Bachelor's thesis

Information and Communications Technology

Game Technology

2019

Jarkko Kaunela

A LOCALIZATION SYSTEM FOR GUIDER GAME

– Localization's role in game design

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology | Game Technology

2019 | 34

Jarkko Kaunela

A LOCALIZATION SYSTEM FOR GUIDER GAME

- localization's role in game development

This thesis goes through the process of finding and applying a localization method to a Finnish mobile serious game, called Guider Game ? aimed for immigrants, and created with Unity. The game was required to support Finnish, Modern Arabic and English texts.

The primary goal of this work was thus to find or create a suitable method to implement localization with for the Guider Game. To aid that goal, a detailed developer manual was also required to explain how to setup and use the new system.

The key problems in this stage of the project were enabling Modern Arabic character support, replacing all the hardcoded texts with a system that uses an ID-based method to support more languages, which could be easily added when the system is in place.

The suitable methods for both the Arabic support and the translation system were surprisingly quick to find. This was mostly due to a good timing of the fortunate events such as: open sourcing of an Arabic support plugin for Unity; finding another open sourced project that handled translations for Unity in a suitable manner and it supported the forementioned Arabic support plugin and also the text components that were already used in the game.

The implementation stage proceeded as planned. A major task was tagging all the game text with an ID, and placing that text to the Google Sheet. Another time-consuming task was documenting all the required changes and instructions to the various project-files for the other developers into a new developer manual.

In conclusion, planning and implementing a localization system before the release of the game was the right choice. If implemented long after the project launch, localization may require restructuring many systems controlling what the user can view on the screen. Therefore, it is important to plan for a localization system pre-emptively to avoid any extra work in the future. It is also adviced to give translators enough time and context to improve the quality of their work.

KEYWORDS:

Serious game, Unity, Localization, Culturalization, Hardcoding

OPINNÄYTETYÖ AMK | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka | Peliteknologia

2019 | 34

Jarkko Kaunela

LOKALISOINTIJÄRJESTELMÄ GUIDER GAME -HANKKEESEEN

- lokalisoinnin rooli pelikehityksessä

Tässä opinnäytetyössä käytiin lävitse lokalisointijärjestelmän haku-, kehitys-, ja käyttöönottoprosessia suomalaiseen opetuspeliin. Peli kehitettiin Unity-ohjelmistolla mobiililaitteille. Pelin kohderyhmä on nuoret maahanmuuttajat, joten pelin lokalisointijärjestelmän oli tuettava tekstejä suomen lisäksi myös arabiaksi ja englanniksi.

Työn päätarkoitus oli siten löytää tai kehittää Guider Game -peliprojektille sopiva lokalisointimenetelmä implementoitavaksi sekä tehdä työskentelyopas muille tämän projektin kehittäjille lokalisointijärjestelmän käyttöönottoa sekä tulevaa jatkokehitystä varten.

Avainongelmia tässä työssä olivat arabiankielen vaatiman merkkituen toteutus Unity-ohjelmistolle sekä kaiken kovakoodatun suomenkielen korvaaminen dynaamisella koodilla käyttäen ID-tunnisteita, jotta pelitekstit on mahdollista vaihtaa sujuvasti kielestä toiseen.

Sopivien ratkaisujen löytäminen ja toteuttaminen sekä arabian merkkituelle että käännösjärjestelmälle oli yllättävän nopeaa. Tämä johtui pääosin siitä, että molempiin ongelmiin oli vastikään julkaistu avoimen lähdekoodin lisenssin alaiset ratkaisut. Näihin ratkaisuihin tutustumalla, pienellä modifioinnilla ja itsetehdyillä lisätoiminnoilla sai aikaan käyttötarpeisiin sopivan järjestelmän.

Toteutusvaiheen työstä puolet oli pelitekstien lisäämistä käännöstaulukkoon ja keksiä niille sopivat tunnisteet. Toinen puolikas työstä oli erilaisten muutosvaatimusten yksityiskohtaista dokumentointia kehittäjäoppaaseen, jotta uusi lokalisointijärjestelmä olisi mahdollista ottaa käyttöön.

Lokalisointijärjestelmän suunnittelu ja toteutus selvästi ennen pelin julkaisuajankohtaa oli ehdottomasti hyvä ratkaisu. Sekä teoria, että käytäntö osoittivat lokalisoinnin vaativan runsaasti työtä. Lokalisointia varten ohjelmistorakenteen on täytettävä tietyt kriteerit, tai se on tehtävä lähes kokonaan uudestaan, mikä vie aikaa. Myös käännöksiin on varattava runsaasti aikaa, jotta kääntäjät voivat panostaa laatuun.

ASIASANAT:

Opetuspeli, Unity, Lokalisointi, Kulturisaatio, Kovakoodaus

CONTENTS

FIGURES

LIST OF ABBREVIATIONS

2D	Two-dimensional, a perspective related to graphics
3D	Three-dimensional, a perspective related to graphics
Git	A name of a popular version control system for tech-oriented projects
JSON	JavaScript Object Notation, a data file format
ID	Identification
MMO	Massively multiplayer online, usually used to reference a type of a video game
RPG	Role-playing game
Script	A data file or a section of a data file containing source code written in a programming language to perform a function or task
STRING	A variable type in many computer programming languages
UI	User interface

1 INTRODUCTION

This thesis was carried out as a part of the “Guider Game”-project, a Finnish-made mobile game aimed for young adult immigrants, by “GAMU RY” (registered association). The goal of the game is to teach about the Finnish culture and society through the form of play.

At the beginning of this thesis work, there was the following problem: all the text content in the game was first developed in Finnish and almost entirely embedded into the game in such way that it was not easy to change that content in a short enough time. Part of the problem was that the game text content was scattered within the project files. All that Finnish content was then later required to support translations to at least Modern Arabic and English. Because of the educational nature of the game, the translations must be accurate (vertified). Due to the verification process, any automatic translation system was not going to be used for the project.

