Emacs as my Canvas

Vasilij Schneidermann

August 2015

Outline

Section 1

Introduction

Who?

- Vasilij Schneidermann, 23
- Information systems student
- Working at bevuta IT, Cologne
- v.schneidermann@gmail.com
- https://github.com/wasamasa
- http://emacshorrors.com/

What?

- Emacs is an extensible, QWAN platform for all things text
- Very fun to hack on
- Yet very little graphical demos
- Let's change that!

Why?

Figure: The Mad Scientist

http://lastmechanicalmonster.blogspot.com/2014/08/page-91.html

Section 2

Basics

Emacs

- A text editor from 1985
- Ported to all kinds of common and lesser common platforms
- Mostly implemented in Emacs Lisp
- Extensible at runtime
- Many interesting features

Buffers

- Basic data structure
- Basically two strings and a gap
- Can represent both files and text in memory
- Used heavily for text processing

Text Properties

- Supported for both strings and buffers
- Property-value pairs attached to ranges of characters
- Some of the properties are special, like `face` or `read-only`
- The `display` property allows abusing the display engine!
- See `(elisp) Special Properties` and `(elisp) Display Property`

Display Engine

- Emacs assumes that text is set on a two-dimensional grid
- The `display` property allows for exceptions...
- What we're after is the `image` descriptor
- Works for both files from disk and string data!
- Code to display an image in the current buffer:

```
(insert
 (propertize " " 'display
             '(image :type jpg :file "keyboardcat.jpg")))
```

Supported Image Types

JPEG, PNG	Very common, mostly used with files from disk
GIF	Animation!
XBM	Monochrome bitmaps, easily generated
XPM, PBM	Textual format, easily generated
SVG	XML format for vector graphics
PostScript	No idea why you'd use *that*
TIFF	See above
ImageMagick	Support for nearly any format plus tunables

Locations

Images can be used in:

- Buffer
- Mode line
- Header line
- Echo area
- Tooltips (non-native)

Mouse Events and Tooltips

- Mouse generates movement, click and scroll events
- Movement can be tracked via `track-mouse` (CPU-intensive)
- Trigger tooltips with `help-echo` and cursor changes with `cursor` property
- Tooltips can be text and even images!
- It's possible to write mouse handlers by using the `:map` property in the image descriptor
- Alternatively bind a command on the mouse event and examine positions

Timers

- Emacs is a single-threaded application, but can pretend it's not
- Timers belong to this category and can be run when Emacs isn't busy
- Idle timers are run after a specified time of inactivity has passed
- Regular timers can be scheduled and are either of the one-shot or repeat type
- If you use too many timers with small intervals in your Emacs session, fun side effects like cursor flicker can happen...

Section 3

Demonstrations

nyan-mode

http://github.com/TeMPORaL/nyan-mode

svg-mode-line

- Previous demonstration was about a *segment* of the mode line
- Some Crazy Russian™ did replace the whole mode line
- http://github.com/sabof/svg-mode-line-themes
- http://github.com/ocodo/ocodo-svg-modelines

BGEX

```
(bgexi-create (bgexid-create
               nil 'bgex-identifier-type-default)
              t nil "white"
              (expand-file-name "~/rms.png"))
```

- Some Crazy Japanese™ ported XEmacs' background pixmap support
- Requires a patched Emacs
- Supports files from disk and strings
- Animation doesn't work well, only tiling is supported
- https://github.com/wachikun/emacs_bgex

svg-2048

- Remember 2048?
- Web Designers did mods of the original things
- Emacsers did ASCII versions of the game
- I went after a graphical version
- Turns out it's as simple as generating SVG, deleting the game buffer contents and inserting the image on each command
- Purely event-driven
- No animations yet
- https://github.com/wasamasa/svg-2048

xbm-life

- XBM is an always built-in monochrome image type
- This was a test to find out how suitable it is
- Bool vectors are funky, but other than that...
- Timers are sort of weird, but useful
- Learned about the UI aspect of a game/simulation
- https://github.com/wasamasa/xbm-life

retris

- I really love the NES Tetris
- As I've already experimented with SVG and XBM for generating images, XPM was the next candidate
- While this is a simple game, it involves more than the other two and needs to run at a constant 60 FPS
- Is that kind of thing doable in Emacs Lisp?
- https://github.com/wasamasa/retris

Section 4

Insights From Retris

Retro Games Are Great To ~~Steal~~ Learn From

- Creative use of resources
- Interesting implementation techniques
- No game engines, every game is custom-tailored
- I'm cloning a retro game after all...
- Notable exception:
  https://gist.github.com/dto/4112806

