# Pfff visual

## Google Maps for Source Code

## version 0.1

Yoann Padioleau
pad@facebook.com

September 24, 2010



## Contents

# 1 Introduction

## 1.1 Motivations

Note that this is not a replacement for Emacs or Vi, but more a companion that works with Emacs or Vi, a little bit like the Speedbar Emacs project, except it is using a treemap instead of a classic hiearchy browser.

```
%integrate visualizer and source code ! separate skill for now
%later: integrate more artifacts. See vision.txt
```

```
 * Archi: There are different kinds of "drawings":
 *  - paint, which does the heavy and expensive rendering
 *  - expose, which assemble the already painted pixmaps/layers and allow
 *    moving parts such as overlay rectangles
```

```
(*
 * This file is the basis for a new kind of code visualizer,
 * with real time zoom on a treemap and partial thumbnails with anamorphic
```

```
 * code; A google maps but on code :)
 *
 *
 * By playing with colors, size, fonts, and transparency, can show lots
 * of stuff.
 *
 * Assumes the treemap contains the absolute paths to existing files/dirs.
 *
 * There is not a single view that can accomodate all
 * navigation/code-understanding programmer needs. So we provide multiple
 * features that can display things at different levels:
 *  - minimap, for context and quick navigation
 *  - zoomable/draggable map
 *  - content thumbnails, with anamorphic text for more important entities
 *  - magnifying glass on the zoomable map (=> have then 3 layers of zoom
 *    where can each time see the context)
 *  - clickable map so redraw treemap on focused dir (focus, but no more
 *    context, except in the minimap maybe one day)
 *  - speedbar for view histories
 *    (could also provide thumbnails on view histories :) )
 *  - zoom and mouse-follow
 *
 * That's lots of features. In a way tools like Powerpoint also provide
 * multiple displays on the same data and with zoomable slides, global
 * view on the set of slides, slides thumnails, etc.
 *)
```

## 1.2 Getting started

```
 * ./configure -visual
 *
 * port install gtk2
 * port install cairo
 * port install freetype
 * port install mysql5-devel
 *
```

## 1.3 Copyright

The source code of pfff is governed by the following copyright:

4 ⟨*Facebook copyright* 4⟩≡ (65 80 92 94b 98b 101 102a)

```
(* Yoann Padioleau
 *
 * Copyright (C) 2010 Facebook
 *
```

```
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public License
 * version 2.1 as published by the Free Software Foundation, with the
 * special exception on linking described in file license.txt.
 *
 * This library is distributed in the hope that it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the file
 * license.txt for more details.
 *)
```

### 1.4 About this document

This document is a literate program [1]. It is generated from a set of files that can be processed by tools (Noweb [2] and syncweb [3]) to generate either this manual or the actual source code of the program. So, the code and its documentation are strongly connected.

## 2 Examples of Use

### 2.1 Viewing the Linux Kernel

Here are the basics: As you move the mouse, the blue highlighted areas are the next level of directories. Double-clicking zooms in on the blue-highlighted area. Right-clicking zoom directly to the file under the cursor. Middle-clicking open the file under the cursor in your emacs provided you have M-x server-start and have emacsclient in your path.

### 2.2 Viewing Pfff Itself

### 2.3 Semantic Visual Feedback

```
% big = use
% green = tested
% purple = bad code

%todo: more scenario/workflow showing cool use of pfff_visual

%sgrep connection ?
```

# 3 Gtk/Cairo

## 3.1 Gtk

## 3.2 Cairo

6a ⟨*new_pixmap sig* 6a⟩≡ (91c)

```
val new_pixmap :
  width:int -> height:int -> GDraw.pixmap
```

6b ⟨*new_pixmap()* 6b⟩≡ (92)

```
let new_pixmap ~width ~height =
  let drawable = GDraw.pixmap ~width ~height () in
  drawable#set_foreground `WHITE ;
  drawable#rectangle
    ~x:0 ~y:0 ~width ~height ~filled:true () ;
  drawable
```

