LOCALIZATION GUIDE

Vol 2.0

allcorrect

Contents

Introduction

In this second edition of our “Localization Guide”—a collection of localization tips for game developers—we have added new chapters, updated data on language markets, and refreshed information in existing chapters. We’re sure this new guide will help your project achieve the success it deserves in the global market!

Game development is a complex process that demands an incredible amount of management from developers of all types and sizes—from the solo indie developer to the biggest, most experienced teams. And localization of in-game texts is a challenge every team must grapple with. It’s a crucial element for entry into new markets.

The Allcorrect team has been working in the localization field for 15 years, so we know how to best organize processes to this end. By using these processes, developers can ensure they receive high-quality translations on time, while keeping costs and stress levels down. In this book, we share insights to help you work with localization teams more easily and efficiently.

If you have any questions after reading the book, please feel free to reach out to us by email at pr@allcorrectgames.com.

Development

Development is the biggest and most labor-intense stage. It's also the right time to start thinking about localization. By preparing technical grounds for seamless localization (that is, by providing access to the translation directly inside the code), you'll save quite a bit of time, money, and sanity. We'd like to share 6 principles that will make you (and whoever is responsible for localizing your product) a little happier.

Six Principles of Localization-friendly Development

1. Separate the text from the code

This needs to happen at the development stage for any kind of project. Doing this at a later stage may result in various problems. For example, it may not be possible to localize certain strings later on. If localizable resources are separated from the code and stored separately, they can be managed. Nowadays, there are good (and free!) systems that allow you to do just that.

2. Make sure you support a variety of characters

Perhaps you'd like your game to become a hit in the UAE or China someday. It would be quite a pity to discover that it's difficult or even impossible to integrate characters or languages written in right-to-left fonts. We'll leave it at this for now, since we'll be talking more about fonts later.

3. Make sure you support different input methods and keyboard layouts

Traditional European input methods and keyboard layouts aren't suitable for Asian languages. But let's not forget that the Asia-Pacific region makes up half of the world's gaming market!

4. Don’t abuse concatenation (gluing strings together)

If sentences are composed of pieces of code, then when the language changes, those sentences will look like a bad translation. Or sound like Master Yoda they will.

Let’s take a closer look at this. Many developers who aren’t experienced in localization think that replacing different phrases with variables will save time and resources. However, in reality this might lead to many of those phrases containing such variables being translated rather generally.

One project we completed had complicated string generation. We had to ask the translator to rewrite some of the game’s code. The original English version had a number of beginning and ending phrases that matched up well and produced a multitude of possible storylines for the player. But when translating to German, some of the parts didn’t match up when they were “glued” back together. Therefore, we had to add a special tag system. Most of the text remained unchanged, but the grammar component was stored in tags.

5. Localize the graphics

Text within graphics should also be localized. Any self-respecting localizer will have no problem accepting .psd files and many other popular graphic formats. But it will need redrawing, which is more expensive. That’s why it’s best to remove text from the graphics beforehand and then return the localized strings back.

There may be other additions to this list, depending on the nature of the product being designed; it’s impossible to create a universal checklist.

For example, many apps might find it useful to use libraries for date formats, currencies, addresses, phone numbers, and other cultural nuances. By the way, there are lots of these libraries available, and they’re open source. For example, try searching for i18n libraries. Ideally, the app should load the data directly from the browser or the operating system it was installed on.

6. Don’t forget about testing

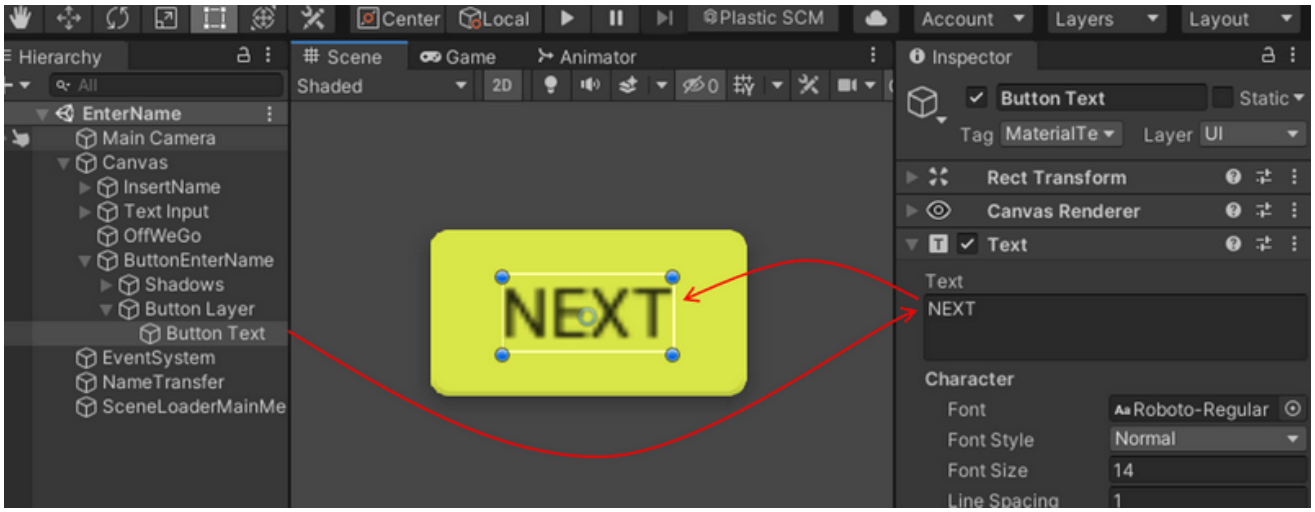
Let’s assume that we separated the localized resources from the code, made sure all the above requirements were met, and even added the libraries. If all this is true, then it should be easy for localizers to use machine translation to add target languages such as Russian, Greek, or Japanese to the product. This trick lets you see if the volume of text increases or decreases, if all the symbols are supported, how lists are displayed, and whether or not pieces of code end up in the final product. This is called pseudolocalization. It’s good practice to include pseudolocalization as part of your standard testing. For the vast majority of localized products, this allows you to identify many shortcomings and eliminate them efficiently. In other words, pseudolocalization will help you make sure that the localized build retains initial functionality and corresponds to the declared parameters. This is far better than getting material from the localizer a few days before release, having to embed those changes, and discovering that nothing fits!

Game Localization in Unity and Unreal Engine

There’s a huge number of things to consider when localizing a game, and one of them is the engine the game was created in. We decided to take a deep dive into the topic of game localization in two of the most popular engines, Unity and Unreal Engine.

Game localization in Unity

The easiest way to add text to your game in Unity is to make it a property of a UI object. For example, when we create a button, a text object is automatically generated and linked to it. We can then add the text we need (“Next” in the image below), and voilà—the text appears on the button.



This text is saved in a Unity scene file, with the .unity file extension. If our button never changes, then that's all there is to it. But what if we want the text on the button to change depending on the context? Let's say we've decided to localize our game. This means that we'll need to instantly switch between translations of all the text elements that appear on the screen as required.

Luckily, there's no need to painstakingly create dozens of new buttons, tables, and boxes for every language. Instead, we can just link each UI text element up to a text file that contains the phrases we need. These files are called dictionaries. Once we have dictionaries set up, the game will just load the right text from the right dictionary in the right language. Unity supports various text file formats: .txt, .csv, .xliff, .json, .po, and more.

We'll still have quite a bit of coding to do, though, as our game will need:

- A function that sets the language.
- A function that loads text from the dictionary into the device's memory.
- A function that transforms the loaded text into a form Unity can understand.
- A function that makes the translated text appear in the game's interface.
- A localization manager, which is a script that links text objects in a scene up with dictionaries.

- A loading manager, which waits for the localization manager to do its job (when the player chooses a language) before loading a new scene.

Next, a little manual work will be needed to:

- Link the localization manager up to every localizable text object.
- Assign a script to every text object that gets the key for the right text and puts the localized text in the Text field.

Complicated, isn't it? Thankfully, Unity recently released its own localization tool, unsurprisingly called Localization (it did exist before, but it was only prereleased in April 2021—and judging by all the bug reports on the tech support forum, it's still early days). It automates a lot of the localization process, allowing you to set locales and then play sounds and show different strings and images on the screen for each locale. It also carries out "pseudolocalization", or machine translation, so you can see what the interface will look like in one locale or another. The Localization tool also lets you export data into Google Sheets, not just text files.

But way before the official Localization tool came into being, there were various third-party game localization management plugins, such as the I2 Localization plugin by Inter Illusion. It has similar functionality to Unity's own tool:

- support for Google Translate.
- automatic search for missing, duplicate, or unused translations.
- support for plurals.
- support for RTL (right-to-left) languages such as Arabic, Hebrew, and other languages.
- non-text object localization (you might need different images or audio files for different languages).
- font and texture management (different object fonts/textures can be set for different languages).

It costs $45.

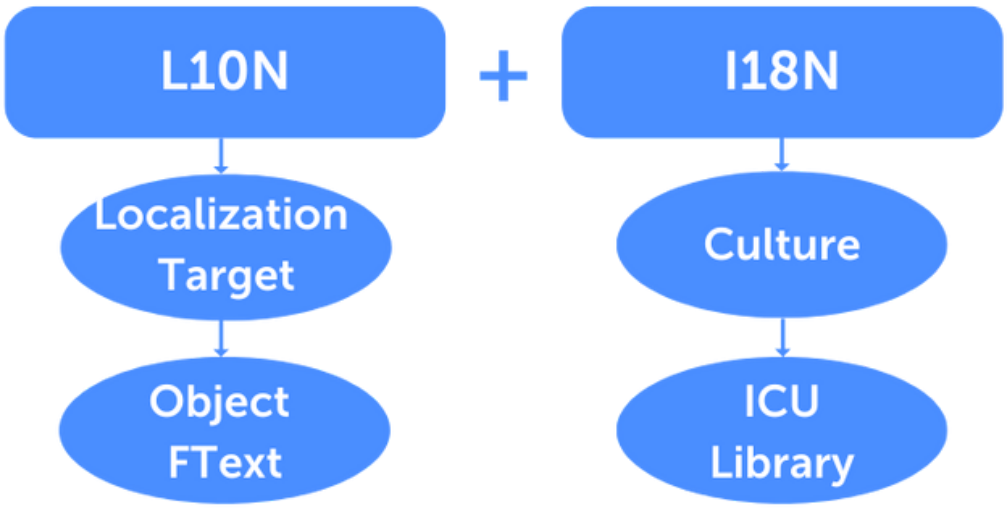
So that's languages dealt with. But game text isn't just about the meanings of words. It's also about how they look. In game localization, it's not just what's inside that counts: exteriors also need attention.

UI Text, Unity's standard tool for working with text, is very simple and doesn't support text effects (dynamic reflections, shadow volumes, etc.). To get text effects, you'll need to use the TextMesh Pro plugin developed by Digital Native Studios. It generates a volume mesh that enables you to apply various effects to text and even create animations. TextMesh Pro was a great success, so Unity acquired it and made it available for free with Unity's main editor from the 2018 version onwards.

Game localization in Unreal Engine

Unreal Engine has done a much more thorough job of solving localization problems than Unity. It has had its own standard localization tool for a while, so there's no real need for third-party plugins.

It's worth bearing in mind that Unreal Engine distinguishes between localization and internationalization. Localization involves adapting a game for a specific region or language by translating the text and adding components specific to the locale, while internationalization refers to developing a game in such a way as to make it easily adaptable for different languages without changes to its architecture.



Localization in Unreal is built around a data type called FText, while internationalization uses the ICU (International Components for Unicode) library.

