number of video games, such as *Heavy Rain* or *The Witcher* series.

There are different approaches to interactive systems, such as immersive storytelling, emergent storytelling, plot based systems and character based systems [19]. Despite the variety of approaches a set of key problems in interactive storytelling has been identified: a trade-off between interactivity and storytelling, the duality between character and plot, narrative causality, the problem of narrative control and relations between story generation and presentation [20]. In addition, the attributes of the used technology and characteristics of the medium have a significant impact on the system.

At the architectural level an interactive drama system consists typically of three components: the drama manager, the agent model and the user model. The drama manager is acting as a story engine, managing the narrative logic and the plot. The agent model is the actor engine which controls the behaviour of non-playable characters. The user model tracks the interactions and inputs of the user [21]. The three parts can be interpreted as interdependent components, emergent from the user's actions. In simple systems characters might even manifest qualities only through the plot narrative, making the agent model part of the story engine. The user model can also be affected by the other two components. For instance the plot or a certain character can encourage the user to make certain decisions.

Interactivity can be characterized by three levels of interaction: non-interactive, reactive, or responsive [22]. Providing content regardless of the user input is considered non-interactive, content that is a direct reaction to user input can be seen as reactive, and if the system takes the actions of the user in to account more holistically it can be regarded as responsive. Video games, along with other forms of new media, are characterized by interactivity [23] — they simply require user input in order to progress. From text based interactive fiction to modern role playing games interactive story elements have been utilised in varying degrees. Interactive games offer personal experiences allowing the players to immerse themselves in the recurring environment, time and role of the game.

2.2 Smart speaker

Smart speaker is a voice command device (VCD) controlled or interacted with a voice user interface (VUI). Smart speakers are powered by an integrated virtual assistant that allows interaction between the user and the speaker. A smart speaker with a screen is called a smart display. There are a variety of smart speaker platforms available, with Amazon having introduced the first one, Amazon Echo, in 2014 [24]. Since the initial release, Amazon Echo series of smart speakers and smart speaker accessories have paved the way for smart speaker development by introducing several new products and features. Amazon Echo family includes products such as the original middle sized Amazon Echo smart speaker, small puck sized Echo Dot, a portable version of Amazon Echo called Amazon Tap, smart display Echo Show, camera equipped Echo Look, Echo Auto for cars, and accessories such as Echo Buttons for playing games, Echo Sub for additional subwoofers, and Echo Clock for showing timers for Echo devices [25].

Google has brought forth similar products, premiering with the mid-sized Google Home in 2016, and followed by a smaller smart speaker Google Home Mini and later Google Nest Mini, its second generation variant, a larger speaker Google Max, smart display Google Nest Hub, and a bigger smart display equipped with a camera, Nest Hub max [26]. The trend has been followed by multiple parties: Apple revealed its own smart speaker product, HomePod, in 2017, and Sonos is marketing its product, Sonos One, as the "smart speaker for music lovers" [27, 28]. Additionally, Chinese companies, such as Baidu, Alibaba and Xiaomi, have also manufactured their own products, and especially Baidu has enjoyed a surge in smart speaker shipments [29]. The smart speaker market is currently led by Amazon followed by Google, the two taking up roughly half of the market in 2019, with market shares of 29,9% and 19,1% respectively. The rest consist of Chinese electronic companies Baidu (13,9%), Alibaba (13,5%) and Xiaomi (11,3%) and other companies (12,3%) [30].

2.2.1 Virtual assistants and the capabilities of smart speakers

Virtual assistant, also called an intelligent virtual assistant (IVA) or an intelligent personal assistant (IPA) is a computer program that performs actions on behalf of the user, based on the commands or questions given to it. There are different types of virtual assistants, based on their function and input methods, with the most common input methods being text and voice input.

For instance, virtual assistants used in online chat services are often called chatbots, and virtual assistants that can be interacted with using voice and that respond with synthesized speech, are called voice assistants. Typical platforms for virtual assistants include mobile devices and their operating systems, smart devices, like smart speakers, messaging apps, mobile apps, chat services and appliances, such as cars [31].

The voice assistants usually listen to a wake-up phrase or a keyword, such as "Ok Google" or "Alexa", that activates the assistant to pay attention to the user. Once the assistant recognizes the keyword, it starts recording the user and sends the input to a specialized server, which analyses and processes the input using natural language understanding (NLU), and matches it to executable commands. Voice assistants differ from earlier voice activated technologies in that they are always connected to the internet and the processing is done in the cloud, thus allowing access to larger pool of information and therefore more comprehensive responses. Many assistants use machine learning techniques to continually train themselves [32].

While each assistant has their distinct features they share many common functionalities. For instance, most voice assistants are capable of carrying out basic tasks, such as sending and reading messages and emails, making phone calls, answering informational queries, setting timers, alarms, reminders and calendar entries, creating lists, controlling media playback from connected services, such as *Spotify*, *Netflix* and *iTunes*, controlling Internet of things devices, such as lights, thermostats and locks, and telling jokes and stories. In addition, the abilities of voice assistants can be expanded by accessing third-party applications developed for the specific assistant. For instance *Actions* are applications created for Google Assistant and *Skills* are applications for Alexa.

Recent advancements in natural language processing (NLP) have improved the functionality and conversational capabilities of the virtual assistants, allowing them to respond quickly with meaningful responses. These improvements are achieved by four factors: a vast increase in computing power, the availability of very large amounts of linguistic data, the development of highly successful machine learning methods, and a much richer understanding of the structure of human language and its deployment in social contexts [33]. The cheaper and more powerful personal computers and the availability of human generated online text for analysis has allowed virtual assistants to be trained to listen and respond to requests in meaningful and natural ways. For instance, if the user would like to know where the car was parked, the user could ask about location in a natural manner, like "Where did I park my car?", "Remember where I left my car?", "Where is my car parked" or "Where did I leave the car", and the assistant would

probably trigger the expected response. While the processing follows a certain model, namely matching keywords and phrases, there is no need of using and finding the specific phrases or patterns that were required in earlier voice recognition systems, thus avoiding the user frustration due communication errors [32].

Siri, developed by Apple, was the first virtual assistant that was incorporated in a smartphone, being included in the iOS in 2011. Siri was followed by *Google Now* in 2012, which extended *Google Search* of *Google app* for Android and iOS devices. Google Now presented informational cards to the user, that were based on the users habits, and tried to predict what the user might need. Microsoft introduced *Cortana* in 2013. The first virtual assistant to operate in a smart speaker, was Alexa, developed by Amazon. Alexa debuted in the first Amazon Echo smart speakers in 2014. In 2016 Google presented their virtual assistant, Google Assistant, as a part of the Google's messaging app *Allo* and later Google Home smart speakers [24].

Many assistants can be integrated into third party hardware utilising their respective software development kits (SDK). For instance Sonos smart speakers do not have a dedicated virtual assistant, instead both Alexa and Google Assistant are built-in [28]. While Alexa is the main assistant in Amazon's smart speakers, the users can connect to Microsoft's virtual assistant, Cortana, by adding Cortana as a Skill to the device. Alexa can be embedded into third party devices and hardware, using *Alexa Voice Service Device SDK*. The SDK offers different components to handle needed functionalities, providing a direct access to cloud-based capabilities of Alexa. Similarly, Google Assistant can be integrated into third party hardware using *Google Assistant SDK* [34, 35].

2.2.2 Content for Smart speakers

The amount of Alexa Skills has been accounted to surpass 100 thousand, although, not all of the Skills are available in one country. The most prominent amount of Skills belongs to the US region, having access to over 65 thousand Skills [36]. While being in the market for a shorter duration, Google promotes a selection of over million Actions [37]. However, there are reports of the number of Actions being lower, for instance counting a bit under 19 thousand English Actions. It is hard to determine how many Actions there are, due to the way the information about the Actions is displayed, and because Google does not report official numbers for the Actions [38].

Both Actions and Skills have categories for different types of functionalities, such as business and finance, communication, education and reference, games and fun or games and trivia, health and fitness, kids or kids and family,

music and audio, productivity, and shopping. Many of the categories contain plenty of content. For instance for Alexa, games and trivia gives over 10,000 Skills as a result. For Actions, games and fun reveals further categorization, including quizzes, jokes, puzzle games, and adventure games. However, many of the Skills and Actions seem to be simple applications that perform a single function, such as giving a quote, presenting a trivia question, or reading a story. Both Skills and Actions have some interactive stories, such as *Storyflow - Interactive stories* Skill, or the Actions *Strangest Day Ever*, *The Darkness at Innsmouth* or *Jungle Adventure* [39, 40].

2.2.3 Privacy concerns

Smart speakers have raised several concerns about the privacy of the user and the security of the smart speakers. If the microphone is not turned off, the smart speakers listen continuously for the wake-up word. Once recognized, the speaker starts recording the user and sends the recording to the server for processing. However, there have been several cases of false awakenings where the device has started to record when hearing the wake-up word mistakenly, and even sent a private conversation to a random contact [41–43]. There have also been unexpected issues, that have posed privacy risks, such as Google Home mini touch control flaw, which resulted in the device to start recording on nearly every sound [44].

Privacy can be a major deterring factor for smart speaker adoption [45]. However, some users are indifferent or ignorant about their privacy. A study found, that nearly half of the respondents of the survey knew that their recordings are stored indefinitely and only 5% mentioned using the feature for disabling the microphone [46]. While the privacy of the smart speakers leave a lot to be desired, with only Apple's HomePod passing the privacy benchmark conducted by Common Sense Media, many a user accept trading their privacy for convenience that the speakers provide [45, 47].

Chapter 3

Environment

Smart speakers provide content through their integrated assistants such as Alexa, Google Assistant and Siri. Thus, to develop content for smart speakers means essentially developing content for the assistants the smart speakers use. There are currently two major platform options for smart speaker application development — Alexa and Google Assistant. There are also other possible candidates, for instance Siri, if one wished to create an applet for iOS or HomePod, but which have not accumulated as prominent user base or development options — at least yet. As for the language support, Google Assistant has an advantage in terms of variety of languages, having support 19 languages for smart speakers, while Alexa currently supports eight [37, 48].

The architecture of an application often structures the composition of the application enabling and enhancing or disabling and hindering certain aspects or characteristics of it — let alone of the effect it has on the development process itself. There is even a Finnish proverb that goes "Well planned is half done". The goal was to create an interactive drama for a smart speaker, lasting roughly from 15 minutes to 30 minutes. One of the first decisions is choosing the platform for the drama. There are some questions that can be helpful in that matter: Are there any specific platform preferences, such as preferred smart speaker? Is the language something else than English and in that case which speakers support it? Is the goal to be available to as large a user base as possible? If the exact platform itself is irrelevant and the language is English, both Google Assistant and Alexa are viable options. Both options have adequate tools for development of different magnitudes, and well documented processes, application programming interfaces (APIs), SDKs, and other resources, although, Google seems to have a slight edge in terms of clarity and continuity in the documentation.

Our implementation was a prototype of making content for smart speakers, serving as means of preparing for the possible upcoming Finnish language

support. Therefore the language was a prominent factor. This made Google Home a more suitable candidate, being more probable to receive Finnish language expansion, since it already had support for Nordic languages, such as Swedish and Norwegian. In order to understand the different concepts and tools related to the development of smart speaker applications, we will now discuss different elements regarding virtual assistant applications and the conversational application platform *Dialogflow*, which forms the core of many virtual assistant applications. We will focus on concepts related to Google Assistant, since it is the platform of the implementation.

3.1 Virtual assistant applications

Unlike traditional applications the virtual assistants use cloud based technology to process the user input and access different functionalities. The applications, such as Skills for Alexa or Actions for Google, are developed in their respective environments and deployed in the cloud. In other words, there is no need to download the Skill or Action. Google Assistant can access all the published Actions directly from the cloud using the invocation phrase of the Action or finding the Action by the exploration function, while with Alexa, the user needs to enable the Skill in order to use it. There are some different approaches for building virtual assistant applications, ranging from using a local environment and resources to using an integrated development environment (IDE) and ready-made building blocks created for the dedicated purpose. However, the documentations naturally emphasize supported methods.

3.1.1 Elements of a virtual assistant application

On an abstract level, the task of a virtual assistant application is to fulfill the requests of the user. When a user makes a request to a Skill or an Action the corresponding virtual assistant matches the user input to the defined functionalities of the given application. Typically, the virtual assistant applications are characterized by matching the perceived intention of the user to a structure that is often called an intent. An intent represents the action that is executed to fulfill the request of the user. Thus, intents form the core functionality of a Skill or an Action. The workflow for different applications between different virtual assistants is similar and therefore shares many similar concepts, such as training phrases and sample utterances or entities and slots, even though they may not share a common name.

The workflow for a typical virtual assistant application is illustrated in

figure 3.1. To start the interaction, the user wakes up the virtual assistant and launches the application by saying the invocation of the given application, such as "Hey Google talk to X" for an Action. The assistant launches the matched application and the conversation shifts into a two-way dialogue between the user and the application. The assistant structures the user input and matches it to the intents using the training phrases or the sample utterances of the intents, invoking an appropriate intent. The training phrases or sample utterances may contain words that are resolved to predefined variables, called entities or slots, that act as keywords for matching and processing. The intent returns a defined response or triggers a webhook request for the web service in order to return a dynamic custom response. The web service returns the response for the application which forms the final result for the assistant. Depending on the application, the response usually contains either a follow up question or marks the end of the conversation, if the needs of the user are fulfilled. Virtual assistants typically use text-to-speech (TTS) or Speech Synthesis Markup Language (SSML) for generating responses.

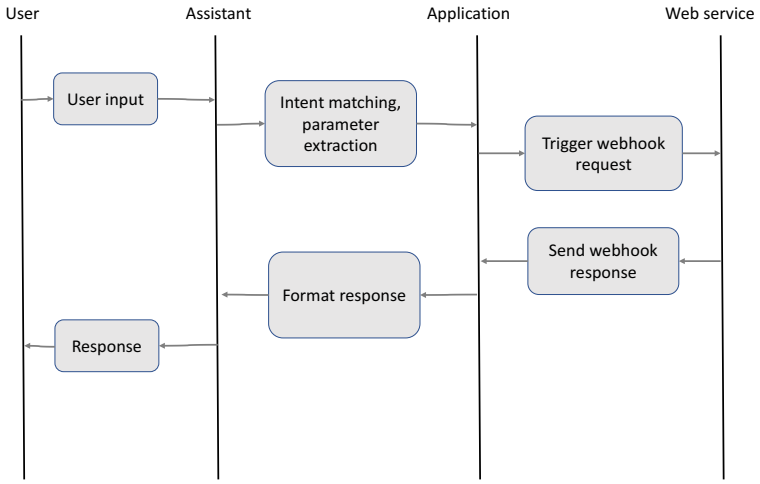


Figure 3.1: The workflow for a typical virtual assistant application

What happens if the agent cannot associate the request with any intent?

Or if the speaker does not receive any input? In such cases, a special intent called a fallback is triggered. Fallback intent can give a generic response, like "I didn't quite catch that" or it can be configured to give dynamic responses depending on the situation. However, there are some restrictions. For instance, Google Assistant forces a conversation exit after three consecutive failed matchings for an intent, regardless of the configuration of the fallback.

3.1.2 Speech Synthesis Markup Language

Speech Synthesis Markup Language (SSML) is an XML-based markup language for assisting the generation of synthetic speech. SSML offers a standardized way to control different aspects of speech, such as pronunciation, volume, pitch and rate [49]. With SSML one can control how the virtual assistant generates the response, for instance by adding pauses and sound effects. However, assistants use a subset of the SSML specified for each assistant. There are also some non-standard SSML features that enable extended functionality to the standard SSML. For instance Actions on Google platform is provided with a *par* element that allows parallel execution of other elements. Similarly, Alexa has an *amazon:emotion* element for controlling the expressed emotion and its intensity [50, 51].

Google Assistant supports many useful SSML elements, such as *prosody*, *break*, *audio* and *media*. The root element of a SSML response is *speak*. It wraps around all the other elements. Elements *prosody* and *emphasis* can be used to control prosodic qualities of the speech. The current support for the *prosody* element allows the control of rate, pitch and volume attributes. *emphasis* allows placing prominence on the generated text, adjusted by the optional level attribute, such as *strong* or *moderate*. *break* element can be used to add pauses or other prosodic boundaries between segments. *say-as* element indicates how the containing information is presented, allowing for instance numbers to be spoken as ordinals or words as characters. *audio* element allows the insertion of audio files into the output, with controllable attributes such as ending and starting times, sound level and playback speed. *media* element depicts a media layer that can wrap *audio* and *speak* elements. *par* element is a media container that allows simultaneous playback of multiple *seq*, *media* or other *par* elements. *seq* element is a sequential media container for consecutive playback of *seq*, *par* and *media* elements. The Action format limits the duration of SSML to a maximum of 240 seconds and the file size to five megabytes. Additionally, the source URLs must use HTTPS protocol.

3.1.3 Google Assistant and Google Home devices

There are several devices available from Google Home or Google Nest family, the latter indicating the second generation of Google smart speakers and smart displays. Each device features bluetooth support, far field microphones and capacitive touch controls. Each device supports several audio formats, including MP3, WAV and FLAC with support for high-resolution streams. Thus, each speaker has similar audio capabilities with differences in the output quality and the number of microphones. While most of the devices have a mono output, each speaker can be paired wirelessly with a similar speaker to form a stereo pair [52]. This should be taken into account when creating soundscapes or otherwise mixing audio for smart speakers — the user listens to the audio probably in mono but there is a chance of stereo. Even though mobile devices would have a stereo output, for instance with headphones, the Actions use mono output in mobile devices, at least at the time of the implementation.

