PyBoard and WiPy Editor: Small Python Text Editor

Looking for a code editor that would fit onto Pyboard (and now WiPy), I made my way through the Micropython forum and found **pfalkon**'s Python editor code, which I took and ported it to PyBoard. It's really impressive how few lines of code pfalkon needed to implement a reasonable amount of functionality. Since the code looked clean, and it seemed so easy to add features, I could not resist adding a little bit, using some ideas of **dhylands** for screen and keyboard handling, and yes, it got a little bit larger. The file size increased by a factor of 5 (only some of that caused by commenting), and the footprint in memory by a factor of 2 (now 14k), depending on what you keep. It still contains the code for the Linux/Darwin environment, so you can run it in Linux/Mac MicroPython (if you install the os module) or Python3. I sprayed C Preprocessor statements in it (arrgh!), so you can use cpp to remove the stuff which is not needed for PyBoard or WiPy. So, what did I change and add:

- Use USB_VCP or UART for input and output on PyBoard, stdin/stdout on WiPy.
- Changed the read keyboard function to comply with slow char-by-char input on serial lines.
- Added support for TAB, BACKTAB, SAVE, FIND, REPLACE, GOTO Line, YANK (delete line into buffer), ZAP (insert buffer), REDRAW, UNDO and GET (file).
- Join lines by Delete Char at the end or Backspace at the beginning, Autoindent for Enter.
- Moved main into a function with an optional parameter for tabsize
- Added a status line and single line prompts for Quit, Save, Find, Replace, Goto and Get file. The status line can be turned (almost) off for slow connections.
- Support of the basic mouse functions scrolling up/down and setting the cursor (*).

The editor works in Insert mode. Cursor Keys, Home, End, PgUp and PgDn work as you would expect. Some functions are available with Ctrl-Keys. Keyboard Mapping:

Keys	Alternative	Function
Up		Move the **cursor up** one line.
Down		Move the **cursor down** one line.
Left		Move the **cursor left** by one char, skipping over to the previous line.
Right		Move the **cursor right** by one char, skipping over to the next line.
PgUp		Move the cursor up by one screen height.
PgDn		Move the cursor down by one screen height.
Home		Toggle between the start-of-line and start-of-text.
End		Move to the end-of-line.
Enter	\n	Insert a line break at the cursor position. Auto-indent is supported.
Backspace	Ctrl-H	Delete the char left hand to the cursor. At the beginning of the line Backspace joins the previous line. (*)
Del		**Delete char** under cursor. At the end of the line join the next line. In line edit prompts, Del deletes the whole entry.
Tab	Ctrl-I	**Tab**. Insert spaces at the cursor position up to the next tab location, moving the cursor.
BackTab	Ctrl-U	**Back Tab.** Remove spaces left to the cursor position up to the next tab location or the next non-space char, and moves the cursor.
Ctrl-Q		**Quit** the editor or the line edit mode. If the edited text was changed and it was loaded from a file, a confirmation is asked for.
Ctrl-S		**Save** to file. The file name will be prompted for. Saving to internal flash of PyBoard is really slow, so don't get nervous. Watch the red LED.