Unreal refers to the set of rules for formatting text in a particular region as a “culture”. Cultures include rules for:

- Handling numbers and special symbols (currency signs, percentages, decimal separators, etc.).
- Formatting dates and times.
- Using plurals.

Each culture has a code consisting of between one and three parts:

- An obligatory two-letter ISO 639-1 code (such as “zh”).
- A supplementary four-letter ISO 15924 code (such as “Hans”).
- A supplementary two-letter ISO 3166-1 country code (such as “CN”).

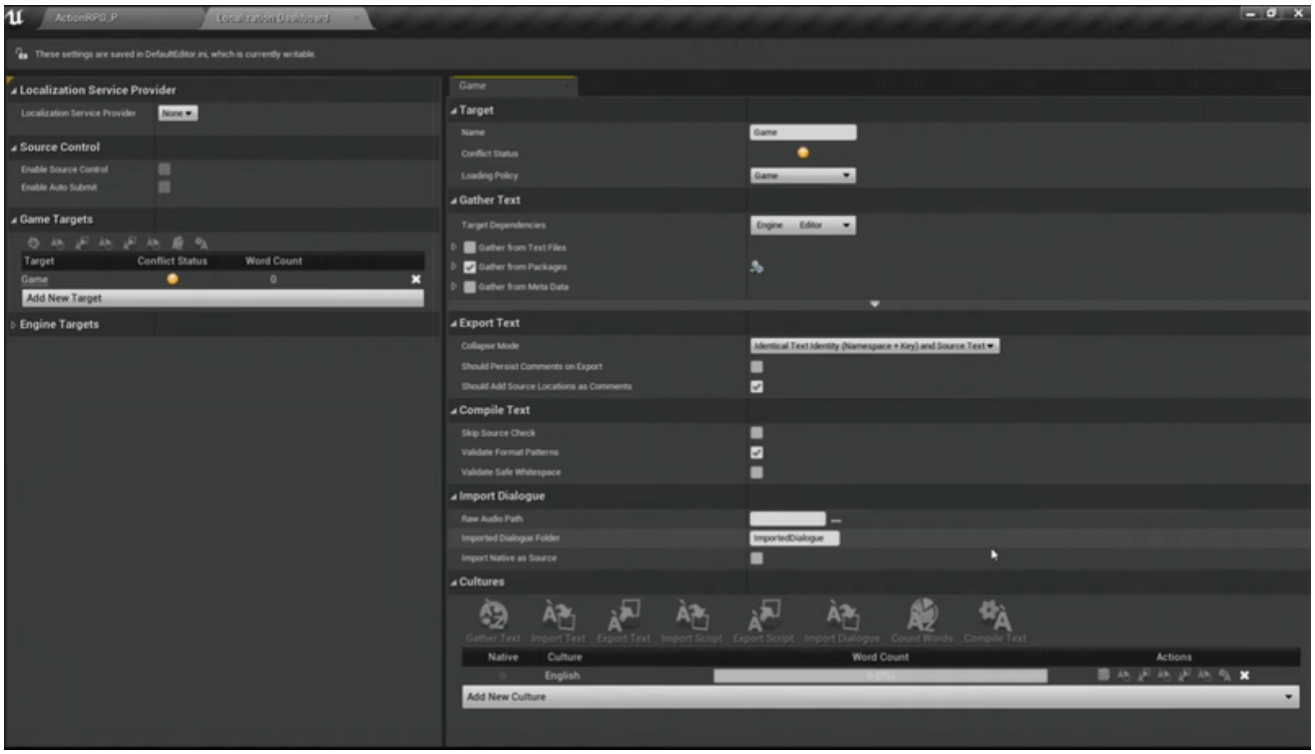
This gives us “zh-Hans-CN”, where “zh” means Chinese, “Hans” means Hànyǔ Simplified, and “CN” is the variant of Chinese used in the Republic of China.

The ICU (International Components for Unicode) library:

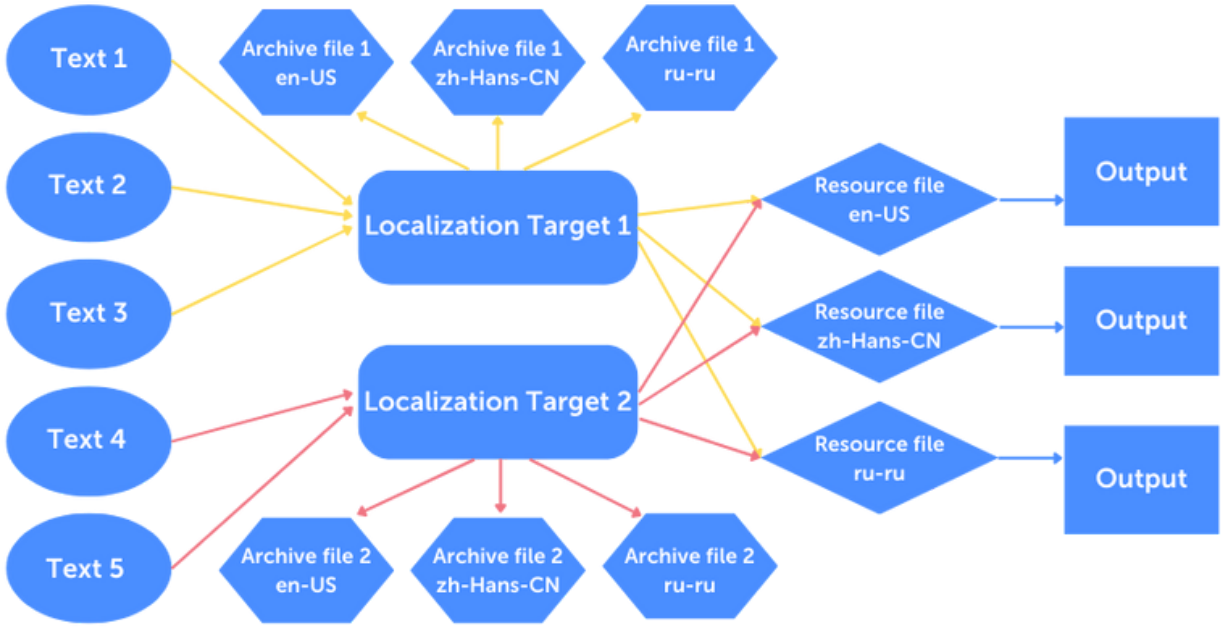
- Asks the operating system what culture to use.
- Identifies the correct text direction.
- Analyzes any internal restrictions in the text (e.g. string wrapping).

Localizable objects in Unreal can include text (or rather text and fonts), textures, audio and video files, cutscenes, scene sequences, and animations.

Unreal calls its localization interface the Localization Dashboard.

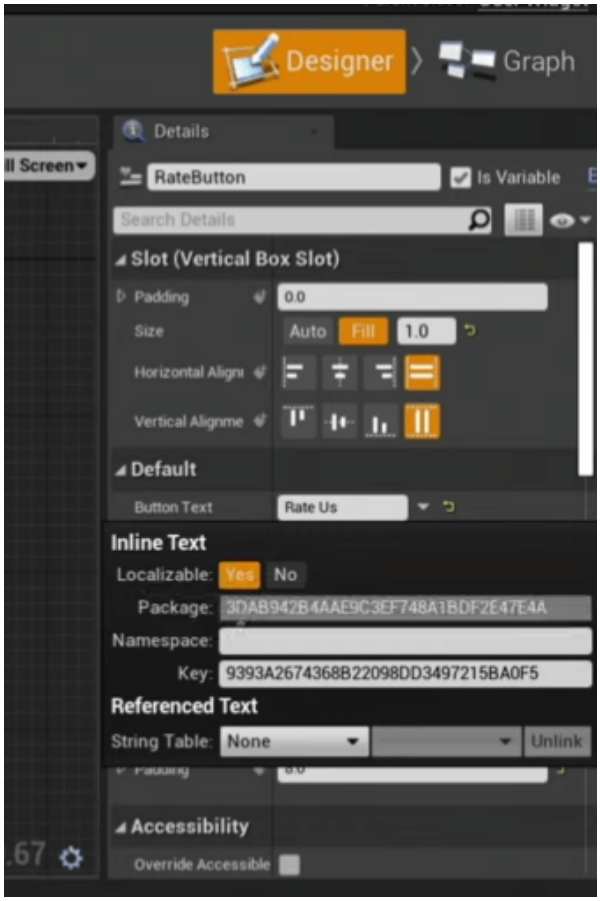


Unreal's term for modules containing localization data is Localization Targets. These modules contain text taken from a specified set of sources. They are stored in a manifest file, translated in culture-specific archive files, and compiled into culture-specific localization resource files, which are what the game displays. It all looks a bit like this:

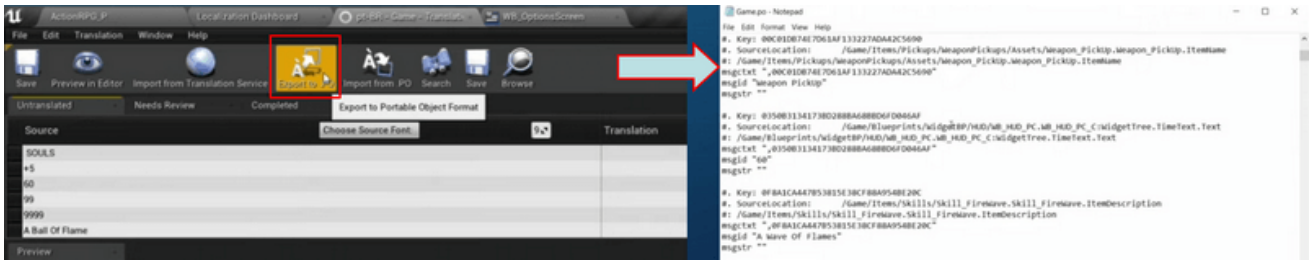


Yep, even in Unreal, localization is a tricky business.
The localization process in Unreal generally works like this:

- Text elements (a button with text, a box, etc.) are added to the game.
- A list of cultures for which localization is needed (en-US, ru-RU, etc.) is added.
- Localizable text elements are brought together in the localization system.
- An FText object is then created for each UI text element. This isn't just a string of text: it has a special structure that allows you to change cultures on the fly. When an FText object is created and the Localizable parameter is set to Yes (as it is by default), the text element is assigned a key that looks like this: "9567B129548DD2468752BA0F5". This key allows the text element to be identified during localization.



- Unreal's localization system creates a list of localizable FText objects in the game.
- The localizable UI elements are exported as .po files, translated, and re-imported.



But how does Unreal handle text effects? In 2019, the engine acquired a plugin called Text 3D, which lets you create masterpieces like this:

Text 3D supports font effects such as shadow volumes, beveled and rounded character edges, different distances between symbols/words/strings, proportional scaling, horizontal and vertical alignment, different fonts, letter-by-letter animation, on-the-fly changes to animated text, and much more. Just one caveat: as of 2022, Text 3D is still in beta, so we don't yet know how much of its functionality will make it into the final version.

Selecting a Market for Expansion

There are several ways to earn more from your game. One of the most obvious ways is to release your game in new markets, which is what we'll dive into now.

Before calculating an estimate, most localizers will conduct an analysis to show which markets can be most profitable. This refers to language markets, since translating into particular languages allows you to release a product in several countries simultaneously.

It may be a surprise that the English language comes in second here. In the mobile games market, Simplified Chinese takes the first place. China's large population population and true passion for mobile devices among all ages are not the only reasons for this. It's also due to the willingness of players in China to spend money on games. However, there are downsides to such a promising market. First, China doesn't have a singular platform like Google Play, so you'll need to deal with several platforms all at once. Second, China's government places significant legal barriers for foreign developers.