The purpose of this thesis was to find a solution to fix these issues. The proposed solution was to change the Unity-scripts, that control the already developed content, into a format which supported the text content in multiple languages. This change would be achieved by replacing all the hardcoded text into ID strings. Those IDs would be embedded into the program code and the actual text content would be returned from a separate dictionary type location depending which language was selected in the game. Naturally any image and sounds, that have content intended to be localized, required a switch mechanism tied to the language selection as well.

The primary goal for this thesis, and the related work therefore is about:

- Finding and implementing a more flexible localization system for the game

To better identify a good way to do localization, this thesis also delves into explaining:

- Localization in general, and what role it should play in the game design

In summary, this thesis covers: research, the found methods, the chosen method, the summary of the implementation steps and the conclusion. The more detailed documentation of the implementation and the use of the applied solution was written into a separate developer manual for the developer team.

2 LOCALIZATION AND CULTURALIZATION

Localization in general is a method of showing the same base content to a different audience. That audience might require another language, different units of measurement and the content needs to be adapted to their laws. One step further from localization is the concept of culturalization, which aims to adapt the content to the cultural norms as well, so the content feels like it is a part of that target culture.

These development processes have been created mostly in order to better satisfy the needs of a larger potential customer base. Reaching a wider audience can increase the revenue, but the cost for localizing the mentioned product need to be weighted against the possible returns. So, localization is not always an option for a small company, but small companies usually work with smaller projects that also have a lower localization cost. So, implementing localization is nevertheless a widely practiced part of the game industry. (Munoz, 2014.)

In Figure 1, there is an example of a globally localized version on the left with a more western feel and a Chinese version of it on the right. The content of the latter is clearly created specifically for the Chinese audience, with more vibrant colors and art that matches the target region more distinctively.

Figure 1. A visual example of localization and culturalization (Munoz, 2014).

A quick Google search can return many definitions by different parties, but a more through examination of the results yields at least one explanation from a seemingly reputable source (GALA, Globalization and Localization Association, founded in 2002). They define the term localization as follows: "*Localization (also referred to as "l10n") is the process of adapting a product or content to a specific locale or market. Translation is only one of several elements of the localization process. In addition to translation, the localization process may also include: Adapting graphics to target markets; Modifying content to suit the tastes and consumption habits of other markets; Adapting design and layout to properly display translated text; Converting to local requirements (such as currencies and units of measure); Using proper local formats for dates, addresses, and phone numbers; Addressing local regulations and legal requirements. The aim of localization is to give a product the look and feel of having been created specifically for a target market, no matter their language, culture, or location.*" (GALA, 2018).

It looks like the key is to not to just fixate on translating the words, when attempting to reach a wider audience. Localization has everything to do with the content, not just the text, it is also images, sounds and what is lawful to publish.

It seems clear that the message of any shown content can vary for different audiences. So, in our case, questions such as: "What makes Finland, Finland?", "How do we explain Finland to the target audience?" and "What is appropriate for our target audience?", are all very important questions when this kind of game is being designed and developed, if the game is to succeed.

In our case, Guider Game is especially aimed to teach the Finnish way of living to an immigrant, that is likely not yet familiar how the society in Finland works. Therefore, we cannot change the whole context of the game to have a non-Finnish feel, because the game tries to portrait the Finnish experience as accurately as possible. To reach this goal there are several pictures, animations and text form explanations in game that are designed to help players understand the context more easily. Even the player avatar options and some of the non-player-characters are designed to look more exotic than the more native Finnish society, to make it perhaps easier for the target audience to relate to the game sitsuation. Still it has not been forgotten that not every art choice that would work for a Finnish native, is issue free for someone with a different background.

So, in theory we are recognizing that there is not just a language barrier separating a new immigrant from the new society, but many other factors that could also affect the

way the game is designed. Recognizing and portraiting what is authentically Finnish is important, but so is the target audience's situation and background. That cultural contrast and interaction in between is what the game is trying to simulate and enhance. The game is designed to give more context and guidance where these various possible conflicts or problems might arise.

The designers of Guider Game have had to observe the needs of their target audience, and figure out the specific situations that in practice need to be highlighted in the game. This process takes time to get right, and is likely to be the key factor deciding the game's success.

3 GAME LOCALIZATION PRACTICES

We have discussed about localization and culturalization in general, but how are these concepts viewed on the game industry level? The International Game Developers Association (IGDA) has shared a best practices document on its website, that is going to be referenced several times in this section: "*This is the second draft of a "Best Practices" or "How To" guide for the translation and culturalization of video game content. It is a compilation of suggestions by people who have had years of experience in the field, all members of the IGDA Game Localization SIG. The aim is to help new-comers learn the trade, as well as to offer insights into tricks and tips that even more experienced localization staff can adapt and apply to their future projects.*" (IGDA, 2014, 1/36). We are going to examine a few of those suggestions to establish a base to build on. A keen observation here is that IGDA recognizes the separation between translation and culturalization, the same separation discussedin the previous chapter.

3.1 Culturalization

IGDA's best practices document informs us that when making any game: "*…don't forget the global, multicultural audience who will be participating in your vision, and hopefully enjoying it without any cultural disruption. Well-executed culturalization within a development cycle isn't turnkey; it takes time to implement successfully. However, the benefits to a company's content quality, government relations, and public image amongst local gamers will prove to be a valuable long-term investment."* (IGDA, 2014, 7/36). So, the wider the target audience is, the wider is the range of issues that need to be considered. If the goal of the developer is not to offend anyone, finding the right balance is the key and not every game concept will work with everyone. Implementing great changes like changing the graphical style or much of the content to suit many different cultures will be a sizeable investment, and is not always required. With some game genres or themes, the bar for someone to pickup the game surely becomes lower when the game does not differ too much from what they are used to as a consumer, but for some players the game just has to be localized and the more exotic or unfamiliar themes might feel exciting and fresh instead.