6c ⟨*cairo helpers functions sig* 6c⟩≡ (101b)

```
val fill_rectangle:
  ?alpha:float ->
  cr:Cairo.t ->
  x:float -> y:float -> w:float -> h:float ->
  color:Simple_color.emacs_color ->
  unit ->
  unit

val draw_rectangle_figure:
  cr:Cairo.t ->
  color:Simple_color.emacs_color ->
  Figures.rectangle -> unit

val draw_rectangle_bis:
  cr:Cairo.t ->
  color:Simple_color.color ->
  line_width:float ->
  Figures.rectangle -> unit


val prepare_string : string -> unit
val origin : Cairo.point

val device_to_user_distance_x : Cairo.t -> float -> float
val device_to_user_distance_y : Cairo.t -> float -> float
val user_to_device_distance_x : Cairo.t -> float -> float
val user_to_device_distance_y : Cairo.t -> float -> float
```

```
val device_to_user_size : Cairo.t -> float -> float
val user_to_device_font_size : Cairo.t -> float -> float
val cairo_point_to_point : Cairo.point -> Figures.point

val show_text : Cairo.t -> string -> unit
val text_extents : Cairo.t -> string -> Cairo.text_extents
val set_font_size: Cairo.t -> float -> unit

val clear : Cairo.t -> unit

val surface_of_pixmap :
  < pixmap : [> `drawable ] Gobject.obj; .. > -> [ `Any ] Cairo.surface

val distance_points : Cairo.point -> Cairo.point -> float

val is_old_cairo : unit -> bool
```

