Byte Compare v0.1b

released 08.07.08
©www.prawnstarproductions.com

Contents

About	1
Disclaimer	1
Goal	2
What Byte Compare can do	3
What Byte Compare can do that is (probably!) unique	3
What Byte Compare cannot do	4
How it works	5
Instructions	5
Settings	6
Limitations	7
Contact	7
Thanks	7
Version History	7
Appendices	8

About

Byte Compare is a tool for comparing any two files for likely patterns in their byte structure.

The application uses (what I understand to be) a unique approach to compare two files and return a percentage indicating how much content is likely to be shared between the files (see: ‘How it works’).

It has been developed primarily as a tool to help organise the many forgotten games and applications that have emerged online (the preservation of which is championed through organisations such as GoodTools and TOSEC). It also has obvious applications as a plagiarism detection tool.

Byte Compare is freeware and will eventually be released as open source.

Disclaimer

You use Byte Compare and at your own risk. By using this application you agree that the program author is in no way responsible for any damage which may occur.

Goal

At the present time, the process of identifying and categorising the many thousands of files for past consoles and computers is done by hand. This is never likely to change. Nonetheless, Byte Compare can evolve to serve researchers as:

1. A **detailed reporting** tool. Future versions of Byte Compare will be able to scan a folder and output the findings to an xml file. The file will group similar games into one family. Obvious matches – such as regional variations – will be shown, as well as a separate optional group showing instances where just a few percent are shared. Finding possible links to other games may prompt researchers to review the make-up of that game in more detail, and lead to a better understanding of its construction. Re-use of logo animations, graphic libraries, and mini-games may prove be amongst the explanations for such 'micro-matches'.

2. A tool to automate **deep reporting**. Scan files across other folders. Compare them regardless of file type, intended system, and create game families in the same way as you can with a single folder scan. Testing has been limited to sample files with the TOSEC ZX Spectrum sets: TAP, Z80, TZX (please see 'What Byte Compare cannot do that is (probably!) unique' below). Consolidating sets in this way is a major goal of this project, as it helps to develop more accurate lists of unique games (initial indications are that this consolidation will not always be possible. Please see 'What Byte Compare cannot do' below).

3. A **confirmation** tool. It will flag similar files which may have been missed during the manual checks, alerting the user that they may wish to add a unique game to a particular family (as with GoodMerge, or the MAME parent/clone relationship). See Appendix A.

4. A tool for **flexible classification**. In future versions of Byte Compare, multiple instances of game compilations will be able to exist. Most organisation tools presently cite game compilations as unique games, while this app will allow the user to add them as clones to each original 'component' game. For space saving tools (such as GoodMerge), of course, this facility should not be used.

5. A **research** tool. This possibility is a little bit more optimistic. Old games which do not feature a publisher, year, or even title during runtime can be difficult to identify (they may be homebrew, demo, beta, etc.). This tool *may* help to identify some of the numerous 'ZZZ-UNK-' prefixed files in TOSEC sets, by finding similarities with games already well documented. This would work by flagging blocks of code which could have been reused from other games.

It is hoped that Byte Compare's settings profiles can be tweaked enough to offer a reliable scanning configuration which may eventually assist in creating TOSECMerged lists. Whether this process can be fully automated with an acceptable error rate is as yet unknown.

Cross-folder scan lists such as these may be useful with frontends such as GameEx, which features an 'Emulator Group' facility, but no way of dynamically merging respective emulators' games into families. This also relates to the pursuit of identifying unique games, discussed above.

What Byte Compare can do

1. Successfully match obvious files to one another, such as the following in the SNES GoodMerge set: Super Mario World.7z :

```
Super Mario World - Super Mario Bros. 4 (J) [!].smc
Super Mario World (E) (V1.0).smc
Super Mario World (U) [!].smc
```

What Byte Compare can do that is (probably!) unique

1. Byte Compare can match files of quite **different sizes**. The following files are from GoodSNES (using default settings as of v0.1b):

```
Super Mario World (E) (V1.0).smc
Super Mario All-Stars + Super Mario World (U) [!].smc
Highest match: 17 percent of smallest file
```

In this example, we can see that even though the region, file size, and probably some encoding and padding patterns are different between the files, a high match rate was returned. This scan took 1 second to perform.

2. Match files where similar **blocks of bytes exist but are in a different order**. The following files are taken from the ZX Spectrum TOSEC sets. The files contain similar information, but some byte chunks (blocks) have been moved around. There is also a large amount of padding in the larger file. This is a difficult scan for most tools, and a byte-for-byte scanner with even a very high tolerance will struggle to match one percent.

```
Jet Set Willy (1984)(Software Projects).tap
Jet Set Willy (1984)(Software Projects).z80
Highest match: 45 percent of smallest file
```

Byte Compare managed to match this very well, though a bespoke adjustment was necessary to reveal the extent of the similarity (using default settings as of v0.1b, except for Padding Tolerance: 50%).

3. Tolerate minor differences when comparing blocks (setting: Mismatch Tolerance). Blocks are not automatically discounted because they are not identical (i.e. because of MD5 or SHA1 discrepancies). A user-defined percentage-per-block is allowed to mismatch.

What Byte Compare cannot do

1. Deal with **encoding / bespoke compression**. Unfortunately, the following files are not matched even though they play almost identically (from GoodNES and The TOSEC Nintendo FDS set respectively):

```
Metroid (U)(PRG0) [!].nes
Metroid (v1.01).fds
```

The fds file seems to have been compressed differently to the nes one, meaning that there are no clear blocks of bytes which match one another.

