# PoGER
A Bachelor's project

David Rodriguez Soares

## Abstract

Game development has evolved from an obscure activity only teams of people with specialized training could work on to an accessible art through the use of powerful tools and shared already-made code and assets, springing a healthy and successful independent game-making scene. Unfortunately, this is not the state of most fan-made game creation communities, in particular the one dedicated to the popular Pokémon franchise. It is plagued by a combination of issues, mainly the inadequacy of used tools, resulting in its apparent abandonment.

This paper presents PoGER, an software preservation and game engine recreation project aimed at proposing systemic solutions to these issues. I discuss the goals of this project, the reverse-engineering effort and its findings and design decisions taken. I also present a framework and tools for both porting old projects and facilitating future development, which could kickstart a new wave of enthusiasm in its field and hopefully give motivated developers an opportunity to create their own worlds and adventures.

## Contents

## 1 Introduction

Since its creation by japanese game designer Satoshi Tajiri in 1995 [1, 2], the Pokémon franchise (also known as *Pocket Monsters* in Japan) had a significant impact on culture around the world [3, 4] and on its own genre (the JRPG, for Japanese Role-Playing video Game) [5].

The Pokémon games, originally launched on the Nintendo Game Boy handheld, experienced immediate and global success, selling over 46 (or 31 if not counting *Pokémon Yellow*) million units on its first generation [6] and more than 368 million copies sold overall [7].

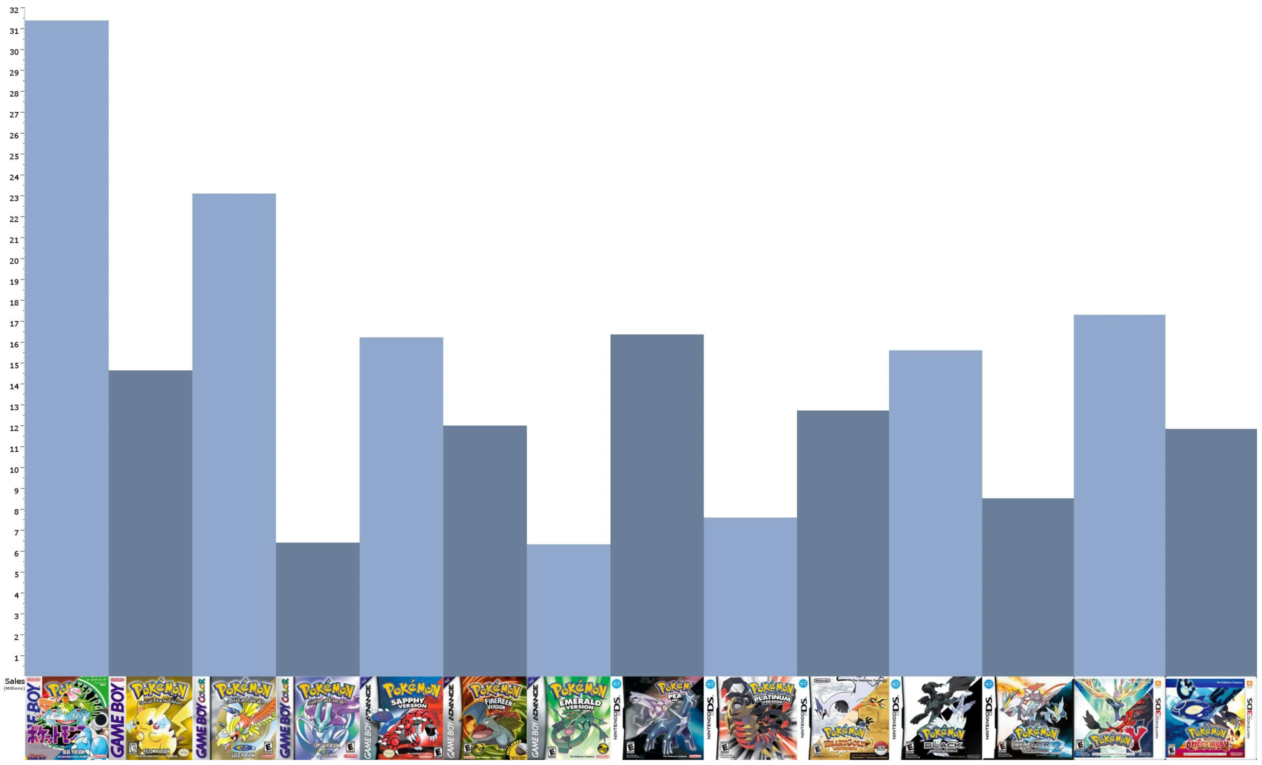

Figure 1: Unit sales of Pokémon mainline games, in millions [6, 8]

Since then, it influenced generations of people who created art [9], named animals and proteins [10, 11], wrote novels [12], did scientific research, etc. Simply search for `Pokémon` on an academic search engine like `Google Scholar` and you would be surprised of the amount of results and the diversity of represented fields !

