BlueBox Documentation

last updated 7/25/2014

1 Introduction

The BlueBox library is a Python module based on libtcod for writing applications that simulate the appearance and capabilities of the Buttech CAI-1 "Blue Box," an educational computer developed in the 1980s. It provides a simple object-based interface to a text-and-graphics console, essentially aiming to provide a basic subset of standard I/O routines.

1.1 History

The "Blue Box" came about in the early 1980s under the request of the Butte County Board of Education to procure a low cost terminal or personal computer that could be installed in classrooms. After initial complaints from the board regarding the potential procurement cost involved in existing solutions, a proposal was floated to instead commission one from a local electronics manufacturer according to the logic that "it can't be that difficult, that Apple guy did it in his garage." (actual quote from the minutes)

After a very short bidding round (few electronics makers even existed in Butte County, and fewer still would accept the low maximum bid), the task fell to Buttech, a small maker of video display terminals. A range of demands were placed on the machine despite already being short on both budget and time before the development phase had even begun. Once started, project difficulties began almost immediately. Buttech refused to commit much manpower to the project, citing low cost. The council demanded a color terminal "for art classes," but insisted it be cheaper than any terminal presently available.

Endless debate raged over the choice of built-in interpreter, with a licensed BASIC rejected as "too expensive" by some and even an in-house version rejected because the board chairman insisted it was "too complicated," vetoing it for a PILOT-based proposal. Eventually the programmer on the PILOT project was fired, and an intern hastily assembled VIOLET from the ashes of several competing prototypes and all of a day and a half reading a textbook on language implementation. Even after VIOLET was finished, many Buttech-developed applications were actually written and compiled in LISP or Forth.

Due to the development difficulties, manpower and budget shortages, and manufacturing issues, the Buttech CAI-1 didn't actually make it to classrooms

until 1987, by which point it was already utterly obsolete compared to contemporary hardware. Nevertheless, the "Blue Box" (nicknamed for its unusual case color and a fault that turned its 4-tone monochrome output a light shade of baby blue) continued in operation well into the early to mid 1990s in a number of Butte County schools owing to the exclusive contract the BoE had with Buttech (which was later the subject of a massive tax appropriations scandal that eventually led to most of the boxes being seized and destroyed).

2 Usage

2.1 Initialization

To use BlueBox, one must first begin by importing libbluebox and creating an instance of the BlueBox object, which starts the interface and whose methods will provide interaction with said interface. This can be done simply enough (provided you want only the default, 40-column interface) just so:

```
import libbluebox

box = libbluebox.BlueBox()
```

However, there are a number of initialization options that can be used to change the parameters of BlueBox, which are listed as follows, with their defaults:

win_name='BlueBox' A string which provides the name of the console window.

boot_msg=True If True, the newly created console will begin operation by printing the original ROM boot message from the CAI-1. False will suppress this message.

graphics_mode=False When true, the box will boot into graphics mode, rather than text mode. Graphics mode provides a semigraphics screen with horizontal and vertical resolution equal to double the width and height of the text mode, and can be interacted with through the graphics methods described later in this documentation. Changing this at boot is not necessarily recommended, but either way it can be toggled during runtime.

width=40 The number of characters wide the text mode screen will be. If set to >=80, will use the half-width 80-column font. For authentic CAI-1 operation, this should only ever be 32, 40, or 80.

height=24 The number of characters high the text mode screen will be. On an original "Blue Box," this is only ever 24.

fps=24 The frame limit for the libtcod console rendering. 24fps seems to be a reasonable middle ground between usability and accuracy.

foreground=color_on The default foreground color for either text or graphics. Chosen from one of four pre-defined values (`color_bold, color_on, color_half`, or `color_off`) based on the original CAI-1 4-intensity mode, or can be assigned a libtcod color constant instead if using for some other purpose. See the libtcod documentation for details on other colors.

background=color_off The default background color for either text or graphics.

