INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Game Porting Optimization Techniques

PEDRO ANTUNES DE AZEVEDO BARROSO
Junho de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Game Porting Optimization Techniques

Pedro Antunes de Azevedo Barroso

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática, Área de Especialização em

Sistemas Gráficos e Multimédia

Orientador: Carlos Miguel Miranda Vaz de Carvalho

Júri:

Presidente:

[Nome do Presidente, Categoria, Escola]

Vogais:

[Nome do Vogal1, Categoria, Escola]

[Nome do Vogal2, Categoria, Escola] (até 4 vogais)

Porto, Junho de 2022

Resumo

Com o constante crescimento tecnológico da nossa sociedade, o uso de tecnologias digitais para entretenimento tornou-se um fenómeno comum, sendo que grande parte dos cidadãos utiliza os seus computadores, tablets ou telemóveis, como dispositivos de entretenimento onde jogam jogos eletrónicos.

Este crescimento tem levado ao aparecimento de inúmeras empresas dedicadas ao desenvolvimento de videojogos; no entanto, a procura por jogos diferentes supera a oferta, embora esta seja vasta. Como é um mercado bastante competitivo e exigente, é necessário utilizar-se formas consistentes de criar produtos com qualidade elevada e um custo apelativo para atrair os clientes.

Com a existência de diferentes plataformas (algumas físicas e outras digitais, como os serviços cloud), tem-se tornado recorrente o uso de *porting,* isto é, a adaptação de um jogo existente numa plataforma para novas plataformas, de forma a alcançar novos jogadores.

Um dos principais desafios no desenvolvimento do *porting* de um jogo ou de um outro software é garantir que a nova versão terá uma performance igual, ou até superior à original.

Neste trabalho pretendeu-se propor, desenvolver e avaliar uma metodologia de padronização do processo de *porting,* que permitirá acelerar o desenvolvimento de um jogo em projetos futuros, salvaguardando a qualidade e eficiência do produto. No final deste trabalho demonstrou-se, recorrendo a um conjunto de profissionais da indústria dos jogos, os resultados da implementação. Estes permitiram concluir que a metodologia desenvolvida cumpriu o seu objetivo, demonstrando ser um bom recurso para o desenvolvimento de jogos.

Palavras-chave: Videojogos, *Porting,* Otimização, Metodologias

Abstract

With the ever-growing technological advance in our daily day, the use of technology for entertainment has become a common thing for people. That entertainment can be listening to music, watching movies, playing games, among other activities. And games have been on the fastest growing part of that industry.

A sub-part of the gaming industry is *porting,* which is the adaptation of an existing game from one, or more, platforms to new, unsupported ones. With the increase of popularity with ported games, due to games exclusivities and different platforms, there was even the appearance of new studios that are specialized in porting games.
But both these specific studios, and the "regular" ones need to be able to make the best game in the shortest time they can. That is what this project aims to improve, by creating an optimized methodology for porting that can be used for most porting projects and that can help save time for the development, especially among indie developers or indie studios, while keeping it simple to understand.

Keywords: Porting, videogames, optimization, methodology

Acknowledgement

I would like to thank my advisor, Professor Carlos Vaz de Carvalho, for accepting my request for this dissertation, for all the help provided during this journey and for all that he taught me in the 2 years of this Master's Degree.

To the company Nerd Monkeys, that has shown me a warm welcome since the first contact and that accepted my request to work with them. It was a huge opportunity to be able to work on a game studio for the first time, and it helped me to understand more about what it takes to work in this area. A special thanks to Diogo Vasconcelos, my supervisor, for making me go beyond my comfort zone and believing I could make it.

To my family, for always being with me and giving me all the means to chase my dreams and for supporting my decisions, not only now, but during all these years.

To the friends I met during this Master's Degree, especially Francisco and Nelson.

To the friends I met during these 6 years in ISEP. This goes specially to the ones from my haze, which gave me unforgettable memories.
Even more, to my friends Inês, Mariana, André, and Daniel, that have been with me since the first day on this Institute, and that despite all that went wrong and bad during these years, are still presently with me, and that convinced me to keep pushing forward when the world is against us.

To Ricardo, Filipe, Fernando, and Rafael, who helped me save a lot of my mental sanity during these pandemic times by allowing me to always relax with them.

And last but not least, to Ana and Inês. Ana for being with me for more than 10 years and still accompanying me actively. For all the nights out, parties, trips, even for all the serious moments, all the help that she always gives me, regardless of if it was school, work, or personal. For always making me feel at home. And Inês, for being present almost every day of this academic journey, for always making me see reason when I couldn't, for making me company and making these pandemic lockdowns actually bearable, and for being a source of inspiration.

To all that helped me to get to where I am and to be who I am today, my deepest thanks!

Index

List of Figures

List of Tables

List of Code Excerpts

Acronyms and Glossary

List of Acronyms

OS	Operational System
PS1	PlayStation 1
PS2	PlayStation 2
PC	Personal Computer
NDA	Non-Disclosure Agreement
DAG	Directed acyclic graph
AO	Aspect-Oriented
OO	Object-Oriented
SPO	Small Project Observatory
Kb	Kilobytes
Fps	Frame per seconds
API	Application Programming Interface
IDE	Integrated Development Environment
SDK	Software development kit
UI	User Interface

Glossary

indie	Independent.
Dev Kit	Development Kit, which refers to the hardware used in development. Certain platforms have a specific hardware for this.
Console War	An expression used by the community to talk about the competition between PlayStation, Xbox and Nintendo
cloud	Cloud Service is a software that is hosted by a third-party provider that is available to the users through the internet. The user simply needs a computer connected to the internet to access it. This allows for the user to use some heavy software without needing to have a high-end machine, since all the data is sent through streaming.
Game key	A combination of several characters, in a specific format, that can be used in certain website or application to obtain a product that is paid for free and legally.

1 Introduction

This chapter contextualizes the work described in this dissertation and explains the problem that it aims to solve. The process behind the work is defined in the next chapters and in the end, the results and conclusions taken from this project are exposed.

1.1 Context

The videogame industry is one of the many outcomes of the technological advances in our society. Although games have not always been seen favorably by most, nowadays people recognize them as something typical in the everyday use, with people no longer associating videogames with dependency and health issues, but as a simple mean of entertainment that can provide several benefits, even regarding health and mental issues, on the contrary to what was initially believed. It's also one of the most profitable markets.

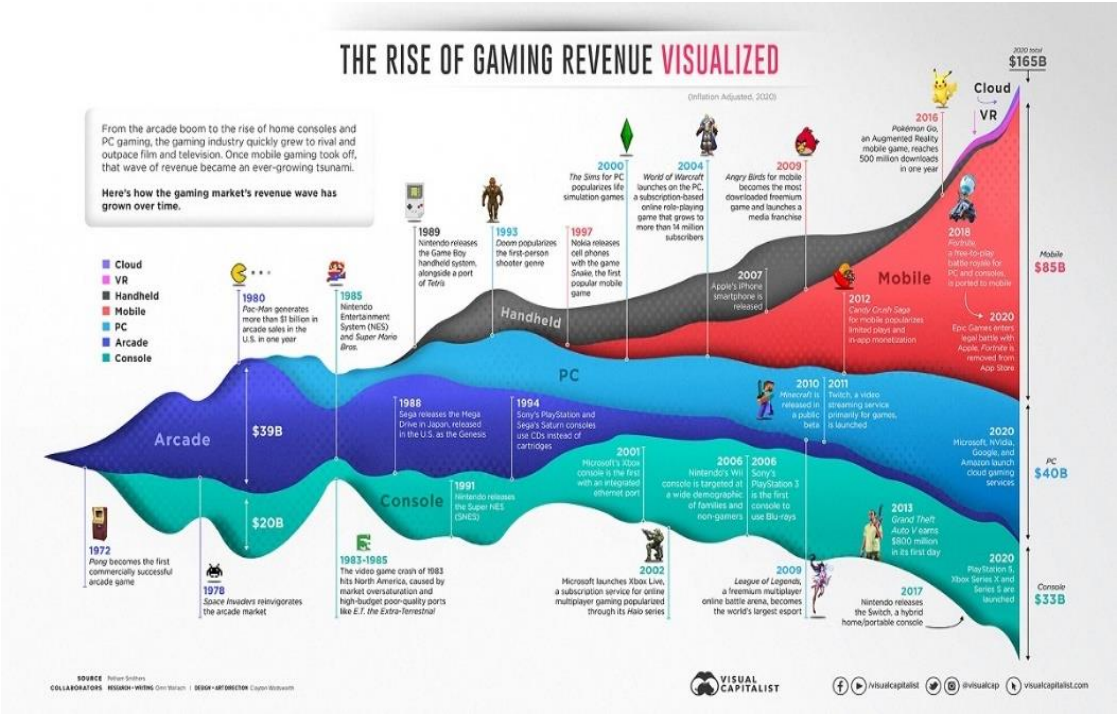


Figure 1 - Evolution of the gaming industry. (Visual Capitalist, 2020)

The definition of a "gamer" is not something consensual, and there are often discussions about it, but a big part of the citizens have already experienced at least one videogame, even if it was a game like *Candy Crush* or *Farmville,* played on social medias, or games like *Angry Birds* on mobile devices. Independently of which game they play and in which platform (either one of the above mentioned, or more recent and complex ones like *Fortnite* on computers and the current generation of consoles), the users all have the same goal: to have a good and fun experience while playing. To be able to provide that, companies need to develop the best software they can, so the player can always leave satisfied and return to play again.

To have a clearer view of this, a survey about games and porting was created by the author to better understand the current positioning of games and the point of view of the people regarding these topics. This survey and the answers provided can be seen in the first attachment at the end of the document. In the survey, 47.4% of the respondents did not consider themselves to be gamers. Despite these numbers, only one person of that group

has never played any game from the list displayed. The numbers indicate that almost everyone has already played at least one game at a certain moment of their life.

Of all the respondents, 52.6% said not knowing what *Porting* meant. After a quick explanation, 94.7% said to consider that *porting* is important. However, not all of them consider that a game studio allocating resources to do a *port* instead of new games is a good move.

One videogame can take years to develop, with some taking over a decade of development time, which can be a big setback for companies. Every year, more ambitious games are released, and the bar is constantly being set higher than before, making the videogame industry one of the most competitive ones. This competitive environment requires companies to come out with methodologies that can improve their quality or reducing the development time.

1.2 Problem

With the arrival of new generations of consoles, better computers and mobiles, alongside the constant development and updates of the technology used in game development, it's always possible to create more complex and incredible games. Besides the development of newer games, it also allows for older games to be played with higher quality or higher performance. But that is not always possible since games and the software they were built upon have restrictions. A certain game that has been released for mobile may only work in specific OS (Operational System) versions, which will eventually make the game die for not being compatible to the newer, and eventually most common, devices. Even when playing in consoles or computers, similar kind of restrictions happen. A game that was released for PS1 (PlayStation 1) could run on PS2 (PlayStation 2), but nowadays, the current generation of consoles won't be able to run that game.
Even inside the same generation period, compatibility problems can happen. A game released exclusively for console X, won't run on the "rival" consoles or on computers.

So, what do game studios do if they want to make a game more accessible to the players and let the players of the other platforms play? The answer is *Porting*.

Porting is the adaptation of a software that was developed with the intent to run on a specific platform, to some new platform where the software was not available, like an exclusive game from platform X being now available for players in platform Y.

Lately, porting has been a very common trend in the industry, since players want to be able to play any game they want in whatever device they own. This popularity has even made room for new studios and sub-studios to emerge, some being fully specialized in porting games. For example, PlayStation PC LLC is a publisher for Sony games on PC, which is a sub-company of Sony. But porting can sometimes be as troublesome as the initial development of an original game.

When we are porting any software, we want it to be as faithful to the original software as possible, regardless of the platform or device we are porting it into. For games, that can get even more complicated, since a game has several aspects to be considered, in comparison to any typical software.

For a game, developers need to have in mind the controls of the game, and the specific controller features that it may have, the performance and visual quality, the way the new system handles some methods, like the way the game handles saved data on a device, among other aspects. It's not only about fitting the image on a different screen.

Giving the current rise of porting and ported games, and the creation of porting studios, it has become necessary to find ways to make the process of porting a game or another software faster, keeping the original experience alive: maintaining the same performance while changing the features as less as possible.

1.3 Objectives

The purpose of this project is thus to create an optimized porting methodology that can be used in future projects to help and accelerate the porting process, while keeping it simple

for developers to understand and apply. To reach this goal, the following tasks have been identified as necessary:

- Study previous porting projects to determine what parts of those projects need to be changed, regardless of the game, to try and establish some common ground.
- Research about porting techniques in games and software that may be used as reference.
- Port one or more projects, optimize them and evaluate the optimized techniques to determine if they can be used for all projects.
- Synthesize and document the work done.
- Through users' feedback, evaluate the solution created.

1.4 Motivation

As someone who is very keen on playing videogames and technology, and due to the desire of working in this industry, the author considered a good idea to combine some of the knowledge learned during the course with his career goals.

Knowing the current state of the gaming industry, both globally and in Portugal, and knowing how much the develop time of some projects can hurt studios and make the players unhappy, it was thought that a way to help both the creators and the consumers would be a great addition. The creators will be able to create games more quickly and with the same quality, so the players won't have to wait so much between releases for the games they want to play.

This is a highly ambitious project, as a big part of the industry use specific tools and software that are exclusive to them, and those may do what this project aims to do at a different scale. Not only that, but also the fact that the competitive level of the industry creates a necessity to always have people working to advance the industry. However, a general approach that may be

used by all can be of great help to indie and less-experienced developers, and even contribute to work of some big studios.

1.5 Structure

This dissertation is divided in 6 chapters. This section provides a short description of each chapter to provide some insight into the work done.

The first chapter presents the problem and its context, the objectives and what was the drive for choosing this project.

In the second chapter, the state of art is described. This consists in an analysis of studies and existing works related to the theme of this dissertation.

The third chapter is the Value analysis in which is exposed the business model for this project, the value it intends to create to the client and how it will provide that value. This is done with the help of the CANVAS model.

In the fourth chapter all the implementation is described. All the steps that were made during the development of this project will be explained and detailed in this chapter.

In the fifth chapter the solution developed is evaluated according to some methodologies that are described in detail along this chapter, as well as the results of that evaluation.

As fort the sixth and last chapter, it presents all the results and conclusions taken from the whole work that was done during this project.

In the end, there is also a section related to the annexes which serves to complement the content that is described during the document.

2 Context

This chapter is dedicated to the state of the art. Here is displayed the analysis of the previous studies regarding optimization techniques and porting techniques on porting projects, both in games and outside games.

2.1 Games

What is a game?

Games are a topic that has raised the interest of several researchers, and is now a common topic under study: what is a game, why do games make people addicted, are they good or bad, etc. Even with some answers, these questions are still being studied. And so is the definition of what is a “game”. Since games are considered art, or a mix of arts, everything ends up being very subjective when it comes to the topic.

According to Elliot Avendon and Brian Sutton-Smith, “Games are an exercise of voluntary control systems, in which there is a contest between powers, confined by rules in order to produce a disequilibrial outcome.” (Schell, 2008, p. 31)

Another definition is that “A game is a closed, formal system, that engages players in structured conflict, and resolves in an unequal outcome”, this one provided by Tracy Fullerton, Chris Swain and Steven Hoffman. (Schell, 2008, p. 33)

Both definitions are right and have a solid ground to sustain them. Different definitions can have similar considerations or not. In this case, both have a similar approach to the definition. However, there are still a lot more different definitions, all of them can be considered right, but they can also differ in so many different aspects.

Salen and Zimmerman also have their own definition by saying that "A game is a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome." (Salen and Zimmerman, Rules of Play: Game Design Fundamaentals, 2003)

None of these definitions mention the digital part or the serious part about games because the definition of a game can be extended to both physical and digital games, serious or non-serious. Each of these terms are indeed a game, but they are also more specific in other senses.

2.1.1 Digital Games

A digital game can be simplified to the definition of any game that is played in a digital device instead of a physical one, but they still have some other differences than be considered:

- Immediate interactivity: in a digital game, any action of the user has an almost instant response from the system.
- Manipulation of (large) information: If in a game millions of players are doing different actions in different parts of the game, they will all experience the immediate interactivity. The game can handle an immense number of inputs and outputs at the same time.
- Automate complex systems: All the responses and outputs given by the game to the player are all automated by the system. This is the reason that a digital game can manipulate large amount of information and still give that immediate interactivity to the player
- Networked communication: While this might not happen to single-player games, the games that are played online in multiplayer require a network that can connect all those players together with the system and with each other. (Salen and Zimmerman, Rules of Play: Game Design Fundamaentals, 2003)

2.1.2 Serious Games

Serious games is also one of the terms that have been becoming increasingly more popular. As the other terms spoken before, it does not have a unique definition, even if the overall idea of the term is already established. For the most part, a serious game is a game that focuses on training or education contents and not on the fun and entertainment part. However, it must still use that entertainment part to captivate players. A game can teach the player new knowledge or skills directly, by being heavily focused on that teaching part, or indirectly by having the teaching mixed in the entertainment part. A good example of these later ones is *Assassin's Creed*, a game where you play as an assassin and eliminate some people, but the game is constantly teaching history with its lore and setting. Each game takes place in different time periods and countries, all during important historical events. (Tarja Susi, 2007)

2.2 Porting Techniques and Projects

While trying to find related works it was soon realized that this area is still lacking a lot of information. To restrain the information even more, all the three big companies fighting in the "console war" (Sony with PlayStation, Microsoft with Xbox and Nintendo with the Nintendo Switch) have strict NDA's (Non-Disclosure Agreements) with their developers. This makes it illegal for any relevant information to be shared.

Videogames are software, and in that sense, some things that happen in the game industry can be generalized to the whole software industry and vice-versa. For that reason, it was made a choice to search for related information about Porting with software in general. The foundation of the idea is the same, so it can be applied in both cases.

2.2.1 Draco Paradigm

A complex software has enormous amounts of resources invested in its construction. During the development, or in a post-release update, the software might suffer some changes in its requirements. Those changes can happen due to changes in hardware or a different environment, due to performance upgrades that requires changing some logic, or some other reason. Both users and developers can be the ones causing the changes.

But why do a lot of software companies take a long time to change and correct problems or add new features? Sometimes, those changes don't even happen. The software stays the same since the release.

