Corentin Jabot (corentinjabot@gmail.com)

P2675R0

Date: 2022-10-12

LWG3780: The Paper

format's width estimation is too approximate and not forward compatible

Abstract

LWG3780 describes an issue with width in std::format estimation. This paper offers more information.

For the purpose of width estimation, format considers ranges of codepoints initially derived from an implementation of wcwidth with modifications (see P1868R1).

This however present a number of challenges:

- From a reading of the spec, it is not clear how these ranges were selected.
- Poor forward compatibility with future Unicode versions. The list will become less and less meaningful overtime or require active maintenance at each Unicode release (which we have not done for Unicode 14 already).
- Some of these codepoints are unassigned or otherwise reserved, which is another forward compatibility concern.

Instead, we propose to

- Rely on UAX-11 for most of the codepoints)
- Grand-father specific and fully assigned, blocks of codepoints to support additional pictograms per the original intent of the paper and existing practices. We add the name of these blocks in the wording for clarity.

Note that per UAX-11

- Most emojis are considered East_Asian_Width="W"
- By design, East_Asian_Width="W" includes specific unassigned ranges, which should always be treated as Wide.

This change:

- Considers 8477 extra codepoints as having a width 2 (as of Unicode 15) (mostly Tangut Ideographs)
- Change the width of 85 unassigned code points from 2 to 1

- Change the width of 8 codepoints (in the range U+3248 CIRCLED NUMBER TEN ON BLACK SQUARE … U+324F CIRCLED NUMBER EIGHTY ON BLACK SQUARE) from 2 to 1, because it seems questionable to make an exception for those without input from Unicode

Additional Observations following Mailing List discussions

This paper is not trying to change, in any way, the definition of double width characters for the purpose of padding in fmt, but only to make it forward compatible with future unicode versions.

Unicode considers width only for the purpose of east asian character and does not necessarily encourage using it to estimate width. Indeed that is highly dependent on fonts.

> Note: The East_Asian_Width property is not intended for use by modern terminal emulators without appropriate tailoring on a case-by-case basis. Such terminal emulators need a way to resolve the halfwidth/fullwidth dichotomy that is necessary for such environments, but the East_Asian_Width property does not provide an off-the-shelf solution for all situations. The growing repertoire of the Unicode Standard has long exceeded the bounds of East Asian legacy character encodings, and terminal emulations often need to be customized to support edge cases and for changes in typographical behavior over time.
>
> https://www.unicode.org/reports/tr11/#Scope

However we adopted P1868R1 in C++20, to do just that. It was always a partial solution, which is better than nothing. No perfect solution exists.

P1868 was based on the grandfather of all unicode wcwidth implementations, and so it is interesting to look at its comments (emphasis mine)

In fixed-width output devices, Latin characters all occupy a single
"cell" position of equal width, whereas ideographic CJK characters
occupy two such cells. Interoperability between terminal-line
applications and (teletype-style) character terminals using the
UTF-8 encoding requires agreement on which character should advance
the cursor by how many cell positions. **No established formal
standards exist at present on which Unicode character shall occupy
how many cell positions on character terminals. These routines are
a first attempt of defining such behavior based on simple rules**

applied to data provided by the Unicode Consortium.
For some graphical characters, the Unicode standard explicitly defines a character-cell width via the definition of the East Asian FullWidth (F), Wide (W), Half-width (H), and Narrow (Na) classes. In all these cases, there is no ambiguity about which width a terminal shall use. For characters in the East Asian Ambiguous (A) class, the width choice depends purely on a preference of backward compatibility with either historic CJK or Western practice. Choosing single-width for these characters is easy to justify as the appropriate long-term solution, as the CJK practice of displaying these characters as double-width comes from historic implementation simplicity (8-bit encoded characters were displayed single-width and 16-bit ones double-width, even for Greek, Cyrillic, etc.) and not any typographic considerations.

Much less clear is the choice of width for the Not East Asian (Neutral) class. **Existing practice does not dictate a width for any of these characters.** It would nevertheless make sense typographically to allocate two character cells to characters such as for instance EM SPACE or VOLUME INTEGRAL, which cannot be represented adequately with a single-width glyph. **The following routines at present merely assign a single-cell width to all neutral characters, in the interest of simplicity.** This is not entirely satisfactory and should be reconsidered before establishing a formal standard in this area. **At the moment, the decision which Not East Asian (Neutral) characters should be represented by double-width glyphs cannot yet be answered by applying a simple rule from the Unicode database content.** Setting up a proper standard for the behavior of UTF-8 character terminals will require a careful analysis not only of each Unicode character, but also of each presentation form, something the author of these routines has avoided to do so far.

```
 http://www.unicode.org/unicode/reports/tr11/


Markus Kuhn -- 2007-05-26 (Unicode 5.0)
```

So, this script was written against Unicode 5.0. And admits it is doing its best in the absence of a standard.

```
*    - The null character (U+0000) has a column width of 0.
*    - Other C0/C1 control characters and DEL will lead to a return
*      value of -1.
*
*    - Non-spacing and enclosing combining characters (general
*      category code Mn or Me in the Unicode database) have a
*      column width of 0.
*
*    - SOFT HYPHEN (U+00AD) has a column width of 1.
*
*    - Other format characters (general category code Cf in the Unicode
*      database) and ZERO WIDTH SPACE (U+200B) have a column width of 0.
*
*    - Hangul Jamo medial vowels and final consonants (U+1160-U+11FF)
*      have a column width of 0.
*
*    - Spacing characters in the East Asian Wide (W) or East Asian
*      Full-width (F) category as defined in Unicode Technical
*      Report #11 have a column width of 2.
*
*    - All remaining characters (including all printable
*      ISO 8859-1 and WGL4 characters, Unicode control characters,
*      etc.) have a column width of 1.
```

It considered East Asian Wide (W) or East Asian Full-width (F) codepoints to be 2, and the rest 1 or 0.

We decided not to consider control characters in P1868R1, and because we only consider the leading codepoints in graphemes, we do not need to care about combining marks (ditto for Jamo medial and final). Note that lone combining marks usually have a width of 1, so the standard is more correct to consider leading grapheme than trying to pretend combining marks have no width.

Either way, this script does not pretend to be a standard, nor was it ever updated in the last 13 years.

It has, however, been forked. Notably, the ruby port is compatible with Unicode 15 https://github.com/janlelis/unicode-display_width

(Other projects fo Python and go seem to have fallen behind in their unicode support)

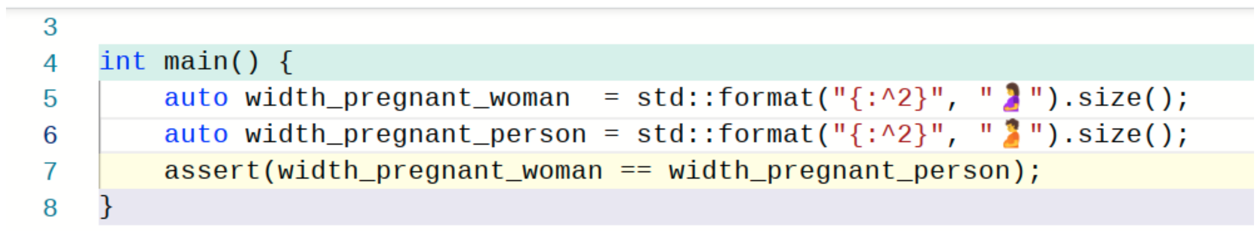
I think the comments above are illustrative of the state of the art and existing practices:

East Asian Wide (W) or East Asian Full-width (F) codepoints have a width of 2, a half width, or, for anything else,...we really don't know. This is covered extensively in https://www.unicode.org/reports/tr11/

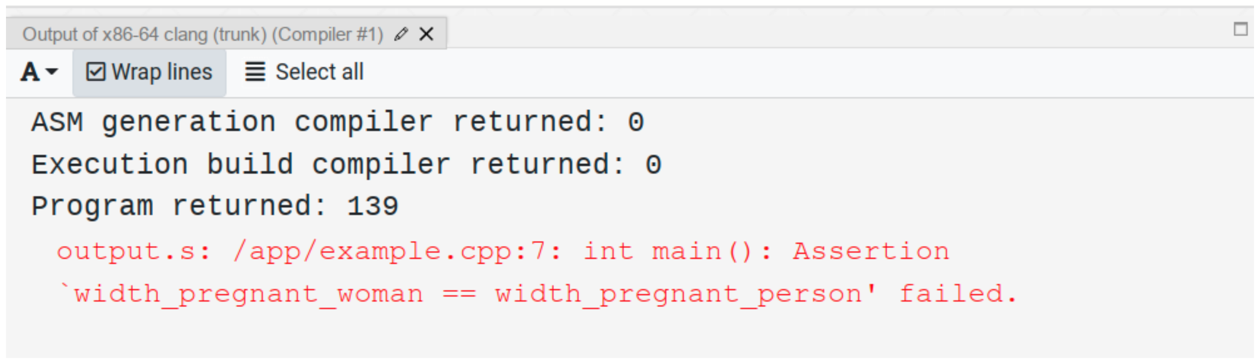
So there is no perfect solution.

Yet, the standard's definition leaves a lot to be desired:

- It does not consider all W and F codepoints as being of width 2. This lead to the following nonsense:



```
3
4  int main() {
5      auto width_pregnant_woman  = std::format("{:^2}", "🤰").size();
6      auto width_pregnant_person = std::format("{:^2}", "🫄").size();
7      assert(width_pregnant_woman == width_pregnant_person);
8  }
```



```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 139
  output.s: /app/example.cpp:7: int main(): Assertion
  `width_pregnant_woman == width_pregnant_person' failed.
```

Another example: https://godbolt.org/z/McoG64n1v

- It specifies ranges of wide characters in terms of non fully allocated ranges. This goes against TR11 which specifies that only

Unassigned code points in ranges intended for CJK ideographs are classified as Wide. Those ranges are:

the CJK Unified Ideographs block, 4E00..9FFF

the CJK Unified Ideographs Extension A block, 3400..4DBFthe CJK Compatibility Ideographs block, F900..FAFF

the Supplementary Ideographic Plane, 20000..2FFFF

the Tertiary Ideographic Plane, 30000..3FFFF

All other unassigned code points are by default classified as Neutral.