In addition, the initialization routine also creates internal values and objects used by the main class methods, but could potentially be referenced externally (though messing with them is probably not the best idea).

BlueBox.cursor A simple object that contains the x, y, and character values for the Cursor. x and y should not be messed with, however if you wish to change the cursor, sending a new character in the form of a `chr()` or single-character string to self.cursor.char *should* be safe, but has not been tested.

BlueBox.screen A 2 dimensional list containing the current contents of the text display, referenceable with `self.screen[x][y]`.

BlueBox.img This parameter contains the libtcod format image for the current graphics mode page. Passing anything to this value at startup is probably not recommended, as you will need to create it through the relevant libtcod image toolkit routines. At boot, if `graphics_mode` is set `True`, the `__init__` method will create one anyhow.

2.2 Methods

The following subsections will describe and document the usage of the various internal methods provided by the BlueBox class.

2.2.1 BlueBox.change_resolution(target_width, target_height=24)

Changes the current number of rows and columns of the text display. Note that unlike defining these values at boot, changing the resolution during runtime currently does not change the font used in rendering. This appears to be a limitation of the libtcod library underlying BlueBox.

2.2.2 BlueBox.check_interrupt():

A static method that calls libtcod's input routines to check for either an Escape key event or a window_close event, and returns True if one has occured. Sometimes finicky.

2.2.3 BlueBox.clear_graphics():

Clears the current graphics page by setting all pixels within `BlueBox.img` to `BlueBox.foreground`.

2.2.4 BlueBox.clear_screen():

Clears the entire screen buffer and graphics page both, and resets the cursor to 0,0.

2.2.5 BlueBox.display_screen():

Renders the current screen buffer and (if graphics mode is on) draws `BlueBox.img`. Generally not necessary when working in text mode (as most routines for dealing in text mode automatically call it), but useful when dealing with graphics for ensuring that changes to the graphics page are properly drawn when needed.

2.2.6 BlueBox.draw_line(sx, sy, tx, ty, color=None)

Draws a line on the graphics page from point sx, sy to point tx, ty in the color designated. If color is left out, will set color to the current foreground color.

2.2.7 BlueBox.draw_point(x, y, color=None)

Draws a pixel on the graphics page at point x, y, in the color designated. If color is left out, will set color to current foreground.

2.2.8 BlueBox.set_graphics(flag=None)

If no flag is given, toggles the state of `BlueBox.graphics_mode`, otherwise, if set to `True` or `False`, sets it to those values.

2.2.9 BlueBox.text_out(text, newline=True)

Prints out the contents of string `text` to the console at the current cursor location. If `newline=True`, it ends by moving the cursor to the next line.

2.2.10 BlueBox.text_in(newline=False, prompt=False, prompt_text='> ')

Takes input from the keyboard at the current cursor location. If `newline=True`, will move the cursor to a new line before taking input. If `prompt=True`, will print the contents of `prompt_text` to the cursor location before taking input.

2.2.11 BlueBox.new_line()

Moves the cursor to a new line, and scrolls the text vertically if at the end of the screen.

2.3 Colors

The libbluebox library also provides four color constants which can be referenced in order to easily access the standard pixel intensities of the CAI-1 terminal screen. They are:

color_off Pixel off, ie. black.

color_half Pixel half on, dark blue.

color_on Pixel on, light blue.

color_bold Pixel extra-bright, white-blue.

Internally, these colors are actually defined as gradients between libtcod.black and libtcod.light_azure, at 0, 50%, 75%, and 100% respectively.

3 Included Software

The BlueBox library repository also contains additional programs that make use of libbluebox.py. These are currently:

violet.py The interactive editor for the VIOLET programming language. Run this to use VIOLET in its original environment. Further documentation is provided with this library.

runviolet.py The actual interpreter for the VIOLET language. Currently this still requires a BlueBox object to operate, so is of limited utility without modification, but can be imported alongside libbluebox to provide access to one. Currently undocumented internally because the code is scary.