One of the biggest reasons is that "*For most software the original design is inaccessible: the original requirements analysis and specifications, if recorded, are out of date. The existing realization of the software usually contain implicit assumptions about their environments.*". A project that was done several years ago, and that since that time has seen technology also improved repeatedly, along with some changes that can occur in the company during that time, are bound to have their original design lost. The company might still have the source code (or sometimes even the source code is lost) but doesn't have a registry of user cases and functionalities. And some of the code might had assumptions for a specific machine to run the code, but with the advance of technology, that machine is no longer viable and a new one with new specifications has replaced it.

Depending on the changes that the company decides to make, it can seem like a simple change with no elevated efforts. But just like in on-going development, a simple change can cause unpredictable breaks and malfunctions that will elevate the time and resources needed, according to what was initially considered.

To aid in software construction, a paradigm was created some years ago. This paradigm is called "the Draco paradigm".

This paradigm is used both before and during the development of a project. It assumes that the company will develop similar software, in which this similarity is based upon the fact that these software will all operate on objects from one or more domain. The domains that are needed in the development of the software for a specific application create a "*domain structure graph",* in which the nodes represent the domains and the refinements between

each of them are the *arcs.* In general, it exists more than one path inside a domain network, in which they go from an abstract domain node to an implementation domain node.

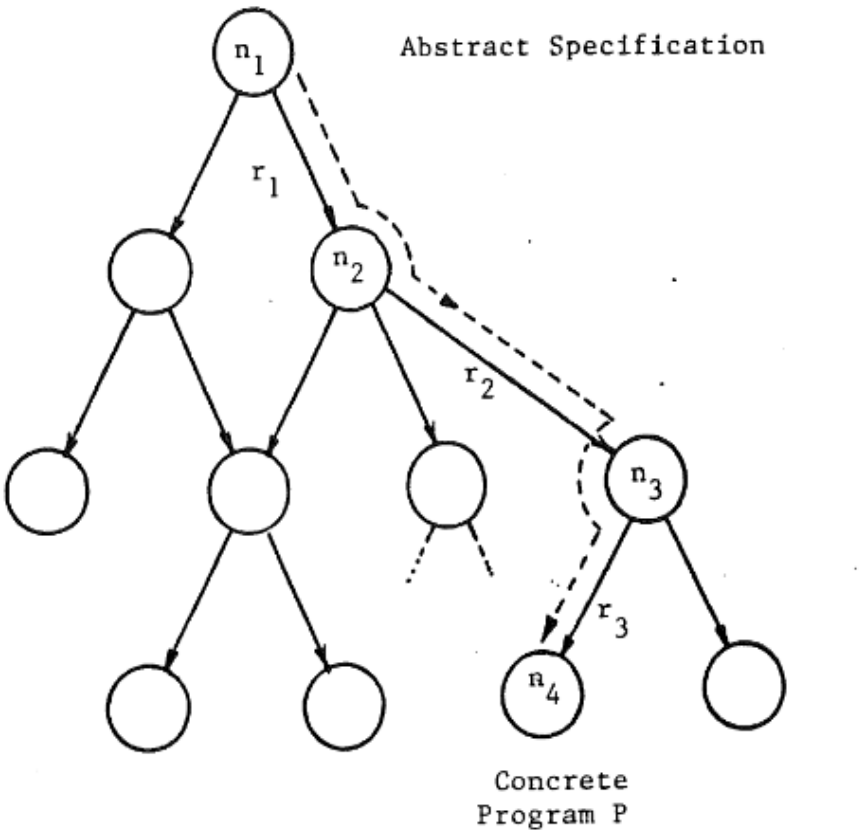


Figure 2 - Domain Network with an implementation path

The development starts from a high-end, abstract specification that will go through several possible implementations, in which each one will be less abstracted, until a final possible implementation is reached, which is the most concrete implementation. This creates a "*possible refinement DAG (directed acyclic graph)",* in which nodes represent specifications for the application. The "root" of the DAG is the initial specification, and the "leaves" are the specifications.

In the development process, a path from the "root" to one of the "leaves" are decisions made in the implementation. Side-by-side paths can occur having different styles: some being focused on implementation speed, others for performance.

After having the whole graph planned out, when a change is needed, one of the ways to approach it is to simply follow a new path from the initial specification to the new desired implementation.
Using this approach, all the changes only require a “backtracking” of the application. For porting, if the application will own the same functionalities in the new version as in the original, the company only needs to change the hardware or OS specific parts. (Guillermo Arango, 1985)

2.2.2 Aspect-Oriented Porting

The mobile games’ segment is currently the biggest segment in the industry, with more than 50% of the revenues being related to mobile. Acknowledging the expansion of this segment, more and more studios, and even indie developers, started betting on mobile games. This included both original games as well as ported versions from games that already existed in another platform or even a mix of both. An example of this last one is *League of Legends* from *Riot Games.* The game is exclusively available on PC, but they have recently created a mobile version called *League of Legends: Wild Rift.* This version plays like the original game, but with several changes to fit the mobile gaming: game controls, the way certain skills act, exclusive skins, shorter games, re-balanced stats, etc. Although they are two different games, *Wild Rift* is seen as a mobile version of *League of Legends.*

A mobile device is several times more limited than a standard platform, so the porting process when going to a mobile is a lot different. One of the crucial requirements is the portability. With tons of different mobile devices on the market, with different OS, the game has to be easily ported. To help in this matter, a study was conducted about Aspect-Oriented programming, which shows to be a good solution to the portability issues. Two games from two different studios were used to analyze the results of AO(Aspect-oriented) over OO(Object-oriented) programing.

In one of those games, the company had already tried both AO and OO, and made available a snippet from their source code for comparison:

```
ts
class C {
  T f
  fs
  ms
  T1 m(ps) {
    body
    this.f = exp;
    body1
  }
}
```

Code Snippet 1 - OO implementation

```
ts
class C {
  fs
  ms
  T1 m(ps) {
    body
    body1
  }
}
privileged aspect A {
  T C.f;
  after(C cthis, ps) ret(T1 t):
    exec(T1 C.m(T2(ps)))
    && this(cthis)
    && args(aps) {
      cthis.f = exp[cthis/this];
    }
}
```

Code Snippet 2 - AO implementation

In this sample, we can see that the use of AO turns the code bigger in terms of lines of code, and that will be later reflected into the size of the packaged application. Given three specific mobile devices, in the end of development, the combined size of the 3 applications was 179.8Kb when implemented with OO, while with an AO implementation, it summed 288.2Kb. In a computer or a console, these numbers are insignificant, but on a mobile device, every space of memory counts.

Despite the increase in size, using an AO can make the code simpler, as it compacts modules. By having these modules compacted and well defined, it's possible to faster understand where the porting needs to act and how. By leaving the generic modules out of the equation, the development teams only need to look at the modules which includes hardware or software specifics.

In terms of organizational structure, using an AO implementation also has some effect. Considering that both companies had small and stable teams, with specific roles, and considering that OO is the way they act regularly, when choosing to operate with AO implementations, that would end up creating extra roles, and redefining the roles they

already had. Both teams considered that to be unnecessary and that would cause some chaos in the teams while they adapted to the new organization structure.

Considering the size increase, the more compacted methods and the changes in the organization, the use of an AO implementation did not prove to be significantly positive for the change to occur. With the advance of technology and new systems, if the size increase can be lowered and the coupling of methods increase even more, then this might end up becoming a definitive upgrade for development teams. (Tammay Bhowmik, 2013)

2.2.3 Desktop tools to web

A good part of software visualization tools are desktop tools and not web based. Being a desktop tool, it requires installation and extra configuration that a web tool would not need. Sometimes those configurations can be annoying for the user, so they end up not using the tool itself. In that sense, a group of researchers that had experience in creating desktop tools tried to create two web-based tools. Those two tools are *Churrasco* and *SPO (Small project observatory).*

Each tool had its own features and specifications, and that included some troubles that usually did not happen on desktop versions, since the web is considered to be almost a different platform. Despite the differences between the two applications, the group was able to create a table of promises and perils that were common to both and could even be generalized at a greater scale.

Some of those promises and perils were the following:

Table 1 - Promises and Perils in web-based tools

Promises	**Perils**
More accessible	Privacy problems
Easier to update	Performance for high numbers of users
Easier to collaborate	Harder testing
Selective Deployment	Fast evolution in web technologies

When it comes to gaming, it can't be seen exactly as those tools. Even though consoles have internet and a browser, a game can't be deployed on the web for all platforms to play it via the web. However, if we consider the desktop tools as a game in a console, and the web tool as the game in PC, some comparisons start making more sense. Not everyone has high-end PCs, but most part of console owners also have a computer which they also use to play some games. Therefore, it's safe to assume that PC is the platform that has most active users, excluding mobile. With that condition, we can now reconstruct the table of promises and perils into a gaming scenario:

Table 2 - Promises and Perils of a ported game to PC

Promises	**Perils**
More accessible	Privacy problems
One update will reach more players	Servers' capacity and performance

According to the study, even though there are pros and cons to the porting, if the development team does it right, the pros can heavily out weight the cons. Having a game, or an app, potentially having a big increase in users can be enough to make it already worth to port. So, in the end, it's not a question of how many pros or cons we will have if we port the game, but the weight of its parts. Not every game is worth porting, just like not all applications are worth to have on a web. (Marco D'Ambros, 2010)

2.3 Good and bad Ported games examples

The process of Porting a game is something that an independent developer, or a big AAA studio, can develop. And even those big studios, which are recognized worldwide, have some products that when released, they fail to appeal the audience due to several issues with the game. In this section, a contrast between two ported games that are considered to be great and two that are considered to be bad for the community overall is made, as well as the reasons behind the discrepancies. The games selected also show a clear example of what was mentioned before. The two bad ports were made by AAA studios, while one of the good ports was made by a much smaller studio.

2.3.1 Ori and the Blind Forest: Definitive Edition

Ori and the Blind Forest is a platform-adventure game that was released for Xbox One and Windows on March of 2015. This game was very well received since the first day. It has a unique beautiful art style, smooth controls, and an appealing story. Despite that, the game is also considered to be very challenging, creating an intense gameplay. It also won some awards, even the Xbox Game of the Year, showing how great the game is.

Figure 3 - Ori and the Blind Forest: Definitive Edition game cover

In 2019, a Nintendo Switch version was released. Due to the power of the console in comparison with the Xbox, the game was expected to be worse in terms of performance than the original. But that was not the case.
The Nintendo Switch version was able to run the game even better than the original. For this reason, this edition was called “Ori and the Blind Forest: Definitive Edition”.
The Game Director of the studio behind the game revealed that during the years between releases, they learned how to make even better games, and in the process, they were able to create a better version of the game. The game could even run some animations at 60 Fps (frames per second), while the original was locked at 30. (Screenrant, s.d.)

Figure 4 - Performance comparison Xbox vs Nintendo Switch

The gameplay, despite being the same, reportedly feels even more natural in this new version, which could be related to improved latency in inputs, the use of an analog stick, or the simple layout of the buttons in the Joy-Cons.

2.3.2 Witcher 3

Figure 5 - Witcher 3 for Nintendo Switch

One of the most famous role-playing games of the last decade, The Witcher 3, the final entry of this franchise, is considered a masterpiece worldwide. Not only does it provide a huge world full of interactions, lore, quests, and places to visit, but it provides a beautiful scenario in most parts of the game. The towns are full of people, the roads filled with monsters and enemies, and the game renders all that with great quality for most consoles. The performance is also very stable, even though it's locked at 30 for some consoles due to their power. The game even won the Game of The Year award in more than one ceremony, with one of them being *The Game Awards 2015*.

The game released for Windows, PlayStation 4 and Xbox One in 2015 but only came to Nintendo Switch in 2019. And for most people's surprise, the game ran great. Nevertheless, the development team had to sacrifice the visual quality in order to provide a stable performance. On the Switch, the resolution was reduced to 540p when used in handheld mode. But this sacrifice allowed the game to run without problems, not having any performance issues, being most of the times at 30fps. But even the other consoles didn't manage to always get a steady 30 fps. (Polygon, s.d.)

This version shows that, when not possible to port a game 100% as the original, sacrifices must be made, and the team has to choose what they sacrifice and what they optimize. Considering that a stable game, with lower resolution is preferred to a game with great graphics, but unplayable, the developers chose the right option and created a great game for a very limited console.

As it can be seen below, despite the reduced quality, the game still looks good on the portable console.

Figure 6 - Witcher 3 graphics comparison

2.3.3 Nier: Automata

Nier: Automata is an action role playing game that was released for the PlayStation 4 in February of 2017. The game won several awards and was nominated for several others. The rankings by the critics were all very high. It had an average of 88 points in 100, on Metacritic. The experience the game provided on PlayStation was unique.

Figure 7 - Nier: Automata game cover

After the original release, the game had a release in Steam that seemed to be a downgrade of the original version. The game presented rendering problems, some performance issues, even crashes. These problems can be seen in several gameplay videos of users on the internet. With PC versions being usually better than console versions, due to the power of computers, this version was considered a disappointment to lots of fans.

Only in mid-2021 did the developers announce a patch to fix some problems with the game, meaning that the game stayed troublesome for more than 3 years.

The critics and players community consider this one of the worst recent ports, when considering the prestige and capability of the developer team behind the game and what seems to have been a simple effortless work.

2.3.4 Dark Souls: Prepare to Die Edition

Figure 8 - Dark Souls: Prepare to die edition

The Souls franchise has been around since 2009 and is the “father” of the genre souls-like in gaming. These games are known for their heavy difficulty, with the game punishing the player for every little mistake.

Dark Souls was released for PlayStation 3 and Xbox 360 in 2011, and it was a good game for that time, considering the limitation of the technology of that generation of consoles. Most players and critics gave positives reviews to the game, despite the heavy difficulty, and the game even won some awards.

FromSoftware, the company behind the game, announced the game was also coming to PC, which made PC-players all very expectant. But as soon as the game released, a heavy surge of disappointment fell over the players.

Dark Souls: Prepare to Die edition released on PC in the end of the summer of 2012 and it was full of flaws. Since PC’s have more power than consoles, usually the PC version is the better one, but in this case, the PC version didn’t have key-mapping features, didn’t allow the use of mouse and keyboard, the resolution and frame rate stayed the same as in consoles, and it even turned out unplayable for some users, with crashes and frames dropping a lot. The team confirmed they rushed the game into PC because players wanted to play it, and they did not bother to optimize or fix the game. "We did know there were PC-specific features like key-mapping and use of the mouse and keyboard, high resolution and higher frame rate, stuff like that, but… It's not that we ignored it, but it would have taken too much time for us to

implement it, test it and get it up to the level people expected" – Takeshi Miyazoe, Dark Souls 2 Producer. (PCGamer, s.d.)

It was needed the intervention of modders to create some fixes for the game so that players could actually play the game. The development team did not try to fix the game, as they said they were working to create the sequel to be a better experience, and so they left the fixing to modders.

3 Value Analysis

The Value analysis aims to measure how to increase the value of a given service or product while reducing the cost without sacrificing the performance and quality of the same. To describe this analysis, this chapter will have a detailed sub-section for the Value Proposition and for the Business Model, which will consist in the Canvas model.

3.1.1 Value Proposition

The Value Proposition is what the company or organization can provide, in terms of benefits, to its users. What their product/service will have that it's new, or simply better, than the existing ones. Who and how it will benefit them comparing to the rivals.

The value created in this project is the creation of a methodology that will serve to reduce the development time of a porting game, by having a process described that can be used for almost any project, while allowing the porting to be faithful to the original. This process will also be simple enough for any developer to understand what the methodology requires them to do, and how to do it. The reduction of the development time will not only benefit the company, by making the development cost less, but will also indirectly benefit the players, whom will be able to have new releases more often.

For this project, more than one *porting* project will be worked on. During the development, the steps taken will be documented. These steps will include description of important common points that exist between all games and that must be tackled for the game to be operable. Optimization of the port will be considered while trying to determine those points.

3.1.2 Business Model

When a company or organization wants to define their value and how to obtain it, the business model is used, and in most cases, it resumes itself to a CANVAS model. This is a model that easily explains the primary points of the business. Each section is the answer to some of these questions: (EINOV, 2021, pp. 26-31)

- **Customer Segments:** For whom are we creating value? Who are our most important customers?
- **Value propositions:** What value do we deliver to the customer? Which one of our customer's problems are we helping to solve? What bundles of products and services are we offering to each Customer Segment? Which customer needs are we satisfying?
- **Customer Relationships:** What type of relationship does each of our Customer Segments expect us to stablish and maintain with them? Which ones have we stablished? How are they integrated with the rest of our business model? How costly are they?
- **Channels:** Through which channels do our Customer Segments want to be reached? How are we reaching them now? How are our Channels integrated? Which ones work best? Which ones are most cost-efficient? How are we integrating them with customer routines?
- **Revenue Streams:** For what value are our customers really willing to pay? For what do they currently play? How are they currently paying? How would they prefer to pay? How much does each Revenue Stream contribute to overall revenues?
- **Key Activities:** What Key Activities do our Value Proposition require? What Key Activities do our Distribution Channels requires? What Key Activities do our Customer Relationships require? What Key Activities do our Revenue Streams require?

- **Key Resources:** What Key Resources do our Value Propositions require? What Key Resources do our Distribution Channels require? What Key Resources do our Customer Relationships require? What Key Resources do our Revenue Streams require?
- **Key Partners:** Who are our Key Partners? Who are our Key Suppliers? Which Key Resources are we acquiring from partners? Which Key Activities do partners perform?
- **Cost Structure:** What are the most important costs inherent in our business model? Which Key Resources are most expensive? Which Key Activities are most expensive?

Table 3 - Canvas Model

<table>
<tr>
<td rowspan="3">Key Partners

GILT;
Game studios;
Platform creators (Sony, Microsoft, Nintendo)</td>
<td>Key Activities

Definition of the porting.
Development of the porting.
Tests in the new platform.
Detailed documentation</td>
<td rowspan="3">Value Propositions

A methodology that can help during the development of any porting by describing and simplifying several steps of the porting process, with the intent to reduce development time while keeping the game equal to the original.</td>
<td>Customer Relationships

Social Media;
Email;
Website;</td>
<td rowspan="3">Customer Segments

Game studios that include porting games in their portfolio;
Porting studios;
Independent and less-experienced developers;</td>
</tr>
<tr>
<td>Key Resources

Development frameworks;
Platform dev kit;
Development team;
Documentation;</td>
<td>Channels

Website;
Social Media;
Email</td>
</tr>
</table>

Cost Structure	Revenue Streams
Development team; Dev kits. Marketing;	Units distributed;

3.1.3 FAST

The Function Analysis System Technique (FAST) is a technique that consists in a graphical representation of the logical relationships between the functions of a project based on the "how" and "why" questions. It allows to show how a proposed solution fulfills the requirements of the project.

