Bachelor’s thesis

Information and Communications Technology | Game development

2019

Matti Mänty

CREATING A DIALOGUE EDITOR FOR THE UNITY GAME ENGINE

TURKU AMK

TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR’S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2019 | 35

Matti Mänty

CREATING A DIALOGUE EDITOR FOR THE UNITY GAME ENGINE

The goal of this thesis was to develop a prototype dialogue editor for the Unity game engine. Additionally, this thesis touches on the history of dialogue in digital games and compares currently available dialogue editor tools and solutions for digital games. A game development company Plush Pop Soft, which mainly focuses on mobile game development, commissioned this thesis. The commissioner already has a paid existing tool for creating dialogue for their games, but they had a need for a more lightweight editor that should be easily extendable.

To establish an understanding of and an agreement on the importance of dialogue and its presentation in digital games throughout history, the subject was researched using relevant research data and books. To gain practical knowledge, commercially successful games’ dialogue editors and Unity game engine’s dialogue assets were compared with ease of use in mind. All the tested dialogue editors had their advantages and disadvantages, but the commercial games’ editors stood out perhaps due to the editor being customized specifically for its target game. The dialogue assets and editors for Unity had often too many features; extra functionalities and menus might come in the way and ultimately slow down the development or performance.

Originally, the dialogue editor end product was to be a plain text-based editor that should resemble Neverwinter Nights 2 game’s dialogue editor visually and in regard to the basic functionalities. During the development, this plan had to be altered because of unforeseen problems with the game engine’s limitations and editor programming’s complexity. To prioritize a usable end result, an open source graphical node-based interface was used as a starting point for the editor. In the end, the final lightweight dialogue editor tool can be used to create branching dialogue structures with different optional conditions and actions for each dialogue line. While the resulting editor tool did not completely match the original design and the goals, the finished product has all the features originally needed. Those who are interested in developing similar tools might find this thesis helpful to gain some insight into the possible limitations and problems in editor development.

KEYWORDS:

dialogue, game programming, game design, software development

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikan koulutusohjelma (peliteknologia)

2019 | 35 sivua

Matti Mänty

DIALOGIEDITORIN KEHITTÄMINEN UNITY-PELIMOOTTORILLE

Opinnäytetyön tavoitteena oli ohjelmoida prototyyppiversio dialogieditorista Unity-pelimoottorille. Lisäksi opinnäytetyössä tutkittiin dialogin historiaa digitaalisissa peleissä sekä olemassa olevia ratkaisuja ja työkaluja dialogin tuottamiseen digitaalisiin peleihin. Opinnäytetyö toteutettiin toimeksiantona Plush Pop Soft -yritykselle, joka kehittää pääasiassa mobiilipelejä. Asiakkaalla oli jo käytössä maksullinen työkalu dialogin tuottamiseen, mutta yrityksellä oli tarve kevyemmälle ja laajennettavalle editorille.

Tutkimusosiossa tutustuttiin pelien historiaan eri kirjallisuuslähteiden kautta, joilla luotiin perusymmärrys dialogien tärkeyteen ja ilmaisutapaan peleissä. Tämän jälkeen tehtiin vertailua menestyneiden kaupallisten pelien dialogieditorien ja Unity-pelimoottorille saatavien dialogilisäosien välillä. Kaikissa editoreissa oli sekä hyviä että huonoja puolia, mutta tietyille peleille suunnatut editorit olivat aina tarkoitusperiään varten parhaiten toteutettuja. Unityn puolella testatuissa editoreissa tuli niiden mukana aina ylimääräisiä ja ei-haluttuja toiminnallisuuksia, jotka haittaavat pelinkehitystä.

Alkuperäisenä tavoitteena oli kehittää Unity-pelimoottorille pelkistetty tekstipohjainen dialogieditori, joka muistuttaisi ulkoasultaan mahdollisimman paljon Neverwinter Nights 2 -videopelin dialogieditoria. Kehitysvaiheessa kuitenkin törmättiin ennalta odottamattomiin ongelmiin Unity-pelimoottorin rajoituksissa sekä työkalujen ohjelmoinnin monimutkaisuudessa. Jotta työn tuloksena olisi toimiva dialogieditori, päädyttiin editorin pohjana käyttämään valmista avoimen lähdekoodin graafista vuokaaviotoiminnallisuutta. Lopputuloksena oli kevyt dialogieditori, jolla pystyy luomaan haarautuvia keskusteluja sekä asettamaan ehtoja eri haarojen välillä liikkumiseen, sekä keskusteluista aiheutuvia toimintoja. Vaikka lopputulos ei vastannut täysin alussa asetettuja tavoitteita, saattaa opinnäytetyöstä toimeksiantajan lisäksi hyötyä esiin nousseista ongelmakohdista vastaavanlaisen työkalun kehityksestä kiinnostuneet.

ASIASANAT:

dialogi, peliohjelmointi, pelisuunnittelu, ohjelmistotuotanto

CONTENTS

PICTURES

LIST OF ABBREVIATIONS

Bark	One or more lines of dialogue that is delivered without initiating proper dialogue functionality, usually with floating text or an audio clip. (Gamasutra 2019)
Mod	Short for modification, an alteration by users that change the original video game in some way (Wikipedia 2019).
NPC	Non-playable character. (Wikipedia 2019)
Plugin	A software component that adds a specific feature to an existing computer program (Wikipedia 2019).
RPG	Role-playing game. (Wikipedia 2019)
UI	User interface. (Wikipedia 2019)

1 INTRODUCTION

For decades, dialogue between the player and computer has been a part of video games, but most of the academic studies have focused on the narrative structure or the delivery methods of the dialogue. There has been little actual research on the dialogue editor solutions for producing, testing and modifying the dialogues. In addition to researching current professional dialogue editor solutions, the aim of this thesis was to create a lightweight dialogue editor for the Unity game engine for a game development company, Plush Pop Soft, which commissioned this thesis.

The theoretical part of this thesis first briefly discusses the history of dialogue in video games, because dialogue has been a major component in video games for decades now. While the discussion of the history relies heavily on academic studies, comparing current professional solutions for dialogue editors required resorting to the tested tools' official documentations and using the tools to gather necessary data.

The dialogue editor developed for the thesis was created as a plugin for the Unity game engine. Plush Pop Soft is a game development company focused on primarily developing mobile games using Unity. To optimize their games, they need a dialogue editor that does not affect their games' performance and is easily extendable. The practical part of the thesis discusses the Unity game engine extending its editor functionality through editor scripting. Then the thesis explains the criteria and goals for the lightweight dialogue editor solution, details the stages of programming and implementations of the lightweight editor and, finally, it evaluates whether the result is a viable alternative to the tools already available.

2 DIALOGUE IN GAMES

Dialogue is a natural way for humans to communicate with each other and many games implement dialogues in some way. While dialogue in games can occur between different human players, the focus of this thesis is on the dialogue between the player and other non-playable characters or entities within a game. The range and complexity of the dialogue varies from game to game, from simple narration to branching and reactive conversations with non-playable characters. (Brusk & Björk. 2009.)

Traditionally, dialogue in games has been turn-based and chunk-based as described by Brusk and Björk. This means that the dialogue is delivered with a set amount of lines at a time, and, until the more modern games, the dialogue was on hold until the player reacted to it somehow – by continuing the dialogue, exiting the dialogue or choosing a player dialogue option from a list. Modern games offer more natural-feeling solutions where the dialogue is delivered in real-time, the dialogue can be interrupted and continued again where the conversation was cut off without breaking the illusion of realism with careful design patterns. These solutions require much more effort from the development team and extra audio clips, so these dialogue mechanics have mainly been limited to the largest productions. There are also games that implement natural language communication or voice commands, but in this thesis, we will be studying and implementing a turn-based dialogue system that uses only text. (Brusk and Björk 2009; Ryan etc. 2016.)

