PoGER : a game engine recreation and software preservation project

A Bachelor's project

David Rodriguez Soares

Centre Universitaire d'Informatique,
University of Geneva, Switzerland

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 and software preservation challenges.

This paper presents PoGER, an software preservation and game engine recreation project aimed at proposing systemic solutions to these issues. We discuss the goals of this project, the reverse-engineering effort, its findings and design decisions taken. We 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

Acknowledgments

I would like to express my sincere thanks to the team that followed my work with interest for their guidance, continued support and encouragement: Prof. Didier Buchs, Dr. Dimitri Racordon, Dr. Damien Morard and Mr. Aurélien Coet. I couldn't have made as much progress without you.

Additionally, I would also like to thank Clara, Sofia, Juan and my other relatives for allowing me time away from them to dedicate to this project and produce a report in time. My mission will be to replace the many occasions I couldn't attend with you.

Finally, to my loving girlfriend I treasure, Tiffany. Your cheers kept me going at the hardest times and your unyielding devotion was profoundly appreciated. May the enchanted butterflies of my gratitude fly to thee.

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].

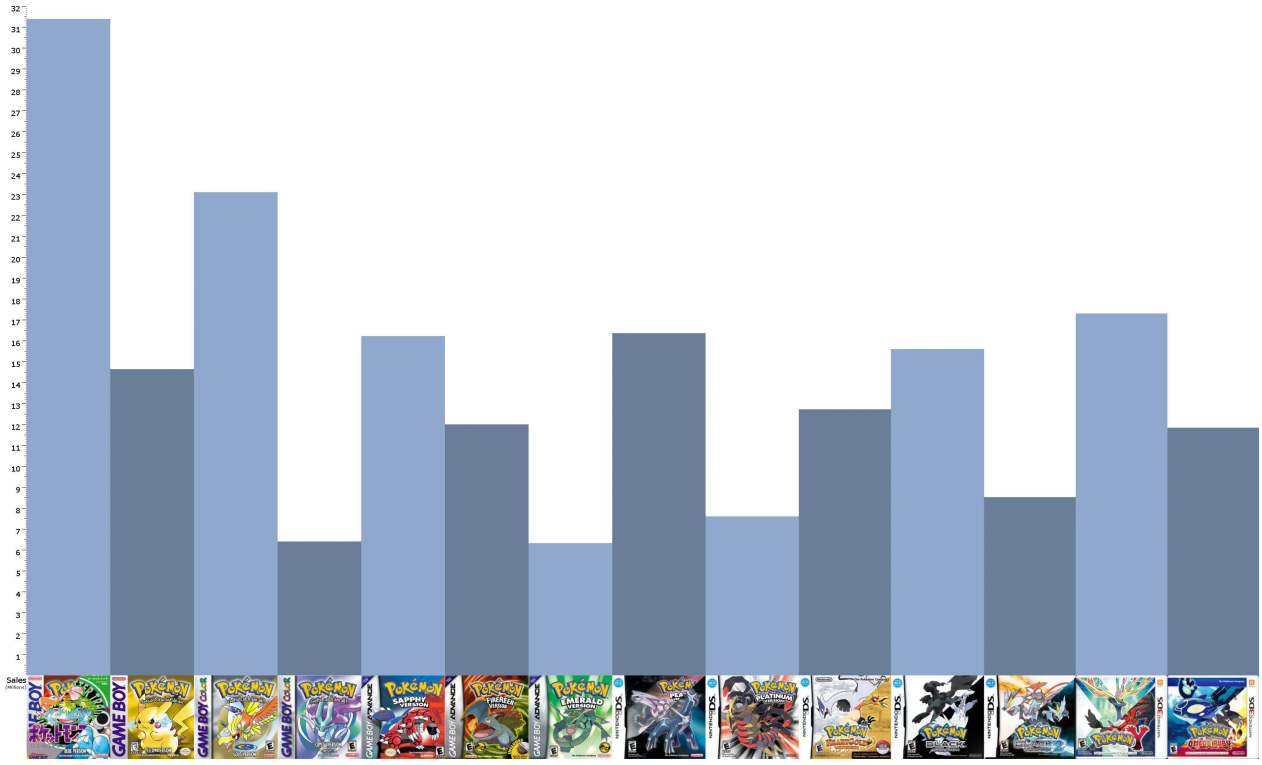

Figure 1: Unit sales of Pokémon mainline games, in millions [6, 8]

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 we discovered is 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 was an indication there was a need for a solution that would address these issues, but we knew it was unrealistic to do so for every category of games.

1.3 Focusing on Pokemon Essentials

See Section 3 for more detailed information.

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 natively compatible with Windows. Note that compatibility layers such as Wine on GNU/Linux are able to launch games made with it such as `Pokémon Uranium` [23]. Also, performance is typically poor due to its aging engine [24].

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 `Section 3.3` for a case study).

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

1.4 Defining the project

Problem definition : RPG Maker XP/Pokemon Essentials isn't adapted to creating a game in a modern way, and suffers from its age and design decisions, and unfortunately there doesn't seem to be a superior alternative tool for the purpose of creating a Pokemon game. Also, games based on these tools present significant software preservation challenges.

Further research made it clear that we should not try to build upon it, but rather bring it up somehow.

The solution we came up with : implementing an engine functionally similar to Pokemon Essentials that would be decoupled from RPG Maker XP and would run games as programs written in an interpreted domain-specific language we would create. A complementary piece of software would automate the port of games written for the original Pokemon Essentials engine, therefore satisfying our goals of software preservation and 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
- is available (os independent)
- is straightforward and simpler to develop for
- has online capabilities
- has engine code separate from game code
- provides documentation, sets standards
- is 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 nine sections (*plus reference list*), 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.

1.6 About unincluded assets

For practical reasons, a few elements couldn't be included in this document. These are mainly the software produced for the purpose of demonstration, proofs of concept, etc. Any referenced document/directory/file that isn't located in an `Appendix` shall be available in the following GitLab repository :

`https://gitlab.unige.ch/David.Rodriguez.1/poger`

2 About software preservation

Digital preservation describe the efforts aimed at insuring access to digitally stored information and the strategies, technologies and actions involved in resolving the challenges said effort comes across [25, 26]. This is a broad endeavor that spawned multiple projects with different scopes.

A common scope for digital preservation is *web archival*, which aims at insuring access to web pages long after their content is changed or rendered inaccessible [27]. A well known service implementing web archival is the `Wayback Machine` [1], which enable anyone to visit a web page at a previously stored state given its address.

Software preservation is mainly focused on insuring access to means of *running software* and *preserving source code*.

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**.

From `David S. H. Rosenthal's article` [28]:

"**Virtualization** is a technique for implementing a VM on a host computer. It depends on the host computer's instruction set being the *same* as [...] the VM's instruction set, and having certain specific hardware properties that enable virtualization. Almost all instructions executed by the VM are directly executed by the host computer's CPU; only a few instructions are intercepted, using the specific properties, and performed by host software. Virtualized systems run *unmodified* software binaries.

[...]

Emulation is a technique for implementing a VM on a host computer whose instruction set is *different* from the host computer's. None of the instructions executed by the VM are directly executed by the host computer's CPU; all are translated by host computer software from the VM's instruction set to the host computer's instruction set by host software before being executed. Emulated systems run *unmodified* software binaries."

Both these technologies have a lot in common : software compatibility (the capacity of running software by matching compatible hardware interface), overhead (the additional computation necessary for the hardware abstraction) and isolation (the capacity of running software without impact on the host system) [29]. Most importantly, by abstracting the hardware expected from the software for it to be executed, it allows it to be executed on any hardware that implement these technologies.

In some instances, yet another strategy may be used : **Translation**. This is a technique to convert software from an archaic language to a modern one, therefore insuring its preservation in another form, but typically requiring its source code [30].

Each approach has its benefits, drawbacks, scopes and challenges, therefore there isn't a universal approach to this aspect of software preservation.

[1] https://web.archive.org/

2.2 Preserving source code

The long-term preservation of source code is motivated by the observation that software is omnipresent in our modern life, scientific research and enables access to digital information. As such, it can be conceived as a common good, another storage medium for *knowledge*, therefore worth conservation [31]. Unfortunately, it presents unique challenges, including but not limited to :

Availability : Software in *executable form* is unreadable by humans, and its *source code* counterpart, which contains actual *knowledge*, may not be available [31]. The rise of Free/Open Source Software (FOSS), founded on principles of source code accessibility, is therefore a welcome phenomenon [32].

Artifacts : Software development is a process that produces artifacts by its developers and users: documentation, change logs, guides, FAQs, issues discussion threads, etc. Such artifacts are of crucial importance and offer context about its software [31, 33]. Source code preservation projects may decide to include this knowledge.

Mutability : As a digital object, source code is bound to evolve over time, to the point of generating incompatibilities between different versions. Version management technologies are needed to capture its history and organize it in a comprehensive way [31, 34].

Source code has interesting particularities : It's hard/expensive to produce and compact in size. It follows that simply storing valuable amounts of it isn't technically or financially challenging, rather the capacity of presenting it in a useful way is. Various projects aiming at long-term software source code preservation are focused on developing archiving and cataloging technologies to offer effective access to their stored knowledge (not an exhaustive list) :

- Software Heritage : Focuses on source code, indiscriminately of origin [31, 35].
- Zenodo : Focuses on data and software involved in scientific research. They use GitHub/GitLab integration for importing source code [36].
- GitHub Archive program : A project with multiple well-known partners, with "very-long-term" archival objectives and a focus on storage technologies lifespan and redundancy [37]

2.3 Link with PoGER

Some of the goals of this project are to present a candidate framework for recovering source code, give it an alternative representation, insure the ability to execute it independently to its original platform and do so in a open-source-oriented fashion, which would enable software and source code preservation much easier.

This approach admittedly deviates from the norm : not fitting the description of neither virtualization, emulation nor translation, rather being some kind of *transformation*. It was chosen because the commonly found practices were either not applicable or too ambitious for the scope this project.

3 Research subjects

In order to define the scope of this project, becoming familiar with the games and tools used to make them was a crucial step. In this section, we detail relevant information on RPG Maker XP/Pokémon Essentials and present a case study for a particular game : `Pokémon Uranium`

3.1 RPG Maker XP

According to its official website[1] and its Steam listing[2], 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 an article on this subject[3] :

"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 [38, 39].

3.2 The Pokemon Essentials project

According to fan-made articles[4,5], 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 **RPG Maker XP 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 5^{th} 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.

[1] https://www.rpgmakerweb.com/products/programs/rpg-maker-xp
[2] https://store.steampowered.com/app/235900/RPG_Maker_XP/
[3] https://rmvxace.fandom.com/wiki/RGSS
[4] https://essentialsdocs.fandom.com/wiki/Essentials_Docs_Wiki
[5] https://pokemon-fan-game.fandom.com/wiki/Pok%C3%A9mon_Essentials

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[6] (release date unclear). On Reddit[7], Maruno stated "hopefully it won't be long now" in June 2020.

`Pokemon Essentials v17.2` is notably hosted on `archive.org`[8].

3.3 The Pokemon Uranium project

According to its Wikipedia page[9], its official website[10] and its fan-made Wiki website[11], 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 [40, 41]. 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 [42].

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) [43].

Fans have been able to patch bugs and update the game for some time[12]. It's indicative 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 these efforts, the game has a long list of unresolved bugs and game breaking/crashing situations[13].

We weren't able to identify what versions of RPG Maker XP or Pokemon Essentials were used by the original developers, but it was determined to be of little importance, because any porting project would imply working with the given version regardless.

GitHub user *acedogblast* launched a re-implementation project[14] 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[15]. Current version is `1.2.4` and was released on October 29$^{\text{th}}$ 2018.

[6] https://www.reddit.com/r/PokemonRMXP/comments/ckeaov/pok%C3%A9mon_essentials_v18_progress_report/
[7] https://www.reddit.com/r/PokemonRMXP/comments/hb6i6m/should_i_wait_for_essentials_v18/
[8] https://archive.org/details/PokmonEssentialsV17.220171015
[9] https://en.wikipedia.org/wiki/Pok%C3%A9mon_Uranium
[10] http://pokemonuranium.org/
[11] https://pokemon-uranium.fandom.com/wiki/Main_Page
[12] https://pokemon-uranium.fandom.com/wiki/Official_Patch_Notes
[13] https://pokemon-uranium.fandom.com/wiki/Bugs_and_Errors
[14] https://github.com/acedogblast/Project-Uranium-Godot
[15] https://www.reddit.com/r/pokemonuranium/comments/a0cw0i/download_links/

4 Extracting existing game assets

In order to proceed with research about how to best preserve games made with RpgMaker XP/Pokémon Essentials, access to every game asset had to be insured. This is not trivial, as some exist in a format only its original engine knows how to handle.