As shown below in the example, this diagram has different ways to be read, but it can be seen as the left most function the final goal (in this case eliminate the mouse) and that function is decomposed into steps as we go to the next function on the right. To eliminate a mouse, we need to kill it. To kill it, we need to swing the striker. To swing the striker….

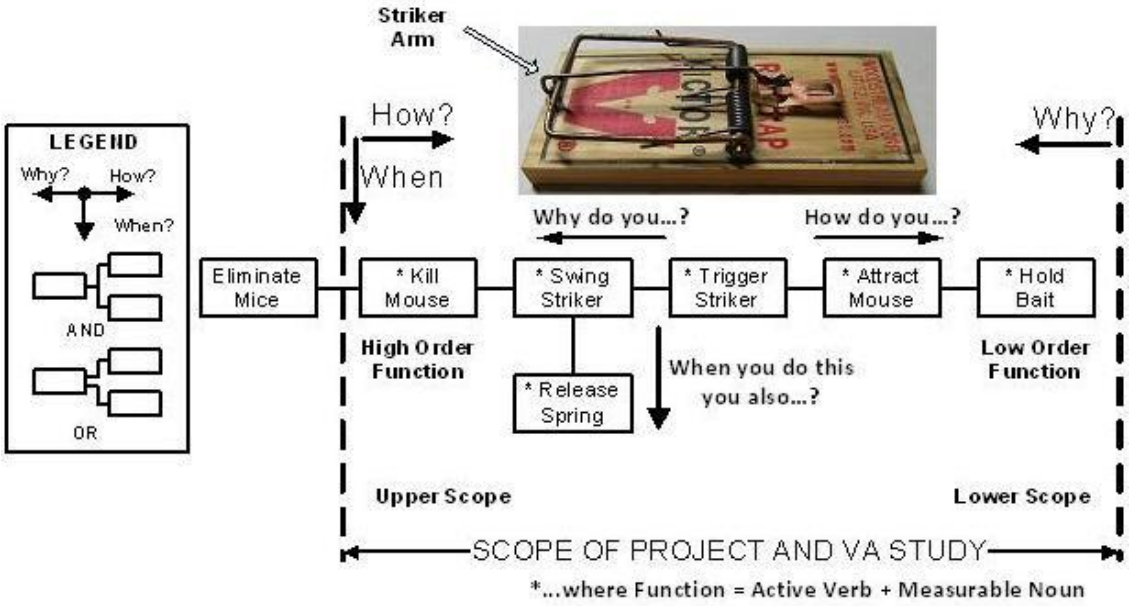


Figure 9 - Mouse trap (FUNCTION ANALYSIS SYSTEM TECHNIQUE (FAST), s.d.)

The creation of this diagram can help a development team to identify missing functions or simplify the problem clearly.

To better understand the process of the methodology being made, a FAST diagram was also created regarding the functions that are needed for the development of a porting. The specifications of each function will be approached during the development.

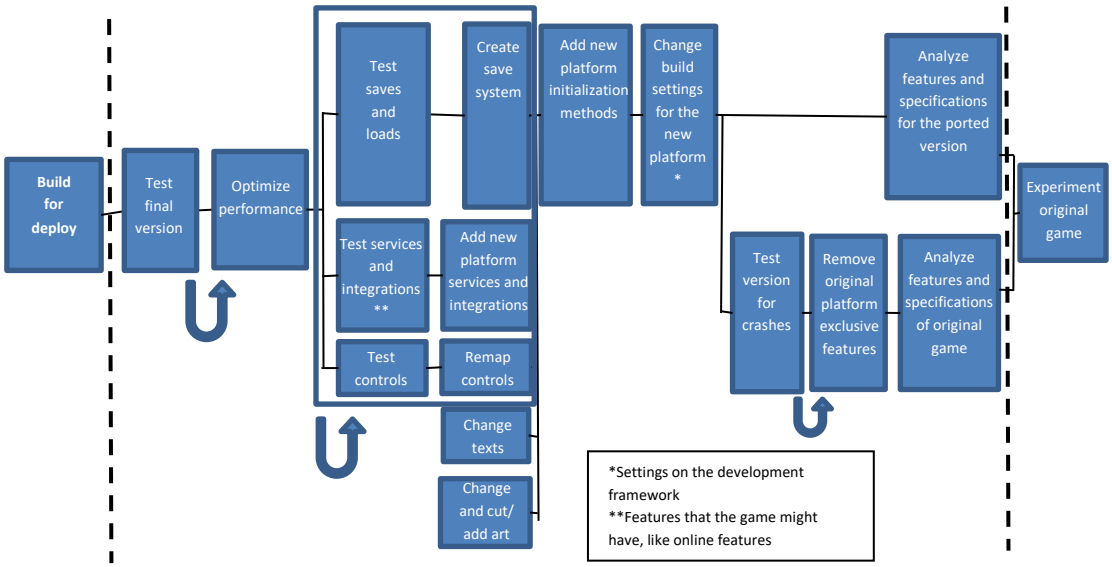


Figure 10 - FAST Diagram

4 Implementation

This chapter describes how the solution was created. For this, two portings were worked on. For each one, the process behind the analysis and implementation of the changes for the porting version is explained. Due to the NDA, pseudonyms will be used for the games and for the platforms the porting is destined to.
The first game will be referred to as *Game 1* and the second one *Game 2.* As for the platform, it will be called *Platform X.*

4.1 Design

In order to start the development of the solution, it was necessary to design how it was going to be made. After some discussion with the company about what would be the best approach, it was decided the solution was going to be made by developing some Ports, which would allow to identify the fundamentals steps of the porting process and, for each game, the necessary methods and steps. In the end, by analyzing both projects, it was possible to determine what is common between each porting, and what is specific, to create the solution itself.

With the experience of the company's members with several projects, and after analyzing the games meant to be ported, the following topics are the ones that can be observed in every game by experimenting the game and accessing the source code:

- Gameplay, how the game is meant to be played and what experience are we trying to transmit to the player,
- Controls, what actions and movements does the player need to do to play the game. This includes the UI menus (main menu, settings, pause menus, etc) and the levels itself.

- Services and integrations, what extra functions or features does the game use (like geolocation, camera, online features, voice commands, etc)
- Art and audio, what are the assets used in game, what is essential and what can be disposable or optimized.
- Settings, what kind of game settings are available, like resolution, v-sync, anti-aliasing, audio volume, etc.
- Save data, what does the save files contain, what is written, what is read, when are the save operations called and how is it handled.

Each project can require more or less work in each of these areas, and in some cases, they might not even exist. However, besides those, there two other important topics which can't be determined by inspecting the game, but only after being involved in the development of the porting:

- Setup, which can be started even before having the game's project files and it consists in readying everything for the programming parts. This includes registering as developers for the platform's portal, obtaining the necessary materials and installing all that is necessary for the development.
- Optimization, improving the performance of the game after the development has finished in order to get the maximum possible quality with the current conditions.

4.2 Game 1

At the start of the project, no previous experience with porting or with *Platform X* was owned, so for the first few days, the work only involved reading and analyzing the official documentation for the platform, to have a better understanding of how it operates. This provided a good theorical help for the development, but never provided concrete examples and samples of how to code some parts, as that differs between games. For that reason, the development still had a good part of "try and error" until a good result was found.

4.2.1 Gameplay

After the theorical part was understood, the next stage was the analysis of the original game. A game key was provided to obtain the version that is available for purchase, and not a development version. The game was then play tested and analyzed in detail, covering the topics previously mentioned.

The development started as soon as the play test concluded.
Since the game came from mobile, it had several features that were specific to the platform. So the first step was to remove all those features. The version was the same regardless of the

OS of the mobile device, so the project had both Google and Apple calls and references. These included accounts login and handling, achievements, online scoreboard to compare with friends' scores, remote databases and saves to the cloud (besides local save, it also saved the game to a cloud service associated with the user's account), and other external API's.

All of it was removed and the project tested after each removal to guarantee the game was still runnable. After passing the last test, the project was now considered to be on a *master* version, a version that is fully runnable on PC without any issues or platforms-specific calls, although it is not in a releasable form.

4.2.2 Setup

From the *master* version, the following stage was starting to implement the changes according to the *Platform X* features. To be able to develop for some platforms, it's necessary to be a verified developer by the platform makers (Sony, Microsoft or Nintendo). For this, it's necessary to contact them directly to their websites and web-portals for developers. Some information will be requested by them and then a process of verification is going to take place. This can take couple days or several months to be made.
By the information obtained through the company's members and from forums of developers, this process can be made shorter. If a person is registering with a personal email, as a solo developer, or not associated to any game studio, the process can be considered less priority and the verification take longer. If the person is associated with some game studio instead, and especially if the studio is already working with the platform, the process can be quite fast. However, this will still vary from platform to platform, and from time to time. Periods with lots of user's requests for registration will take longer.

With the verification complete, the *Platform X* SDK and all other necessary materials were now able to be obtained, and it was possible to start the actual development.

4.2.3 Controls

After the SDK installation and full setup of the necessary things, the first stage was to change the controls of the game. Since the game was from mobile and it was on a *master* version, the only controls present were touch/ mouse inputs. Not all platforms have touch inputs, and there are several that use a controller, so the inputs had to be remade from scratch for those, leaving the touch and mouse still functional.

The game presented than one situation where the player could interact with the game (main menu, settings, in-game, etc.) so the core of the game was the starting point of the implementation, the in-game, when the player is actually playing through the levels of the game.

Despite having several scenarios where player inputs are necessary, the game can be simplified to two. The In-Game User Case represents the scenarios when the player is actually playing the game and the Menus User Case represents the different menus the game possesses. All the different menus have the same structure, so they can all be defined by one User Case.

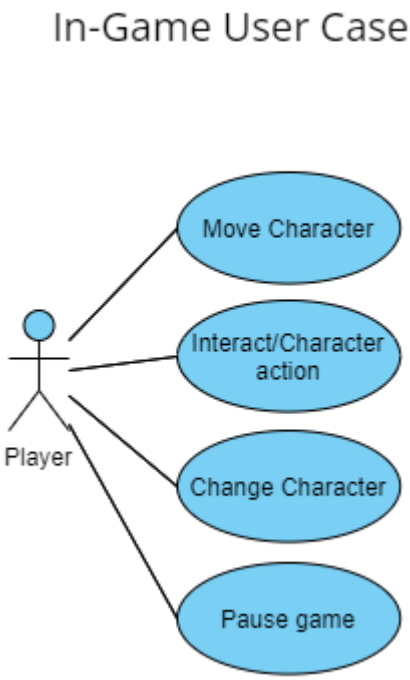


Figure 11 - In-Game User Case

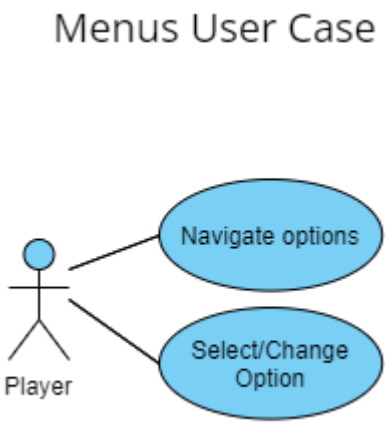


Figure 12 - Menus User Case

Despite not existing any explicit rule that forces developers to assign a certain type of action to a specific button (except for platform specific buttons like the PS button on the controllers, which is not changeable), the inputs usually follow the same logic between games. A racing or a shooting game needs precision and smoothness on the movements, one to drive the cars and the other to aim the weapons. This could be done with the D-pads (the arrow buttons) on a controller, but it restricts the movement to 4 angles, or 8, if simultaneous inputs are allowed. The analog sticks are a better solution for this situation because they allow for 360º movement.
On the other hand, a game like Pac-Man, which only allows for movement in 4 directions, can use the D-pads since each direction is assigned to one of the buttons. Using the analog stick in games like this can lead the player to be frustrated when he doesn't push the analog stick in the exact angle that its needed, and the character doesn't act the way he wanted.
Actions like jumping, which can be assigned to any button, also has a "standard", and its usually on the south button for the controllers (X button for PlayStation, A for Xbox, B for Nintendo Switch).

For *Game 1,* with the character's movement being 360º, it was decided to choose the analog sticks. However, after assigning all the other actions to their buttons, the D-pads were left unused. Due to accessibility and to consider more users, the movement was also assigned to those buttons. So in the end, the player could choose which way to use to move the character, according to what they prefer, even if one way provides more freedom than the other.

The other actions were assigned to different buttons based on what felt more natural and easy for the player. This was based on what the development team considered that would be better while holding the specific controller, as the team acted as testers to their own game.

For the different menus the game had, like shown on the User Case above, the actions were simple enough and the logic used to implement inputs was the same as for in-game. Selection is done with D-pads/ analog sticks, and selection/ cancel options are assigned to buttons that are commonly used for so.

Both of the implementations mentioned above were done based on the original methods and functions that existed on the source code. The inputs were not re-done from scratch. Instead, some changes were made to detect the new controller's inputs and those would call the original functions to perform the actions.

After implementing the new inputs, and before proceeding to the following stage, a build is created to test them on the dev kit. Based on the feedback from the tester and the errors or messages the dev kit showed, a period of fix and test was needed. When testing and fixing inputs on a controller, to make sure the player's actions are accurately carried out by the game, it's necessary to have a particular attention to the sensibility. If the player pushes the analog stick completely down, a character should move straight south, and not diagonally. Depending on the game, the speed can also be different based on how much the analog stick is pushed.

4.2.4 Art and audio

With all the controls tested and implemented, it's necessary to give some information to the player on which button perform each action. When dealing with a mobile game, the player can only tap, hold, or drag. Those actions can usually be done directly on the game objects (for example, dragging a candy to another position in Candy Crush) or by interacting with the interface, like virtual analog sticks and buttons for every action, as seen in the example below.

Figure 13 - Epic Conquest interface for mobile

Even if it's intuitive for a person to realize the inputs on a mobile game, often there's still a tutorial or some in-game guide to explain how to play the game and what does each button do. So, for consoles, and computers, where there are several more buttons and keys to press, and even combinations of those, having some tutorial or hints are almost mandatory.

Game 1 had a tutorial level explaining how to play the game and which actions (taps, holds, drags) did something. This tutorial had to be adjusted for the new platform, by having the exact same content, but changing the buttons displays for each action.

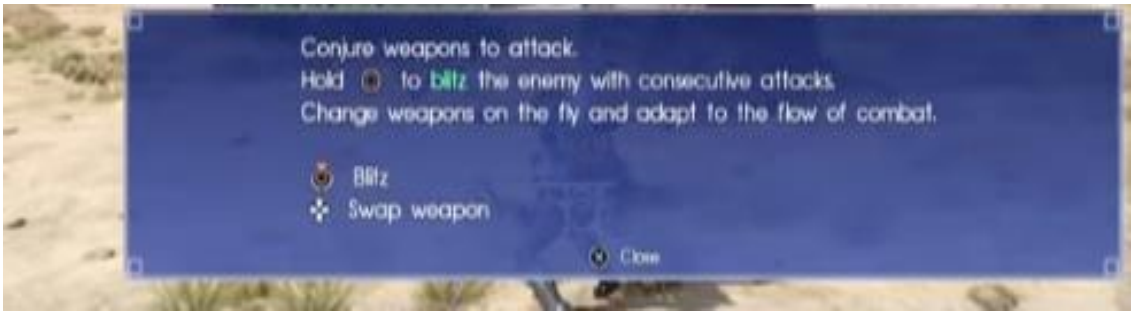


Figure 14 - Tutorial example from FFXV on PlayStation

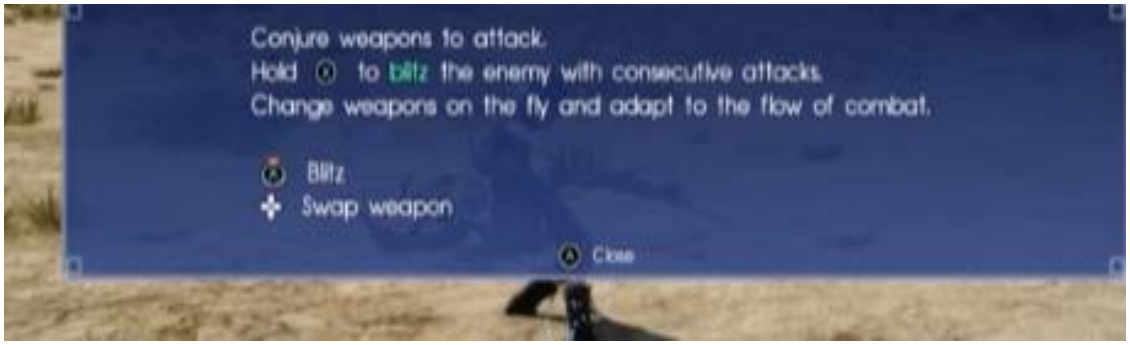


Figure 15 - Tutorial example from FFXV on Xbox

Besides changing the tutorial, and other places where inputs were mentioned, it was also added some labels on the menus to hint the player which inputs to use to select the options and navigate the menus.

These changes to the texts and icons required the art team to be involved, so they could design the images for each new button. The programmer can be responsible for doing such tasks, but in most studios, there will be someone with another role that will be responsible for those parts, allowing it to be worked on at the same time as other tasks are being made. Depending on the game, and the performance the game is having on the new platform, they can also need to change the art of the game itself, by changing the resolution of the sprites, or in 3D games, the number of polygons for each asset. When a port is made from mobile to any other platform, usually the art can have increased quality, more details, since mobile is the weaker platform when it comes to power. On the other hand, when it's from any platform to mobile, or from PC to consoles, it's usually required a visual downgrade in order to maintain a good and stable performance.

The art team also created new assets. In some games, they are weapons or skins (which are cosmetic items that only change the character's appearance). Sometimes, they are exclusive to the new platform, as a "reward" for the players that had to wait for the port, and to let them stand out from the other platforms' players. Since these are new additions to the game, it's necessary to implement them on the code along the rest of the assets, or in the Editor itself, when it comes to game engines like Unity. This implementation will differ significantly between games.