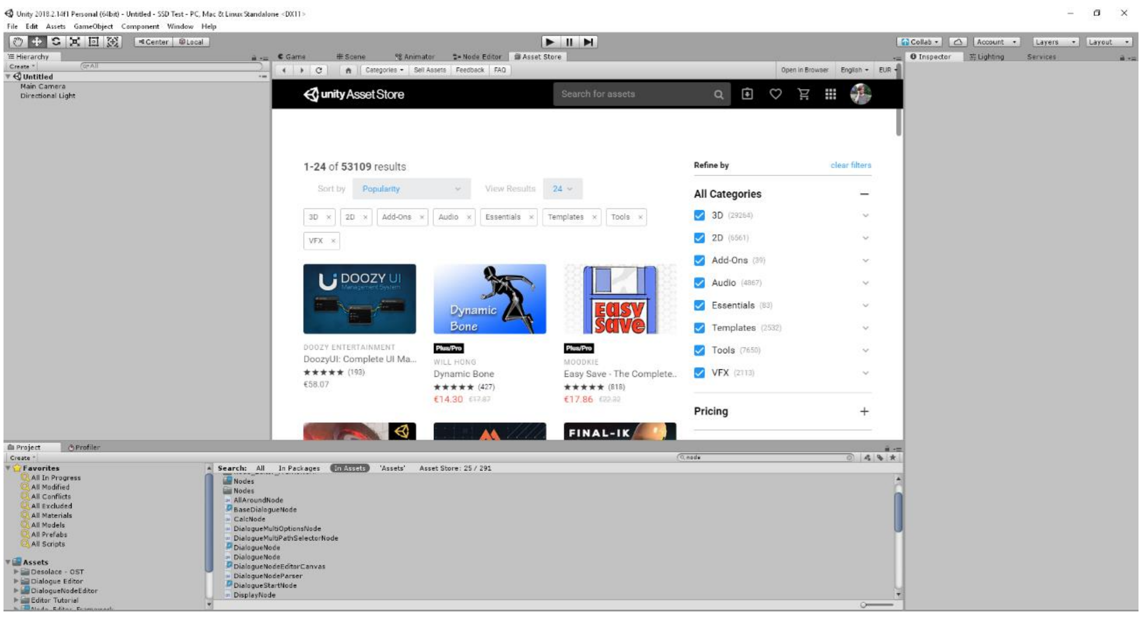
In order from simplest to most complex, four techniques for creating a dialogue system for users to interact with a computer have been defined: finite-state-based, frame-based, plan-based and finally agent-based techniques. Most games implement the finite-state-based technique for its simplicity and the linear nature of progression in games. This finite-state-based model of dialogue can be depicted as a graph of connected nodes, showing the hierarchy and transition between the different dialogue options or branches, and the dialogue will move from node to node based on the player's input. (Brusk and Björk 2009; Allen etc. 2001.)

Some games also implement the frame-based technique, which means that the computer will analyze spoken language and performs a certain action if it recognizes the speech as a valid command. It is rarely used as a dialogue tool, but rather as a voice command tool for controlling the player character or navigating the user interface. Both the finite-state-based model and the frame-based technique are simple enough that it is easy to build rather complex patterns, but they require a lot of forethought and planning for each use case. The two remaining dialogue models are too complex and require a more abstract approach for them to be currently used in game development, which still heavily relies on linearity. (Allen etc. 2001.)

3 UNITY GAME ENGINE

Unity is a powerful game engine and editor that allows game developers to create games for multiple platforms at the same time, and it fully supports the development of both 2D and 3D games (Tadres 2015, 2). Unity also has a less demanding learning curve for new developers, so it has become the most used game engine on the market. Some of the success can be credited to the extensibility of the game engine and the amount of ready-made extensions and assets for it. While on paper Unity is the most popular game engine, it is worth noting that most of the so-called triple A games use their own or other engines. (The Next Web 2016.)

Unity was first released in 2005 with the aim of “democratizing” game development with no entry fees and easy to use features. It can be argued that it has done just that with record number of independently created games being released each year; currently roughly half of all digital games (Dillet 2018). More and more people are now developing games, which has sparked criticism about Unity for creating an unsustainable situation and lowering the overall quality of games. As the number of developers has gone up, so has the markets for game-ready assets, extensions and tools, and Unity has a store for these called Unity Asset Store. Currently Unity Asset Store has over 53000 assets – 5291 of them are available for free. This store can be accessed via a browser or directly inside Unity game engine (see Picture 1). (Axon, 2016; Unity Asset Store 2019.)



Picture 1. Unity Asset Store inside the Unity Editor

3.1 Features

The most notable feature in Unity game engine is their all-in-one approach to both developer and target platforms. Unity is available for Windows, Mac and Linux, and games created on it can be targeted at mobile devices, PC and/or consoles. Both 3D and 2D games are fully supported and there are many specific development features for each. (Unity 3D 2019.)

Other notable features are a physics engine, AI pathfinding tools, monetization and advertising services, preconfigured game objects, integrated asset store and a built-in source control. The most relevant feature from this thesis' point of view is the user interface system that allows developers to create their custom tools and customize the visual appearances of those tools with Unity's own editor programming language built on C# (Tadres 2015, 3). (Unity 3D 2019.)

4 DIALOGUE EDITORS

There are sometimes multiple software or methods to edit a game's dialogue, but commonly game developers create their own dialogue editors or buy a ready-made system to use. Depending on the game engine and the game's complexity, dialogue editor tools offer a variety of extra functionality besides just writing text and linking one line of dialogue to the other. Some of these extra functionalities may include animation tools, voice acting, lip synchronization, event triggering and so on. (GDC 2016.)

Without seeing the editor or its source code, it is often impossible to tell how a dialogue system and its tools were designed and programmed. However, there are games that have been released with developer or modification tools. These tools usually allow the players to create new content or modify the existing game, and often those tools include some sort of a dialogue system. Some of the classic role-playing games are good examples of dialogue-heavy games, because their narrative and world-building rely heavily on dialogue. Two dialogue editors from games of that category, Neverwinter Night 2's Electron Toolset and Skyrim's Creation Kit, which offer players tools for creating additional content, were picked for comparison with two dialogue editor assets for the Unity game engine. (Moody 2014, 20-23.)

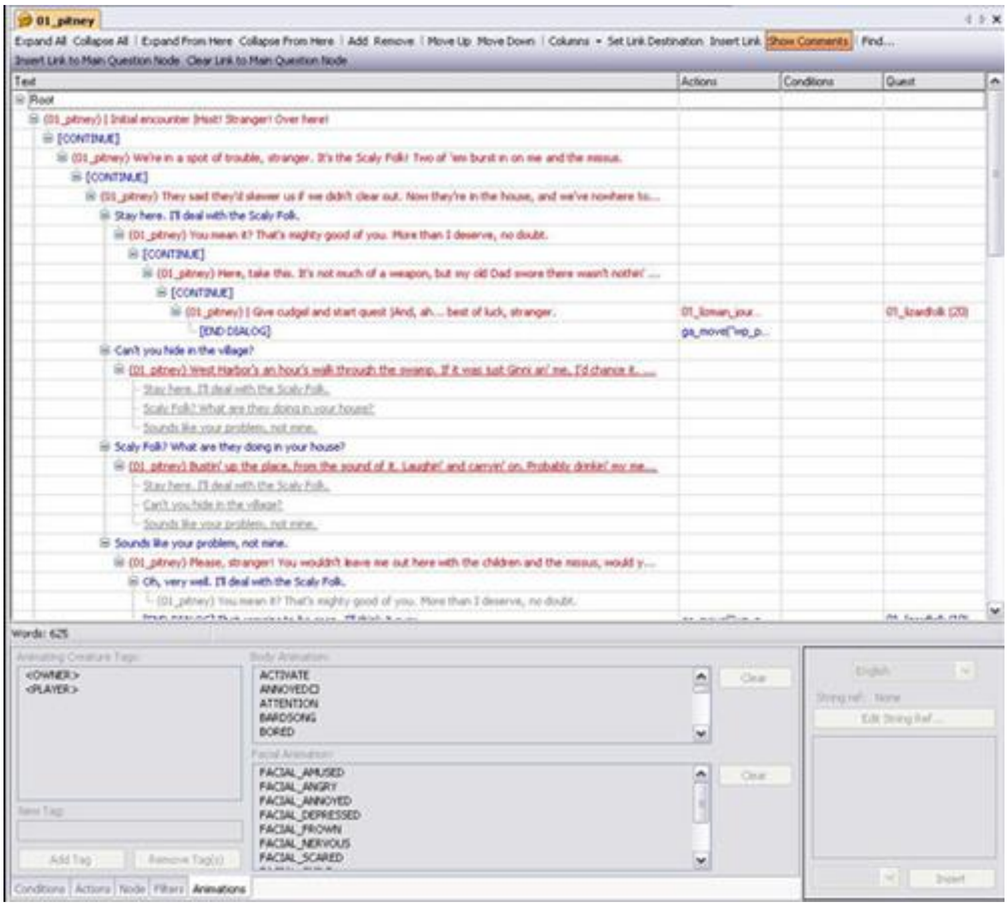
The Unity Asset Store has a large and varied collection of assets and other tools for developers to buy for their Unity projects. There are several dialogue editors already on the asset store; some are free and other editors are paid. All these dialogue editors aim to deliver ready-to-use solutions mostly for solo or indie developers, and often they are made for specific types of games in mind; for example, there are several dialogue systems that were made for visual novel types of games. (Unity Asset Store.)