This section covers the complementary pieces of software that assist and automate the extraction process of assets for Pokemon Essentials. Said process should be applicable to other games based on Pokemon Essentials, 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, formats include TTF and OTF.
↳ Graphics	Contains the graphical assets of the game, typically tile sets, using mainly PNG format, 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

While simpler assets (audio, fonts, graphics) exist in a *standardized media format* that can be opened by dedicated media reading software, the contents of the `Data` directory contain more complex and specific objects. These are *compiled* by the RPG Maker XP engine, stored in a *non-standard format* which makes information extraction more complicated. Here is a list of identified data categories :

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 (including events), one per file	`MapXXX.rxdata`
Dialogue	Contain dialogue	No standard

Extraction efforts are focused on these data categories.

4.2 Scripts

Most of the implementation work of the Pokémon Essentials project resides in scripts that makes the game behave and look like a mainline Pokémon game. It contains vast amount of Ruby code, a lot of which is has no comments, but at least it seems its developers tried to keep names clear enough and limit code complexity for it to be legible.

Findings :

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

Quite a lot of time was spent working on a python script able to decrypt `rxdata` files, and `Scripts.rxdata` in particular, with little success, so we tried looking elsewhere.

Trying something else : After some more digging, we found a functional utility that was created for the purpose of editing/extracting the scripts from this file : Gemini Editor [45, 44].

With it, we were able to extract Pokemon Essentials scripts. It proved invaluable to be able to explore/search the whole code base using more powerful editors and utilities (Visual Studio Code, Atom, grep, etc), saving lots of time when creating the extraction script.

Note that extracting the game scripts isn't really necessary : the objective is to get away from the RPG Maker XP-tied original code base, therefore these assets are only used for reference. They are useful for the purpose of understanding how RPG Maker XP, RGSS and Pokémon Essentials work and how features are implemented. This knowledge was then used to craft the extraction script and other tools like the interpreter.

4.3 Maps and Events

Maps are essential to games made with RPG Maker XP, 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. Events are used for anything that can move or interact with the player : Doors, NPCs, items, etc. You can find more detailed information in sections 5 and 7.

Findings :

- Trying to extract everything from compiled files would require a lot of effort. Creating scripts in RPG Maker XP that extract data is way easier.
- Tilesets and autotiles (graphical assets found in dedicated subdirectories of `Graphics`) contain the textures used by maps, maps only contain an array of texture references.
- Class declarations can be found in the code and/or in the official documentation included in RPG Maker XP, because most classes represented in the following diagram are standard RGSS classes.

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

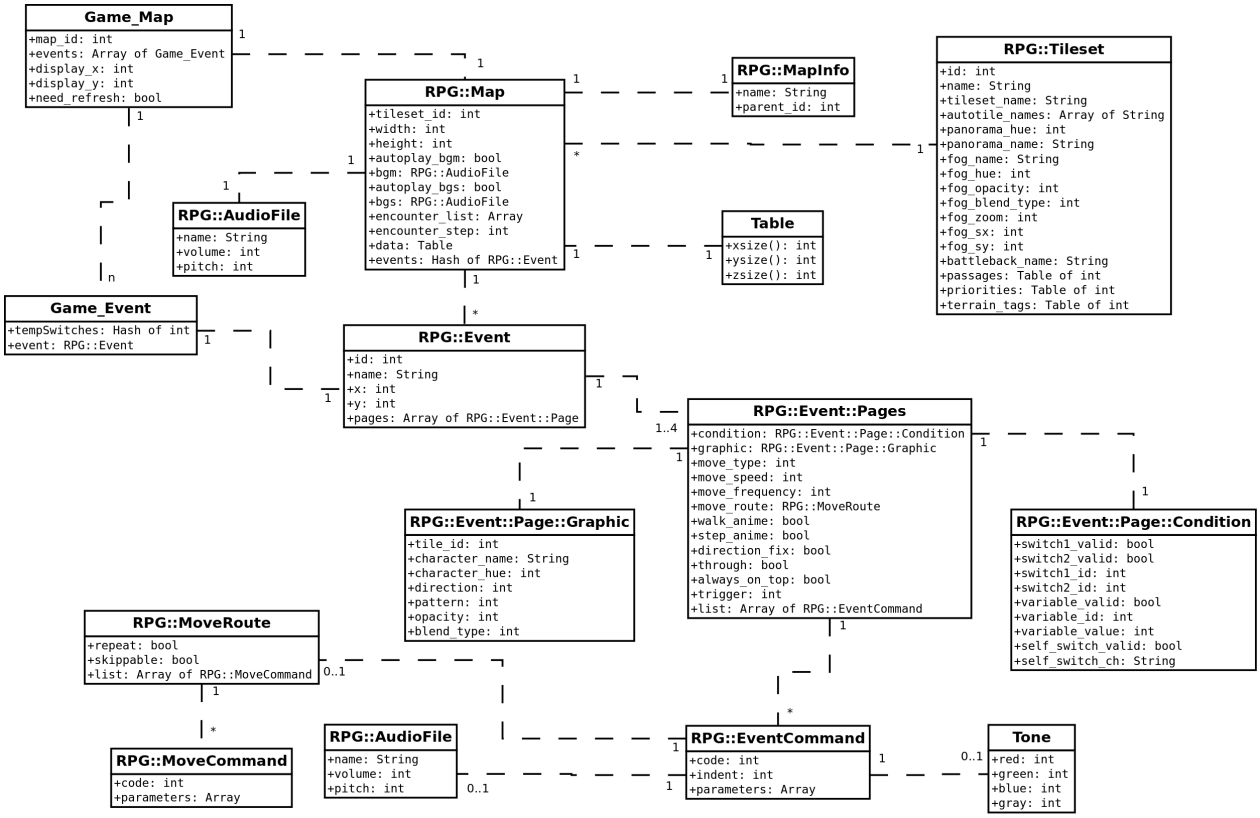

Figure 2: Simplified class map representation for Map/Event

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.

Fortunately, only a subset of the information contained in this diagram is absolutely necessary and effectively extracted as map and event objects. Some can be abstracted or simply aren't used by Pokémon Essentials.

Dialogue

English dialogue is hard-coded in events, therefore in order to propose alternative languages, derived games must implement their own dialogue translation system. More research is needed on that front.

4.4 Data extraction process overwiew

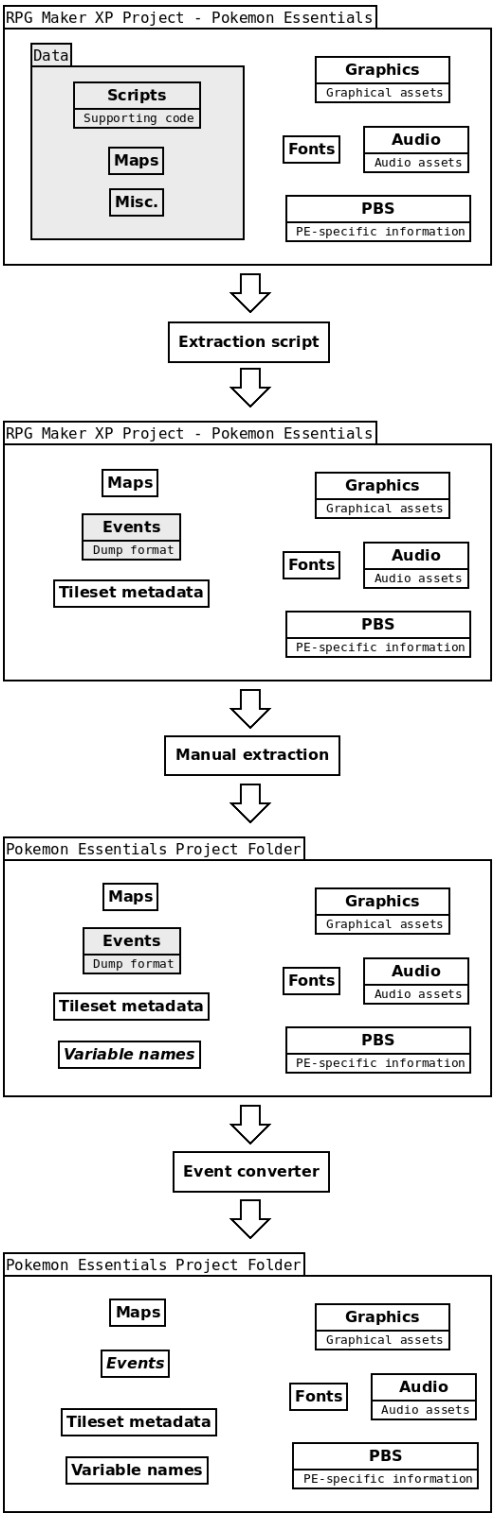

Figure 3: Extraction workflow

Pokemon Essentials, and other RPG Maker XP projects, have a clearly defined and standardized structure. Assets and data that make the content and logic of the game come in various formats, but for PoGER's purpose there is one important distinction :

- Readily accessible data : Represented with white background, they exist in a standard format that can be interpreted/read without using RPG Maker XP.
- Obfuscated data : Represented with gray background, they are typically compiled and need the RPG Maker XP engine to interpret them.

PoGER's extraction effort aims to retrieve *obfuscated data* and give it a standardized representation for use outside RPG Maker XP.

Extraction script

This script needs to be integrated to the game and executed. It does most of the heavy lifting, extracting **Maps**, **Events** (though only dumps them for further processing) and **Tileset metadata**.

Manual extraction

Unfortunately, no other way of retrieving the names associated with in-game variables and switches was found. They are needed because working with variable names is easier than with integer IDs.

Event converter

Events are surprisingly complex elements in RPG Maker XP, and the need for a new representation drove the development of an utility dedicated to that process.

Events are converted to a **domain-specific language** and packaged into individual easily editable files.

Final result

At the end of the extraction process, all assets and data used by the game exist in a RPG Maker XP-independent form.

This means that it's now possible for a third-party interpreter to run the game just like it ran on RPG Maker XP's!

4.5 Automated extraction process

Unfortunately, the first steps require the use of RPG Maker XP (version 1.02 tested, other should work also), and therefore Microsoft Windows (XP-10). It may be possible to run RPG Maker XP on other operating systems using a compatibility layer (such as Wine for GNU/Linux).

4.5.1 Opening a project with RPG Maker XP

Requirements : A copy of the project to extract. It should have the same structure as described in `section 4.1.1`. Also, as previously mentioned, you need RPG Maker XP.

The following information was obtained through a trial-and-error process when trying to open Pokemon Essentials and a derivative game (Pokémon Uranium) in RPG Maker XP and execute 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.

The following points should help you successfully opening a RPG Maker XP/Pokémon Essentials project :

- 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 RPG Maker XP needs a `Game.ini` file and game can specify an other name (hard coded in scripts)
  - `Game.rxproj` : Entry point for RPG Maker XP.
  - `RGSS102E.dll` : Contain RGSS dynamic libraries, necessary tu run the executable correctly.
- For projects with *compressed data* (no `Data` directory but there is one `*.rgssad` file at root), you need to use an extraction utility.