The Google Home smart speakers are set up by using Google Home application with a device that is in the same wireless network as the speaker. Also, the speaker requires a constant connection to operate, since the processing is performed in the cloud. Each Google Home smart speaker has access to the same Actions. However, that is not necessarily the case with other devices. Depending on the surface capabilities of the device, some Actions may not be available to that specific device. For instance, some Actions may require a screen output, and thus be unavailable for smart speakers but operable on smart displays.

There are some different ways to build content for Google Assistant, such as integrating services and content, like mobile applications or web content, with the assistant, or creating customised conversational Actions with *Actions on Google* development platform. Android applications can be extended to the assistant with deep links that link the users to the specific activities in the application. The mobile app extension is built via *App Actions* defined in actions.xml file that is registered to the mobile application. The XML-file includes app capabilities semantics as built-in intents and fulfillment instructions. To present web content, such as podcasts, recipes, news, guides and more to the user in an interactive way, it can be marked up with structured data using *schema.org* defined vocabulary. This allows Google Assistant to automatically generate an Action for the content, allowing the users to access the content through the assistant.

3.1.4 Actions on Google

Actions On Google is a developer platform for creating applications for Google Assistant called Actions or Conversational Actions. When in conversation, the Conversational Actions handles the requests from the Assistant and returns responses containing audio and visual components. Actions can be accessed by Google Home smart speakers or other devices that have Google Assistant once the Action is successfully published. The Action is created and configured in the *Actions console*. The console can be used for management of the configurations of the created *Actions project*, such as the project settings, resources, directory information and deployment options. The console also offers a simulator to test the Action with different surface capabilities, and access to analytics for the usage of the Action. When creating an Action one can select the foundation for the Action from different categories, varying from a blank custom Action to s ready made trivia template. The selection of the category affects the components, such as intents, that are included in the created project, if any. The templates even allow building of simple actions, such as quizzes, with minimal configuration and without coding using *Google Sheets*, Google's online spreadsheet.

There are currently three supported ways of developing an action: with *Actions SDK*, with *Dialogflow* or the newest option announced in June 2020, *Actions Builder* [53]. Prior to the update the main choice for creating conversation fulfillment was between Dialogflow and Actions SDK. The main difference between them was how the fulfillment was built. The legacy Actions SDK lacked natural language understanding (NLU) capabilities and therefore it was up to the developer to provide one — for instance to map the intents to their fulfillment and to provide query patterns as example phrases of what the user might say. Dialogflow, on the other hand, provided a conversational platform with a NLU solution and machine learning capabilities that can be wrapped in the functionality of Actions SDK, which made it the most popular tool for creating Conversational Actions [54]. After the update all three approaches come with NLU capabilities and potential workflows to create Actions.

Action SDK is a set of development tools consisting of a standardized open file-based schema to build and represent an Action, libraries to interact with the assistant, such as Node.js library for webhooks, and a command-line interface (CLI) for management and deployment of the Actions project. Action Builder is a web-based IDE integrated in the Actions console and built on the functionality of Actions SDK. Action Builder offers a graphical interface for managing different elements of Actions SDK, such as visualizing the conversation flow and managing NLU training data, enabling develop-

ment and deployment in the Actions console. Action Builder can be used in conjunction with Actions SDK. The resulting Action uses the *Assistant Natural Language Understanding* NLU solution.

Action console offers the following fundamental components for building an Action: Actions project, invocation model, conversational model and the webhook. Actions project is a logical container of the Action. It defines project settings, resources and deployment on Actions on Google. Invocation model specifies how the Action can be invoked, starting the conversation with the Action. In turn, conversational model defines conversational rules, describing how users interact with the action after it has been invoked. It specifies, for instance, what users can and may say, and how the Action will respond to the user inputs. Webhook can be used for performing additional or external logic, such as generating dynamic responses, validating the user inputs with back-end data or placing orders in external systems. Additionally, Actions on Google offers tools for supplemental operations, such as transaction functionality and user engagement activity, like push notifications and daily updates.

An Action can be invoked according to the explicit invocation of the Action or through specified invocation phrases. Explicit invocation means that the Action is invoked with the name of the Action, being typically of the form "Hey Google, talk to The Example Action". Optionally, one can define deep links, invocation models that allow the user to invoke specific intents or functions of the Action. For instance, instead of invoking an Action called "Wine recommender" and then asking for a recommendation for red wine, the user might say "Ok, Google, talk to the wine recommender about red wine" which would directly lead to the correct intent and recommendation. Invocation phrases can also be used to invoke an Action implicitly without using the name of the Action. This implicit invocation occurs when Google Assistant receives a user request with no Action name provided, such as "Hey Google, recommend me a white wine". Google Assistant attempts to match the user request to a suitable Action, search result, or a mobile app, and returns it as a recommendation to the user. Therefore, providing implicit invocation phrases increases the discoverability of an Action [54].

The conversation between the user and an Action consists of user requests and Action responses. The interaction is built by defining valid user inputs, logic to process the inputs and corresponding prompts to respond back to the user. Actions SDK components for achieving the interaction are intents, types, scenes and prompts. Assistant NLU matches the user input to an appropriate intent according to the language model of the intent. The language model is defined by specifying training phrases, examples of possible user inputs. The assistant then expands upon the training phrases, creating

the language model of the intent. Additionally, assistant NLU can extract parameters from the user input, defined by annotating the training phrases for the parts to be extracted. The parameters have an associated type, such as date or a name, or they can be something custom defined. Scenes serve as the logic executors of an Action, processing the matched intent. Scenes also send the response prompt back to the user.

Prompts specify how the Action renders the response. Google Assistant currently supports five types of responses, simple responses, rich responses, visual selection responses, media responses and interactive canvas. Simple response use text-to-speech (TTS) or speech synthesis markup language (SSML) for sound and chat bubble for visual part of the response. Simple response is the only response that is supported on every device type. Rich responses can contain visual and functional elements, such as images and buttons, that are displayed as cards. Visual selection responses allow the user to select an option of a set of items, displayed as a vertical list or a horizontal collection. Media response provides an media interface for playing audio content, allowing for playing longer lasting audio files than with the SSML. Interactive canvas is a web framework for creating and adding immersive visualizations, such as animations and graphical user interfaces (GUI), to actions. Interactive canvas functions as a full-screen interactive web application that is sent to the user as the response of the conversation. The canvas is built with web technologies such as HTML, CSS and JavaScript, having a dedicated Interactive Canvas API JavaScript library.

While being deployed in the cloud as a test version, and accessible with different Google Assistant devices during the development phase, to become accessible to others, the Action needs to be published. There are three levels of releases. Alpha release is a channel for testing the early version of Action for a defined set of maximum of 20 people. Alpha release does not require a full verification or policy compliance review of the Action. Beta release allows the Action to be distributed for a larger set of people than Alpha and needs to pass Google's full review of the Action. The maximum number of defined users for the release is 200 and once the beta version has passed the review, the Action can be made public at any time without another review. Once the Action is ready for the public, it can be deployed to production. The alpha and beta releases support most of the features of an action, but for instance templates or smart home functionalities are not supported.

Google offers a comprehensive documentation and learning resources depicting the different functionalities of Action on Google and examples for them, such as Google's codelab tutorials, Actions on Google documentation and GitHub repositories [54–56]. However, the tools and services are in active development and many practices have become updated and some even

deprecated or flagged for deprecation, such as Node.js 8 engine and Dialogflow V1 API [57, 58]. As such, it is always important to review the current state of the different applicable technologies when starting the development.

3.1.5 Webhook

A webhook is a method of altering the state of a web application with a custom callback from an external web service. Actions use webhooks as a form of fulfilment — fulfilling the user's request by augmenting the functionality of an intent. Actions usually trigger a webhook for instance to handle internal logic, such as checking the parameters, reading and writing session data of the conversation, triggering follow-up events and checking the capabilities of the users device, and to generate responses. Webhook consists of handlers that are correspondent to the intents whose fulfillment they execute. To enable webhook callback for an intent it needs be enabled for the specific intent.

When a webhook is triggered Google Assistant sends a request with a JSON payload to the defined fulfillment. The request contains the name of the handler to process the event, information about the conversation, such as current intent, and the end-user expression. The webservice or fulfillment endpoint processes the request and returns a response with a JSON payload. The requests are handled using HTTPS protocol and the URL must be publicly accessible. It is recommended to authenticate the webhook requests to ensure that only authorized parties can make requests.

While the request and the response is delivered as a JSON payload, it might be beneficial to implement the webhook using a Node.js library that supports Actions features, such as Actions on Google or Dialogflow Node.js client library. The Node.js library wraps the conversation webhook API and the low-level details of the JSON payload in a layer of abstraction making the implementation easier and more straightforward. It also handles the communication between the fulfillment and the Actions on Google platform. Actions on Google Node.js library is the recommended way of implementing the webhook for an Action, since it supports all features of Actions on Google API [59].

Fulfillment webhooks are usually hosted on serverless computing platforms such as *Cloud Functions for Firebase* or *AWS Lambda*. They can also be hosted on self-hosted servers, for instance using Express Node.js web framework. Additionally, Dialogflow and Action Builder offer built-in code editors that allow creating fulfillment in their respective development consoles and deploying them to the Cloud Functions.

3.2 Dialogflow

Dialogflow is a framework for natural language based conversational applications. It is a Google service running on Google cloud platform utilising Google's machine learning resources. A Dialogflow agent can be interacted with multiple types of inputs, mainly text or audio inputs, and it can respond back to the user using text and audio, usually with speech synthesis [60]. Dialogflow agent can be integrated to several platforms, including Google Assistant, and it can be exported in Alexa skill format to be imported to Alexa skills. Other platforms can interact with Dialogflow using Dialogflow's API, and managing the interaction with the end-user themselves.

The standard edition of Dialogflow is free of charge, but it contains more constraints than the paid enterprise edition, mainly concerning quotas and some other limitations. For instance the input quotas for standard editions are 180 text and 100 audio requests per minute, 1000 audio requests per day and 15000 audio requests per month with maximum audio length of 60 seconds. Whereas, the corresponding figures for enterprise edition are 600 text requests and 300 audio requests per minute. These quotas can be increased upon request, for instance in the context of Google Assistant Actions. It should be noted, however, that a Google Assistant input and output is considered a text request [61].

3.2.1 Dialogflow agent

A Dialogflow *agent* is a NLU module and a top-level container holding the settings and the data of an Dialogflow application. It also handles the conversation with the user. The conversation between the user and the Dialogflow agent is called a session. The core components of a Dialogflow are intents, entities, contexts, events, integrations and fulfillment. Dialogflow agent is usually created by using the Dialogflow console but it can be also created with Dialogflow API.

3.2.2 Intents

Intents are representations of the assumed intention of the user similar to other virtual assistant application schemes. An intent can also be seen as an actualization of the state of the conversation that is invoked every time a request is passed to the agent and thus, is the most basic block of content in a Dialogflow application. For example one could build an application by defining only intents and their contents, but a Dialogflow application without intents would not have any functionality. The functionality of an intent is

dependent of the components associated with the intent, shown in figure 3.5, such as *training phrases*, *actions and parameters*, and the *response* of the intent.

The matching or intent classification is based on the training phrases and the contexts of the intent. Training phrases are examples of end-user expressions - in other words what the end-user is expected to say. The practice is to provide a sufficient amount of varying training phrases so that the agent is able to match the correct intent when an end-user expression resembles the training phrases. For instance if an intent expects an answer to the question "What is your favourite food?" some of the training phrases could be: "I like pizza", "It's lasagna", "Fish and chips" or "My favourite food is a good steak". Not all the possible alternatives need to be declared because the built-in machine learning expands the given list with similar phrases. However, it is recommended to give at least ten alternatives [61].

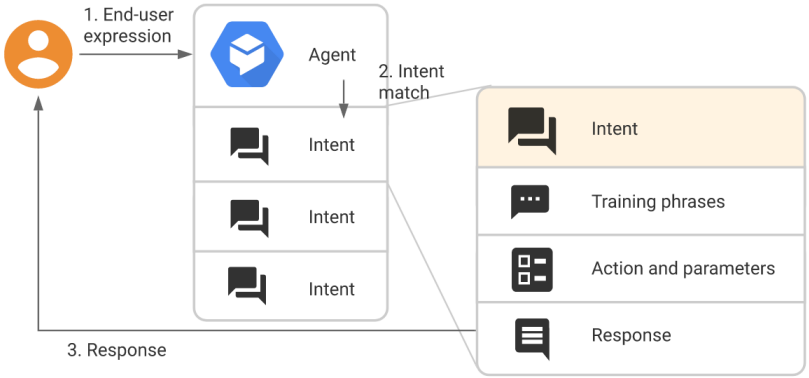


Figure 3.2: Components of an intent [61]

3.2.3 Contexts

Another factor affecting the matching of the intents are *contexts*. Contexts denote the contextual state of the conversation. They can be used to control the flow of the conversation by creating a logical routing for the agent, defining which intents can be matched in which point of the conversation. There are two kinds of contexts - *output contexts* and *input contexts*. The input context decides whether intent can be matched or not, while the output context sets the context to an active state.

An intent is only matched if all of its input contexts are active and the end-user expression matches its training phrases. The input contexts do

not need to correspond to all active contexts, although, intents with input contexts matching most closely the active contexts are also most likely to be matched. From this, it follows that intents with no input contexts can be matched anytime, but intents with input contexts have a higher priority for matching. However, when there are no active contexts only intents with no input contexts can be matched.

Once intent is matched it activates its output context. Output contexts have a lifespan attribute that specifies the lifespan of the context, or the number of conversational turns the contexts stays active. Contexts can also serve as a temporary container for collected parameters that can be accessed as long as the context is active. When an intent with a parameter and output context is matched the parameter is stored in the output context and the following intents can refer to that value during its lifespan.

3.2.4 Action and parameters

When an intent is matched the agent can provide *parameters* based on the training phrases and an *action* to help in execution of additional logic. The *action value* is a value from the action field that is passed to the *fulfillment* webhook request or API interaction response. Parameters are structured data extracted from the training phrases that can be used for creating dynamic responses or perform other logic inside the application. Each parameter is linked to an entity type which specifies what kind of data it accesses. Parameters can be added in the Dialogflow console by annotating a part of the training phrase and selecting the corresponding entity type, the result illustrated in figure 3.3. Dialogflow also detects possible parameters from the training phrases and does annotations automatically. The entity can be a system entity that is accessible to all agents such as date, time and location or a custom entity that is created specifically for the agent.

The parameter consists of the parameter name, entity, a value and two selections — *Required* and *Is list*. The *name* is an identifier for the parameter that also forms the value which usually is of the form *$parameter-name*. The value denotes a parameter reference that contains the extracted parameter value at run-time according to the format of the parameter reference. Using the default format, *$parameter-name*, the parameter is extracted as configured in the corresponding entity. Other options include *$parameter-name.original* which extracts the parameter as a raw user input and *$parameter-name.sub-entity-name* which extracts the sub-entity of an entity. The entity field identifies the entity type, denoted with @ with the sys-prefix indicating a system entity. The *Is list* option can be selected if the parameters are returned as a list, such as a grocery list. In this case the end-

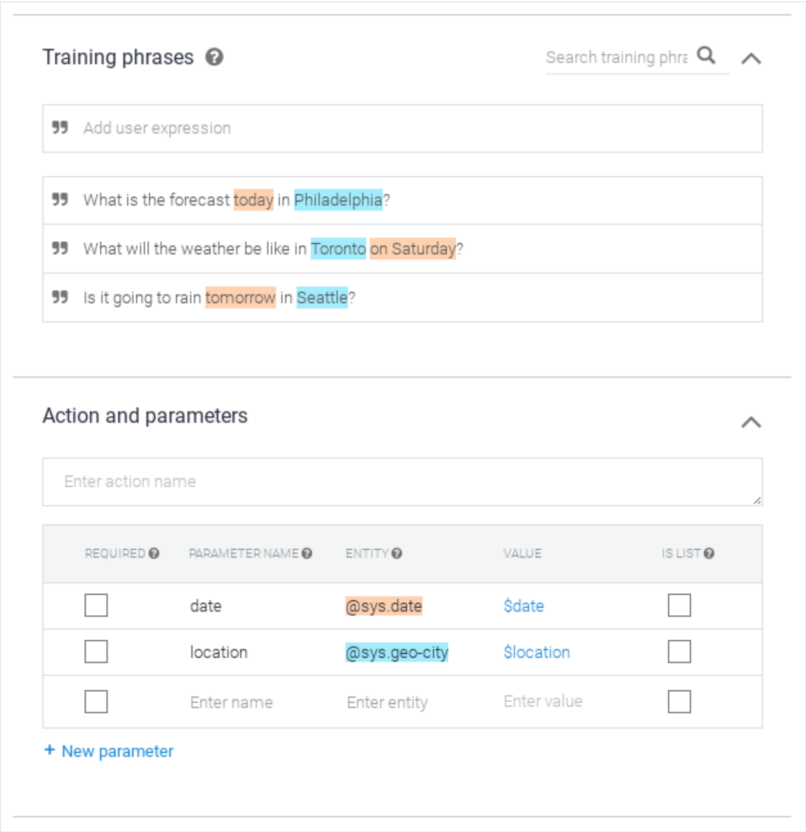


Figure 3.3: Example of parameters and annotated training phrases [61]

user expression can contain multiple items of the entity in question and one should highlight each item from the training phrases. The *Required* option marks the parameter as mandatory in order for the intent to be complete. If checked a *Prompts* field appears in the parameter row where one can define a prompt and its variations the agent asks if the parameter is not provided. This leads to a process called slot filling.