The standard should generally be *very* careful not to specify arbitrary range of codepoints containing unassigned code points or ranges of blocks that are not fully allocated as this introduces forward compatibility issues when new codepoints are assigned. It's the exact issues which created forward compatibility and portability issues with identifiers in C++11.

C++ should rely on Unicode properties which have default values.

It is also very hard to make sense of the current specification as the ranges are not motivated, and were derived from an unmaintained script.

Proposed solution

- Specify display width in terms of Unicode properties such that the behavior remain over time
- Keep certain ranges for compatibility with C++20, and because some pictograms that have emoji presentation have an East Asian Width of 1. These blocks are fully allocated.

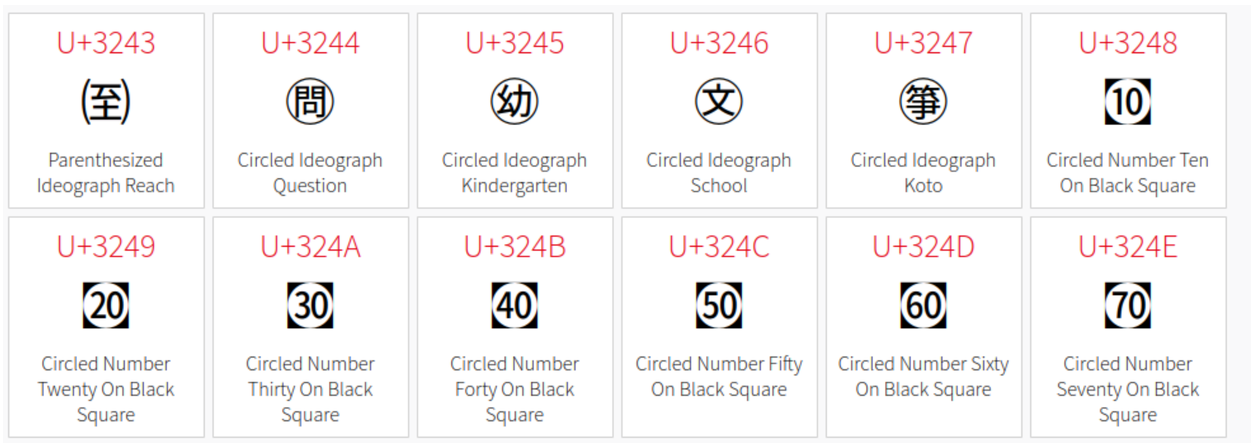
But what about my terminal/font?

The annex shows the rendering of the impacted characters in different environments (Thanks #include C++ and Twitter). Even when the fonts are not present (not surprising given a lot of impacted codepoints are from Unicode 14 and 15), 2 columns are used in most environments. And if terminals behaved poorly, I really don't think the C++ standard is the place to look at that problem.

Open Question:

Do we want to consider wide U+3248 CIRCLED NUMBER TEN ON BLACK SQUARE .. U+324F CIRCLED NUMBER EIGHTY ON BLACK SQUARE which are not East Asian Width = "W" but are currently considered wide in C++, knowing their glyphs are usually wide?

| | | | | | |



U+3243	U+3244	U+3245	U+3246	U+3247	U+3248
㉃	㉄	㉅	㉆	㉇	㉈
Parenthesized Ideograph Reach	Circled Ideograph Question	Circled Ideograph Kindergarten	Circled Ideograph School	Circled Ideograph Koto	Circled Number Ten On Black Square
U+3249	U+324A	U+324B	U+324C	U+324D	U+324E
㉉	㉊	㉋	㉌	㉍	㉎
Circled Number Twenty On Black Square	Circled Number Thirty On Black Square	Circled Number Forty On Black Square	Circled Number Fifty On Black Square	Circled Number Sixty On Black Square	Circled Number Seventy On Black Square

I really do not care either way, even if I think we should avoid legislating specific codepoints.

Future Works

We could

- Reconsidering the width of control characters
- Convince the Unicode consortium to offer a better solution to this problem

However these are outside of the intent of the original feature which focused on east asian width.

Wording:

This wording is relative to N4917.

1. Modify 22.14.2.2 [format.string.std] as indicated:
   -12- For a string in a Unicode encoding, implementations should estimate the width of a string as the sum of estimated widths of the first code points in its extended grapheme clusters. The extended grapheme clusters of a string are defined by UAX #29. The estimated width of the following code points is 2:
   (12.1) — U+1100 – U+115F
   (12.2) — U+2329 – U+232A
   (12.3) — U+2E80 – U+303E
   (12.4) — U+3040 – U+A4CF
   (12.5) — U+AC00 – U+D7A3
   (12.6) — U+F900 – U+FAFF
   (12.7) — U+FE10 – U+FE19
   (12.8) — U+FE30 – U+FE6F
   (12.9) — U+FF00 – U+FF60
   (12.10) — U+FFE0 – U+FFE6
   (12.11) — U+1F300 – U+1F64F
   (12.12) — U+1F900 – U+1F9FF
   (12.13) — U+20000 – U+2FFFD
   (12.14) — U+30000 – U+3FFFD
   (?.1) — Any code point with the East_Asian_Width="W" or East_Asian_Width="F" Derived Extracted Property as described by UAX #44
   (?.2) — U+4DC0 – U+4DFF (Yijing Hexagram Symbols)
   (?.3) — U+1F300 – U+1F5FF (Miscellaneous Symbols and Pictographs)
   (?.4) — U+1F900 – U+1F9FF (Supplemental Symbols and Pictographs)
2. The estimated width of other code points is 1.

References

Unicode® Standard Annex #11 EAST ASIAN WIDTH

P1868R1 🦄 width: clarifying units of width and precision in std::format

https://github.com/janlelis/unicode-display_width

https://github.com/jquast/wcwidth

https://www.cl.cam.ac.uk/~mgk25/ucs/wcwidth.c

Acknowledgments

Thanks to the many people who contributed screenshots!

Annex: Exhaustive list of changes

For the following code points, the estimated width used to be 1, and is 2 after the suggested change:

- U+231A WATCH .. U+231B HOURGLASS
- U+23E9 BLACK RIGHT-POINTING DOUBLE TRIANGLE .. U+23EC BLACK DOWN-POINTING DOUBLE TRIANGLE
- U+23F0 ALARM CLOCK
- U+23F3 HOURGLASS WITH FLOWING SAND
- U+25FD WHITE MEDIUM SMALL SQUARE .. U+25FE BLACK MEDIUM SMALL SQUARE
- U+2614 UMBRELLA WITH RAIN DROPS .. U+2615 HOT BEVERAGE
- U+2648 ARIES .. U+2653 PISCES
- U+267F WHEELCHAIR SYMBOL
- U+2693 ANCHOR
- U+26A1 HIGH VOLTAGE SIGN
- U+26AA MEDIUM WHITE CIRCLE .. U+26AB MEDIUM BLACK CIRCLE
- U+26BD SOCCER BALL .. U+26BE BASEBALL
- U+26C4 SNOWMAN WITHOUT SNOW .. U+26C5 SUN BEHIND CLOUD
- U+26CE OPHIUCHUS
- U+26D4 NO ENTRY
- U+26EA CHURCH
- U+26F2 FOUNTAIN .. U+26F3 FLAG IN HOLE
- U+26F5 SAILBOAT
- U+26FA TENT
- U+26FD FUEL PUMP
- U+2705 WHITE HEAVY CHECK MARK
- U+270A RAISED FIST .. U+270B RAISED HAND
- U+2728 SPARKLES
- U+274C CROSS MARK
- U+274E NEGATIVE SQUARED CROSS MARK

- U+2753 BLACK QUESTION MARK ORNAMENT .. U+2755 WHITE EXCLAMATION MARK ORNAMENT
- U+2757 HEAVY EXCLAMATION MARK SYMBOL
- U+2795 HEAVY PLUS SIGN .. U+2797 HEAVY DIVISION SIGN
- U+27B0 CURLY LOOP
- U+27BF DOUBLE CURLY LOOP
- U+2B1B BLACK LARGE SQUARE .. U+2B1C WHITE LARGE SQUARE
- U+2B50 WHITE MEDIUM STAR
- U+2B55 HEAVY LARGE CIRCLE
- U+A960 HANGUL CHOSEONG TIKEUT-MIEUM .. U+A97C HANGUL CHOSEONG SSANGYEORINHIEUH
- U+16FE0 TANGUT ITERATION MARK .. U+16FE4 KHITAN SMALL SCRIPT FILLER
- U+16FF0 VIETNAMESE ALTERNATE READING MARK CA .. U+16FF1 VIETNAMESE ALTERNATE READING MARK NHAY
- **U+17000 TANGUT IDEOGRAPH-# .. U+187F7 TANGUT IDEOGRAPH-#**
- **U+18800 TANGUT COMPONENT-001 .. U+18CD5 KHITAN SMALL SCRIPT CHARACTER-#**
- **U+18D00 TANGUT IDEOGRAPH-# .. U+18D08 TANGUT IDEOGRAPH-#**
- U+1AFF0 KATAKANA LETTER MINNAN TONE-2 .. U+1AFF3 KATAKANA LETTER MINNAN TONE-5
- U+1AFF5 KATAKANA LETTER MINNAN TONE-7 .. U+1AFFB KATAKANA LETTER MINNAN NASALIZED TONE-5
- U+1AFFD KATAKANA LETTER MINNAN NASALIZED TONE-7 .. U+1AFFE KATAKANA LETTER MINNAN NASALIZED TONE-8
- U+1B000 KATAKANA LETTER ARCHAIC E .. U+1B122 KATAKANA LETTER ARCHAIC WU
- U+1B132 HIRAGANA LETTER SMALL KO
- U+1B150 HIRAGANA LETTER SMALL WI .. U+1B152 HIRAGANA LETTER SMALL WO
- U+1B155 KATAKANA LETTER SMALL KO
- U+1B164 KATAKANA LETTER SMALL WI .. U+1B167 KATAKANA LETTER SMALL N
- U+1B170 NUSHU CHARACTER-# .. U+1B2FB NUSHU CHARACTER-#
- U+1F004 MAHJONG TILE RED DRAGON
- U+1F0CF PLAYING CARD BLACK JOKER
- U+1F18E NEGATIVE SQUARED AB
- U+1F191 SQUARED CL .. U+1F19A SQUARED VS
- U+1F200 SQUARE HIRAGANA HOKA .. U+1F202 SQUARED KATAKANA SA
- U+1F210 SQUARED CJK UNIFIED IDEOGRAPH-624B .. U+1F23B SQUARED CJK UNIFIED IDEOGRAPH-914D
- U+1F240 TORTOISE SHELL BRACKETED CJK UNIFIED IDEOGRAPH-672C .. U+1F248 TORTOISE SHELL BRACKETED CJK UNIFIED IDEOGRAPH-6557
- U+1F250 CIRCLED IDEOGRAPH ADVANTAGE .. U+1F251 CIRCLED IDEOGRAPH ACCEPT
- U+1F260 ROUNDED SYMBOL FOR FU .. U+1F265 ROUNDED SYMBOL FOR CAI
- U+1F680 ROCKET .. U+1F6C5 LEFT LUGGAGE
- U+1F6CC SLEEPING ACCOMMODATION
- U+1F6D0 PLACE OF WORSHIP .. U+1F6D2 SHOPPING TROLLEY
- U+1F6D5 HINDU TEMPLE .. U+1F6D7 ELEVATOR
- U+1F6DC WIRELESS .. U+1F6DF RING BUOY
- U+1F6EB AIRPLANE DEPARTURE .. U+1F6EC AIRPLANE ARRIVING
- U+1F6F4 SCOOTER .. U+1F6FC ROLLER SKATE
- U+1F7E0 LARGE ORANGE CIRCLE .. U+1F7EB LARGE BROWN SQUARE
- U+1F7F0 HEAVY EQUALS SIGN