With such a motivating and long-running game franchise, it was inevitable some fans would look into the possibility of making their own addition to the series.

### 1.1 The state of Pokémon fan-made games

As the name suggest, these games are created by fans of the franchise and are based on its core concepts and gameplay mechanics [13]. Over the last two decades, many game projects saw a release on the Internet and even a few can be found in physical media form [14, 15] !

But what does that development scene looks like ? What I discovered was a clustered universe of fan-made games. There are 3 main categories :

**ROM hacks** Build upon the code and assets of commercial products, typically GameBoy games [13], they can be played on common emulators (and on original hardware [16]), therefore available for most platforms. Unfortunately, programming for them is very technical, with limited functionality and questionable legality [17].

**Games made with RPG Maker XP** Build on top of the *Pokemon Essentials* project [13, 18], they are easier to program for and have less limitations than ROM hacks.

**Original games** These are original software, typically built for Windows or the web. Thanks to that, they can have great performance and novel game mechanics (eg. MMORPG style), but remain typically closed-source, and only a few projects exist [19, 20, 21].

## 1.2 Problems

A lot of people tried to make their own games by love for the franchise, but from my early research into Pokémon fan-made games, a few common issues can be identified.

Except for ROM hacks and the occasional web-based game, which run on emulators and web browsers respectively, software is **platform-specific** (typically made for the Windows operating system). This is not optimal, because it restricts the amount of devices they can be run on, thus their market. Combined with the fact that they are almost universally **closed-source**, it becomes an issue of **software conservation**.

Another common issue is the **high amount of technical knowledge** that anyone willing to work on them needs, even with tools like RPG Maker XP.

Due to the **legality challenges** and the **personal nature** of fan-games, but also the **lack of collaborative code source management tools** (like *git*), they are typically a **single-developer endeavor**.

The combination of previously mentioned issues is, in my opinion, a perfect storm that explains the lack of quality fan-made games in a time independent game development is on the rise : It kills most attempts before they are release-worthy and keeps the rest from ever becoming polished and rid of bugs as commercial games.

This happens because their developer finds no more time to invest or lose interest in them, and picking them up once that happened can be incredibly difficult, or even impossible.

This motivated me to find a solution that would addresses these issues, but I knew it was unrealistic to do so for every category of games.

## 1.3 Focusing on Pokemon Essentials

RPG Maker XP is an popular but aging RPG engine that offers convenient tools that allows anyone to make their own role-playing video game [22].

Unfortunately, programming for it is technical and it's only compatible with Windows (even running games on compatibility layers such as Wine on GNU/Linux isn't always functional). Also, performance is typically poor due to its aging engine [23].

**Pokemon Essential** is a project that aims at offering an implementation of the Pokemon game engine that runs on RPG Maker XP. Few projects using it were completed [13].

Even with the facilities introduced, programming for RPG Maker XP/Pokemon Essentials remains complex, and is typically done in very small teams of amateurs, not using collaborative source code management tools like *git*. This results in ambitious project requiring years before they can be released and are hard to maintain (see `Pokémon Uranium` for a case study).

I believe this is the reason most projects never see a release, and the ones that do are buggy.

## 1.4 Defining the project

The problem can be stated more simply : RPG Maker XP/Pokemon Essentials isn't adapted to creating a game in a modern way, and suffers from its age and design decisions, to the point it's basically abandoned and deprecated. Also, there doesn't seem to be a superior alternative tool for the purpose of creating a Pokemon game. It's clear to me that I should not try to build upon it, but rather bring it up somehow.

I won't detail every design decision here, but each one was made toward the goal of addressing the identified shortcomings of the studied software. Please read the *vision document* for more information.

The solution I came up with : implementing a similar engine that would be decoupled from RPG Maker XP and would run games as programs written in an interpreted domain-specific language I would create. A complementary piece of software would automate the port of games written for the original Pokemon Essentials engine, therefore making it more relevant than an alternative without any retro-compatibility path.

There is a name for this approach : game engine recreation. That's where the name for this project comes from, **PoGER** (***Po****kémon Essentials* ***G****ame* ***E****ngine* ***R****ecreation*).