3.2 Internationalization

The second topic examined is the internationalization (I18N), whose goal is to make sure all content can be displayed on every supported language. The main topics under this have been categorized as follows: "*User Interface, Architecture, Programming, Advice for Development Teams"* (IGDA, 2014, 3/36). Because the "Guider Game" -project must deal with these areas when implementing a localization system, these topics are examined more closely in the following subsections.

3.2.1 User Interface

Selecting the game fonts is an important issue for any developer team aiming for an international audience. The font must be chosen early in the development cycle. The chosen fonts should be verified by a native speaker of the language to ensure it looks right. Not all languages, for example, share the same characters, letter sizes, or the space width between characters. (IGDA, 2014, 8/36.)

The display windows or the "speech bubbles" that contain text are a very important part of the game UI. It is recommended that the methods to implement these should be flexible enough for the varying text sizes for different languages. If possible hardcoding linebreaks should be avoided, or at least should be left as a problem for the developers and not the translators. Auto sizing font to a certain range is also recommended to support more variance in the screenspace resolution. (IGDA, 2014, 8-9/36.)

3.2.2 Architecture

It is important to store the localization assets of the project to a separate file structure where localization teams can more easily access them. Different languages are often coded with both a language code and a country code. This distinction helps with distinctions such as American English and British English. (IGDA, 2014, 9-11/36.)

Keeping track of all the game texts in order to get everything translated is also crucial. Having an Excel file or a version control system in place helps in this task. It is also important to not forget either 2D or 3D textures with any text content. A very important point is also to not "hard code" any translated text within the source code, because it will

cause problems with the translation process as someone has to then comb through each script or code file for that text. It is much better practice to have such a system in place where each text uses an identifier combined with a programmed method, that issues a call, which then retrieves the text content for each language from somewhere else. Where ever the text is stored, the location where translators can review or edit the text should be well structured. The structure is important, because it will help to follow the order of events or the storytelling. Just including labels, descriptions, or comments will help tracking the translated texts. This part may require some managerial skills and creative thinking, but it will solve many problems in the long run, especially for games heavy with text content. (IGDA, 2014, 9-11.)

The encoding of any text resource file needs to be optimized to not take too much memory space (IGDA, 2014, 10-11). Although memory is not that much of an issue anymore as it once was. As it stands at the end of the current decade (year 2019), even the regular mobile devices have plenty of both the system memory and storage capacity available. It is still a preferable practice to optimize the resource usage to the minimum to ensure a more stable performance of the target device.

3.2.3 Programming

The same point of no “hard coding” text as within the last section is repeated here within programming as well. The recommendated tip is as follows: “*The best practice is to isolate all text strings used in game into a resource file for each language.” (IGDA, 2014,* 11/36).

All the elements that show text within the game should always remain in the same place regardless of the text and resize around that center position when a longer or shorter text is given. This method of fixing each individual position for the varying sized translated texts is avoided and it saves much of a coder’s time that can be spent elsewhere. (IGDA, 2014, 11/36.)

The text resource files should be separated by language unless the versioncontrol supports editing the same file simultaneously. The resource files should be readable by the game project builder software, and not copied by a human hand from those files, to avoid easy mistakes as well. The less human hands there are in between the translator and the game the better, because there is less chance for an error. It is also good to

have a localization changes history with each build version to help track the text in game. (IGDA, 2014, 11-12/36.)

The order of variables in text can also change when the language changes because the sentences can follow a different order. So, either the text input format should not have variables inside the text, allow translators to edit those variables or program different handling for each supported language. (IGDA, 2014, 12-13/36.)

Remember that date, time, currency and number formats in the input text content written for the program can change between the supported languages. It also easy to overlook that, for example, changing currencies require a change in values as well. (IGDA, 2014, 13/36.)

It is good to have an automatic line breaking and wrapping of text in the game to avoid having an increased amount of text overflow bugs. It is also good to have a method of overriding that automatization in case the text needs to fit in the right way into the game elements. Different languages have special characters and symbols that the game may need to support within the text, and those must not overlap with any hidden mechanism for the text display system. Translators should also add tokens or commands within the text where the text could be split, to make it easier for the automated system to fit the text into the game elements. (IGDA, 2014, 13-18/36.)

3.2.4 Advice for Development Teams

It is highly recommended to work with localization as early as possible if it is planned for the game. The later the localization work starts the higher the risk of having to redo and revisit parts of the project that have already been completed. So, planning localization into the project from start simply saves time and also improves the localization's quality in the long run, because it is not rushed into the project at the end. It is good to have standards for how to proceed with each work phase of the localization process that each team can utilize. This leaves less room for errors when adding different languages by different localization teams. The developer teams should not forget good communication skills and it is also important to provide all the necessary information to the the translator teams that need it to avoid delays and mistakes. (IGDA, 2014, 19/36.)

3.3 Localization

In the video game industry the localization process often include producing localized resources or assets (translated files, voice acting, translated packaging material, translated promotion material). Game testing is in a big role as well. IGDA has broken the process into following categories: "*Familiarization, Glossary and Style Guide Creation, Translation, Voice Over Production, Linguistic Quality Assurance, Master Up and Sign Off*". (IGDA, 2014, 20/36.) All these categories are summarized in the following subsections as they contain useful information for the Guider Game project.

3.3.1 Familiarization

Every time something needs to be translated it is better to get familiar with the context and content surrounding the text. So, familiarization is about giving the translator teams time to play and experience the game and its content from the players perspective. A few days of familiarization is often enough to start getting the right "feel" for the game. In case of larger games such as longer RPGs or MMORPGs a much longer (a month or more) familiarization perioid is usually required, because the amount of content is typically much more overwhelming. The translators should keep a record of useful information or ideas they come across during gameplay to speed up the process later. (IGDA, 2014, 20/36.)