The downside for most of these plugins is that they are either very hard to customize to work outside their intended style of game or they can be overly complicated to use due to trying to do accomplish a great variety of features at once. Offering a big list of features is understandable when selling a product, to make the price more appealing for the consumer. Sometimes, the abundance of features makes the plugins cumbersome to use and increases the odds for human errors. (Unity Asset Store.)

There are no academic studies or comparisons conducted about different dialogue editors in video games, so two professional dialogue editors and two among the most popular dialogue assets from Unity Asset Store were chosen for a comparison of their features and advantages and disadvantages. From Unity Asset Store Fungus and Dialogue System for Unity were chosen. Fungus is a free asset while Dialogue System for Unity is a commercial asset, which costs about 58 euros (03.2.2019). Both assets promise an easy integration with no programming skills required, and both implement a node-based editor for editing the dialogue. The professional counterparts that were chosen were the Electron Toolset from Neverwinter Nights 2 and the Creation Kit from The Elder Scrolls IV: Skyrim (The Annex 2019; Creation Kit 2019). (Unity Asset Store.)

4.1 Neverwinter Nights 2: Electron Toolset

The Electron toolset was shipped with the Windows version of Neverwinter Nights 2 video game; both were developed by Obsidian Entertainment. The Electron toolset is a tool that allows players to use the game’s assets and engine to create their own games, adventures or small modifications. The Electron toolset is based on an older toolset called Aurora, which was made for the original Neverwinter Nights game. The Aurora toolset was simple and extensive enough that it became widely popular among the game modification communities of its day. The newer toolset offers more functionality but is also much harder to use, which results in the Electron toolset not gaining quite as much popularity. Both the Aurora and the Electron toolsets had a proper dialogue editor in them, which became the first inspiration for pursuing a new dialogue editor solution for Unity. (The Annex, 2019; Hall, 2013.)



```
Root
  (01_pitney) | Initial encounter |Hoi!! Stranger! Over here!
    [CONTINUE]
      (01_pitney) We're in a spot of trouble, stranger. It's the Scaly Folk! Two of 'em burst in on me and the missus.
        [CONTINUE]
          (01_pitney) They said they'd skewer us if we didn't clear out. Now they're in the house, and we've nowhere to...
            Stay here. I'll deal with the Scaly Folk.
              (01_pitney) You mean it? That's mighty good of you. More than I deserve, no doubt.
                [CONTINUE]
                  (01_pitney) Here, take this. It's not much of a weapon, but my old Dad swore there wasn't nothin' ...
                    [CONTINUE]
                      (01_pitney) | Give cudgel and start quest |And, ah... best of luck, stranger.      01_lizman_jour...      01_lizardfolk (20)
                        [END DIALOG]                                                                   ga_move("wp_p...
            Can't you hide in the village?
              (01_pitney) West Harbor's an hour's walk through the swamp. If it was just Ginni an' me, I'd chance it. ...
                Stay here. I'll deal with the Scaly Folk.
                Scaly Folk? What are they doing in your house?
                Sounds like your problem, not mine.
            Scaly Folk? What are they doing in your house?
              (01_pitney) Bustin' up the place, from the sound of it. Laughin' and carryin' on. Probably drinkin' my me...
                Stay here. I'll deal with the Scaly Folk.
                Can't you hide in the village?
                Sounds like your problem, not mine.
            Sounds like your problem, not mine.
              (01_pitney) Please, stranger! You wouldn't leave me out here with the children and the missus, would y...
                Oh, very well. I'll deal with the Scaly Folk.
                  (01_pitney) You mean it? That's mighty good of you. More than I deserve, no doubt.
```

```
<OWNER>
<PLAYER>

ACTIVATE
ANNOYEDCI
ATTENTION
BARDSONG
BORED

FACIAL_AMUSED
FACIAL_ANGRY
FACIAL_ANNOYED
FACIAL_DEPRESSED
FACIAL_FROWN
FACIAL_NERVOUS
FACIAL_SCARED
```

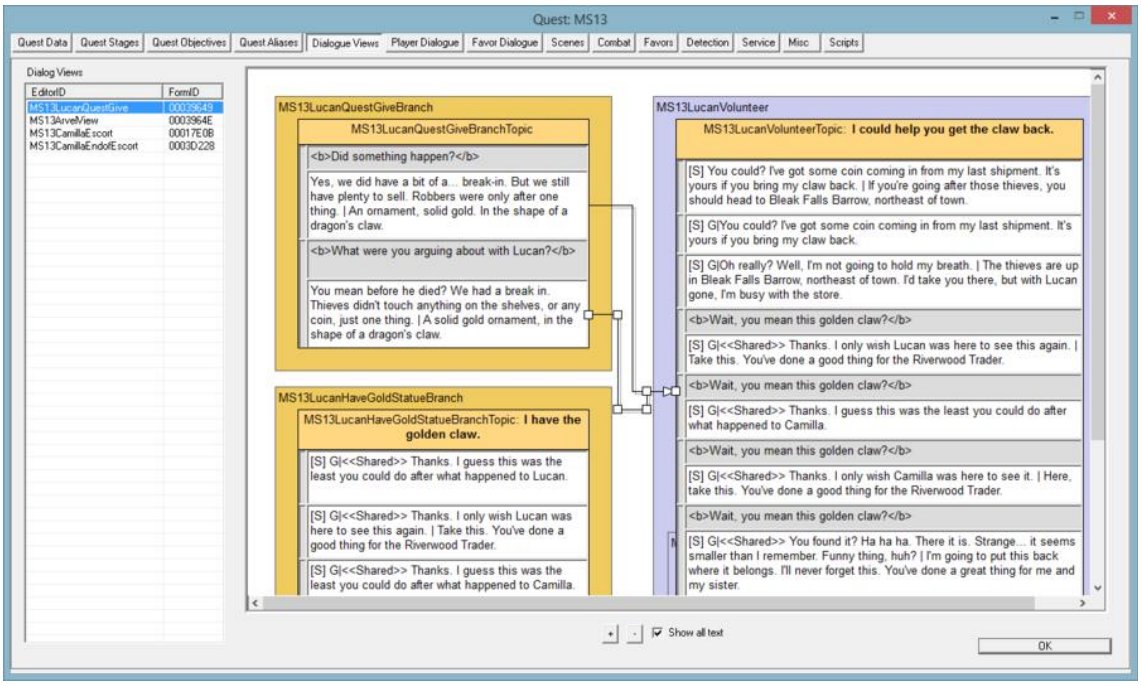
Picture 2. Electron toolset's dialogue editor (Electron Toolset)

The dialogues in Electron's toolset are stored in .tlk files and are displayed in a tree view in the editor (see Picture 2). All the spoken lines, actions, conditions etc. are displayed as coloured text; red for the non-playable characters and blue for player's choices or lines. The grey text colour marks a link to another dialogue line. The dialogue files can be imported and exported as xml-files, which allows them to be edited outside of the toolset. Each line can be assigned multiple actions, conditions and other properties such as camera angle, lip synch data and animations. The same character can speak multiple lines in a row and multiple characters can be added into the conversation. (The Annex 2019; Electron Toolset)

The strengths of the Electron toolset's dialogue editor are its ease of basic use, xml compatibility and a clutter-free editor view. The downside of the editor is that it is hard to extend to do anything outside its normal scope. Using scripts for conditions and actions requires either a deeper understanding of the Neverwinter Nights 2 game's architecture or writing custom code in Electron's own scripting language NWScript, which is based on C and Java. While the dialogue editor is not complicated, the Electron Toolset never gained similar popularity as its predecessor Aurora Toolset and as a result learning resources are scarce. However, many dialogue editor's features are nearly identical to what were already in the Aurora toolset, which has plenty of learning material. The ease of use and clean look of Electron's dialogue editor were considered as key factors when designing the Unity's dialogue editor plugin. (The Annex 2019; Electron Toolset)

4.2 The Elder Scrolls IV: Skyrim: The Creation Kit

The Creation Kit is a modification tool made for The Elder Scrolls IV: Skyrim video game developed by Bethesda Softworks. Like the Electron toolset, it's based on an older game's modification tools and it offers a complete dialogue editor solution. The Creation Kit also offers the players almost unlimited access to the game's data to modify it and create new experiences using Skyrim's assets or custom assets. The Elder Scrolls series' games have become known for their impressive scope regarding open world games and an active modification community. (Elder Scrolls Fandom, 2019; Wikipedia 2019.)



