About PoGER

David Rodriguez Soares

August 24, 2020

Contents

1 Introduction

This document proposes a high-level view of the PoGER project. For more detailed information, please read the other accompanying documents.

It is aimed at readers unfamiliar of this project and interested in its contents.

1.1 Related documents

This is but one in a collection of manuscripts, each with its scope (*sorted older to newer*) :

- **Vision document** : A very early analysis about Pokemon Essentials, focused on identifying its issues and properties of a hypothetical software solution.
- **PoGER** : Gives information on the project itself, including motivations and objectives.
- **RMXP doc** : Contains the results of my research about RPG Maker XP's implementation of game data (mostly *Maps* and *Events*), with the objective to extract them to an interpretable format.
- **Extraction** : Contain a practical guide on the extraction process for anyone trying to reproduce it.

1.2 What PoGER is

PoGER stands for **Po**kemon Essentials **G**ame **E**ngine **R**ecreation. It's a research and implementation project that looks into the feasability of running Pokemon Essentials (and related games) independently of RPG Maker XP, its native platform and interpreter.

2 Research stage

2.1 Motivation

Pokemon Essentials and derived games run on RPG Maker XP's interpreter, which is the main limiting factor.

For more details, read `VisionDocument.pdf`.

Here is a non-exhaustive list of reasons why a game engine recreation attempt would be justified :

- RPG Maker XP is bound to Windows, which means that any game for it is a Windows exclusive.
- Poor performance : Pokemon Essentials runs on an antiquated engine and performs poorly even on modern hardware.
- RPG Maker XP's engine is basically deprecated at this point.
- Coding complexity : Making a game based on Pokemon Essentials requires being fluent in RPGXP-specific Ruby.
- Using RPG Maker XP to edit the game is mandatory : A lot of data/assets are compiled and can only be interpreted by it.
- Un-optimized for collaboration : Keeps projects from having more than 1 or 2 developers
- Semi-closed source approach : Allows projects to die by making it hard for anyone to pick up an abandoned project.
- General lack of multi-player and online features.

2.2 Findings

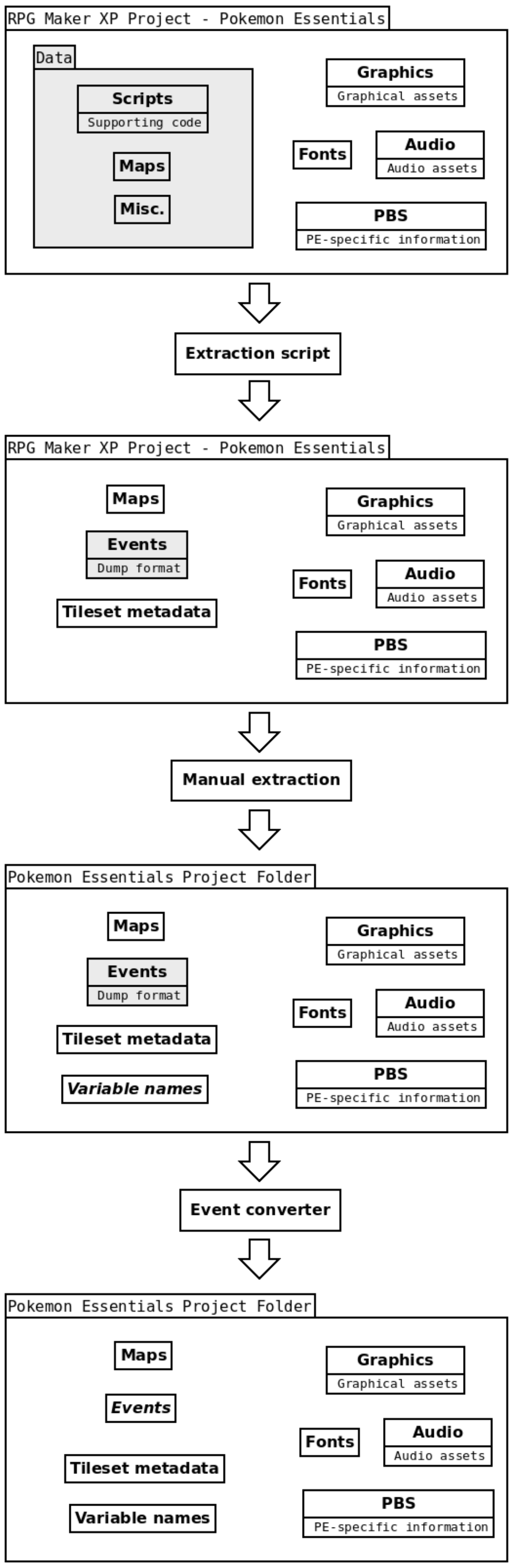
For more findings and details, read `PoGER.pdf` and `RMXP doc.pdf`.

- It's possible to retrieve compiled data from a RGSS script. This is useful for extracting **Maps**, **Events** and **Tileset metadata**
- Because of Autotiles (a RPG Maker series feature), it's probably not possible to export maps to an existing standard map format.
- Variable/Switch names must be obtained manually.
- In order to make events humanly readable and keep their functionality, they need to be converted to a domain-specific language.
- For a successfully port a game to a "new engine" to be functional, it would be necessary to integrate any script called from any event to it.
- A lot of "common events" (doors, boulders, etc) have complex behavior and can be greatly simplified by integrating it into the engine.

2.3 A note on PBS files

Some Pokemon Essentials-derived projects have releases without PBS files. However, it should be possible to extract them using a script. Said script probably exists within that project's code, but if it isn't, it's likely another project's extraction script would work.

3 Data extraction process



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.

You can find detailed information about this process in `Extraction.pdf`.

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** (documented in `RMXP doc.png`) and packaged into individual files and easily editable.

Final result

At the end of the extraction process, all assets and data used by the game exist in a RMXP-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 Implementations

4.1 RPG Maker XP extraction script

This extracts data from compiled files in the `Data` directory.

It can be found at `RMXP_research/Extraction script/Extraction.rb`. Please read `Extraction.pdf` for usage instructions.

In the same directory, you can find old versions of the same script and an attempt at a PBS-extraction script I wrote before realizing I was reinventing the wheel.

It was tested on both **Pokemon Essentials** and **Pokemon Uranium**.

4.2 Event converter

This converts events from their "JSON dump" representation to a human-readable format.

It can be found at `RMXP_research/Event converter/Event_reader.py`.

Features :

- Extracts events into individual files.
- Translates events to domain-specific language.
- Lists scripts called by events for ease of implementation.

Note : `RGSS_Command_conversion.py` contains a translation function for every event command and `PE_variables_switches.py` contains dictionaries for variable/switch names.

4.3 Map reader

This reads an map from its own file and its tileset/autotiles assets files and reconstructs it.

It can be found at `RMXP_research/Autotile research/LoadMap.py`.

Features :

- Can process multiple maps from a list.
- Implements a map reader class.
- Can decode both **Tilesets** and **Autotiles**.
- Has the ability to output **static** and **animated** maps.
- Useless beta feature : Reducing tilesets according to the specific needs of a particular map.

4.4 Interpreter

This executes an event from its file.

It can be found at `RMXP_research/Interpreter/eventReader.py`.

Other files :

- `Interpreter` : Interprets events DSL
- `pbInterpreter` : Interprets events script calls.

4.5 PBS reader

This loads data from PBS files and stores it for further use by game's logic.

It can be found at `RMXP_research/PBSreader/`.

4.6 Demo

TODO

5 Remarks

5.1 Contact

Contact the author by email : David.Rodriguez.1@etu.unige.ch