Slot filling is a process to collect required information from the user. It commences when an intent is matched and there are one or more parameters that are flagged as required. The agent continues to ask about the parameters with the defined prompts until it has collected data of all the required parameters. Fulfillment webhook requests are not sent during slot filling unless a webhook for slot filling is explicitly enabled. Slot filling can be used for a variety of purposes in order to collect a set of data, for instance to book a table to a restaurant - the extracted parameters being such as date, time, size of the party and so on. Additionally, each parameter can be set up with

a default value that is provided if the end-user expression does not contain one.

3.2.5 Response

The responses define how the intent responds to the user. The response handler of an intent can return simple static responses or dynamic responses based on the extracted parameters. The default response options are text response or custom payload. Additionally, if multiple response variants are configured the agent will select one at random creating variety in the behaviour of the agent. The text response can contain parameter values extracted from the input. However, if a parameter can resolve to an empty value a response variant without the parameter should be provided. With a custom payload response the agent can respond with a non-standard payload in JSON in a form that is defined by the platform the payload is returned to, such as Facebook or Slack. More advanced responses can be given through fulfillment or an integration, such as Google Assistant integration.

3.2.6 Entities

Entity is a categorization of the information that is extracted from the end-user expression. It can also be seen as a set of keywords or phrases that the intent is trying to find from the request. The instance of an entity is called an entity type. Entities or entity types are used as parameters for intents as seen previously. Predefined system entities can be used to match common data, such as colours, dates, times and emails, and even extended to include new values of the entity type, for instance adding combination colours, such as blue-green, for the entity *@sys.colour*.

Custom entities can be created to match data that is specific to the intents of the agent. Adding a new entity creates an entity type that can have multiple entity entries. Entity type dictates the type of information it is used to extract from the user input and entity entries contain the keywords that are used for matching. Each entry can contain a set of equivalent words or phrases called synonyms. For instance an entity type called *@animals* could have entity entries called cat and dog which in turn could contain words dog, puppy, pup and hound, and cat, kitty and kitten.

There are some options that affect the behaviour entities in data extraction and matching, however, these cannot be changed for the system entities. There are four types of entities: map, list, composite and regexp entities which each offer different ways of extracting information. The exact value that is resolved depends on how the parameter reference is configured

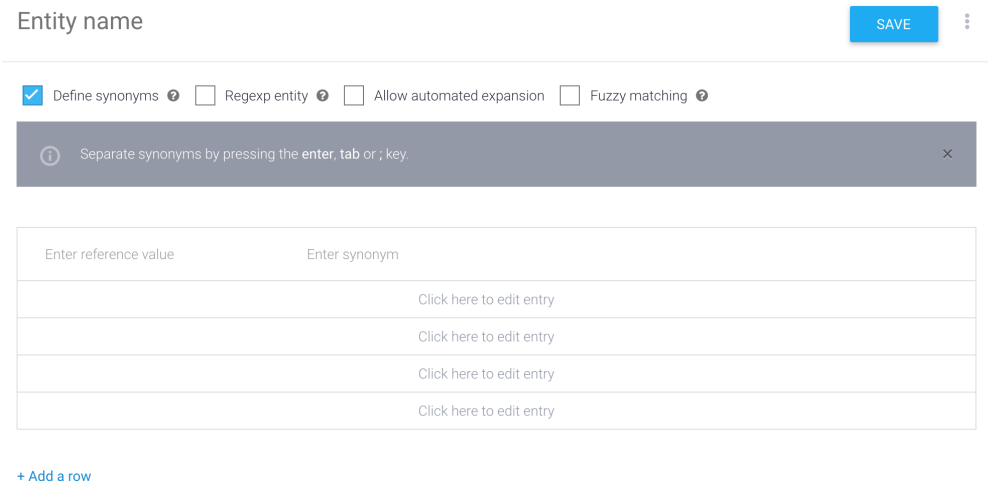


Figure 3.4: Empty entity in Dialogflow console [61]

in the intent. Entity types can be created with Dialogflow console as shown in figure 5.4 or by defining them using the API.

The default entity type is called a map entity. With a map entity each entry has a reference value and a set of synonyms that are mapped to the synonym when matched. Thus, when any word or phrase in the synonym is matched the reference value is returned. When creating the entry through the console the reference value will be automatically added to the synonyms, although, it can be omitted with the result that the reference value is not matched from the user input - this could be desirable in a situation where the reference value is a string that is not expected to be used by the user.

A list entity is an entity with a list of single value entries without synonyms or mappings to reference value. When the value is matched it is extracted for the parameter as it is. A composite entity is a special list entity which contains other entity types. The reference to another entity type is called an alias. An alias consists of the referenced entity and a property name given in the form *@entity-name:property-name*.

Regexp entities match patterns rather than specific words or phrases using regular expressions. A regular expression is a set of characters that represents a pattern for matching text and characters. For instance a four digit pincode could be declared using a regular expression \d{4}. A regexp entity corresponds to a single pattern and each entry of an entity is combined with the alternation operator "—" forming a compound regular expression. A special remark is needed when using speech recognition. When using regular

expressions one needs to take into account that the speech recognizer might parse the user input in different ways, each needing a separate handling in order to match. For instance, utterance "123" might be parsed as "123", "1 2 3" or "one two three" and thus a compound regular expression of the form \d{3} — \d \d \d — (zero—one—two—..) (zero—one—..) (zero—one—..) would take the different forms into consideration.

Session entities can be used to extend the custom entities during the conversation. A session entity exists only during the session it was created. For instance, the @animals entity, containing dog and cat, could be expanded to include bears and birds as well. Session entities can be created with fulfillment.

There are two other options to alternate the behaviour of a custom entity, fuzzy matching and automated expansion. By default, an entity is matched if one of its entries is matched exactly. With fuzzy matching enabled the entity entries are matched approximately, disregarding the ordering of the words and trying to take misspelling or missing some of the words defined in an entry into account. Fuzzy matching can not be used with regexp entities, since they are mutually exclusive features. Automated expansion can be used to automatically expand the entity to recognize values that have not been declared explicitly. The agent then recognizes values that are similar to the provided entries. However, enabling automated expansion does not guarantee the entity extraction and it is important to provide plenty of entity value examples to help the agent to expand the entity type.

3.2.7 Fulfillment

Dialogflow agent uses by default static responses for the matched intents. With fulfillment enabled for an intent, the Dialogflow agent calls the defined webhook service when the intent is matched. The workflow for a Dialogflow agent using Google Assistant integration and fulfillment is illustrated in figure 3.5. Google Assistant sends a conversation request in JSON to the Dialogflow agent containing the user request. Dialogflow agent matches the query to an intent that has fulfillment enabled and triggers a webhook request sending a Dialogflow JSON request to the webhook service. The webhook service handles the request and sends a JSON response back to Dialogflow which in turns sends a JSON response back to the assistant which finally responds to the user.

Dialogflow console has a built-in inline editor for creating fulfillment code and deploying it to the Cloud Functions. However, the inline editor only supports Dialogflow Node.js library. If an intent uses slot filling, by default it does not trigger a webhook request until all required data is collected.

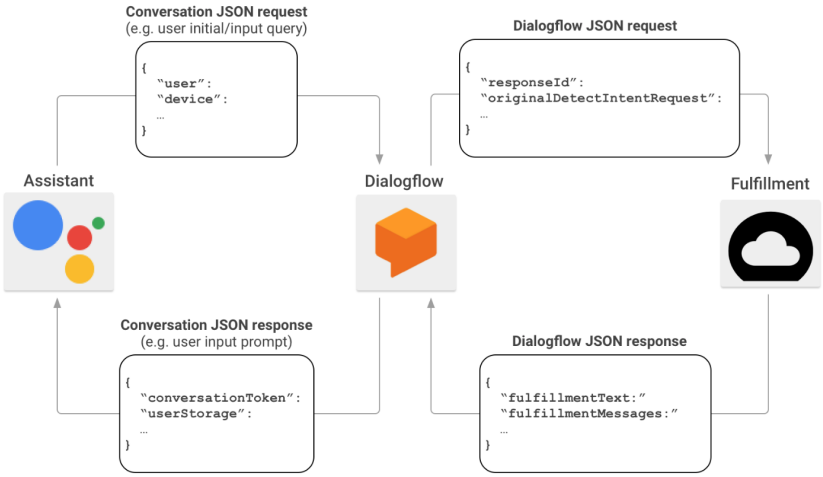


Figure 2. Actions on Google invoking your fulfillment through Dialogflow.

Figure 3.5: Assistant workflow with Dialogflow [61]

Chapter 4

Story

When the approach to an implementation is to create something yet unspecified for a new format, such as interactive drama for smart speakers, the topic can sometimes be a hard choice. There are several factors affecting the choice, like the target audience, story format, attributes of the medium, available resources, and personal preferences. Also, it is worth considering which segments of target audience are or could be open and adaptable to the kind of technology in question. In the beginning our working committee was leaned towards content for children and our executive team began to come up with ideas for different ways the children could interact with the smart speaker, such as blowing towards the speaker and clapping — interactions which are presumably yet out of scope of the current smart speaker tools. However, the theme of the topic shifted towards radio drama with possible horror and thriller elements with the actual topic emerging soon after from our own interests bearing also potential for an interactive non-linear mystery story — the infamous and mystery shrouded Tattarisuo case.

Interactive radio drama has potential on smart speaker platforms, as indicated by the BBC:s analysis of the Inspection Chamber — the users like to talk to the device and want to have personalised and impactful story experiences [62]. However, the speech synthesized, sometimes awkwardly scripted interactive stories available did not reflect this potential in our opinion. These kinds of deficiencies affect the listening or playing experience negatively, sometimes even driving away the user. Thus, our aspiration was to create a story driven audio game with appreciation for radio drama aesthetics using real voice actors and proper sound design, authentic to the setting, giving it a plausible feel and striving for an immersive experience. Hence, our writing process reflected the customs of radio drama combined with the needs of the interactive speaker format in the scope of a story driven interactive system.

4.1 Setting

When the story is based on real events and the story is composed upon the source materials the writing process can be a bit different opposed to a more traditional fictional writing — not to mention the effect of the format being completely audio based. In our case, the Tattarisuo case itself was full of mysterious circumstances and suspenseful speculations that offered plenty of material to work with. We decided to use the elements involved in the case, both real events and the rumours that spread at the time. Some of the events and rumours were quite declamatory which offered an opportunity for creating inconceivable chains of events with twists and turns that hopefully would keep the listener engaged.

Our approach was to weave the pieces of the story together guided by the format specific point of view in mind, for instance referring to the questions "What would be interesting to experience as a player" or "What choices would be effective or potentially hard to make". As a groundwork we explored the documents in the National Archives regarding the case. The documents consisted for instance of police reports, public letters, pictures of the bodies and even a strand of hair. The aim was to gather a good picture of the setting at the time and weave it with intriguing details in order to create a corresponding and a convincing setting in our story. Authenticity can play a major role in the immersive feel of a story.

4.1.1 Tattarisuo 1930

The Tattarisuo case started in August 1930 when a hand was found in the pond in Tattarisuo, Helsinki. Since, the times were busy and there was no corpse to be found, nor did anyone seem to miss a hand, the investigation was set aside. A year later, in September 1931, a loose thumb was found at the pond. This time, a thorough search of the pond revealed the shocking truth — eight hands, nine feet, twelve fingers, handful of hair and a head. This sparked a media sensation. Due to the lack of evidence and sparse statements from the police, the press was printing news almost on a daily basis, presenting their own theories about the case together with the public. This resulted in a number of rumours and false accusations, that in the end destroyed reputations and even took lives [63, 64].

The possible culprits were tried to be found anywhere from Romany people to an international satanic league of high class people in bowler hats. As of the motivation, there were speculations of a deranged mass murderer, anatomy students cutting dead bodies, secret rituals and basically anything

the public or press could come up with. It turned out that the body parts were cut from the bodies in the pauper's graves in Malmi, which turned the public eye to the janitor of the Morgue, Johan Emil Saarenheimo, who was apprehended and regarded as guilty in the public opinion. This tainted the family name in such a manner, that Johan's wife was hospitalized in Lapinlahti mental institute for the rest of her life, his son killed himself and relatives changed their names and even in the present day do not want to comment on the case [63–65].

There was also political tension and agendas present but the truth turned out to be disappointing or even shameful for some. After a year long investigation it turned out that the perpetrators belonged to the lower class and that the mutilation of the bodies was part of a black magic ritual. The goal of the ritual was to provide the pond with sigils. These sigils would then have the following effects. Firstly, they would help to claim the riches that were said to be hidden in the pond. Secondly, the radical nationalist Lapua movement was stirring unease, so the sigils would hinder their actions. And finally, the seals would help the universe to align and correct the wrongdoings of the suffering. The case remained as a cult classic and has intrigued many. So, it was a fruitful setting for a drama script.

4.2 Writing interactive drama for smart

When creating drama for a specific medium it is important to consider how the format affects the script. In the case of interactive drama for smart speakers, the story needs to be presented purely by audio like radio dramas but the listener is not just a listener but rather a proactive participant affecting the course of the story. If these factors are not taken into consideration already when writing the script, the work will accumulate into later development phases. And the end result might not turn out to be as good as it could have been otherwise.

4.2.1 Narrator and conveying the story

As with any story, one of the fundamental choices when writing an interactive drama is the means of narration. In literature the first person and the third person narratives are among the most common. These can be divided further into different methods of utilising the point of view, for instance the first person narrative can be presented as an interior monologue or with a document such as a diary. However, literature relies on written text and imaginary whereas radio plays use spoken text. Radio dramas thrive on nar-

rators, unlike for instance film and theatre, where care has to be taken to avoid the storyteller from coming between the story and the listener [66].

Since there is no visual cue with a radio drama or a smart speaker it is up to the narrator and the soundscape to convey the needed information to the listener. The narrator for an interactive drama can be anything from an omniscient figure, that watches and describes characters and the events, even throwing witty remarks for the listener, to a single speaking character, in the style of a radio monologue. The narrator can be an external entity or a character or several characters of the story. The latter brings an option to incorporate the narration into the dialogue — even completely if one so wishes.

Interactiveness brings some additional challenges into the play, such as how are the player choices presented and how much freedom is the player given, or in other words, what is the balance between the player interaction and the story? The player choices should arguably fit into the narration and thus, can even affect the employed narrator of the story. There are some different approaches to this. One option is to use an external narrator, that does not in itself appear in the story, that can ask a question or read aloud player options. This is especially fitting if the narrator is already used elsewhere, for instance in cuing in settings or telling relevant matters to the player. However, there is a danger of breaking the immersion of the story and clunkiness of too gamified player options - unless the goal is to achieve that kind of text adventure feel.

Another option is to embed the choices and other useful descriptive information in the dialogue itself. The advantage of embedding the narration into the dialogue is that the story flows uninterrupted regarding the narration, since the player is conversing with the characters rather than an unknown entity. However, this raises a question of how to fit the options or other information to the dialogue so that the character lines remain natural? The choices would have to be embedded in a way that the player could imagine that the character might ask such a question or make such a remark. There is a risk of awkwardness if the options are too clunky, or if they are presented in a redundant or gamefied manner.

We implemented the narration through the dialogue of the characters mixed with an external narrator. The choices and relevant information about the setting, such as dates, are embedded in the dialogue whenever it suited the dialog and otherwise the external narrator is employed. Most of the player choices are presented by the characters as part of their natural dialogue, such as Vanamo's question in figure 4.1. On the other hand, choices that reflected more of an inner decision of the main character and did not feel natural or practical to be voiced through the characters were narrated by an external

(kirjoituskoneen näppäilyä)

Narrator: Are you going to mention Saarenheimo's name in the express edition?

1. yes Pomo +1

2. no

valintojen kertaus jos ei inputtia: 215K

jos ei olla käyty vaimolla ja kahvilassa 219A, jos ollaan käyty vaimolla ja kahvilassa 219B
(kirjoituskoneen näppäilyä, bling-ääni ja paperi pois koneesta. MUSAA ja siirtymä toimituksen aulaan)

Jos ei olla vielä käyty vaimon ja silminnäkijän luona: **Vanamo:** I did some inquiries of my own. The Saarenheimos live at Hämeentie 17 A,. Maybe we could have a talk with Mrs Saarenheimo. And the address of the café whose owner was monitoring Saarenheimo's activities at the morgue is Aleksis Kiven katu 22. Who are we going to see first, the wife or the café owner?

219K **Vanamo:** Who first – the wife or the café owner?

Valinta

1. **Vaimo**
2. **Kahvilan omistaja**

Figure 4.1: A sample from the script with narrator and a dialogue choice

narrator as in figure 4.1. The same principle was followed with re-prompts and fallbacks as can be seen from the previous figure.

Since the smart speaker dramas are dealing with audio, sound effects can and should be utilised to enhance the storytelling, for instance to provide information about the environment and occurring actions. There are different aural elements that can create narrative meaning, such as music, noise, fading, cutting, mixing, the stereophonic positioning of the signals, electroacoustic manipulation, the actuality of the sound and silence [13]. For instance sounds of a forest such as birds chirping or leaves rustling can already tell the listener or the player enough information about the environment, while the the volume and other characteristics of the different signals can indicate the nature of the actions in regard of the environment, thus nullifying the need of telling explicitly the setting and giving space for the listener's imagination. As an example, the rustling of the leaves might intensify in volume and in other traits when something approaches while the direction could be suggested by the positioning of the signal. Music can be used to connote

a certain atmosphere or used to denote changes in settings alongside fading and cutting.