  `RGSS Tool`[16] was used :

  ```
  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 `Game.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.

[16] https://gitlab.com/rgss/rgsstool

Tips :

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

The role of the handcrafted script is to dump information accessible at runtime, some of which will need further processing to give it a useful representation.

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`[17] 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 during execution.

   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 !*"). Also, you should be able to see the name of the window changing rapidly : it shows progress of the extraction process.

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 at the `root` :

- `Maps` : Contain 2 files for each map (one for the map itself, the other for its events) and mirror the tree structure of the original project.
- `Tileset_data` : Contain one file per tileset/autotile, storing metadata.

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

[17] gitlab repository/master/RMXP_research/Extraction%20script/Extraction.rb

4.6 Manual extraction

Unfortunately, we 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 easy to manually obtain from the UI.

4.6.1 Switch and Variable names

There are around 34 switches and a dozen variables used by Pokémon Essentials. They are used to store long-term information that are typically not specific to a particular event. Commands using them use their *id* as reference, 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 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, we 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).

4.7 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.

5 Redefining Events

5.1 How RPG Maker XP stores data

A crucial first step in any reverse-engineering effort in data extraction is to understand used data structures. As RPG Maker XP games run on a *Ruby interpreter*, every element we encounter is either of a *primitive type* or an *object* (class instance).

Ruby primitive types :

- Arrays
- Hashes
- Boolean
- Symbols
- Numbers
- Strings

For classes associated with maps and events, most of which are part of the RPG Maker XP library (other are defined in Pokémon Essentials scripts), see `Figure 2`.

5.2 Events

An event, or more precisely a *map event*, is a way to introduce elements with behavior, therefore bringing flexibility and dynamism into the game world. Events have two representations :

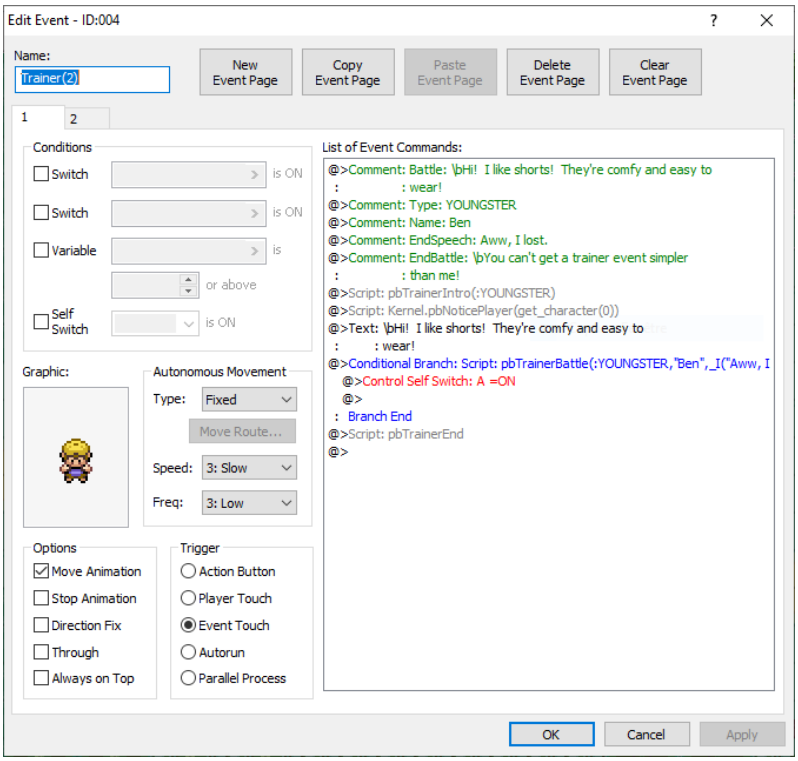

Figure 4: How an event looks like using RPG Maker XP's GUI

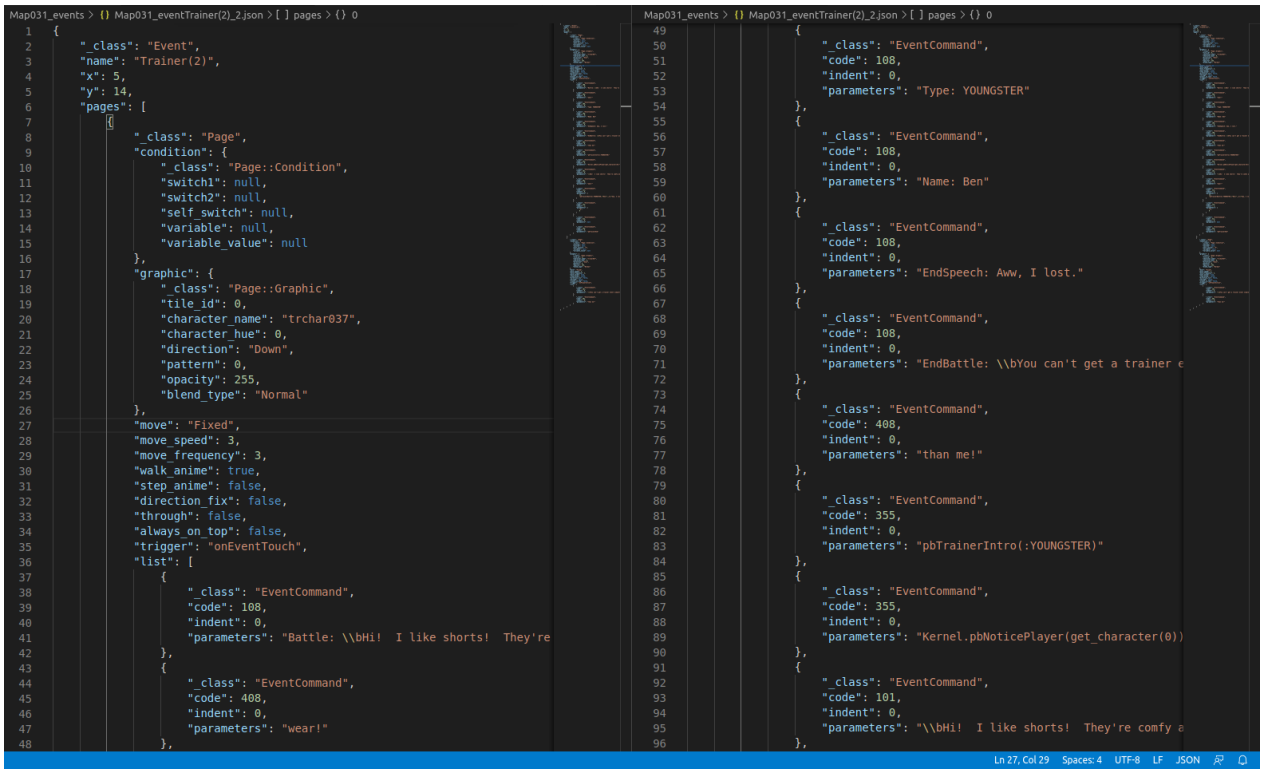

Figure 5: Its data class instance counterpart `RPG::Event`

Events are highly complex objects, providing game makers with ample amounts of customization and tweaks to create any dynamic map element they may need. The following sections will explore functionalities, with a focus on the ones used by Pokémon Essentials.

5.2.1 Basic functionalities

These are the easiest and most straightforward behavior to implement into an event :

- Giving an element a *sprite* (texture) : This is useful for objects capable of movement, NPCs, etc.
- *Movement* : Select how the element moves with presets (speed, frequency, pattern, etc).
- *Event commands* : Select the trigger for behavior and what the element does when triggered (movement, dialogue, etc) within the extensive command list.

5.2.2 Advanced functionalities

These require an understanding of conditional execution and scripting :

- *Conditional execution* : branching instructions based on the value of : global variables, global switches, self switches, script return, etc.
- *Pages* : Allow to give an element different behavior depending on conditions.
- *Move routes* : Define a sequence of movement commands to be executed.

- *Script calls* : Call a script to be executed for more complex behavior, launching mini-games, retrieving data, etc.

Script calls are particularly powerful because it allows arbitrary code execution. In simple terms, it enables events to do almost anything, providing it can be done through RGSS scripts.

As seen in `Figure 2`, every event can have up to 4 pages, and each page can have an arbitrary number of `EventCommands`. Once the event is triggered, the appropriate page's command list will be executed.

The page decision goes as follows : Each page's preconditions are evaluated, and the latest page with satisfied preconditions is executed. We found that a tacit rule is that the first page typically has no precondition, effectively making it the "default" behavior and avoiding game crashes caused by an inability to trigger events successfully.

5.3 Commands

Commands are a mechanism, through which most of an `RPG::Event`'s behavior is defined. Although they are very similar in structure and use, a distinction is made between `RPG::EventCommand` and `RPG::MoveCommand`.

EventCommands are the representation of elements present in the "List of Event Commands" in the GUI. They are the building block of event's behavior.

MoveCommands are the representation of an individual movement the event is capable of, typically found in sequences `RPG::MoveRoute` associated with a dedicated *EventCommand*.

They both have, at least :

- A *code* : An integer that uniquely identifies the particular command.
- *Parameters* : Depend on the particular command, can be empty, a variable, an object, or a list of objects.

Additionally, *EventCommands* have an *indent* integer value, tied to the layout visible in the "List of Event Commands" in the GUI.

5.3.1 Methodology

In order to successfully *extract semantic from events*, it was decided that *documenting* every command used in Pokemon Essentials and finding an appropriate (human-readable) *representation* was the way forward.

The objective is to formalize a **DSL** (Domain Specific Language) into which events will be translated to, which exhibit desirable properties.

For the sake of brevity, a lot of content was moved to **Appendix A**, mainly detailed information about each command used by Pokémon Essential events.

5.4 Command Representation

The representation chosen is a result of careful consideration of its future usage requirements, including but not limited to :

Readability : It is destined to be read and written by humans, therefore it should be as straightforward and non-cryptic as possible.

Brevity : In the interest of anyone (human or software) reading/writing it, the *less is more* approach is to be applied : instructions should not be longer than what is necessary.

Unambiguity : As any formal language, its use and syntax should be unambiguous.

Simplicity : Limiting the amount of available instructions by combining related ones is good practice.

Expandability : There should be room left for additional behavior to be implemented.

5.4.1 Particular decisions

Python syntax style : Reduces explicit syntax (like semicolons and curly braces), therefore reducing syntax error opportunities. Takes advantage of "implicit" syntax by making indentation itself meaningful.

Case insensitive : Simplification allowing any program to simply make everything lowercase when reading an *Event*, and game makers to use any casing style they prefer. This also makes it harder to have variable/switch name collisions by forcing users to explicitly name their variables. An exception is made for double quoted strings, which typically contain text to be displayed and should be excluded (its format should be preserved).

Switches, self-switches and variables : Should be all represented as **symbols**.

Proposed representation : `":s" [String]` *(string beginning with colon)*

Let `s` be the string representation (name) of the Switch/Self-switch/Variable. `s` of length 1 is to be reserved to self-switches.

Ideas

- Timers could be implemented as integers : Let `:PlayTime` be a read-olny integer variable that counts the seconds of play time (an `epoch`[18] of sorts).

  Then, setting a timer for x seconds could be as simple as storing (`:PlayTime` $+ x$) in a variable and testing it later against the current value of `:PlayTime` !

- Commands have parameters (see examples) :
  - No parameter : command line must contain the command keyword only.
  - 1 parameter : command line must contain the command keyword, plus the expected parameter `parameter_name = parameter_value` (parameter name recommended but not mandatory; parameter may be facultative)

[18] https://en.wikipedia.org/wiki/Epoch_(computing)

– n>1 parameters : command line must contain the command keyword, and the expected parameters `parameter_name = parameter_value` as a comma-separated list (no brackets; parameter name mandatory; parameter may be facultative).

– Note on *facultative* parameters : marked with a `*`.

– Strictly equivalent : '`:ON`'≡'`True`', '`:OFF`'≡'`False`', '`is`'≡'`==`' (when placed where a comparator would be).

5.5 Proposed command representation

In an attempt to formalize a Domain Specific Language (DSL) responding to previously stated requirements, a command list and formal grammar were included in **Appendix B** and **Appendix C** respectively. We named it `PEN` (Pokémon Essentials Next). This is an internal name and a better "public name" should be proposed. Events written in the `PEN` language and following the proposed event format are qualified as "PEN compliant".

5.5.1 Configuration variables

These configuration variables are inherited from RPG Maker XP's events. They should be mostly contained at the root of the event, but can be overridden by the pages.

CONFIG_VAR	**type**	**Default**	**Description**.
`name`	`String`	N/A	Identifies the event.
`xy`	`[int, int]`	N/A*	Position of the event.
`graphic`	`String/int`	None	Texture of the event.
`pattern`	`int`	0	Column index for the sub-texture from `graphic` to use.
`opacity`	`int`	255	`0-255` Opacity for event's texture.
`transparent`	`bool`	False	Transparency flag. When enabled, `graphic` isn't displayed.
`direction`	`String`	S	[N,E,S,W] Initial facing direction (if has `graphic`).
`trigger`	`String`	"onPlayerAction"	Trigger for the behavior of the event.
`move_animation`	`bool`	True	Whether the graphic should be animated when moving/walking.
`stop_animation`	`bool`	False	Whether the graphic should be animated when not moving/walking.
`direction_fix`	`bool`	False	The direction (of the texture) of the event cannot be changed when True.
`through`	`bool`	False	"Walk through walls" switch: when True, collision is ignored and the event can go anywhere (walk on water, walls, holes, etc).
`always_on_top`	`bool`	False	Event's graphic should be drawn last, as to always be "on top" of everything else. Scarcely used.
`movement`	`String`	"Fixed"	"Fixed", "Random" or "Approach".
`movement_speed`	`String`	"Slow"	"Slow" or "Fast". Vaguely defined. Player's movement is "Fast".
`preset`	`String`	None	Proposed "preset" for simple, common events (boulder, door, etc).

* : Mandatory configuration, therefore no default.

Notes :

- `graphic` can be either a string (name of a character file) or an int (*tile id* from the current tileset). Defaults to None : the event has no texture.
- `move_animation:False, stop_animation:True` is mostly used for berry trees.

- `trigger` can have values :
  - *onPlayerAction* : The event is triggered by the player interacting (using action button) with it.
  - *onPlayerTouch* : The event is triggered by the player touching (walking into) it.
  - *onTouch* : The event is triggered by the player touching (walking into) it OR the event touching the player.
  - *onAutorun* : The event is triggered when the map is loaded.
  - *ready* : The event is always triggered, its execution is controlled through its behavior conditions.
  - *onSeen n* : The event is triggered when the player is on the event's line of sight, within n tiles (n is optional, defaults to no limit)

5.6 Command conversion caveats

Unfortunately, it is not possible to cover every command used in Pokémon Essentials, just as it is not possible to extract the underlying semantic behind every event, for the simple reason that Pokémon Essentials doesn't always implement behavior in a straightforward way.

Reading some events, it appears obvious that the authors of Pokémon Essentials had to work around limitations of event commands, relying on scripts to expand capabilities. These result in artifacts.

These include, but are not limited to :

Recurring events : Some events, like *doors*, are common and have straightforward behavior, so authors just copy-pasted them everywhere they were needed, only applying modifications when necessary (eg: appearance and transfer destination on doors).

These were the motivation behind the creation of the `preset` configuration variable on events. It would allow for all recurring commands to be abstracted away, allowing for shorter and cleaner events.

Text hack : According to Pokémon Essentials's wiki [46], there are plenty of modifiers that can be integrated to text in order to modify its behavior : changing text font, size, setting it bold, italic, changing its position or alignment, displaying a selection menu, etc..

This would need to be re-implemented entirely to replicate behavior.

Arbitrary code : The ability to execute scripts in events, and the globally accessible nature of most elements in the game, allows scripts to perform basically any action.

Replicating this much flexibility without a Ruby interpreter and Pokémon Essentials's original scripts would be completely impractical. The only sensible way forward would be to implement the basic function calls and elevate the abstraction level for complex behavior, or fixing it by hand. Decision are to be made on a case-by-case basis. Fortunately, only a few events require such attention.

Arbitrary variable use : This is not exactly an artifact or RPG Maker XP's limitations, but one of Pokémon Essentials's authors. They sometimes use variables, whose name imply a certain usage scope, for unrelated purposes.

This could have been simply avoided by creating new variables. It is not technically an issue : the game already runs with these artifacts, therefore it doesn't need fixing. There is no other solution than fixing it by hand (by creating new variables and changing usage).

5.7 Event conversion

A piece of translation software is provided in the `/RMXP_research/Event converter/` directory. It takes event files dumped by the extraction script (see `Section 4.5`) and outputs their "PEN compliant" equivalents.

Usage : The `Maps` folder must be copied to root (where `Event_reader.py` is) and `PE_variables_switched.py` must have its content filled (see `Section 4.6`). Then, simply launch it :