**Requirements** : A solution would be an implementation of Pokémon Essentials that is :

- performant
- available (multi-platform)
- straightforward and simpler to develop for
- has online capabilities
- has engine code separate from game code
- provides documentation, sets standards
- fundamentally open-source to allow contributors to maintain projects.
- has a way to port games from the original Pokémon Essentials

## 1.5 Outline

This report is divided in six sections, including this introduction.

- Section 2 briefly explores the modern preoccupation of software perservation and discusses the application of its principles to the current project.
- Section 3 gives a detailed summary of identified software and projects of interest.

- Section 4 looks into the feasibility of asset and feature extraction from an existing Pokémon Essentials project. Then, it presents the implementation results of the extraction process and details the steps involved in a practical guide fashion.
- Section 5 presents the reverse-engineering results for a particularly complex category of extracted assets : Events. This is followed by a discussion on the necessity of a new representation for them and the design decisions behind the proposed domain-specific language. A formal definition is given, with examples and a Python implementation.
- Section 6 presents the early work on `PEN`, an interpreter that can run events. This would be the basis for an interpreter capable of running a full game. A portion is dedicated to PBS files and information that can be imported from them.
- Section 7 covers the reverse-engineering efforts aimed at interpreting RPG Maker XP's proprietary map and map asset format. Then, it presents a proof of concept implementation in Python.
- Section 8 explores the future of the project, the path towards an actual implementation of an interpreter capable of executing games, providing directions for future work.
- Section 9 closes this report with a summary and a critique of its contributions.

## 2 About software preservation

### 2.1 Maintaining the ability to run software

A modern preoccupation is the preservation of aging hardware and software, because as time passes, functional ancient hardware becomes increasingly scarce, and with it the means to run software made for it.

The problem of maintaining the possibility to run software despite the loss of access to its original platform pushed the development of **virtualization and emulation technologies**.

* Mendel Rosenblum: The Reincarnation of Virtual Machines. ACM Queue 2(5): 34-40 (2004)

* Bowen Alpern et al.: The Jikes Research Virtual Machine project: Building an open-source research community. IBM Systems Journal. 44(2): 399-418 (2005)

### 2.2 Preserving source code

Another related preoccupation is the long-term preservation of source code.

## 3 Research subjects

In order to define the scope of this project, I had to become familiar with the games and tools used to make them. In this section, I detail relevant information on RPG Maker XP/Pokémon Essentials

### 3.1 RPG Maker XP

According to its Wikipedia article and its official website, RPG Maker XP was developped by Enterbrain and released on July of 2004 for Microsoft Windows.

It allows its users to create their own role-playing game with powerful dedicated tools and its engine is based on the Ruby programming language, making complex behavior possible through the use of *scripts*.

This is only one instance among the long line of RPG Maker releases, some of which were released on other platforms, including consoles. Probably due to its early popularity, new projects continued to use this version way after it was superseded by newer ones.

#### 3.1.1 Ruby Game Scripting System

From this article :

"RGSS stands for Ruby Game Scripting System, a library that has been used in RPG Maker engines since RPG Maker XP. It's a Ruby library, and as such, all scripting is done in the Ruby language.

In all of its incarnations, RGSS has included objects and methods with which to handle graphics, audio, and data, including basic data structures for the RPG Maker engine.

The original version, RGSS, [...] was based on Ruby 1.81."

RGSS's latest version, RGSS3, was introduced with RPG Maker VX Ace in 2011-2012.

## 3.2 The Pokemon Essentials project

According to its wiki fandom page and another wiki article, Pokemon Essentials is "a collection of gameplay-altering original code designed for use in an RPG Maker XP game".

This means that it's a **RMXP project**, made to be the basis for a game. It was forked from Flameguru's Pokémon Starter Kit and developed by Peter O. (2007-2010) and Maruno (2011-today).

This project is up-to-date with the 5th generation of the mainline Pokemon games and is the basis of a fair proportion of Pokemon fan-made games. It was last updated in October of 2017, when it was hit by DMCA takedown notice.

### 3.2.1 Digging for information on Pokemon Essentials

Users have been able to produce content-adding patches, like support for newer generation creatures, items and abilities.

There is a v18 in the works (release date unclear). On Reddit, Maruno stated "hopefully it won't be long now" in June 2020.

It is notably hosted on archive.org but I downloaded it from this source because it has a Wikia dump and later file modification timestamps.

## 3.3 The Pokemon Uranium project

According to its Wikipedia page, its official website and its Fan-made Wiki website, Pokemon Uranium is a fan-made game based on Pokemon Essentials (version used is unclear) and has unique region, creatures, assets, plot and quests. It is often cited as one of the best Pokémon fan-made game [24, 25]. It was developped by a small team for about 9 years and released in August 2016.