Additionally, sound effects can be deployed in liaison with the script emphasizing an action or a setting that is addressed in the dialogue — in a sense assuring the player that the action that was just mentioned, like opening a box or closing a door, is indeed happening. This also works vice versa. A sound effect might be unrecognizable or disconnected on its own, but together with the dialogue it highlights the effect of the action, conveying the listener the needed information.

4.2.2 Characters

Radio plays and other audio format dramas should strive for having few and distinct characters, introducing only the relevant minimum of side characters, since with no visual cue the listener will get confused with too many characters that are in worst case indistinguishable by voice only [66, 67]. As a guideline if there are more than three people on a scene the listener will be having hard time keeping track of the characters — this is of course dependent on the casting and the actual performance. As such, with interactive drama for smart speakers, characters should be given particular attention.

We employed many real characters from the time of the events, such as specialists and culprits, and used them and their statements in the story. After filling out the missing roles with fictional characters we devised each character a description in order to flesh out the personality of the characters and eventually to provide the actors the background for the roles. Additionally, the characters were processed and each non-essential character was cut out, and a few characters were merged together. For instance, the ring of the actual culprits consisted of several people but without any prominent function for our story they were cut to only two people and one additional fictional person - the story was unaffected, since they bore identical roles in terms of the narrative. As an example of the character fusion, the bootlegger who found the hand in the pond and the barkeeper were initially two separate characters, but they were put together, and their distinct features further developed the character of a bootlegging barkeep in times of prohibition. The fusions help in reducing the amount of characters and thus number of castings with an additional effect of increasing the significance of the outcoming character — in a way the character now bears the function of two characters.

In many stories a strong, likeable main character serves as an attraction. However, the format offers an interesting possibility that we seized — a tabula rasa main character. Leaving the main character blank or with minimal

information can be seen as giving the frame of being the actual main character, or at least in the boots of one. This way the voice of the player becomes the voice of the character and there is no need to establish age, sex or other identifiable traits for the character — the player creates them when experiencing the story. This might require emphatising on behalf of the player, since everyone might not feel identifiable to this kind of protagonist. Also, if such character is employed the script should be processed such that there are no unintended insinuations about the main character, such as personal pronouns that would indicate the gender — which is easy in Finnish due to the gender-neutral "hän".

The disadvantage or difficulty with a voiceless neutral main character is that the character can not be alone in a scene. To tackle this problem we utilised the principle of a loyal sidekick in the spirit of Doctor Watson used in Sherlock Holmes stories. We created a helpful side character, Miss Vanamo, to carry the investigation and the story, to voice thoughts that are interesting for the player and most importantly ask questions from the player as the means of presenting the player options. This gave us the possibility to test out a blank protagonist owning only a surname and a title.

4.2.3 Linearity

Perhaps the most important element to interactive drama is the freedom of choice — player interactions and their outcome — or the illusion of it. Depending on the employed approach there are different matters of different significance that might not be applicable to all scenarios, such as story generation and character autonomy. However, whether the story is approached through character based decisions or through the plot or even fully emergent from the player actions, one of the most prominent issues is the trade-off between interactivity and the story, or in other words how much control is the player given over the events. In one end of the spectrum the player can be given full control over the story, and in the other player can only make predefined decisions that do not actually have an influence on the story.

Guiding the player through linear predefined drama sequences without giving the player practically any power over the events can be seen as a shortage — what is the point of interactivity in the first place? - but it is the easiest scenario to control. On the other hand, giving the player full control would mean that the advancement system is practically completely non-predetermined and any input would generate the story further with infinite options - which requires a completely different, AI based, methodical approach compared to a linear system - and probably notably more resources. For a plot based drama a sensible approach is to define predetermined paths

the player can explore - and hopefully find interesting. The player wants to make a difference or at least to feel like they have an impact on the story. That does not mean that the story structure has to represent an ever expanding tree structure. While the structure essentially can be defined this manner, by creating new independent branches for each choice, the result will probably contain a lot of redundancy — unless each node represents a completely unique scene — and can be constituted in a more efficient manner.

The Dead Are Speaking is a story-driven game, meaning that all the events are predetermined and scripted, but the progression and occurring events are based on the player choices. As an interactive system it utilises lightweight interactive narrative and plot based approach, with some elements of character based storytelling. Most choices affect the events of the story and character behaviour is inside the events usually unaffected by the player choices. There are two exceptions, however. The boss and the assistant of the main character have a hidden scoring system. In a set of situations, these characters react directly to the actions of the player and the accumulated points can unlock different endings.

The story-flow is quite linear. Even though there are several places where the story branches to different situations and locations, the branches eventually join together as illustrated in figure 4.2. The only singular choice that differentiates the branches permanently is positioned at the very end of the drama. This kind of linear segments help in keeping the story coherent and the workload in check in the branching aspect. However, different events can trigger specific actions, such as setting or changing variables that can act as switches or with other logic affecting the events or even result in branching in later story nodes. For instance, some of the branches may contain information that is impossible to attain otherwise. There are also sections, where the player can experience a set of events in any given order, but not necessarily all of them.

The ending for The Dead Are Speaking depends on three factors: the final choice, the interactions with the boss, and the interactions with the player's assistant, Miss Vanamo. The player can gain points based on the choices made through interaction with these two characters. The basic idea is that choices that are to the liking of the character increment the score of the said character and if the score is above a certain threshold in the end the story the player unlocks an ending where the character is favorable towards the player. With this setup there are all in all eight different endings consisting of two basic sequences based on the choice and their further variations depending on whether the boss likes or dislikes the player, and if Vanamo likes or dislikes the player. Our aim was to create choices that maintain the player's illusion of choice by producing sufficiently significant branches to keep the player

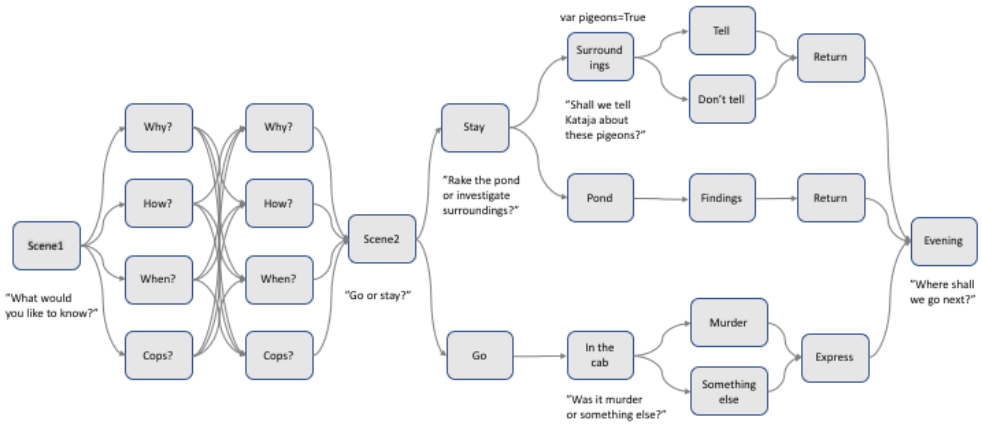


Figure 4.2: A flowchart of part of the script

intrigued and engaged but keeping the impact of the actions small enough so that the amount of content does not grow exponentially.

4.3 Script

A good script arguably makes things easier. The script serves as a reference for many other tasks, such as acting, sound design and coding. It is thus a reasonable idea to keep it tidy and professional at all stages of development [66]. We wanted our script to contain all the relevant information in regard to the end product so that in addition to the dialogue, we could see with one glance what elements are required in terms of sound design and technical point of view.

4.3.1 Format processing

The story was divided into four acts, each marking a day in the story. Due to the technical restrictions, the audio segments between choices or user input had a maximum duration of 120 seconds. This changed later in an update to 240 seconds. The chunks of story between the required user input consisted often of the dialogue leading to a natural choice, such as a question, but when there was no natural choice to be made too lengthy sections were disrupted

with the narrator requiring user input by stating "Say ready to continue". This had two effects. For one it helped us to divide the script into applicable pieces to fit the audio within the time limits, and secondly to help keeping the player engaged by avoiding too lengthy sections without user input — to prevent the player from taking a role of a listener instead of an active participant, and to help the player to follow the story. Additional helping factor with the length of the segments was trimming off surplus content. The script was processed with the idea that all the words and sentences that did not contribute meaningful information for the story or the atmosphere were removed. This also helps in the clarity of the events assisting the player in following the story.

The choices were embedded mostly in character dialogues so that they would appear natural but at the same time so that the options available would be obvious for the player. It is important to avoid frustrating players by helping the player to steer clear of actions that are unavailable for the current intent. Therefore, we mostly followed a paradigm of potential player interest: the options available are suggestive and offer interesting objectives in line of the plot, in order that the player would want to respond in accordance to the options. As a rule, the number of choices does not exceed four, in order to keep the available choices memorable and reasonable. If there was a need for more choices, the choices were presented in sets of four and additional choices were presented only in follow-up questions after room was made by selecting one of the options. Presenting too many options could break the natural flow of the dialogue confusing the player - it is also unusual to present a plethora of options in a natural conversation. The first choices in the drama act as an awakening tutorial and help the player to get acquainted to the format - although the second choice in game already accumulates points that affect the ending.

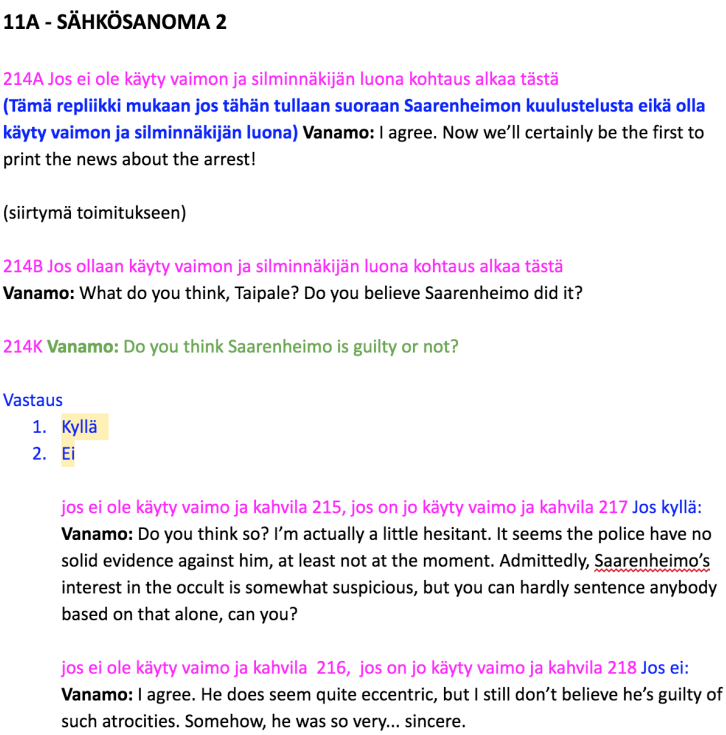
The format allows interactions that differ from basic choice situations. For instance by utilising game state variables and saved responses one can create more complex events, such as mini-games or other interactions requiring additional logic. Open questions work well with speech synthesis, since the user input can be used in the response. We used some open ended questions, such as asking for the name of the player or the opinion of the player, that allowed the user to answer with any input, but could not use the input in our response - it would have been possible with speech synthesis but that would not have worked. The open questions might work well if the user answers in conjunction to the nature of the dialogue, but if the user is unsure or decides to answer with something more far fetched the response in the story might seem unfitting. For example, if the player is asked "Who do you think did it?" and the player answers "Well it was you", it might be weird for the

character to respond with "Hmm, maybe". Therefore, it could be a good idea to be prepared for unexpected player inputs. Also, names and other words in unsupported languages are hard for the speaker and can bring hilarious results.

4.3.2 Notation

The notation of the script should be clear and functional - this is what we aimed for. From the technical point of view, we wanted to include as many elements in the script as possible to maintain clarity and integrity in all development phases. These elements included game logic and audio file structure, possible sound effects, the interactive choices and re-prompts as illustrated in the figure 4.3 together with their colour coding. Each scene consists of sequences that may contain multiple interactions. The scenes are marked with a number and sequences are differentiated with a letter, for instance *11A* in the aforementioned figure indicates scene 11 and A the first sequence of that scene. Normal black text is used for dialogues and actions that relate to sound design. Green colour is reserved for character reprompts, if the speaker does not understand the user input, if the user input does not match any of the possible expected inputs or if the speaker does not receive a user input. Pink colour is used for the audio files and logic descriptions, such as which events lead to selecting this route or audio file. Blue colour is used for choices, narrator and some additional information.

In order to keep the links between the script, code and audio editing clear the script was notated with the names of the audio files which were structured according to their function and localization. Each day reserves a corresponding audio file name range with the first day starting from 100, second day from 200 and so on. The number is then incremented for each consequent audio file. Additionally, a character switch at the end of the number was used to indicate a special function, a variation or different circumstance when needed. For instance, in the figure 4.3 the letter 'K' indicates a re-prompt and 'A' and 'B' indicate with certain logic whether certain events have already occurred in the story. A clear and structured naming convention is extremely helpful in regard of the technical implementation, especially since almost every audio file correspond to a single intent.



11A - SÄHKÖSANOMA 2

214A Jos ei ole käyty vaimon ja silminnäkijän luona kohtaus alkaa tästä
(Tämä repliikki mukaan jos tähän tullaan suoraan Saarenheimon kuulustelusta eikä olla käyty vaimon ja silminnäkijän luona) Vanamo: I agree. Now we'll certainly be the first to print the news about the arrest!

(siirtymä toimitukseen)

214B Jos ollaan käyty vaimon ja silminnäkijän luona kohtaus alkaa tästä
Vanamo: What do you think, Taipale? Do you believe Saarenheimo did it?

214K **Vanamo:** Do you think Saarenheimo is guilty or not?

Vastaus
1. Kyllä
2. Ei

jos ei ole käyty vaimo ja kahvila 215, jos on jo käyty vaimo ja kahvila 217 Jos kyllä:
Vanamo: Do you think so? I'm actually a little hesitant. It seems the police have no solid evidence against him, at least not at the moment. Admittedly, Saarenheimo's interest in the occult is somewhat suspicious, but you can hardly sentence anybody based on that alone, can you?

jos ei ole käyty vaimo ja kahvila 216, jos on jo käyty vaimo ja kahvila 218 Jos ei:
Vanamo: I agree. He does seem quite eccentric, but I still don't believe he's guilty of such atrocities. Somehow, he was so very... sincere.

Figure 4.3: A sample from the script. The green font indicates a re-prompt

Chapter 5

Implementation

The minimum requirement for the technical implementation was to build a conversational progression of the drama according to the script with the player choices operable on the smart speaker. This can be divided into following low level tasks: invoking the application and entering the execution loop, which starts by presenting the audio content, detecting and parsing the user input, performing logic based on the game state, responding the user with the new scene, based on the previous choices and starting the loop from the beginning. Since this was an exploration to creating smart speaker content we wanted as a principle to use tools and technologies that were created or intended for developing Actions. The reason for this was to ensure that the used technologies would be capable of the creation of the Action and to dive in the possibilities of the development framework.

5.1 Architecture

At the time of the implementation there were two routes for creating a conversational Action: using Dialogflow or using Actions SDK. Dialogflow offered NLU features and intuitive tools for building the conversational functionalities using intents and entities, while Actions SDK would have required implementation of our own NLU solution and a JSON file based representation of the Action. Since we had no necessity to create our own solution for matching the intents, and Dialogflow offered tools that allowed a certain degree of freedom for the user input, we decided to build the core of the application with Dialogflow. The agent was created with Dialogflow console rather than with API because the console felt intuitive and easy to modify whenever necessary. Furthermore, this allowed other members of the team to add entries to entities and training phrases when needed.

While the basic agent offers intent matching and context-routing to handle simple logic, it needs custom fulfillment in order to access external resources and for more complex mechanics. Therefore, we needed a webhook service to handle the playing of the audio files and to perform functions that needed any more sophisticated logic, such as if-clauses and reading and writing session state data - even though they could be simulated with contexts to an extent but which would as a byproduct increase the number of intents notably. Additionally, the audio files itself needed to be hosted somewhere, wherefrom the webhook would be able to fetch them during the conversation. The hosting of audio files and accessing them with their URL allows an independent modification process for the audio. The audio files can be changed and updated without need to modify the fulfillment code or any other parts of the Action.

The interaction between the different components of our implementation is illustrated in the picture 5.1. The process happens as follows: user gives a command to the Google Assistant which parses the input and passes the user utterance as a JSON request to Dialogflow. Dialogflow matches this utterance with one of its intents and triggers a webhook request to Cloud Functions for Firebase. The webhook handler then performs the required operations based on the game state and the received parameters. The webhook function generates a response, that contains a reference to the new audiofile in Firebase, and returns the response payload to Dialogflow. After Dialogflow has processed the response, it returns the response to the assistant, which in turn plays the content to the user.