Evaluating a country's willingness to pay is also important. A game released in geographically tiny Taiwan and Hong Kong will likely earn more than the same game released in the massive country of Brazil (especially considering Brazilian gamers' notorious digital piracy).

There's one more thing worth paying attention to when choosing languages for localization: the prevalence of the English language. For example, users in Scandinavian countries are willing to pay. They are ready to buy games and make in-game purchases. At the same time, many Scandinavians speak English fluently. It's unlikely they'll refuse to buy a game simply because it hasn't been localized into Swedish, Danish, or Norwegian.

Therefore, when choosing a market to launch your project in, it's important to note the following:

- The number of people who speak your target language;
- The income level in your target country;
- The level of English fluency;
- Any legal barriers for entering the market;
- The popularity of this genre in your chosen country;
- The competitive landscape.

Localization

This is the biggest section of our book. This is the stage at which the localization company works the closest with the client, helping to create a quality translation before the game's release.
There are several ways to localize a game: hire in-house employees, find freelancers, translate using a fan community, or contact an agency. All those methods have their own pros and cons but, as always, it all comes down to the classic project management triangle.

If you want complete and flawless texts in all the languages you need, contact a reliable localization studio. This will cost more, but you'll have less stress and minimize the chances of needing to redo everything.
Next, we'll tell you the best way to comfortably outsource with a studio. Even if you choose a different route, the general principles and approaches we share in the section will help optimize your localization tasks.

Which Language Should You Translate Your Game Into?

A localizer's standard language set includes major European languages (EFIGS: English, French, Italian, German, and Spanish) and major Asian languages (Chinese, Korean, and Japanese). After that, there's demand for translation into Russian, Polish, Hindi, Indonesian, Arabic, and Turkish. If you're not ready to experiment and invest into localizing the game into Afrikaans and promoting it in South Africa, then it's best to stick with the standard set of languages.
We publish yearly gaming market research for developers. Here are 2022's top 15 languages that generated the most income for developers:

Language	Population	Gamers	Revenue
English	946 931 460	392 485 000	$51 851 149 000
Simplified Chinese	1 450 000 000	744 000 000	$45 800 000 000
Japanese	125 800 000	77 100 000	$20 000 000 000
Korean	51 320 000	34 100 000	$7 930 000 000
German	83 890 000	49 500 000	$6 620 000 000
French	73 920 600	43 640 000	$4 896 800 000
Spanish (Mex)	35 727 300	22 470 000	$4 815 000 000
Italian	60 320 000	36 100 000	$3 020 000 000
Portuguese (Bra)	214 700 000	103 000 000	$2 610 000 000
Traditional Chinese	34 901 360	24 638 600	$2 440 913 000
Spanish	46 730 000	32 300 000	$2 270 000 000
Hindi	1 250 620 000	335 880 800	$1 866 997 000
Indonesian	277 700 000	138 000 000	$1 800 000 000
Arabic	45 630 000	31 490 000	$1 413 000 000
Turkish	85 590 000	46 500 000	$1 120 000 000

Another obvious way to choose a target language is to learn from your competitors. You can look at the languages used by successful games similar to yours and duplicate their strategy. In the case of mobile games, you can use automated analytical services, like AppMagic.

How Much Does Localization Cost?

So, how are localization costs and time frames calculated? We get all sorts of questions like: how much does a project cost? A symbol? A word or Excel file? An A4 page? What about a mobile game or desktop game? It's impossible to answer those questions on the spot. Cost and time frame are always based on the number of words in the project (the exception is Asian languages, for which the cost depends on the number of characters in the source text). However, the actual text is needed to determine an exact price and time frame. The text provides us with the information needed to:

1. Assess the text's complexity and artistry.

2. Calculate the number of repetitions in the text. Each localization company handles repetitions based on their own policy. Some companies subtract 100% of the cost per match or offer a large discount (up to 90%). Other companies rely on a match table and set separate rates for each type of match.

3. Evaluate available resources. A task may take longer to complete during busy periods, and contractors are sometimes unavailable during holidays. For example, during the Lunar New Year in February, Chinese and Vietnamese translators take a break.

4. Assess the feasibility of working in a particular language pair. Many languages are localized from English. Some language pairs like Maltese-Icelandic are difficult to work with, especially if you need a professional translator with a background in game localization. The chances of finding such a specialist are slim, and if you manage to actually locate them, it won't come cheap.

5. Offer discounts for volume (some companies).

To provide you with a rough budget estimate, here are the average rates for common language pairs (please keep in mind each company sets their own rates):

- English to French: 0.13—0.14 USD;
- English to Italian: 0.13—0.14 USD;
- English to Spanish: 0.13—0.14 USD;
- English to German: 0.13—0.14 USD;
- English to Chinese: 0.13—0.15 USD.

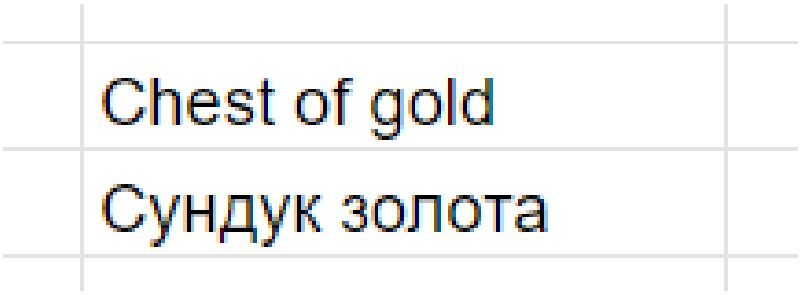
Note that these are just approximate prices, and rates may vary considerably from company to company.
Localization costs usually include the translation, proofreading by a second linguist, and automatic error checks. Some companies will require an additional payment if you need not only translators but also a project manager. Certain companies have a minimum order requirement. Also, you should find out the bank fee for transferring money and clarify whether that fee is covered by you or the service provider.

How to Make a Good Lockit?

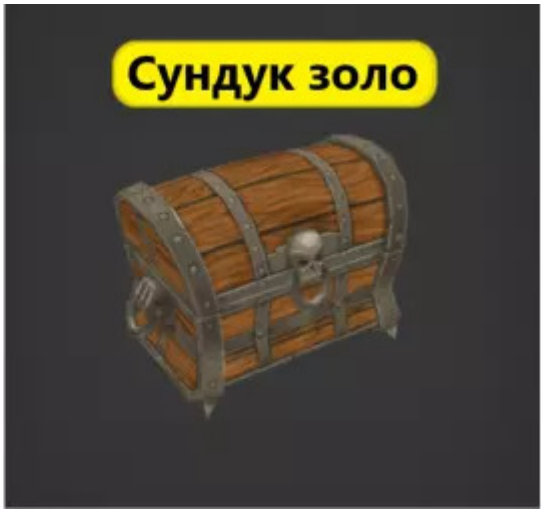
Before starting a job, a localizer will always ask the developer for a lockit. It's a document that contains all in-game texts and in-game information.
Sending a usual TXT file (containing a dump of all strings in no particular order with technical IDs and placeholders sprinkled here and there) is not recommended. Even the most experienced specialists have trouble working with such documents. It might result in mismatched dialogs, buttons and messages taken out of context, code broken by excessive translation, and other unpleasant surprises.

However, it is possible to use even a TXT if it's split into logically ordered segments and the IDs are located above or below the segments. In the current era of CAT-tools, the best file formats are XLS, XLM, DOC, RTF, HTML, CSV, and XLIFF.
Table formats are convenient because the information is clearly structured, which works great for game texts. However, if you plan to upload the text to a website, then it's best to choose HTML format or even a text document. That way the translator can immediately see the format and layout of the text on the page.

What should you include in the document?
Of course, include the original text in a logical order. As a side note, even if you plan to translate only part of the text, it's best to submit the entire lockit and indicate which portion you'd like translated. That will ensure the translator has the appropriate context. Avoid sorting the text alphabetically, by string length, or by other parameters, since this will break its structure.
The text can be accompanied by string identifiers, especially if they contain meaning, and game sections in which the strings will be used (for example: "Store", "Chat", "Inventory", "Graphics Settings").
If the game's text field has a symbol limit, please indicate this. Please keep in mind that fonts can differ. When comparing the same number of symbols in Russian and English, the Russian version will likely be longer. Compare these two phrases that are both 13 symbols:



Here’s what can happen:



If there’s risk of text not fitting due to such an issue, the limit can be specified using letters. For example, «13 A» means that the capital letter A will fit exactly 13 times in the string. It takes longer to compare a translation to this kind of limit, but the chance of having to redo the text is reduced.
For example:

Chest of gold
Сундук золота
AAAAAAAAAAAA

We see that 13 characters in one language are not the same as 13 characters in another. IBM has more info on this topic in their guide to creating user interfaces.

Overall, try to add as many comments to strings as possible. Even adding a simple comment like “Button” to the word “Download” helps a translator choose the correct part of speech and word form. If a game contains a lot of dialogues, then it’s essential to include comments that provide context. For example, comment on who is speaking, who is being spoken to, the gender of the participants, the situation itself, and the sequence numbers of replies if they’re not in a linear order. Who wants a cute sorceress to suddenly be addressed in masculine form? Or a dialogue to turn into gibberish because the strings are out of order? This is why it’s good to give localizers a heads-up if you change text or rearrange any strings. Otherwise, something like this may happen...

	A	B
	I'm going to visit Granny. She must have come home already, right?	Я собираюсь к бабушке. Она ведь наверняка уже дома.
1	Feel like tagging along with me?	Хочешь со мной?
2	Suppose so.	Хочу, наверное.
3	↓	↓
4	I'm going to visit Granny. She must have come home already, right?	Я собираюсь к бабушке. Она ведь наверняка уже дома.
5	Suppose so.	Хочу, наверное.

And since we’re already talking about dialogs, let’s go over one more component of a good lockit: descriptions. Descriptions of characters and their relationships can help translators better understand a character’s personality, status, and goals as well as choose the correct form of address and the most suitable style of speaking. A description of the game world provides insight into its mechanics and the environment surrounding the characters. It also helps avoid things that don’t make sense. For example, a character wouldn’t use the expression “sick as a dog” in a world where dogs don’t exist. Describing game mechanics is helpful too, since players need a clear explanation of what certain actions lead to and how they can achieve their goals. In short, share your vision, your associations, and your inspirations. This will help localizers be in tune with what you’re thinking.

Don’t forget about the visual component. The game lets players customize characters? Then share images of equipment, weapons, and skins. There’s a hidden object search in the game? Then share images of the objects. The game has locations with poetic names? Then share images of the locations. In short: the more images, the better! Otherwise, you’ll get lots of questions like “Is the ”bow” in string 15 a weapon or a hairpiece?” or bugs like this:

The same goes for videos. Do you need subtitles translated? Piece of cake, just send the video clip.

“But that takes so much time!” a developer might complain and instead just send the game build to a localization company with the words “here’s everything, do what you need to do.”

The build provides valuable supporting material. Playing the build gives an overview of the game world, the structure of the interface, the size of the windows, and the position of the text. However, there's a catch. While the build is very helpful if a game is small and linear, it can only provide a general overview of a complex game (even if you share cheat codes). Also, certain windows appear only in specific circumstances that may be hard to replicate (for example, error messages or purchase notifications). So don't be surprised if the translators bombard you with requests to explain a particular name or show how a certain window or item look like.

The more info you can share with the localizers about your project, the quicker they can provide you with a high-quality localization. It will also save you time. It's better to take screenshots early on than to hastily answer a bunch of questions later—and still have to take the screenshots anyway. No one wants to repeatedly get incomplete files, import edits, and track down bugs. Best to save yourself time, money, and sanity. A good lockit makes it possible to create a unified, beautiful picture from lots of little bits and pieces. This can then be used to easily translate the app into other languages. What could be better?