- Game designer and developer : JV12345 (aka. ~JV~)
- Game developer and creative director : Nageki (aka. Involuntary Twitch)

They were hit by DMCA takedown notices shortly after release and stopped development in September 2016 [26].

### 3.3.1 Digging for information on Pokemon Uranium

The game was mostly complete when released, but its website and following game updates are community efforts. Fans were able to add elements to push it farther towards completion but there still are some hanging threads, particularly in the post game (after the main plot is over) [27].

Fans have been able to patch bugs and update the game for some time. I suppose there is a small group of people with access to the project's source code and files that have been maintaining it and its online services. Despite efforts to patch it, the game has a long list of unresolved bugs and game breaking/crashing situations.

I wasn't able to identify the used versions of RPG Maker XP or Pokemon Essentials, but I determined this wouldn't significantly impact on working on it.

GitHub user *acedogblast* launched a re-implementation project on the Godot game engine in early 2019. As of writing, he states that : "Only a limited portion of the game is playable currently."

Download links are hosted on Reddit. Current version is `1.2.4` and was released on October 29th 2018.

## 4 Extracting existing game assets

This section covers the complementary pieces of software that assist and automate the port of Pokemon Essentials. Games based on Pokemon Essentials should be able to be ported through the same process, with perhaps a few tweaks.

The information presented here was obtained through the official RPG Maker XP built-in documentation, user content found on the internet (forum posts, videos) and the author's reverse-engineering work.

Unless otherwise specified, the following findings pertains to the `Pokémon Essentials v17.2` project files and may not always apply to derivatives.

### 4.1 Technical findings

#### 4.1.1 Structure

Pokemon Essentials and its derivatives can be downloaded and their root directory explored without any specialized tool but perhaps an archive extractor utility. They all share this project structure :

| | |
|---|---|
| ○ Root | Contains compiled game files and libraries, plus a few useful programs, some probably made by Pokemon Essentials developers. |
| ↳ Audio | Contains the musical assets of the game, typically MIDI and OGG files. |
| ↳ (Data) | Contains the logic of the game and most of its data. |
| ↳ Fonts | Contain fonts used for displaying text. |
| ↳ Graphics | Contains the graphical assets of the game, typically tile sets, in a well organized folder structure. |
| ↳ (*PBS*) | Contains human-readable data to be compiled, like items, species, types, dialogue (and translations), etc. |

Note that in certain circumstances, directories shown between parenthesis may be omitted, because they exist in a compiled form. This is typical for game releases as it's a more efficient format for execution. This will be expanded upon thereafter.

#### 4.1.2 Data representation

The data that constitutes a PE project is diverse :

| **Category** | **Description** | **Storage** |
|---|---|---|
| Scripts | Logic of the game, classes, ect | `Scripts.rxdata` file |
| Objects | Files containing sets of class instances, grouped per file | `dat` and `rxdata` files |
| Maps | Contain map representation (incl. events), one per file | `MapXXX.rxdata` |
| Dialogue | Contain dialogue | TODO |

## 4.2 Research

In order to decouple PE from its original code base and RGSS, it became obvious that all useful data had to be extracted from the project.

Since `Audio`, `Graphics` and `Fonts` directories contain already-accessible data, they didn't require any extraction efforts.

See section *Data representation* for categories of data that were identified to need extraction.

## 4.3 Scripts

Findings :

- `Scripts.rxdata` is the second largest compiled file of the project, and contains over 120k lines of code.
- The script is unique : it is divided in named sections (that appear in RMXP's editor on the left hand side) that allow to manage its colossal line count.
- Like other compiled files, its content are almost impossible to read outside of RMXP.

I spent quite a lot of time working on a python script able to decrypt `rxdata` files, and `Scripts.rxdata` in particular, but didn't have much success, so I tried looking elsewhere.

**Trying something else** : After some more digging, I found a functional utility that was created for the purpose of editing/extracting the scripts from this file : Gemini Editor.

Here are links to a forum post about it and a github repository.

With it, I was able to extract Pokemon Essentials scripts (see '`PE_scripts`' directory).

It proved invaluable to be able to explore/search the whole script base using a more powerful code editor, saving lots of time from my attempts to create my own scripts to export data.

Note that extracting scripts isn't really useful : the objective is to get away from the RMXP-tied original codebase. It's only useful for the purpose of understanding how RMXP/RGSS/PE work and how features are implemented.

## 4.4 Maps and Events

Maps are essential to games made with RMXP, and a huge portion of the interface is dedicated to them. They contain textures that build the world and events that add elements of behavior to them.