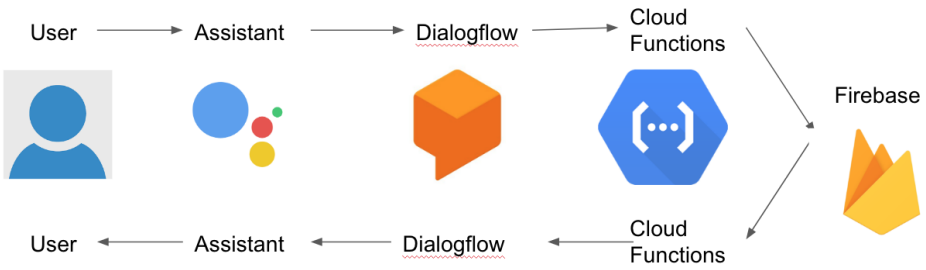


Figure 5.1: The workflow between different components of the implementation

5.1.1 The Actions project

To make the Dialogflow agent accessible for Google Assistant, it needs to be configured with an Actions on Google integration. The integration manages the communication between Dialogflow and Google Assistant through a conversation protocol without the need of parsing requests and generating responses explicitly. The Actions project also contains the directory information and other resources that are needed for the Assistant, such as invocation details, surface capabilities and contact information. The Dead Are Speaking is set available in all countries and regions, although the implementation language is English. For the surface capabilities, the audio is set to required, but screen output is not, making the Action available on smart speakers, mobile devices and smart displays. The invocation phrase is The Dead Are Speaking and the Action uses only explicit invocation. This means that all the sessions go through the default welcoming intent, and the control for different events and content, for instance option to continue a saved game, is handled by the webhook.

The deployment options and required information, such as descriptions, location targeting and privacy policy, are also managed through the Actions console. Each change in the Dialogflow agent or the Actions project itself needs a new publication for the changes to take effect in any version but the test version of the Action. However, fulfillment webhook runs independently of the Action and thus, is modifiable without the need of a new publication. While the alpha-version of the Action could be released early on for a small group of testers, the beta-version required passing the Google review. The review verifies that the Action follows the policies Google has set for Actions, such as respecting content restrictions or demonstrating meaningful functionality. Our first couple of attempts to publish the beta version was disapproved. The reason was a lacking privacy policy. At first, we tried to publish the Action using Yle's general privacy policy. However, Google required an Action specific privacy policy in the form of a public document, which includes a disclosure about how the Action collects, uses and shares any user data. Our implementation uses and saves data only for game-play purposes, but the statement was required nevertheless.

5.1.2 Dialogflow Agent

The Dialogflow agent was built around the intents that lay out the structure of the application in the frame of the story. Each intent represents a part of a scene as a consequence to the player's actions - save for some helper intents and a couple of intents which are used as logical routers. However, a single

intent can represent the situation to which the choice leads containing all the different possible responses to each of the choices. Therefore, an intent can present two different situations in different locations but which are based on the preceding choice.

The intents were structured according to the chapter structure of the script. The intent name is of the form X_Y{descriptive name}, demonstrated in figure 5.2, where X denotes the number of the chapter and Y the number of the sequence in the chapter in ascending order, although some of the sequences run in parallel branches of the story. Additionally, as some chapters run on parallel branches, the intents are differentiated as the chapter names, with a letter after the chapter number.

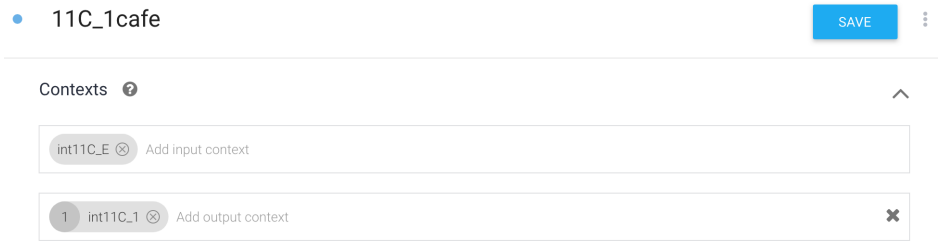
Figure 5.2: Intents for the chapter 1, taken from Dialogflow console

The same naming convention is used with contexts, entities and events. In other words, intent X_Y triggers in most cases an output context intX_Y, the intent can be invoked with event X_Yevent and the specific training phrases correspond to the entity entX_Y. This kind of structuring keeps the relations of the components clear and organized, making the development process smoother and less prone to careless errors. Additionally, it helps in logical functions as we see later.

Contexts follow the intent structure strictly in order to ensure the logical integrity of the story. The routing done by context is used to ensure that only the intents to which the story could progress from the current point can be matched - even though the user input would otherwise match to the training phrases of another intent. To achieve this and to avoid any lingering contexts and hence, accidental matchings, it was essential to set the lifespan of each context to one and manage the contexts for each conversational turn. Additionally, this helps in preventing intents from becoming out of reach, since ideally the right context is activated at each intent - or fallback intent - and thus matched correctly. The compact ordering helps in keeping the routing clear and controlled - especially when there are more branches.

In our implementation, each intent has an input context that is either the output context of the sequence preceding the intent as in figure 5.3 a. If there are multiple sequences that can lead to the intent in question, an entrance context is used, marked with the letter switch *E*, as in figure 5.3 b. Similarly, the output contexts of an intent consists of the output context specific to the current intent, or entrance contexts to the intents that are possible outcomes of the current intent, or sometimes both. Most of the contexts were defined in Dialogflow, but there were also many that were evoked during run-time with the webhook.

(a) Context1



(b) Context2

Figure 5.3: Contexts of an intent

The form in which the users are expected to answer and how well the speaker actually understands what the user says or intends can play a major role in the user experience. Consequently, we wanted to present the options in a natural manner and that the user could respond freely within the contextual frames of the conversation; our entities reflect that ideology. We created an entity for each choice corresponding to the intent that is the result of that choice. Each entity was implemented as a map entity, illustrated in figure 5.4, since we needed to extract the reference values of each choice. Entity entries were formulated with different ways of saying the options. The entries were supplemented with variants that were missing words or contained misspellings, since the user might pronounce the phrases carelessly or the assistant might simply interpret words falsely. Entities use fuzzy matching in order to improve the matching, if for instance the user input misses words or if there is a misspelling we did not take into account.

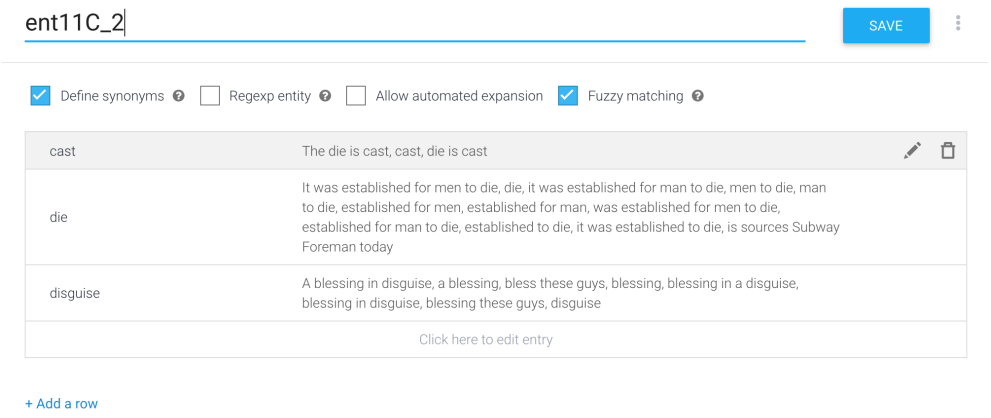


Figure 5.4: Example entity

While most of the intents used entities as their training phrases, some intents that did not need parameter extraction, used pure training phrases. In these cases there was no actual choice to be made, instead the player is expected to reply in a certain way. For instance, in the training phrases of an certain intent, depicted in the figure 5.5, the player is expected to say Rolf Rivasto in order to continue. In most cases, the user input is enforced, in other words the user input needs to match the training phrases in order for the player to continue. However, in other cases it is irrelevant what the user utterance is, since the story is progressed by a fallback to the next intent. This might create hilarious visual outputs with mobile devices, especially in the latter, since the application is not interested and passes the progression

with any input.

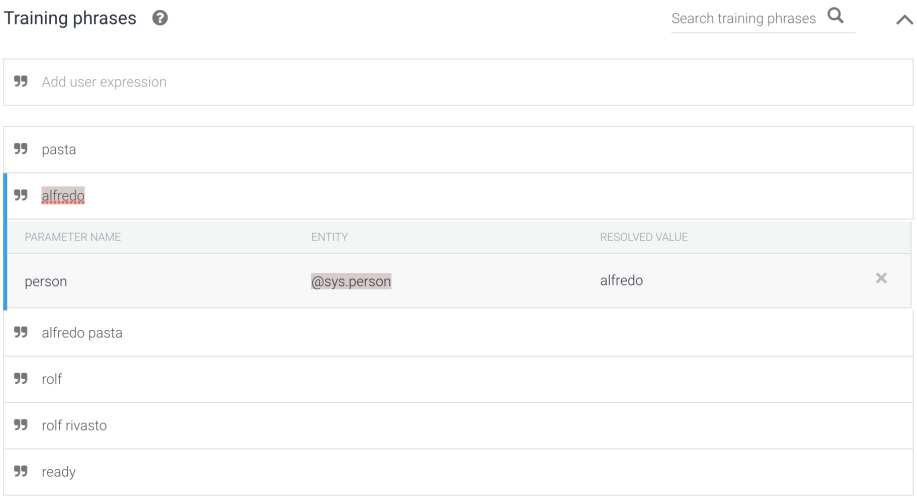


Figure 5.5: Different training phrases for an intent to train on different interpretations of *Rolf Rivasto*

5.1.3 Webhook

While the Dialogflow agent and the intents create the frame for the content, the webhook generates the content for the intents and handles the logic behind the branching of the story. The webhook service receives a request from the Dialogflow agent every time an intent is matched, since each intent needs fulfillment, and generates a functional response back to it. The communication uses JSON format for both requests and responses and is done via HTTPS requests. The webhook was implemented with Actions on Google Node.js library, since it supports all Actions on Google features.

The webhook was deployed to Cloud Functions for Firebase, because it provides a scalable computing platform, with an already established solution for handling the webhook requests. For the sake of convenience and functional simplicity, the audio files were also hosted in Firebase. Both Firebase functions and Firebase hosting of the files were managed with a Firebase CLI tool. This meant that with a simple Firebase setup and the export of the app object in the webhook file, the webhook was ready to receive requests from the Dialogflow agent and have access to the audio files, without need for additional authentications or other configurations. Additionally, Fire-

base functions provided run-time logging for the function calls and potential errors.

With *Cloud Functions for Firebase*, the webhook is typically implemented in index.js file located in the functions folder. We implemented all the webhook handlers for the intents to the index file and used other files for helper functions, such as retrieving texts for the intent handlers. The fulfillment is accessed with a global *app* instance, which encapsulates the fulfillment logic and handles the communication between Actions on Google and the fulfillment service. The *app* instance creates a conversation object that represents the conversation session. The intent handler is created with the *intent()* function of the *app* instance, illustrated with the *1_2boss* intent handler in code segment 5.1. The intent handler consists of the name of the intent it is registered to, a *conv* variable, that is used to access the conversation object, and a handler function. All the executable processes are defined inside the handler function.

```
1  app.intent('1_2boss', (conv) => {
2      conv.data.fallbackCount = 0;
3      conv.data.previous = ['1_2event','ready','int1_1'];
4      const audiourl = host + '197.ogg';
5      const txt = Texts.bubble(conv.data.previous[0]);
6      const ssml = Utils.playSimple(audiourl);
7      if (conv.user.verification === 'VERIFIED') {
8        conv.user.storage.day = 1;
9        conv.user.storage.previous = conv.data.previous;
10     }
11     conv.ask(new SimpleResponse({speech: ssml, text: txt}));
12 });
```

Code segment 5.1: A webhook intent handler

The response was usually returned as a simple response, returned with the response function of the conversation object, conv.ask(). The rich response consisted of the audio generated with SSML and the displayable text. Both the SSML and the text were created using helper functions. The different SSML functions are illustrated in code segment 5.2. *playSimple()* was used for almost all intents, since the simple functionality was suitable for most cases: it took the source URL of the audio file as a parameter and returned a SSML to play the audio file. In contrast for instance, *playMulti()* expected

a set of *media* elements, allowing simultaneous playback of multiple audio instances. The texts were listed in and fetched from a case structure within the helper function.

```
1  //Audio helpers
2  playAudio(srcurl, startPoint, endPoint){
3    return `<speak><par><media>
4    <audio src="${srcurl}" clipBegin="${startPoint}s" clipEnd
     ="${endPoint}s"/>
5    </media></par><prosody volume ="silent">a</prosody></
     speak>`
6  },
7  playSimple(srcurl){
8    return `<speak><par>
9    <media><audio src="${srcurl}" /></media>
10   </par><prosody volume ="silent">a</prosody></speak>`
11 },
12 speak(content){
13   return `<speak>${content}</speak>`
14 },
15 playMulti(content){
16   return `<speak>
17   <par>${content}</par><prosody volume ="silent">a</prosody
     >
18   </speak>`
19 },
```

Code segment 5.2: SSML helpers for audio playback

The Dialogflow app instance creates a params object, which represents the parameters of the request. The extracted values can be accessed through the properties of the params-variable and utilised in the handler function, using the names of the entities as the properties. We used a destructured form of the parameters, applied with the curly brackets. The destructuring takes the properties out from the parameter object and allows access to them as local variables. The processing of the parameters is illustrated in code segment 5.3. In most cases, two entity properties were used: response, which consisted of numbers and ordinals, and the intent specific entity, such as ent2_3, containing the actual answer options. The reason was that the user could answer either way, using the presented options or numbers, for instance if the speaker was having troubles recognizing the given options. However,

only the used parameter is extracted and contains a value. The extracted value is then checked in the handler function, and the audio and game state handled accordingly. If the extracted entity properties did not contain any of the expected values, for instance if the intent was matched incorrectly without the parameters, the fallback event was triggered. The *followup()* function of the conversation object allowed invocation of intents based on their events. This was used for triggering fallbacks, certain story intents based on the game events, and to resume the story from various points, such as fallbacks, previously saved games and other operative intents.

```
1  app.intent('2_3handuja', (conv, {response, ent2_3}) => {
2    var answ = 'two';
3    var audiourl = '';
4    if (response === 'one' || ent2_3 === 'bootlegger') {
5      answ = 'one';
6      audiourl = host + '121.ogg';
7      conv.data.sreveal = true;
8    } else if (response === 'two' || ent2_3 === 'reveal') {
9      audiourl = host + '122.ogg';
10   } else {
11     return conv.followup('fallevent', {
12       response: 'fallback'
13     });
14   }
15   conv.data.fallbackCount = 0;
16   conv.data.previous = ['2_3event',answ,'int2_2'];
17   if (conv.user.verification === 'VERIFIED') {
18     conv.user.storage.previous = conv.data.previous;
19     conv.user.storage.sreveal = conv.data.sreveal;
20   }
21   const txt = Texts.bubble(conv.data.previous[0]);
22   const ssml = Utils.playSimple(audiourl);
23   conv.ask(new SimpleResponse({speech: ssml, text: txt}));
24 });
```

Code segment 5.3: A webhook intent handler with parameter processing

The game state was handled with a set of variables placed within the *conversationToken*, using the *data* property of the conversation object. The current scene was configured with a special array, *previous*, that contained all the required information to evoke or resume a specific intent. The array

consisted of the name of the event to invoke the intent, the extracted user's selection, and the input context for the intent.

The default fallback is used as fallback for most of the intents. The purpose of a fallback is to re-prompt the user for input, and in most our cases this meant playing the re-prompt audio and setting the appropriate contexts, which are usually the same as the current intent has. We felt that for such a simple function, it was more sensible to let the webhook handle the fallback call with a singular collective fallback handler, rather than to create almost 200 distinct fallback intents and repetitive webhook handlers for them. Fallbacks that required more logical handling, for instance for managing different contexts, were implemented with separate intents and webhook handlers. This allowed customised behaviour depending on the situation that led to the fallback.

The default fallback is called, when a fallback occurs and there are no fallbacks defined with input contexts matching the active contexts. The default fallback handler uses game state variables to fetch the name of the appropriate audio file and the reprompt text, and to set the active contexts. The fallback handler will play the re-prompt for the first two fallbacks and initiate the next sequence on third. The reason behind this was to help the players to progress, if they were to become stuck in a choice. The no-input fallbacks work similarly, except the re-prompt is played once and on the second try the sequence is repeated. On the third input Google Assistant forces a conversation exit, so we implemented the fallback handler to close the conversation. See Appendix A for more information.