- U+1FA70 BALLET SHOES .. U+1FA7C CRUTCH
- U+1FA80 YO-YO .. U+1FA88 FLUTE
- U+1FA90 RINGED PLANET .. U+1FABD WING
- U+1FABF GOOSE .. U+1FAC5 PERSON WITH CROWN
- U+1FACE MOOSE .. U+1FADB PEA POD
- U+1FAE0 MELTING FACE .. U+1FAE8 SHAKING FACE
- U+1FAF0 HAND WITH INDEX FINGER AND THUMB CROSSED .. U+1FAF8 RIGHTWARDS PUSHING HAND

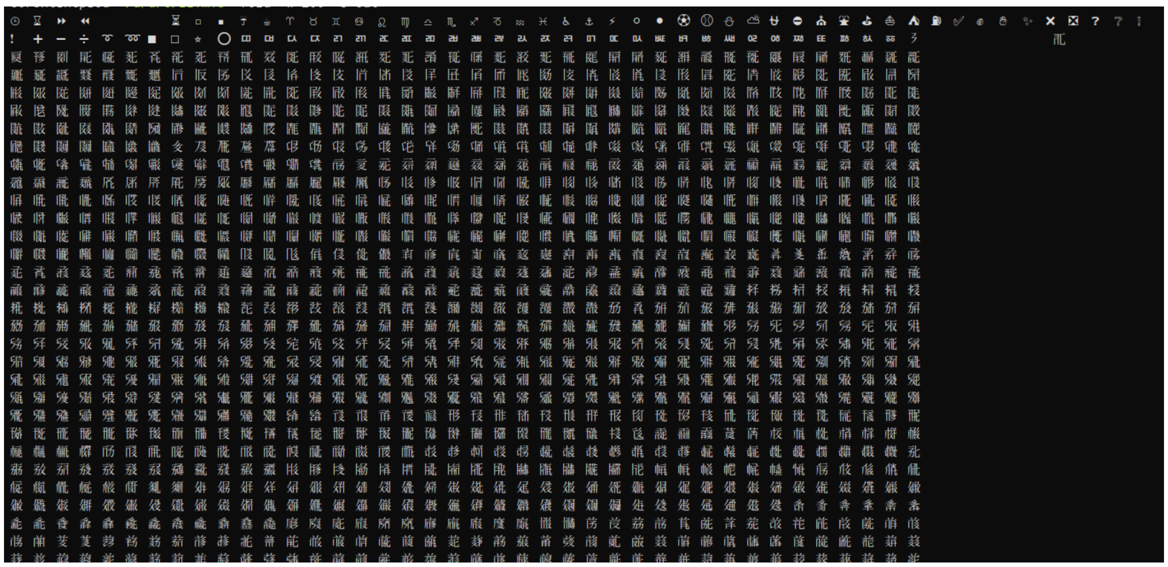
For the following code points, the estimated width used to be 2, and is 1 after the suggested change:

- U+2E9A
- U+2EF4 .. U+2EFF
- U+2FD6 .. U+2FEF
- U+2FFC .. U+2FFF
- U+3040
- U+3097 .. U+3098
- U+3100 .. U+3104
- U+3130
- U+318F
- U+31E4 .. U+31EF
- U+321F
- U+A48D .. U+A48F
- U+A4C7 .. U+A4CF
- U+FE53
- U+FE67
- U+FE6C .. U+FE6F
- U+FF00
- U+3248 CIRCLED NUMBER TEN ON BLACK SQUARE .. U+324F CIRCLED NUMBER EIGHTY ON BLACK SQUARE

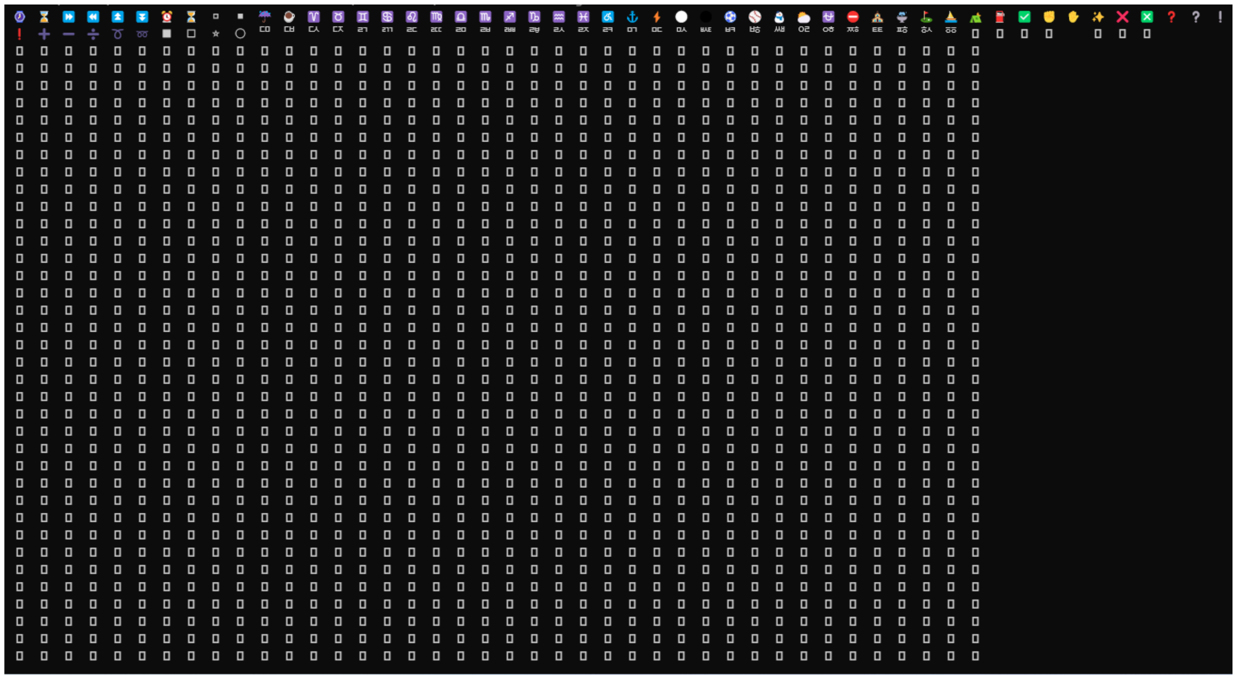
The script used to collect these data
https://gist.github.com/cor3ntin/deecab6d8d43713edd49e305b8140802