I documented all I could find on maps the events they contain and more in `RMPX_doc`.

What I found :

- Trying to extract everything from compiled files would require a lot of effort.
- Creating scripts in RMXP that extract data is way easier.
- A solid understanding of the Ruby language and classes involved is mandatory.
- Class declarations can be found in the code and/or in the official documentation with a bit of digging.

Here is a diagram for classes involved in Map/Event/Tileset data extraction.

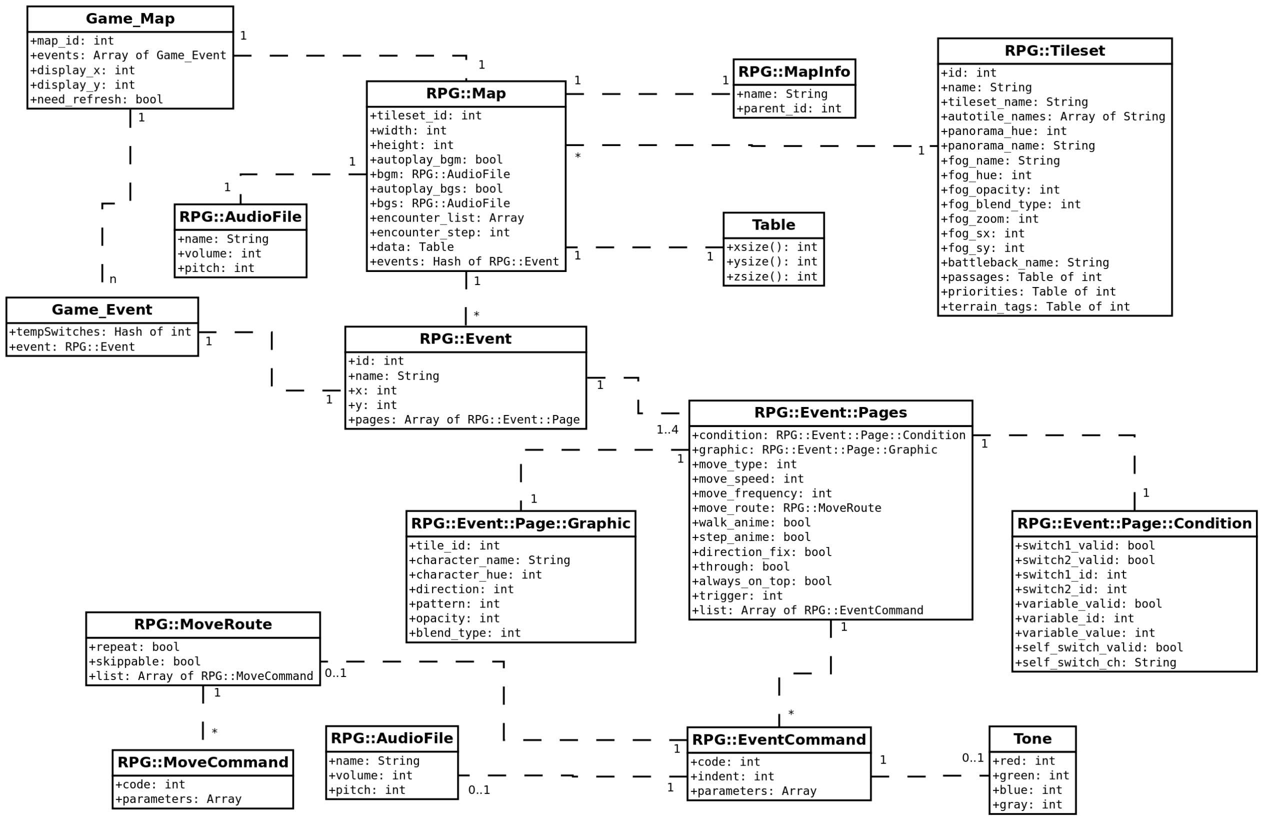

Figure 2: Simplified class map representation for Map/Event

*Semantic/Syntax* : Linked classes (with arity) display an **associative relationship**.

*Note* : There is no inheritance relationship between any two classes represented. Arities are logically deduced and may not be exact depending in proprietary implementation details. Class `RPG::AudioFile` was duplicated for ease of association routing.

### 4.4.1 Dialogue

English dialogue is hard-coded in events, therefore in order to propose alternative languages some trickery is needed.

More research is needed here.

## 4.5 Automated extraction process

Unfortunately, the first steps require the use of RPG Maker XP (version 1.02 tested, other may work also), and therefore Microsoft Windows (XP-10).