7 ⟨*cairo helpers functions* 7⟩≡ (101c)

```
(* work by side effect on the (mutable) string *)
let prepare_string s =
  for i = 0 to String.length s - 1 do
    let c = String.get s i in
    if int_of_char c >= 128
    then String.set s i 'Z';
  done;
  ()

let origin = { Cairo. x = 0.; y = 0. }

let device_to_user_distance_x cr deltax =
  let pt = Cairo.device_to_user_distance cr { origin with Cairo.x = deltax } in
  pt.Cairo.x
let device_to_user_distance_y cr deltay =
  let pt = Cairo.device_to_user_distance cr { origin with Cairo.y = deltay } in
  pt.Cairo.y

let user_to_device_distance_x cr deltax =
  let pt = Cairo.user_to_device_distance cr { origin with Cairo.x = deltax } in
  pt.Cairo.x
let user_to_device_distance_y cr deltay =
  let pt = Cairo.user_to_device_distance cr { origin with Cairo.y = deltay } in
  pt.Cairo.y

(* TODO: this is buggy, as we can move the map which can led to
 * some device_to_user to translate to x = 0
 *)
```

```
let device_to_user_size cr size =
  let device = { Cairo.x = size; Cairo.y = 0.; } in
  let user = Cairo.device_to_user cr device in
  user.Cairo.x

(* still needed ? can just call device_to_user_size ? *)
let user_to_device_font_size cr font_size =
  let user_dist = { Cairo.x = font_size; Cairo.y = font_size } in
  let device_dist = Cairo.user_to_device_distance cr user_dist in
  device_dist.Cairo.x


(* floats are the norm in graphics *)
open Common.ArithFloatInfix

let cairo_point_to_point p =
  { F.x = p.Cairo.x;
    F.y = p.Cairo.y;
  }


let show_text2 cr s =
  (* this 'if' is only for compatibility with old versions of cairo
   * that returns some out_of_memory error when applied to empty strings
   *)
  if s = "" then () else
  try
    prepare_string s;
    Cairo.show_text cr s
  with exn ->
    let status = Cairo.status cr in
    let s2 = Cairo.string_of_status status in
    failwith ("Cairo pb: " ^ s2 ^ " s = " ^ s)

let show_text a b =
  Common.profile_code "View.cairo_show_text" (fun () -> show_text2 a b)

let fake_text_extents =
  { Cairo.
    x_bearing   = 0.1; y_bearing   = 0.1;
    text_width  = 0.1; text_height = 0.1;
    x_advance   = 0.1; y_advance   = 0.1 ;
  }

let text_extents2 cr s =
  (*if s = ""
```

```
  then fake_text_extents
  else
  *)
  Cairo.text_extents cr s

let text_extents a b =
  Common.profile_code "CairoH.cairo_text_extent" (fun () -> text_extents2 a b)


let set_font_size2 cr font_size =
  Cairo.set_font_size cr font_size

let set_font_size cr font_size =
  Common.profile_code "CairoH.set_font_size" (fun () ->
    set_font_size2 cr font_size
  )



(* see http://cairographics.org/FAQ/#clear_a_surface *)
let clear cr =
  Cairo.set_source_rgba cr 0. 0. 0.   0.;
  Cairo.set_operator cr Cairo.OPERATOR_SOURCE;
  Cairo.paint cr;
  Cairo.set_operator cr Cairo.OPERATOR_OVER;
  ()

let surface_of_pixmap pm =
  let cr = Cairo_lablgtk.create pm#pixmap in
  Cairo.get_target cr


let distance_points p1 p2 =
  abs_float (p2.Cairo.x - p1.Cairo.x) +
  abs_float (p2.Cairo.y - p1.Cairo.y) +
    0.

let is_old_cairo () =
  let s = Cairo.compile_time_version_string in
  match () with
  | _ when s =~ "1\\.[89]\\.*" -> false
  | _ -> true

let fill_rectangle ?(alpha=1.) ~cr ~x ~y ~w ~h ~color () =
  (let (r,g,b) = color +> Color.rgbf_of_string in
  Cairo.set_source_rgba cr r g b alpha;
```

```
  );

  Cairo.move_to cr x y;
  Cairo.line_to cr (x+w) y;
  Cairo.line_to cr (x+w) (y+h);
  Cairo.line_to cr x (y+h);
  Cairo.fill cr;
  ()

let draw_rectangle_figure ~cr ~color r =
  (let (r,g,b) = color +> Color.rgbf_of_string in
  Cairo.set_source_rgb cr r g b;
  );
 let line_width = device_to_user_size cr 3. in

  Cairo.set_line_width cr line_width; (* ((r.q.y - r.p.y) / 30.); *)

  Cairo.move_to cr r.p.x r.p.y;
  Cairo.line_to cr r.q.x r.p.y;
  Cairo.line_to cr r.q.x r.q.y;
  Cairo.line_to cr r.p.x r.q.y;
  Cairo.line_to cr r.p.x r.p.y;
  Cairo.stroke cr;
  ()

(* factorize with draw_rectangle. don't use buggy device_to_user_size !!!
*)
let draw_rectangle_bis ~cr ~color ~line_width r =
  (let (r,g,b) =
    color +> Color.rgb_of_color +> Color.rgbf_of_rgb
    in
   Cairo.set_source_rgb cr r g b;
  );
  Cairo.set_line_width cr line_width;

  Cairo.move_to cr r.p.x r.p.y;
  Cairo.line_to cr r.q.x r.p.y;
  Cairo.line_to cr r.q.x r.q.y;
  Cairo.line_to cr r.p.x r.q.y;
  Cairo.line_to cr r.p.x r.p.y;
  Cairo.stroke cr;
  ()
```

**User vs device coordinates**

# 4 Main

11a ⟨*main flags* 11a⟩≡ (102b)

```
let test_mode = ref (None: string option)
let proto = ref false
let screen_size = ref 2
(*
let db_path = ref (Database.database_dir "/home/pad/www")
*)
let db_file = ref (None: Common.filename option)
```

11b ⟨*main_action()* 11b⟩≡ (102b)

```
let main_action xs =
  (*
    GtkMain.Rc.add_default_file "/home/pad/c-pfff/data/pfff_browser.rc";
  *)
  if !Flag.debug_gc
  then Gc.set { (Gc.get()) with Gc.verbose = 0x01F };

  (* see http://www.elehack.net/michael/blog/2010/06/ocaml-memory-tuning *)
  Gc.set { (Gc.get()) with Gc.minor_heap_size = 2_000_000 };
  Gc.set { (Gc.get()) with Gc.space_overhead = 200 };

  let model = () in

  (* the GMain.Main.init () is done by linking with gtkInit.cmo *)
  pr2 (spf "Using Cairo version: %s" Cairo.compile_time_version_string);

  Common.finalize (fun () ->
    View2.mk_gui
      model
      !db_file
      ~screen_size:!screen_size
      !test_mode
      xs
  ) (fun() ->
    ()
  )
```