Impedance Mismatches

- Don't use game buffer changing functions outside buffer
- Buffer and window point can be different
- Displaying windows is icky
- Deleting and inserting doesn't play well with scrolling, region, clicks, etc.
- A game loop inside an event loop feels wrong
- Timers were not made for this purpose (but can be made to work well enough)

Data Structures

- "They Called It LISP for a Reason: List Processing"
- Support for other compound data structures than lists is very basic
- Contrast with CL (polymorphic functions that work on more than just lists) or Clojure (seqs as general abstraction, first-class support for vectors, hash tables, sets, etc.)
- This includes vectors, hash tables and strings(!)
- Sets aren't a thing, structs are an ugly hack from `cl-lib.el`
- Lists are abused for emulating other data structures, including sets and hash tables or used instead of vectors

Writing Vector Functions

- The most natural way of representing tiles, grids, etc. is a vector
- Coercing vectors into lists and back is a no-no
- Let's write our own functions and macros!
- I consider releasing these (and many more) as v.el

Writing Vector Functions

```
(defalias 'v-copy 'copy-sequence)

(defun v-deep-copy (vector)
  (copy-tree vector t))

(defun v-grid (width height init)
  (let (grid)
    (dotimes (_ height)
      (push (make-vector width init) grid))
    (vconcat grid)))
```

Writing Vector Functions

```
(defmacro v-do (spec &rest body)
  (declare (indent 1))
  (let ((s (make-symbol "s"))
        (i (make-symbol "i")))
    `(let ((,s (length ,(cadr spec)))
           (,i 0)
           ,(car spec))
       (while (< ,i ,s)
         (setq ,(car spec) (aref ,(cadr spec) ,i))
         ,@body
         (setq ,i (1+ ,i))))))
```

Mutating Strings

```
/* XPM */
static char *graphic[] = {
/* width height colors chars_per_pixel */
"4 4 2 1",
/* colors */
"o s #ffffff",
"x s #000000",
/* pixels */
"ooxx",
"ooxx",
"xxoo",
"xxoo"}
```

Instead of mutating a buffer and repeatedly creating a string of its contents. . .

Mutating Strings

```
/* XPM */
static char *graphic[] = {
/* width height colors chars_per_pixel */
"4 4 2 1",
/* colors */
"o s #ffffff",
"x s #000000",
/* pixels */
"xxoo",
"xxoo",
"ooxx",
"ooxx"}
```

. . . I went for treating a string as a mutable array, simply to conserve RAM.

Reimplementing React

- Wrote primitives to modify XPM image
- Redrawing the whole grid is too slow for 60FPS
- A clever hack was necessary!
- React does this with a virtual DOM on animation timeouts
- If a dirty flag is set, compare snapshots of the grid, then redraw the differences
- Ugly, but works surprisingly well

Reimplementing React

```
(let (coords)
  (dotimes (y board-height)
    (dotimes (x board-width)
      (let ((old-piece-char (aref (aref old-board y) x))
            (new-piece-char (aref (aref board y) x)))
        (when (/= old-piece-char new-piece-char)
          (push (list x y (tile-char-lookup
                           new-piece-char))
                coords)))))
  coords)
```

Reimplementing React

```
(when dirty-p
  (dolist (item (diff-boards))
    (-let [(x y tile-char) item]
      (render-tile x y tile-char)))
  (setq old-board (copy-tree board t)
        dirty-p nil)
  (with-current-buffer "*retris*"
    (let ((inhibit-read-only t))
      (erase-buffer)
      (insert
       (propertize
         " " 'display
         (create-image (concat board-header board-body)
                       'xpm t :color-symbols palette))
       "\n"))))
```

Scheduling Events

- Trying to outsmart the built-in timer support...
- Many concurrent timers with small intervals make Emacs flicker
- List of vectors representing events
- Internal clock advancing every frame
- Any event with clock modulo interval equal remainder is collected
- Run accumulated functions later
- Oneshot events: Remove them from the list after running
- No flicker!

Scheduling Events

```
(let (tasks)
  (dolist (event events)
    (when (= (mod time (aref event 0)) (aref event 1))
      (push (aref event 2) tasks)))
  tasks)
```

Scheduling Events

```
(dolist (task (scheduled-tasks))
  (funcall task))
(redraw-board)
(setq time (1+ time))
```

Section 5

Wrapping up

Was It Worth It?

- Definitely!
- Working around the deficiencies of Emacs was sort of bothersome
- Developing interactive demos in Emacs is fun
- I did learn a lot from this (like, why nobody wrote platformers, shooters or anything else than puzzle games)
- Join me!

Other Stuff To Work On

- GIF authoring
- Bitmap editor
- Vector editor
- Pixelart (CSS export?)
- Demos (scene.org)
- Image preview tooltips (IRC clients)
- ...

Questions?