EditorID	FormID
MS13LucanQuestGive	00039649
MS13ArvelView	0003964E
MS13CamillaEscort	00017E0B
MS13CamillaEndofEscort	0003D228

MS13LucanQuestGiveBranch

MS13LucanQuestGiveBranchTopic

<b>Did something happen?</b>

Yes, we did have a bit of a... break-in. But we still have plenty to sell. Robbers were only after one thing. | An ornament, solid gold. In the shape of a dragon's claw.

<b>What were you arguing about with Lucan?</b>

You mean before he died? We had a break in. Thieves didn't touch anything on the shelves, or any coin, just one thing. | A solid gold ornament, in the shape of a dragon's claw.

MS13LucanHaveGoldStatueBranch

MS13LucanHaveGoldStatueBranchTopic: **I have the golden claw.**

[S] G|<<Shared>> Thanks. I guess this was the least you could do after what happened to Lucan.

[S] G|<<Shared>> Thanks. I only wish Lucan was here to see this again. | Take this. You've done a good thing for the Riverwood Trader.

[S] G|<<Shared>> Thanks. I guess this was the least you could do after what happened to Camilla.

MS13LucanVolunteer

MS13LucanVolunteerTopic: **I could help you get the claw back.**

[S] You could? I've got some coin coming in from my last shipment. It's yours if you bring my claw back. | If you're going after those thieves, you should head to Bleak Falls Barrow, northeast of town.

[S] G|You could? I've got some coin coming in from my last shipment. It's yours if you bring my claw back.

[S] G|Oh really? Well, I'm not going to hold my breath. | The thieves are up in Bleak Falls Barrow, northeast of town. I'd take you there, but with Lucan gone, I'm busy with the store.

<b>Wait, you mean this golden claw?</b>

[S] G|<<Shared>> Thanks. I only wish Lucan was here to see this again. | Take this. You've done a good thing for the Riverwood Trader.

<b>Wait, you mean this golden claw?</b>

[S] G|<<Shared>> Thanks. I guess this was the least you could do after what happened to Camilla.

<b>Wait, you mean this golden claw?</b>

[S] G|<<Shared>> Thanks. I only wish Camilla was here to see it. | Here, take this. You've done a good thing for the Riverwood Trader.

<b>Wait, you mean this golden claw?</b>

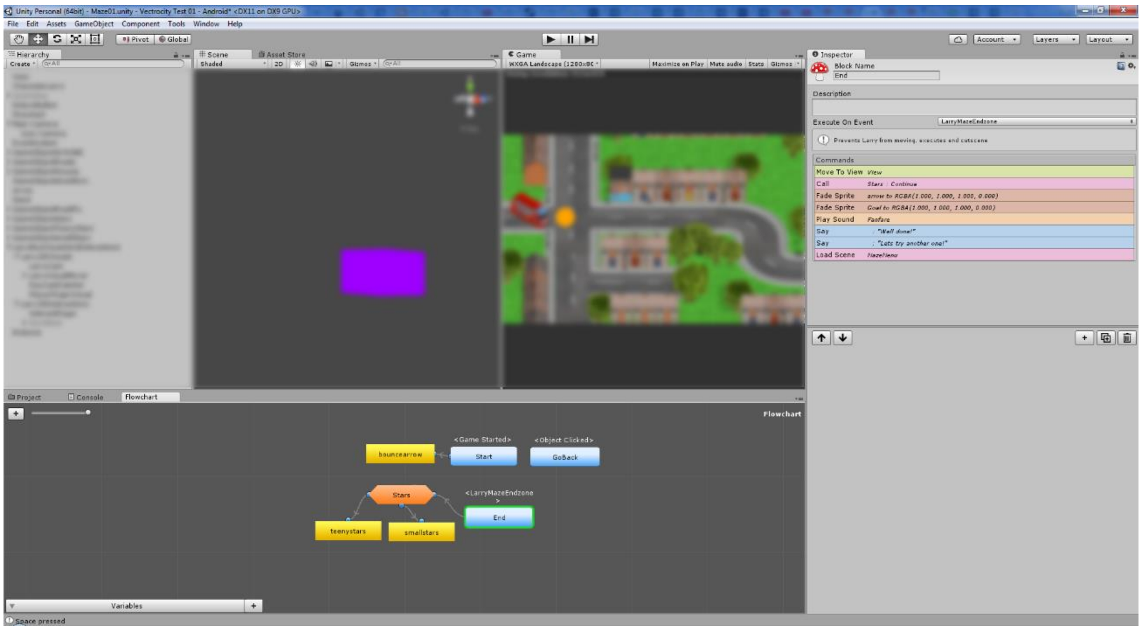
[S] G|<<Shared>> You found it? Ha ha ha. There it is. Strange... it seems smaller than I remember. Funny thing, huh? | I'm going to put this back where it belongs. I'll never forget this. You've done a great thing for me and my sister.

Picture 3. Creation Kit's dialogue editor (The Creation Kit)

In the Creation Kit, all dialogue is organized into Quests, which contain various information and data besides the actual spoken dialogue. The dialogue can be split between different branches, which can be used to better manage more complicated and lengthy dialogues by separating dialogue lines by topic (see Picture 3). Skyrim is one of the most modified games with over 60 000 mods, and with such popularity there are numerous resources for anyone who wants to learn how to use the Creation Kit. (Elder Scrolls Fandom, 2019; Creation Kit.)

4.3 Fungus

Fungus was chosen for comparison because it is a popular, free and open source plugin made for the Unity game engine. According to the official Fungus documentation, Fungus is best suited for making visual novels, RPGs, hidden object, puzzle and interactive fiction games. The dialogue editor is visualized as a node tree with different colour coded nodes for different types of dialogue nodes (see Picture 4). Fungus is a good tool for absolute beginners because it uses a visual scripting system and offers a great number of tutorials and resources. As additional features, Fungus also supports Spine animation system, localization, offers ways to control cameras and comes with some third-party extensions and plugins. (Fungus documentation 2019.)



Block Name: End

Description

Execute On Event: LarryMazeEndzone

Prevents Larry from moving, executes and cutscene

Commands

```
Move To View   View
Call           Stars : Continue
Fade Sprite    arrow to RGBA(1.000, 1.000, 1.000, 0.000)
Fade Sprite    Goal to RGBA(1.000, 1.000, 1.000, 0.000)
Play Sound     Fanfare
Say             : "Well done!"
Say             : "Lets try another one!"
Load Scene     MazeMenu
```

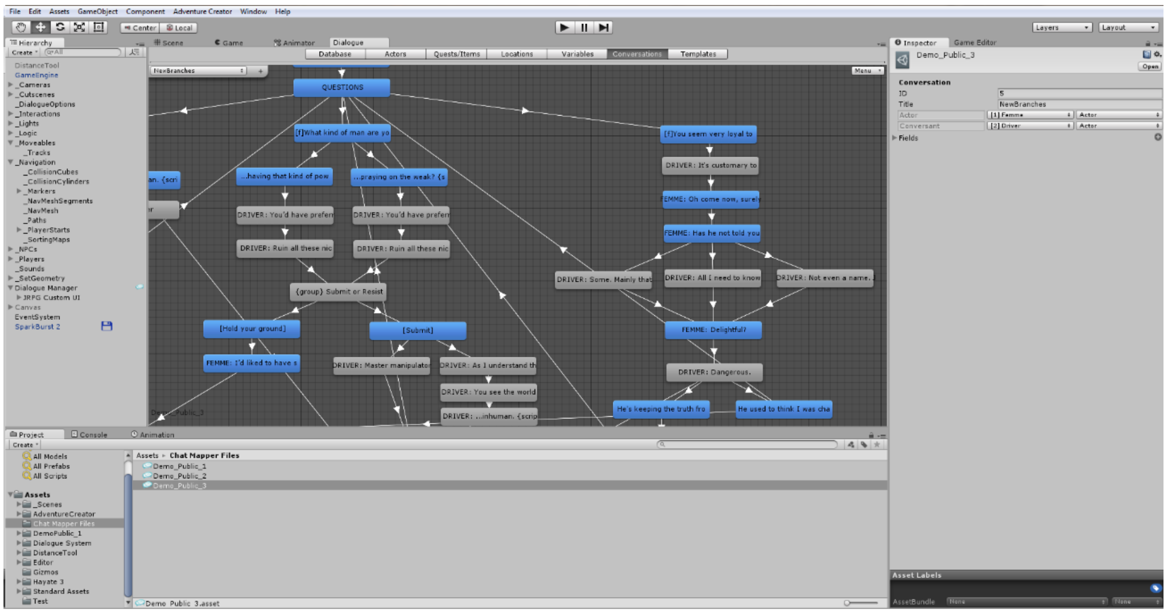
Picture 4. Fungus node-based editor and visual scripting (Fungus)

While Fungus is mainly used with visual coding, it offers more functionality for advanced programmers using supported commands with the Lua programming language. With Lua, developers gain extra control over their dialogue and can add persistence to the dialogue related data. More functionality and control can also be achieved by extending the functionality within Unity editor with custom scripts. Due to the popularity of Fungus, there are a plethora of official and community-made learning materials for developers. (Fungus documentation 2019.)