```
python Event_reader.py
```

The program scans the `Maps` directory for map's events dump files, reads them, separates data into individual events, translates them and finally outputs them into individual files.

Current state : The basic functionality (translation) is entirely implemented for events from Pokémon Essentials. This is because each event's command call must have a translation method (see `RGSS_Command_conversion.py`) assigned to it and, as stated in **Appendix A**, only the 81 commands used by Pokémon Essentials were documented, and therefore implemented here. Expanding the list of supported commands may be needed to process derivative project's events.

An additional feature is present : preset conversion. We previously pointed out the existence of recurring events like doors and the motivation to add a `preset` for these events, greatly simplifying their contents by abstracting their generic code. This feature introduces generic/common event detection and transforms them automatically. Unfortunately, every generic/common event pattern and replacement strategy has to be implemented manually and the current code covers only Pokémon Essential's `door`.

6 Event Interpreter

During the course of the extraction process, event's data was dumped and translated into the proposed Domain Specific Language. The last logical step would be to provide an interpreter capable of executing them, and that's what this section is about.

6.1 Proof of concept

You can find the files for this implementation in the `Interpreter` directory. It was coded in Python (version 3.x) but an equivalent program could be written in most popular languages.

The proposed interpreter can be decomposed in two parts. The first builds an `Event` object from its file, and the second executes it. Let's focus on the first part :

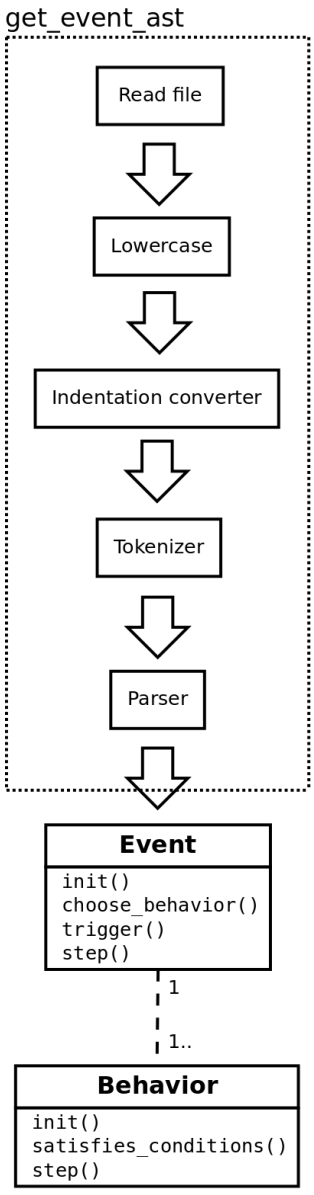

Figure 6: Interpreter

Flow diagram syntax : Rectangles are data manipulation functions and big arrows represent the passage of said data.

Read file : The contents of the event file are read.

Lowercase : As discussed in previous sections, the DSL is case-insensitive. To simplify next steps, lowercase transformation is applied. Note that text between double quotes must be excluded, and that uppercase transformation could have been chosen instead.

Indentation converter : The DSL's specification indicate that indentation is significant. This step converts it because only relative indentation (from one line to the next) has semantic value and should be represented, and also because the tokenizer we used didn't produce intentation tokens.

Tokenizer : We used the tokenizer generator from `funcparserlib` [47] to generate a stream of tokens from a string (the data at the input of the tokenizer) and a list of rules. This lexer is particular because it was made to work with the parser of the same library, which we use for the next step !

Parser : We used the parser generator from `funcparserlib` [47] to generate an abstract syntax tree (AST) from the DSL's formal grammar.

`get_event_ast` is called by an `Event` during its initialization process: from its file, it gets an AST representation from which it can easily reconstruct its configuration and behaviors. This allows the creation of events with a very simple interface !

This approach has a few key properties : Because the AST is generated from the DSL's formal grammar, its format is known in advance and it can be generically processed by `Event`s and its `Behavior`s, making it possible to expand the DSL without necessarily having to adapt their implementation. Also, the event file may contain syntax or grammar errors. If so, the lexer or parser will raise an error with information to correct it.

6.2 Executing an event

Once an `Event` object is correctly initialized, it may be triggered. This is done by calling its `trigger()` method with the trigger identifier, which is compared to the event's trigger (defined via configuration). If matched, the event is successfully triggered.

A triggered event will evaluate which behavior to execute based on the returned value of each associated `Behavior`'s method `satisfies_conditions()`. See `Section 5.2.2` for details.

Then, the event's `step()` method may be called for it to execute one "step" of its behavior. What instructions represent a "step" or not is a decision left to the interpreter, but the general rule is : Only the instructions that have an observable impact on the game world (displaying text, moving the event, playing a sound, waiting, etc) should end a step. Therefore, calling `step()` implies the execution of an arbitrary number of instructions.

Behavior execution is achieved by maintaining a representation of the instructions left to execute and passing it to the interpreter's `execute()` method, which takes care of executing instructions one by one and returning an updated "remaining instruction" object and whether the step is over or not.

The implementation strategy for the `Interpreter` is straightforward : It holds the state of the game (keeps track of global variables) and each instruction of the DSL has its own function which is called by the `execute()` method. This modular approach was chosen for its ease of maintainability and expandability. See `Interpreter.py`

A secondary interpreter was created, with a very similar structure, but with the goal of enabling *script interpretation*. As previously discussed, some events rely on RGSS script calls to implement part of their behavior, but writing a full RGSS interpreter would be too complicated. The envisioned solution was to re-implement these specific scripts in that interpreter, massively reducing its scope while keeping functionality intact. See `pbInterpreter.py`

6.3 Current state of the interpreter

For the moment, the event interpreter only exists as a "proof of concept" (POC), therefore is very basic and implements only a few instruction of the proposed DSL. It was made tested with basic handcrafted event files and Pokémon Essentials's *Fossil reviver* (found at map 11, chosen because its behavior is neither too simple nor too complex), so the best it can do is a short demo.

With some more work, it could be made into a fully-featured interpreter capable of executing every event in Pokémon Essentials or a derived project. Such a piece of software would be a module of the PoGER project, which aims at running these games.

7 RPG Maker XP Maps

As stated previously, maps are instances of `RPG::Map`, which constitutes the game world, describes its appearance and hosts its events. Here are its components :

`tileset_id`	int	Value of a `RPG::Tileset` unique identifier component `id`. The `RPG::Tileset` object can be retrieved through the global hash `$data_tilesets` using the id as the key.
`width,height`	int,int	Attribute equivalent to `data.xsize()` and `data.ysize()`.
`autoplay_bgX`	bool	Indicated whether an audio is to be played as soon as the map is loaded.
`bgm/bgs`	AudioFile	The audio to be played when loading the map. *bgs unused by PE*
`encounter_list`	Array	*Unused by Pokémon Essentials.*
`encounter_step`	int	*Unused by Pokémon Essentials.*
`data`	Table of int	Contain the *map* representation of the 3 tile layers.
`events`	Hash	Contain the *Event* representation (`RPG::Event`) for this map.

Crafting a map data representation for extraction (and further use) requires understanding the meaning of these components, the role of associated classes and the functioning of graphical assets. This is what this section explores.

7.1 Associated classes

In `Figure 2`, we can observe that instances of `RPG::Map` have ties with instances of different classes:

- About the associated `RPG::MapInfo` instance : It contains a few useful informations :

`name`	String	The name of the map.
`parent_id`	int	In the map tree, the id of the parent map.

This information can retrieved directly from the compiled *MapInfos* file :