It may be possible to run RPG Maker XP on other operating systems using compatibility layers (such as Wine for GNU/Linux), but I didn't try it.

### 4.5.1 Opening a project with RPG Maker XP

First things first, a copy of the project to extract is needed. It should have the same structure as described in `section 1.2.1`. If so, it should be possible to open it and run it using RPG Maker XP.

Read more about obtaining a copy in `PoGER`.

The following information was obtained through a trial-and-error process when trying to open the Pokemon Essentials and a derivative game (Pokemon Uranium) in RPG Maker XP and run them in order to extract data from them. As such, they may contain erroneous statements or not apply/work for someone else trying to replicate the same process.

- You need the following files in the root directory :
  - `Game.exe` : The executable that launches the game.
  - `Game.ini` : A configuration file. If the `*.ini` in the project has an other name, *rename a copy* and **leave the original untouched**, because it seems RMXP needs a `Game.ini` file and game can specify an other name (hard coded in scripts)
  - `Game.rxproj` : Entry point for RMXP.
  - `RGSS102E.dll` : Contain RGSS dynamic libraries, probably necessary tu run the executable correctly.

  Renaming files may be needed for some project files.

- For projects with *compressed data* (no `Data` directory but there is one `*.rgssad` file at root), you need to use an extraction utility.

  I used RGSS Tool :

  ```
  python rgsstool.py -x -d "path\for\output\files" "path\of\rgssad\file"
  ```

  This worked with Python 3.x, google search it if you're not sure if you have a recent version of Python and how to launch it on your computer.

  Exemple for Pokemon Uranium when command line is at root :

  ```
  python rgsstool.py -x -d . Uranium.rgssad
  ```

- For projects missing the `*.rxproj` file, simply create a new empty file and write the following:

  ```
  RPGXP 1.02
  ```

  Save it with UTF-8 encoding and no "new line" character.

- Save files should be located in `C:\Users\<user_name>\Saved Games\<name_of_the_game>\`

- Tip : When running the project from RPG Maker XP, some files in the `Data` directory may be recompiled, which can cause errors. Keep a copy of the content of this directory in case you need to restore its content (you can exclude the `Scripts.rxdata` if you're editing the scripts, so you don't lose progress on this file).

### 4.5.2 RMXP script

The first step uses RPG Maker XP itself and a handmade script.

Steps :

1. Open the project in RPG Maker XP. Make sure the game can be launched (press `F12` or click on the arrow next to the note icon).
2. Open the script editor.
3. Open `Extraction.rb` and copy-paste its content to a new page in the script editor.
4. Next step is tricky : You need to call the script from somewhere.

   Recommended way : Run the game within RPG Maker XP and play until you can gain control of the character and save. Close the game. Then, create an event close to your character and add a script command with parameter `pbSaveAll()`.

   "Fast" (theoretical, untested) way : Find the initial event that runs when the game launches and add to it a script command with parameter `pbSaveAll()`.
5. Now, Launch the game again and trigger the event.

- Tip : Add "show text" commands before and after the script call to better visualize its execution (ex: "*Click to save data.*" and "*Done !*").

It works for Pokemon Essentials but may not work for its derivative works, especially if they changed data representations (unlikely) or if they contain unexpected data (likely).

After running the script, a few new folders should have appeared :

- `Maps` : Contain 2 files for each map (one for the map itself, the other for its events)
- `Tileset_data` : Contain one file per tileset.

There should also be a `Extraction.log` file, used to help debug issues with the script.

### 4.5.3 Event processing

Unfortunately, event porting is particularly complex for multiple reasons, including but not limited to :

- The extracted objects are representations of RPG Maker XP's event interface, therefore inherits its structure additionally to useful information.
- The bulk of the behavior is expressed as event commands, which are numerous and need individual attention.
- The goal is to extract semantics into a new domain-specific language.

For this reason, a Python program was written for the purpose of processing the JSON-formatted event files from the previous step into usable files.

## 4.6 Manual extraction

Unfortunately, I found some elements of RPG Maker XP to not be accessible using scripting or external utilities, thus making it impossible to automatically extract. Fortunately, they are scarce and it's easy to manually extract them from the UI.

### 4.6.1 Switch and Variable names

There are around 34 switches and a dozen variables used by PE. Commands that use them use their *id* to reference them, not their name.

While it is not strictly necessary to extract their corresponding names, it makes resulting commands way easier to read.

The most straightforward id-to-name conversion strategy is to create a Python dictionary with ids as key and name as values. This allows the conversion with a simple dictionary call.

You can find such an implementation in the `PE_variables_switches.py` file.