Even though the implementation was aimed for smart speakers, we implemented visual features for mobile devices and Google smart displays. The reason was simply that the Action was accessible with at least mobile devices, regardless of our choices - except if we did invoke a conversation ending intent in our webhook, when screen capabilities would have been detected. In addition, we did not want to exclude potential users, especially when the content could easily be composed based on already existing material. In order to deploy the Action for devices with the screen there had to be visually displayed texts for each intent, that are a phonetic subset of the SSML output. Since the space for the texts was limited and all the texts were displayed simultaneously, we devised the displayable texts as short descriptions of the scenes. Although, we would have wanted to display the options, but only after they would have appeared in the audio.

```
1 app.intent('NewGame', (conv) => {
2   var audiourl = host + '101.ogg';
```

```
3   conv.data.previous = ['newgame','new game','
     DefaultWelcomeIntent-followup'];
4   const txt = Texts.bubble('Welcome');
5   if (conv.surface.capabilities.has('actions.capability.
     WEB_BROWSER')) {
6     audiourl = host + '101P.ogg';
7   }
8   const ssml = Utils.playSimple(audiourl);
9   conv.ask(new SimpleResponse({speech: ssml, text: ' '}));
10  if (conv.screen) {
11    conv.ask(new BasicCard({
12    text: txt,
13    title: 'The Dead Are Speaking',
14    image: new Image({
15      url: 'https://tattar-oudbew.web.app/LOGO.png',
16      alt: 'Tattarisuo'
17    }),
18    display: 'CROPPED'
19  }));
20 }
21 });
```

Code segment 5.4: Intent handler using visual features

Furthermore, applying support for displays allowed us to explore the visual aspects of the Google Assistant functionalities. The starting and the ending intent utilised rich responses, displaying a *basic card* consisting of a title, an image and a text section. The implementation of a *basic card* is illustrated in the code segment 5.4. The webhook handler checks whether the device in the conversation has a screen surface capability and returns a *basic card* for the assistant if that is the case. As an additional matter of interest, the *NewGame* intent required an additional inspection for mobile devices, which can be identified by the *WEB_BROWSER* surface capability, and an devoted audio file, because the original audio sometimes triggered the listening mode of the assistant on the mobile device it was playing on, interrupting the Action. The instructions were stripped of the phrase "Hey Google". This emerged after publication on smart speakers as well.

5.2 Implementation details

Games often include functional features that do not affect the actual gameplay directly, such as a saving system or the option to pause the game,

that expand the possibilities of the game, enhance the user experience, and grant the player more freedom on how to play the game. We wanted the functionalities of the Action to support different styles of interaction, driving the story at the user's terms. Additionally, for the sake of the story-flow we needed to implement functionality that did follow the typical intent matching pattern.

5.2.1 Features

One of the most essential helper features was the ability to save the game, and continue the story where the user left it. In addition to convenience, this was arguably a crucial feature. Our Action had a lengthy running time, and any error occurred within the Action, network, or Google Assistant itself, could have interrupted the action, forcing the user to start from the beginning. The saving was implemented with the user storage functionality. Actions use a token that is saved for a particular user and saved between the conversations. The token is contained within the *userStorage* field of the *AppResponse object*, which is the response that is sent to the assistant. The *userStorage* field can be accessed with *conv.user.storage* object using Actions on Google Node.js Client Library.

The user storage allows saving data between conversation for verified users only. The verification status of the user is based on different factors, such as user's settings and signed in status, but for instance a user that has logged in to the Google Assistant on mobile device has a verified status. However, this is not explicitly stated for the user and some of the options might be turned off, and thus, limit the experience. Once data is stored to the user storage, the storage does not expire, as long as the assistant can match the identity of the user. The user can view the stored data from the Assistant application. Even though our action did not save any identifiable information, we implemented an option to clear the user data with an intent. The *userStorage* field can be emptied by setting the value of *conv.user.storage* to an empty object, denoted by curly brackets {}.

The saving feature was implemented as an auto-save-like system - each time a conversation variable changes, the corresponding user storage variable is updated. Since, the game-state is upheld with a variable, the user storage is accessed in each non-fallback intent. Initially, we tried to implement saving of the game-state variable in the exit-intent, but for some reason the user storage did not update. When the assistant receives one of the exit commands from the users, ACTION_INTENT_CANCEL event is triggered. When the Exit-intent of our implementation was invoked, it failed to store the provided values for user storage variables, whereas a custom exit intent with a non-

exiting training phrase, and thus without the ACTION_INTENT_CANCEL event, was able to perform the user storage update and exit the conversation. However, since the assistant launches the exit-event, we were compelled to use the event.

When the welcoming intent recognizes a returning user and the user has saved data in the user storage, the load game context is enabled. If the user wishes to continue a saved game, the load game intent reads all the user storage variables to the conversation data variables that are used within the webhook. Then the lastly played intent is invoked. The invocation occurs with another helper intent, repeat, that is used for repeating sequences or intents. The repeat intent replays the current intent, or more specifically the intent that is specified with the game-state variable, allowing quick invocations from other intents. Such invocation return was utilised in some of the fallbacks and pausing system. Pause-intent is an intent whose response is filled with silence. When a user wishes to continue, phrase "Hey Google, continue" needs to be used, in order to wake the speaker to listen and receive the command to continue. This triggers a pauseRelease intent, which calls for repeat intent, returning the story to the last story-intent visited. See Appendix A for further information about the implementation of functionality intents.

We implemented a small mini-game, where the user is asked of the number of three different types of body parts found at the pond. The mini-game was implemented with managing the answer for each body part with a distinct intent, that took a number as a parameter. The received number is saved into a conversation data array and if the number corresponds to the right answer, a mini-game variable is incremented. Once all the answers are given, the player is given an option to revise the numbers. If the player chooses to revise the numbers, the variables regarding mini-game are emptied, and the first mini-game intent is invoked with a new audio that takes the revision into account. Once the user is finished, the mini-game counter is checked and boss points allocated as follows: if the player remembers at least the number of one type of body parts correctly, one boss-point is rewarded, but if all the numbers are remembered correctly, the player receives two boss-points. See Appendix A for further information.

5.2.2 General intents

There were some scenes that required continuity of the story regardless of the user input, or sometimes even the lack of it. Many of these scenes were utilising an open question to which the user could answer freely. Naturally, we could not use the answer for our logic or response, save for some potential

internal logging, but the freedom offered the user a change of pace from selecting only from available options. Situations where the story progresses even without user input are rarer, since usually the participation of the user is a wanted measure. But in cases where staying silent would be a natural option or would have consequences otherwise, it was the most suitable policy.

We did not want to employ the system entity @sys.any to let any input lead to matching the intent, since it would have affected the machine learning of the training phrases. Instead, we created two fallbacks, anyfall and anynoinput, to catch a fallback or a no-input event, and to handle the story progression. Intents that could lead to anyfall or anynoinput fallbacks were provided a corresponding output context to differentiate the matching process of the default fallbacks. Similarly to default fallback, the anyfall and anynoinput fallbacks fetch the appropriate events and contexts utilising the game-state variable, and invoke the next intent as a follow-up event.

5.2.3 Audio combinatorics

The structuring of the audio file names was utilised in sections that required at least two-multidimensional logical processing. In these situations the operations were affected by more than one factor, for instance the current and the previous choice. By an apt naming convention the name of the audio file itself bore meaning, acting as a variable. Using the information given by the audio file name we could avoid for instance nested structures, such as nested if-statements.

A good example of such a situation is a scene where the culprits are interrogated in a car, with code segment 5.5. illustrating the webhook logic behind it. There are potentially five different options for the player, but on each turn a maximum of three are presented. Each option has an individual response and the follow-up question consists of the subset of the remaining options. The response audio was separated from the follow-up questions to reduce the complexity of the resulting audio combinations. The name of follow-up audio consisted of a prefix *PA* and the presented options as numbers, such as *PA123* for the first three options. The webhook handler illustrated below uses simple if-statements and string-methods to determine the options and to construct the follow-up audio.

The first section of 24_3 car intent is used to assess the status of the interrogation. This means namely what choices have already been made, represented in the conv.data.pakys variable, and what are available choices. The Utils.pusher function returns all the previously non-selected options, in other words excluding the selection that led to the intent. The options are returned as a string of numbers and array, containing the values the intent

specific parameters would resolve to. The options are determined by the values that are not found in the given parameter. The string is used to construct the audio URL and the array is needed to determine the selection of the user, if the user had used numbers or ordinals to make the selection. The Utils.selector function is then used to extract the entity type if numbers were used. Now that the answ-variable contains the selection, it is a matter of a simple if-statement to determine the response audio and to delete the number of the selected entity from the follow-up audio URL, resulting in an appropriate follow-up audio URL. Both audios are tied together consecutively with SSML and returned as a simple response with a corresponding text bubble.

```
1  app.intent('24_3car', (conv,{ent24,response}) => {
2    var audiourl = host;
3    //Contains the previously selected choices
4    var pos = conv.data.pakys;
5    //Retrieve the possible resources for the
6    //question audio name and the available player choices
7    const pushed = Utils.pusher(pos);
8    var audiourl2 = 'PA' + pushed[0];
9    var choices = pushed[1];
10   // Select the player choice according to the
11   //response (if used) or use the ent24 entity
12   var answ = Utils.selector(choices,response);
13   if (answ === 'notresponse') {
14     answ = ent24;
15   }
16   if (answ === 'do') {
17     //Add the first choice to the selected options
18     conv.data.pakys += 'A';
19     //Remove the first option from the followup question
20     audiourl2 = audiourl2.replace('1','');
21     audiourl += '431.ogg';
22   } else if (answ === 'part') {
23     conv.data.pakys += 'B';
24     audiourl2 = audiourl2.replace('2','');
25     audiourl += '432.ogg';
26   } else if (answ === 'long') {
27     conv.data.pakys += 'C';
28     audiourl2 = audiourl2.replace('3','');
29     audiourl += '433.ogg';
30   } else if (answ === 'pigeon') {
31     conv.data.pakys += 'D';
32     audiourl2 = audiourl2.replace('4','');
33     audiourl += '434.ogg';
34   } else {
35     conv.data.testi = 'fallevent24_3';
```

```
36        return conv.followup('fallevent24', {
37          response: 'fallback'
38        });
39      }
40      conv.data.fallbackCount = 0;
41      if (conv.data.kyyhky) {
42        conv.contexts.set('int24_3',1,{});
43      } else {
44        conv.contexts.set('int24_4',1,{});
45      }
46      audiourl2 =  host + audiourl2 + '.ogg';
47      conv.data.kysurl = audiourl2;
48      conv.data.previous = ['24_3event',answ,'int24_2'];
49      if (conv.user.verification === 'VERIFIED') {
50        conv.user.storage.previous = conv.data.previous;
51        conv.user.storage.pakys = conv.data.pakys;
52        conv.user.storage.kysurl = conv.data.kysurl;
53      }
54      // For selecting the correct text
55      const eve = '24' + conv.data.pakys.charAt(conv.data.pakys
    .length - 1);
56      const txt = Texts.bubble(eve);
57      const ssml = Utils.playMulti(`<media xml:id="audio1">
58        <audio src="${audiourl}"/></media>
59      <media xml:id="audio2" begin="audio1.end-0.0s">
60        <audio src="${audiourl2}" clipBegin="0s" clipEnd="100s"
    />
61      </media>`
62      );
63      conv.ask(new SimpleResponse({speech: ssml, text: txt}));
64    });
65
66      //The utils helper functions
67      selector(choices,response) {
68      //Checks which ordinal was given as a response
69      //if any
70      switch (response) {
71        case 'one':
72          return choices[0];
73        case 'two':
74          return choices[1];
75        case 'three':
76          if (choices.length > 2) {
77            return choices[2];
78          } else {
79            return 'notresponse';
80          }
81        default:
82          return 'notresponse';
```

```
83      }
84    },
85    //Returns string of possible follow-up choices
86    //Returns an array of the choices that have not
87    been asked yet
88    pusher(str) {
89      var choices = [];
90      var url = '';
91      if (str.length < 1) {
92        return ['XXX',['do','part','long']]
93      }
94      if (str.indexOf('A') === -1) {
95        choices.push('do');
96        url += '1';
97      }
98      if (str.indexOf('B') === -1) {
99        choices.push('part');
100       url += '2';
101     }
102     if (str.indexOf('C') === -1) {
103       choices.push('long');
104       url += '3';
105     }
106     if (str.indexOf('D') === -1) {
107       choices.push('pigeon');
108       url += '4'
109       }
110     if (str.indexOf('E') === -1) {
111       choices.push('pond');
112       url += '5';
113       }
114       return [url,choices];
115     }
```

Code segment 5.5: Utilising the audio file name structure to for logical processing

Chapter 6

Evaluation

After publishing of the Action, the Analytics-section of the Action console has registered 350 unique users. Majority of the users stem from November and December, the first months of the publication, with some additional users in April and June. Otherwise the analytic shows gaps with no usage. The numbers seem peculiar and unreliable due to Google's data policies. In order to anonymise the data, low usage data is hidden or shown as zero, leaving gaps to the charts [68]. However, it still indicates a certain degree of the usage of the Action. While the implementation was successful, fulfilling and to some extent exceeding the requirements and expectations that were set during the planning phase, improvement in current technology and methodology would elevate the potential of interactive drama for smart speaker platforms. Our perception of the potential and the shortcomings of the concept was formed based on the implementation process, our personal testing and quality control, and the conducted user tests. To evaluate the implementation, I will analyse our findings of our personal testing process and the user tests.

6.1 Testing

Testing is an important part of the development cycle - especially when implementing a new concept. During the development phase, the application was tested frequently in order to maintain integrity, and to interfere with bugs or other problems early on. The testing was conducted in two parts, which can be described as modular and holistic testing strategies. The aim of the former was to verify the behaviour and technical premises of the individual parts of the implementation, whereas, the latter was utilised to test and to improve the actual usage and functionalities of the Action in a real environment - the smart speaker. Since every input can not be tested, we

employed a few coverage criteria to assess the functionality of relevant areas. While we did not employ formal criteria or test requirements, appropriately applied coverage criteria is deemed to be the best strategy for deciding what should be tested [69].

6.1.1 Modular testing

The role of the modular testing was to ensure that each individual component, or intent, and its transitions functioned correctly. The basic idea was that if each individual part was functioning, then ideally the whole structure would also be operational. For an intent this generally means, that the intent is entered correctly, the intent plays the corresponding output, and that the intent is followed by an appropriate intent. However, the modular testing addressed only the technical integrity of the Action, taking into account the behaviour with some correct and incorrect inputs, neglecting for instance partially correct inputs or different variable states from a longer span.

There are three possible expected scenarios that follow the execution of an intent. Intent can lead to the next intent, raise a fallback, or raise a no-input fallback. In case of either of the fallbacks, the Action needs to be able to proceed to the correct intent. Additionally, the received parameters and internal processing of an intent were crucial and not necessarily self-evident from the inputs. The testing was conducted according to two coverage criteria, one regarding the structure and transitioning of the intents, and the other regarding the data flow, to ensure that the received and created values are formed and used correctly.

The testing was conducted amidst the implementation, usually after creating a few intents and their webhook handlers. The reason for this was to maintain and to build upon a continuously functional Action. To verify structural correctness, the testing of each intent covered the following situations: entering the intent and playing the correct content, raising appropriate fallbacks and recovering from them, recovering from helper functions, such as repeat, and proceeding to the next intent - which can also be seen as the first task of the next intent. This was needed to be conducted for all intents. To tackle this, we implemented a shortcut-function in order to be able to travel into convenient points in the story, being a crucial task for the testing process.

The data flow criterion consisted of checking the received parameter in different situations, mainly through natural transitions and invoking the intent through the repeat-function, and that the conversation variables receive the expected value. The data flow was checked for critical intents, since not all intents changed other variables than the global game-state variable. The

defects in the specific game-state variable would most likely have stood out in the structural testing. Furthermore, once saving was implemented, the correspondence of the user data variables and the conversation data variables was checked.

6.1.2 Holistic testing

The testing of individual intents served as a premise for a playable Action, at least for the parts that were covered. However, it did not address the quality of the drama or the interaction. The holistic testing focused on how the story and the choices were presented, how the audio, sound design and music sounded, and how the making of the choices worked out. It fundamentally addressed the overall quality and cohesion of the different elements of an interactive audio drama on a smart speaker. We conducted the holistic testings as weekly play-throughs of the content that was implemented at the time.

The holistic testing revealed shortcomings of the employed entities, problematic or confusing choices, and incorrectly functioning sequences. The choice situations were assessed from a conversational standpoint, examining the performance of the matching, and the suitability of the different forms of choices in relation to the response that followed. The choices were selected using a natural form of the answer, or in other words how the choice might be made in natural dialogue, with some different possible variations. When the entity was not matched, due to the altered phrase or misinterpretation, we updated the entity entries accordingly, in order to take some common alterations and misinterpretations into account. Holistic tests also exposed faults that did not occur in the modular tests, or were not yet covered by them.

6.1.3 Findings

The implementation served the concept suitably well. While not being necessarily the most robust or powerful way of implementing a story engine, even simple control elements, such as Dialogflow contexts, conversation data variables and if-statements, can simulate complex logic and handle different scenarios. There were no occasions where the required story progression logic would have been out of scope of the tools, although our implementation did use a relatively linear approach.

Due to the technical limitations of the used framework, the user cannot respond to the speaker while it is playing audio, unless the user overrides the response of the Action with "Hey Google". Similarly, when the Action

awaits for the user's input, the Action can not play audio. The interaction model could be described as turn based, and it therefore utilises a limited form of reactivity. This is partly due to the request and response scheme. The assistant receives a request from the user and presents a response based on the request. The Action cannot be interacted with reasonably during the response. The audio can be interrupted, but the logical effects have already taken place, restricting the possibilities of for instance intensive scenes.

The smart speaker had surprisingly good interpretation abilities compared to our previous experiences on speech parsing technologies. However, the assistant seemed to perform better on the smart speaker than for instance on mobile devices. The matching capabilities were upgraded at least once during the development. With the update, fuzzy matching was added to the entity-options. This improved the matching of the intents, since the fuzzy matching allowed small variations in the input. To further enhance the interpretation and matching capabilities of the Action, it could possibly use the active contexts as clues in the future. The context would function as a gateway to a pool of possible upcoming utterances, serving as a suggestion for the interpretation.