AI and Localization: Friends or Foes?

ChatGPT is a heated topic at the moment, particularly with thoughts on how it can affect the localization industry:

- *It will revolutionize our approach to handling texts!*
- *It will make localization cheaper!*
- *It will make the interpreter's job obsolete!*

For the sake of completeness, let's substitute "ChatGPT" with "AI" or "Machine" and consider the implications of these new technologies. We're going to go over the things the machine can and cannot do. The list of reasonable "cans" is quite short, so we'll begin with those.

Consistency

There is one thing that just can't be taken from AI—it can preserve the consistency of the texts, though sometimes in a somewhat overzealous manner.

Now onto the "cannots."

Style

"Style expectations" is a term invented to create an illusion of standards. With localization, each linguist's style is unique and can't be replicated accurately and consistently (not yet, ChatGPT, don't give me that look). Of course, there is an argument to be made that if a standard is ephemeral, then a machine could just as well create its own norm and stick to it. Sounds about right? Well, not quite.

You can prompt AI to include informal words, local idioms, words starting with the same letter, and so on. You can even make a glossary of terms that must be used in every sentence. But how would you prompt it to be annoying? That's a thing humans excel at, so it makes sense they (we mean, WE) would be able to reflect that effectively in a translation.

Let's imagine a game where all your actions are accompanied by the comments of a pestering companion that you just can't get rid of:

EN: Experience gained: {0}. Got smarter, did ya?
SPA (LatAm): Experiencia obtenida: {0}. ¿Ya te sientes superior?
FR: Expérience acquise : {0}. Vous vous coucherez moins bête ce soir!
DE: Erfahrung erhalten: {0}. Hältst dich wohl für ganz schlau, was?
EN: Quest completed! Satisfied?
SPA (LatAm): ¡Misión cumplida! ¿Ya estás feliz?
FR: Quête terminée ! Tu es content de toi ?
DE: Quest abgeschlossen! Hast du ganz doll gemacht!

Euphemisms

It takes a lot of time and skill to find an obscure yet clear word to effectively convey an intended meaning.

Mr. Sanderson, allow me to use an example from your brilliant Stormlight Archive.

Imagine a world where obscenities took a form that is different from ours. Common swear words are not being used, instead they are substituted with “storm” and its derivatives. Or, should the speaker be a child, the verb “starve” would make for a milder replacement.

Want to have a guess at what is encrypted in this seemingly innocuous phrase?

EN: Storming fool!

SPA (LatAm): ¡Necio de las tormentas!

FR: Crétin des foudres!

DE: Sturmverdammter Narr!

As much as we trust AI to handle a passable, straightforward translation, having this localized as “a raging knucklehead” would certainly NOT raise the book’s appeal. This part of the text should be handled by a person in 100% cases.

Puns

Translating texts is good, localizing is better. And a clever pun squeezed in a wall of text never fails to spike a reader’s interest.

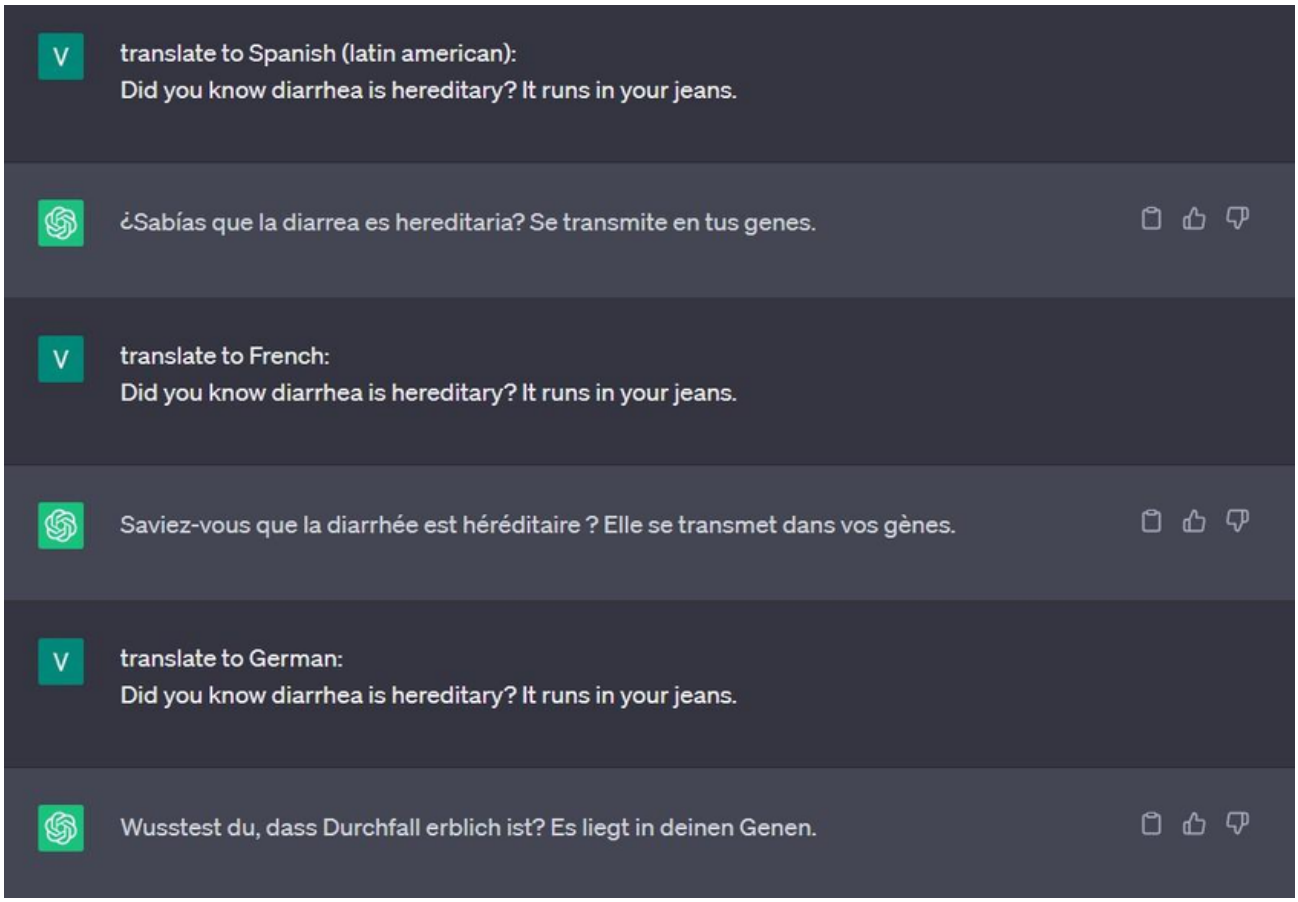
EN: Did you know diarrhea is hereditary? It runs in your jeans.

SPA (LatAm): ¿Sabías que hallaron el gen de la timidez? Sí, estaba escondido.

FR: La diarrhée atteint plus de gens qu’on le croit. C’est une maladie courante.

DE: Was steht auf dem Grabstein eines Mathematikers? Tja, damit hat er nicht gerechnet.

A machine wouldn’t blink an eye (or is that screen?) and would translate this to the letter. Don’t just take my word for it, here is a screenshot from ChatGPT.



This couldn't have been any further from the original meaning. We can't blame AI though—this play on words can be a tough nut to crack even for an experienced linguist. And this is exactly where human ingenuity shows its strength—an untranslatable joke can be substituted with a similar one without losing the intent of the original. Flexibility is the key!

Context

No, this won't be about lack of context in one-word lines. Try getting the correct meaning of "Order" on a first attempt. Unless the client provides an explanation, a linguist is as likely to be wrong as a machine.

What we're referring to here is the transparent context. For a human, of course. Take a look at this sentence.

EN: Do you know that humans eat more bananas than monkeys? Bananas are just tastier.

Sure, a really smart AI would've already learned this joke by heart (core?), considering the joke's age, but let's pretend it's a brand-new joke. We've seen people getting confused at this even after a hint. For AI to nail this translation, it would have to get a manual cue. By who? That's right, by a human.

Of course, some of these points are exaggerated for fun's sake. We're not trying to talk anyone out of using machine translation in its entirety. It's quick, it's cheap, it gets the job done. But it's flawed in its current state, and using it blindly will inevitably lead to negative feedback from the players, which is exactly the thing all developers and, by extension, Language Service Providers are trying to avoid. The point of this article is to illustrate the hilarious outcomes AI can deliver if used as a blunt tool without supervision of an experienced linguist. In the end, the choice around AI's use is totally up to you. After all, what good is having a tool that collects dust on a shelf without even attempting to use it?

Fonts in localization

Each alphabet has its own peculiarities. Let's go over what to consider when working with different writing systems.

Non-Latin writing systems

Your game will almost surely need fonts based on different writing systems: Cyrillic, Greek, Armenian, Georgian, etc. However, it's unlikely that all symbols of the required languages will fit in an atlas (a file with font). Trying to do this will result in lower quality, since the atlas has a limit. There is a simple solution. Combine the symbols in a font resource according to their writing system (Latin, Cyrillic, etc.) and then combine these groups later. Create a master font resource that includes all Extended Latin characters. Then get a font resource for Cyrillic and any other writing systems your game needs. These extra resources are called "fallbacks". When an unknown symbol is encountered, the game engine will check the main font resource. If nothing is found, the game engine will then check the first fallback resource, then the second, and so on.

Logographic writing systems

Some languages are written using logograms—characters that represent sounds and syllables, as well as morphemes, words, and entire concepts. One might think this isn't a big deal to a developer, right? All you need to do is create and use a character-based font. But there's one thing to keep in mind about all logographic languages: they have lots (and lots!) of characters. For example, Japanese actively uses about 3,000 characters. Chinese uses around 6,500 characters (and the Zhonghua Zihai, the largest character dictionary, contains 85,568 characters).

Quantity isn't the only issue at play here. From a technical point of view, the characters themselves are big. The English symbol encoded in UTF-8 usually takes up 1 byte, but the Chinese character requires 3 bytes. Therefore, to display a screen of beautiful Chinese characters, the device on which your game is running will need more hardware resources. This means the game's performance might suffer, which will surely disappoint your players.

In order to solve this issue of quality and performance, character sets are usually divided into categories.

Chinese is usually divided into three categories based on usage: the top 3,500 characters, a set of less commonly used 3,000 characters, and then the remaining rarely used 1,605 characters. First, a master font resource is created. It contains all the characters used in the game. Then, a resource is created for each category of characters. Upon encountering a character, the game will systematically search for it in each resource based on order of frequency.

A similar approach is ideal for Japanese characters. The only difference is that it's better to divide them into categories by written form rather than by prevalence. These forms are: hiragana (46 unique symbols that form the basis of Japanese writing), katakana (46 unique symbols used mainly for foreign words), and kanji (2,136 Chinese characters used in Japanese).

Right-to-left writing systems

Certain languages are written from right to left. This isn't an issue by itself. However, many developers are surprised to learn that for languages whose written forms are based on Arabic, the letters within a word must connect to each other in a special way. If this isn't taken into account, then the phrase «New game» will appear in Arabic as **ة د ي د ج ة ب ع ل** when it should be **لعبة جديدة**.

This means that the game engine needs to understand where a letter appears: at the beginning of the word, in the middle, at the end, or if the letter stands alone (see table). This problem can be solved in different ways by different engines. For example, the RTLTMPro plugin has been developed for Unity.