3.3.2 Glossary and Style Guide Creation

After achieving a suitable level of familiarity with the content of the game, the translators should get all the relevant information and guidelines from the source developer team, and if possible in a well structured format. Then the translators proceed to create a guidelines and organize all the main elements that require translation. The scope of the project matters how long this stage will take. It is also good to brainstorm in groups and divide the team into more specialized groups in case of a larger project to generate a lot of ideas to use. The main development team should be available when needed and also the translators should provide the quality assurance team any relvant information in a document format when they start testing. (IGDA, 2014, 20-21/36.)

3.3.3 Translation

Once the guidelines are set the translation can begin. It is good to have some specialized proofreaders for the translations as well to track the consistency of the work. The number of editors and translators can vary depending on the target language. Translators should be fluent in both the source language and the target languages. Translators also require a reasonable amount of time for their work. All this should reduce the number of errors that slip through. (IGDA, 2014, 21/36.)

3.3.4 Voice Over Recording

Voice over work usually requires more time than text. Scripts for the voice acting also require to be translated first. Some games require lip-syncing, which need to be specified in the scripts as well. Voice over files can require a lot of storage space within the project and handling them withing the game system can easily become quite complex task, so voice overs are usually not something small companies do. It is always good to have the right voice for each localization and always have a translator present during any recording session. (IGDA, 2014, 22-23/36.)

3.3.5 Linguistic Quality Assurance

Always have a good mix of professional linguists and the fluent speakers of the target language in the quality team. Let the testers write down their suggestions, but if possible always let the orginal tranlators verify the found bugs indeed are bugs and not intentional. Have a policy in place for who has the final say in case there is a conflict. (IGDA, 2014, 24/36.)

3.3.6 Master Up and Sign Off

This stage is when the game is almost ready for release and the last minute changes are made. It is good that a translator has played the localized version through at least once or more, to avoid embarrassing mistakes slipping through to the final product, because it is often the first player impressions that matter the most. (IGDA, 2014, 24/36.)

3.4 Project Planning

Plans can usually vary a lot in between different companies. Still some guidelines for the project's localization planning are introduced by IGDA. Localization is not a trivial task, and thus requires a proper plan.

3.4.1 Pre-Production

This is the time to plan the supported platforms, languages, project budjets, and the release date. Once the initial plans and concepts are set, then it is the right time to start revieweing possible culturalization isssues and to select the candidates for implementing the localization, with its various stages. (IGDA, 2014, 25/36.)

3.4.2 Production

Work with the source language begins. It will follow similar route as the described in the section 3.3 Localization. Once the source texts are set they can be taken over by the other localization teams. It is also good to have established requirements for the international build from the start, to avoid repeating any of the work. (IGDA, 2014, 25/36.)

3.4.3 Alpha Phase

This phase begins the testing phase with a build that includes most of the localized text and audio. It is also recommended to try establish the worst case test scenarios for the UI, but this is not mandatory. Once the internationalization bugs have been addressed, the quality assurance for the localization can go into full gear. Communication between teams during this phase is important as it will improve both quality and speed up the process. (IGDA, 2014, 25-26/36.)

3.4.4 Beta / Localization Sign-Off / Gold Master Phase

For this phase to begin all the translation work should be complete and all the risen issues must have been addressed. One last project overview is done and all the useful information should be documented to help with the new projects. (IGDA, 2014, 26/36.)

4 LOCALIZATION REQUIREMENTS FOR GUIDER GAME

Before the right localization method for the “Guider Game” can be chosen, the requirements need to be listed down and considered. The developer should perhaps consider questions such as what type of a game it is; how much text the game has; does the game content change with localization; are there many different game elements to localize (text, images, sounds…)?

The Guider Game is a 2D-game with most of its content viewed from the side, like a comic or a cartoon. The player explores typical Finnish environments, which offer games inside the adventure mode, to learn more about Finland and living in Finland.

The player has an option to select between a male or a female character, which are portraited as someone who are both in their youth and have just immigrated to Finland. The player character meets native characters and also other immigrants in the game.

Most of the game dialog happens inside speech bubbles. The user-interface has text in various formats, either guiding the player or offering more information about the topics inside the game. The text is rarely too complex and while there is text in many different places, it is not too long. The main issue with the text is in sorting the source texts neatly, so it can be overviewed and edited from one place. That place needs to contain enough information about the text, so the developer knows where in the project it fits without browsing through the whole project within Unity. Having the text separated from Unity also allows better accessibility for those who are not familiar with Unity, but could help with the translation process.

The content is aimed for a certain group (immigrants), and the whole game design is based around that. So, the content stays the same and the translations are there to merely help getting the message through. Still, the early development version’s text is almost entirely in Finnish. In the next phase of the development, the game requires support for both English and Arabic translations. Also, not all of the text in the game is in text format. Some game text is embedded in images for aesthetic reasons. So, there is a need for a Unity-script that handles switching that kind of content when the selected language changes.

The early development version of the game has sounds, but those are mostly sound effects or background music, not spoken words. While it might be useful to have a system

in place for changing sound with the selected language, it is not a high priority requirement.

To summarize: a 2D-game in a comic style; Finnish base texts; a lot of short texts all over the game; a few images that contain text; the text content, or the main message, stays the same for the translated text; no full voice acting; and the translations must be accurate.

It is likely that there is no one complete system for everything, that could be simply put into the project as it is. The new localization system will contain different parts that need to work together with parts of the old system. Since the text is in the main role, the system that controls the visible text is the highest priority. Every used Unity-component needs to be supported by the system. The controlling element of the system will be the language selection setting, and its handler script.