Compared with the other dialogue systems on Unity's asset store, Fungus stands out because of its simplicity, popularity and price. It offers some ready simple solutions for common dialogue construction, but using the advanced features requires knowledge of Lua or deeper understanding of the plugin to extend the editor. While Fungus is one of the more lightweight dialogue editor solutions on the Unity's own asset store, it also has some extra features and plugins integrated into it, which are often unnecessary. (Fungus documentation 2019.)

4.4 Dialogue System for Unity

Among the paid dialogue editors on Unity asset store, the Dialogue System for Unity (later referred as just Dialogue System) has for long been listed as the most popular asset. It is also the dialogue editor this thesis' client has been using so far in their game development workflow (Plush Pop Soft 2019). At the moment of writing this thesis, its price was roughly 58 euros without value added taxes. The Dialogue System has many of the same features as Fungus, such as Lua scripting, node-based dialogue editor and localization (see Picture 5). The asset has many more features over Fungus that justify the price; the editor itself has more control over dialogue flow, barks, cutscenes, quick time events and more. (Unity Asset Store 2019.)


Picture 5. Dialogue System's node-based editor (Dialogue System for Unity)

The Dialogue System is a very robust solution with custom database solution for storing different dialogues and offers multiple ways for customizing the visual output for the dialogues. According to the client, the Dialogue System has caused some crashing in the Unity game engine while developing and some features have been difficult to implement or understand (Plush Pop Soft 2019). The plugin's developer has provided an online documentation depicting different features, classes and tutorials for some of the most common elements. According to the store's user feedback, the documentation is partly incomplete and outdated, but there are channels for additional support if needed. (Unity Asset Store 2019.)

All in all, the Dialogue System is a very complex system. It is a complete dialogue system package for developers that offers ready-made solutions for more than just conversations – it includes a system for quests and a save system. The plugin is compatible with multiple other major assets on Unity's asset store. With such a long list of features, the asset's usability and stability have suffered as result, and developers who don't need the extra features might appreciate a more intuitive and easily expandable solution. (Unity Asset Store 2019.)

4.5 Comparison conclusion

There was a mutual preference with the client in favour of the text-based editors in the game modification tools over the node-based editors in the Unity Asset Store. The minimalist designs of the Electron toolset and the Creation Kit were appealing and made it easier to navigate through long dialogues, and allowed more dialogue options to be viewed at once. However, it can be argued that the node-based editors in Dialogue System for Unity and Fungus visualize the flow of dialogue better. It is also simple to separate parts of the dialogue as different groups by organizing the nodes on the screen – the Creation Kit also offers a similar functionality, but is less straightforward and intuitive.

Based on the discussions with the client, the most important features for a dialogue editor from their perspective are the visualization of the dialogue flow, adding and displaying conditions and the gameplay actions that can be triggered from dialogue lines. Lip synchronization, camera angles and animation tools are useful features, but most of the client's projects do not require these and it would be best if these features could be added as a separate plugin if there ever is a need for it.

The extra features and tools that came with the Unity plugins are not needed in all, if not most, projects. These features can at worst slow down the development and sometimes lead to unstable performance or crashing. The reason for this instability can be debatable and likely a big part of the reason is the wide range of different available versions of the Unity game engine. The two other tested dialogue editors were specifically designed to work inside the current version of the modification tools, and all the users should be using the same version.

Because the two dialogue editors in the modification tools were designed for writing dialogue for a single game, it is no small wonder that they are more efficient for the purpose they were created for. They have all the necessary features that are needed for the target game and nothing unnecessary has been included. The dialogue editors for Unity game engine were designed to be more general in nature so that they can be used in multiple ways in a wide range of game genres and types. It is this universal dialogue editor approach that makes these tools unwieldy and difficult to use.

This conclusion is compelling enough of a reason for creating a simple dialogue editor with only the essential features. This means that the resulting editor is not likely a production-ready tool for any project and the developers will need to add the missing features on a case-by-case basis. Additional time for creating or adding previously created features is perhaps not the most optimal solution, but if a dialogue editor is too specific, it will limit the way it can be implemented. If the dialogue editor is made too universal, it will ultimately run into the problem of having unnecessary or obsolete features, and it might lead to human errors, slower development time and unstable performance.

5 CREATING A DIALOGUE EDITOR FOR UNITY

Creating a dialogue editor plugin, which is basically an extension, for the Unity game engine requires a different method of programming than the average Unity game programming, and it is called editor scripting. Editor scripts will need to be placed under a folder called Editor, which will mark them as editor scripts for Unity and they will not be included in the final build of the game. Unity has a great range of built-in components for editor scripting to make getting started easier and with time and dedication, a skilled programmer can create just about anything within Unity.

5.1 Initial goals, requirements and limitations

The dialogue editor prototype was commissioned by Plush Pop Soft and it was to be an internal tool only, so the editor's features were designed to suit the company's needs. The main goal defined for the new dialogue editor was it to have similar basic functions as the existing dialogue editors on the Unity Asset Store. The editor should be simple to use and work reliably without compromising the stability of the game engine. This means the visual look of the editor should be minimalistic and the Electron toolset's text-based editor for Neverwinter Nights 2 was chosen to be the main inspiration. The simplicity and minimalistic nature would optimally lead to faster development times and improved workflows.

In the beginning of the design phase, the specific requirements of the custom dialogue editor were discussed and agreed with the client company, Plush Pop Soft. The work was agreed to be done in an iterative manner to allow necessary changes to be made according to the feedback and any possible problems or bugs. The initially listed required features in order of their importance were as follows:

1. Ability to create branching dialogues inside the Unity game engine
2. Setting conditions on individual dialogue lines
3. Setting different premade actions to be called on dialogue lines
4. Setting custom actions on dialogue lines
5. Have the editor be easily extendable
6. A clean interface
7. A demo of the functionality

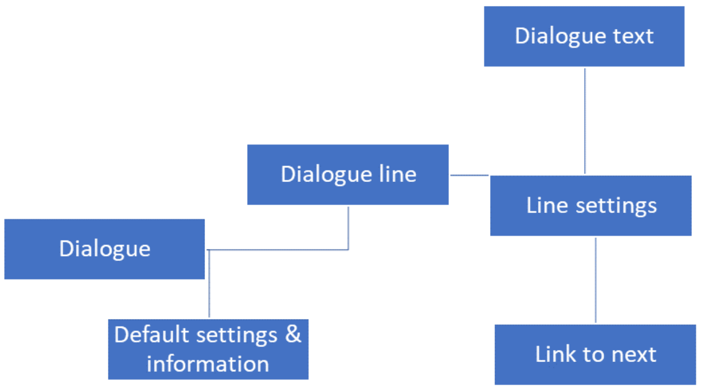
The agreed minimum requirement for the editor was the ability to create and save branching dialogues and an example scene with a script that would demonstrate how to assign dialogues to NPCs and how to progress in the assigned dialogue in a game scenario. It was debated whether the finished dialogues' save format should allow editing outside the Unity game engine, but since the editor was to be seamlessly integrated into the company's Unity workflow, this feature was not considered essential.

The only limitations for the project were the deadline from the company and what could be done in Unity game engine in such time. Because the client company had a quick need for a new editor, the deadline was set to be only a few months. This limit meant that the prototype version of the editor could not implement a very complex custom data structures or database solution because either of those would be a large undertaking worthy of a thesis in and of themselves.