4.2.5 Services, integrations and Settings

With the gameplay and all the visual parts adjusted, it was necessary to verify the settings and additional features or services the game might had been relying on, like online features or motion movements. In this case, the game was very simple and did not have any extra features besides audio, languages and vibrations, so the settings menu only covered those options. Audio can be considered intrinsic to any game, and it can be added with no effort

through the game engine Editor's interface most of the times. However, if the way the audio works in game is more complex than simply have a background song playing, and sound effects for the actions, it can be implemented via scripting with platform-specific functions or some universal ones.
For some controllers, like the DualShock, some audio can be emitted through the controller itself. This is how the configuration of audio specifically to one platform can create unique sensations.

Regarding languages and vibration, it is something that depends on the game, and it can be completely optional. Some games don't have language options, it's all in one language, and some don't even have a language since they don't possess any text or speech in-game. In *Game 1* the languages available are diverse, and since new text was added and some was edited, the localization team had to create new translations for those entries.
For the vibration, not only does it depend on the game, but also on the platform. Every platform can have vibrations except for PC (unless the game is built for PC, with controller support. In that case, vibrations are supported), but each one can be implemented differently. Game engines like Unity have some built-in methods that allow for universal vibrations (for PlayStation, Xbox and Nintendo Switch, at least) but each platform has their own vibration features. Implementing this should be considered carefully because it has both pros and cons.

Pros	**Cons**
Increased immersion	Higher battery usage
Can be used as "sensorial" cue for in game warnings	Possible damage to controller
Can create new mechanics in-game	Increased chance of causing a hand disorder on the player
	Wrong use can break the experience

Even though the original version had vibration on the mobile, it was decided not to add it to the port because it did not create the same experience on Platform X as it did on a mobile. When balancing the pros and cons of adding a feature like vibration, it's also necessary to consider the "feel". If the feature does not fit well in a new version, and does not contribute to anything for the game, then it should be considered removing it for the sake of the player's experience.

When removing, altering, or adding new content or features, like the vibration, the text and tutorials, and the new assets, respectively, these implementations should only happen after a contact with the client. *Game 1* was a project from an external client, so all the decisions that affected the game had to be verified with them. The development team has freedom over how the coding is made but not on matters that alter the port from the original. They can, however, during the development, make some considerations on changes they consider would benefit the game or that are necessary for the port to function, but those considerations need to be discussed with the client. Therefore, good and regular communication with the client is crucial.

For internal projects, any change from the original plan should also be communicated with the product owner/project manager/ team leader, depending on the organization's hierarchy, to prevent a developer to implement any change that will be discarded later, and making their work useless.

4.2.6 Save data

With all the features and functionalities done, and the game being playable as intended, the only thing that remained undone was the Save System. Different games have different save systems: autosaves (both per time, or in specific places, like checkpoints), manual save anytime the play wants, or manual saves only in specific places. Some games don't even have saves at all.

Due to the nature of *Game 1,* there was no need to save all the time. During a level, if the player quits, he knows he will lose the progress. Since the game is a puzzle game, level-based, the game only saves when finishing a level, both when winning or losing. The game only saves the scores and rankings of the players, so that when they load the game, they can see which levels they have or not beaten already, and what is their score in each one, so they can try to do better or simply proceed with the levels not beaten.
When dealing with consoles, there are some restrictions to saves, which can be due to the format or size of the files, due to the way it is implemented, among others. Therefore, everything must be thoroughly considered. *Platform X* has such restrictions, so to create the save system, it was used samples from the platform, which already had demos for saving. Using those samples and adjusting the save scripts inside so they could contemplate the information we need to save, the final step left was testing the changes.

Figuring out how to properly make the save systems can take a long time, because even by having some samples, there are always changes on the scripts specific to the game that is being made and the settings required. When playing directly through the game engine, like playing on Unity, the game may seem to run perfectly, or not even run at all, and when playing it on the dev kit, the behavior might be totally different. There might even be differences between the development build running on a dev kit, and the release build running on a normal platform. For that reason, it's recommended for this part, and all that include specific platform parts (like the controllers) to have a closer contact with the test team. Not only will it help to understand what happens when some errors occur, but since every change will require a new build, it will make the job simpler.

Since the game can only run the code that is specific to the current platform, and can crash if it detects another platform specific code, Unity has a simple solution that allows the same project to be used for different platforms, without needing to create new versions: the *define declaratives.* (#define directives, s.d.)
This is a feature that tells Unity to only run that specific code if the platform is the one specified in the directive. Below are two code snippets that demonstrate the use of this

feature. In the first Snippet, the code is only executable if the current platform is either a Nintendo Switch, a PS4 or an Xbox One. When referring to the “current platform”, the game must be running on the dev kit. If the game is running on Unity, regardless of the platform that is toggled on the Player Settings, it will be considered “UNITY_EDITOR”.

The second snippet prevents the code from running in that situation, by telling the system to only execute the Write method if the current platform is **not** Unity Editor. In this situation, it will run in any other platform, either the ones mentioned above, or Windows, Android, Linux, and any other platform supported by Unity.

```
#if UNITY_SWITCH || UNITY_PS4 || UNITY_XBOXONE
        Write();
#endif
```

Code Snippet 3 - Define declarative for some platforms

```
#if !UNITY_EDITOR
        Write();
#endif
```

Code Snippet 4 - Define declarative for Unity Editor

By restraining the code to run in the specified platforms, we can have the save script in the game with the reference to each platform save methods, without having the game crashing due to unsupported methods.

To test the save systems, which is something that can cause several issues on a platform, it should be done in both the platform and on computer, since inspecting files is easier on the computer. Even if some declaratives are implemented, the game should create the save data file on the computer, when playtesting, exactly as if it was the Platform X. The way it creates doesn’t have to be exactly the same, since some functions could be platform specific, but the output should be the same. In that way, it is possible to inspect the content of the saves. This inspection allows to read the information written (and compare it to the original game saves to verify if the right information is present), check if there are any *Nulls* presents due to some error in the code or corruption in the process, the format of the save and its size.
In this game, since the save is only happening at the end of levels, the test should be made several times, with different levels being played and different settings. It’s important to close and open the game sometimes to test the Load, as well as the reset/delete save data and ensure no remains of the previous save are present to create conflict with the newly save data.

4.3 Game 2

During the development of *Game 1*, due to some changes in the organization, *Game 2* also had its development started, which made both games being developed at the same time. Unlike the *Game 1*, this porting was being done side-by-side with the original game development. The client was still developing and creating the game while the team was adapting the game for Platform X, which forced the porting to be made through iterations. With the use of Branches and versions control, each team had their own branch with different version of the game. The client would change the code on their side several times while developing the game, and when they considered that a feature or part of the game they were working on was ready, the porting team would then merge those changes into their own version, so it would be possible to test and change what was made to support the new platform.

Even though the development of *Game 1* and *Game 2* were two completely different situations, the main topics focused during the porting were the same, although they were approached differently due to those situations. Since *Game 2* was started halfway through *Game 1* development, it was not required to do a Setup like in the first one since almost everything was already installed and was only required to start a new project.

4.3.1 Gameplay

Game 2 is a 2D story-based, adventure game in which you control a character and move around the world interacting with objects and people to proceed through the story. Although the game mechanics can be limited to those, the game it's more complex than *Game 1* due to it having a story, which implies different cutscenes, events and interactions.

The game was being developed with Unity, although with a much more recent version. This allowed the use of newer and more efficient features during the process. With the game being in its initial phase of development, there wasn't any functionalities and API's to remove like in *Game 1*, so it was possible to immediately start applying changes to the project.

4.3.2 Controls

As in the previous game, this game features different scenarios with different inputs from the player. For simplicity, they are represented in only two scenarios, the In-Game User Case and the Menus User Case, in which the later one represents different menus from the game, but which all feature the same logic of inputs.

Save data

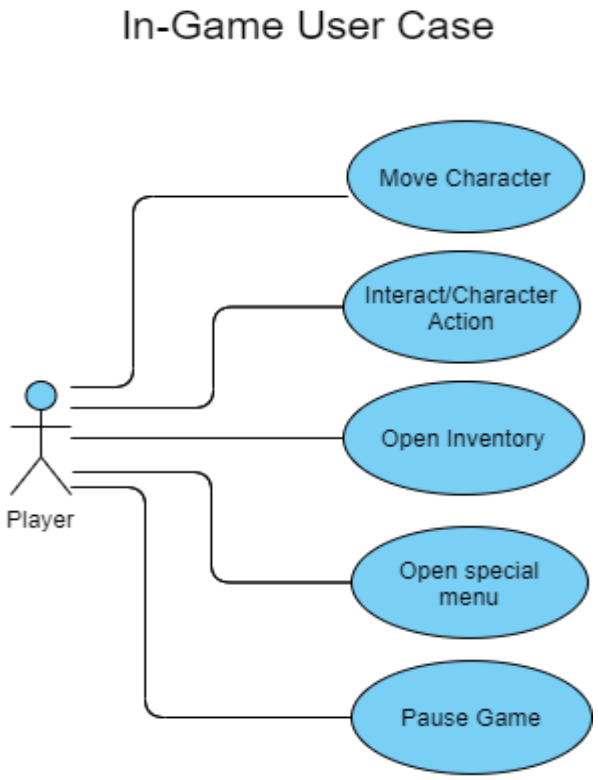


Figure 16 - Game 2 In-Game User Case

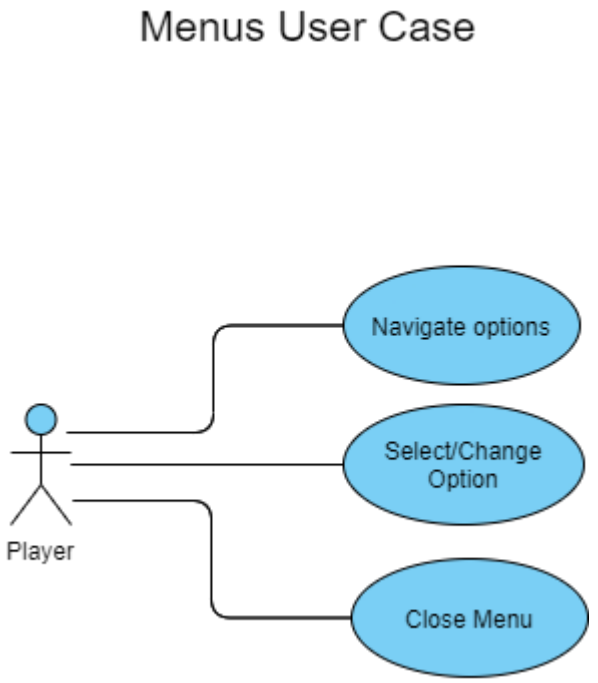


Figure 17 – Game 2 Menus User Case

For *Game 2,* the “face buttons” were used for the character’s actions and opening the menus, and for selecting options while in the Menus, since they were what felt more intuitive for the player to press for each action. As for the character’s movement, it was assigned to the analog stick due to it also being a 360º movement. On the Menus, it is also possible to use the D-pad buttons to navigate the different options. All the player’s actions are explained in-game, during the tutorial part, with buttons prompts showing up on the screen so the player can visually see which button to press.

As explained before, since the game was being created at the same time as the porting, there wasn’t a lot of content done to test, so while the client’s team was creating it, the porting team had time to implement and test all the inputs. With the use of one of the new Unity functionalities, it was possible to have different input mapping at once and with less resources. This allowed to define the inputs for all the necessary controllers, and even for a keyboard system (for testing directly on Unity’s Editor). This new input system simplifies the code by not being necessary to call each controller button press every time. Instead, the code calls an input action which was assigned to the input mapping defined before.

4.3.3 Art and audio

As in *Game 1* and as mentioned above, while the team was implementing the input actions on the application, the art team was responsible for creating the button prompts that showed up during the tutorial, and some that persist through the whole game. After verifying with the client if the assets created by the team were good to be added, the team inserted them into the game.

After each step done by the port team, and with each branch merge that would happen between the client’s team and the porting team’s branch, mainly to add more content, it was necessary to fully test what was implemented, as far in the game as possible. With each playtest, all bugs and issues detected were written and described in detail, with steps on how to reproduce each one. Those were later separated into issues that were porting-related, or the client’s side-related, so each side would know which issues each one had to resolve. With the game being in development, not every of those issues were possible to resolve at the time. For the most part, the issues persisted until the end when the game was being polished, excluding the issues that would crash the game. Those were always tackled immediately.

4.3.4 Save data

With the initial playtests finished, and after verifying the implemented inputs were working as intended, the team started working on the game save system, since the client’s had also already implemented it for the PC version. This game did not allow for manual saves, the saving was automatic. Using the original scripts for saving, when the game calls the save method, instead of calling the PC save method it would call a new method created by the team for the saves for Platform X.

With the use of more samples and with the previous work's experience, the implementation of the save system started more easily than in *Game 1*. However, the save for the current project needed to be more complex. Due to the way the client had created their code, especially their save system, the porting team couldn't create a new system. Instead, the team had to use several parsers, encoders, and different formatters to be possible to save the data. Due to the big quantity and complexity of the data saved, the team used the exact file that was created for PC saves and passed it through some formatters so it would be readable and editable on the Platform X. These processes took a few days to fully implement due to all the restrictions and requirements for the save data on Platform X, and to be able to use the formatters mentioned above with those conditions. With each attempt, a playtest was needed to determine if the saving was being done properly. By recurring to breakpoints and console logs, the team would be able to read what was being written or read every time a save or load happened to help pinpoint where the problems were located. After several tests and tweaks to the code, the saves ended up working as intended, but from that point forward, every time a playtest was conducted, the saves had to be tested again to ensure that the new content and updates to the game hadn't affect it, and the new content would also be saved.

4.3.5 Settings

Having the save system working, it was then required to create a second save, one related to the player settings, usually known as PlayerPrefs (especially on Unity). Unlike the game data that is saved, this specific data is called "Persistent data" and usually requires a different approach. These settings could be saved in several different ways: saved along with the save data, saved the same way as the save data but in a different file, or in a specific way used for PlayerPrefs data. All of these options have their advantages and disadvantages, so in order to use a more wide and generic option that would allow for future changes without compromising the rest, it was decided to use the second option. The team used the same save system but stored all the player's settings in a separate file. This way, the file could also be updated independently from the main save data file, and it would not be required to rewrite all the data stored in that file. Once again, this was thoroughly tested, by changing the settings at different points of the game, saving and loading several times, always ensuring the values would be the correct ones.

4.3.6 Services and integrations

During the development, the client decided to implement some features like vibration for consoles' versions. After having the previous steps tested and working properly, the team started searching for ways to implement vibration into the game. For this game, the idea was to use it on a few specific places in order to provide some extra "feeling" to the player, according to what was happening on the screen at that time. This meant that the calls for the vibration feature would have to be in the scripts where the story was, and not on the inputs part (like some games that produce a vibration every time the player presses a specific button

to do an action, like jumping). This feature could be implemented by two different ways: with the use of the platform specific code snippets or with some snippets that Unity provides that can be used for each specific platform. After testing both options, and according to what was intended, the team opted for the first option, using internal methods to make it work.

4.3.7 Optimization

All the features and implementations for the team's side were finished. Since the game was not completed yet, the team still had to constantly do several tests with every new update and merge that the client would do. Along with these tests, the team prepared the game for a pre-submission for the Platform X, so they could test the game in the current state and inform if there were any guideline that wasn't being respected, or any information that was wrongly provided in the meta information of the game (information like Parental Guiding rating, languages, features, etc). Besides the pre-submission, the team also started improving the performance of the game since it was not fully stable.

Regarding performance, the game would hit the expected FPS in most of the game (this would be different than the values expected on PC or the other Platforms, since each one has their own limitations). However, in some heavier scenes, the frames would drop significantly. Several settings were changed to try and get a better result, which included combinations of changes in the overall quality and some specific settings for textures, lightning and shadows, compressions, post-processing, among others. Most of the settings did not show a lot of improvement on the game's stability besides post-processing, which ended up being one of the heaviest settings. After finding a combination of those settings that did not compromise too much the quality of the game but that would increase the performance, it was possible to obtain an increase of around 15 frames per second in those heavier scenes, which made the game stable as intended.

4.4 Methodology's development

With the development of the two portings mentioned previously finished, and the study of previous portings and some of the projects the rest of the studio was working on, it became possible to start developing a methodology to help the development of future porting projects. This solution aims to be used independently of how the porting is being made. However, all the study subjects and the work done during this time period were all made with Unity. In this sense, some parts mention Unity-specific features to allow for a more concrete explanation.

A porting can be made in at least two different stages: after the original game is finished, like *Game 1* in which the game was already released in another Platform before, or the porting is being made simultaneously with the original game. In the second one, the porting is being

made at the same time but with some "delay", since the porting team can only work after having something already made by the main developers.

Regardless of when it's being made, the porting can be divided into several stages, with some being significantly bigger or more complex than others, and between all of them, testing is always crucial. The more tests are done during development, the less likely the porting is to reach the end phase with bad results. Even in small studios and indie projects, there can be someone that is responsible for those tests and doesn't do any development itself. This way, while the team is doing their job, that person can be testing everything that has been done. With several different stages that the development can be divided into, the testing should be present during all of them, and between each of them.

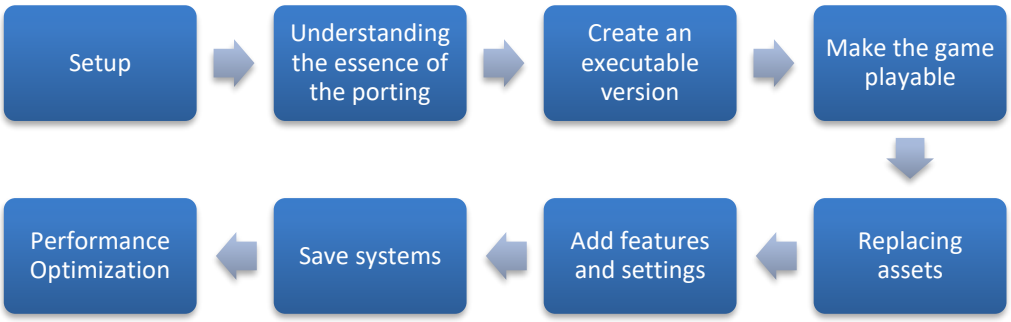


Figure 18 - Porting development stages

4.4.1 Setup

Before starting the development, and as early as possible, all the conditions should be ready, both hardware and software related. When the porting is being made for a console, like PlayStation, Xbox or Nintendo, the respective dev kit and SDK's are necessary, and those are only obtainable after registering as a developer for that platform. That platform needs to accept the registration in order for a developer to access the necessary materials and to be able to order a dev kit. Since this process can take a long time, it should be done as soon as possible. For solo developers, it is recommended that they register as a company, even if it is a company of one person, as the chances of being accepted are higher that way. In the case of a developer already working for a game studio, if they already own any dev kits, the developer should only need to register to be able to access the information regarding programming to that platform, specially to access forums and documentation, and to download and install the required materials, as the SDK.