The text source should be examinable from one place to make the editing of it easier for a human developer. The translation process from the base text (Finnish) is too big of a task to automatize, because the translated text must be verified (by a professional translator) and Finnish is a very hard language to translate in such a way. So, the source text file, will contain keys (IDs) for every form of text such as: whole sentences, dialogs and UI-labels. All the text content produced with Unity will be indexed in this way.

To help the human developer edit and overview the list of keys and texts better, the keys will require an intuitive naming logic (scene_topic_number_subtopic_number) rather than a more confusing string-type-text-ID (“kjjdhs32jd21kf23”), which are more suited for an automated system. Automated systems can afford more confusing ID naming logic, because there is no human observing these string values. Even an automated system would need a view where the developer could examine the whole created content in a way a human can understand. So, why not just create the IDs with a such logic from start, since there is nothing we want to hide behind the confusing IDs?

Both Arabic and Finnish words require special characters. How to support these characters in Unity needs to be examined. The Arabic letters are also connected, which makes them require a specific way to be shown correctly in Unity, there are Unity plugins specifically for that purpose.

4.1 Finding the Suitable Parts

This section explains the major parts, that were examined or considered during the research for a suitable solution for the Guider Game's localization needs.

4.1.1 Unity Localization Manager Tutorial

Since the "Guider Game" project is done with Unity, one good option could be to follow Unity's own tutorial for localization (Unity, Localization Manager, 2018). Many games, that are made with Unity, utilize this method, and it follows the best practices guidelines by IGDA (IGDA, 2014). Basically this method separates the localized language files into a localization folder. Each language has its own text resource file and folder. The localization manager Unity-script reads only one of them when the language is selected. The files are in JSON-format and they contain an array of JSON-objects. Each JSON-object contains at least a key and a value (the actual text content) variables.

Unity provides a dedicated tutorial page, along with a video series hosted on YouTube, for this method. The tutorial is easy to follow (labeled as "beginner level" by Unity) and the system is easy to implement for many Unity project text translation needs.

Main problem with this option for Guider Game is that, the text is not very easy to view and edit in JSON-format compared to something like "Microsoft Excel" or "Google Sheet" table-format. That is not the best scenario in our case, because we would like to avoid as many human errors as possible during the process of adding the translated texts to the game.

Another criticism for this tutorial is that it does not fully cover what localization is, and it only focuses on handling multiple translations for the game with their method. The developer needs to come up with another manager(s) for handling the content changes if the localization requires changing more than just the text for different languages or target groups. So, it by no means is a complete system.

4.1.2 PolyglotUnity Project

“PolyglotUnity” is an interesting Unity-plugin for a Unity project localization needs. It is open sourced and free to use. It contains two major parts: localization components for Unity and a customizable Google Sheet that can be used to inject all the translatable text content into the Unity project. The PolyglotUnity plugin comes with a manual, that covers both installation and how to get started. (Agens, 2018.)

A key part of “PolyglotUnity” is its Google Sheet “PolglotGamedev”. That sheet can be copied and used freely. So, the sheet solution looks very useful for our “Guider Game” Unity project localization needs. In the optimal scenario, the Google Sheet could be used for handling the version control for the translators. The sheet can also be imported directly into the Unity project with “PolyglotUnity”, thus eliminating the need of manually copying and pasting the text from the sheet. The importing of the sheet can be done even over the Internet when the settings for “PolyglotUnity” are set. (Polyglot Gamedev Project, 2014.)

In total this method of handling the translations seem to fit our requirements, and be quicker to implement. PolyglotUnity has been tested by many other users and it works with multiple Unity versions. The Polyglot solution might still require a little editing and it definitely requires implementation of few additional Unity-script managers for other localized content to complete the localization system that Guider Game requires.

4.1.3 Making of Complementary Unity-Scripts

Since there was no complete solution available for free, or very low cost, when this matter was researched, the remaining requirements for the “Guider Game” project can be achieved simply by creating those Unity-scripts with the skills and resources at our disposal.

Guider Game requires specific script manager(s) for changing images, sounds and certain fixed sets of Unity “GameObjects” for each supported language.

There is also a need to go through all the existing Unity-scripts for any “hard coded” text, that require flexible editing, because it is better to not start from zero at this point. This

flexible editing may require helper scripts that contain utility functions to reduce unnecessary repetition of certain lines of code.

5 IMPLEMENTATION OF THE LOCALIZATION SYSTEM

The localization method for the text translations, that was chosen for the “Guider Game” project was the open sourced “Polyglot Project” (Agens, 2014). The choice was made mainly because of the customizable and user-friendly “Google Sheet”; “TextMesh Pro” Unity-plugin support; and the “Arabic Support for Unity” Unity-plugin support.

“PolyglotUnity” also contains a localization manager, which can be accessed by other Unity-scripts. That manager can for example be used to initiate the “change language” process, which automatically changes all the content listening to that event, or get translated text for a matching key, among other things. So, this localization manager is something, that can be used as controller component for all the additional localization components that need to be implemented for example for images and sounds.

The targeted custom copy of the “Google Sheet” can be linked to the Unity project by changing the “PolyglotUnity” configuration settings in Unity. Then the sheet can be downloaded as a new asset, which the “PolyglotUnity” system then turns into a dictionary. All the correctly added text, visible on the “Google Sheet”, can then be accessed through a matching key (string-ID). The plugin also has a search feature included within its text localizer components. The search feature returns a list of close matches, to select from, even when only a part of a key is written to the input field. All this speeds up the process a developer has to go through and also reduces the possibility of human errors by automating few of the steps in getting the text into the actual project scene.