* *Is compression / re-encoding the reason why Super Mario bros. 3 (GoodNES) and Super Mario All-Stars (GoodSNES) do not match up? Might one otherwise expect the underlying routines to be shared between files?*

How it works

1. Each file is split into a predetermined number of parts (referred to as blocks).
2. Each block is then analysed byte by byte. An integer value exists in the program for each of 256 possible bytes for each block. The corresponding integer is incremented by one as each byte is revealed.
3. Once all of the bytes have been analysed and matched to corresponding values, the same process takes place for the next block, and the next, until all have been processed.
4. Each block's 256 integer values are then compared one by one to the equivalent value in the other block to see if they are of similar values (if they are, the blocks are tagged as matching).
5. Once all viable blocks have been compared to one another, a matching percentage is returned to the user.

Instructions

1. Unzip the ByteCompare.zip file and extract all files and folders to a readable directory on your local disk.
2. Run the ByteScan.exe.
3. When the application is running, choose whether you wish to compare two individual files or all files in a folder.

Profiles

Profiles save settings so that you do not have to re-enter them manually each time you run the program or scan another set. Try to keep a consistent naming convention. You may wish to create a distinct profile for SNES, N64, Amiga, etc. Byte Compare will hopefully feature user submitted profiles in future releases.

Save Output

Scan files can be saved once a scan is completed. They are currently saved as text files while other format types are considered. These files detail the findings of every file compared.

Log Files can also be saved post-scan. These detail only the files that were matched and show the percentage matched and percentage of parts ignored (usually because they are mostly padding). Also likely to change from the current txt file format.

Settings

The following settings are found on the Byte Compare user interface.

Number of Blocks (Blocks). Default 100. This is the preferred number of parts you would like to break the smallest of the two files you are scanning in to. If your smallest file is 10,000 bytes in size, for example, and you set the value to 100, each file will be broken into blocks of 100 bytes.

Minimum Block Size (Bytes). Default 128. This setting will prevent files being split into blocks that are too small for a reliable scan to be performed. If they are smaller than the selected value, the Number of Blocks value will be adjusted to accommodate this restriction.

Mismatch Tolerance (% of Block). Default 10. If the total accumulative discrepancy of the 256 integer values for a block varies by more than x% to the block it is being compared to, the blocks are considered too different to be similar, and a mismatch is returned.

Padding Tolerance (% of Block). Defaults to 20. If more than x% of a block consists of a byte value in an uninterrupted sequence, the scan of the two blocks is abandoned. This is done on the basis that one block is likely made up largely of padding. This is a streak of uninterrupted consecutive bytes, and applies to any value (not just FF or 00).

Number of Checks. Default 3. Dictates the starting point from which the first byte of the smallest file will be compared against the first byte of the bigger file. If a block is 600 bytes in size for example, and the value is 3, the program will work out the starting positions using this formula:

```
total block bytes / (No.of Checks+1)
```

The value 1 is added to the above equation to ensure we do not bleed into the second block. If this is not clear please email me and I'll explain better (and amend this documentation). Hopefully Fig1 will help to clarify:

Fig1.

Block Size = 600 Header 1 Offset = 0 Header 2 Offset = 0	Check	Smallest file start position (bytes from start)	Largest File start position (bytes from start)
	1	0	0
	2	200	0
	3	400	0

Max Block Increase. Not yet implemented – will help to manage scans between very big and very small files.

Header 1 & 2 Offset (Bytes from start of file). Default 0. Gives you the option to ignore the first x bytes of a file before scanning. Useful for when you know that header values will vary, like they do with NES ROMs.

Limitations

Limitations which will likely persist even in future versions:

- Scans probably won't be effective if files contain the same blocks of information, but they are encoded in different ways (see: the example under 'What Byte Compare cannot do').

Limitations which are on the To-do list and should be fixable:

- **Scan speed.** Speeds will be much faster next release. Priority #1
- Unzip / Unrar support.
- ~~Bulk scan by comparing all files in a given folder.~~
- Scan multiple folders at a time.
- Group similar files into 'families' in an XML file.
- Overall Progress Bar.
- Won't detect blocks of padding if filler bytes alternate: e.g. FF E0 FF E0 FF E0.
- Exclude certain files (usually BIOS), e.g. diskSys.rom for TOSEC FDS set.
- Intelligent exclusion for BIOS / misfit files (user prompts at scan start)?
- Read split MAME ROM sets as whole and scan against other files.
- Comment up, tidy up, and release as open source.
- Identify image size limits for files.
- Make the interface sexier.
- Very slow with CD sized images and may even crash with DVD sized images.
- Does not yet have fail-over support for very small files.

Contact

sean@prawnstarproductions.com

www.prawnstarproductions.com

Thanks

Thanks to polarz at the Good Tools forum for coming up the idea that led me to do this.

Thanks also to all the ROM and disk image organising groups: Cowering, TOSEC, MAME, No-Intro, Roman, GoodMerge team old and new, and many more...

Version History

0.1b Initial release.

Appendices

Unless stated otherwise, all results were obtained using default settings as of v0.1b.

Appendix A

The following files are merged into two distinct sets in GoodMerge NES (as these were released as separately marketed games, this may be intentional. The example is merely being used to demonstrate the independence of the application from some other categorisation methodologies).

```
Tiny Toon Adventures (U) [!].nes
Super Mario Bros 6 (Unl) [!].nes
```
Highest Match: 69 percent of smallest file

Appendix B

The following files are merged into two distinct sets in GoodMerge SNES. Interestingly, MAME categorises the two game titles into one family, and the high match percentage here would indicate that relationship should be shared for the SNES conversions too.

```
Ultimate Mortal Kombat 3 (U).smc
Mortal Kombat 3 (U) [!].smc
```
Highest match: 19 percent