4.4.2 Understanding the essence of the porting

With the setup process completed, and everything ready to start the development of the porting (or during the waiting time for the registration to be accepted, while the team doesn't

obtain the necessary materials), the next stage can be initiated. Before any development itself, the porting team should always playtest the original game first, especially if the game has already been released. When the porting is being developed simultaneously with the original game, this playtest has to be divided. It should be done every time the original game's team sends new updates to the porting team. This helps to ensure that the porting team gets the feeling of the game right. Reading the source code or watching a gameplay of someone playing the game doesn't provide the right and full experience of the game, which is what is intended to replicate, or improve, on the porting.

4.4.3 Create an executable version

After the initial testing phase, to start the actual development, the first thing to be done should be to make the game simply run on the desired platform. It's effortless to implement changes to the controls or creating a save system first if there isn't a way to test it and run the game. To achieve this, it could be necessary to remove or deactivate several features of the game, some libraries or modules used, or any other component that is platform-specific. A platform will not recognize code related to another, and that can cause the game to crash. Besides removing some of the code, it will also be necessary to add new code, according to the platform target, to effectively run the game.

On Unity, if the environments have been setup correctly, the desired platforms should appear in this menu. It is crucial to properly select the platform when doing development builds, as well as selecting the appropriate options on the right side and on the Player Settings.

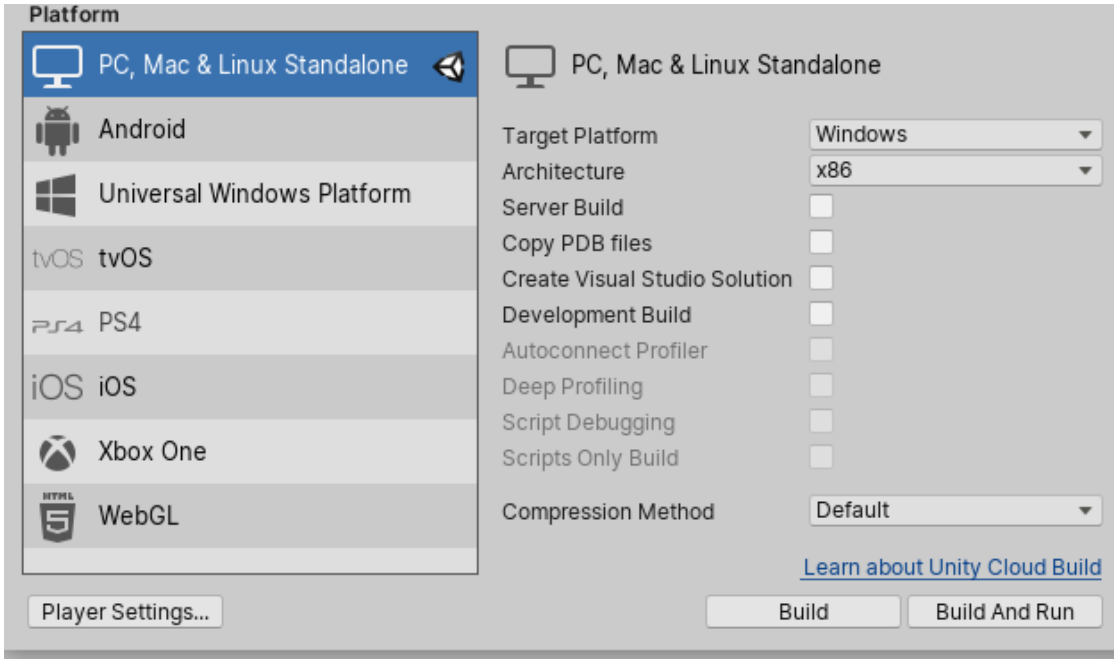


Figure 19 - Unity's builds menu interface

4.4.4 Make the game playable

With the game being runnable on the new platform, the following stage is to make it playable, by allowing the user to control the character or make some actions in-game. These actions will vary according to the game itself, as well as the target platform, since some actions could be exclusive.

The initial playtest that was mentioned before should have left the porting team with a good understanding of the controls. When adapting those controls to the new controller, the player should be able to do the same actions as easily and naturally as in the original platform. When implementing the new controls, besides assigning each button with some actions, in the case of controllers or a mouse it's important to consider the sensitivity values. Different platforms have different controllers, which have different values to settings, like the dead zone. Even though these values can differ between platforms, the output may still be the same, so it's important to configure those values and thoroughly test them to ensure they create an output similar to the original one. Although there aren't any rules defined as to what should each button do in a game, the player will most likely have an assumption made due to their past experiences. Most of the times, players expect a shooter or a racing game to have the main actions on the triggers buttons, or walk with WASD/arrow keys in a computer, and changing that can cause the player some discomfort and make them not enjoy the game. So, depending on the type of game, the controls should also try to follow the general trend of other similar games in order to make the controls easy to memorize. However, if possible, allowing the player to change the controls as they intend is considered a good feature that covers some accessibility issues.

4.4.5 Replacing assets

Porting a game isn't a work exclusive to programmers. The artists also have their own role on the projects. When the controls are implemented, it's then necessary to inform the player of those controls. Depending on the game, this can be done with tutorials, with some image (usually on the menu) showing what each button does, or even by interaction with some character or object in game, which will then tell the player the controls, via audio and/or text. All this falls into the art team's side. In some smaller teams, that job can fall under the programmer's functions, but for bigger teams, that is assigned to different people. Usually, this work will require the creation of new icons, symbols and images that represent the new controls as the original game does it. While the artists commonly have room for some creativity when doing their part, it's important to consult with the clients and follow the game's art style. Besides that, there are also some considerations when doing art that will represent anything related to the new platform. When doing the icons for the controls and adding the name of the new buttons, the team has to follow the platform's guidelines.

Due to the limitation of each platform's hardware, the original art can suffer some modifications too. In porting projects where the target platform is weaker than the original, different options can be made:

- Removing non-essential assets, like background assets, to lighten the rendering;
- Redoing part or all assets to simplify and reduce the number of polygons and details;
- Removing or reducing visual effects, like the overall visual quality on the engine's settings, or more specific settings like shadows, anti-aliasing, bloom, etc.

All these options reduce the visual quality of the game in order to try and make the game smoother and under the platform limitations. This is related to the performance part, which can be done in the end of the porting as a last stage, as described in the following pages.

For portings with a better target platform, the art can be left untouched and the game should automatically have a higher performance, or the assets and visual effects can be improved, like adding the use of Raytracing, if the new platform supports it.

4.4.6 Adding features and settings

Most games take advantage of extra features that the different platforms provide. These features can go from online features, like having multiplayer or co-op modes, scoreboards, or friend lists, to more basic features like the use of motion or vibration. Each game will have their own features, depending on the game itself and the platform, but in most cases, that dependency should not affect the porting. If a feature is platform-exclusive and it's essential to the gameplay, then making a port would not be a good option.

Considering the new target platform does indeed support the features that the original game had, this is a stage that will differ almost completely between projects. Not only does this depend on the platform, but also on what features are being used and how they are used. When porting a game, this should be intensively tested, especially if it involves online features, to assure everything turns out as intended, and there are no risks of one player getting another player's personal data and information.

Some of the features in a game can be hardcoded into the source code and not allow the player to change anything, while others can be changed by the player most of the times, like audio values and the strength and/or use of vibration. Those settings need to be saved in a place that allows the game to always be able to read them, so the player doesn't need to change them every time he opens the game.

Settings changed by the player, or sometimes called PlayerPrefs (on Unity), can be saved in different ways. They are data, just like any other data from the game, and they can be saved along with the game data's save file. If they are saved on the same file, every time the player changes any value, even if just reducing the Master Volume by 1 value, the system would have to open the save file, rewrite it, and save again, which can be a problem in big and heavy games. Some games "freeze" when doing a save since the operation can take a few seconds to complete, especially in big games, so the player would have those freezes every time they wanted to change any setting. In addition, the player can inadvertently cause some system overload if they do several changes repeatedly in a short period of time, which is likely to happen when the player is testing the settings in order to find what best suits them. In some

platforms, those repeated actions cause a guideline violation that can either crash the game, or not even allow the game to be published, since they do not follow the platforms rules. Another way to save settings is to create a save file, just as the one for the game's data, that will only store this information. This way, the file will be significantly smaller and have a lot less information than in the previous option, which won't cause any system overload when doing several save operations repeatedly.

When using Unity, there is also another possibility for this type of data that involves using Unity specific commands to save PlayerPrefs. This may require less intervention from the developer's part, but might also mean less flexibility, as some of those commands can't be reconfigured.

```
12    void Start()
13    {
14        //create a new player data
15        SavePlayerData savedPlayer = new SavePlayerData()
16        {
17            playerName = "Valhala",
18            playerHealth = 57.5f,
19            playerPos = new Vector3(10,7, 5.5f)
20        };
21
22        //use Unity's JsonUtility to save the above data.
23        //JsonUtility.ToJson will return a string so we can use it with player prefs
24        PlayerPrefs.SetString("SavedPlayer", JsonUtility.ToJson(savedPlayer));
25
26        //make sure to Save
27        PlayerPrefs.Save();
28    }
```

Code Snippet 5 - Saving PlayerPrefs on Unity

(PlayerPrefs, s.d.)

4.4.7 Creating Save system

With all the previous stages completed, and after doing playtests to ensure everything works as intended, in terms of development the only missing part is the save system. Although it was mentioned during the settings, a game's data can't be simply saved like the PlayerPrefs. Each platform deals with saving in different ways, and each one has their own restrictions. Depending on the original game and those restrictions, the new save system can be able to use the original save file in its original format and just require the addition of some methods to store it on the new platform, or it can require some formatters and parsers to adapt and convert the file to a new format that is allowed, which can turn the whole process a lot more complex. If the porting is for PC, then save system can be made in infinite ways. If it is intended towards a console, then the save system has to follow their documentation thoroughly to work or be accepted by the platform.
When following the samples and documentations, it is important to start testing in an ascending way. If the team tries to immediately save the game data as intended, chances are that the errors will be harder to fix that way, as the whole process is more complex. With the use of development features like the logs and prints, it is recommended to start by trying to write and read some simple characters to a file. It is possible that the writing works and

reading doesn't, or vice-versa, so using logs for every action can be a bigger help than using breakpoints in the IDE. Those logs might inform the team of what exactly is being stored. After ensuring that the game can write to a file, and read it when intended without any conflict, then the developers can move on to save the desired information. This way, it is more likely to succeed when saving the game's data since almost all the problems that should occur have probably been dealt with when using simple and smaller information. If some issues appear only with the game's data, often is due to either size or format of the data.

4.4.8 Performance optimization

At this point, the game should be ready to be played from start to finish without crashes and according to the platform's guidelines. However, that does not mean the game will provide the desired experience to the players. During the development, the use of all those playtest sessions during each stage helped to have a better product at the end, but even with that, the performance of the game can still be worse than what is required. When analyzing and evaluating a porting, it's not a matter of the game's story being all present on the porting or the gameplay content being the same. Players and critics will expect the game to use the power of the new platform as best as possible, and to make the game run as smooth as it can. When the community considers a porting to be bad, most of the times is due to the performance. If the game is targeted at a specific framerate, and during gameplay, especially in heavier scenes or during fighting parts, it's constantly falling down to lower values, the experience will be damaged by that.

In other cases, a game can be always stable at a specific framerate (usually 30 or 60 FPS), and still provide a bad experience. Some games sacrifice content or too much of visual quality in order to achieve those values, and that can be as harmful. The players won't enjoy the game as much if the difference in visuals is too brusque. The best way to ensure a good porting is balance. Sacrificing only as much quality as needed for the performance to be high enough, without damaging the visuals. Depending on the game, there could be several settings that can be adjusted to increase the performance. As mentioned during the assets section, settings like shadows and anti-aliasing are some of those, but everything that is related to post-processing can help. The rendering of assets can take a toll on performance too, if the number of polygons or assets is too big, but the post-processing techniques usually have a very high impact on FPS, since they aren't just loading an image but applying some filters to them to then render again. However, in terms of visual quality, sometimes the difference is very little, which makes them some excellent settings to change to increase performance.

Figure 20 - Demonstration of LOD usage

One usual tip used in game development that uses a good balance of performance and sacrificing visual quality is the use of LOD (Level of Detail). This is used to increase the performance on 3D games by creating some pseudo-layers between the player’s camera and the horizon in the game. The objects that are farther from the player will show with less quality and detail, and as the player gets closer to them, it will slowly increase their quality. This way only the objects that are close to the player are being rendered at full quality.

A low LOD can cause some objects to seem to just pop-in, or the difference when the asset is being rendered to a higher quality can become more noticeable to the player. The improvement in FPS by using LOD can go as high as 20%. (What does LOD Level of Detail do - Graphics Settings Explained, s.d.)

However, art isn’t the only part responsible for performance. The way the code is done will also affect it. The weight of some methods and functions in the source code can have the exact same impact as using some post-processing settings. On Unity, with the use of the profiler, a feature that allows to see what is being called in each frame, it’s possible to see

what methods are being called the most, and how much that affects the FPS.

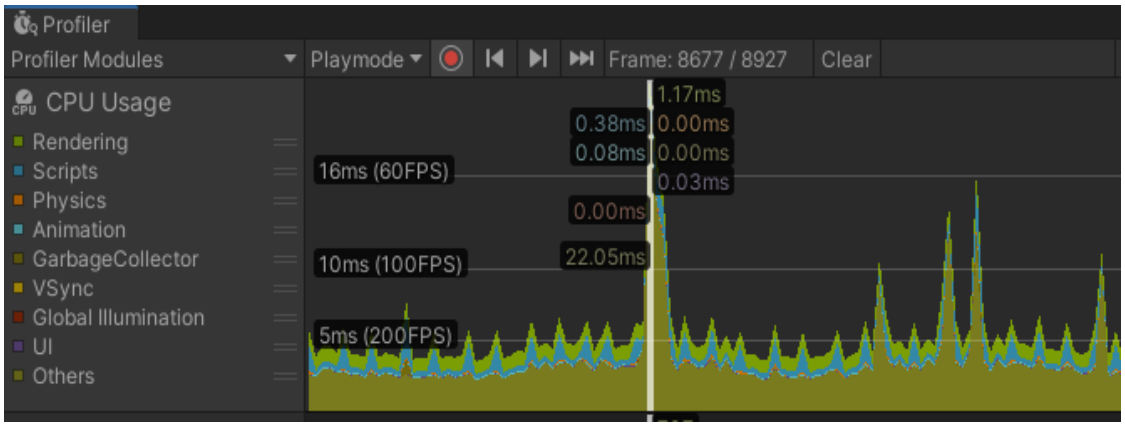


Figure 21 - Profiler's graph for each frame during a playtest session

Figure 22 - List of all the calls for the current frame, with the weight of each

A lot of those methods include math, which sometimes include a lot of complex operations. The movement of a character can cause several methods to be called thousands of times per frame, and all those calls will have a corresponding weight. Even using *integers* versus using *doubles* or *floats* can have an impact on the performance.

In the end, every good practice for common programming can be considered for game development. There might be several possible approaches to do a specific action in a game, but some might be simpler to process than others. Optimization of a game, or of the way someone codes is a never-ending process. The more experience someone has, the better they should be at coding and improving their games.

5 Evaluation and Experimentation

In order to determine the effects of the methodology and evaluate its results, it was necessary to undergo a process of testing and evaluation. The users to whom this methodology could help, developers that work or have worked in any portings, were the ones evaluating it. This chapter describes the different hypotheses, and the way were tested. It also presents an analysis on the final results to verify the effectiveness of the methodology created.

5.1 Hypotheses

To test whether the methodology created is actually useful in aiding the development of a porting among game developers, two hypotheses were tested:

- **H0 –** The use of this methodology does not allow the developers to reduce the developing time. The process described is not explained sufficiently and does not approach the important steps that should be taken during the development.
- **H1 –**The use of this methodology will allow the developers to reduce the developing time by describing the process for the porting, with the different steps and stages that a porting will be submitted to, and how to efficiently approach them. The methodology will also be simple enough for the developers to easily understand the process without external help.

To evaluate this solution, three different criteria were established in order to provide concrete evaluation aspects. Those criteria are the following:

- Precision – If the methodology proves to be correct and if it actually helps the development, or if the content is wrong and results in errors

- Relevance – If the methodology provides information that is important to the porting process and if there are important information lacking, or if it's information that, even if correct, does not help the porting at all
- Simplicity – If the methodology is easy to understand and apply when being used by a developer on his own, or if it's too complex.

5.2 Evaluation methodology

The evaluation methodology chosen for this project was the use of a questionnaire among a specific target audience.

The target audience are game developers from different countries and game studios, or even independent developers, in order to have a more diverse group of people with different backgrounds. This is an important aspect since the way people learn to code and develop games is different between places, especially between companies. It was expected to have, at least, 15 different respondents so the results could be more significant. However, only 11 respondents gave their opinions.

In order for the users to test and evaluate the solution, a document with the methodology, which described each stage of the process separately, was given to each of the respondents so they could develop a porting while applying it or, if it was someone with previous experience on portings, they did not need to apply the methodology in a project. Instead, they could just answer the questions based on their previous knowledge, and how they considered that it would affect the development. The respondents could participate in this test regardless of what game or platforms they had worked with.

After reading and analyzing the methodology, they were provided with two sets of questions: one regarding themselves, to have a profile of the developer, and one for the actual methodology.