There was simply no competition for this decision in the short timeframe reserved for localization system implementation. It would have taken more resources and experience to create everything from zero, and also to test it all on top of that. The Arabic support alone would have required much more specialized skills, including the language itself, without an open-sourced solution created by a native speaker. The project was also beyond the early planning stage, so creating a completely new system during mid-development would have also been too problematic. So, we were lucky, and can be thankful, that the authors of those projects have provided us with solutions. The only task left was a little flexible script editing and in depth examination of the manuals and testing to implement these solutions into our project.

To summarize the reasoning behind choosing PolyglotUnity with few modifications and Unity-scripts as the implementation method:

- Supports: “Arabic Support for Unity”-plugin
    - (**open-source and free to use**)
- Supports: “Text Mesh Pro”-Unity-plugin
    - (**free to use and already in the project**)
- Has gone through multiple years of development cycles
    - (**less bugs and more optimized**)
- Has Unity-components to localize a supported text components
    - (**with a search feature**)
- No need for a separate language controller object
    - (**can be used to control language selection dependent events**)

The separate “GAMU Guider Game Developer Manual for the Localization System Implementation and Usage” was written for the development team of Guider Game. The manual contains much more detailed explanation for all the implementation steps. The manual is written as an installation manual and all the instructions are also paired with the relevant screen captures. First the manual explains what files and settings need to be in place. Second part explains the 3rd party components, and their use for the project. Third part covers every game scene, that was ready at the time, describing their source code and Unity Editor related changes. The last part explains my complementing Unity script components and their intended use.

The main purpose of the following subchapters is to summarize each implementation step, and to explain the reasoning behind them.

5.1 Step 1 Requirements and Preparation

“PolyglotUnity” files and the basic instructions were available at the Git-repository of “PolyglotUnity” (Agens, 2014). The “Arabic Support for Unity” files and the basic instructions were available at its respective Git-repository (Konash, 2016).

When the files of both “PolyglotUnity” and “Arabic Support” are in place, the settings for them need to be set in the target Unity-project. Since “PolyglotUnity” supports the “Arabic

Support for Unity" Unity-plugin, the only settings that need to be set are the "PolyglotUnity" settings.

The custom sheet needs to be created by copying the free-to-use original "Polyglot Official"-sheet, and then hosting it on a separate Google account's "Google Drive" space (Polyglot Gamedev Project, 2014). The settings (Docs Id and Sheet Id) related to this new location must be set in the PolyglotUnity's configuration settings in Unity, before the remote sheet can be accessed through Unity. Google provides helpful instructions (Google Developers, 2018) for determining the correct values for the targeted file, that can be read through a link.

Guider Game -project also uses "TextMesh Pro" Unity-plugin (Unity Technologies, 2017, TextMesh Pro). "TextMesh Pro" is supported by the "PolyglotUnity" system, so there is also a setting for that within PolyglotUnity configurations (Agens, 2014). This plugin also provides the tool "Font Asset Creator" for creating new fonts, that it can use. So, this tool is therefore utilized to create new fonts for: both "Scandinavian" and "Modern Arabic" characters. To create these fonts, a correct Unicode block range for the target characters is required. The Scandinavian characters can be found within the "Latin-1 Supplement" subset of the Unicode blocks version 11 (Unicode Inc, 2017). The required Arabic symbols can be found within: "Arabic", "Arabic Supplement", "Arabic Extended-A", "Arabic Presentation Forms-A", "Arabic Presentation Forms-B" (Unicode Inc, 2017).

An important reminder, the problem of showing Arabic letters is not only a font issue. Without a specific plugin or script that makes the letters connect correclty, the text would look wrong with the default Unity-text-components or the "Text Mesh Pro"-Unity-text-components. "Arabic Support for Unity" fixes that problem for us (Konash, 2016).

5.2 Step 2 Google Sheet

The Polyglot's official sheet's "About"-sheet contains plenty of information about the Polyglot project, of which maybe the most important are the licensing related information and the helpful links (Polyglot Gamedev Project, 2014).

The "Master"-sheet contains the information related the translations (Polyglot Gamedev Project, 2014). First row is for the column titles and, by default, it is color coded. Supported languages are separated in columns, so for each row there can be text content for that specific language. The most important column of the sheet is the first

column (named: “STRING ID”). This column contains the key to access that text content through the dictionary created in Unity. In our case the IDs for each scene or scenario started with the same first part then went into more specific details. For example: “SCENENAME_CONVNUM_DIALOGNUM”. There is also a “DESCRIPTION”-column where each row can have additional details for the user of the sheet to set or read more about the text. Descriptions in our case were about where in the project the text is located and where the original source text was.

To smoothen the user experience of this sheet a little, it is also useful to customize the master sheet. In our case hiding the unused language columns and resizing the used language columns helped editing the document. Adding background colors to the cells also helped navigating through the information in the sheet. Setting the cell’s format setting: “text wrapping” to “clip” also made the document easier to view and edit, because the text content’s would still be visible when a specific cell is accessed, but any long text stays hidden when the user is for example browsing the “STRING ID”-column to find a specific text or adding more content to the sheet.

5.3 Step 3 New Scripts

While “PolyglotUnity”, “TextMesh Pro” and the “Guider Game” project itself already had many useful Unity-scripts, combining all these systems together created a need for new helper scripts. So, I created a few new utility scripts. These new scripts can be used to do specific tasks. They work either alone or acting together with these other scripts. So, I named these scripts “LocalizationHelpers”, which is also the name of the C#-namespace for them. The main functionalities of the “LocalizationHelpers” have been explained in the “GAMU Guider Game Developer Manual of the Localization Components and Required Modifications”.

5.3.1 Localization Switches

These are three simple Unity-scripts, acting as switch components:

- “LocalizationAssetSwitch.cs”
- “LocalizationImageSwitch.cs”
- “LocalizationSpriteRSwitch.cs”

These component look for a change in the selected “PolyglotUnity” system language. When that selected language changes, the scripts switch the related Unity components to a version that is customized for the selected language. The scripts need to know which Unity component setup is meant for what language, and there is also a fallbackoption when there is no support for the selected language. The three versions support different Unity components, but share the same principle. It is possible to expand the supported Unity-components to create more of these switches in case the project demands them.