Hebrew, Arabic, Thaana, and certain other Middle Eastern and Asian languages are bidirectional. This means they can be written both right- to-left and left-to-right.

If you'd like to localize a game into such a language, there are several things to keep in mind during development. We'll take a look at Hebrew, but these points also apply to the other languages as well.

— A combination of numbers and letters.

In Hebrew, letters are written from right to left while numbers are written from left to right. If you don't account for this aspect of the language, then when letters encounter numbers, the letters might display in the wrong direction. Or the numbers will end up being reversed.

— Strings should start from the right.

1) This is how we write in English.

.כך כותבים בעברית

2) It shouldn't be like this in Hebrew.
זה לא בסדר בעברית.

In the screenshot, the first line is written correctly, but there's a mistake in the second.

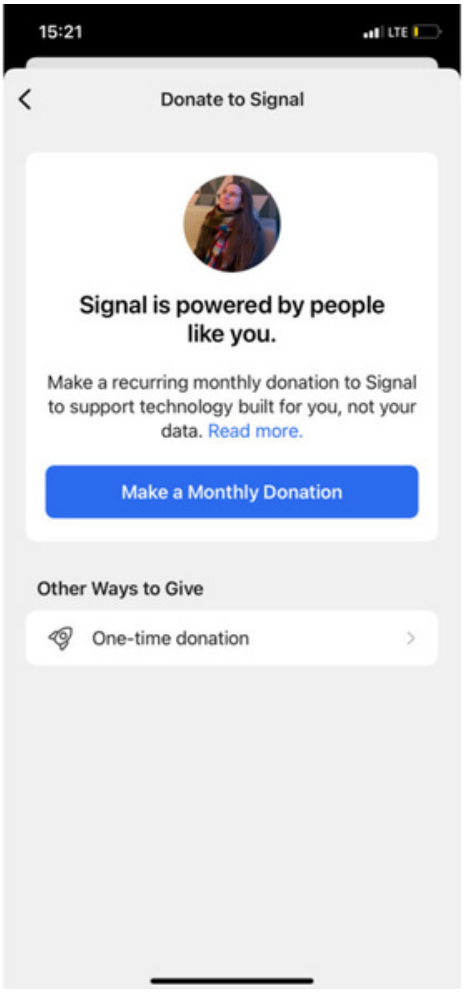
Additionally, the interface itself should be mirrored to make it familiar to the user. For example, the leftmost column in a table should become the rightmost column in a Hebrew table. Meanwhile, in music apps it's more convenient to tap the left arrow for the next song.

— A combination of Latin/Cyrillic alphabets and a right-to-left writing system.

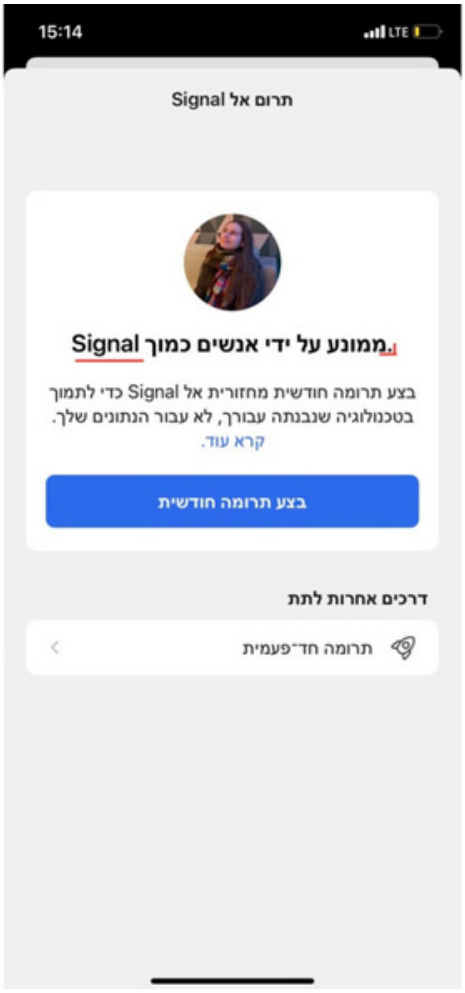
If the text contains both Hebrew and English words, things might get a little tricky. Possible bugs include:

- A period at the beginning of a sentence instead of the end;
- Left-aligned text;
- Mistakes in the sentence word order;
- Right-to-left text displayed in the opposite manner;
- Symbols displayed improperly, as in a string of question marks or boxes.

Let's take a look at how the messenger app Signal was localized into Hebrew.

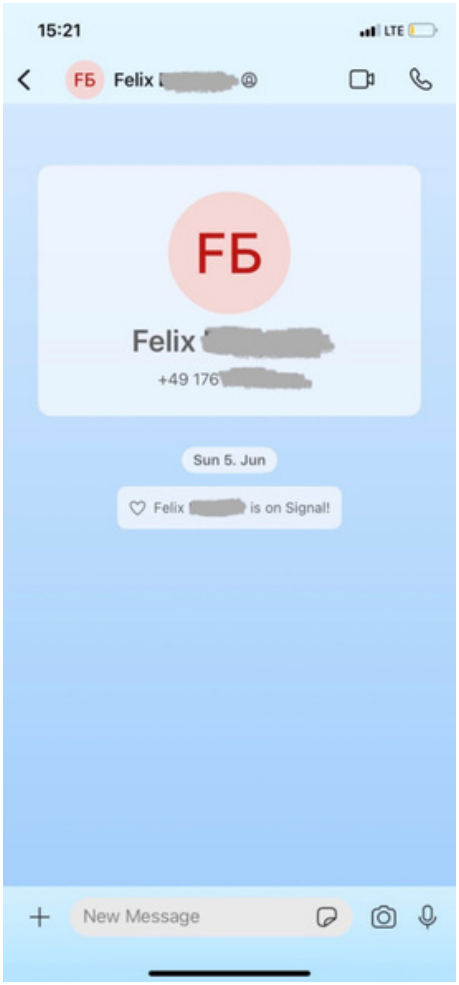


The page in English.

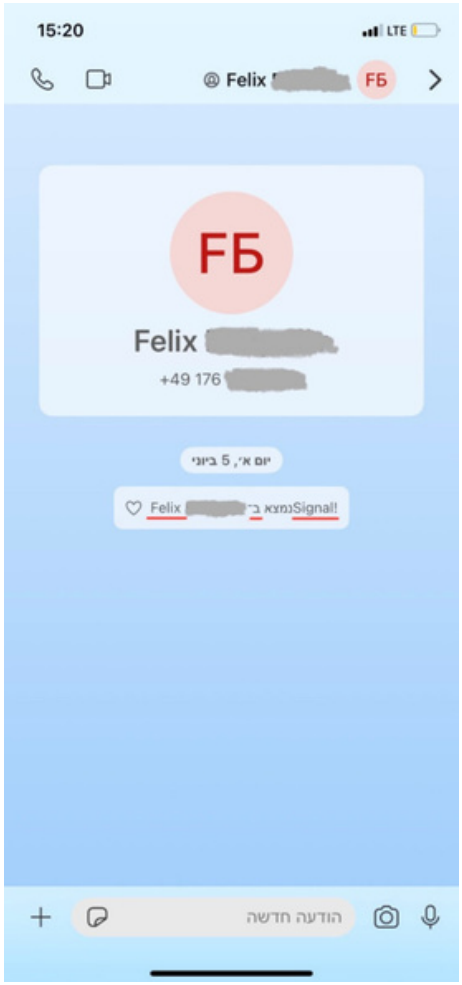


The same page in Hebrew.

The developers mirrored the interface and made it right-aligned, but they failed to consider the compatibility of Hebrew and the Latin alphabet. Because of this, the word “Signal” moved to the end of the sentence, while the period moved to the beginning.
Here’s another example.



The English version.



The same window in Hebrew.

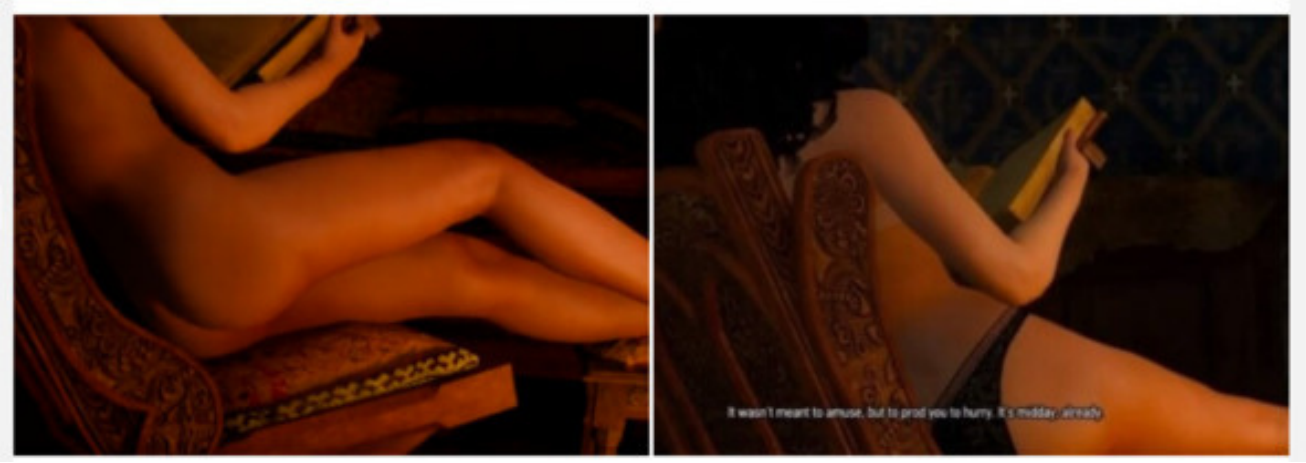
It has been mirrored and the numbers are displayed correctly, but the words have been mixed up again and are out of order. That is, “Felix” should be on the right, at the beginning of the sentence, and “Signal” should be on the left, after the preposition. What’s more, the exclamation sign appears at the beginning, and the space between words has vanished. As a result, the English translation of what appears in the screenshot above is “Signalis on Felix”.

If you keep language localization quirks in mind while creating your app and test your product before release, then you can avoid these problems. The results will be a beautiful app and proper text.

Culturalization

All humans belong to one culture or another. The way we brush our teeth, the way we look, the places we go, the things we like—all this makes us bearers of a specific culture. Belonging to a particular culture isn't only about history, art, and politics; it's the sum of all the details that make up our distinct reality. When we speak about developing video games, we're referring to the creation of unique worlds and realities. However, almost all of these worlds and realities will refer somehow to existing cultures.

Let's start with a little theoretical background. Localization can be reactive and proactive. Reactive localization means removing things from the game that can have a negative effect on a player's experience. This is the minimum action required for the user to perceive the game in the intended way. For instance, it was necessary to put Yennefer in underwear in versions of The Witcher released in Japan and the Middle East.



Proactive localization means adding something pleasant and known to the player. For this purpose, there are special expansion packs for certain languages.

The Chinese version of Plants vs. Zombies

What should you pay the most attention to?