Table 4 - Questions about the profile of the developer

Questions	**Answer system**
1- What is your age?	Any number
2- What's your nationality?	Open answer
3- Are you a solo indie developer or do you work for any game studio?	○ Solo ○ Working for a company
3.1- Which studio?	Open answer (optional)
4- Have you done any porting before?	Yes/No
4.1 – Did you had any documentation, samples, or methodologies to help you in the development of those portings?	Yes/No

Table 5 - Questions about the use of the solution itself

Questions	**Answer system**	
5- Did you apply the methodology in your porting or are you only evaluating it based on your previous experience?	○ I made a porting while applying the methodology	○ I am only evaluating the methodology based on previous experience from portings
6- If you are not under any NDA and can answer, what game did you port?	Open Answer (optional)	
7- What platform did you port to?	Open Answer (optional)	
8- How long did this development take?	Open Answer (optional)	
9- On a scale from 1 to 5, with 1 being "It was all wrong, it only created problems" and 5 being "All of the methodology was correct. It didn't cause any problem", how was the content presented in each section?	Rate from 1 to 5 each section of the methodology	
10- On a scale from 1 to 5, with 1 being "The methodology was useless" and 5 "The methodology was very important", how relevant for the porting was the methodology in each section?	Rate from 1 to 5 each section of the methodology	
11- On a scale from 1 to 5, with 1 being "Extremely hard" and 5 being "Extremely simple", how simple was the methodology to understand on your own in each section?	Rate from 1 to 5 each section of the methodology	
11.1 – If you found it hard, please explain why	Open Answer (optional)	
12- Were you able to finish the porting?	Yes/No	
12.1- If not, why weren't you able to finish? (e.g. The project was too big, the project was cancelled, etc)	Open Answer (optional)	
13- Overall, on a scale from 1 to 5, with 1 being "Didn't help at all" and 5 being "Helped significantly" how do you consider that this methodology helped you during development by reducing development time?	Rate from 1 to 5	

14- Overall, on a scale from 1 to 5, with 1 being “Not helpful at all” and 5 being “Very helpful”, how helpful do you think the use of this methodology or an improved version of it, as a standard, would be to developers, especially indie/ solo developers?	Rate from 1 to 5
15- Please write any feedback or suggestion	Open Answer (optional)

5.3 Results analysis

This section describes and analyzes the answers obtained through the questionnaires that were answered by the target audience.

In order to reach different potential respondents, several channels were used. From social media like LinkedIn and Twitter, to forums and servers of game development, and even by personally trying to contact several game studios, porting companies and developers. However, despite the extensive and prolonged attempts, most were not willing to collaborate and only 12 people answer the survey.

Every participant was personally contacted to properly explain what this test involved and to assure that no confidential information was being presented or asked of them, as this was one issue that some people were not feeling assured before joining. After the explanation and having their confirmation, each person was handed a PDF document with the methodology, and a link to the questionnaire, which was made using Google Forms due to the simplicity for users to answer, and for collecting and compiling the data.

5.3.1 Questionnaire Results

This section will be divided into three sections. The first one will present the answers regarding each person, the ones that allow to create a profile of the developers. The second one is related to the answers regarding the methodology itself. As for the last one, it features all the feedback provided by each of the respondents.

5.3.1.1 The developers

Based on the data gathered from survey, the group of respondents had between 22 and 45 years old and are from 4 different countries, although 5 answers are shown, since two different answers are related to the same Country

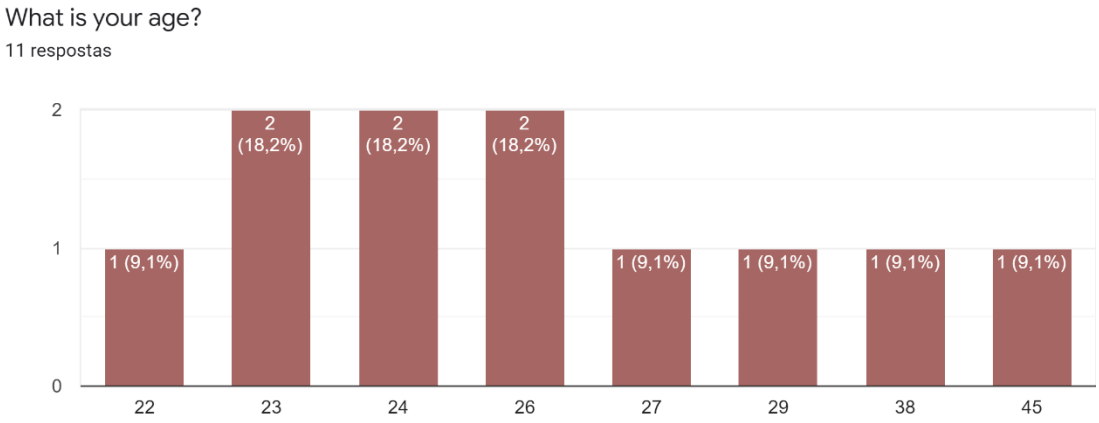


Figure 23 - Age question

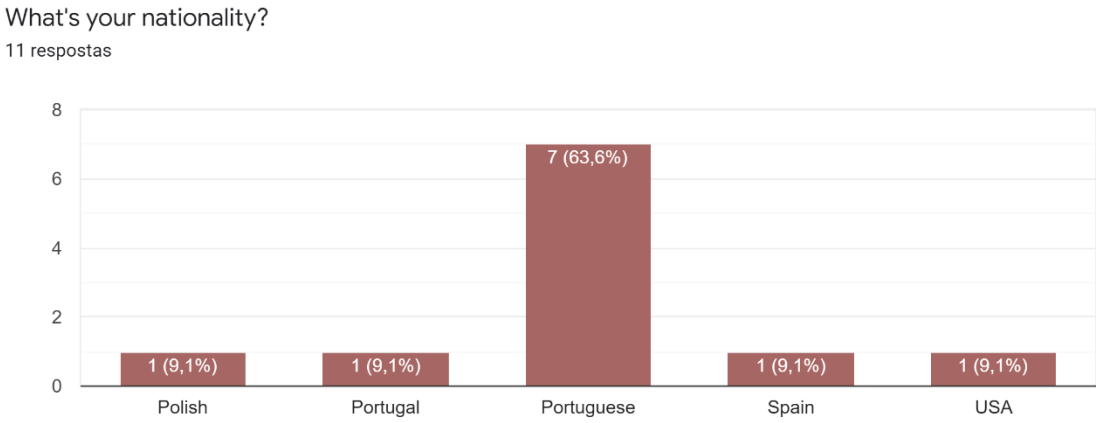


Figure 24 - Nationality question

From all these developers, 72.7% are currently working for a company, which ensures that most of the answers were given by someone really working in the area and with some professional experience. Although the other 27.3% might be just as experienced, they might also come from someone that has only worked in a porting as a personal experience.

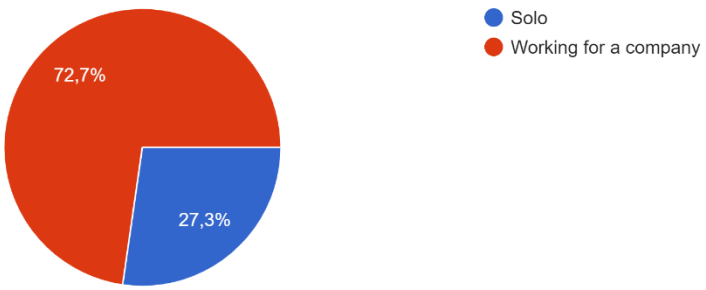


Figure 25 - "Are you a solo indie developer or do you work for any game studio?" question

To the 8 people that answered they were part of a company, it was asked which was that company. Based on their answers, they work at 5 different game studios, with one of them being the acclaimed Epic Games, owner of Epic Games Store and Unreal Engine, a powerful engine for developing video games.

All the respondents were inquired whether they had done any porting before or not. This question serves to understand the experience each developer has. From the 10 developers that answered affirmative, it was then asked if they had any support while doing those portings, or if they had to do everything from zero. Only one of those developers answered to not have any kind of support while doing it.

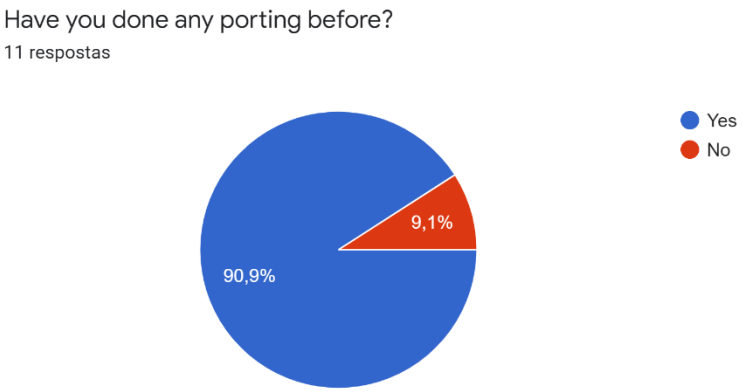


Figure 26 - Porting experience question

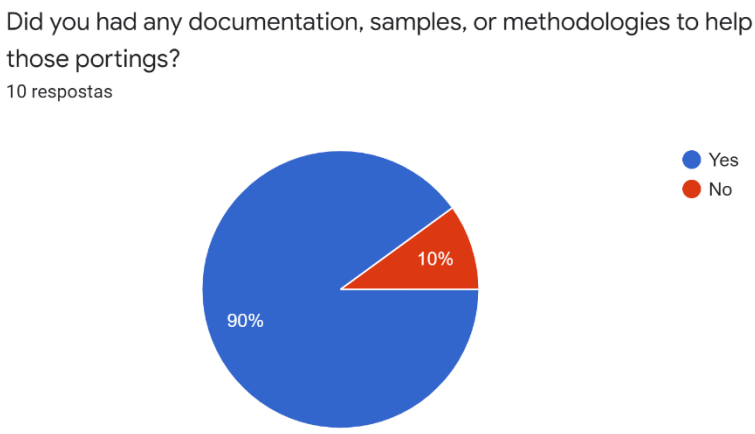


Figure 27 - Previous porting's support question

Based on these previous results, it's possible to say that most of the respondents have professional experience on the area, making them suitable developers to rate the provided methodology. Most of them also mentioned having some document or help while doing their porting, which creates a perfect subject for comparison between what they had before and what was given to them during this test.

5.3.1.2 The methodology

To start analyzing the answers regarding the methodology itself, it was necessary to separate the answers from those that used it during a port from those that have only answered based on their previous experiences since the answers could possibly differ due to that reason. Based on the answer, each person was given one of two different set of question. The questions are, however, essentially the same.
Having that concern in mind, the following question was asked, in which only one developer answered to use the methodology during a porting project.

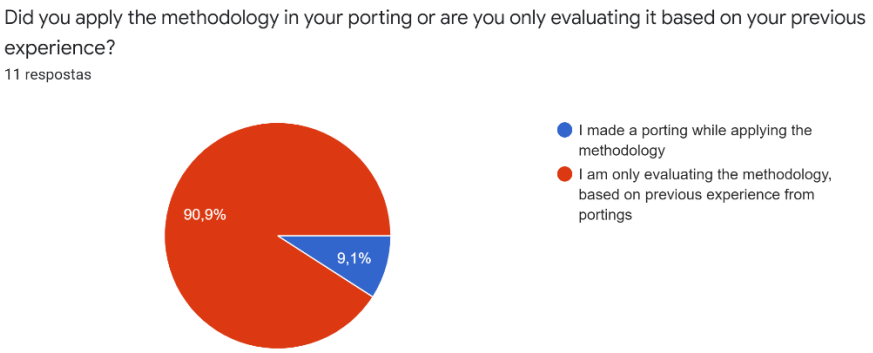


Figure 28 - Sorting question to separate the two different situations

For the developer that mentioned that used the methodology during a porting development, a question regarding each of the three different criteria that are being evaluated was asked. In each question, it was asked to rate each section of the methodology on a scale from 1 to 5.

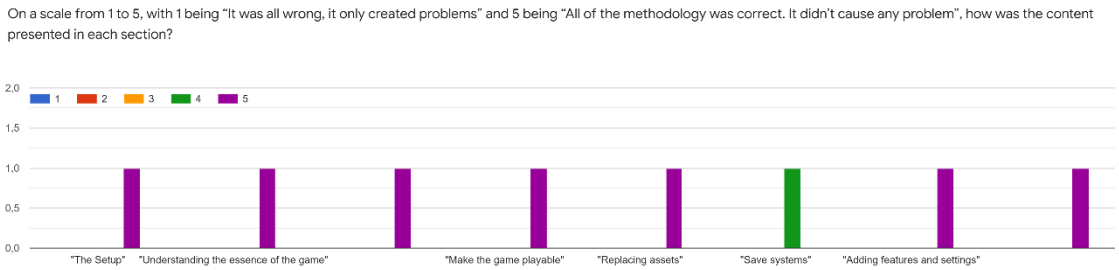


Figure 29 - Ratings of the "Precision" criteria

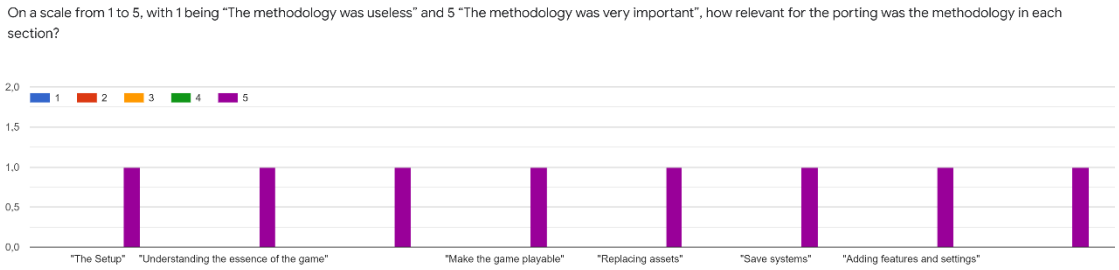


Figure 30 - Rating of the "Relevance" criteria

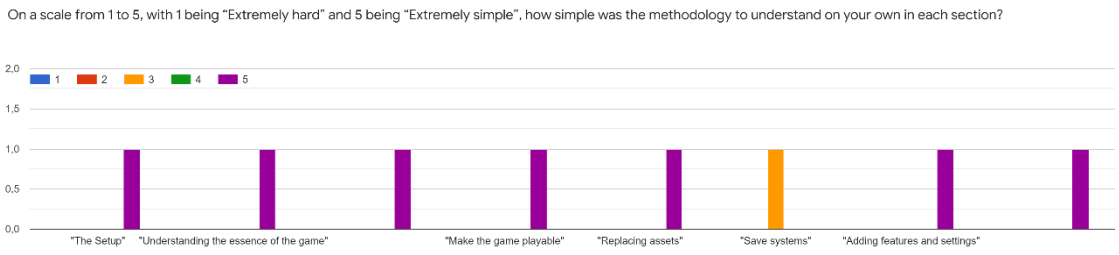


Figure 31 - Rating of the "Simplicity" criteria

After having inquired about each of the individual criteria and for each stage, it is possible to easily see what was considered to be better or worse developed on the methodology, with everything having the highest grade with the exception of the "Save systems" section.
Since the developer also answered that was able to finish the development of the porting, another two questions were made to inquire about the overall opinion of the methodology as a whole: if the methodology did indeed help reduce the development time, and if the

methodology would be helpful, as a standard, for developers (especially indie / solo developers).

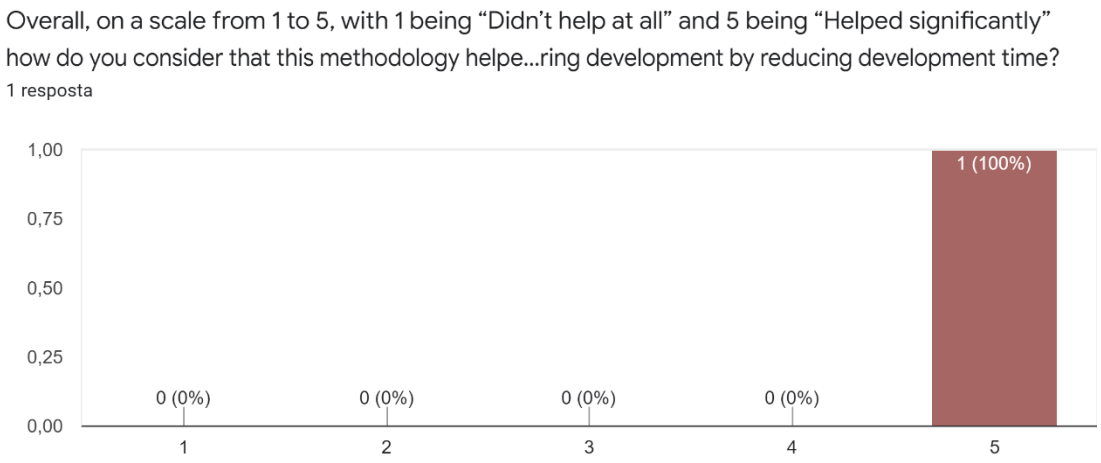


Figure 32 - Helpfulness of the methodology in the respondent’s project question

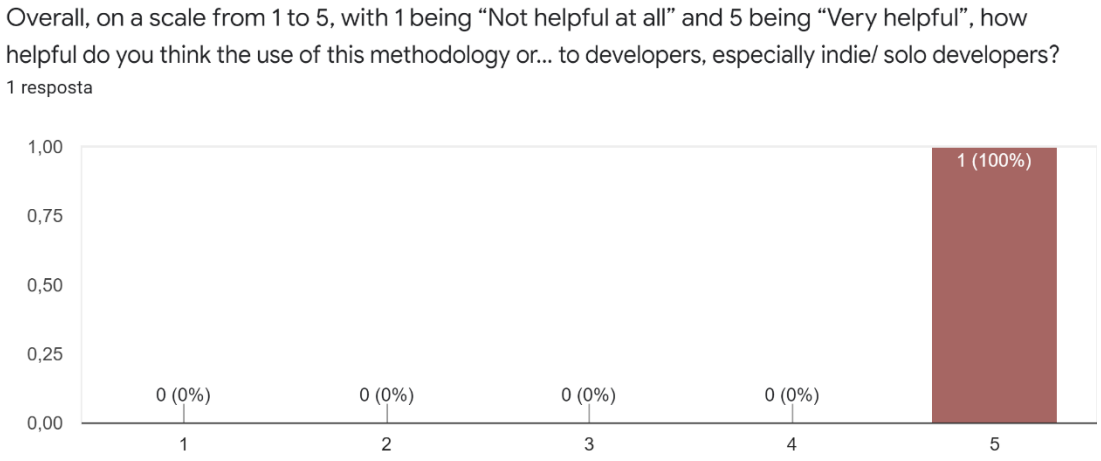


Figure 33 - Helpfulness of the methodology for any developer question

Despite the respondent not considering the methodology to be perfect, as it was shown that there was a section that was not rated as high as the others, in the end the overall opinion is that the methodology still greatly fulfils its purpose by helping the development, according to the answers provided.