```
mapinfos = pbLoadRxData("Data/MapInfos")
map_name = mapinfos[id].name
parent_map_id   = mapinfos[id].parent_id
parent_map_name = mapinfos[parent_map_id].name rescue nil
```

RPG Maker XP allows maps to be structured in a hierarchy, where a map can have a "parent map". This information is used to replicate this property on extracted map data.

- `RPG::Tileset` : Represents a normal tileset :

`id`	int	The id of the tileset.
`name`	String	Its name (no extension).
`tileset_name`	String	Name of tileset. *Moved to Map.*
`autotile_names`	Array of String	Names of associated autotiles (up to 7). *Moved to Map.*
`panorama_*`		*Unused by Pokémon Essentials.*
`fog_*`		*Unused by Pokémon Essentials.*
`battleback_name`	String	Name of the texture that appears during combat. *Moved to Map.*
`passages`	2D Table of int	Properties of individual textures.
`priorities`	2D Table of int	Properties of individual textures.
`terrain_tags`	2D Table of int	Properties of individual textures.

- The role of the associated `Game_Map` instance is to be studied further, but current understanding indicates that it's an alias derived from its `RPG::Map` instance that is tailored for Pokémon Essentials's needs. Therefore, it holds no useful data and can be safely disregarded.
- `RPG::AudioFile` : Basic data container :

`name`	String	The name of the audio file (no extension).
`volume`	int	Acts like a volume slider, normalized at 100.
`pitch`	int	Allows to adjust sound pitch, normalized at 100.

  A global variable (`$data_tilesets, Array`) stores this information.
- `Table` : Used for 2D/3D arrays, with 3 class mathods to retrieve dimensions. x and y correspond to their GUI map representation and z the map layers (background, intermediate, foreground).

7.1.1 Values for tileset tables

Each table contains `int` values, one per texture for each category. This value is to be interpreted .

`passages`: let's consider the 6 least significant bits

- bit 0 : Can't come from S
- bit 1 : Can't come from W
- bit 2 : Can't come from E
- bit 3 : Can't come from N
- bit 4 : Is a bush
- bit 5 : Is a counter (tile through which interaction is possible)

Examples : a bush at which the player can come from any direction has value `0b01 0000 = 16`, and an obstacle (which isn't a counter) has value `0b00 1111 = 15`.

`terrain_tags`: see `PBTerrain` module for values. They are used by Pokémon Essentials, mostly to determine if an action is compatible with a tile (eg : surfing, diving, climbing, etc).

`priorities`: default value is 0. Any higher value means that the texture has priority. In practice, it means that the texture is "above" its own layer (drawn last). Combined with passability, it's possible for it to be on the same layer as NPCs/the player and give the illusion that they can pass behind it. It's mostly used for the top of tall trees, houses, etc.

Proposition : integrate `priorities` to `passages` as a 7^{th} bit (*bit 6 : Can be passed behind*). Note that this proposition is implemented in the extraction script.

7.2 About graphical assets

When dealing with (most) graphical assets in RPG Maker XP, including tilesets, autotiles and maps, a few constraints apply :

- The base unit is the **tile**, which is a 32 by 32 square.
- Tilesets must have a horizontal dimension of exactly 8 tiles (256 pixels), but can be arbitrarily tall (n tiles tall).

- Tilesets can have up to 7 associated autotiles. This has proven to be a limiting factor in Pokémon Essentials.
- Autotiles are special assets. They can be put on a map like any texture from a tileset, but have the properties of being context-aware and animation-capable :

  They form coherent surfaces by manipulating components of their texture in order to give the impression of continuity when multiple instances of the same autotile are neighbors. They can be used to create basically any surface that needs to deal with how its border with an other surface looks.

  This property is generated at the moment of placing the autotiles by automatically determining the right index for each, and therefore is an artifact of the map editor.

  Two formats were identified (*frames are situated side-by-side horizontally*) :

  - $3N \times 4$: a 3 tile horizontal, 4 tile vertical, N frame image. This is the official autotile format for RPG Maker XP.
  - $N \times 1$: a 1 tile horizontal, 1 tile vertical, N frame image. This is used for animated individual tiles.

  Animation is produced by iterating over frames of an autotile each time the screen is drawn.
- Every tile has properties (see section "values for tileset tables"). This includes autotiles, which notably have a single `passages`, `terrain_tag` and `priorities` value.

We believe the decision of tying tiles and autotiles together in their object representation, and then using the original tileset file name to refer to said object in maps, was a poor design choice. In consequence, the extraction script decouples maps, autotiles and tilesets, allowing for more flexibility (for example, it raises the possibility of removing the 7 autotile per tileset limit).

7.3 Autotile format

Researching how autotiles are adaptively generated proved to be a tedious manual process of reverse-engineering.

We used a "test map" we created as a starting point for studying Pokémon Essentials (included at **Appendix D**). By placing autotiles in various shapes and extracting map data, we were able to manually correlate tile content to their index (and understand both the absolute and relative indexing systems).

An article about the format of autotiles for another RPG Maker software [48] proved helpful for understanding how autotiles work. It turns out that autotiles tiles are divided into four separate "sub-tiles" of 16 by 16 pixels, and each index correspond to a combination of sub-tiles into a unique tile.

As you can see, only 10 tiles are actually used out of 12. Some autotile assets for Pokémon Essentials contained texture for the redundant/unused tiles for some unidentified reason, which make us suspect there might be something we are missing about their format.

The result of our research can be found in the `AutotilesSubtileMap.ods` spreadsheet file. It lists indexes and their corresponding subtile components. A Python implementation (`LoadImage.py`)

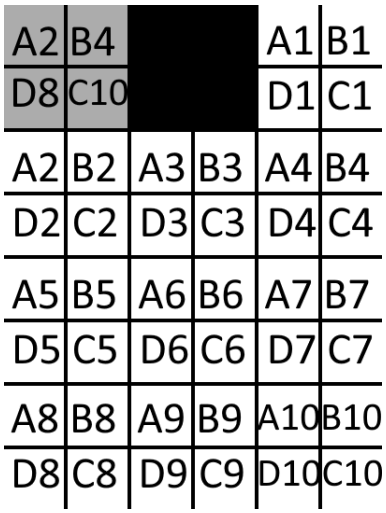

Figure 7: 3 by 4, single frame autotile format

is also provided : it can reconstructs maps from map data, tileset and autotiles, including GIF output (see `Map032_out.gif` for example), demonstrating the feasibility of lossless map extraction and reconstruction.

7.4 Map representation

At its core, a RPG Make XP map is a *tiled map* : it is a 2-dimensional grid of textures (tiles) of fixed size.

The `data` component is a 3-dimensional table of integer values with the following properties :

- The 3rd dimension is used to represent **layers** : exactly 3 tiled maps are used for any RPG Maker XP map :
  - Background : Contains mainly the textures that make the ground and other elements that are always "below player's level". Any event or texture of higher layer will be displayed *above* it.
  - Intermediate : Contains mainly the textures that should be "at player's level", which includes most elements the player can interact with (including bumping into).
  - Foreground : Contains mainly the textures that should be "above player's level", typically used for elements below which the player can stand.

- Each layer obeys the same indexing convention :

Value(s)	Description
`0`	Reserved value for "No texture"
`1-47`	Reserved, unused.
`48-95`	Range used for autotile 1.
`96-143`	Range used for autotile 2.
⋮	⋮
`336-383`	Range used for autotile 7.
`384-`	Range used for tileset.

  This way of reserving 48 indexes per autotile results from autotile's mechanism for generating context-aware textures.

  Note : each range describes the *absolute positions* between which a *relative indexing system* takes place. For example, index 52 is interpreted as index 4 for autotile 1 and index 484 is index 100 on the tileset (remember that indexes begin at zero, not one).

With this information, a map can be reconstructed from indexes by mapping them to the corresponding textures and drawing the layers in order.

7.5 Map and Tileset files

There should be a well-defined way of representing extracted map and tileset information.

Map

- Located in the "`Maps`" directory.
- Naming convention : `<id>_<name>.json` . The *name* is only here as a quality-of-life addition, for developers to easily know what map they're dealing with. It must be present, though.

  The extension isn't really important but reflects the content's format.
- Location agnostic : A map file can be at the root of the "Maps" directory, or in any subdirectory, allowing for the same flexibility as RPG Maker XP's UI.
- Content : utf-8 encoded JSON with the following entries

`_class`	String	Must be `"Map"`.
`name`	String	Map's name. Used for displaying current location.
`width`	int	Map's width. Should be checked against the content of `table`.
`height`	int	Map's height. Should be checked against the content of `table`.
`battleback`	String	Picture to be displayed behind battles.
`tileset`	String	Map's tileset (file name without extension).
`autotiles`	Array of String	Map's autotiles (file name without extension).
`autoplay_bgm`	bool	Sets whether the bmg should be played when entering the map.
`bgm`	String	Map's background music.
`table`	3D array of int	Describes map's texture placement (3 layers).

  Note : `bgm` formats : `"<name>"` or `"<name>,<volume>,<pitch>"`

  Note : `table` must have dimensions `width x height x 3`

Tileset and Autotiles

- Located in the "`Tileset_data\Tilesets`" and "`Tileset_data\Autotiles`" directories respectively.
- Naming convention : `<name>.json` . The *name* is only here as a quality-of-life addition, for developers to easily know what map they're dealing with. It must be present, though.
- Content : JSON with the following entries

`_class`	String	Must be `"Tileset"`.
`name`	String	Tileset's file name (no extension). Referenced by map's property `tileset`
`passages`	(1D array of) int	Tileset's passage table.
`terrain_tags`	(1D array of) int	Tileset's terrain tag table.

  Note : for autotiles, all values in `passages` and `terrain_tags`, therefore these fields were simplified to a *single integer value*.

  Note : RPG Maker XP seem to fail to load tilesets/autotiles after an extraction with destination "`Graphics\Tilesets`" and "`Graphics\Autotiles`" directories, probably because there are foreign files in the graphic folder. For that reason, other directories were used.

- Advantages : Decoupling a `Tileset` from the autotilesets used for a particular map allows added flexibility.

7.5.1 On naming decisions

Originally, we kept the use of a *3-digit numerical identifier* in the file names, because it was RPG Maker XP/Pokémon Essntials's identifier format for map. We were asked to consider switching to *name identifiers* instead. Numerous considerations ensued :

- Transitioning to another identification system would be complex, for questionable gain.
- Using a file name identifier means that identification information needs to reside inside the file itself due to restriction on file name contents imposed by various file systems.

  Without it being in the name, some other strategy of file identification would be needed :

  – Reading each file, looking for the one containing the right id

  – A database associating each file with its id

  None of these are particularly engaging.

- The idea of using names for identification is attractive at first glance : user-friendly and intuitive, it's great !

  Unfortunately, there are some issues with this solution, including but not limited to :

  – As stated before, it would be complex to transition to this new identification system.

  – Not all characters are usable : every file system imposes limitations, which would impose unwanted restrictions for developers.

  – Dealing with names/strings means that ids are typically longer and less formally defined (can contain more caracters, including casing, accents, etc), therefore amplifying the risk of any id containing errors.

  - Developers involved in this type of project are typically capable of dealing with numerical ids without issue already.
  - For large projects with hundreds/thousands of maps, name collisions may force developers to use convoluted or cumbersome naming schemes.

  Note that some of the issues identified may be circumvented by re-introducing the "name" field in the files themselves, but at this point the percieved value of the change would become nil.

- While it is true that numerical id collisions is possible and using name ids may help preventing them, there are other strategies available.

  Planning, for example : reserving ranges of values for specific uses ahead of time.

In conclusion, it is our estimation that using numerical ids is a reasonable choice, as it is a tried-and-true solution that doesn't compromise on usability (at least not too much).

On the other hand, the suggestion of abandoning the 3 digit format for numerical ids was considered and adopted, as it would allow larger projects to exist and didn't pose any technical problem.

7.6 Map reconstruction

One of the technical challenges facing the PoGER project is its ability to generate maps from extracted information and original graphical assets. Without this capability, no graphical interpreter could run games and provide the same experience as the original game engine.

Current state : A proof of concept implementation is available here : `Map reconstruction/LoadMap.py`. It contains reader classes for `Tileset`s and `Autotile`s, allowing to reconstruct a whole map texture by texture, and even supports animation (for autotiles) and GIF output.

The program is feature-complete but its performance isn't satisfactory, because it takes a few seconds to generate a map frame, which is considered excessive. This may be because of the lack of hardware acceleration or inadequacy of the used image library. We believe that an actual game engine, with optimized graphic libraries wouldn't exhibit these performance shortcomings.

7.6.1 Note on tileset compacting

During map reconstruction attempts, we found out that most maps only use a fraction of tiles available to them and programmed a way of converting maps and their tilesets to a "compact" format (creating an alternate tileset with only used tiles, and adapting map data accordingly).

Although it is functional and may be advantageous for running on low memory systems, the scattering of graphical assets and the loss of tile adjacency render this approach pointless or impractical.

8 Roadmap for future works

Due to the large scope of the PoGER project, a lot of research and implementation work is left before the objective of proposing a Pokémon Essentials alternative game engine is met. We will discuss here the main subjects for future developments.

8.1 Event interpreter

As of now, the event interpreter is far from complete. Further development would mainly consist of implementing scripts call functions that replicate the behavior of the original RGSS scripts.

8.1.1 PEN

Along with the event interpreter, PEN may need to be expanded to fit unforeseen event commands or other features we have yet to consider.

8.2 PBS files

As mentioned in `Section 4.1.1`, PBS files contain data specific to the mechanics of a Pokémon world (items, species, types, etc).

We tried to code a reader for PBS files (see `/RMXP_research/PBSreader/`). What we found is that these files don't have a strictly uniform format, and therefore each requires a custom reader. About half of Pokémon Essentials's PBS files have a reader. Further development may try to continue implementing readers to cover every PBS file or explore another way of reading its data.

8.3 Maps

As in any RPG, the game world is a collage of contiguous maps the player can walk through. While map contents are well understood at this point, we still don't know how maps are assembled. Resolving this mystery and finding a way to implement "map coherence" in the final game engine will be an important step.

On the subject of maps, a suitable tiled map graphical library is needed to display them, and more broadly any graphical user interface needs to be implemented.

8.4 Save files