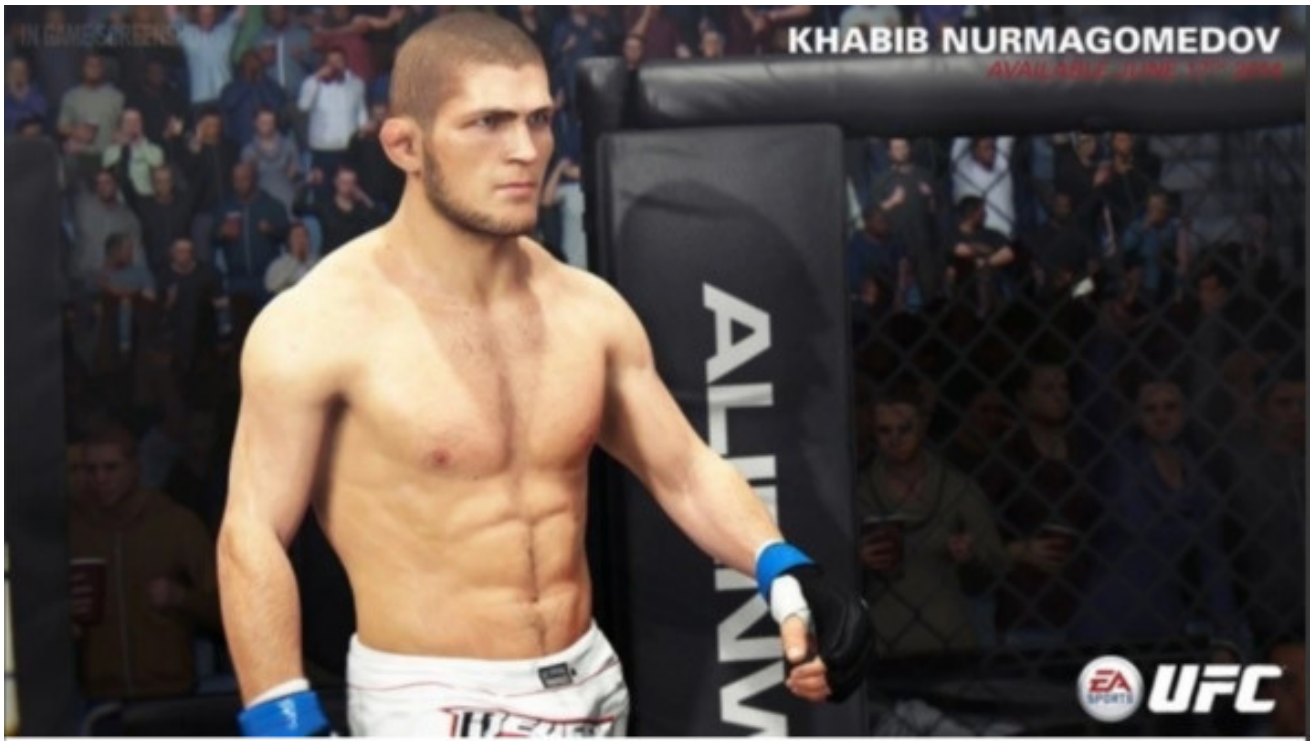
Politics and history

These subjects are especially sensitive in regions that have recently experienced a conflict.

For example, the Korean version of Age of Empires is quite different from the game available to other users. This is because, according to the plot, Japanese armed forces invade the Korean Peninsula and seize power. This event takes place back in the Middle Ages and does have a historical basis. However, the Korean Ministry of Information declared that no such event ever happened. This stance contradicts historical documents that claim the opposite.

Faith and religion

Matters of faith and religion are such delicate issues that people often avoid discussing these topics even with loved ones. If we're trying to successfully enter a new market, we must do our homework and respectfully deal with any little thing that can affect the player's feelings. In a game based on the Ultimate Fighting Championship brand, fighter Khabib Nurmagomedov makes the sign of the cross. What both the developer and the publisher disregarded is the fact that the fighter is Muslim in real life.



As shown in the tweet, Khabib appealed to EA and asked to set the record straight, saying that he and many of his fans are Muslims and his detail should be respected.

Laws (what a surprise)

Laws are a fact of life that can't be ignored. For example, if showing alcohol, cigarettes, and drugs is forbidden in Russia, no one will display them in ads there. Similarly, the laws of other countries must be respected. The strictest censorship laws of our times are enforced by the Chinese government. They check not only the games themselves but also updates and major expansion packs.

Developers resort to dubious tactics to get around these issues. For instance, things might change in appearance, turn black and white, or even completely disappear.

Speaking of market leaders, we would be remiss to not mention the US. As they say, America is a free country, which is exactly why developers have to accommodate so many restrictions. As surely everyone is aware, discrimination (whether it is based on gender, race, or religion) is a hot topic these days in the US. Games released into the US market must try to be as inclusive as possible.

This screenshot shows how developers had to entirely redraw a scene because of a tiny detail. This is Lenora, the only dark-skinned character in Pokémon. No one raised an eyebrow at her outfit in the Japanese version. However, American audiences were outraged by the girl's apron, which led to Lenora donning a regular shirt instead.

Traditions and beliefs

It's widely known that Japan is a rather unique place. One strange thing you should know is the opposition of the Japanese government to four-fingered heroes:

There are several possible reasons for this. According to one explanation, the number 4 is considered unlucky in many eastern cultures (even going as far as not having hospital rooms with this number and skipping from floor 3 to floor 5 in residential buildings). Another possible explanation has to do with yubitsume, the act of cutting off a finger as a sign of loyalty or as a punishment in criminal groups like the yakuza.

Over-culturalization

Sometimes part of a game's charm is that it shares a developer's culture with the world. If a Japanese game switches out sushi for hamburgers and a kimono for jeans, players might be disappointed and won't hesitate to share this in reviews.

There haven't been many examples of this happening. Most developers and publishers understand that it's not a great idea to plop a furry hat on Godzilla and put a balalaika in its hands. Perhaps the most striking example of this was an attempt to assimilate Spider-Man in India (though, in the comics, not in the game).

So, what's the result?

While money motivates some people to work in the gaming industry, others see their work as a commitment to art. A considerable number of talented indie developers don't want to conform to regional conventions, explaining that it would ruin their vision of the game.

As a process and as a service, culturalization can't force anyone to comply. Developers can still do what they want to do. Culturalization is a tool for identifying which moments in a game might affect the chosen audience. That's why the final decision always rests with the developer.

Learn more about culturalization by reading our Culturalization Guide.

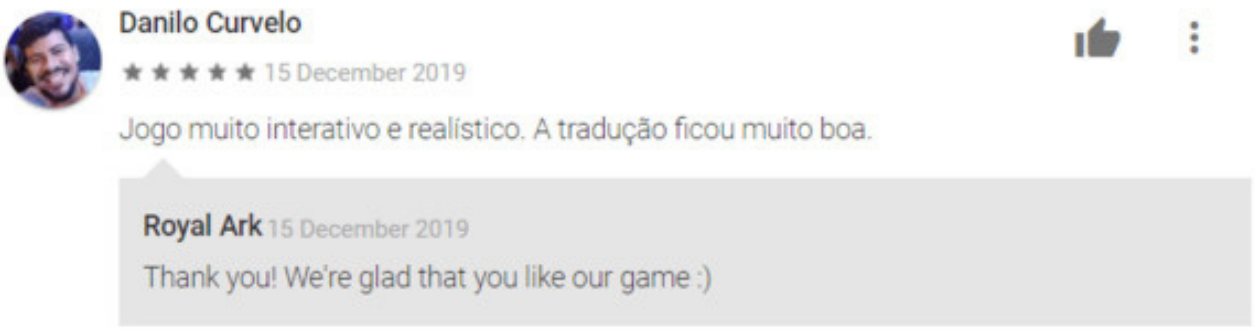
Assessing localization effectiveness

What makes an effective localization? For players, an effective localization is the one they don't even notice (no bugs or major errors).
For developers, it's a localization that helps them earn more.

How to Analyze Localization Quality Using Reviews

Reviews can make it clear what players think of the localization. We implemented an algorithm for working with feedback. This algorithm helps us control and improve the quality of a localization.
Working with player reviews is done in several stages.
— First, the reviews are downloaded from open sources (Google Play, App Store, Steam).
— Next, the results are combined into a single document and sorted. We are interested in reviews involving localization, so only segments that contain phrases like "localization", "translation", "language", etc. are taken for further analysis. Every language has its own set of keywords.
— Then, the collected reviews are translated into English using machine translation. It doesn't provide a perfect translation, but it's sufficient for this task.
— Finally, a "sentiment" analysis is done to find out what players wanted to say with their reviews. They might have been satisfied with the localization, upset over a poor translation, or dreaming of playing the game in their own native languages.
The analysis results in a document containing relevant localization reviews as well as their translation and rating. The project team and client can then use the data.
From there, we sort reviews by type.
So, now we've received a selection of relevant localization reviews sorted into various categories. This is not the final result. We want to use this data to determine a plan of action.

A positive review

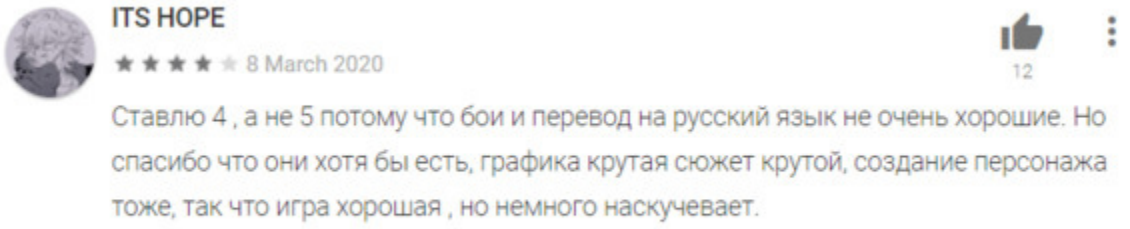


A very interactive and realistic game. The translation was excellent.

First of all, reviews like this make us happy. Second of all, we note that the translation team working on this project (in this case, translating into Brazilian Portuguese) is skilled at translating games of this type. We will invite those translators back to work on similar projects in the future.

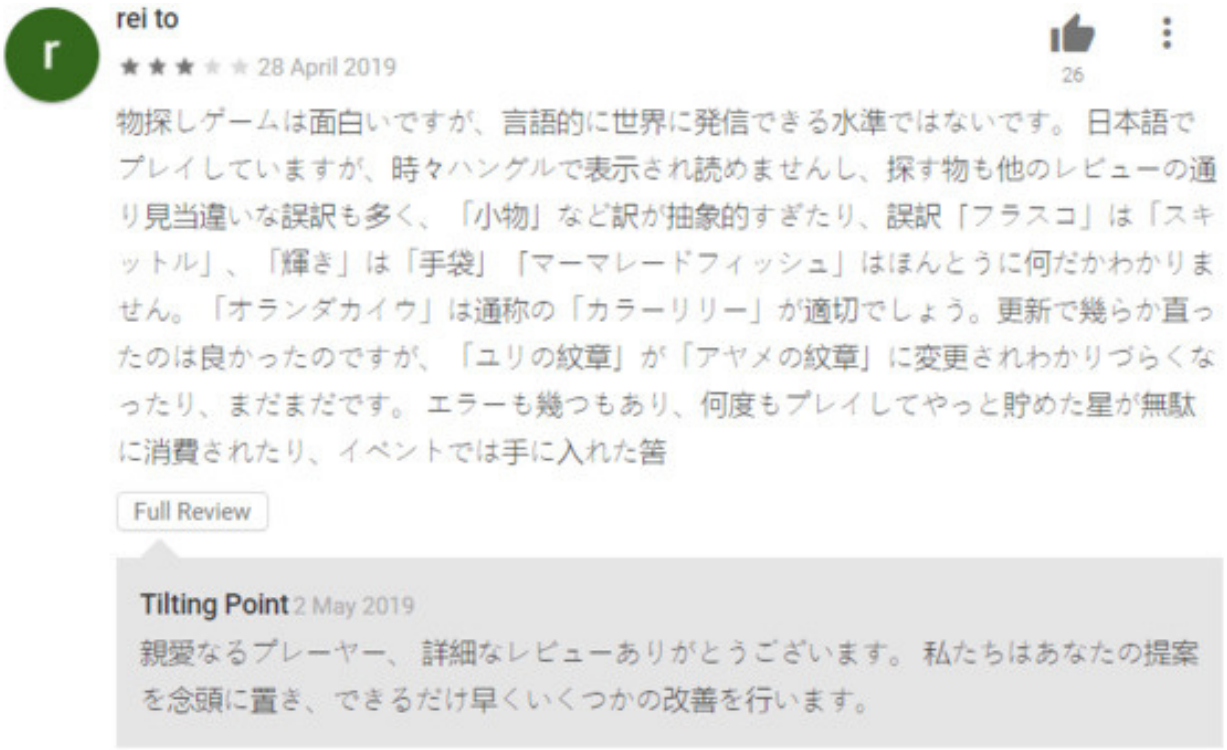
A negative review

Negative reviews are ranked in groups. Some reviews are overly emotional. The user might have been dissatisfied with the overall game (or individual mechanics) so they target everything in their review, including the translation.