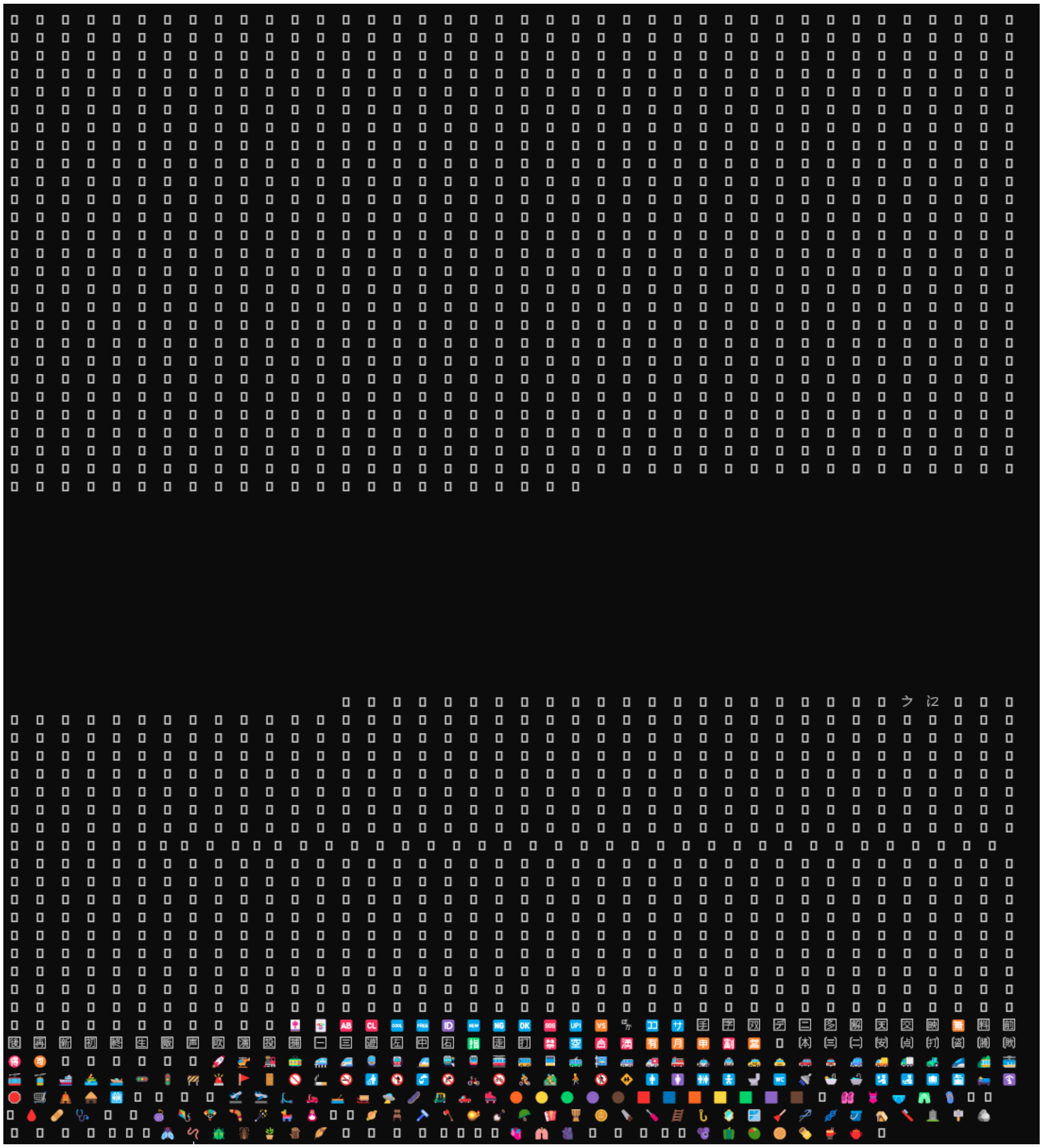
Annex: Terminal rendering

Konsole

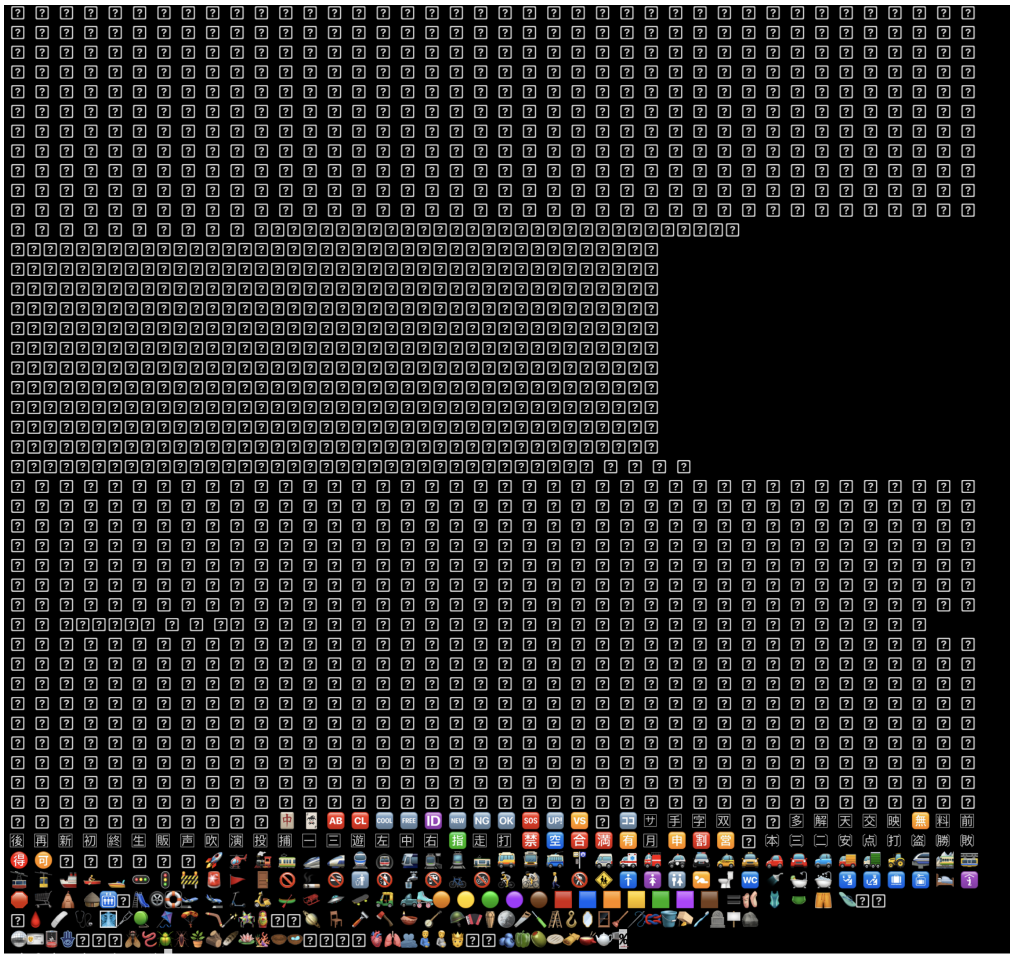


Windows Terminal

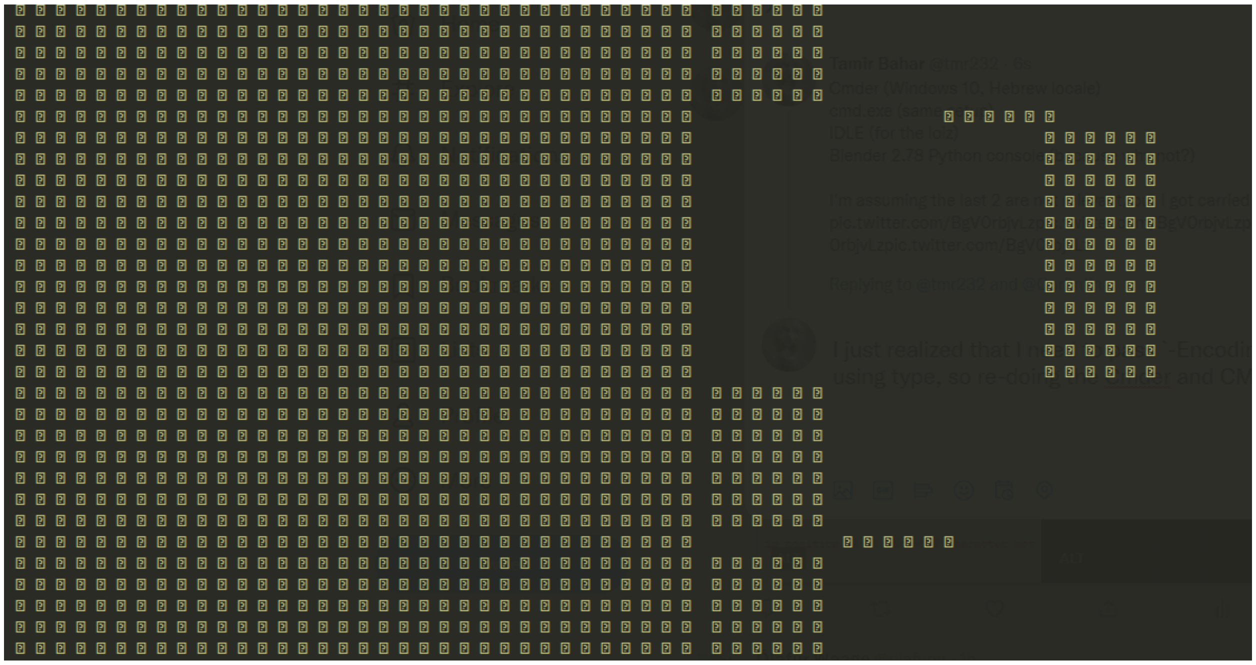




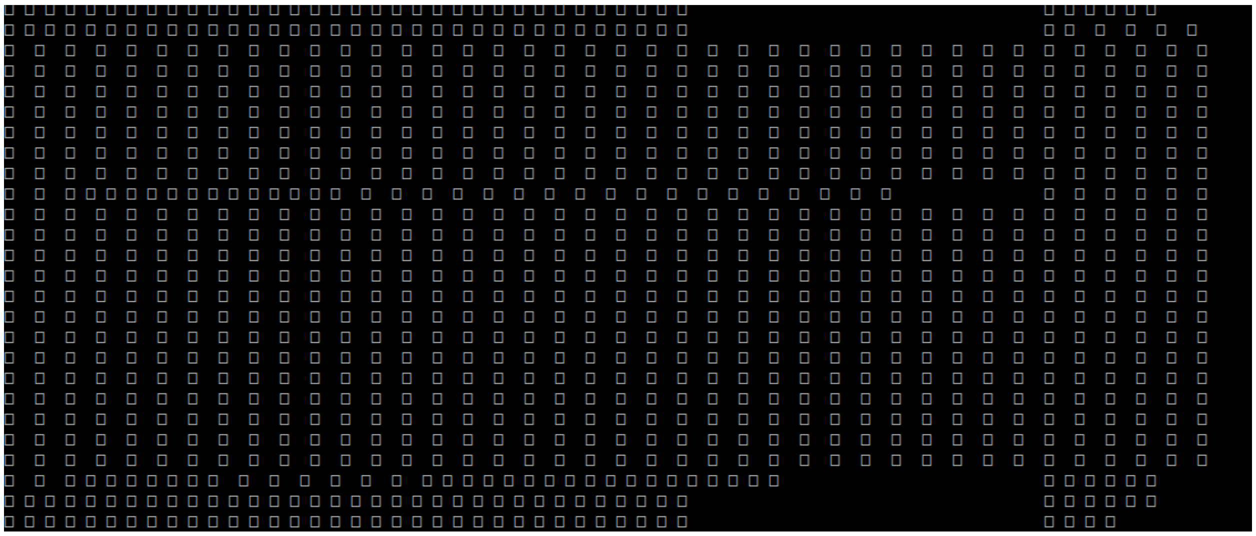
Mac OS (Thanks Victor)