Ctrl-E		**Redraw** the screen according to the actual screen parameters width, height and number of lines in the content. Under Linux with CPython, window size changes result in automatic redraw.
Ctrl-F		**Find** text. Whether the search is case sensitive or not, can be set by the Ctrl-A command. The last search string is memorized. Search stops at the end.
Ctrl-N		**Repeat find** starting at the column right to the cursor.
Ctrl-R		Find and **replace**.(*)
Ctrl-G		**Go to** Line. It prompts for the line number.
Ctrl-A		**Settings**. Sets the state of search case sensitivity, auto-indent, tab size and write-tabs flag. Enter ‘y’ or ‘n’ or a number in up to three, comma separated fields (e.g. n,y,4,n). An empty field leaves the respective value unchanged. The Default values as Case sensitive: n, auto-indent: y, Write Tabs: n (*)
Ctrl-B	Ctrl-Home	Go to last line (kind of obsolete, go to a huge line number will do the same). (*)
Ctrl-T	Ctrl-End	Go to first line (kind of obsolete, go to line 1 will do the same). (*)
Ctrl-X	Ctrl-Del	**Delete the current Line** and keep it in a line buffer. A sequence of these command will keep all lines in the order they were deleted in the buffer. Any other command in-between will start over the game. Together with the Ctrl-V this implements a very basic Cut & Paste feature.
Ctrl-D		**Copy the current line** into a line buffer and go down one line. A sequence of these commands will keep all lines in the order they were deleted in the buffer. Any other command in-between will start over the game. Together with the Ctrl-V this implements a very basic Copy & Paste feature.
Ctrl-V		Insert the content of the line buffer before the actual line.
Ctrl-O		Insert the content of a file before the actual line. (*)
Ctrl-Z		**Undo** the last change(s). Every char sequence change/replaced item/deleted line/inserted line(s) counts as a single change. The default for the undo stack is 50 on PyBoard and WiPy and 500 on Linux/Darwin systems. It can be changed.

Functions denoted with (*) are not supported in the minimal version (see below). The editor is contained in the file pye.py. Start pye from the REPL prompt e.g. with

```
from pye import pye
res = pye([object][, tabsize][, undo][, device][, baud])
```

If *object* is a string, it's considered as the name of a file to be edited, and the name of the file will be returned. If *object* is a list of strings, these will be edited, and the edited list will be returned. Otherwise, pye() will give you an empty screen, creating a list of strings, unless you save to a file. In that case, the file name will be returned. Optional named parameters:

tabsize=n	Tab step (integer). The default is 4
undo=n	Size of the undo stack (integer). A value of 0 or False disables undo.
device=n	Device to be used for screen/keyboard on PyBoard (integer). On PyBoard, 0 means USB_VCP. 1 means UART 1, and so on. The default is 0 (USB_VCP). On Wipy, 0 means Telnet, 1 means UART 0, 2 means UART1.
baud=n	UART baud rate (integer). The default is 115200.

The Linux/Darwin version can be called from the command line with:

```
python3 pye.py [filename]
```

Using python3 (not micropython), content can also be redirected or pipe'd into the editor. The file pye.py is pretty large (for PyBoard). As told, it contains C pre-processor statements allowing trimming it down a little bit. For that reason, comments start with ## instead of #. So for PyBoard, you might run:

```
cpp –D PYBOARD –D DEFINES pye.py  >pe.py
```

That will result in a file with all functions supplied, but smaller footprint when loaded. The directive DEFINES will replace symbolic key names with numeric constants, reducing the file size and the demand for symbol space. You may strip down the file size (not the compiled footprint) by removing comments and empty lines (that's what I do), e.g. by:

```
cpp -D PYBOARD -D DEFINES pye.py | sed "s/#.*$//" | sed "/^$/d" >pe.py
```

Doing that also removes dead code like the one for the Linux environment. If the footprint is still too large, you may choose:

```
cpp -D BASIC -D PYBOARD -D DEFINES pye.py | sed "s/#.*$//" | sed "/^$/d" >pemin.py
```

That removes the code for Mouse support, Replace, Goto first line, Goto last line, line join by backspace, flag settings, get file, and write tabs. There are still lines left like "if sys.platform == "pyboard". If you do not like these, delete them manually (and take care of the indents). If you just want to get rid of the cpp preprocessor stuff, run:

```
cpp  -D PYBOARD –D LINUX –D WIPY pye.py | grep -v "^#.*$" >pe.py
```

The WiPy version will be generated with:

```
cpp -D BASIC -D WIPY -D DEFINES pye.py | sed "s/#.*$//" | sed "/^$/d" >wipye.py
```

Only the minimal version runs on WiPy.