At some point the matching of the intents began to over-perform. The intents were being matched, even though the input did not match any of the entities or training phrases. A common phrase, that was able to access many intents it was not suited for, was "yea yea". We tried setting the parameters as required for some intents, but it created more problems and unpredictable behaviour, such as giving responses, that we were unable to find from the whole project. If we would have wanted to use the required parameters, we should have turned on the webhook for slot filling and managed the process with the webhook handler. Instead, we considered that it was easier to examine the extracted parameters in our webhook, and invoke a fallback if they were empty. In any case, the suddenly started behaviour was strange.

While our test methodology did cover extensively the operation of the Action, there were situations that we did not account for. While intent can indeed lead to the next intent or either of the fallbacks, a fallback can also lead into a fallback. In most cases, this is irrelevant, since the processing used by the fallbacks are similar and interchangeable. However, sequences which used special functionality were prone to problematic situations. For instance, such a situation arose with an intent that expects a certain input, but leads to the next intent even if fallback is invoked, using the anyfall-context. When tested, both the regular fallback and no-input worked as intended. However, when an incorrect input is used after a no-input fallback was triggered, the default fallback without the anyfall-context was triggered, and the Action gets stuck in a loop, until the default fallback redirects it to the next intent.

One unfortunate feature turned out to be the default Google Assistant functionalities. Google Assistant uses a set of wake-up words in order to invoke some basic functionalities, such as accessing calendar or email. If those words were used within the Action, Google Assistant quit the Action and invoked the corresponding functionality. For instance, we had one choice using the word *meeting*, that had to be revised, because the assistant opened the calendar when the input was used. We did not find a list of reserved words to check up on either. Besides, this is probably unwanted behaviour on behalf of the assistant, which will hopefully be fixed in the future.

The saving system worked in the end seemingly well - after the user had quit the Action, or the Action or assistant had crashed, verified users could continue the story from the latest intent. There were troubles with updating the variables of the user storage in cases, where event-invocation or quitting of the Action were used, even though the processing of the intent was otherwise executed. The sole remaining problem is that the saving is possible for verified users only. Moreover, the Assistant does not express the effects of the verified-status or the factors that affect it exactly clear for the end-user. One option would be to implement a voluntary login, and handle the saving with a custom database, that could be hosted for example in Firebase.

Open questions offered special interaction that gave the player an opportunity for free form responses. However, we could not seize their full potential, since the input could not be used in the audio. For future implementations, since SSML supports simultaneous execution and control of different audio, one or more characters could be speech synthesized. With the capabilities and the authenticity of the speech synthesis improving, as indicated by for instance Google Duplex [**?**], in the future one could have synthesized characters, that sound real and can express defined emotional traits. This would open up many possibilities for personal customization of the story, and even procedural methods, while keeping the sound design aesthetics of radio drama.

6.2 User testing

The purpose of the user testing was to evaluate the functionality of the Action with real users operating in realistic conditions. The user tests gave us insight on how the concept is received by the end-users. The user tests addressed the overall user experience of the interactive drama, focusing on for instance how the the story is conveyed and how the user perceives it, how the audio sounds, how the user feels making the choices, how the technical implementation supports the experience, how immersive the game-play

seems, and how intuitive the concept is to interact with.

6.2.1 Methods

We avoided using testers, who were familiar with the concept and the story, by having read the script or having played the text version of the drama for instance. We chose both people who were familiar with the technology and who had not used smart speakers before. We arranged supervised testing sessions, where the tester played the drama without guidance, with us monitoring silently and registering all meaningful remarks. Additionally, we conducted more informal testing sessions on our own accord, and the drama was exhibited and tried out in a few events. But we still remained from interfering with the interaction of the user and the drama as much as we could. In the supervised sessions, we monitored the reactions of the tester, the given inputs and what the assistant interpreted from them, and how the response of the tester related to the playing drama. Once the play-through was over, we collected free-form feedback and the tester usually expressed his or her thoughts.

6.2.2 Test results

The interactive drama was generally well received, with the concept and the story piquing interest of many. The implementation received commendation for its voice-acting, sound design, music, the story, and the interpretation abilities of the speaker, although the few mediocre voice-acting performances stood out, and some words and phrases gave difficulties for the speaker. Users enjoyed interacting with the speaker, with some even experimenting how freely and playfully the responses could be given. The atmosphere was deemed enjoyable and fitting. Some suggested utilising smart home functionalities to further enhance it. While the general consensus seemed to be that interactive drama needed focus and sitting down to play it, some reported that they had successfully played it in a busier environment, such as in a car while driving.

We found out that the format might take a bit of time to get accustomed to. Majority of the interaction problems occurred in the beginning section of the game, when the player was not yet adjusted to only listening to the story and making choices based on that. Quite many a player gave unsure responses before they got the hang of the interaction. For some, it was hard to keep on track of what was happening, since a lot of information was delivered even in the beginning. This was most evident with people with no previous experience on smart speakers and who were not proficient in English. It was

noted that in such situations subtitles would have been helpful, especially for the choices, because sometimes the user did not quite understand what was happening and how the user was expected to respond. Some users had also some troubles in distinguishing characters and remembering who they were when they re-appeared. However, all users achieved a smooth interaction with the drama after playing for a while.

The choices were generally perceived as interesting and meaningful, although there were situations the players would have wanted to have more impact on. Many players would have liked to explore the different branches to see what would have happened, if they had chosen differently, indicating apt difficulty of choice. Some suggested different game slots, so that the story could be re-played from certain points instead of being forced to start from the beginning in order to explore various branches. The "Say ready to continue" intersections evoked mixed feelings. Since there was no choice to be made, some found it disruptive and unnecessary, although many then understood the technical limitations of the concept.

The different choices were encompassed in the user testing, meaning that the users explored the different branches. However, there were two choices which almost invariably led to a specific branch. When the user has to decide, whether to stay on the pond, when the body parts are found, or to go quickly to the editorial office to be the first one to publish the findings, almost everyone chose to stay. Similarly, when deciding whether to go into the coffin or let Miss Vanamo go instead, the players chose to go themselves. Perhaps the other option did not seem to offer sufficiently interesting consequences in these cases.

Open questions were sometimes found confusing. The open questions tended to work well with people who were able to throw themselves and act along the drama. However, others were sometimes confused about what they should answer, since no options were given, sometimes even pondering if they had missed something important. Additionally, since the user could have said anything, the given response did not always match what the user said. Such occasions were usually disregarded as fleeting comical moments. Similarly, the mini-game about the number of different body parts found in the pond startled many, since they did not expect that they were supposed to remember the numbers. In various cases, the user remembered the correct answers after answering one or two questions. Therefore, we implemented an option to revise the number before proceeding with the story.

As the most common operational problem, many a user gave their response too immediately after the audio, which often led to the speaker not registering or recognizing the input, and thus confusion of the user. To tackle this we edited the audio to be as tight as possible, but it did not remove the

problem. The smart speaker interface is equipped with blinking led lights, that indicate for instance when the assistant is expecting an input and when it is processing the given command. However, most users were not aware or did not take this into account otherwise. Some users would have wanted the speaker to give a sound signal after receiving an input.

The instructions for using the helper features *repeat* and *pause* were cut out of the beginning, due to the issue of the speaker listening to itself, that had emerged after the user tests and the publication. The problem did not occur even once during the tests. However, there might have been some updates on Google's behalf in between the occasions. The repeat functionality was successfully used in the user tests to replay the scene, and some tried out also the pause feature. However, few testers voiced their wishes for a functionality for skipping the dialogue or re-playing only the presented choices, instead of listening to the whole sequence again. While it could have been done, it would have required separated audio files or processing from the webhook's side, and yet new commands for the player to remember. We deemed that the gained benefits would not outweigh the work needed, since it was a marginal issue.

Entities needed to be quite creative at times. On some occasions, when the assistant did not catch what the user was saying, the user dumbed down the following responses unnecessarily. Also, different ways of saying the correct options could sometimes lead to the wrong choice, such as "third one" going to the first one or "no they're fine" going to no. During the user tests, we collected all the misinterpreted user inputs that were likely to recur, and added them to the entity entries, in order to catch the slightly mispronounced or misinterpreted inputs. For instance, "ready" was quite often misinterpreted as "reddit", so adding it to the entity enhanced the matching abilities of the agent. Once the most common mistakes and different forms of input were updated, the matching worked typically well, receiving a couple of positive remarks from the testers as well.

While we tested and presented the Action, we noticed that not too many invocations of the Actions came from outside the effect of these events. One shortage in our implementation was that it uses only explicit invocation. This means that the Action has to be invoked with its name, The Dead Are Speaking or discovered through the explore-functionality of Google Assistant, using part of the name or apt key-words. If we had defined an implicit invocation, Google Assistant might have redirected users to our Action, if they were looking for interactive games, radio drama or something comparable. This would have probably increased the discoverability of the Action.

Chapter 7

Conclusions

Smart speakers have rapidly gained popularity in recent years, with their virtual assistants, such as Google Assistant and Alexa, becoming available in increasingly many devices. The functionalities of virtual assistants and smart speakers are expanded by third-party applications, for instance Actions for Google Assistant and Skills for Alexa. The functionalities of the different applications range from alarm applications to interactive fiction, but proper interactive drama is few and far between. We implemented an interactive drama for Google Home smart speakers, as a collaboration between Yle and Aalto University.

Interactive drama for smart speakers is a format that has its roots in radio plays, interactive game-books, and other interactive narratives. There are several characteristics that should be taken into account when writing an interactive drama, such as delivering the narrative, how to formulate and present interesting and meaningful choices, executing branching story-lines, and forming distinct and relevant characters. The format also provides many possibilities for the drama.

In our implementation of the story, we respected the conventions of radio plays and utilised different interaction types. We approached the story from the narrative point of view of audio drama, putting emphasis on natural dialogue and embedding the choices in the dialogue when it could be done sensibly. The story unfolds fairly linearly, with regional branchings giving the player a sense of influence and offering interesting story development. We employed an event based system in the different story branches, with certain story events having consequences later in the story. The story contained multiple endings.

There are multiple ways of creating Actions for Google Assistant. Google offers tools, platforms and libraries for implementation, testing, hosting and deployment of the Action, but own solutions can be used to an extent. Many

of the technologies are in active development, and updates have been made on a steady basis since the beginning of the implementation. We chose to use supported and documented solutions.

The basis of the implementation was created with Dialogflow, a natural language understanding framework. The main elements of a Dialogflow agent are intents, entities, contexts, events and fulfillment. The Dialogflow agent uses Action on Google integration for building an Action. The fulfillment logic was implemented as a webhook, that was deployed to Firebase along the hosted audio files. The webhook handles the internal logic of the intents, being responsible of presenting the correct content, and the different features and functionalities of the Action.

In the evaluation we determined that the implementation was successful and fulfilling its role, and therefore established the potential of the technologies and the methodology. We also found out that smart speaker interactive drama is a format that might take time to get accustomed to, although the barrier seems to be low. It shares many challenges with radio drama, presenting new ones through its technical nature. However, it also presents many possibilities and future potential. The users like to interact with the drama and the speaker, appreciating the sound aesthetics and their impact on the story. To improve the interactive experience, the format could benefit from changes and enhancements on different areas, such as using plausible enough speech synthesis for providing more personalised story, using the smart home for the atmosphere, and accessing functions for customized assistant behaviour, in order conceal the technical side behind immersive elements that support the story.

Bibliography

[1] Global smart speaker q4 2019, full year and forecast, April 2019. https://www.canalys.com/newsroom/canalys-global-smart-speaker-installed-base-to-top-200-million-by-end-of-2019. Accessed 5.6.2020.

[2] Paul J. Sallomi. Smart speakers: Growth at discount, 2018. https://www2.deloitte.com/us/en/insights/industry/technology/technology-media-and-telecom-predictions/smart-speaker-voice-computing.html, Accessed 23.3.2020.

[3] Kansallinen Radiotutkimus. Finnpanel kansallinen radiotutkimus. Technical report, 2019.

[4] Bret Kinsella. Smart speaker shipments in the nordic countries reached 900k in 2019. https://voicebot.ai/2020/02/29/smart-speaker-shipments-in-the-nordic-countries-reached-900k-in-2019/ Accessed 25.5.2020.

[5] Wesa Aapro. The dead are speaking - a feature-length audio drama for smart speakers, 2019. https://yle.fi/aihe/artikkeli/2019/12/09/the-dead-are-speaking-a-feature-length-audio-drama-for-smart-speakers. Accessed 20.4.2020.

[6] Amy Watson. Global box office revenue from 2005 to 2018, 2020. https://www.statista.com/statistics/271856/global-box-office-revenue/. Accessed 27.03.2020.

[7] 2019 year in review. digital games and interactive media, 2020. https://www.superdataresearch.com/2019-year-in-review. Accessed 27.03.2020.

[8] Davey Winder. Google reveals new privacy options for 500 million users at ces 2020, 2020.

https://www.forbes.com/sites/daveywinder/2020/01/08/google-confirms-new-privacy-options-for-500-million-users-at-ces-2020/. Accessed 30.3.2020.

[9] The inspection chamber - the insider story. https://www.bbc.co.uk/taster/pilots/inspection-chamber Accessed 21.5.2020.

[10] Gamebook. https://en.wikipedia.org/wiki/Gamebook Accessed 5.6.2020.

[11] Radio drama. https://en.wikipedia.org/wiki/Radio_drama Accessed 5.6.2020.

[12] Newman, Barry. Wall Street Journal; Thrilling Days Of Yesteryear — Via the Internet, 2010. https://www.wsj.com/articles/SB10001424052748704240004575085313479028540?mod=rss_Arts_and_Entertainment. Accessed 28.3.2020.

[13] Elke Huwiler. A narratology of audio art: Telling stories by sound. *Audionarratology: Interfaces of Sound and Narrative*, 52:99, 2016.

[14] Egil Törnqvist. *Transposing Drama: Studies in Representation*. Springer, 1991.

[15] Raumstation. Der ohrenzeuge. interaktives krimihörspiel, 2005. http://www.raumstation.de/?p=303. Accessed 19.07.2020.

[16] Brian Magerko. Story representation and interactive drama. In *AIIDE*, pages 87–92, 2005.

[17] Michael Mateas and Andrew Stern. Façade: An experiment in building a fully-realized interactive drama. In *Game developers conference*, volume 2, pages 4–8, 2003.

[18] Interactive story-telling. https://en.wikipedia.org/wiki/Interactive_storytelling Accessed 15.7.2020.

[19] Marc Cavazza, Fred Charles, and Steven J Mead. Interacting with virtual characters in interactive storytelling. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 318–325, 2002.

[20] Fred Charles, Miguel Lozano, Steven Mead, Alicia Fornes Bisquerra, and Marc Cavazza. Planning formalisms and authoring in interactive storytelling. In *Technologies for interactive digital storytelling and entertainment, TIDSE 03 proceedings*. Fraunhofer IRB Verlag, 2003.

[21] Barbaros Bostan and Tim Marsh. Fundamentals of interactive storytelling. *Academic Journal of Information Technology*, 3, 08 2012.

[22] Sheizaf Rafaeli. Interactivity: From new media to communication, 1988.

[23] Leah A Lievrouw and Sonia Livingstone. *Handbook of new media: Social shaping and consequences of ICTs*. Sage, 2002.

[24] Ava Mutchler. Smart speaker timeline. `https://voicebot.ai/2018/03/28/timeline-voice-assistant-smart-speaker-technology-1961-today/` Accessed 25.5.2020.

[25] Amazon echo. `https://en.wikipedia.org/wiki/Amazon_Echo` Accessed 8.4.2020.

[26] Google nest and google home device specifications. `https://support.google.com/googlenest/answer/7072284?hl=en` Accessed 20.5.2020.

[27] Cartenberg, Chaim. Apple announces HomePod speaker to take on Sonos, 2017. `https://www.theverge.com/2017/6/5/15732144/apple-homepod-speaker-announced-siri-price-release-date-wwdc-2017`. Accessed 5.6.2020.

[28] Sonos smart speakers. `https://www.sonos.com/en-us/shop/smart-speakers` Accessed 20.5.2020.

[29] Global smart speaker q4 2019, full year and forecast, 2019. `https://www.canalys.com/newsroom/smart-speaker-market-q2-2019`. Accessed 5.6.2020.

[30] Global smart speaker q4 2019, full year and forecast, 2020. `https://www.canalys.com/newsroom/-global-smart-speaker-market-Q4-2019-forecasts-2020`. Accessed 5.6.2020.

[31] Virtual assistant. `https://en.wikipedia.org/wiki/Virtual_assistant` Accessed 22.5.2020.

[32] Matthew B Hoy. Alexa, siri, cortana, and more: an introduction to voice assistants. *Medical reference services quarterly*, 37(1):81–88, 2018.

[33] Julia Hirschberg and Christopher D. Manning. Advances in natural language processing. *Science 349*, 3:261–266, 2015.

[34] Alexa voice service (avs) sdk overview. https://developer.amazon.com/en-US/docs/alexa/avs-device-sdk/overview.html Accessed 20.7.2020.

[35] Google assistant sdk overview. https://developers.google.com/assistant/sdk/overview Accessed 20.7.2020.

[36] Bret Kinsella. Google assistant actions grew quickly in several languages in 2019, matched alexa growth in english. Accessed 20.7.2020.

[37] Google assistant homepage. https://assistant.google.com/learn/ Accessed 30.3.2020.

[38] Bret Kinsella. Google assistant actions grew quickly in several languages in 2019, matched alexa growth in english, 2020. Accessed 20.7.2020.

[39] What can your assistant do? https://assistant.google.com/explore. Accessed 26.7.2020.