# 5 The Model

11c ⟨*type model* 11c⟩≡ (91c 92)

```
type model = {
  db: Database_code.database option;
```

```
    ⟨model fields hook 14a⟩
  }
```

12a ⟨*treemap_generator* 12a⟩≡ (94b)

```
let treemap_generator paths =
  let treemap = Treemap_pl.code_treemap paths in
  let algo = Treemap.Ordered Treemap.PivotByMiddle in
  let rects = Treemap.render_treemap_algo ~algo treemap in
  Common.pr2 (spf "%d rectangles to draw" (List.length rects));
  rects
```

12b ⟨*build_model* 12b⟩≡ (94b)

```
let build_model2 root dbfile_opt =
  let db_opt = dbfile_opt +> Common.fmap (fun file ->
    if file =~ ".*.json"
    then Database_code.load_database file
    else Common.get_value file
  )
  in
  let hentities = Model2.hentities root db_opt in
  let hfiles_entities = Model2.hfiles_and_top_entities root db_opt in
  let all_entities = Model2.all_entities db_opt in
  let idx = Completion2.build_completion_defs_index all_entities in

  let model = { Model2.
           db = db_opt;
           hentities = hentities;
           hfiles_entities = hfiles_entities;
           big_grep_idx = idx;
  }
  in
  (*
    let model = Ancient2.mark model in
    Gc.compact ();
  *)
(*
  (* sanity check *)
  let hentities = (Ancient2.follow model).Model2.hentities in
  let n = Hashtbl.length hentities in
  pr2 (spf "before = %d" n);
  let cnt = ref 0 in
  Hashtbl.iter (fun k v -> pr2 k; incr cnt) hentities;
  pr2 (spf "after = %d" !cnt);
  (* let _x = Hashtbl.find hentities "kill" in *)
*)
```

```
    model

  let build_model a b =
    Common.profile_code2 "View.build_model" (fun () ->
      build_model2 a b)
```

13a ⟨*type drawing* 13a⟩≡ (91c 92)

```
  (* All the 'float' below are to be intepreted as user coordinates except when
   * explicitely mentioned. All the 'int' are usually device coordinates.
   *)
  type drawing = {

    (* In user coordinates from 0 to T.xy_ratio and 1 for respectivey x and y.
     * Assumes the treemap contains absolute paths.
    *)
    treemap: Treemap.treemap_rendering;
    (* coupling: = List.length treemap *)
    nb_rects: int;

    (* to compute zoomed treemap when double click *)
    treemap_func: Common.path list -> Treemap.treemap_rendering;

    (* This is to display readable paths. When fully zoomed it's a filename *)
    root: Common.path;

    (* computed lazily *)
    model: model async;

    ⟨fields drawing query stuff 63a⟩

    settings: settings;

    ⟨fields drawing main view 79a⟩

    ⟨fields drawing viewport 61a⟩

    ⟨fields drawing minimap 60b⟩
  }
    ⟨type settings 88a⟩
```

13b ⟨*hentities sig* 13b⟩≡ (91c)

```
  val hentities :
    Common.path -> Database_code.database option ->
    (string, Database_code.entity) Hashtbl.t
```

14a ⟨*model fields hook* 14a⟩≡ (11c) 49c▷

```
(* fast accessors *)
hentities : (string, Database_code.entity) Hashtbl.t;
```

14b ⟨*hentities()* 14b⟩≡ (92)

```
(* We want to display very often used functions in bigger size font.
 * Enter database_code.ml which provides a language-independent database of
 * information on source code.
 *
 * We compute the entities outside init_drawing because
 * init_drawing can be called multiple times (when we zoom in)
 * and we dont want the heavy entities computation to be
 * repeated.
 *)
let hentities root db_opt =
  let hentities = Hashtbl.create 1001 in

  db_opt +> Common.do_option (fun db ->

    let actual_root = actual_root_of_db ~root db in

      (* todo sanity check that db talks about files
       * in dirs_or_files ? Ensure same readable path.
       *)
      db.Db.entities +> Array.iter (fun e ->
        Hashtbl.add hentities
          e.Db.e_name
          {e with Db.e_file =
              Filename.concat actual_root e.Db.e_file
          }
      );
  );
  hentities
```