A very basic feature of any game is its progress saving mechanic. It shall be implemented as part of the game engine. One interesting extension to this feature would be the capacity of importing save files from the original RPG Maker XP/Pokémon Essentials engine, allowing seamless transition for its players. We estimate that its probably feasible but it may require an extraction process similar to what can be seen in `Section 4.5`.

8.5 Dialogue

One of the obvious limitations of the original Pokémon Essentials was that dialogue is explicitly included in events, meaning that any game needing translations needed to implement it own dialogue translation system. This is bad because, with no standard, porting these games may involve a lot of manual work to keep this feature. Even worse, there would need to be one version of each game for every supported language.

Further research is needed to define a satisfactory way of introducing multi-language support for the PoGER game engine, and a way of porting games that feature multiple languages.

8.6 Online features

From generation 4 and onward, Pokémon games have featured online capabilities, mainly online creature exchange and battle. Even before that, local wired and wireless console-to-console technologies have allowed players to perform these interactions. The PoGER game engine must implement similar features, therefore more research on this subject is needed.

8.7 Mini games

A "mini game" here refers to any temporary gameplay interruption in which the player doesn't explore the game world, including the Pokémon battles which are the hallmark of the series. Support for these "mini games" is an apparent necessity, and making it possible for game developers to create their own seems like a good idea.

We believe this is an important feature that would require a lot of research and implementation effort.

8.8 PoGER game engine

As previously discussed, there are a lot of considerations that would go into the design of the final PoGER game engine. Unfortunately, this is a bit outside of our capabilities and would require the guidance of people experienced with game engines.

9 Conclusion

This section offers a summary of the work presented through this report and offers a critique of it.

9.1 Summary of contributions

Although the topic of fan-made Pokémon games isn't popular in the academic sphere, we believe we offered a pertinent insight into its challenges and put it into context with larger current trends and research fronts such as the software preservation efforts.

While it is admittedly incomplete, we laid the foundations for the PoGER project, which aims at providing solutions to identified challenges, addressed feasibility concerns with proof of concepts and proposed a framework for future development.

Analysis : We observed the state of the current game development scene and tried to identify systemic issues and their root causes. This enabled us to propose potentially effective solutions.

Research : A large part of our contribution comes as information, sometimes in bulk in table format. This is the result of tedious reverse-engineering effort complemented with online research.

Implementation : Multiple pieces of software were created to demonstrate the feasibility of the proposed framework and a language was defined (PEN) to answer open source and user friendliness issues.

9.2 Critique of contributions

Originally, we thought the PoGER project would be mostly finished by the deadline we fixed, but history proved us wrong and `Section 8` is a testament to our underestimation of the task we took on. We estimate that the only realistic way we can ever push it to completion would be to continue working on it for at least a few thousand more developer hours.

Legality concerns for this type of project were never seriously engaged. Stakeholders of the Pokémon franchise may decide this project goes against their interest and order its content to be censored or taken offline. This is a blind spot for us, as this is thought of as a grey area for research and development and we have no expertise on it. As far as we know, only media assets reused from official games may be protected by copyright law, not any code we wrote or information we gathered.

Finally, PoGER is a large project born out of the intuition that bringing solutions to motivated game developers would revitalize a dormant game making scene. History is full of similar projects, and a lot of them failed. If it is ever completed, then only time will be able to tell if its destiny is to become successful or yet another failure.

Bibliography

[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] *Pokemon Uranium - Lutris*. URL: https://lutris.net/games/pokemon-uranium/.

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

[25] *Definitions of Digital Preservation*. ALA Annual Conference, Washington, D.C., June 24, 2007. URL: http://www.ala.org/alcts/resources/preserv/defdigpres0408.

[26] Kyong-Ho Lee et al. "The state of the art and practice in digital preservation". In: *Journal of research of the National institute of standards and technology* 107.1 (2002), p. 93.

[27] Julien Masanès. "Web archiving: issues and methods". In: *Web archiving*. Springer, 2006, pp. 1–53.

[28] David SH Rosenthal. "Emulation & virtualization as preservation strategies". In: *Report commissioned by The Andrew W. Mellon Foundation* (2015).

[29] Mendel Rosenblum. "The reincarnation of virtual machines". In: *Queue* 2.5 (2004), pp. 34–40.

[30] Melanie Swalwell. "Towards the preservation of local computer game software: Challenges, strategies, reflections". In: *Convergence* 15.3 (2009), pp. 263–279.

[31] Roberto Di Cosmo and Stefano Zacchiroli. "Software heritage: Why and how to preserve software source code". In: 2017.

[32] Charles M Schweik and Robert C English. *Internet success: a study of open-source software commons*. MIT Press, 2012.

[33] Edward M Corrado. "Software Preservation: An Introduction to Issues and Challenges". In: *Technical Services Quarterly* 36.2 (2019), pp. 177–189.

[34] Helge Holzmann, Wolfram Sperber, and Mila Runnwerth. "Archiving software surrogates on the web for future reference". In: *International Conference on Theory and Practice of Digital Libraries*. Springer. 2016, pp. 215–226.

[35] *Software Heritage*. URL: https://www.softwareheritage.org.

[36] *Zenodo*. 2019. URL: https://zenodo.org/record/3553949#.X0jIaC1h3UI.

[37] *GitHub Archive program*. 2020. URL: https://archiveprogram.github.com.

[38] *RGSS Specifications*. URL: https://rpgmaker.fixato.org/Manual/RPGVXAce/rgss/rgss.html.

[39] *RPG Maker VX Ace Release*. Feb. 21, 2012. URL: https://blog.rpgmakerweb.com/announcements/rmvx-ace-release/.

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

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

[42] *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/.

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

[44] *Forum thread about Gemini Editor*. URL: https://forum.chaos-project.com/index.php/topic,10420.0.html.

[45] *GitHub repository for Gemini Editor*. URL: https://github.com/terabin/Gemini.

[46] *Pokémon Essentials - Messages*. URL: https://essentialsdocs.fandom.com/wiki/Messages.

[47] *GitHub - vlasovskikh/funcparserlib*. URL: https://github.com/vlasovskikh/funcparserlib.

[48] *Tutorial: Making an Autotile*. URL: https://blog.rpgmakerweb.com/tutorials/tutorial-making-an-autotile/.

Note : Unless not applicable or stated otherwise, access date for every reference's hyperlink is `August 2020`. Future consultation may require the use of Internet Archival services such as the `Wayback Machine` [1].

[1] https://web.archive.org/

Appendix A

This appendix contains data relative to the reverse-engineering efforts on RPG Maker XP events.

Miscellaneous information about RPG Maker XP events

Codes used in Pokemon Essentials 17.2 (*81 total*) :

> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 33, 34, 37, 38, 39, 40, 41, 42, 44, 101, 102, 104, 106, 108, 111, 112, 113, 115, 118, 119, 121, 122, 123, 125, 201, 202, 208, 209, 210, 221, 222, 223, 225, 231, 232, 235, 236, 241, 242, 247, 248, 249, 250, 314, 354, 355, 401, 402, 404, 408, 411, 412, 413, 655

Implementation details :

- `RPG::MoveCommand` use range [1-45]
- `RPG::EventCommand` use range [101-x], $x \geq 655$
- A "frame" is defined as $\frac{1}{20}$ second $\Rightarrow$ change into milliseconds $m = n * 1000/20 \equiv n * 50$.
- Every event has an ID (integer > 0). Actions that can affect other events can target the player using id `-1` and the current event using id `0`.
- Special variables : MapID, PartyMembers, Gold, Steps, PlayTime, *Timer*, SaveCount.

  They should all be read accessible. Underlined ones should also be write accessible. *Italic ones are probably not used.*

List of commands

These are the commands used by Pokémon Essentials events, indexed by their *command code*.

0	Description	Nothing, empty command or end of the event command list
	Parameters	None
	Notes	Will not be represented
1	Description	`RPG::MoveCommand` - Move to the South
	Parameters	None
	Notes	See footnote[1]
2	Description	`RPG::MoveCommand` - Move to the West
	Parameters	None
	Notes	See footnote[1]
3	Description	`RPG::MoveCommand` - Move to the East
	Parameters	None
	Notes	See footnote[1]
4	Description	`RPG::MoveCommand` - Move to the North
	Parameters	None
	Notes	See footnote[1]
5	Description	`RPG::MoveCommand` - Move to the SouthWest
	Parameters	None
	Notes	See footnote[1]
6	Description	`RPG::MoveCommand` - Move to the SouthEast
	Parameters	None
	Notes	See footnote[1]
7	Description	`RPG::MoveCommand` - Move to the NorthWest
	Parameters	None
	Notes	See footnote[1]
8	Description	`RPG::MoveCommand` - Move to the NorthEast
	Parameters	None
	Notes	See footnote[1]
9	Description	`RPG::MoveCommand` - Move at random (N,E,S,W)
	Parameters	None
	Notes	See footnote[1]
10	Description	`RPG::MoveCommand` - Move towards player
	Parameters	None
	Notes	See footnotes[1,3]
11	Description	`RPG::MoveCommand` - Move away from player
	Parameters	None
	Notes	See footnotes[1,3]
12	Description	`RPG::MoveCommand` - Take 1 step forward
	Parameters	None
	Notes	See footnote[1]

13	Description	`RPG::MoveCommand` - Take 1 step backward
	Parameters	None
	Notes	See footnote[1]
14	Description	`RPG::MoveCommand` - Jump to relative coordinates on the same map
	Parameters	[2] - **0**:deltaX `[signed integer]`, **1**:deltaY `[signed integer]`
	Notes	
15	Description	`RPG::MoveCommand` - Wait n seconds
	Parameters	[1] - **0**:number of seconds to wait n `[integer` $\in \mathbb{N}^*$`]`
	Notes	Typically $n == 2$, but values up to 15 were found in Pokémon Essentials.
16	Description	`RPG::MoveCommand` - Turn towards South
	Parameters	None
	Notes	See footnote[2]
17	Description	`RPG::MoveCommand` - Turn towards West
	Parameters	None
	Notes	See footnote[2]
18	Description	`RPG::MoveCommand` - Turn towards East
	Parameters	None
	Notes	See footnote[2]
19	Description	`RPG::MoveCommand` - Turn towards North
	Parameters	None
	Notes	See footnote[2]
20	Description	`RPG::MoveCommand` - Turn 90° right, relative to current position
	Parameters	None
	Notes	See footnote[2]
21	Description	`RPG::MoveCommand` - Turn 90° left, relative to current position
	Parameters	None
	Notes	See footnote[2]
22	Description	`RPG::MoveCommand` - Turn 180°
	Parameters	None
	Notes	See footnote[2]
23	Description	`RPG::MoveCommand` - Turn 90° to the left or right, at random
	Parameters	None
	Notes	See footnote[2]
24	Description	`RPG::MoveCommand` - Turn at random (90° or 180°)
	Parameters	None
	Notes	See footnote[2]
25	Description	`RPG::MoveCommand` - Turn towards player
	Parameters	None
	Notes	See footnotes[2,3]
26	Description	`RPG::MoveCommand` - Turn away from player
	Parameters	None
	Notes	See footnotes[2,3]

33	Description	`RPG::MoveCommand` - Turn ON walking animation
	Parameters	None
	Notes	
34	Description	`RPG::MoveCommand` - Turn OFF walking animation
	Parameters	None
	Notes	
37	Description	`RPG::MoveCommand` - Turn ON "through"
	Parameters	None
	Notes	Equivalent to activating "walk through walls", making it possible to walk through impassable tiles/characters.
38	Description	`RPG::MoveCommand` - Turn OFF "through"
	Parameters	None
	Notes	Equivalent to deactivating "walk through walls".
39	Description	`RPG::MoveCommand` - Always on top ON
	Parameters	None
	Notes	Elevate the display priority, therefore bringing the event graphic to the forefront (above any tile/character)
40	Description	`RPG::MoveCommand` - Always on top OFF
	Parameters	None
	Notes	
41	Description	`RPG::MoveCommand` - Change event's graphic
	Parameters	[2] - **0**:texture file `[String]`, **1**:hue, **2**:direction d `[integer]`, **3**:step `[integer 0-3]`
	Notes	See note[5]. **0** without extension. **1** is unused.
42	Description	`RPG::MoveCommand` - Change event's graphic opacity
	Parameters	[1] - **0**:new opacity value n `[integer 0-255]`
	Notes	
44	Description	`RPG::MoveCommand` - Play a sound effect
	Parameters	TODO
	Notes	
101	Description	`RPG::EventCommand` - Show text
	Parameters	[1] - **0**:text s `[String]`
	Notes	s must be properly double-quoted and formatted (inner double-quotes and backslashes must be escaped).
401	Description	`RPG::EventCommand` - Show text (continued)
	Parameters	[1] - **0**:text s `[String]`
	Notes	Continuation of 101.
102	Description	`RPG::EventCommand` - Show choices
	Parameters	[2] - **0**:array of size n `[Array of Strings]`, **1**:cancel behaviour `[integer 0-4]`
	Notes	Displays up to 4 selectable options in a message window. Cancel behaviour : 0 disallow canceling, 1-4$\leq n$ selects choice by default.
104	Description	`RPG::EventCommand` - Change text options
	Parameters	[2] - **0**:position p `[integer 0-2]`, **1**:window border b `[integer 0-1]`
	Notes	Sets message window position and border. p follows "common relation 1", b follows "common relation 2"
106	Description	`RPG::EventCommand` - Wait
	Parameters	[1] - **0**:number of frames to wait n `[integer` $\in \mathbb{N}^*$`]`
	Notes	Conversion to milliseconds chosen for its more precise and general use : $m = n * 1000/20 \equiv n * 50$, TODO:research its use

108	Description	`RPG::EventCommand` - Comment
	Parameters	[1] - **0**:comment text *s* `[String]`
	Notes	Has no effect. TODO:research link to particle effects.
408	Description	`RPG::EventCommand` - Comment (continued)
	Parameters	[1] - **0**:comment text *s* `[String]`
	Notes	Happens after a 108.
111	Description	`RPG::EventCommand` - Conditional branch
	Parameters	See "Conditional branch" section.
	Notes	Complex but essential command.
112	Description	`RPG::EventCommand` - Loop
	Parameters	None
	Notes	Loops over commands until broken. TODO:research usage
113	Description	`RPG::EventCommand` - Break loop
	Parameters	None
	Notes	Escape innermost loop. TODO:research usage
115	Description	`RPG::EventCommand` - Exit Event Processing
	Parameters	None
	Notes	TODO:research usage
118	Description	`RPG::EventCommand` - Label
	Parameters	[1] - **0**:label name *s* `[String]`
	Notes	Sets a label to allow jumping to.
119	Description	`RPG::EventCommand` - Jump to Label
	Parameters	[1] - **0**:label name *s* `[String]`
	Notes	Jumps to a label.
121	Description	`RPG::EventCommand` - Control switches
	Parameters	[3] - **0**:starting switch *ssa* `[integer]`, **0**:ending switch *ssz* `[integer]`, **0**:state *n* `[integer]`
	Notes	Batch control is unused in PE, therefore deprecated. *n* follows "common relation 3".
122	Description	`RPG::EventCommand` - Control variables
	Parameters	See "Control variables" section.
	Notes	Batch control is unused in PE, therefore deprecated.
123	Description	`RPG::EventCommand` - Control Self Switch
	Parameters	[2] - **0**:SS character *s* `[String of length 1]`, **1**:new state *n* `[integer 0-1]`
	Notes	*n* follows "common relation 3".
125	Description	`RPG::EventCommand` - Change Gold
	Parameters	[3] - **0**:operation *o* `[integer 0-1]`, **1**:operand *n* `[integer 0-1]`, **2**:value *v* `[integer]`
	Notes	Values of *n*: 0:*v* is a constant, 1:*v* is a variable(id). *o* follows "common relation 4"
201	Description	`RPG::EventCommand` - Transfer Player
	Parameters	[6] - **1**:map *m* `[integer]`, **2**:coordinate *x* `[integer]`, **3**:coordinate *y* `[integer]`, **4**:player direction *d* `[integer]`, **5**:fading *f* `[integer]`.
	Notes	{0} must be 0, 1 unused in PE. *d* follows "common relation 5". *f* follows "common relation 3".

202	Description	`RPG::EventCommand` - Set Event Location
	Parameters	[5] - **0**:event id e `[integer]`, **2**:coordinate x `[integer]`, **3**:coordinate y `[integer]`, **4**:direction d `[integer]`
	Notes	Change an event's location on the current map. {1} must be 0, other values unused in PE. d follows "common relation 5".