[40] Alexa skills. https://www.amazon.com/alexa-skills/b?ie=UTF8&node=13727921011. Accessed 26.7.2020.

[41] Eric Hal Schwartz. Voice assistants very prone to accidentally waking up and recording long audio clips: Study. Accessed 20.7.2020.

[42] Devin Coldewey. This family's echo sent a private conversation to a random contact. https://techcrunch.com/2018/05/24/family-claims-their-echo-sent-a-private-conversation-to-a-random-contact/ Accessed 20.7.2020.

[43] Jeb Su. Why amazon alexa is always listening to your conversations: Analysis. https://www.forbes.com/sites/jeanbaptiste/2019/05/16/why-amazon-alexa-is-always-listening-to-your-conversations-analysis/ Accessed 20.7.2020.

[44] Liam Tung. Google home mini flaw left smart speaker recording everything. GoogleHomeMiniflawleftsmartspeakerrecordingeverything/ Accessed 20.7.2020.

[45] Josephine Lau, Benjamin Zimmerman, and Florian Schaub. Alexa, are you listening? privacy perceptions, concerns and privacy-seeking behaviors with smart speakers. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):1–31, 2018.

[46] Nathan Malkin, Joe Deatrick, Allen Tong, Primal Wijesekera, Serge Egelman, and David Wagner. Privacy attitudes of smart speaker users. *Proceedings on Privacy Enhancing Technologies*, 2019(4):250–271, 2019.

[47] Aj Dellinger. Gifting a smart speaker this holiday? only one passes privacy tests. `https://www.forbes.com/sites/ajdellinger/2019/11/30/gifting-a-smart-speaker-this-holiday-only-one-passes-privacy-tests/` Accessed 20.7.2020.

[48] Alexa skills documentation. `https://developer.amazon.com/en-US/docs/alexa/custom-skills/develop-skills-in-multiple-languages.html` Accessed 15.5.2020.

[49] Speech synthesis markup language (ssml) version 1.1. `https://www.w3.org/TR/speech-synthesis/`. Accessed 7.6.2020.

[50] Skills documentation ssml reference. `https://developer.amazon.com/en-US/docs/alexa/custom-skills/speech-synthesis-markup-language-ssml-reference.html`. Accessed 7.6.2020.

[51] Google cloud documentation ssml reference. `https://cloud.google.com/text-to-speech/docs/ssml`. Accessed 7.6.2020.

[52] Pair two speakers for stereo sound. `https://support.google.com/googlenest/answer/7559493?hl=en` Accessed 20.5.2020.

[53] Announcing actions builder actions sdk: New tools optimized for the google assistant. `https://developers.googleblog.com/2020/06/announcing-actions-builder-actions-sdk.htm`. Accessed 27.6.2020.

[54] Actions on google documentation. `https://developers.google.com/assistant/conversational/`. Accessed 7.6.2020.

[55] Build actions for the google assistant (level 1). `https://codelabs.developers.google.com/codelabs/actions-1/`. Accessed 7.6.2020.

[56] Actions on google github. `https://github.com/actions-on-google`. Accessed 7.6.2020.

[57] Manage functions deployment and runtime options. `https://firebase.google.com/docs/functions/manage-functions`. Accessed 29.7.2020.

[58] Release notes. `https://cloud.google.com/dialogflow/docs/release-notes`. Accessed 29.7.2020.

[59] Actions reference, node.js v2. `https://developers.google.com/assistant/actions/reference/nodejsv2/overview` Accessed 1.5.2020.

[60] Dialogflow homepage. `https://dialogflow.com/` Accessed 30.3.2020.

[61] Dialogflow documentation. `https://cloud.google.com/dialogflow/docs/` Accessed 30.3.2020.

[62] Do users respond favourably to long-form fictional interactions with an alexa skill?, 2018. BBC RD, IRFS - January 2018. UX Analysis Recommendations.

[63] Perttu Häkkinen and Vesa Iitti. *Valonkantajat*. Like Kustannus, 2015.

[64] Pyry Waltari. Kun kansa itse ratkaisee arvoituksen : Tattarisuon tapauksen määrittely käytännöllisessä diskurssissa. Master's thesis, University of Helsinki, Faculty of Social Sciences, Department of Political and Economic Studies, 2011. `http://hdl.handle.net/10138/26612`.

[65] Perttu Häkkinen, Ville Similä. Lähteellä. *HS Kuukausiliite*, (11):61–69, 2010.

[66] Claire Grove and Stephen Wyatt. *So You Want to Write Radio Drama?* Nick Hern Books, 2013.

[67] Ten tips for writing a play for radio. `https://www.bbc.co.uk/programmes/profiles/tnkQgSgPJVWM4ZpZ3hHbjv/ten-tips-for-writing-a-play-for-radio` Accessed 15.5.2020.

[68] Analytics. `https://developers.google.com/assistant/console/analytics/`. Accessed 23.7.2020.

[69] Paul Ammann and Jeff Offutt. *Introduction to software testing*. Cambridge University Press, 2016.

Appendix A

Webhook code

A.0.1 Fallback handlers

```
app.intent('Default Fallback Intent', (conv) => {
  conv.data.fallbackCount++;
  const cevent = conv.data.previous[0];
  const prompt = Reprompts.getURL(cevent);
  const audiourl = host + prompt[0] + '.ogg';
  const ctx = prompt[1];
  var eve = prompt[2];
  var cparam = prompt[3];
  conv.contexts.set(ctx,1,{})
  if(cevent === '22B_1event') {
    conv.followup('22B_2action', {
      response: cparam
    });
  }
  if (prompt[0] === 'repeater') {
    eve = 'repeat';
    cparam = 'repeat';
    conv.data.fallbackCount = fbc;
  }
  if (conv.data.fallbackCount < fbc) {
    if (conv.data.fallbackCount > 1 && Utils.tipsy(cevent) &&
    cparam !== 'ready' && Reprompts.getType(cparam) !== '
   binarr') {
    conv.ask(new SimpleResponse({speech: Utils.playSimple(
   host+'99K.ogg'), text: 'When making a choice, you can also
    use numbers.'}));
    }
  const txt = Texts.bubblef(conv.data.previous[0]);
  const ssml = Utils.playSimple(audiourl);
  conv.ask(new SimpleResponse({speech: ssml, text: txt}));
```

```
27    } else if (eve === '3D_2event' || eve === '3D_3event' ||
       eve === '3D_4event') {
28      conv.followup(eve, {
29        number: cparam
30      });
31    } else {
32      conv.followup(eve, {
33        response: cparam
34      });
35    }
36  });
37
38  app.intent('AnyFallback', (conv) => {
39    const cevent = conv.data.previous[0];
40    conv.data.testi = 'anyfall';
41    conv.data.fallbackCount = 0;
42    const prompt = Reprompts.getURL(cevent);
43    const ctx = prompt[1];
44    const eve = prompt[2];
45    const cparam = prompt[3];
46    conv.contexts.set(ctx,1,{})
47    conv.followup(eve, {
48      response: cparam
49    });
50  });
51
52  app.intent('NoInput', (conv) => {
53    const repromptCount = parseInt(conv.arguments.get('
       REPROMPT_COUNT'));
54    const cevent = conv.data.previous[0];
55    const prompt = Reprompts.getURL(cevent);
56    const audiourl = host + prompt[0] + '.ogg';
57    conv.contexts.set(prompt[1],1,{})
58    if (repromptCount === 0 && prompt[0] !== 'repeater') {
59      const txt = Texts.bubble(cevent);
60      const ssml = Utils.playSimple(audiourl);
61      conv.ask(new SimpleResponse({speech: ssml, text: txt}));
62    } else if (conv.arguments.get('IS_FINAL_REPROMPT')) {
63      conv.close('Let us try this some other time!')
64    } else {
65      conv.followup('repeat', {
66        response: 'repeat'
67      });
68    }
69  });
70
71  app.intent('AnyNoInput', (conv) => {
72    const cevent = conv.data.previous[0];
73    const prompt = Reprompts.getURL(cevent);
```

```
  const ctx = prompt[1];
  const eve = prompt[2];
  const cparam = prompt[3];
  conv.contexts.set(ctx,1,{})
  conv.followup(eve, {
    response: cparam
  });
});
```

A.0.2 Functionality handlers

```
//Intent for loading the previously saved data for the user
app.intent('Load', (conv) => {
    conv.data.previous = ['1_1event','ready','DefaultWelcomeIntent-followup'];
    if (conv.user.verification === 'VERIFIED') {
      conv.data.sum = conv.user.storage.sum;
      conv.data.fallbackCount = conv.user.storage.fallbackCount;
      conv.data.day = conv.user.storage.day;
      conv.data.vpoints = conv.user.storage.vpoints;
      conv.data.bpoints = conv.user.storage.bpoints;
      conv.data.minipeli = conv.user.storage.minipeli;
      conv.data.checkpoint = conv.user.storage.checkpoint;
      conv.data.points = conv.user.storage.points;
      conv.data.experts = conv.user.storage.experts;
      conv.data.testi = conv.user.storage.testi;
      conv.data.visits = conv.user.storage.visits;
      conv.data.kyyhky = conv.user.storage.kyyhky;
      conv.data.rethink = conv.user.storage.rethink;
      conv.data.sreveal = conv.user.storage.sreveal;
      conv.data.previous = conv.user.storage.previous;
      conv.data.peliansw = conv.user.storage.peliansw;
      conv.data.kysurl = conv.user.storage.kysurl;
      conv.data.julkaise = conv.user.storage.kysurl;
      conv.data.nice = conv.user.storage.nice;
      conv.data.kuskikys = conv.user.storage.kuskikys;
      conv.data.pakys = conv.user.storage.pakys;
      conv.data.testi = 'ladattu loppuun';
      conv.followup('repeat', {
        response: 'repeat'
      });
  } else {
    conv.ask('Loading is possible only for verified users.');
  }
});
```

```
34
35 //Intent for handling quitting
36 app.intent('Exit', (conv) => {
37   var audiourl = host + 'QUITNOTSAVED.ogg';
38   var text = 'Goodbye for now!';
39   if (conv.user.verification === 'VERIFIED') {
40     audiourl = host + 'QUITSAVED.ogg';
41     text = 'Your progress has been saved. ' + text;
42   }
43   conv.close(new SimpleResponse({speech: Utils.playSimple(
      audiourl), text: text}));
44 });
45
46 //Intents for deleting stored user data
47 app.intent('Forget', (conv) => {
48   conv.ask('Are you sure you want to erase saved game data?')
49 });
50 app.intent('Forgot', (conv, {binarr}) => {
51     if (binarr === 'yes') {
52     conv.user.storage = {};
53     conv.close('Your game data has been cleared');
54   } else {
55     conv.followup('repeat', {
56       response: 'repeat'
57     });
58   }
59 });
60
61 // Intent for repeating the (current) sequence stored
62 //in conv.data.previous
63 app.intent('repeat', (conv) =>{
64   //helper intent for replaying the current event (intent)
      with the current choice
65   const cevent = conv.data.previous[0];
66   const cparam = conv.data.previous[1];
67   const ccontext = conv.data.previous[2];
68   conv.contexts.set(ccontext,1,{})
69   conv.user.storage.testi = 'repeat';
70   //const param = Reprompts.getParam(cevent);
71   const param = Reprompts.getType(cparam);
72   if (param === 'binarr') {
73     conv.followup(cevent, {
74       binarr: cparam
75     });
76   } else if (cevent === '2_2event') {
77     conv.followup(cevent, {
78       ent2_1: cparam
79     });
```

```
80    } else if (cevent === '3D_2event' || cevent === '3D_3event'
        || cevent === '3D_4event') {
81      conv.followup(cevent, {
82        number: cparam
83      });
84    } else if (cevent === '23_2event') {
85      conv.followup(cevent, {
86        ent23_2: cparam
87      });
88    } else if (cevent === '24_3event' || cevent === '24_4event'
        || cevent === '24_5event') {
89      conv.followup(cevent, {
90        ent24: cparam
91      });
92    } else {
93    conv.followup(cevent, {
94      response: cparam
95    });
96    }
97  });
```

A.0.3 Implementation of the mini-game

```
1   //A mini-game intent for the number heads found in the pond
2     app.intent('3D_1minipeli', (conv) => {
3     conv.data.fallbackCount = 0;
4     conv.data.previous = ['3D_1event','ready','int3A_4'];
5     var audiourl = host + '130.ogg';
6     var txt = 'Uusi Helsinki newsroom.  Where  have you two
      been dawdling?  ';
7     if (conv.data.minipeli !== -1) {
8       audiourl = host + '180.ogg';
9       txt = 'Uusi Helsinki newsroom.  Hands  ?  ';
10    }
11    const ssml = Utils.playSimple(audiourl);
12    if (conv.user.verification === 'VERIFIED') {
13      conv.user.storage.previous = conv.data.previous;
14    }
15    conv.ask(new SimpleResponse({speech: ssml, text: txt}));
16  });
17
18  app.intent('3D_2minipeli', (conv, {number}) => {
19    conv.data.minipeli = 0;
20    const answ = number;
21    if (answ === '') {
22      return conv.followup('fallevent', {
```

```
23          response: 'fallback'
24        });
25      }
26      conv.data.fallbackCount = 0;
27      conv.data.peliansw[0] = answ;
28      if (answ === '4') {
29        conv.data.minipeli++;
30      }
31      conv.data.previous = ['3D_2event',answ,'int3D_1'];
32      if (conv.user.verification === 'VERIFIED') {
33        conv.user.storage.previous = conv.data.previous;
34        conv.user.storage.peliansw = conv.data.peliansw;
35        conv.user.storage.minipeli = conv.data.minipeli;
36      }
37      const audiourl = host + '131.ogg';
38      const ssml = Utils.playSimple(audiourl);
39      const txt = Texts.bubble(conv.data.previous[0]);
40      conv.ask(new SimpleResponse({speech: ssml, text: txt}));
41    });
42
43    app.intent('3D_3minipeli', (conv, {number}) => {
44      const answ = number;
45      if (answ === '') {
46        return conv.followup('fallevent', {
47          response: 'fallback'
48        });
49      }
50      conv.data.fallbackCount = 0;
51      conv.data.peliansw[1] = answ;
52      if (answ === '8') {
53        conv.data.minipeli++;
54      }
55      conv.data.previous = ['3D_3event',answ,'int3D_2'];
56      if (conv.user.verification === 'VERIFIED') {
57        conv.user.storage.previous = conv.data.previous;
58        conv.user.storage.peliansw = conv.data.peliansw;
59        conv.user.storage.minipeli = conv.data.minipeli;
60      }
61      const audiourl = host + '132.ogg';
62      const ssml = Utils.playSimple(audiourl);
63      const txt = Texts.bubble(conv.data.previous[0]);
64      conv.ask(new SimpleResponse({speech: ssml, text: txt}));
65    });
66
67    app.intent('3D_4minipeli', (conv, {number}) => {
68      const answ = number;
69      if (answ === '') {
70        return conv.followup('fallevent', {
71          response: 'fallback'
```

```
72        });
73      }
74      conv.data.fallbackCount = 0;
75      conv.data.peliansw[2] = answ;
76      if (answ === '16') {
77        conv.data.minipeli++;
78      }
79      conv.data.previous = ['3D_4event',answ,'int3D_3'];
80      var audiourl = host + '133.ogg';
81      if (conv.user.verification === 'VERIFIED') {
82        conv.user.storage.previous = conv.data.previous;
83        conv.user.storage.peliansw = conv.data.peliansw;
84        conv.user.storage.minipeli = conv.data.minipeli;
85      }
86      if (conv.data.renewed) {
87        return conv.followup('3D_5event', {
88          binarr: "no"
89        });
90      }
91      const ssml = Utils.playSimple(audiourl);
92      const txt = Texts.bubble(conv.data.previous[0]);
93      return conv.ask(new SimpleResponse({speech: ssml, text:
        txt}));
94    });
95
96    app.intent('3D_5minipeli', (conv, {response,binarr,
        additionals}) => {
97      const answ = response;
98      conv.data.previous = ['3D_5event',answ,'int3D_3'];
99      if (answ === 'one' || binarr === 'yes' || additionals ===
        'change') {
100       conv.contexts.set('int3A_4', 1, {});
101       conv.data.minipeli = 0;
102       conv.data.renewed = true;
103       if (conv.user.verification === 'VERIFIED') {
104         conv.user.storage.renewed = conv.data.renewed;
105         conv.user.storage.minipeli = conv.data.minipeli;
106       }
107       return conv.followup('3D_1event', {
108         response: "ready"
109       });
110     } else if (answ === 'two' || binarr === 'no' ||
        additionals === 'donot') {
111     conv.data.fallbackCount = 0;
112     if (!conv.data.points.includes('3D_5event')) {
113       if (conv.data.minipeli > 0) {
114         conv.data.bpoints += (conv.data.minipeli + 1) / 2;
115       }
116       conv.data.points.push('3D_5event')
```

```
117     }
118     const audiourl = host + '133B.ogg';
119     if (conv.user.verification === 'VERIFIED') {
120       conv.user.storage.previous = conv.data.previous;
121       conv.user.storage.bpoints = conv.data.bpoints;
122       conv.user.storage.points = conv.data.points;
123     }
124     const txt = Texts.bubble(conv.data.previous[0]);
125     const ssml = Utils.playSimple(audiourl);
126     conv.ask(new SimpleResponse({speech: ssml, text: txt}));
127   } else {
128     return conv.followup('fallevent', {
129       response: 'fallback'
130     });
131   }
132   });
```