5.2 Planning the architecture

From the very start, there was an idea of having two sets of systems or configurations in a single dialogue: one set for the whole dialogue and another set that can be altered for each dialogue node (see Picture 6). The configurations for the whole dialogue would contain general information such as the title of the conversation, an optional description for it, default values for the listener and the speaker, and text speed. The configurations for individual lines would contain modifiable information about the current speaker and listener, the dialogue text, conditions, actions and the links to the next dialogue lines.

The Unity game engine has two easy ways to create a container for the data and classes the dialogue files would need. Every object that appears or is used in a game in Unity is called a game object, and every game object can be made into a prefab. The prefab is a blueprint of the object and it will remember all the settings of the game object at the time of its creation, but it can also be modified later. Another way to store data is using a scriptable object, which is a custom class created by Unity. Scriptable objects can be turned into files quickly and can hold a variety of data types in themselves. Scriptable objects take up less memory, because they do not have all the extra components that always come packed with a game object or a prefab. Considering the optimization of a more complex game, scriptable objects were chosen as the initial data format for storing the dialogue data.



Picture 6: Simplified version of the planned data structure

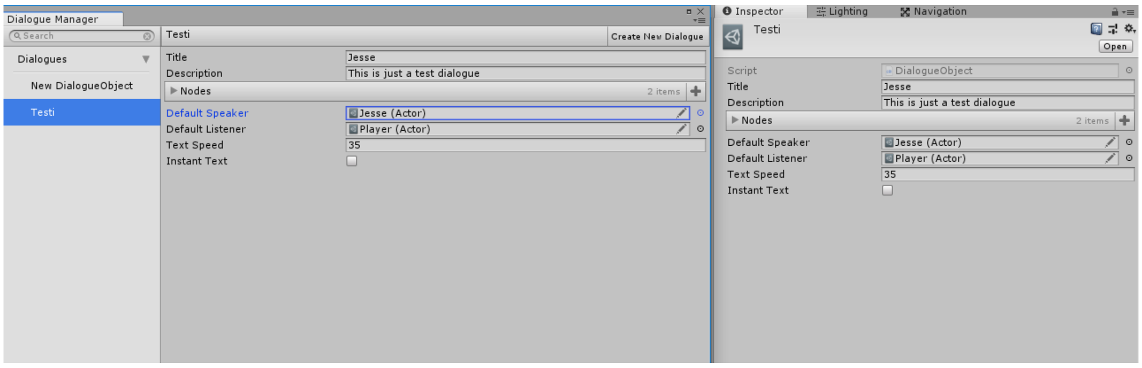
One single dialogue could contain hundreds of lines of dialogue, or a lot more, so the dialogue data storage should be well optimized to ensure a smooth performance. However, the implementation of databases and other complex data structures were ruled out in the beginning, so for the purpose of this thesis, data was to be stored in a simple C# list. Arrays were considered briefly but working with lists in this instance was easier because no custom algorithms for sorting and adding elements would be needed.

The editor itself would be a separate window inside the Unity, which would display a list of all the dialogues in the current project and allow editing them. The window would be split into three separate parts; on the left would be a list of the dialogues, on the top would be the settings for the dialogue or the currently selected line, and on the bottom would be a list of the dialogue lines. To show the flow of the dialogue, the Electron toolset's approach was thought to be the clearest: the child dialogue lines would be indented more compared to their parent line. Should the dialogue line link to another branch within the dialogue, it should show the linked dialogue text below it and have a different highlight colour. These plans were discussed with Plush Pop Soft Ltd and they were approved for production.

From the start, controlling the flow of the conversation with conditions was one of the key features requested. To make sure that the condition system would be easily modifiable or replaceable, it was planned to work as a separate system. The dialogue editor should not have to be modified even if new types of conditions were added later in the development cycle. For the purpose of this prototype tool, only a few types of conditions, and the actions to change them through dialogue, would be added.

5.3 Text-based editor

There are numerous ways to extend the editor functionality and data visualization, but it is implemented in either the Unity game engine's sidebar or as a separate window (see Picture 7). The chosen method was to use a window, which would allow more complex UI layouts.



Testi

Title	Jesse
Description	This is just a test dialogue
Nodes	2 items
Default Speaker	Jesse (Actor)
Default Listener	Player (Actor)
Text Speed	35
Instant Text	☐

Testi

Script	DialogueObject
Title	Jesse
Description	This is just a test dialogue
Nodes	2 items
Default Speaker	Jesse (Actor)
Default Listener	Player (Actor)
Text Speed	35
Instant Text	☐

Picture 7. First dialogue editor draft as a window (left) and as a side panel (right).

Quickly after starting the editor programming it was apparent that editor programming is vastly different from regular game programming in the Unity game engine. Unity provides a handful of tutorials and resources to study editor programming, and there are some user-made tutorials as well, but the amount of resources is relatively small and a lot less comprehensive compared to the regular Unity programming. This made starting out harder than what was anticipated and creating a simple list of clickable texts required a lot of forethought and planning to execute. The syntax for creating the editor layout or functionality is not something a developer would encounter when only programming the gameplay functionality.

More problems were encountered when trying to display the individual dialogue lines and their different variables. It is very straightforward to natively display common data types inside a custom class but displaying the information of the custom data types inside another custom data type is exponentially more complex. This would have required a better knowledge of the editor scripting to anticipate and plan accordingly.

To display all the dialogue data in a clear text format, creating multiple temporary lists to hold all the data from different dialogue lines was considered, but it would have resulted in time-consuming solutions regarding adding or deleting lines or reorganizing the data. In hindsight creating a management system that would pack multiple lists of data into a list of dialogue line data would have been the simplest solution, but programming the packing and unpacking, organizing, and the displaying options along with the other main dialogue editor features would still have resulted in too big of a project for this thesis. Ultimately, the text-based editor idea was abandoned in favour of searching for a more time-efficient solution.

5.4 Open source node canvas framework

Together with Plush Pop Soft it was decided that a quicker solution would be needed to get a working editor ready in a reasonable amount of time. Unity also has a capability of displaying data graphically as a series of connected nodes, which was used in the previously tested Fungus and Dialogue System assets. With no prior knowledge on this topic, and the lack of proper programming documentation, creating one from scratch did not seem feasible or reasonable. It was also noted that there was previously a thesis written about node-based editors in Unity that only focused on creating the node view and system. In the end, an open source solution called Node Editor Framework for Unity (later referred to as the Node Framework) was chosen to become the dialogue editor's framework.

The reason the Node Framework was chosen was because it is not only extendible but also features a ready saving system for all the data that is stored in the nodes. This display and saving system meant that there would be no need for packing and unpacking the dialogue line data container. The framework did not have an extensible documentation, but the source code was properly commented and had plenty of example nodes to study. In the framework's online code repository, there were also two example projects: a texture composer and a dialogue system. The dialogue system was briefly considered as a base for this prototype, but its execution and features were too far from this thesis' design.

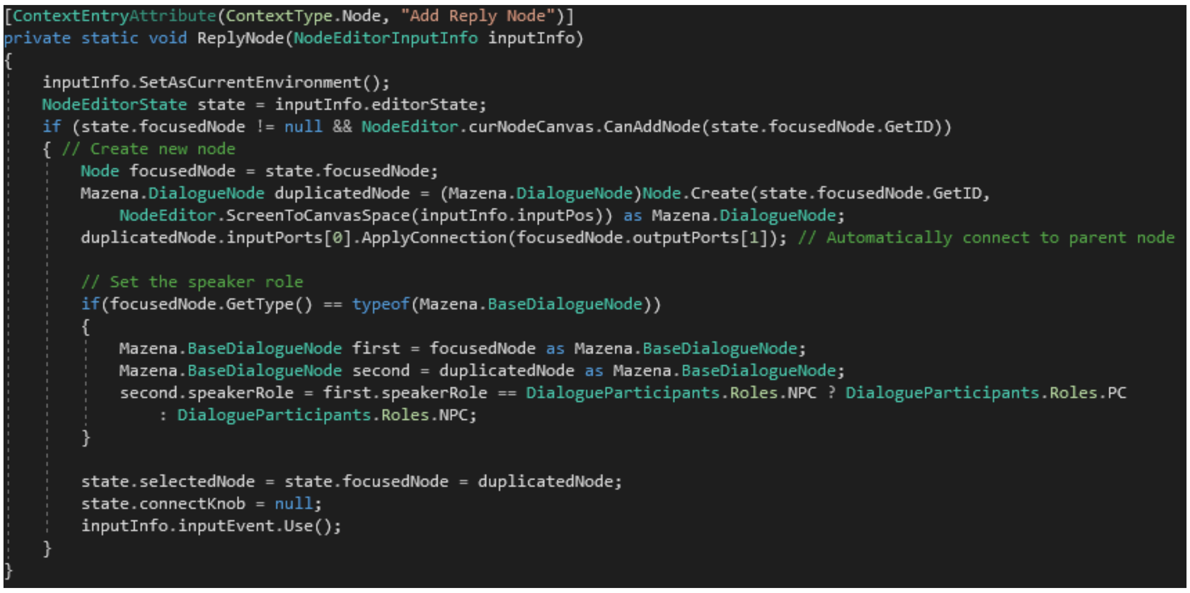
Most of the Node Framework's features and all the node types were removed to create a clean base to work on and to keep the editor as simple and clean as possible. The functionality that was left intact was the ability to create a node, attach them to other nodes and the ability to save the node canvas, which is the area where all the nodes for one conversation are stored.