<u>Backup:</u>

The keyboard mapping assumes VT100. For those interested, I collected the key codes issue by terminal emulators, all claiming VT100 compatible:

Key	**Putty VT100 & Xterm**	**Putty esc-[~**	**Putty Linux**	**Minicom**	**GtkTerm**	**Picocom**	**Linux Terminal**
Up	\e[A	\e[A	\e[A	\e[A	\e[A	\e[A	\e[A
Down	\e[B	\e[B	\e[B	\e[B	\e[B	\e[B	\e[B
Left	\e[D	\e[D	\e[D	\e[D	\e[D	\e[D	\e[D
Right	\e[C	\e[C	\e[C	\e[C	\e[C	\e[C	\e[C
Home	\e[1~	\e[1~	\e[1~	\e[1~	\eOH	\eOH	\e[H
End	\e[4~	\e[4~	\e[4~	\eOF	\eOF	\eOF	\e[F
Ins	\e[2~	\e[2~	\e[2~	\e[2~	\e[2~	\e[2~	\e[2~
Del	\e[3~	\e[3~	\e[3~	\e[3~	\e[3~	\e[3~	\e[3~
PgUp	\e[5~	\e[5~	\e[5~	\e[5~	\e[5~	\e[5~	\e[5~
PgDn	\e[6~	\e[6~	\e[6~	\e[6~	\e[6~	\e[6~	\e[6~
Backspace	\x7f	\x7f	\x7f	\x7f	\x08	\x7f	\x7f
Ctrl-Home				\e[1;5H		\e[1;5H	\e[1;5H
Ctrl-End				\e[1;5F		\e[1;5F	\e[1;5F
Ctrl-Del				\e[3;5~	\e[3;5~	\e[3;5~	\e[3;5~
Tab	\x09	\x09	\x09	\x09	\x09	\x09	\x09
BackTab	\e[Z	\e[Z	\e[Z	\e[Z	\e[Z	\e[Z	\e[Z

Picocom seems sometimes to send the Linux Terminal codes. Nevertheless, I'm using it most of the times. If the KEYMAP is too large, and you know which terminal you are working on, delete or comment out the obsolete lines. If your terminal is different, just change the control codes.

Notes:

- When you save a file on PyBoard, these changes are not visible in the file system of a connected PC until you disconnect and reconnect the Pyboard drive, e.g. by a hard reset of PyBoard. See also the related discussion in the MicroPython Forum.
- For those who wonder why sending data to the screen on PyBoard is more than a simple write(): for USB_VCP.write() stumbles over a large amount of data to be sent in short time. The difference is, that UART.write() waits internally until all has been sent, whereas USB_VCP.write() stops when it cannot send more data. So we have to see what's coming back. And, b.t.w., UART.write() does not like empty strings, which in turn is accepted by USB_VCP.write().
- Windows terminal emulators behave inconsistent. Putty does not report the mouse actions at all. TeraTerm, IVT terminal and Xsh20 just report the mouse click, but not the scroll wheel actions. ZOC reports mouse positions constantly, and sends no key codes for Home, End, PgUp, PGDn and Del. The latter holds also for PowerVT. I could not get Qodem working. Hyperterminal's VT100 emulation is crap. So, after all, I consider TeraTerm, Xsh20 or Putty as the best choices for Windows.
- Gnome terminal sometimes does not send the first mouse wheel code, after the pointer was moved into the window. Mate and XFCE4 terminal do, but have slightly different keyboard mappings.
- Serial connection on WiPy is to be taken with care with fast auto-repeat. Ctrl-C as input in Telnet sessions is caught, such that the editor is not interrupted, but the next input byte is lost, which causes garbage by cursor/function keys.
- When reading files, tab characters (\x09) in the text are replaced by spaces, tab size 8, and white space at the end of a line is discarded. When you save the file, you have the option to replace sequences of spaces by tabs, tab size 8. However, the original state will NOT be restored. So be careful when editing files which tab characters. For micropython, I added a simple version of expandtabs. If at any time the expandtabs method is supported by Micropython, I will be happy to dump this code.

To Do:

- Testing, using, porting, polishing and bug juggling.