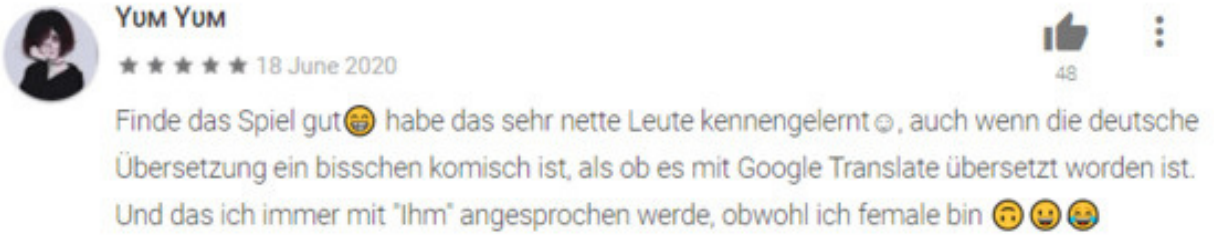
I'm giving it a 4 and not a 5 because the combat and the Russian translation aren't very good. But at least they're there. Cool graphics, cool story, and cool character creation. The game is good but kinda boring.

This does not mean it’s a poor-quality localization, but we want to keep it in mind. The next time reviews are collected (on average, this kind of work is done once every three months) we will need to do an especially thorough analysis of player opinion on localization into the mentioned language.
It’s also possible to receive more than one negative localization review. If a clear trend emerges, then it’s time to start figuring out what the issue is.
In a perfect world, we would ask the developer to get in touch with players and get examples of specific in-game errors (preferably with screenshots), then validate the errors and make adjustments to the localization. In the real world, the developer has about a thousand more important things to do. Instead, we start by trying to locate the problem on our own. For example, we can send the game for LQA (during which it could easily turn out that phrases are properly translated but just don’t fit in the fields or line breaks aren’t formatted properly).
Sometimes (not as often as we’d like, unfortunately) the mistake is described in the review itself. This makes our work easier and takes the edge off the sad fact that there is indeed a mistake.



I play in Japanese, but sometimes the text shows up as Hangul and I can't read it. There are a lot of incorrect translations of the things I'm looking for and in other parts of the game too. The translation of "accessories" is too abstract, and the translation of "flask" is incorrect. I really don't know what "pin", "glitter", "gloves", and "gummy fish" are. The common name "calla lily" should be used for "calla lilies". It's good that some things were fixed in the update, but "Lily Cross" turned into "Iris Cross" and now it makes no sense.

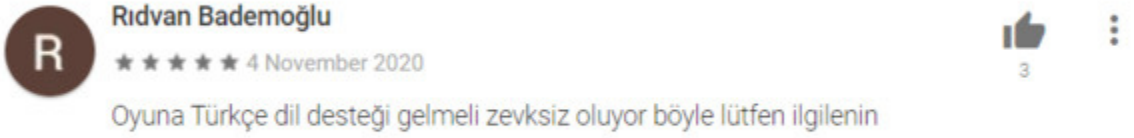
If there are actual errors in the translation, we fix them and send the client an updated translation (this is done for free, as we give a lifetime guarantee for our work). If the problem isn't in the quality of the translation, then we send an email explaining the current problems and suggesting ways to fix them. For example, players might be dissatisfied with the level of formality or unhappy being addressed as the wrong gender.



I think this is a good game. I met great people, although the German translation is a little off, as if it was made using Google Translate. And I was always addressed as "he" in the game, although I'm actually female.

This isn't a translation mistake, but eliminating details that annoy gamers will reward you with a more loyal community and significantly boost players' engagement.

Requests for translation
It's common to see reviews like this:



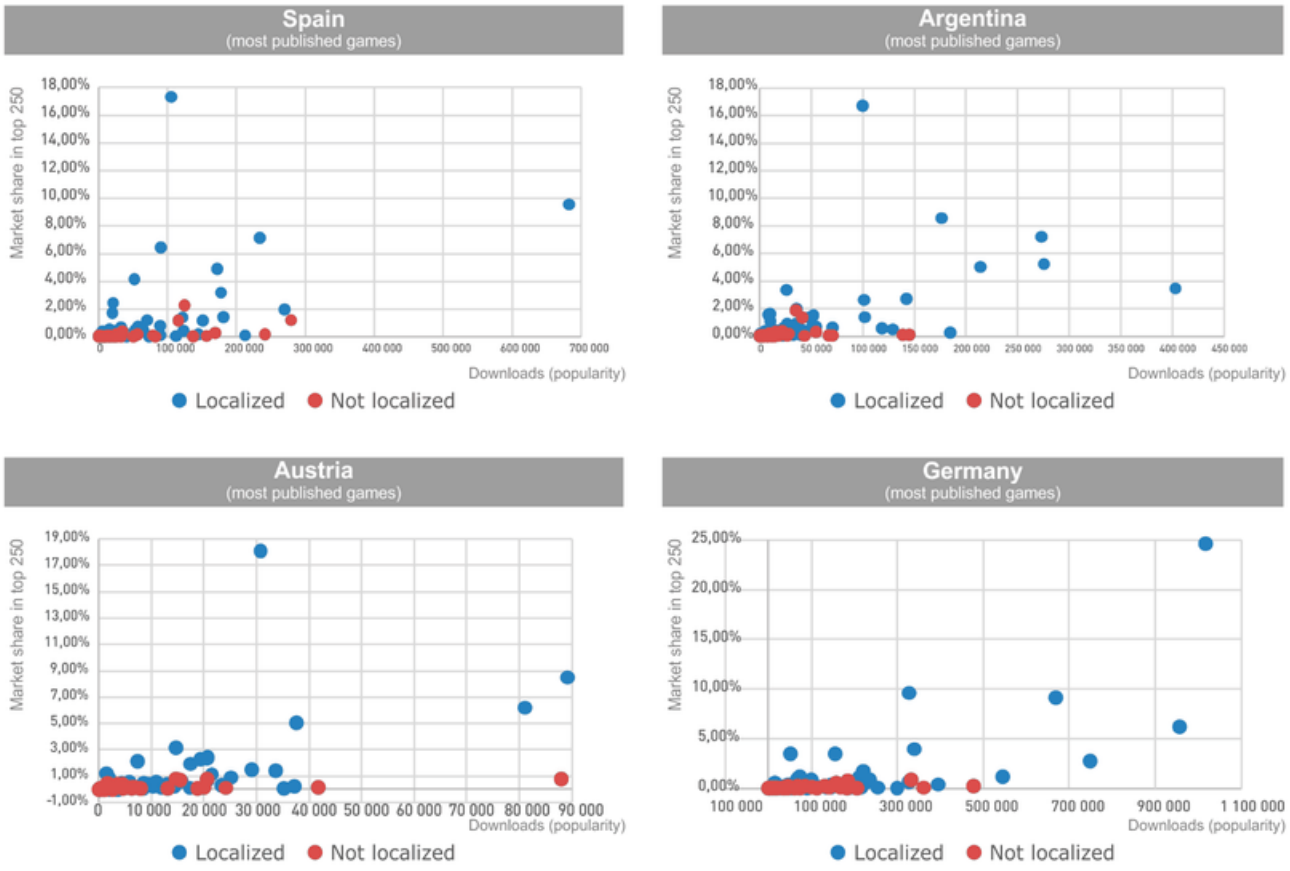
The game needs to add support for Turkish. It's not fun playing without it.

Does this mean that the game needs to be translated into Turkish ASAP? Not necessarily. First, you should evaluate whether the residents of a target region are willing to pay (to help you do this, we publish annual gaming market research on our blog). Also consider their current income, the number of downloads, and the average purchase amount. Still, requests like these are definitely one more reason to consider adding a new language.
It's worth checking the number of such requests before deciding whether to expand into a new market. For example, if a developer has a game that has been translated into EFIGS+CJK and is profitable, that developer will now start looking for opportunities for growth. One such opportunity is expanding into new language markets, and aside from metrics, the presence of requests for localization is a signal to choose that market in particular.

ROLI

Speaking of localization from a business point of view, we should consider ROLI (return on localization investment). This number shows how much money you earned from each dollar you invested into localization.
Naturally, there are many factors that influence a game's success in one market or another. These factors include the willingness of players to pay, ad campaigns, competitors, legal requirements, and cultural norms. However, not having a localization or having a poor- quality localization definitely reduces a game's chances of success. The first few days after release are key. If a game has negative reviews, it will be ranked lower in the store, resulting in less organic traffic.

We researched whether a game's profits are influenced by the decision to localize or not localize. To do this, we studied a selection of 134 games that were released in the most countries (45—54 countries). We also attempted to discover whether there was a link between the number of countries in which a game was released and localization into the language of these countries. We want to mention here that the country in which a game was released isn't necessarily connected with localization: 28 was the maximum number of languages that a game was translated into in our selection.



We plotted this data in a chart in which the X axis is the number of downloads and the Y axis is the market share among the top 250 games. Looking at this chart, we can see there's a direct link between the number of localization languages and how much a game earns (more languages equal bigger earnings). With the help of the results, we were able to calculate the average yearly ROI of localization in several countries.

Formula

$$ROI_{avg} = \frac{(1-0{,}3) \cdot \left(REV_{avg}^{loc} - REV_{avg}^{nl}\right) - CPI \cdot D_{avg} - WC_{avg} \cdot CRW}{WC_{avg} \cdot CRW}$$

ROI_{avg} — Average (for the country) return on investment for localization

REV_{avg}^{nl} — Average revenue of an unlocalized game

REV_{avg}^{loc} — Average revenue of a localized game

CPI — Average cost per app install

D_{avg}— Average number of game installs

WC_{avg} — Average number of words

CRW — Cost per word

Simplified formula

$$ROI_{avg} = \frac{(1-0{,}3) \cdot \left(REV_{avg}^{eff}\right) - \left(Marketing_{est}\right) - Loc_{avg}}{Loc_{avg}}$$

The difference between the average profit of a localized and unlocalized game

Nominal marketing costs = average cost per install x number of downloads

Average localization cost

We introduced the idea of “added localization value” in the formula. This is the difference between the average revenue of a localized vs unlocalized game. We tried to estimate marketing costs (calculated as the cost of the average number of installations for a given type of game) and, of course, included the cost of localization.