5.3.2 StringLocalizer.cs

This Unity-script is used to get a list of keys from the user (developer), that will be used to get a list of text content strings back from the Polyglot’s Custom Google Sheet. The keys need to match exactly to the “STRING ID”-cell content set in the “Polyglot Custom Sheet” to get the correct text for the currently selected language. If no match for a given key is found, then the given “Key”-string-parameter value is shown instead.

This Unity-script component has a mildly customized Unity Inspector view, that is also seen in the figure 2. This view is intended to help the developers to, for example, see all the messages in a certain dialogue for the selected language in the Unity Editor.

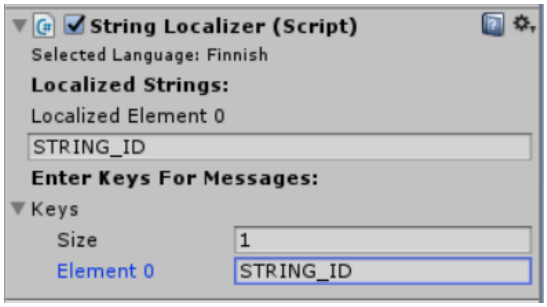

Figure 2. StringLocalizer.cs in Unity Inspector.

Inside other Unity-scripts this script can be called get the localized text from the “PolyglotUnity” system, but it also removes any line-breaks or line-resets from the text. This removing is done to avoid an issue of text getting over the other lines of text when it is given to “TextMesh Pro” while the game is running.

It is also possible to add “PolyglotUnity” text localization components through this script, when the game is running, in case the situation requires a more dynamic solution. This

feature also helps to better ensure the PolyglotUnity's text alignment controls are in place when the translated text requires a different text alignment (English vs Arabic).

5.3.3 TMProHelper.cs

This simple, but useful Unity-script only works in Unity Editor. The purpose of this script is to forcefully update the text content in the "TextMesh Pro" text components, when the selected PolyglotUnity language is changed and a PolyglotUnity localization component should change the text content. Without this component, the forementioned change happens when the game is running, but, for some reason, it is not implemented, or it is bugged, for the Unity Editor mode.

5.4 Step 4 Modifications

This section is about the required Unity-script modifications for the "Guider Game" project covered in the "GAMU Guider Game Developer Manual of the Localization Components and Required Modifications". This significant number of modification is required mostly because of the "hard coded" Finnish text embedded into the Unity-scripts. All of the game scenes and their controlling scripts had to be thoroughly examined in order to replace "hard coded" text with the more dynamic ID based solutions.

This process took for about a month to do, but it also provided back a very good understanding of the mechanics the game worked with. The goal was to not change too much of what was already in place, only the critical parts.

In the Unity scenes, this modification process was usually either of two solutions. The first solution was just adding of relevant localization components and in some cases creating a fitting string-ID key for the PolyglotUnity Custom Google Shhe eet to connect the text to the game. The second solution was replacing the speech bubble dialogue control script messages with the newly created StringLocalizer-script. Both solutions required a little testing afterwards to help avoid mistakes.

In the resource text files, this modification process was more straight forward replacing the Finnish text with string-IDs for the PolyglotUnity system, so the modified scripts could then use those IDs to get the text based on the selected PolyglotUnity language.

In the Unity-script files, this modification process focused on figuring out the logic of the script, replacing the “hard coded” texts with method calls and modifying the other crucial parts of the written source code to ensure the new dynamic solution is better supported.

5.5 Step 5 Testing the New System

5.5.1 Functionality

The last stop before reaching the goal set at the start of this stage of the project is to pass the functionals testing. Everything implemented needs to work as intended. During the implementation phase, there was always testing at tandem with the changes, but it is a good practice to do a separate testing, that focuses on nothing else.

The developer team also went through the new manual, and then reported any possible issues back to me. Then they would implement my proposed modifications to integrate the new localization system. After those initial steps, the new system would be tested once more. If any new major issues arise, we would fix those issues left over as soon as possible. Then last, we would proceed finishing the project with new scenes, where my new role would be more of a supervisor or a consult than a developer.

5.5.2 Results and Benefits

Before the implementation of the new system, we had a set objective based on the conclusions and recommendations referenced in the localization theory chapters:

- The new solution should allow translators to directly edit the texts.
    - Developers would spend their time doing something else.
- The project has multiple scenes with scattered files, containing to be translated text. The implemented solution would centralize the access to those texts into one location.
    - Allowing everyone involved easier access and to save time.

Since this project had to have the Arabic support, implementing the new system was mandatory. Therefore, we decided to first implement the new system and then gather more qualitive data from the developers to see if the solution provided any of those

foreseen advantages. So, the developers had a few months to do their work and then we had the discussion about the new system, and what kind of impact it has had.

The developers were glad that the text team, that is not directly involved in Unity side of the development, was now able to edit the texts, when they wanted to change the text content. This had previously required a multi step process:

1. The text team adds required change to the Trello board for Unity developers to see.
2. The developers would check the Trello backlog for what they need to change.
3. They would go open the project folder and search the correct file that contains that text, sometimes a change would require opening multiple files.
4. Time is spent depending where the text is and how scattered the text is.
    a. One word could be in multiple locations, because scenes contain multiple game elements containing translatable text.
    b. The text file and the Unity script file for a game scene in many cases were the same file.
5. The developer who took the task reports the issue as done in Trello.
6. The text team checks the report in Trello and reports the change as done in a separate document.