Going back to the remaining group of respondents that were separated on a previous question, the following results are those provided by those people.

As mentioned before, the questions were essentially the same, but were made separately for the sake of data gathering.

To start, just like in the previous set of questions, it was asked a question to rate each of the three criteria stipulated for the project.

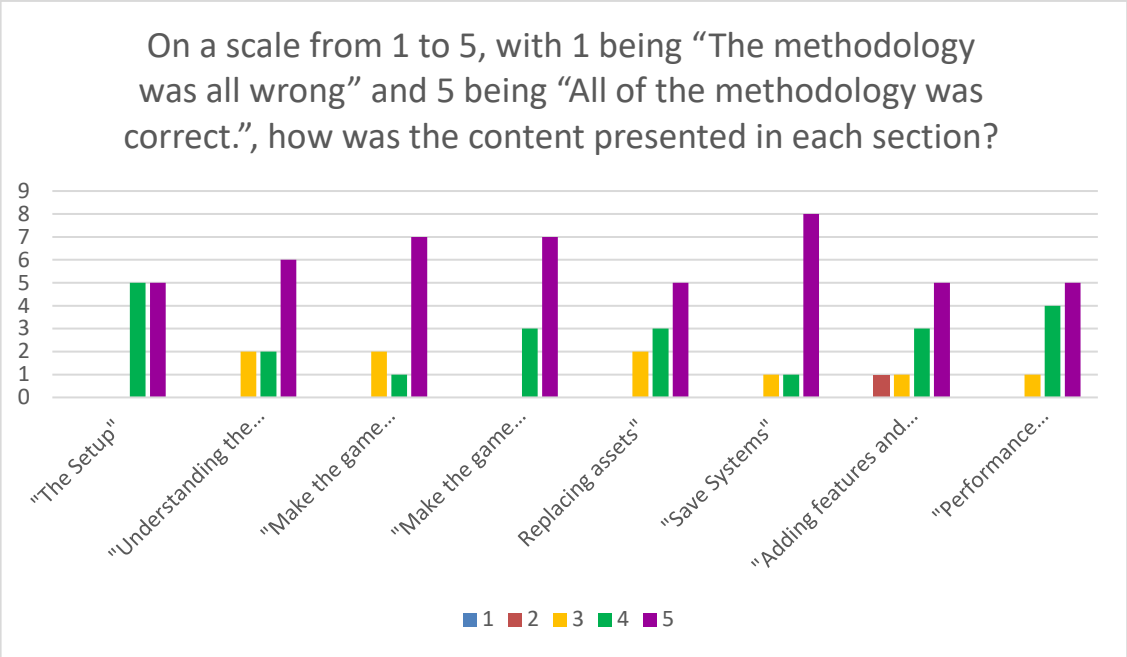


Figure 34 - Ratings for the "Precision" criteria

In contrast to the first set of questions, this features a significant increase in terms of respondents. As expected by that increment, the opinions for each section of the methodology are more diverse. By looking at the different graphs, it’s possible to see that in each section, at least half the respondents gave the highest rating. However, every answer also features at least one rating of 4, which makes every section being graded positively (as the 3, in this case, is considered a neutral rating, and every grade above is positive and every grade below is negative).
Despite each section being mostly positive, the “Adding features and settings” shows to have one negative rating. That, with the fact that 50% answered below 5, suggests that the section might need a review of the information provided, as the ratings suggest that some parts might not be correct.

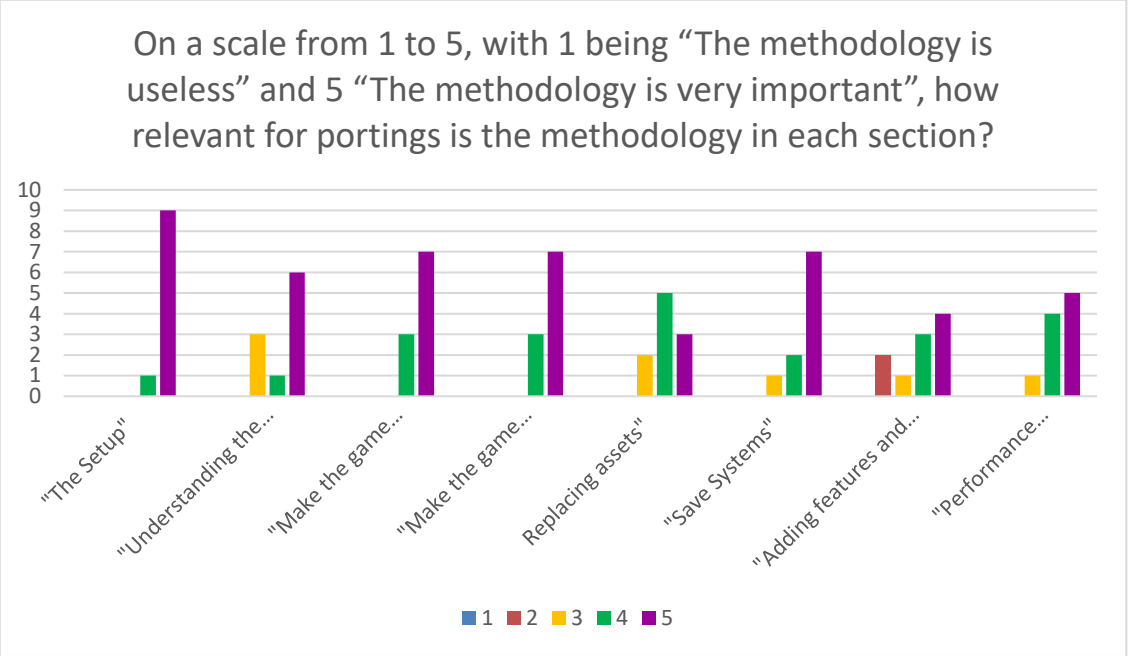


Figure 35 - Ratings for the "Relevance" criteria

Regarding the relevance of the methodology, it’s possible to see that most sections also present a high rating. In comparison to the first graph, there are some differences that are worth highlighting. “The Setup” section shows a 90% of “5” rating in this question, while before it was only 50%. This is an enforcement to show that the different criteria are independent from each other. In this case, that section is said to be very important, however, what is written in it might not totally correct. Some crucial information might be missing, or some of the information provided could have been inducing some mistakes.

In the previous question, all sections were rated with at least 50% with the highest grade, but that is not the case in the current one. The “Replacing assets” and the “Adding features and settings” have less than 50% for that grade. The later section also presents two ratings of the level “2”. Along with the analysis from the previous question, that showed that this same section has the lowest rate, this question is reinforcing that this section is requiring a review and remake of the content written.

Finally, the results for the third and last criteria, the “simplicity” are displayed below. This question focuses on the writing itself. If the developers were able to read and understand what was written. The experience is not the same for every developer and due to that, some terms and expressions might not be familiar to some. This can provoke some errors if the person reading understands what is written in a wrong way.

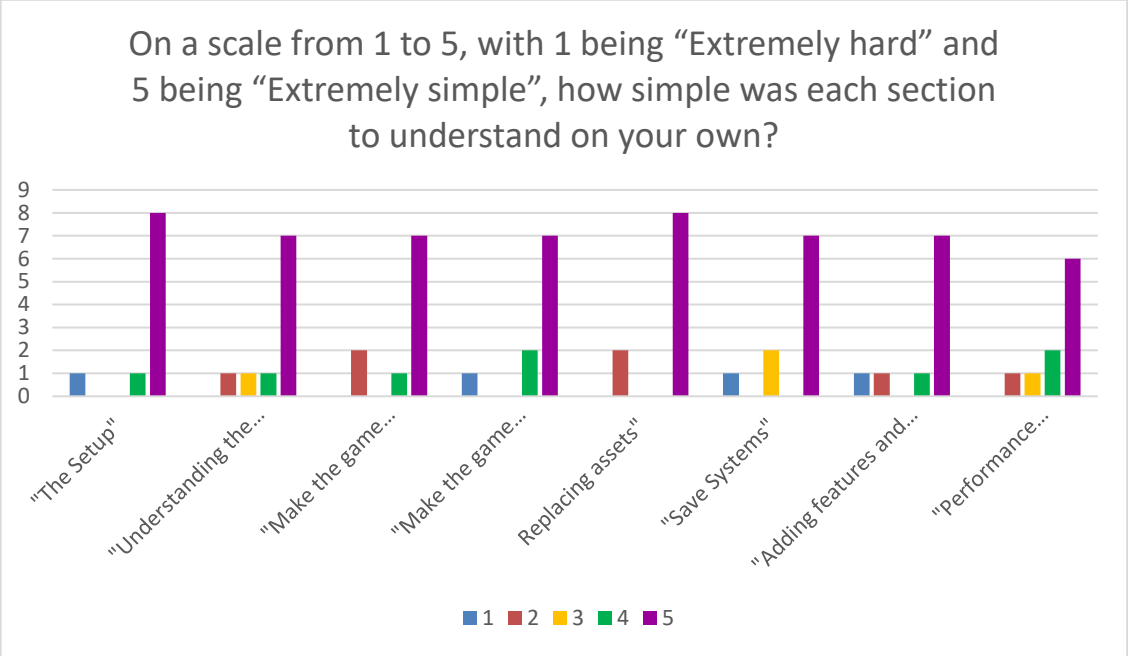


Figure 36 - Ratings for the "Simplicity" criteria

For this question, it’s immediately visible that almost all the sections have been rated mostly with “5”, showing that the methodology is accessible for most of the respondents. However, and unlike the previous two questions, this features several “1” ratings, and at least a negative rating in each section. These negative ratings might be due to several reasons, like poor written sentences, a confusing explanation or the use of unknown terms and expressions, since it was the same user giving ratings of “1” on those four sections, and giving a “2” on the other ones. However, this doesn’t invalidate any of his answers, since there could be more individuals in the same situation as this respondent’s.
Considering all the answers, once again the “Adding features and settings” section shows signals of problems. Based on the three different questions, this section shows to be the most problematic and worse rated, needing an improvement.

After inquiring about each criteria individually, the respondents were asked if they considered the methodology, overall, to be helpful to developers

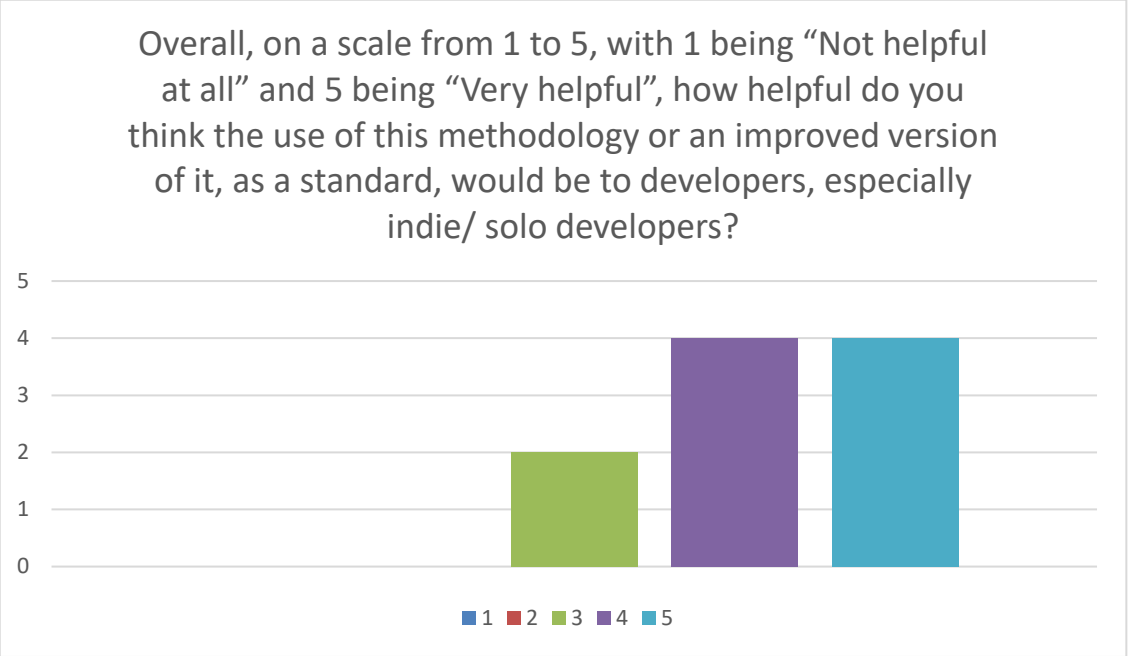


Figure 37 - Ratings of the methodology as a whole question

Based on the previous answers, 40% of the respondents rated the methodology, as a whole, with a rating of 5. Considering the rating 3 as neutral, it was obtained an 80% of positive approval. With these results, the methodology proved to have a positive outcome. However, the results were not perfect. Each section had different ratings, and some were better than others. For that reason, it was calculated the average rating for each section, using the results from each question.

As expected, the “Adding features and settings” showed to be worse than all the others, having an average of 4,1 while the highest section had an average of 4,63. Despite some negative ratings during the individual answers, every section ended up with very positive results, all above 4.

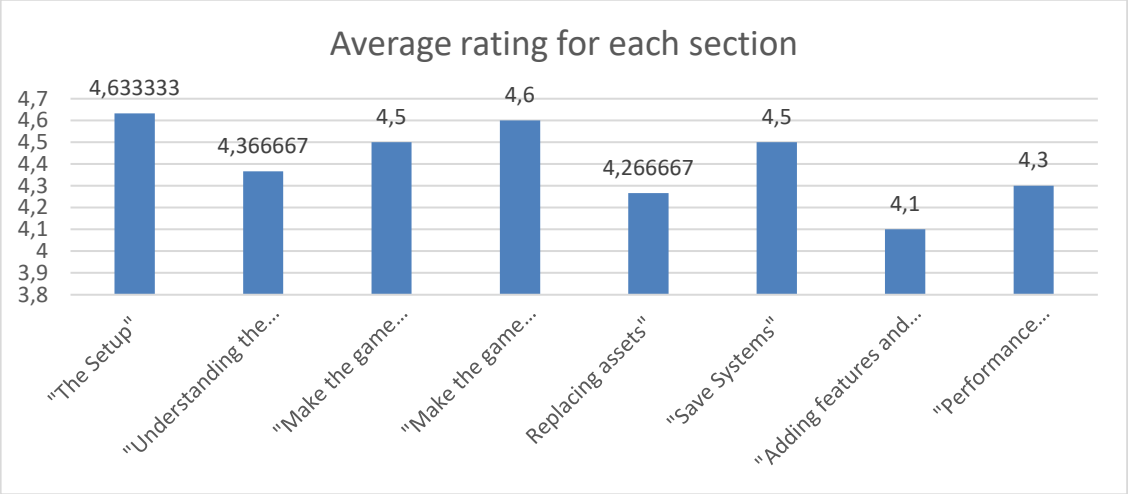


Figure 38 - Average rating for each section

After getting the average rating for each individual section of the methodology and having the rating of the methodology as a whole, it can be concluded that the hypothesis that was being tested, which defended that the use of this methodology would help developers reduce the development time by efficiently explaining the process, while being simple to understand, was successful validated based on the gathered data.

5.3.1.3 Feedback

In each survey, after answering all the other questions, each developer was asked to provide any additional feedback they considered to be worth sharing. Despite not having feedback from all the respondents, the ones that answered provided some extensive and important feedback.

Some developers considered that since this methodology had to be made in a universal approach due to all the NDA's barriers, some sections lacked some specific information that would make a great difference. As a workaround, it was suggested to create different versions of the methodology, one for each platform. Then, it would be possible to share some more concrete information and examples between forums and portal of registered developers of that platform.

Several developers also mentioned that the use of images, graphs, screenshots or even some code examples would be helpful to better understand the methodology.

A few of the developers have given extensive feedback in each section individually about things they considered too vague, important information that could be added and some aspects that might transmit a wrong idea with the way the information is written.

Since it was said that everything was done with Unity, it was also recommended to use more concrete explanations regarding those subjects, since they aren't under such NDA's as the platforms.

Lastly, the lack of a submission stage was also one of the most mentioned aspects. The submission stage is a phase after finishing all the development and optimizations, in which the teams need to undergo several validations processes to release a game. This might include several submissions, which then require changes to the game until it is approved by the platform. This involves both programming and some management, as the process needs to be done along with the platforms and the project managers/leaders or the product owner. In a big studio, such as the AAA studios, the developers probably won't ever need to handle this management part, but for a small studio or a solo developer, since they are the ones doing all the work, it is indeed a necessary step to know.

6 Conclusion

This chapter discusses the objectives that were accomplished and the ones that were not, problems that occurred during this project, the possibility of future works and the final assessment of the methodology developed.

At the beginning of the project, the established objective was to create a methodology that would help reduce the development time of a porting project. That methodology would also have to be simple to understand for the developers using it. Based on the ratings from the inquired developers, it's possible to say that all these objectives were accomplished. With this, the hypothesis that this project intended to assess on section 5.1 was validated.

Despite having accomplished the objectives proposed, the solution was not perfect. Based on the ratings given by each developer, and essentially due to their feedback, in some ways this methodology continues to have limitations. For an improved version, or for future works, some of the suggestions would be followed and this methodology would have better examples to provide explanations, either from some personal project or from open-source projects. This way all the NDA's would still be respected. Besides that, the methodology would be divided in several: divided based on the engine (Unity vs Unreal Engine, as they are the most powerful free engines, and the most used), and based on the platforms. As mentioned in the feedback section, this way, using the platforms portals and forums, concrete and detailed information could be given without having legal issues.

This project also intended to fight the competitive and restricted environment behind the gaming industry as most of the crucial information are locked behind several barriers. For new developers, and those working alone, it could be difficult to find out how to break those barriers. During this development, it was possible to start to understand more about those barriers, and why they exist, and at the same time, how to go around them. This methodology proves that it is possible to safely provide information regarding game development and portings without violating contracts. It is the first step in that direction and in the future, with more experience and improvements on this idea, it is possible to create complete and whole methodologies for anyone to access that could contribute immensely for the developments.

7 References

#define directives. (n.d.). Retrieved from Unity: https://docs.unity3d.com/2019.4/Documentation/Manual/PlatformDependentCompilation.html

EINOV. (2021). Retrieved from EINOV 3 - Value Proposition, Problem Statement e CANVAS - EN: https://moodle.isep.ipp.pt/pluginfile.php/161538/mod_resource/content/2/EINOV%203%20-%20Value%20Proposition%2C%20Problem%20Statement%20e%20CANVAS%20-%20EN.pdf

FUNCTION ANALYSIS SYSTEM TECHNIQUE (FAST). (n.d.). Retrieved from https://www.valueanalysis.ca/fast.php.

