Virtual Reality RPG Spoken Dialog System

Project report

Einir Einisson
Gísli Böðvar Guðmundsson
Steingrímur Arnar Jónsson

Instructor
Hannes Högni Vilhjálmsson

Moderator
David James Thue
Abstract

In computer games, interaction with NPCs have not changed much over time. In Virtual Reality the old ways of getting quest and having conversation with NPCs are not sufficient anymore. The near presence of the NPC gives the player the urge to want to communicate in a more natural way, verbally. In this paper we introduce a Spoken Dialog System for Virtual Reality RPG which can be used to assist in making spoken dialogues with NPCs.

1 Non-Player Character
2 Role Playing Game
Introduction

Approach

Conversation Service

Unity Mecanim animator

States behavior

Answer state

Global state

Exit state

Quest state

Examples

Speech Recognition

Unity’s Dictation Recognizer

Language Understanding Intelligent Service (LUIS)

Tokenizer

Other Features

Rogo Digital LipSync & Eye Controller

Amazon Polly

Conclusion and future work

Acknowledgements

Reference
Introduction

For the avid gamer, NPC communication have not changed that much throughout the years, at least since they have been able to speak and not just show a textbox with their dialog. This is perhaps because these methods have proved to be more than sufficient to keep the player engaged and to feel as a part of the story. In Virtual Reality however, the player no longer plays the game staring at a monitor, but is actually put inside the game with the help of a VR headset. This completely changes the players experience and he becomes much more involved in the game itself.

Because of this we feel the next step in NPC interactions is speaking with it. To make that happen we would need to convert the player’s speech to text and then evaluate that text somehow to find out what he is saying, or what he wants to do. Implementing this kind of a system was never going to be an easy task. But if done well enough, this system could help future VR game developers more easily make NPC dialogues with speech.

Approach

Conversation Service

To organize what state the dialog is supposed to be in, we made use of Unity’s built in Mecanim Animator. For this to work we created a gateway between the monobehaviour base class, which all scripts in game objects must derive from and state machine behaviour base class all scripts on animation states must derive from. These two base classes are not within each others hierarchy so we had to create this gateway formerly mentioned.

We named this gateway Conversation Service (formerly Event Dispatcher). It holds all current information for a conversation to take place. Such as who's turn it is to speak, what the NPC is suppose to say (play) and what the players current intent is in the form of a Token.

Unity Mecanim animator

Each dialog was created and designed with Unity animator in the form of Animator Controller. This allows us to design and visualize each NPC dialog state machine and control the animation of the NPC while he speaks during the conversation.

States behavior

Each state created for the dialog needs to get one of the three state behaviours scripts we created. All the states that provide the NPCs voices have the possibility to provide more than one sound for the given state. This is done so the NPC doesn't sound repetitive when entering the same state again. This also gives the player the feeling that the NPC
remembers what they had previously talked about. This of course works best if the developer of the dialog has a good overview of how the dialog is constructed.

Answer state

The states that needs to wait for a response from player before continuing, will have to get an answer state script attached to them. When an answer is received (in the form of token) the script will compare the answers with the answers the state is expecting to get. If there is a match the conversation will advance to the appropriate state. Some answer states can expect to receive a answer that has an entity attached to it. If that occurs the state will also check if the incoming entity will match one of the entities the state can use. If the state has only one or none entity specified then the state won't check for entity match even though the incoming token has an entity attached to it. The Token on the other hand will be sufficient for advancing to its appropriate state if they match.

Global state

Sometimes we want the NPC to say something without having the player to respond. For those occurrences we created the Global state scripts. They only have the audio and text for the state no answer handling occur in Global states.

Exit state

Player can jump out of a conversation at anytime but when a conversation reaches an endpoint the last state needs to have an Exit state attached to it. This state signals the conversation service that the dialog has ended and lets the Animator know it can safely continue with other animations the NPC was performing before the conversation took place.

Quest state

These kind of states is something we are still experimenting with and each script was made specifically for each NPC quest. One example of a use of this kind of a state behaviour is to calculate if an object the NPC has asked to be returned to him in a previous state has actually arrived or not.
Examples

Part of the dialog design with Kendra the creature.

Example how an answer state (called State Template during development) and Global State are used.
Speech Recognition

Before we can determine what intentions are behind the players voice input, we need to convert the voice recordings into text. After that is done, the process of analyzing the text can begin. For our system we categorized the players intentions into tokens. These tokens can then be used to decide how the NPC should respond to what the player said.