14c ⟨*init_drawing sig* 14c⟩≡ (91c)

```
val init_drawing :
  ?width:int ->
  ?height:int ->
  ?width_minimap:int ->
  ?height_minimap:int ->
  (Common.path list -> Treemap.treemap_rendering) ->
  model async ->
  Common.filename list ->
  drawing
```

14d ⟨*init_drawing()* 14d⟩≡ (92)

```
let init_drawing
  (* This is a first guess. The first configure ev will force a resize. *)
  ?(width = 600)
  ?(height = 600)
  ?(width_minimap = 60)
  ?(height_minimap = 60)
  func
  model
  paths
 =

  let pm = new_pixmap ~width ~height in
  let paths = paths +> Common.map Common.relative_to_absolute in

  let root = Common.common_prefix_of_files_or_dirs paths in
  pr2_gen root;

  let treemap =
    Common.profile_code2 "Visual.building the treemap" (fun () ->
      func paths
    )
  in
  {
    treemap = treemap;
    nb_rects = List.length treemap;
    root = root;
    treemap_func = func;

    model = model;

    current_query = "";
    current_searched_rectangles = [];
    current_entity = None;
    current_grep_query = Hashtbl.create 0;

    pm = pm;
    overlay = Cairo.surface_create_similar (CairoH.surface_of_pixmap pm)
      Cairo.CONTENT_COLOR_ALPHA width height;

    width = width ;
    height = height ;

    zoom = 1. ;
    xtrans = 0. ;
    ytrans = 0. ;
```

```
    drag_pt = { Cairo.x = 0.0; Cairo.y = 0.0 };
    in_dragging = false;
    in_zoom_incruste = false;

    width_minimap = width_minimap;
    height_minimap = height_minimap;
    pm_minimap = new_pixmap ~width:width_minimap ~height:width_minimap;
    drag_pt_minimap = { Cairo.x = 0.0; Cairo.y = 0.0 };

    settings = {
      (* todo: too fuzzy for now *)
      draw_summary = false;

      draw_searched_rectangles = true;
    };
  }
```

16a ⟨*readable_to_absolute_filename_under_root sig* 16a⟩≡ (91c)

```
val readable_to_absolute_filename_under_root :
  root:Common.path -> string -> string
```

16b ⟨*actual_root_of_db sig* 16b⟩≡ (91c)

```
val actual_root_of_db :
  root:Common.path -> Database_code.database -> string
```

16c ⟨*readable_to_absolute_filename_under_root* 16c⟩≡ (92)

```
(* People may run the visualizer on a subdir of what is mentionned in the
 * database code (e.g. subdir ~/www/flib of ~/www). The light_db
 * contains only readable paths (e.g. flib/foo.php); the reason for
 * those readable paths is that we want to reuse the light_db
 * and share it among multiple users which may have
 * different paths for their own software repo (e.g. ~/www4/).
 *
 * When the user select an entity through the search box,
 * we will know the readable paths of the entity he is looking for
 * but we need a full path for refreshing the treemap.
 * We can not just concatenate the root with the readable paths which
 * in the example would lead to the path  ~/www/flib/flib/foo.php.
 *
 * The goal of the function below is given a readable path like
 * flib/foo.php and a root like ~/www/flib  to recognize the common part
 * and return a valid fullpath like ~/www/flib/foo.php
 *
 *)
let rec readable_to_absolute_filename_under_root ~root filename =
```

```
  (* the root may be a filename *)
  let root_dir =
    if is_directory root then root
    else Filename.dirname root
  in

  let root_and_parents =
    Common.inits_of_absolute_dir root_dir +> List.rev
  in
  try
    root_and_parents +> Common.return_when (fun dir ->
      let path = Filename.concat dir filename in
      if Sys.file_exists path
      then Some path
      else None
    )
  with Not_found ->
    failwith
      (spf "can't find file %s with root = %s" filename root)
```