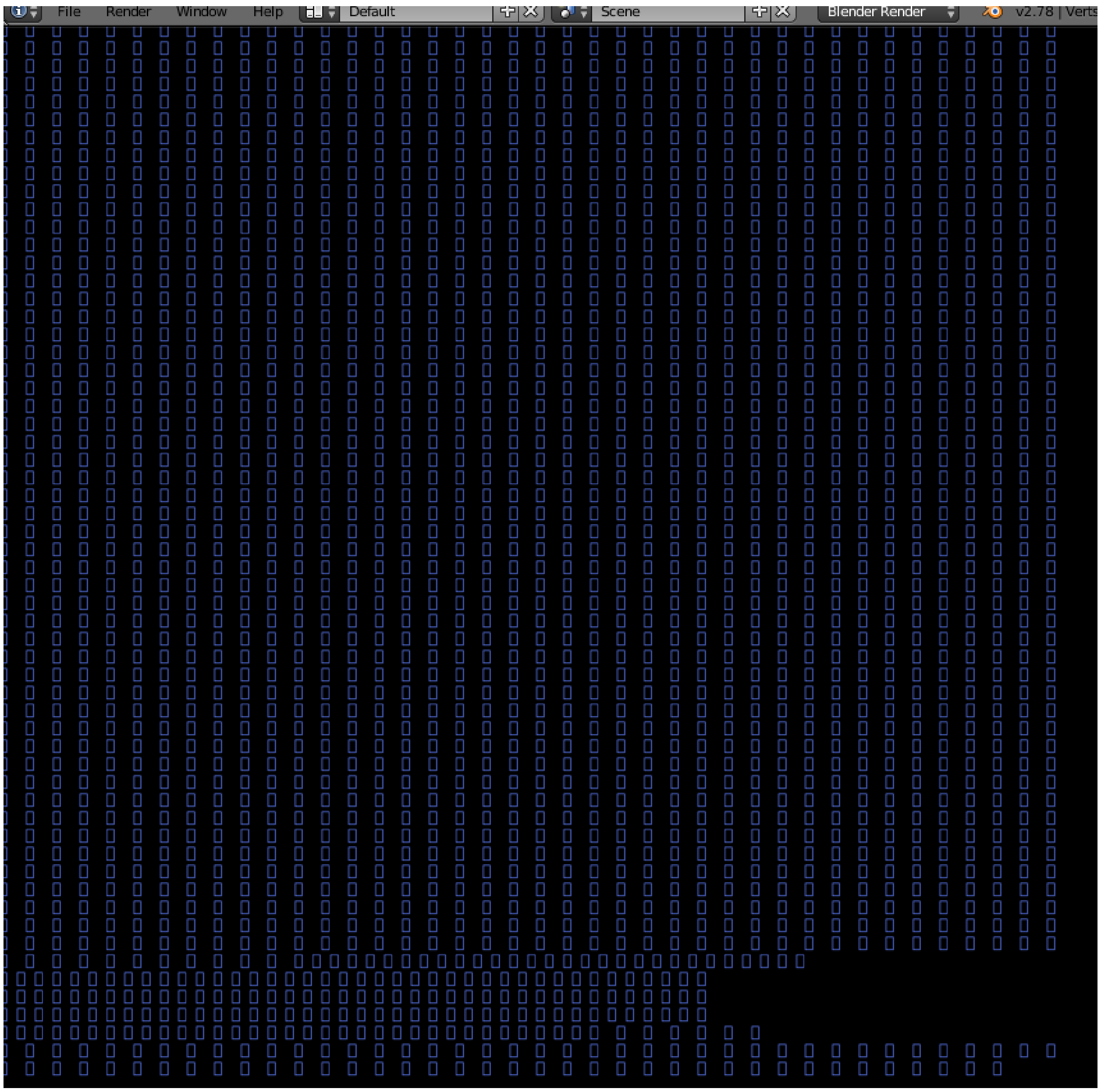
CMDer (Thanks Tamir Bahar)



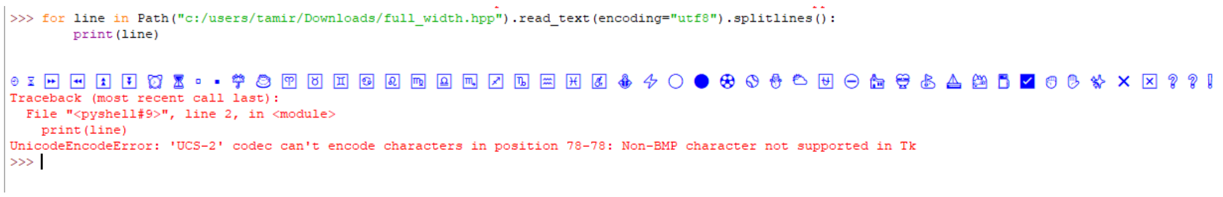
CMD.exe (Thanks Tamir Bahar)



Blender

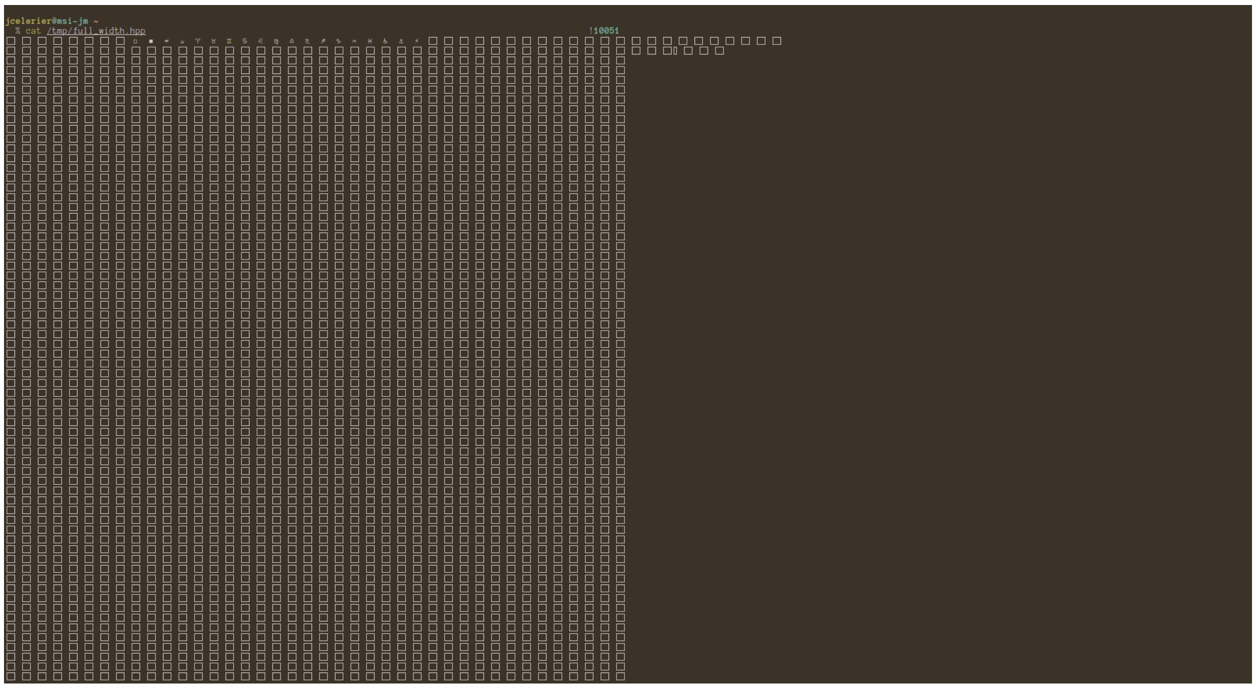


Python Integrated Development and Learning Environment



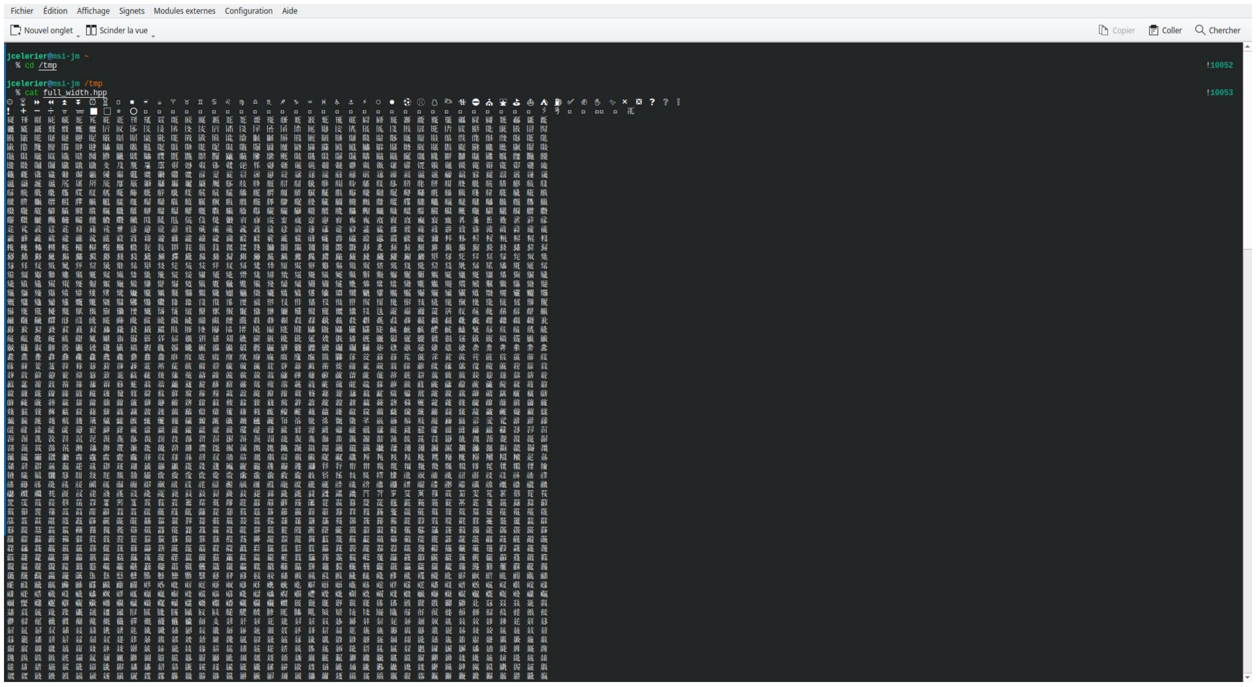
```
>>> for line in Path("c:/users/tamir/Downloads/full_width.hpp").read_text(encoding="utf8").splitlines():
        print(line)

⌚ ⌛ ⏩ ⏪ ⏫ ⏬ ⏰ ⏳ ◽ ◾ ☔ ☕ ♈ ♉ ♊ ♋ ♌ ♍ ♎ ♏ ♐ ♑ ♒ ♓ ♿ ⚓ ⚡ ⚪ ⚫ ⚽ ⚾ ⛄ ⛅ ⛎ ⛔ ⛪ ⛲ ⛳ ⛵ ⛺ ⛽ ✅ ✊ ✋ ✨ ❌ ❎ ❓ ❔ ❗
Traceback (most recent call last):
  File "<pyshell#9>", line 2, in <module>
    print(line)
UnicodeEncodeError: 'UCS-2' codec can't encode characters in position 78-78: Non-BMP character not supported in Tk
>>> 
```

rxvt-unicode (Thanks JEAN-MICHAËL CELERIER)