208	Description	`RPG::EventCommand` - Change Transparency Flag
	Parameters	[2] - **0**:flag d `[integer 0-1]`
	Notes	When transparency is set, the graphic isn't displayed. d follows "common relation 3".
209	Description	`RPG::EventCommand` - Set Move Route
	Parameters	[2] - **0**:target id d `[integer]`, **1**:`RPG::MoveRoute`
	Notes	
210	Description	`RPG::EventCommand` - Wait for Move's Completion
	Parameters	None
	Notes	To be put after a Set Move Route. Without it, further commands can be executed before the end of the walking animation.
221	Description	`RPG::EventCommand` - Prepare for transition
	Parameters	None
	Notes	Freezes the screen, so there's nothing moving during the transition. To be fused with Execute Transition.
222	Description	`RPG::EventCommand` - Execute Transition
	Parameters	[1] - **0**:transition file name s `[String]`
	Notes	Plays the animation. TODO:research how transition work.
223	Description	`RPG::EventCommand` - Change Screen Color Tone
	Parameters	[2] - **0**:`RPG::Tone`, **1**:duration(frames) d `[integer]`
	Notes	Typically used in fade out (to black/white)/fade in cycles. d to be changed into ms.
225	Description	`RPG::EventCommand` - Screen Shake
	Parameters	[3] - **0**:shake power `[integer]`, **1**:shake speed `[integer]`, **2**:duration(frames) d `[integer]`
	Notes	Scarcely used in PE, {0} and {1} are not well defined so they can be deprecated. d to be changed into ms.
231	Description	`RPG::EventCommand` - Show Picture
	Parameters	See "Show Picture" section.
	Notes	
232	Description	`RPG::EventCommand` - Move Picture
	Parameters	See "Move Picture" section.
	Notes	
235	Description	`RPG::EventCommand` - Erase Picture
	Parameters	[1] - **0**:picture id `[integer]`
	Notes	
236	Description	`RPG::EventCommand` - Set Weather effect
	Parameters	[3] - **0**:weather id `[integer]`, **1**:power `[integer]`, **2**:transition duration (frames) `[integer]`
	Notes	*power* and *transition duration* to be removed. TODO:research how weather is generated. *weather id* follows "common relation 13".
241	Description	`RPG::EventCommand` - Play BGM
	Parameters	[1] - **0**:audio a `[AudioFile]`
	Notes	
242	Description	`RPG::EventCommand` - Fade Out BGM
	Parameters	[1] - **0**:duration (seconds) n `[integer]`

247	Description	`RPG::EventCommand` - Memorize BGM/BGS
	Parameters	None
	Notes	
248	Description	`RPG::EventCommand` - Restore BGM/BGS
	Parameters	None
	Notes	
249	Description	`RPG::EventCommand` - Play ME
	Parameters	[1] - **0**:audio *a* `[AudioFile]`
	Notes	
250	Description	`RPG::EventCommand` - Play SE
	Parameters	[1] - **0**:audio *a* `[AudioFile]`
	Notes	
314	Description	`RPG::EventCommand` - Restore All
	Parameters	[1] - **0**:actor id `[integer]`
	Notes	Equivalent to healing and restoring PPs. Ignore parameter.
354	Description	`RPG::EventCommand` - Return to Title Screen
	Parameters	None
	Notes	
355	Description	`RPG::EventCommand` - Script
	Parameters	[1] - **0**:script string `[String]`
	Notes	To be overhauled.
655	Description	`RPG::EventCommand` - Script (continued)
	Parameters	[1] - **0**:script string `[String]`
	Notes	To be overhauled.
402	Description	`RPG::EventCommand` - When
	Parameters	[1] - **0**:choice id `[integer]`, **1**:choice string equivalent `[integer]`
	Notes	Used with choices and conditional branches, has code block per choice.
404	Description	`RPG::EventCommand` - End of When
	Parameters	None
	Notes	
411	Description	`RPG::EventCommand` - Else
	Parameters	None
	Notes	Used with conditional branch `111`.
412	Description	`RPG::EventCommand` - Branch End
	Parameters	None
	Notes	End of a code block (as result of branching). TODO:investigate whether it is present in every code block and if it should be represented (is indentation sufficient?).
411	Description	`RPG::EventCommand` - Repeat above
	Parameters	None
	Notes	Marks end of Loop `112` code block.

[1] Movements consolidated with new *Move* command with argument.
[2] Turs consolidated with new *Turn* command with argument.
[3] Unknown algorithm to determine direction "towards player" and "away from player.
[4] Is part of a command sequence that should be merged in a sensible way.
[5] *step* is the horizontal offset (column), *direction* is the vertical offset.

Common relations

In parenthesis are the proposed representation or information :

1. 0:Top, 1:Middle, 2:Bottom
2. 0:Show, 1:Hide
3. 0:ON, 1:OFF
4. 0:Increase, 1:Decrease (`+=`,`-=`)
5. 0:Keep same, 2:Down, 4:Left, 6:Right, 8:Up (K,S,W,E,N)
6. 0:'`==`', 1:'`>=`', 2:'`<=´`', 3:'`>`', 4:'`<`', 5:'`!=`'
7. 0:constant, 1:variable
8. 0:'`>=`', 1:'`<=`'
9. 0:'`=`', 1:'`+=`', 2:'`-=`', 3:'`*=`', 4:'`/=`', 5:'`%=`' (affectation, increment, decrement, multiplication, division, modulo)
10. 0:coordinate X, 1:coordinate Y, 2:direction (3-5 unused)
11. 0:NW, 1:Centered (picture coordinate origin)
12. 0:Normal, 1:Additive, 2:Substractive (blending type)
13. 0:None, 1:Rain, 2:Storm, 3:Snow

TODO:determine if division is always rounded to an integer (and how) or not.

Complex commands

Some commands have complex behaviour that doesn't fit in the table above, therefore detailed explanation were put here instead.

Conditional branch - 111

This command is RPG Maker XP's equivalent of an 'if' instruction, and therefore hinges on expressing a condition. Given the expansive list of conditions that can be expressed, its syntax is quite complex.

The *first parameter* is crucial : it defines the type of condition. `integer 0-12` :

0 Check *Switch* state.

Parameters	[3] - **1**:switch id n `[integer]`, **2**:switch state d `[integer 0-1]`
Notes	d follows "common relation 3".
Representation	"If, n.toString(), d.toString()"

1 Check *Variable* value.

Parameters	[5] - **1**:variable id n `[integer]`, **2**:what it is compared to m `[integer]` **3**:constant or variable id x `[integer]`, **4**:comparator c `[integer]`
Notes	c follows "common relation 6", m follows "common relation 7".
Representation	m=='constant' : "If, n.toString(), c.toString(), x" m=='variable' : "If, n.toString(), c.toString(), x.toString()"

2 Check *Self-Switch* state.

Parameters	[3] - **1**:self switch character n `[String of size 1]`, **2**:switch state d `[integer 0-1]`
Notes	d follows "common relation 3".
Representation	"If, n, d.toString()"

6 Check *Event* direction.

Parameters	[3] - **1**:event id n `[integer]`, **2**:direction d `[integer 0-1]`
Notes	d follows "common relation 5".
Representation	"If, n.toString(), Facing, d.toString()

7 Check *Player's money.*

Parameters	[3] - **1**:amount n `[integer]`, **2**:comparator d `[integer 0-1]`
Notes	d follows "common relation 8".
Representation	"If, Money, d.toString(), n

12 Check *Script's return.*

Parameters	[2] - **1**:Script s `[String]`
Notes	Script must return a boolean (prehaps returning nothing is OK?)
Representation	"If, Script, s

Values `3,4,5,8,9,10,11` were not found in PE, therefore not researched.

Control variables - 122

Parameters **0** and **1** `[integer]` are indexes for the range of variables that will be affected. Variable is s

As batch control of variables is unused in PE, it is deprecated in the representation (parameter **1** is ignored).

Parameter **2** o `[integer 0-5]` sets the **operation** to be performed on the variable, and follows "common relation 9".

Parameter **3** defines the **operand type** `[integer 0-7]` :

0 - Constant.

Parameters	[5] - **4**:constant n `[integer]`
Notes	
Representation	"Control, s.toString(), o.toString(), n

2 - Random integer.

Parameters	[6] - **4**:constant a `[integer]`, **5**:constant z `[integer]`
Notes	Will choose a number $x \in [a, z]$. TODO:check if a and z are included.
Representation	"Control, s.toString(), o.toString(), [a,z]

6 - Event's attribute.

Parameters	[6] - **4**:event id n `[integer]`, **5**:attribute id d `[integer 0-2]`
Notes	d follows "common relation 10".
Representation	"Control, s.toString(), o.toString(), Event, n.toString(), d.toString()

7 - Only used once, to put the "Money"/"Gold" special variable in a temporary variable to be used in a condition, therefore isn't really needed.

Values `1,3,4,5` were not found in PE, therefore not researched.

Show Picture - 231

This command is only used in the intro.

Description	Display a picture.
Parameters	[10] - **0**:picture priority number p `[integer]`, **1**:picture name s `[String]`, **2**:coordinate origin c `[integer 0-1]`, **3**:unused, **4**:relative position x `[integer]`, **5**:relative position y `[integer]`, **6**:horizontal zoom zx `[integer]`, **7**:vertical zoom yx `[integer]` **8**:opacity o `[integer 0-255]`, **9**:blending type b `[integer 0-2]`
Notes	c follows "common relation 11", b follows "common relation 12".
Representation	"Show Picture, s, priority=p, coordinates=(c.toString(), x, y), zoom=(zx, zy), opacity=o, blending=b.toString()"

Picture priority number p is used when multiple pictures are on display, because overlapping textures need to have an unambiguous drawing order.

Here, let there be pictures p_1, p_2 with priorities $2, 4$ respectively. Therefore, p_1 is drawn first, then p_2. The result is that, if they are overlapping, p_2 will be drawn **over** p_1, removing parts of p_1 from being displayed.

Typically, $x = y = 0$

Move Picture - 232

Parameters are mostly identical to "Show Picture". This is mostly used to animate intro's pictures (movement and opacity).

Description	Move a picture.
Parameters	[10] - **0**:picture priority number p `[integer]`, **1**:duration in frames f `[integer]`, **2**:coordinate origin c `[integer 0-1]`, **3**:unused, **4**:relative position x `[integer]`, **5**:relative position y `[integer]`, **6**:horizontal zoom zx `[integer]`, **7**:vertical zoom yx `[integer]` **8**:opacity o `[integer 0-255]`, **9**:blending type b `[integer 0-2]`
Notes	c follows "common relation 11", b follows "common relation 12".
Representation	"Move Picture, priority=p, coordinates=(c.toString(), x, y), zoom=(zx, zy), opacity=o, blending=b.toString()"

Appendix B

This appendix contains the proposed event command representation.

Command list

	Description	`Step` - Move the event (perform 1 step).
	Parameters	[1] - **0**: `direction - [String]`
	Notes	`direction` ∈ {S,W,E,N,SW,NW,NE,SE,R,1F,1B,1A,1T}, see "Directions" below.
	Examples	"`Step NW`", "`Step 1T`"
	Description	`Turn` - Turn the event (change direction).
	Parameters	[1] - **0**: `direction - [String]`
	Notes	`direction`, see "Directions" below.
	Examples	"`Turn N`", "`Turn W`"
	Description	`Move Event` - Move Event to absolute/relative coordinates on the same map.
	Parameters	[3] - **0**:`event* - [String/int]` (name/id of the event to move) **1**:`relative_coordinates/absolute_coordinates - [list of 2 int]` **2**:`direction* - [int]`
	Notes	`event` is optional, defaults to self. `direction` is optional, defaults to "K".
	Examples	"`Move Event relative_coordinates=[7,-5]`", "`Move Event event=Jack, absolute_coordinates=[4,12]`"
	Description	`Wait` - Pause event behavior execution for a given amount of time.
	Parameters	[1] - **0**: `ms/s - [int]` (time in milliseconds/seconds)
	Notes	If parameter name is unspecified, defaults to `s`.
	Examples	"`Wait ms=3000`", "`Wait s=3`"
	Description	`Set` - Set event properties value
	Parameters	[2] - **0**: `property - [String]`, **1**: `value - [int/String/:ON/OFF]`
	Notes	`property` must be a configuration variable, see "Configuration variables" below. `value` must be a valid value for that property.
	Examples	"`Set property=move_animation value=:ON`" "`Set property=Animation value=:OFF`" "`Set property=graphic value="trchar28,S,0"`"
	Description	`Play` - Play audio.
	Parameters	[3] - **0**: `SE/BGM/ME - [String]`, **1**: `volume* - [int]`, **2**: `pitch* - [int]`
	Notes	`volume` and `pitch` default to 100, their values are relative to 100 (percentage).
	Example	"`Play BGM="022-Field05", volume=100, pitch=100`"
	Description	`Show Text`
	Parameters	[1] - **0**: `text - [String]`
	Example	"`Show Text "Hello, World !"`"
	Description	`Choose` - Give player a list of items to choose from.
	Parameters	[2] - **0**: `choices - [list of String]`, **1**: `default* - [int]` n (behavior on cancel)
	Notes	If `default` not set, the player must choose (no cancel). Otherwise, select n^{th} item on the list.
	Examples	"`Choose choices=["Yes","No"]`", "`Choose choices=["One","Two","Three"], default=1`"
	Description	`Change Text Options`
	Parameters	[2] - **0**: `position* - [Top/Middle/Bottom]`, **1**: `border* - [Show/Hide]` (window border))
	Example	"`Change Text Options position=Middle, border=Show`"

Description	`End Execution` - Ends behavior execution.
Parameters	[0]
Example	"`End Execution`"
Description	`Label` - Marks a line as a target for a `Goto`.
Parameters	[1] - **0**: `name - [String]`
Notes	Please find a good name for the label (not like the example).
Example	"`Label "here"`"
Description	`Goto` - Change line to be executed next.
Parameters	[1] - **0**: `label - [String]`
Example	"`Goto "here"`"
Description	`Transfer Player` - Teleport player.
Parameters	[6] - **0**: `map* - [String]`, **1**: `x - [int]`, **2**: `y - [int]` **3**: `direction* - [String]`, **4**: `fading* - [:ON/:OFF]`
Notes	`map` defaults to the one the player is in. `direction` $\in$ {S,W,E,N,K}, defaults to "K". `fading` defaults to (TODO).
Example	"`Transfer Player map="Kurt's house", x=2, y=4`"
Description	`Set Move Route` - Set a sequence of commands, to be executed by a set event
Parameters	[1] - **0**: `event* - [String]`
Notes	`event` defaults to self. Must be followed by a *code block*. Used to move other events or to semantically indicate a "move sequence/route".
Example	See "Move Route" section below.
Description	`Screen Shake`
Parameters	[1] - **0**: `duration - [int]`
Notes	`duration` is expressed in milliseconds.
Example	"`Screen Shake 600`"
Description	`Transition` - Execute transition visual effect.
Parameters	[2] - **0**: `name - [String]`, **1**: `freeze - [True/False]`
Notes	If `freeze` is enabled, stops every animation.
Example	"`Transition name="battle1", freeze=True`"
Description	`Show Picture`
Parameters	TODO
Notes	TODO.
Example	TODO
Description	`Move Picture`
Parameters	TODO
Notes	TODO.
Example	TODO
Description	`Erase Picture`
Parameters	TODO
Notes	TODO.
Example	TODO

Description	`Set weather` - Set overworld's weather.
Parameters	[2] - **0**: `name - [String]`, **1**: `duration* - [int]`
Notes	`duration` defaults to infinite duration. The effect scope of `Set weather` is to be determined (for current map, radius on the current map, across maps). Provisionally, it's limited to current map.
Example	"`Set weather name="Rainy", duration=12000`"
Description	`Fade out BGM`
Parameters	[1] - **0**: `duration - [int]`
Notes	`duration` in milliseconds.
Example	"`Fade out BGM 3000`"
Description	`Memorize BGx`
Parameters	[0]
Example	"`Memorize BGx`"
Description	`Restore BGx`
Parameters	[0]
Example	"`Restore BGx`"
Description	`Restore All`
Parameters	[0]
Notes	Restore all stats for player's party.
Example	"`Restore All`"
Description	`Return to title screen`
Parameters	[0]
Notes	Quits current game and returns to title screen (without saving).
Example	`Return to title screen`
Description	`Save`
Parameters	[1] - **0**: `allow_cancel - [True/False]`
Notes	Prompts a "save your progress" dialog to the player.
Example	`Save allow_cancel=True`

Directions :

- S,W,E,N : South, West, East, North (vertical/horizontal movement)
- Step directions :
  - SW,NW,NE,SE : South-West, North-West, North-East, South-East (diagonal movement, not recommended)
  - R : random movement (S,W,E,N)
  - 1F,1B : one step Forwards/Backwards (according to current orientation/direction)
  - 1A,1T : one step Away from/Towards the player
- Turn directions:
  - "90 Right", "90 Left" : Turn 90 degrees right/left.
  - "random", "90 random" : Turn at random, turn "90 Right" or "90 Left" at random.
  - "towards player", "away from player" : Turn based on player's position.
- Transfer directions:
  - K : Keep the same (for teleportation)

Execution flow control :

- `if code_block [else code_block]?` : For implementing conditional execution of code blocks.
- `loop code_block` : `code_block` must contain a `break` statement for the loop to not be infinite. Infinite loop detection should be implemented.
- `choice ... [when (value) code_block]+` : For implementing behavior on player's choice.

Summary :

- Command number : 25 (+ 1*new*) + 3 forms of flow control vs. 81 commands !
- Documentation : ≈3 pages vs. ≈9 pages

Appendix C

This appendix contains the formal grammar for the proposed event command representation.

Formal grammar