All these steps could be done for something as trivial as a mistyped word, but nonetheless it had to be done. In comparison with the new system the process changes to:

1. The text team opens the privately shared Google Sheet for the Polyglot system.
    a. They use the automated search by typing in the name of the scene or the full ID of the text they need to change.
    b. They can then view that text in all supported languages side by side on that same row of the sheet, and select the language cell they wish to edit.
    c. They save the changes and inform developers if the change was big enough, to be sure it gets into the new game build.
2. The text team will report the change as done in their own document.
3. The developers will get all the changes done to the sheet, when they download the new version of the sheet through Unity with a click of a button.
    a. The new system has that edited text in place for every element the text is used at in that instant.

The power of using this more flexible system comes from the ability to have all the text content accessible from one easy to access location. Editing that text requires as much knowledge as editing an Excel file's cell content. There are few rules in place for that editing process, but once everyone involved understands those rules, that process becomes simple enough to follow.

For the Unity developers, the Polyglot system offers a way to eliminate the need for opening multiple script files, or worse going through multiple Unity GameObjects in the Unity Editor to change all the texts in those objects manually every time they need to be changed.

The developers also liked the ability to be now able to quickly switch the languages back and forth (a Polyglot system feature). That allowed them to skip the time to develop a new reliable script to do that. This quick switch feature can be utilized while the game is running, and so it can thus be used to educate the end user as well.

So, to the developer team this new system has been a great improvement. Not only the new system brought them the Arabic support they desperately needed, they are now able to allocate more time develop the game mechanics for the scenes and focus less on the text related features.

6 CONCLUSION

The objective of the thesis was to find or create a suitable localization method to be implemented into Guider Game. To summarize what has been achieved in steps. The first step was defining the situation and what was needed to proceed. The second step was research and theory about localization, and its best practices. The third step was reflection and clarifying the localization requirements for the “Guider Game” project based on the main research. The fourth step was to find a suitable solution. The fifth step was the implementation and testing process of the chosen solution.

The implementation of the localization system into a separate test branch of the project went as planned, and the new system was ready to show localized content within a month. The implementation required learning almost everything about the inner mechanics of the developed game version at the time, but it was not an impossible task. The majority of the time after the initial implementation was spent in filling the “Google Sheet” with the base language text content of the project, naming IDs for the text content and modifying the “hard coded” texts within the project’s Unity-scripts. During this latter phase, the detailed developer manual for the implementation of the localization system was also created/compiledin order to better document the process and to leave the other developers with more than just a modified version of the project. With a manual the developer team could more easily learn to adapt to the localization process and its requirements. Having a manual also helps debugging the possible issues that arise during the implementation of localization system to the main development version of the project.

This thesis has demonstrated that it is highly important to plan and research the localization process (or any topic) before any project is fully started. A good plan, especially for localization, reduces a varying amount of extra work, especially for the coders, depending on the scope and complexity of the project. Starting localization early enough also allows translators to have more time to work, both with translating and to work with the developer team, which willsurely improve the overall quality in the end. At the time of this thesis project, it was not possible to fully test the Arabic translated text content with the new localization system when this thesis was being worked with because the translators started their work when the work for this thesis was almost finished. This was mainly because the base texts were still in development. Therefore it

is recommend (based on the thesis section 4: “Game Localization Practices”) that when any part of the translations is in place the translators could see their work in game, and perhaps also help the developer team to adjust anything that needs to be adjusted. If the team was larger, it would have been better to teach the translators to use the new “Google Sheet” and any special tags they can add within the text. The translators filling the “Google Sheet” would in theory help to eliminate extra human errors, when copying translated text documents into the sheet afterwards. This method would require a more closer co-operation with the core team or a very clear documentation for the translation team.

In the end, the new system in place suits the needs and the scope of the “Guider Game” project, and that is a result this thesis related work was set to accomplish. The implemented solution received enough positive feedback from the developers to confirm that conclusion (Sectionr 5.5.2 Results and Benefits).

REFERENCES

Agens AS. 5.9.2015. PolyglotUnity. GitHub. Referenced 11.10.2018 https://github.com/agens-no/PolyglotUnity

GALA, The Globalization and Localization Association. What is Localization. Referenced 10.9.2018 www.gala-global.org/industry/intro-language-industry/what-localization

Google Developers. Introduction to the Google Sheets API. Google Sheets API v4. Referenced 18.10.2018 https://developers.google.com/sheets/api/guides/concepts

IGDA, IGDA Game Localization SIG. 11.9.2014. Game Localization Best Practices. Referenced 10.10.2018 https://cdn.ymaws.com/www.igda.org/resource/collection/65D89F6D-3BD8-46EA-B32E-BE34236408D5/Best-Practices-for-Game-Localization-v21.pdf

Konash, A. 24.12.2016. Arabic Letters Support For Unity. GitHub. Referenced 18.10.2018 https://github.com/Konash/arabic-support-unity

Munoz, L. 6.8.2014. Localization vs. Culturalization. Referenced 10.10.2018 http://web.archive.org/web/20141010172136/http://www.hitcents.com/blog/post/localization-vs-culturalization

Polyglot Gamedev Project. 16.3.2014. Polyglot Gamedev Project: Master Sheet 1.0. Referenced 11.10.2018 https://docs.google.com/spreadsheets/d/17f0dQawb-s_Fd7DHgmVvJoEGDMH_yoSd8EYigrb0zmM/edit#gid=310116733

Unicode Inc. 16.10.2017. Blocks-11.0.0.txt. Referenced 18.10.2018 https://www.unicode.org/Public/UNIDATA/Blocks.txt

Unity Technologies. 28.2.2017. TextMesh Pro. Unity Asset Store. Referenced 18.10.2018 https://assetstore.unity.com/packages/essentials/beta-projects/textmesh-pro-84126

Unity, Unity Scripting Tutorials. Localization Manager. Referenced 11.10.2018 https://unity3d.com/learn/tutorials/topics/scripting/localization-manager