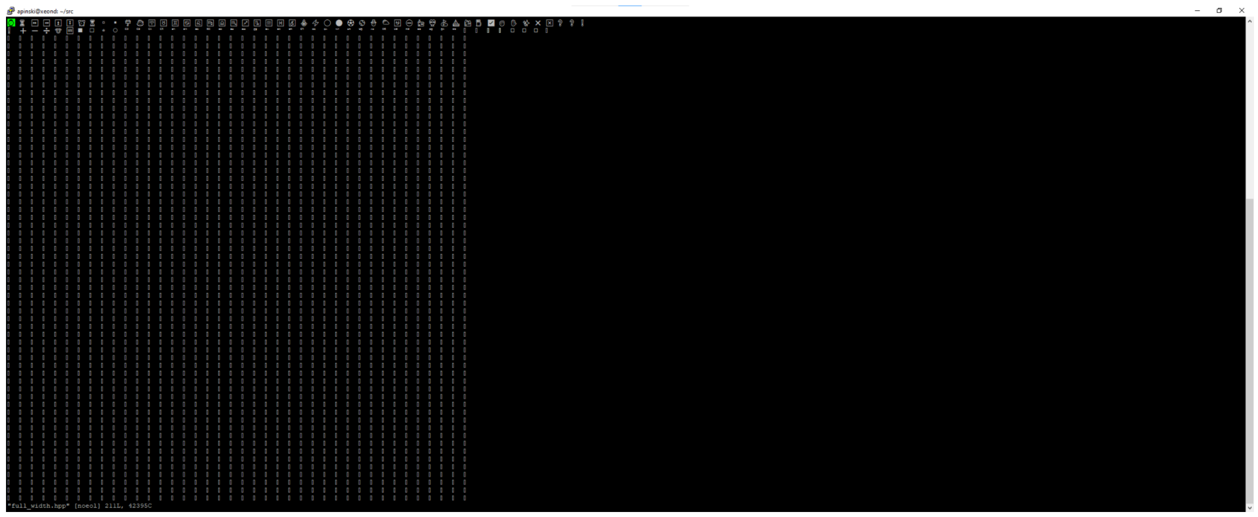
```
jcelerier@msi-jm ~
  % cat /tmp/full_width.hpp                                                    !10051
```

Konsole (Thanks JEAN-MICHAËL CELERIER)



```
jcelerier@msi-jm ~
  % cd /tmp                                                                    !10052

jcelerier@msi-jm /tmp
  % cat full_width.hpp                                                         !10053
```

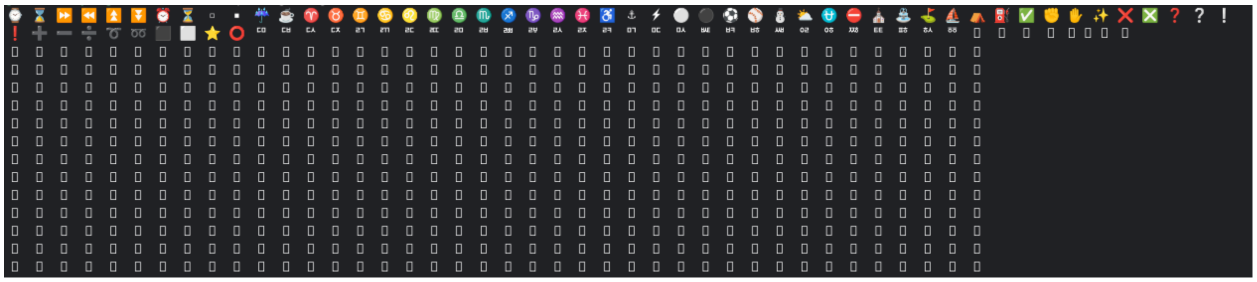
Putty (Thanks Andrew Pinski)