Unity's Dictation Recognizer

Unity offers a use of a speech recognition plugin called Dictation Recognizer, which we chose as our speech to text conversion tool. "Users can register and listen for hypothesis and phrase completed events. Start() and Stop() methods respectively enable and disable dictation recognition." Dictation Recognizer is dependant on Windows 10 and that the user's speech privacy settings is set on. (“Unity - Scripting API: DictationRecognizer”, 2017)

Language Understanding Intelligent Service (LUIS)

Designed to identify valuable information in conversations, LUIS interprets user goals (Intents) and distills valuable information from sentences (Entities), for a high quality, nuanced language model. (“LUIS: Language Understanding Intelligent Service.”, 2017)

LUIS is a machine learning service provided by Microsoft. This service is run in the Azure environment and we connect to it via HTTP. In the beginning we had to import the conversational domain for all the NPCs into LUIS. Then we had to train LUIS to understand the dialogues, by giving examples of utterances for each Intent we made. LUIS will then better learn how to evaluate different utterances provided by various players, meaning it is constantly improving, resulting in more accurate Intents being returned. We utilized this service to work with our Tokenizer, sending in the player input (which we get as a text from the Dictation Recognizer) and receiving a JSON reply which contains the results of LUIS's evaluation of that utterance.

```json
{
   "query": "tell me about your grandson",
   "topScoringIntent": {
      "intent": "KnowMore",
      "score": 0.9996198
   },
   "entities": [
      {
         "entity": "grandson",
         "type": "justin",
         "startIndex": 19,
         "endIndex": 26,
         "score": 0.9579803
      }
   ]
}
```

*Example of a JSON reply from LUIS*
In the LUIS reply we get a top scoring Intent with a score, which is floating point number from 0 - 1 (0 - 100%), depicting how sure LUIS is that this is the right intention of the player. Furthermore it optionally returns a list of Entities, which we use to better evaluate the Intent of the player. With Entities we can search for certain keywords in utterances and map them into a type for better decision making in the NPCs Finite State Machine.

Tokenizer

The tokenizer is a system built by us to establish a connection to LUIS and handle the response we get back. It checks if the received Token is sufficient for further use in the system and if the Token comes with an Entity. With each response we receive the score of how sure LUIS is that this token is the correct one. If the score is below our minimum score variable the tokenizer will send an error token to update Our Conversation Service.

If for some reason a connection to LUIS is unavailable the Tokenizer will make use of our old method of categorizing the text into Tokens. This method used a Trie implementation to search the text representation of the players voice recordings. The Trie is loaded up at startup with all the possible words and sentences we had predicted would connect with our Tokens. At the end of each input to the Trie a ‘$’ symbol is added as a leaf node, to signal that a Token match can be found when a prefix search is applied for comparing the incoming players input at runtime. (Castaño, “A More Efficient Text Pattern Search Using a Trie Class in .NET -.”, 2015)

Other Features

Rogo Digital LipSync & Eye Controller

For animating the lips of our NPCs we used Rogo Digital Lip Sync. To use it we simply have to provide it with an audio file that it will then auto sync it into a lip sync data file. The Data file we get from this process are then used in the dialog system to both play the audio and move the lips of the NPCs.

We also used the Eye Controller which comes with Rogo Digital Lip Sync. The Eye controller is used to keep a believable eye contact with the player while the conversation takes place.

Amazon Polly

All the voices of the NPCs are provided with a feature Amazon Web Services provide named Polly. Amazon Polly is a text to speech application that made it possible for us to simply write in a text as input and receive an audio file of a computer generated voice reading the text. Amazon Polly “uses advanced deep learning technologies to synthesize speech that sounds like a human voice.” (“Amazon Polly – Lifelike Text-to-Speech.”, 2017)
Conclusion and future work

It is our believe that a spoken dialog system like we've made can be a valuable asset for any developer who wants to make a Roleplaying Game in Virtual Reality. We mostly got positive feedback during our Usability tests, even though we hadn't integrated LUIS into our project at that time.

The prototype of the game we made is ready for further development. The environment of a beginning scene of the game is quite done and development of other features can begin. Such as a inventory, combat and quests systems. We are happy with our choice of allowing the player to move around the environment by teleportation as has become the standard in VR today. Specially because we were a bit sceptical of it in the beginning of the project, perhaps because we are more experienced with moving around by pressing the keys of the keyboard in similar non VR roleplaying games.

The spoken dialog system has some room for improvements. It could be made possible to allow the player interrupt the NPC, either to skip quickly to the next state of the dialog if the player is impatient or to make the NPC respond back with an offended tone since he got interrupted.

We've talked about refactoring the answer state into two seperate states that work together in order for the animator to play different animations one for while he is talking and the next one while he is listening to the player. This we imagine would give the player even more satisfying experience during a conversation with the NPC.

Another feature we contemplated on creating was a special spoken Dialog Creator UI. We soon discarded this after we realized that this has the potential to be a whole project on its own. The idea is to build a spoken dialog creator asset, ready to released to Unity asset store. This asset would need to have its own UI editor window where it's possible to create dialogs as we described in this report but with more suitable interface. The interface could for example allow the developer to define his own tokens for some special purpose he wants the NPC to handle and train LUIS accordingly with just few button clicks.

We've also wondered if it would be possible to automating a process of when a developer enters a text he wants for a given state, it immediately gets sent to Amazon Polly text to speech service, from there back to the project in the form of mp3 audio file, where Rogo Digital Lip Sync would auto sync the audio file and then get attached to the state the developer applied the text to.
Acknowledgements

We want to thank our instructor, Hannes Högni Vilhjálmsson, for solid guidance, constructive criticism and the general feedback during the journey this project has been. Also we would like to thank David Thue for valuable feedback throughout this process.

Reference