Localization testing

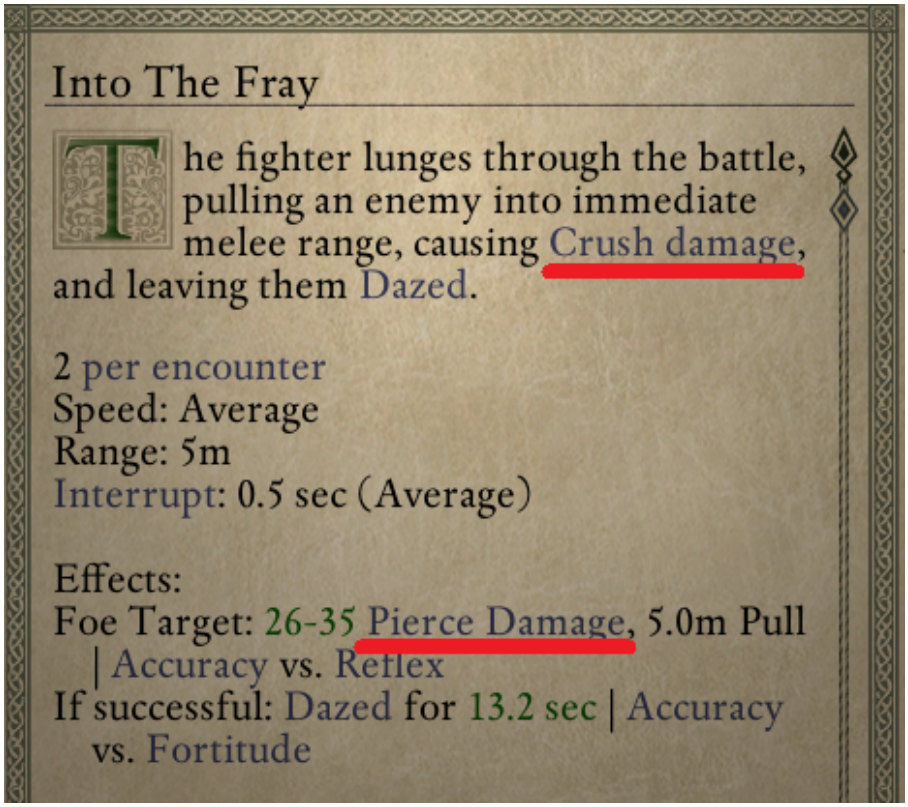
Localization testing is often viewed as unnecessary and done with the scraps of remaining funds. After all, the translation is already perfect. Why check it again? But working in this field, we often see that pitfalls may be lurking in even the best translation.
Let's take a look at where and what the problems can be. And most importantly, how to avoid them.

No standard measure

Bugs often hide in window titles and button text, since these areas have a limited length. Of course, information about character limits is provided at the translation stage. However, it's important to realize that some languages (like German) have longer words than English. And Chinese characters are something else entirely. So even with a set character limit, there's always a chance that a word might grow a little "tail".

A whole phrase is not enough for us

It often happens that a phrase is broken up into parts during translation so that it can later be used in other phrases. This can create a situation in which each part is correctly translated, but the parts don't work when joined together. What's more, you need to keep in mind that these parts might be used elsewhere.



Battle of the sexes

Sometimes it's pretty funny to see characters who out of the blue start talking about themselves in the wrong grammatical gender. The tricky thing about translations is that you can't always be sure who is speaking. This is true even if the name of the speaker has been provided. For example, is Sam Brown a man or a woman?

This is why translation requires both character names and character images.

There may also be technical reasons for this issue. For instance, the image of one character might load in place of another character.

Font affairs

No matter how perfect a translation is, nothing can ruin the impression more than a mysterious oooooo, which can appear at the most inappropriate moment if the font refuses to work properly.

Of course, a font can be universal for all languages. Perhaps the inexplicable oooooo will never make an appearance in the game, but only someone who knows the language can check to ensure all special characters, such as the German umlauts, are in their places.

A word on poor context

It seems like a simple matter: each word has a translation. Everyone goes home happy.

The trouble is that while translating, it's sometimes uncertain how the text will look in the game. It might be unclear exactly where it will be placed.

For example, the word "start" out of context could mean "starting point", "origin", "launch", "beginning of the game", and so on.

The same phrase, when used in a diary, a newspaper article, or a character's line, could be taken in very different ways.

So, sometimes you may encounter text that you understand, but it clearly doesn't fit the situation in the game. And this could ruin your game experience so much that you decide not to play anymore.

There are a few ways to avoid these kinds of mistakes.
First, provide translators with as much information as possible about the game, its characters, and any context.
Keep in mind that every language is unique. For example, the issue of text creeping outside of window boundaries could be resolved at the design stage by determining object sizes in the interface using substitutes from machine translation.
And, of course, do final text checks directly in the game.
Experience has shown that having native speakers test the main languages for at least the first two hours of gameplay (and certainly the menu and key windows) has a positive effect on the audience's perception of the game and, later, its attention retention.

Working with feedback

In order to encourage players' loyalty, developers need to react as quickly as possible to any issues that come up. This isn't an easy task, since not every review contains constructive criticism or even a clear description of the problem. And if the product has been translated into several languages, then the amount of feedback received increases exponentially. Sorting the wheat from the chaff becomes difficult. But let's not complicate things. Instead, let's take a look at how it's possible to support those who play multi-language games without substantial financial investment.

In fact, it's pretty easy: pay attention to user support. To do so, companies need to have a special process. Big companies use a large team of specialists from different countries. These specialists read player reviews on different platforms, forums, social networks, and on the community section of the website. Not everyone can afford to do this, but it's no reason to avoid entering other markets.

Optimizing the player feedback process is just the beginning. It comes as no surprise that every multi-lingual project should have its own separate group or page on all current social platforms like Facebook and Twitter. These platforms should always offer fresh content. Aside from this, there's actually another way to communicate with users that's often ignored by beginning developers. Consider creating a wiki for your game. This allows community members to share knowledge and experience, which creates a sense of belonging and loyalty to the game. It's also worth writing all manner of manuals and FAQs about different aspects and game mechanics. This is especially relevant before a new update to help your audience quickly adapt and avoid unwanted negativity. And above all, if you encounter any trouble adapting your content to other languages, you can always contact a localizer for help.

Remember this: it takes more than just translating your game to successfully enter other markets. You'll need to work with the community and support it. This applies to any size studio. One memorable example of this is how the misunderstanding between Bungie and their users negatively impacted Destiny 2. Although it's not always easy to work with your community, it pays off in the long run. A close-knit community can also help support their favorite game by leaving positive feedback and creating additional content. They might even provide financial support, for example, by raising funds for a new project through crowdsourcing platforms.

Conclusion

While localization isn’t essential, it’s still a very important element of game creation. Localization can significantly affect a game developer’s success in the international market. If you want to win the love of your players, you need to speak with them in a language they understand. This is exactly what localization helps you do.

The Allcorrect team has more than 15 years of experience in game translation and cultural adaptation. We’re always ready to help cool projects reach players all over the world. If you have a question that we didn’t cover in this book, you can always reach us on social media or via email.

Bonus: How to work with a publisher

While this stage isn't directly connected to localization, working with a suitable publisher can significantly impact your expansion strategy. Also, some publishers take care of a game's translation and culturalization.

Before setting out to find a publisher, it's a good idea to understand why you might need one. Every publisher has a specialty. Some handle everything except game design. Others focus their efforts on advertising and promotion. First, determine exactly what kind of help you need. Is it project management? Development? Marketing? Working with traffic? Localization? Analytics? Or is it all of the above?

Then take a look at the games already in the publisher's portfolio. Here, too, there are specializations. Some publishers work primarily with mobile "Match-3" games. Some deal with fantasy RPGs. Others publish games for a certain platform, focusing on projects with original mechanics or unusual visual styles. However, there are many publishers without narrow specializations.

From there, evaluate the readiness of your project. It's unlikely someone wants to buy just an idea. When you're at the prototype stage, then it's time to consider publishers. If you have a fully-developed vertical slice, the number of interested parties goes up.

The next thing you should do before contacting a publisher is check their professional reputation. If a publisher has been involved in legal conflicts with developers, don't sign any contracts before doing more research. Seek out opinions of other developers on forums or in group chats. Check for mentions of the company in the news. The more information you have, the higher your chance of making the right call.

What info you should prepare about your game

A publisher needs as much information as possible about a project before they invest money. However, no one is eager to spend hours and hours getting acquainted with your game. It's best to prepare the following before your meeting:

Create a six-word description of your game. For example, "a post-apocalyptic fantasy platformer with dragons". The description should be succinct and colorful. It should make your project stand out among the crowd.

Write a short description of the game mechanics. Be prepared to share what your players need to do and why the game will attract them.

Describe how the project will be monetized. The publisher will help you with this, but already having some proposals will make you even more appealing.

List ideas for scaling the project. Does the gameplay have potential for new levels, locations, and events?

Share graphics. Show off the existing locations and characters, along with interesting screenshots of gameplay. Animated objects look better, so it's a good idea to prepare a few short clips or even some gifs.

Prepare a prototype. The best way to show off your game is to let someone play it. It's likely you'll be asked to share a build.

Provide information about your team. The publisher will be curious about your team's level of motivation and experience.

Create a budget and timeline for your project. This doesn't need to be extremely detailed. A rough outline is enough for now.

Think over project risks and how you might address these possible issues.

For the sake of convenience, you can **package up all the text and graphics in a short presentation** (called a pitch deck).

Doing a simple market analysis will increase your chances of winning over a publisher. Find your closest competitors (games with similar gameplay and settings that were released on your target platform in the past year). Study their stats and make sure your plans are realistic. This will help you and the publisher make a wise decision on how to position your game.

What to consider asking a publisher

A meeting gives you the chance to evaluate the publisher. So, don't hesitate to ask questions: how profits will be split, what the predicted revenue is (a good publisher with experience working with games like yours will be able to answer this question), how many copies might be sold, and to what extent the publisher gets involved in the development process.

What to be aware of when signing a contract

So, you've decided to sign a contract with a publisher. It's likely the publisher will offer you a contract they've already prepared. Remember that this version of the document is, first and foremost, beneficial to the publisher. Ask questions about things that confuse you and propose changes when appropriate.

One sign of a good contract is if it contains a provision for an advance. Being ready to invest in you at the beginning stages of your collaboration indicates the publisher is ready to help your project succeed.

How are profits split? The standard practice is to subtract the publisher's marketing costs from the profit (this includes localization costs, platform fees, etc.). After that, the money is split between publisher and developer in the agreed manner. What is the usual split? There's no universal formula for this. It depends on the specific publisher, your project's potential, and your negotiation skills.

One last thing. One of the most important points to be aware of is under which conditions you can terminate the contract. After all, you don't want to lose the rights to your creation if something unforeseen happens. We recommend limiting the licensing contract to specific platforms, distribution areas, and licensing periods. For example, the App Store and Google Play worldwide, for two years. Make provisions for the end of the contract with return of exclusive rights to the developer if certain conditions are not achieved, for example, target sales revenue three months after the release date.

List of publishers

Lists of major publishers are publicly available on sites like Wikipedia. Industry publications often release ratings and round-ups. There are both general round-ups and more narrow ones, i.e., for mobile or console publishers. Ratings are formed based on income, sales volume of published games, number of installs, and other formal or less formal factors.

Acknowledgments

This book contains materials and excerpts from:

Anastasia Krzhizhevskaya
Artur Karniev
Daria Tvorilova
Denis Khamin
Dmitrii Antonov
Igor Nyrtsov
Igor Sheynikov
Marina Lekhina
Tatyana Veryasova
Valentin Pronin
Valerii Timchenko
Veronika Shpareva

Translator: Katherine Anderson

Editors: Diana Almeeva, Igor Nyrtsov, Irina Makarova

Project manager: Tatyana Veryasova

www.allcorrectgames.com
pr@allcorrectgames.com