5.5 Creating the core features

With the lessons learned from comparing the different dialogue editors and after performing some initial tests with the Node Framework with a custom node with a text field and other variables it became clear that the node editor's view should be as simplified as possible to increase the readability of the dialogue. Two different solutions were considered to minimize the space needed for the nodes to reduce clutter. In the first solution, the dialogue nodes would only have a small snippet of the dialogue line's text with the full dialogue line information visible on the side panel in the Unity game engine. The second solution also had a minimized view for the dialogue nodes. In the minimized view, only the vital information about each line is visible, but the nodes would expand upon selection to display all the available data. The second solution was chosen with an agreement with the client as a better fit to keep the whole dialogue workflow within one window.

To be able to tell different dialogue branches apart easily, two different kinds of nodes were created. The first line of dialogue for all dialogue branches is stored in a dialogue node that will not change in size – this will make it clear that it is one of the root nodes. The rest of the dialogue nodes will minimize when not selected. To create a clear distinction between the NPC and player dialogue lines, the nodes are colour coded as red and blue for NPCs and player respectively. Each dialogue node has a slot for a picture that can be used to differentiate the dialogue nodes in the editor, or it can be used as an avatar for the speaker in the game.

The default method of adding dialogue nodes in the Node Framework is to right click on the canvas and choose which type of node you want to add. This node then must be manually connected to other nodes to establish the correct flow. The possibility of large number of nodes in a single conversation required that this workflow had to be optimized by adding both an Add Reply button on any selected dialogue node and the ability right click a node to create a reply, duplicate or delete it. This new reply method will automatically connect the newly created node to the original node, place it on the right side of it, and choose the speaker's role to be the opposite of the original node (see Picture 8). This functionality's implementation was not perfect, as using the button instead of right click would sometimes place the new dialogue node in an undesired position, sometimes off the viewport of the canvas. This issue was moved to the development backlog, as other features were considered more important for the overall project.



```
[ContextEntryAttribute(ContextType.Node, "Add Reply Node")]
private static void ReplyNode(NodeEditorInputInfo inputInfo)
{
    inputInfo.SetAsCurrentEnvironment();
    NodeEditorState state = inputInfo.editorState;
    if (state.focusedNode != null && NodeEditor.curNodeCanvas.CanAddNode(state.focusedNode.GetID))
    { // Create new node
        Node focusedNode = state.focusedNode;
        Mazena.DialogueNode duplicatedNode = (Mazena.DialogueNode)Node.Create(state.focusedNode.GetID,
            NodeEditor.ScreenToCanvasSpace(inputInfo.inputPos)) as Mazena.DialogueNode;
        duplicatedNode.inputPorts[0].ApplyConnection(focusedNode.outputPorts[1]); // Automatically connect to parent node

        // Set the speaker role
        if(focusedNode.GetType() == typeof(Mazena.BaseDialogueNode))
        {
            Mazena.BaseDialogueNode first = focusedNode as Mazena.BaseDialogueNode;
            Mazena.BaseDialogueNode second = duplicatedNode as Mazena.BaseDialogueNode;
            second.speakerRole = first.speakerRole == DialogueParticipants.Roles.NPC ? DialogueParticipants.Roles.PC
                : DialogueParticipants.Roles.NPC;
        }

        state.selectedNode = state.focusedNode = duplicatedNode;
        state.connectKnob = null;
        inputInfo.inputEvent.Use();
    }
}
```

Picture 8. The programming logic for adding a reply node.

The condition system for controlling the dialogue flow was created to manage checking whether conditions are met and for changing the condition values. Conditions were stored in a custom data class named Condition, and it would initially hold four types of conditions: integers (whole numbers), floats (fractions), strings (text) and Boolean values (true or false). The Condition data will also hold a name for the condition, a comparison type and the expected value. The comparison types are:

- Equal (=)
- Not equal (!=)
- Smaller/larger than (< / >)
- Smaller/larger or equal (<= / >=)

The condition and comparison type are stored as enumerations, which will enable them to be displayed as a dropdown list in the dialogue editor. The name and value are entered as text into text fields and will be converted to the appropriate data format. All the data type conversions and actual condition logic will happen within the condition system and it will simply tell the dialogue manager whether the condition is true or not.

A system for controlling dialogue-triggerable actions was created in a similar fashion with the action type stored as an enumeration to enable the dropdown menu selection. The parameters for the actions were also inputted as strings and then converted into a suitable format in runtime. For the condition and action system, this is not an optimal solution because if the user inputs data in the wrong format, it could cause unwanted behaviour or errors, unless properly validated. The validation process should work in the editor as the developer is entering the commands but writing a fool-proof validation for all the possibilities is too time-consuming for the scope of this thesis.

A dialogue manager is required to implement the control and the flow of the dialogue. When the dialogue is started, the dialogue manager gathers all the root nodes and orders them by their ID number, and then sends the node data from the lowest number node that either does not have a condition, or has all its conditions met, to a UI component. The information about the connected nodes are also sent to the UI component. On the UI, if the dialogue nodes belong to the player, the available dialogue lines will be presented as buttons. If the next line ends the conversation or does not belong to the player, a prompt to click or tap to continue is displayed.

The dialogue manager, like the conditions, was created as a separate system, but is more tightly coupled with the dialogue editor, and changing the structure of one in the future might require changing the other as well. The dialogue manager is responsible for returning the correct line of dialogue for the user interface of the game, and it will know how to navigate through the dialogue based on the input from the interface. When the dialogue manager returns a text from a node with an action, it will also tell the action manager to perform it. A demo version of an interface solution was created for the demo scene.

5.6 Creating the demo scene

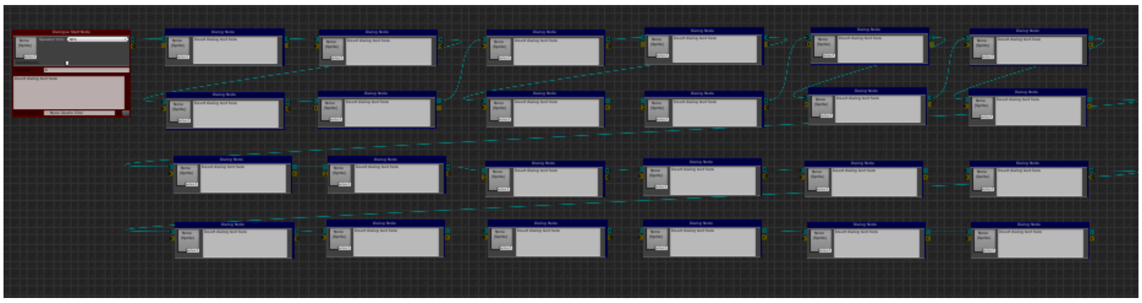
To demonstrate how to integrate the dialogue manager into Unity projects, the demo scene has a simple UI that holds both the dialogue manager, the condition system and a dialogue displayer script. In the demo scene, the dialogue displayer script is responsible for starting the dialogue and allows the user to advance in the dialogue. A simple typewriter effect with adjustable text speed and audio was added to visualize a common way of presenting dialogue text in games.

Writing and testing the condition system alongside with the dialogue manager took longer than expected and the demo scene ended up missing some of the features that were planned for it. The avatars are currently not used in the demo scene at all, and the UI’s visual design is very basic. To emphasize the game feel of the dialogue, the demo was supposed to include a small explorable 2D or 3D environment with characters to talk to. This was not considered a vital part of the demo and it was replaced by simple on-screen instructions on how to start the dialogue.