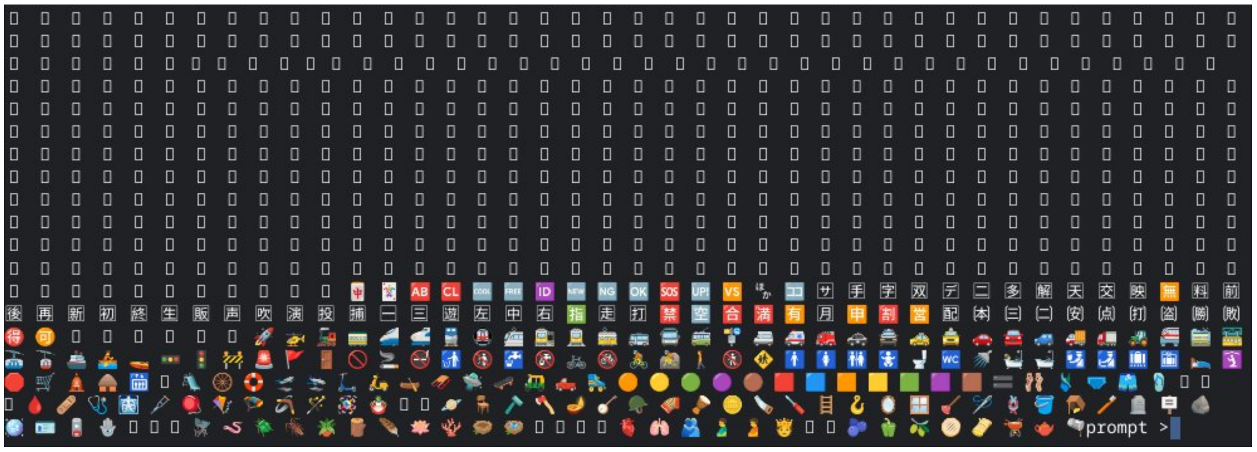


apinski@xeond: ~/src

```
"full_width.hpp" [noeol] 211L, 42395C
```

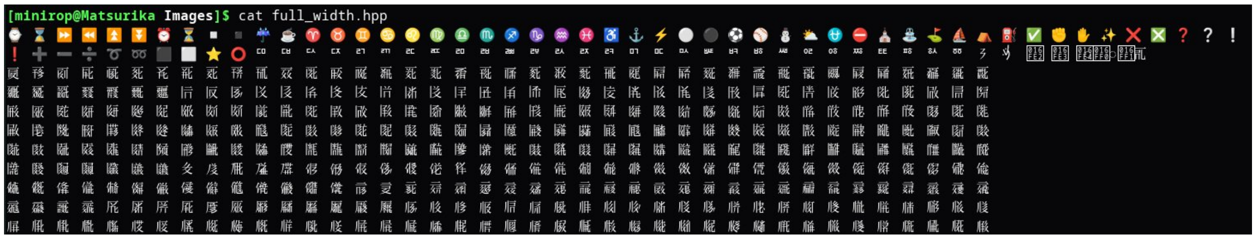
Chrome book



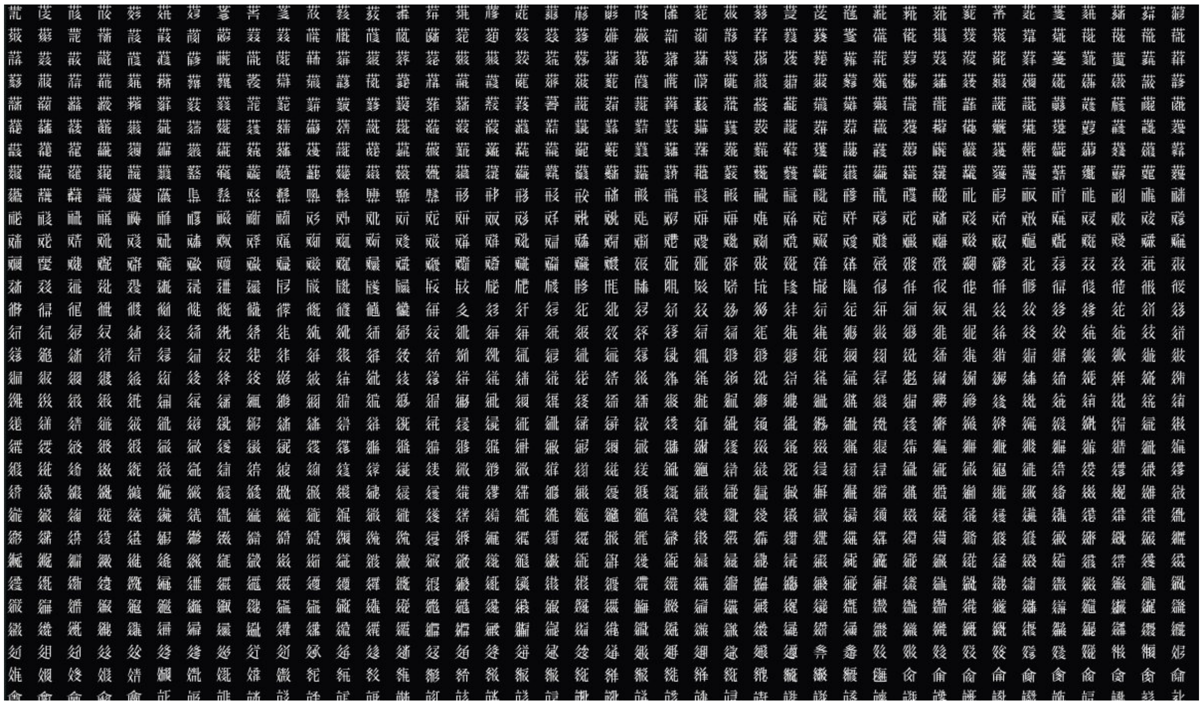


```
prompt >
```

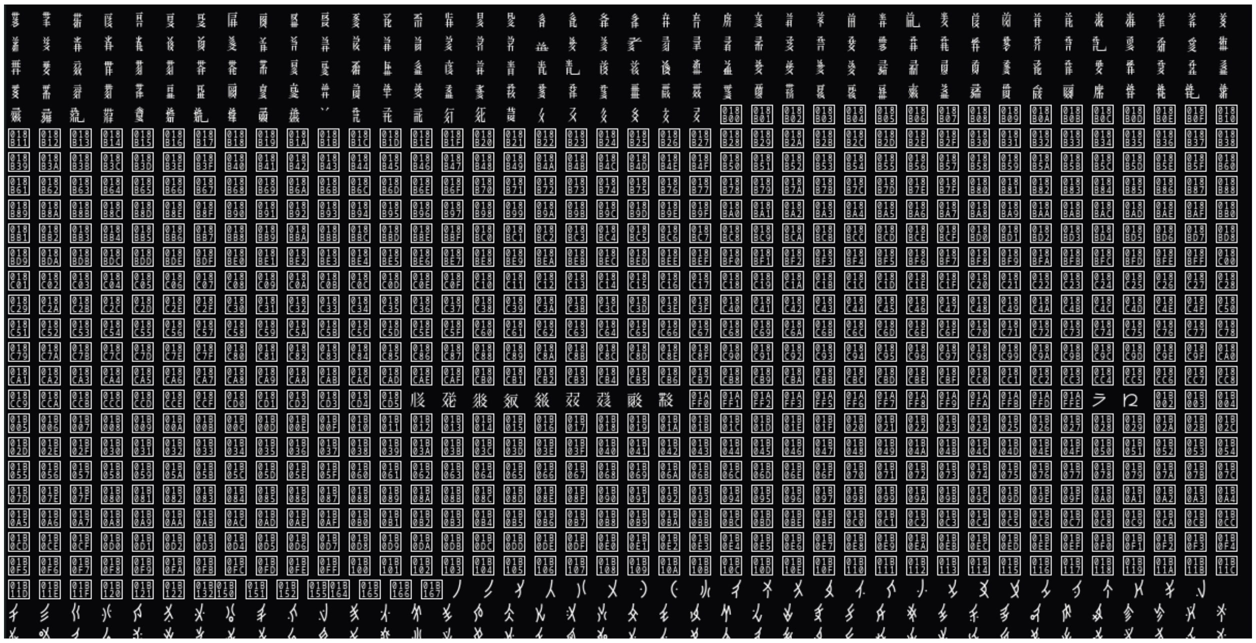
XFCE (Thanks Alexander Roper)



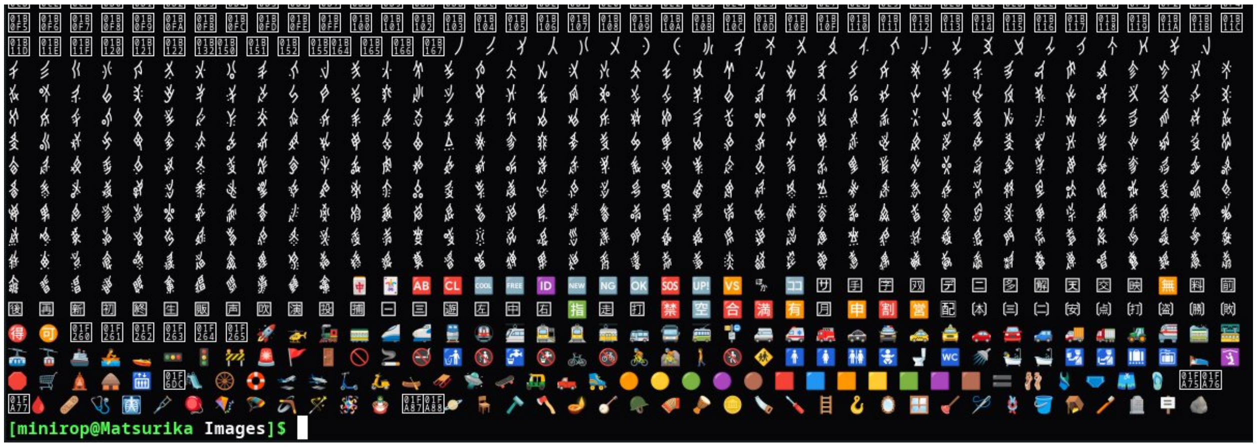
```
[minirop@Matsurika Images]$ cat full_width.hpp
⌚ ⌛ ⏩ ⏪ ⏫ ⏬ ⏰ ⏳ ◽ ◾ ☔ ☕ ♈ ♉ ♊ ♋ ♌ ♍ ♎ ♏ ♐ ♑ ♒ ♓ ♿ ⚓ ⚡ ⚪ ⚫ ⚽ ⚾ ⛄ ⛅ ⛎ ⛔ ⛪ ⛲ ⛳ ⛵ ⛺ ⛽ ✅ ✊ ✋ ✨ ❌ ❎ ❓ ❔ ❕
❗ ➕ ➖ ➗ ➰ ➿ ⬛ ⬜ ⭐ ⭕ [illegible]
[illegible]
```



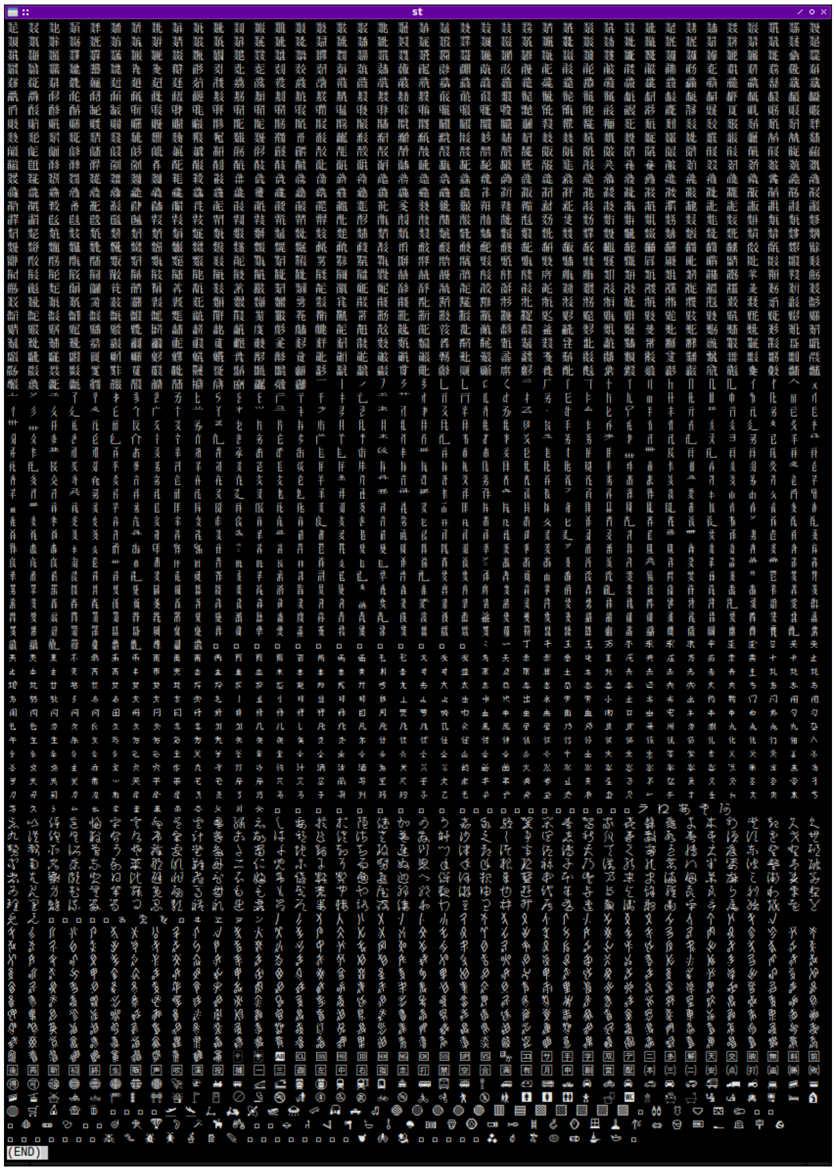
```
[illegible]
```



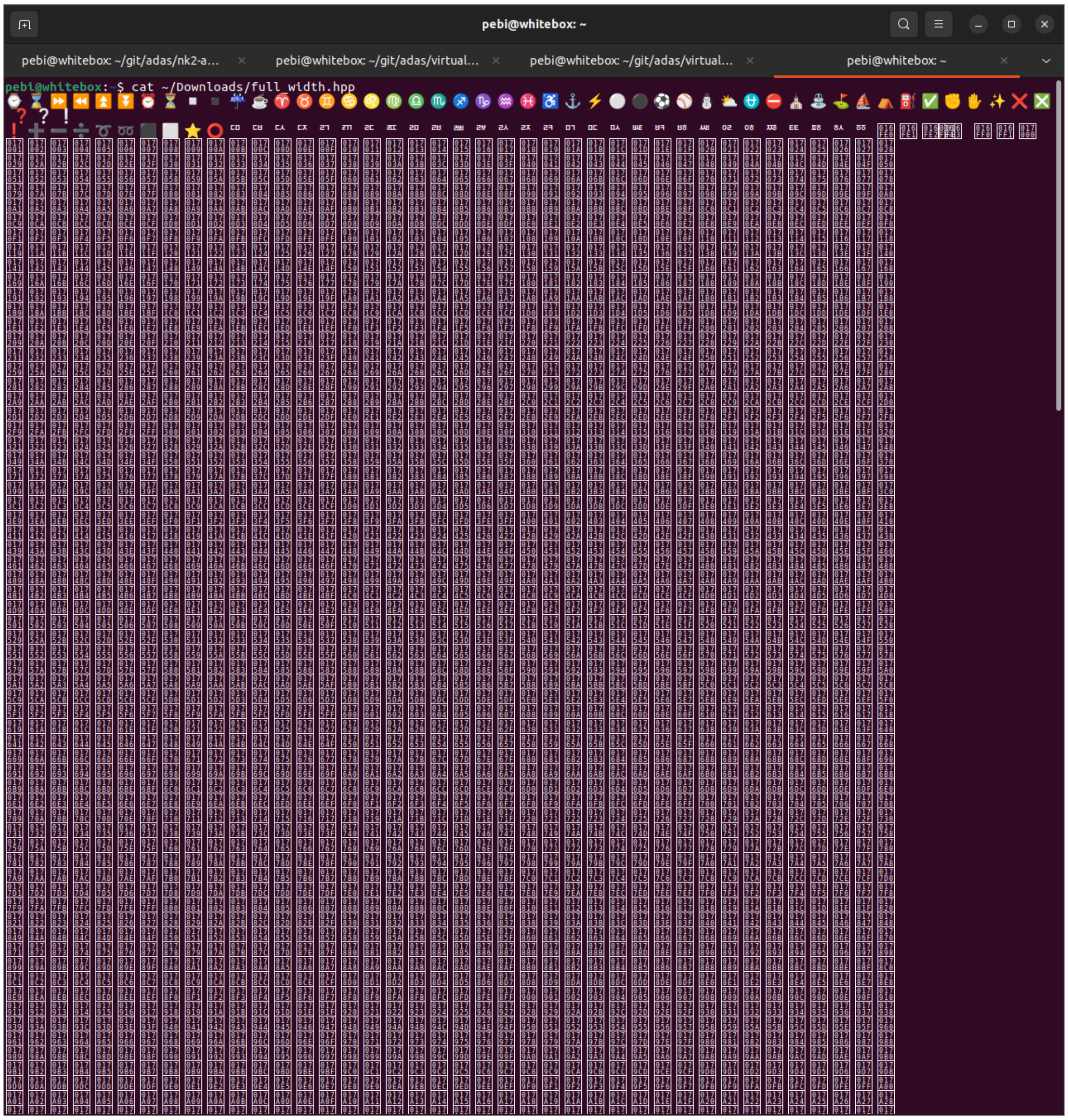
```
[illegible]
```


```
[minirop@Matsurika Images]$
```