Procedure to extract names :

1. Create/Select an event, to display the "Event" window.
2. Create a new event command. In the "Event command" window, select "Conditional Branch". *There are other commands that could do the trick, I just chose this one.*
3. On the "Conditional Branch" window, select either "Switch" or "Variable" and click on the switch/variable selection box immediately to its right.
4. You should see a new window lists of items. Just select them one by one and copy-paste their name in the dictionary you're building (at the corresponding index of course).

## 5 Redefining Events

## 6 Interpreter

## 7 RPG Maker XP Maps

## 8 Roadmap for future works

## 9 Conclusion

# 10 Research

## 10.1 Methodology

Here are the high-level steps I took to study and experiment with RMXP, PE and PU :

1. Installing a working copy of RMXP
2. Opening the PE project

   This helped me understand how the project is structured, where the assets are located and how the scripting language works. Most of my experimentation started from it.

   I spent almost all of my time working on PE.
3. Opening the PU project

   This helped me understand how developers implemented new features and edited PE to suit their needs.

The objective was to understand how PE and its derivative works function and how features are implemented. I also needed to identify issues and come up with solutions to them, determine the specifics of the software I would implement in the next step of this project and document my findings.

## 10.2 Technical findings

### 10.2.1 Structure

PE and PU have a similar structure :

| | |
|---|---|
| ○ Root | Contains compiled game files and libraries, plus a few useful programs, some probably made by Pokemon Essentials developers. |
| ↳ Audio | Contains the musical assets of the game, typically MIDI and OGG files. |
| ↳ Data | Contains the logic of the game and most of its data. |
| ↳ Fonts | Contain fonts used for displaying text. |
| ↳ Graphics | Contains the graphical assets of the game, typically tile sets, in a well organized folder structure. |
| ↳ (*PBS*) | (Facultative) Contains human-readable data to be compiled, like items, species, types, dialogue (and translations), etc. |

### 10.2.2 Data representation

The data that constitutes a PE project is diverse :

| Category | Description | Storage |
|---|---|---|
| Scripts | Logic of the game, classes, ect | `Scripts.rxdata` file |
| Objects | Files containing sets of class instances, grouped per file | `dat` and `rxdata` files |
| Maps | Contain map representation (incl. events), one per file | `MapXXX.rxdata` |

Findings :

- Data representation isn't human-readable. I found `rxdata` and `dat` files are either marshalled data or structured binaries. These can contain RGSS code or serialized objects or lists of objects.
- All `rxdata` and most `dat` files begin by the same two bytes, which is probably the magic number/signature for RPG Maker XP files : $\backslash x04$ $\backslash x08$

  Note that usually, signatures are 4 bytes long.
- I haven't been able to find any built-in way to export data, instead resorting to external tools. I have tried a few tools found on github or forums.

  This means I will either have to reverse-engineer them and build my own tool (probably in its native language Ruby to take advantage of the module/class definitions I may be able to find in the code) or find a way to code my own extraction scripts within RMXP and execute them.

## 10.3 Non-technical findings

Here are some of my research findings that doesn't fit in the previous section.

### 10.3.1 About fanmade games

The projects I researched have a particular characteristic that I wasn't execting, but makes sense : They are personal side-projects from (mostly) individual developers.

With this in mind, it is no surprise that they are very long running projects that took years to release and displayed other properties I used to find odd :

- Closed source projects : Because they were very personal.
- Don't use code source management : Because there's no need, they are 1-2 developer personal projects that happened to be eventually released. Perhaps developers didn't consider the advantages of using one, or one wasn't available or familiar to them.
- Buggy projects (years after release) : With so few maintainers and such code complexity, patching bugs is a challenge.

As if the challenges facing such endeavors weren't great enough already, they were struck with the threat of legal repercussions on their authors, forcing these to abandon their creation. I can't help but having sincere sentiments of admiration for their work and sadness for the tragic destiny of their projects.

Fortunately, we can learn from the failures of the past how to better go forward. It is my firm belief that there are technical solutions to the challenges that brought these projects down.

## 10.4 Extracting data

Due to the documentation-heavy nature of this part of the project, I gave it its own separate manuscript : **RMXP doc**

Here is a high-level view of game data categories and their corresponding extraction methods :

| **Category** | **Extraction method** |
|---|---|
| Scripts | Re-implementation. |
| Events | Extraction script to JSON representation. |
| Maps | Extraction script to JSON representation. |
| Data | May need a tool (see **Extraction**). |
| Fonts | No need for extraction. |
| Graphics | No need for extraction. |
| Audio | No need for extraction. |
| PBS | No need for extraction (usually; extraction by script possible). |

The extraction method is detailed in the **Extraction** document.