5.7 Testing

With addition to personal testing, the finished prototype of the dialogue editor with the demo scene was delivered to the client for testing. Because the editor was still just a prototype, the editor was used in a mock-up project by the client for purely testing purposes. In this project, the client tried to simulate a real workflow in creating dialogue in the game development. The developer, who had been previously in charge of implementing the dialogue to the client’s games, performed the testing with the help of the demo scene and brief verbal instructions.

In the mock-up projects, three different dialogues were created: one where dialogue’s flow was decided by a series of conditions, and one where the flow was decided by player input. The last type of dialogue created was linear but lengthier to test the performance and file size (see Picture 9). Testing the different dialogues was done in the Unity editor, on PC and on a tablet and a phone, both running on Android operation system.



Picture 9. Multiple dialogue nodes for testing performance and file size.

It took only a moment for the developer to get used to the text area being in the canvas in compared to the Dialogue System for Unity asset where the text area was located on the side panel. This feature sped up the writing part of the dialogue according to the developer, as did the automatic speaker role switching on adding a reply node. The condition and action systems were also easy to use but required a lot of conscious effort from the user to not accidentally enter invalid data type into the text fields.

The biggest flaws and areas where the dialogue editor was found lacking, were related to the interface and the controls. Organizing the dialogue nodes was tedious and the nodes could not be dragged in their minimized state, which required the user to click on them once, wait a few milliseconds for the data to load, and then the node is interactable. Currently the editor does not support selecting and moving multiple nodes at the same time, which becomes a problem when organizing larger dialogues.

The developer testing the editor conceded that the minimized nodes are a good idea, but even the minimized node takes a bit too much space and forces the user to pan the view or zoom out to navigate the dialogue. The placement of the buttons and text fields also got some criticism. It was apparent that the impact of the lack of polishing the user interface was greater than anticipated and the user experience solutions should be designed and planned more carefully in the future.

When the ready dialogues were tested on different platforms, no significant performance issues or bugs were discovered. As long as the editor values for the conditions and actions were set correctly, the dialogue behaved as expected. The major drawback that was noticed in the testing phase, was the increase in file size as the dialogue grows larger. The first two shorter branching dialogues were between 17 and 21 kilobytes. The longest test dialogue ended up being 75 kilobytes in size with still only 25 lines of dialogue. It was concluded that the Node Framework’s format for saving data is not optimized for dialogues, because story-driven and role-playing games can have enough dialogue to fill multiple 300-page novels, and the file size would soon become an issue especially when developing games for mobile devices.

5.8 Review

The finished dialogue editor is a functional editor with the capabilities to create branching dialogue that can be displayed on any game’s user interface solution. The user can add a set of premade conditions and actions to the dialogue lines, but the implementation is still partly unfinished because the user’s data is not validated in any way.

Due to time constrains, the feature to assign methods as custom actions in the editor was not implemented. Currently, to add new actions, they need to be written into the action system, which cannot be recommended as a permanent solution. The performance of the dialogue editor and manager was good on all tested platforms, but the Node Framework’s data format is very memory-heavy and is not suited for creating multiple lengthy dialogues.

The design of the editor is a far cry from the original design; the text-based editor had to be substituted with a node-based solution. Majority of the client’s editor issues were related to the usability and user experience, which play a major role in the development cycle – a clunky and uncomfortable interface can heavily hinder the writing process. The switch to the Node Framework presented a wildly different and unexpected design problems compared to a text-based approach. The branching dialogue and other systems function as was planned, but ultimately the interface solutions render the prototype currently unviable as a main dialogue writing tool for the client.

6 CONCLUSION

The goal of this thesis was to create a lightweight dialogue editor for the Unity game engine for a game development company, Plush Pop Soft. The editor was planned to be intuitive to use and easily extendable, so that additional features can be added depending on the needs for each game or project. While all parties came to an agreement on that a simplified text-based editor was the ideal solution, the final editor was a graphical node-based dialogue editor, which was built on an open source Node Editor Framework for Unity.

The new editor can create both simple and complex branching dialogues with different conditions through a condition system that stores different types of variables. Currently the dialogue system supports simple actions to be executed during the dialogue, but it can be extended further with the use of delegates and unity actions. No notable bugs were discovered during the testing of the editor, but some functionality and the user interface were left unpolished and are perhaps frustrating enough to discourage the use of the editor as the main tool for creating dialogue.

All in all, the new dialogue editor plugin is a working prototype, but in no means a finished product. Unity offers different tools for creating plugins for different purposes, and a skilled programmer could create just about anything in it, but the documentation and resources for this type of programming are sparse. There was not enough time to properly research and study the editor scripting and serialization of objects, and these are large enough topics to perhaps warrant a thesis of their own. For future research, other useful topics related to dialogue editors would be creating and using databases in the Unity game engine.

Dialogue is an important part of many video games and with the popularity of mobile games, there is a need for lightweight dialogue editors and dialogue managers. While the resulting prototype is not a completely satisfying solution, the challenges and problems that emerged during the development can hopefully be insightful for anyone considering creating a dialogue editor tool for the Unity game engine.

The prototype has most of the wanted features, but the user interface and file size issues are too big to ignore. The biggest lesson learned from writing this thesis, is how complex and difficult it is to create something that feels effortless to use. Plans for the next dialogue editor are already in place, which will be a truly text-based editor with a cleaner data structure. In addition to the mistakes and problems faced in the creation of this prototype, the research into four different dialogue editors will greatly help when designing the next editor plugin.

REFERENCES

Allen J.; Byron D.; Dzikovska M.; Ferguson G.; Galescu L. & Stent A. 2001. Toward Conversational Human-Computer Interaction. AI Magazine Winter 2001. Vol. 4, 27-38.

Annex. 2019. Electron toolset. Consulted 2.4.2019 https://annex.fandom.com/wiki/Electron_toolset.

Axon Samuel. 2016. Unity at 10: For better – or worse – game development has never been easier. Consulted at 12.5.2019 https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier/.

Brush J. & Björk S. 2009. Gameplay design patterns for game dialogues. DiGRA '09 - Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory. Vol. 5. Available at http://www.digra.org/wp-content/uploads/digital-library/09287.59480.pdf.

Creation Kit. 2019. The Elder Scrolls Wiki. Consulted 3.4.2019 https://elderscrolls.fandom.com/wiki/Creation_Kit.

Dialogue Systems in Double Fine Games. 2016. GDC. Available at https://www.youtube.com/watch?v=0hMiPBe_VRc.

Dillet R. 2018. Unity CEO says half of all games are built on Unity. Consulted 12.5.2019 https://techcrunch.com/2018/09/05/unity-ceo-says-half-of-all-games-are-built-on-unity/.

Fungus. 2018. Fungus documentation. Consulted 3.4.2019 http://fungusdocs.snozbot.com/index.html.

Hall B. 2013. NWN2 Toolset Guide Volume I. Consulted 23.4.2019 https://neverwintervault.org/sites/neverwintervault.org/files/project/726/files/nwn2_toolset_guide_i.pdf.

Mod (video gaming). 2019. Wikipedia. Consulted 1.5.2019 https://en.wikipedia.org/wiki/Mod_(video_gaming).

Moody K. 2014. Modders: changing the game through user-generated content and online communities. University of Iowa.

Non-Player Character. 2019. Wikipedia. Consulted 1.5.2019 https://en.wikipedia.org/wiki/Non-player_character.

Plug-in (Computing). 2019. Wikipedia. Consulted 1.5.2019 https://en.wikipedia.org/wiki/Plug-in_(computing).

Role-playing game. 2019. Wikipedia. Consulted 13.6.2019 https://en.wikipedia.org/wiki/Role-playing_game.

Ryan J.; Mateas M. & Wardrip-Fruin N. 2016. A lightweight videogame dialogue manager. DiGRA/FDG '16 - Proceedings of the First International Joint Conference of DiGRA and FDG. Available at http://www.digra.org/wp-content/uploads/digital-library/paper_439.pdf.

Slabinski M. 2013. 8 key principles of writing effective game dialogue. Consulted 1.5.2019 https://www.gamasutra.com/blogs/MarkSlabinski/20130313/188389/8_Key_Principles_of_Writing_Effective_Game_Dialogue.php

Tadres A. 2015. Extending Unity with editor scripting.

This engine is dominating the gaming industry right now. 2016. The next web. Available at https://thenextweb.com/gaming/2016/03/24/engine-dominating-gaming-industry-right-now/.

Unity. Unity Asset Store Consulted 15.5.2019 https://assetstore.unity.com/.

User interface. 2019. Wikipedia. Consulted 13.6.2019 https://en.wikipedia.org/wiki/User_interface.