```
EVENT
: LF* '[event]' LF+ (CONFIG LF)+ LF+ PAGE+ <EOF>

PAGE
: '[page]' LF+ (CONFIG_OR_COND LF)* LF+ STATEMENTS '[end]' LF+

CONFIG_OR_COND
: CONFIG_VAR '=' PARAMETER_VALUE
| LOG_EXPR

STATEMENTS              // block of lines that define an event's behavior
: (STATEMENT LF+)*

STATEMENT               // line that define an event's behavior
: 'if' LOG_EXPR LF CODE_BLOCK ('else' LF CODE_BLOCK)?
| 'loop' LF CODE_BLOCK
| 'when' WHITESPACE VALUE LF CODE_BLOCK
| 'break'
| CMD
| VAR_MANIPULATION
| SCRIPT

CODE_BLOCK              // block of lines whose execution is subject to flow control
: INDENT STATEMENTS DEDENT

CMD
: CMD_ID PARAMETERS?

PARAMETERS
: PARAMETER (',' WHITESPACE? PARAMETER)*

PARAMETER
: (PARAMETER_NAME '=')? PARAMETER_VALUE

PARAMETER_VALUE
: LIST
| VALUE
| BOOL

VAR_MANIPULATION
: SYMBOL ASSIGN_OPERATOR EXPRESSION

EXPRESSION              // expression that returns a value
: LOG_EXPR
| (MATH_OP | NUMBER | SYMBOL)+ // Imperfect : allows invalid expressions
| SCRIPT
| PARAMETER_VALUE

LOG_EXPR                // expression that returns a logical value
: COMPARABLE LOG_OPERATOR COMPARABLE
| SCRIPT

NUM_EXPR                // expression that returns a numerical value
: TERM (ADD_OP TERM)*
```

```
PARAMETER_NAME
: WORD

LIST
: '[' VALUE (WHITESPACE VALUE)* ']'

SCRIPT
: s\:.*

TERM
: FACTOR (MUL_OP FACTOR)*

FACTOR
: NUMBER
| '(' NUM_EXPR ')'

CMD_ID
: WORDS

SYMBOL
: ':' WORD

VALUE
: NUMBER
| STRING
| WORD
| TURN_VALUE

STRING
: '"' [^"]*  '"'

NUMBER
: -?[0-9]+('.'[0-9]+)?

LF
: '\r\n' | '\n'

WORDS
: WORD+

WORD
: [a-z][a-z\_0-9]*

BOOL
: ':ON' | ':OFF' | 'True' | 'False'

LOG_OPERATOR
: '==' | '>=' | '<=' | '>' | '<' | '!='

ASSIGN_OPERATOR
: '=' | '+=' | '-=' | '*=' | '/='

MATH_OP
: '+' | '-' | '*' | '/'

COMMENT                // Rejected by the lexer
: #.*

SPACE                  // Rejected by the lexer
: [ \t]+
```

Notes :

- `CMD_ID` is expected to be one of the defined operation. An error should be thrown otherwise.
- Comments must be stripped before lexing. Multi-line comments aren't supported.
- Tokens (terminal values) `INDENT` and `DEDENT` should be generated when reading the event file in order to represent indentation, thus allowing for block of statements to be syntactically represented.

Appendix D

This appendix contains images referenced in the body of the report.