ssterm (Thank you to the user "killerbee13" in the include C++ discord)



```
(END)
```



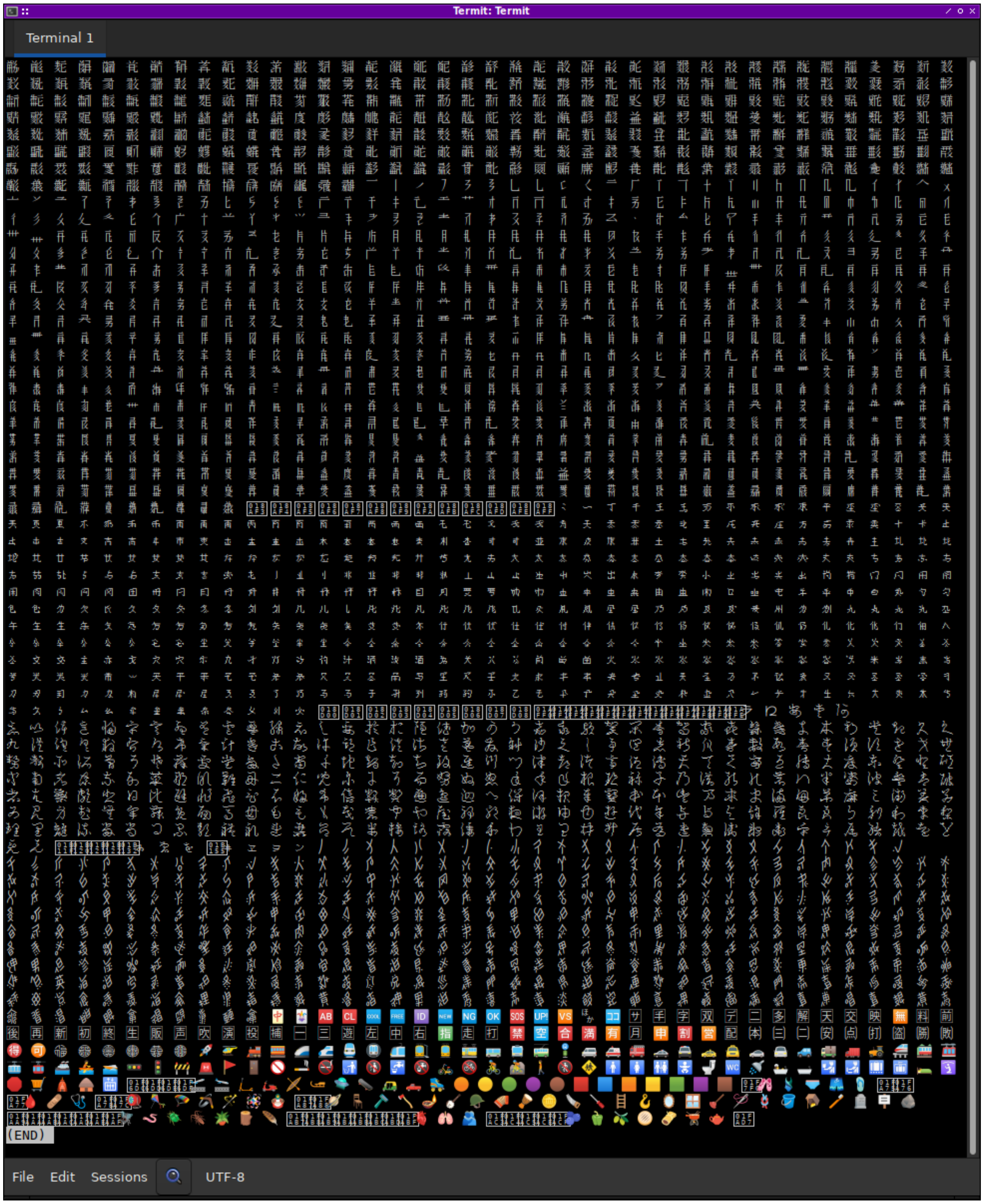
pebi@whitebox: ~
pebi@whitebox: ~/git/adas/nk2-a...
pebi@whitebox: ~/git/adas/virtual...
pebi@whitebox: ~/git/adas/virtual...
pebi@whitebox: ~

```
pebi@whitebox:~$ cat ~/Downloads/full_width.hpp
```

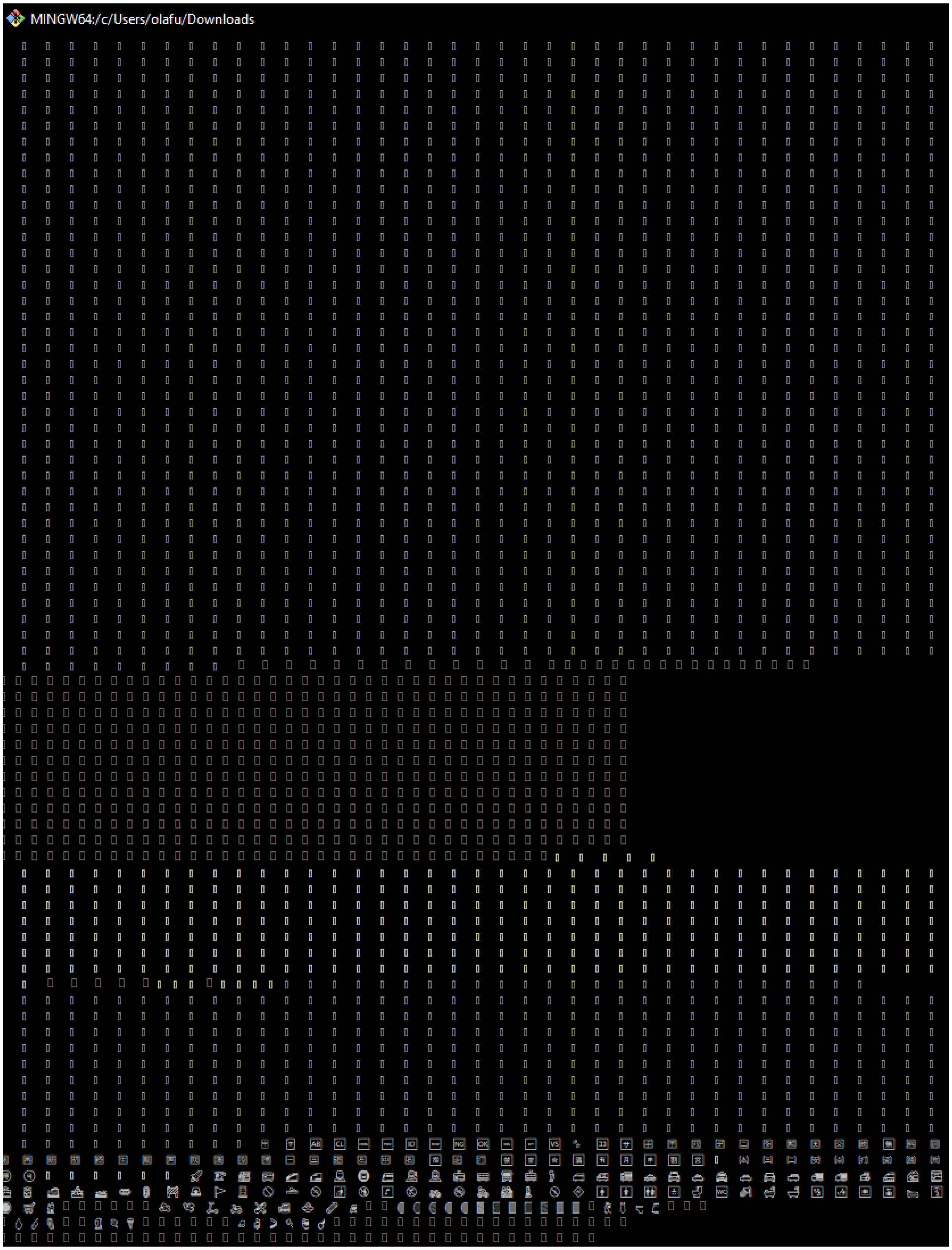
gnome-terminal (Thanks Peter Bindels)

```
pebi@whitebox: ~

pebi@whitebox: ~/git/adas/nk2-a...   pebi@whitebox: ~/git/adas/virtual...   pebi@whitebox: ~/git/adas/virtual...   pebi@whitebox: ~

後 再 新 初 終 生 販 声 吹 演 投 捕 一 三 遊 左 中 右 指 走 打 禁 空 合 満 有 月 申 割 営 配 本 三 二 安 点 打 盗 勝 敗
pebi@whitebox:~$
```

termit(Thank you to the user "killerbee13" in the include C++ discord)



Git Bash (MINGW64) on Windows (Thanks Ólafur Waage)



```
MINGW64:/c/Users/olafu/Downloads
```

Powershell in VSCode (Thanks Miro Knejp)