## References

[1] The Editors of Encyclopaedia Britannica. *Pokémon*. URL: https://www.britannica.com/topic/Pokemon-electronic-game.

[2] Satoshi Tajiri. *Satoshi Tajiri: New Game Design*. Dec. 1995. ISBN: 978-4870258587.

[3] *How Pokemon Became a Pop Culture Sensation in America*. Feb. 25, 2016. URL: https://v1.escapistmagazine.com/articles/view/video-games/15498-How-Pokemon-Became-a-Lasting-Pop-Culture-Phenomenon-in-America.

[4] *The Pokémon Effect: How 20-Year Old Game Boy Cartridges Shaped a Generation*. Feb. 27, 2016. URL: https://www.huffpost.com/entry/the-pokemon-effect-how-20_b_9303926.

[5] *Why Pokemon is the most Influential RPG of All Time*. July 18, 2009. URL: https://venturebeat.com/2009/07/18/why-pokemon-is-the-most-influential-rpg-of-all-time-or-the-crisis-of-the-rpg/.

[6] VGSales. *Pokémon sales*. Sales figures compiled from Nintendo Japan. 2019. URL: https://vgsales.fandom.com/wiki/Pok%C3%A9mon.

[7] The Pokémon Company. *Pokémon in figures*. 2020. URL: https://corporate.pokemon.co.jp/en/aboutus/figures/.

[8] IanMazgelis. *Pokémon sales per game*. Illustration of sales figures from VGSales. 2016. URL: https://www.reddit.com/r/gaming/comments/4kjzq7/pokemon_sales_per_game/.

[9] *Artist Re-imagines Starter Pokemon in Amazing Ancient Mayan Art Style Paintings*. URL: https://grapee.jp/en/115487.

[10] Shigeru Sato, Yoshihiro Omori, et al. "Pikachurin, a dystroglycan ligand, is essential for photoreceptor ribbon synapse formation". In: (Nature Neuroscience 11 (923–931) 2008). URL: https://doi.org/10.1038/nn.2160.

[11] *Etymology: Names from Fictional Characters, section Video and Role-playing Games*. URL: https://www.curioustaxonomy.net/etym/fiction.html.

[12] *Jim Butcher chats about Pokemon, responsibility, and Changes*. URL: http://www.fantasyliterature.com/author-interviews/jim-butcher/.

[13] *Fan Made Pokemon Games List*. URL: https://www.pokemoncoders.com/fan-made-pokemon-games/.

[14] *Pokemon Ultra Violet*. URL: https://www.etsy.com/fr/listing/502285451/pokemon-ultra-violet-fan-made-hack-gba.

[15] *Obscure Pokemon Hacks ON REAL GBA CARTRIDGES!* URL: https://andisgamesrealm.wordpress.com/2013/11/15/obscure-pokemon-hacks-on-real-gba-cartridges/.

[16] *Play Homebrews on Your GBA*. URL: https://www.oreilly.com/library/view/retro-gaming-hacks/0596009178/ch04s16.html.

[17] *Pokémon ROM hack stopped by Nintendo four days before launch*. URL: https://arstechnica.com/gaming/2016/12/nintendo-sends-cease-and-desist-notice-to-pokemon-rom-hacker/.

[18] *Essentials Docs Wiki*. URL: https://essentialsdocs.fandom.com/wiki/Essentials_Docs_Wiki.

[19] *Pokémon Planet*. URL: https://pokemon-planet.com/.

[20] *Pokémon Legends*. URL: https://www.pokemonlegends.com/.

[21] *Pokemon Fan Games List*. URL: https://www.gbahacks.com/p/pokemon-pc-games.html.

[22] *RPG Maker XP: Make your own game !* URL: https://www.rpgmakerweb.com/products/rpg-maker-xp.

[23] *Can we get a solution for Essentials / RPG Maker running poorly in W10?* URL: https://reliccastle.com/threads/1521/.

[24] *Best Pokémon Fan Games 2020*. URL: https://thetecsite.com/best-pokemon-fan-games.

[25] *The 10 Best Pokémon Fan Games Ever Made*. URL: https://gaminggorilla.com/best-pokemon-fan-games/.

[26] *Pokémon Uranium Creators Pull Game After 1.5 Million Downloads*. URL: https://www.kotaku.com.au/2016/08/pokmon-uranium-creators-pull-game-after-15-million-downloads/.

[27] *Unimplemented/Unfinished Sidequests*. URL: https://pokemon-uranium.fandom.com/wiki/Sidequests.

[28] .