Guillermo Arango, I. B. (1985). *Maintenance and Porting of Software by Design Recovery.* California.

Marco D'Ambros, M. L. (2010). *On porting software visualization tools to the web.* Retrieved from https://link.springer.com/content/pdf/10.1007/s10009-010-0171-9.pdf

PCGamer. (n.d.). *Dark Souls producer admits PC port was rushed, promises sequel will "be a good PC experience"*. Retrieved from https://www.pcgamer.com/dark-souls-producer-admits-pc-port-was-rushed-promises-sequel-will-be-a-good-pc-experience/.

PlayerPrefs. (n.d.). Retrieved from Unity: https://docs.unity3d.com/ScriptReference/PlayerPrefs.html

Polygon. (n.d.). *Witcher 3 on Nintendo Switch only makes a few sacrifices*. Retrieved from https://www.polygon.com/2019/10/15/20915391/witcher-nintendo-switch-runs-performance-framerate-resolution.

Salen and Zimmerman. (2003). *Rules of Play: Game Design Fundamaentals.* Retrieved from JOSER.

Salen and Zimmerman. (2021). *Digital Games.* Retrieved from JOSER: https://moodle.isep.ipp.pt/pluginfile.php/95634/mod_resource/content/1/Aula3%202015-2016.pdf

Schell, J. (2008). *The Art of Game Design.*

Screenrant. (n.d.). *Ori and the Blind Forest Runs Better on Switch Than on Xbox*. Retrieved from https://screenrant.com/ori-blind-forest-runs-better-switch/

Tammay Bhowmik, V. A. (2013). *Porting Mobile Games in an Aspect-Oriented Way: An Industrial Case Study.* California. Retrieved from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6642506

Tarja Susi, M. J. (2007). *Serious Games - An Overview.* Sweden. Retrieved from https://www.diva-portal.org/smash/get/diva2:2416/FULLTEXT01.pdf

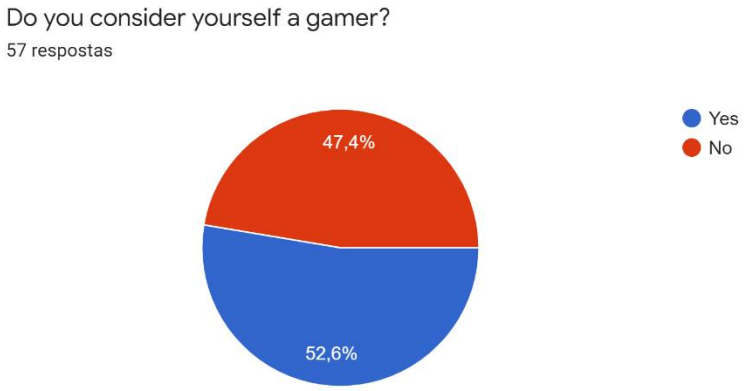
Visual Capitalist. (2020). *50 Years of Gaming History, by Revenue Stream (1970-2020)*. Retrieved from https://www.visualcapitalist.com/50-years-gaming-history-revenue-stream/

Annex

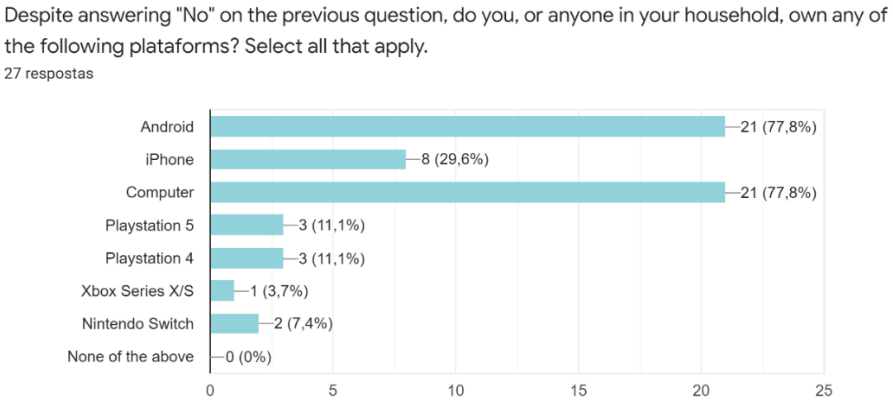
Games and porting questionnaire

Here is presented the survey made for the introduction chapter. This was shared in several forums and social medias to reach the largest audience possible. The purpose of this survey was to determine how many people have already played a videogame, even if they do not consider themselves a gamer. Besides that, it was also intended to determine the view of users on porting.

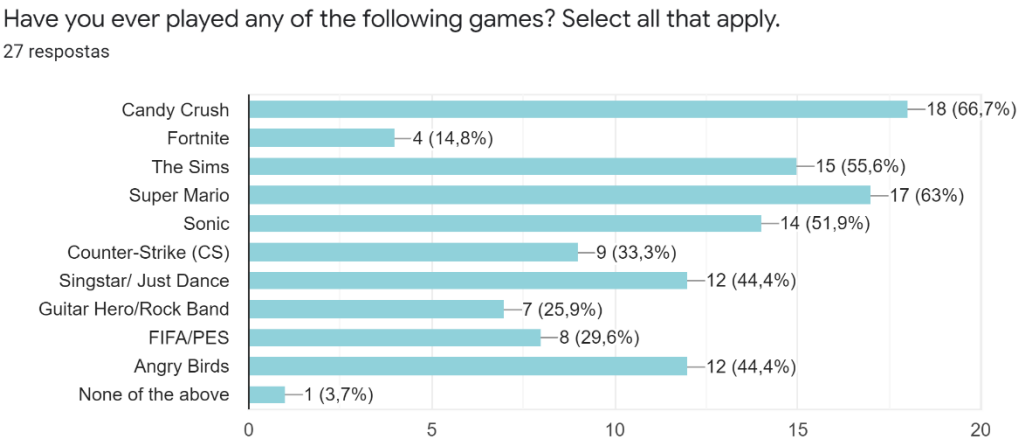
One the first question we can see how many users consider themselves a gamer, regardless of what they consider the term to be.



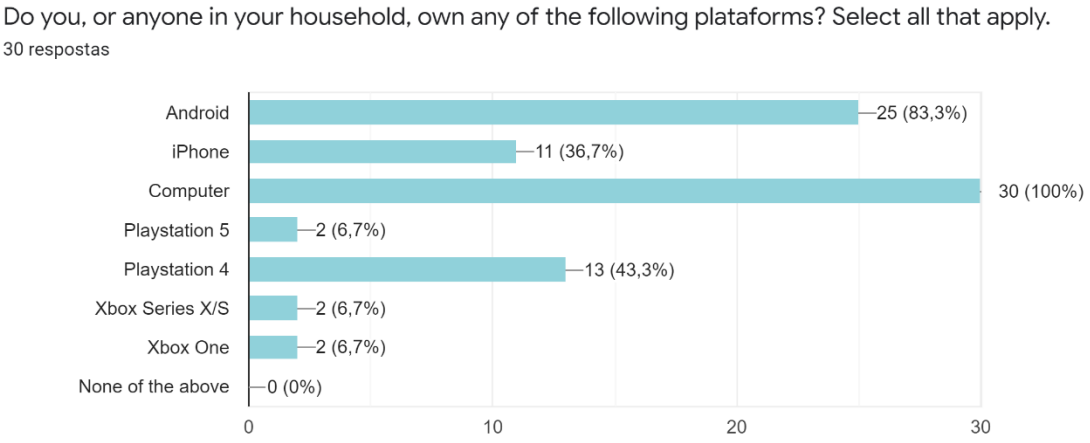
For the ones that do not consider themselves to be gamers, we determined what plataforms they ow in their house, if any at all.



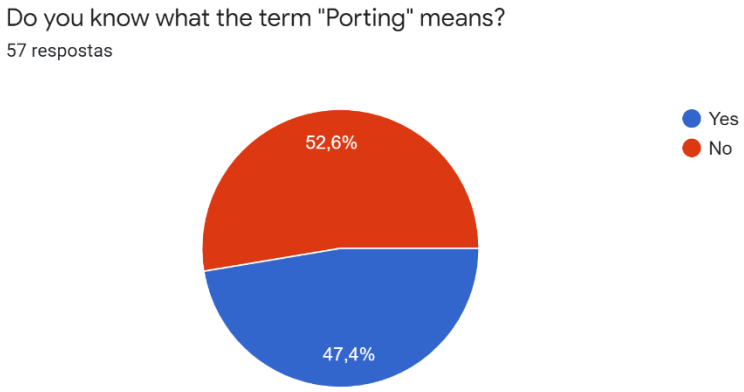
Regardless of the consoles and platforms they own, a group of several common games were presented for those users, and they were asked to select all the games they had already played. This question served to prove that people that do not consider themselves gamers also play games, and therefore might be possible clients of a ported game.



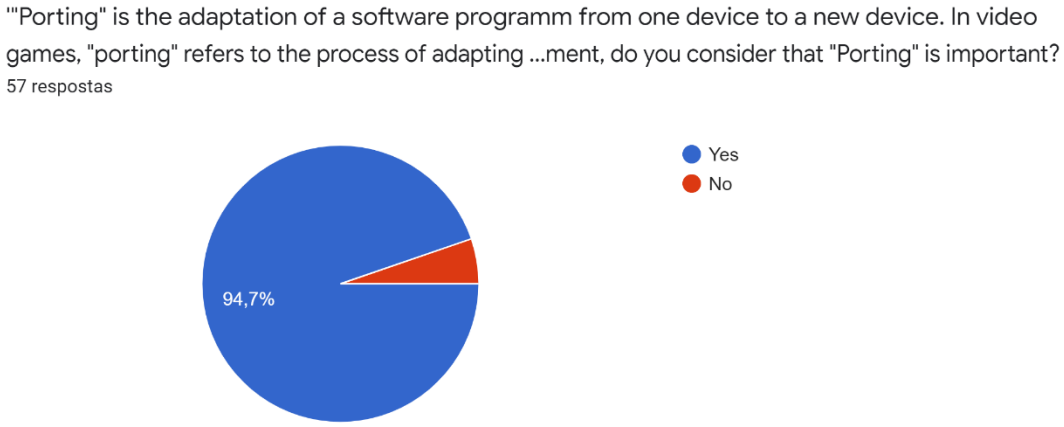
For the respondents that said considered themselves gamers, the same question about the platforms were asked. This was made separately to distinguish the answers of a gamer and a non-gamer.



For all the respondents, it was asked if they knew what *porting* was. With the “No” answer being more than half percent it could be an indication that most people have probably played or seen a *porting* without knowing that it was one.



After a brief explanation of the term "*porting",* almost all respondents considered it be important, even if they had previously said they did not know what *porting* was.



A quick overview of the current policies from game studios was presented and then it was asked the opinions about the users about porting. Despite the majority considering that the companies having works allocated to a porting specifically is a good move, because it allows the game to be played by more people, it was not unanimous.

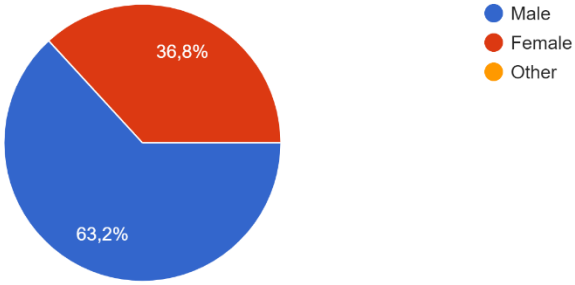
With the advance of the industry, more and more ports are being made, and usually a studio needs to allocate several workers to work on the porting...ou believe this is a good move from game studios?

57 respostas

To confirm if the answers were not influenced by the genre and they were not all from the same one, the user's genre was asked. Although males were the biggest part, as it was expected from the start, a good part was also female.

Are you male or female?

57 respostas



Methodology provided to the developers

A guide to porting Videogames

Context:

The process of porting videogames has become more and more recurrently in the gaming community, with good and bad ports being made all the time.

For my Master's Degree thesis, I've studied the porting process by joining a game studio and I have work in several portings, as well as studied what was done in previous portings.

My goal is to provide a methodology that will help during the development of portings by reducing the development time and doing something that can be made into a standard for the industry.

Respecting all the NDA's from the different involved companies, this methodology will be described in general terms in order to be able to help any developer that is working or will start working on a porting, no matter what game or for which platform they are developing. However, this project wants to help indie and new developers above all, as doing a porting for the first time can be overwhelming.
Some information provided is aimed towards Unity, as it was the only engine used during this project.

The methodology will be presented by stages. It's expected that you use the methodology and try applying it during a porting. After applying it and doing the port, you will be asked to answer some questions to rate the methodology.
(in case you are currently **NOT** doing a porting, but have experienced and made one before, you can answer and rate the methodology immediately after reading it, based on your knowledge and how much you consider this methodology would help)

The Setup

When porting for some specific platforms (like consoles):

- Register as a developer on platform's website/portal as soon as possible (recommended to register as a company, even if single person company, as the acceptance process might be faster that way)
- Order your Dev Kit early.

For all platforms:

- Download and install all necessary materials (software, SDK's, documentation, etc)
- Check platforms guidelines before starting development. Will save you some troubles along the way if you know beforehand what you can or can't do.
- Consult forums and FAQ's. Most of your issues should be there.

Understanding the essence of the game

When doing the porting of a game from a third party's game/ client, regardless of it being made after original game has released, or being developed concurrently with the original:

- Always playtest before starting development. Understand how the game feels, how the controls act, what feels or doesn't feel natural to the player.
- Porting and game development is about transmitting an experience to the player. If you don't know that experience, you won't be able to transmit it efficiently.

Make the game simply run

- Remove or deactivate features, libraries, modules or any thing platform-specific, even connections to databases and servers.
- Make the game be runnable and executable by the new platform. Don't focus on controls, save systems, extra features or any of that for now.
- On Unity, that will start by having to select the correct platform on the build settings. If your platform doesn't show, re-check your development environment and download materials.
- "Define declaratives" are a Unity feature that allows you, in the code, to select parts of the source code to only run on the specified platforms, or to not run. This, along with a few more code tweaks, allow you to switch platforms between Unity and the target platform's Dev Kit and test the project on both sides without changing the code.

Make game playable

- Start implementing controls. Go from small to big. Simply control a character or make some action in game. From there, rise into implementing the rest of the controls
- Keep in mind the possibility of different controllers when implementing, as the logic isn't always the same.
- Different settings (sensitivity, deadzone, etc) can require different values in different controllers to produce the same outcome.
- Follow the standardization in buttons assignments when possible. The more familiar and natural the controls are to the player, the easier will be for them to get into the game with a good experience. (Example: a shooter will probably use the Triggers/mouse buttons for aiming and firing.)

Replacing assets

Porting isn't only for programmers. It's a collaboration and it requires artists, just like normal game development. If the original game also had them:

- Remove icons, images, dialogue lines or any other place where the controls are displayed and change them into new ones that refer to the new platform buttons/keys.
- Respect the game's art style and direction when changing assets.
- The whole game's art can need extra adjusting due to performance, as will be described in a later stage.

Save systems

Some consoles have particular ways of dealing with saves, so try to follow all available documentation and samples when doing it for the first time, as it can be tricky to grasp. Try using the original game's save system as much as possible

- Use the original game's save system as much as possible. At the end of the save system, before the file it's stored into any folder, use the file on the new save system you will create.
- Respecting each platform's restrictions (when they exist) and their save system base, start by saving some simple variables by doing a key press or any simple action before using the save file itself
- Using development tools like logs and prints, confirm if the variable you want to save is indeed the same that is being written in the file.
- Exit the game, check the save folder and the save files content to verify if the saved information stayed after closing application.
- Launch the game again and try loading the data as intended. If it works, test this a few times in different places of the game, with more or less data until you are sure it's working
- After ensuring it's properly working, change the variable you were saving into the data from the original save's file, as mentioned in the first step.
- Not all formats are accepted by all platforms the same way, so by using the original files, you might need formatters, parsers, or any other way to convert it into an acceptable file.
- Change the "when" the game saves into what is supposed to be, not the key button as it was for testing.
- Don't overload the save system. If using auto saves, don't make the game save thousands of data constantly.

By doing the save system in small steps and starting with some random variables, it becomes much easier to troubleshoot if something doesn't work

Adding features and settings

This will differ drastically between portings, as each game and platform have their own features. When porting, keep in mind that it might not be possible to have all the features the original game had. Thoroughly test when implementing, especially online features due to security.

Some of these features or settings (like vibration strength, audio values, etc) are changeable by the player. Those are commonly called "persistent data" and can be stored in different ways:

- In the same file as the game's save data file
- In a second save file exclusively to this data (using the same save system methods)
- Using a unique save system for this data
- In Unity, with *PlayerPrefs.Save()* and related functions

Each has their own benefits and prejudices, but the second option proved to be the best solution overall.

Performance Optimization

Your porting should be basically complete when you reach the optimization point, as it's usually the last part. In a porting, remember that you are dependent on the platform's power, and that may impact the performance and/or quality of the game. However, the player should still get the best experience he can for those conditions. A game capped at 30 FPS is not a problem, a game having constant frames drops is the issue.

To improve performance, there are some tips you can use in your porting:

- Cut unnecessary assets in the game (e.g., some trees in the background, some birds flying, as long as it doesn't affect the outcome)
- If possible, since the assets are from your client, redo some assets to reduce details
- Lower the overall quality of existing assets (Unity provides settings that do it automatically, only needing to choose from a pre-configured option)
- Remove/lower visual effects, like shadows, anti-aliasing, raytracing. Anything that is post-processing will have, in most cases, a huge impact on performance, and a smaller improvement in visual quality.
- Use LOD-level of details to only render what is closer to the player with high detail.
- Balance the sacrifice you make in visual quality. Hurting the visuals too much won't create a good response from the player.

In terms of coding, some measures can also be taken:

- Apply normal programming "good practices"
- When working with numbers, don't use more types with more precision when not needed (using *floats* when you only need *ints*)
- Improving the math, as some operations are heavy and sometimes can be simplified.
- Use development tools, like Unity's Profiler, to determine which actions are taking a heavier toll on the performance.

Thank you for reading and using this methodology. You can access the questionnaire in this link: https://forms.gle/D5haKSM8ThbTBM2v9