17a ⟨*actual_root_of_db* 17a⟩≡ (92)

```
let actual_root_of_db ~root db =
  let a_file = (db.Db.entities.(0)).Db.e_file in
  let absolute_file =
    readable_to_absolute_filename_under_root root a_file in

  if absolute_file =~ ("\\(.*\\)/" ^ a_file)
  then Common.matched1 absolute_file
  else failwith (spf "Could not find actual_root of %s under %s: "
                   absolute_file root)
```

# 6 The UI

```
* Overall UI organisation:
*  - menu
*  - toolbar
*  - mainview (treemap | minimap/legend)
*  - statusbar
*
* Conventions and info: see commons/gui.ml
```

17b ⟨*mk_gui sig* 17b⟩≡ (94a)

```
val mk_gui :
  screen_size:int ->
  'a -> Common.filename option -> 'b option ->
```

```
  Common.filename list ->
  unit
```

18 ⟨*view globals* 18⟩≡ (94b)

```
(* when some widgets need to access other widgets *)

(* Note that because we use toplevels 'let' for the GUI elements below,
 * Gtk must have also been initialized via a toplevel element, or
 * initialized by including gtkInit.cmo earlier in the linking command.
 *)

let statusbar =
  GMisc.statusbar ()

let ctx =
  statusbar#new_context "main"

let statusbar_addtext s =
  ctx#push s +> ignore

let title_of_path s = "Pfff_visual: " ^ s

let _set_title = ref (fun s ->
  failwith "_set_title not defined"
)
let _refresh_da = ref (fun () ->
  failwith "_refresh_da not defined"
)

let dw_stack = ref []

let paint_content_maybe_refresher = ref None
let current_rects_to_draw = ref []


let current_r = ref None

let current_motion_refresher = ref None

let interface_doc = "
This tool displays a \"code map\" of a software project using
Treemaps. \"Treemaps display hierarchical (tree-structured) data as a
set of nested rectangles. Each branch of the tree is given a
rectangle, which is then tiled with smaller rectangles representing
sub-branches. A leaf node's rectangle has an area proportional
to a specified dimension on the data.
```

```
\" - http://en.wikipedia.org/wiki/Treemapping:

In our case the dimension is the size of the file.
Moreover each file is colored according to its
\"category\": display code, third party code, etc.
See the legend. We use basic heuristcs based on the
name of the files and directory.

Files and directories are also sorted alphabetically
and partially ordered from top to bottom and left to right.
So a toplevel 'zzz' subdirectory should be located at the bottom
right of the screen.

As you move the mouse, the blue highlighted areas are the next
level of directories.

Double-clicking zooms in on the blue-highlighted area.
Right-clicking zoom directly to the file under the cursor.
Middle-clicking open the file under the cursor in your
favourite editor (provided you have M-x server-start
and have emacsclient in your path).

"
```

19 ⟨*mk_gui()* 19⟩≡ (94b)

```
let mk_gui ~screen_size model dbfile_opt test_mode dirs_or_files =

  let root = Common.common_prefix_of_files_or_dirs dirs_or_files in

  let model = Model2.async_make () in

  let dw = ref (Model2.init_drawing treemap_generator model dirs_or_files) in
  Common.push2 !dw dw_stack;

  let width, height, minimap_hpos, minimap_vpos =
    Style.windows_params screen_size in

  let w = GWindow.window
    ~title:(title_of_path root)
    ~width
    ~height
    ~allow_shrink: true
    ~allow_grow:true
    ()
  in
  _set_title := (fun s -> w#set_title s);
```

```
let accel_group = GtkData.AccelGroup.create () in
w#misc#set_name "main window";

let quit () =
  (*Controller.before_quit_all model;*)
  GMain.Main.quit ();
in

w#add_accel_group accel_group;

(*-----------------------------------------------------------------------*)
(* Layout *)
(*-----------------------------------------------------------------------*)

(* if use my G.mk style for that, then get some pbs when trying
 * to draw stuff :(
 *)
let vbox = GPack.vbox ~packing:w#add () in

  (*---------------------------------------------------------------------*)
  (* Menu *)
  (*---------------------------------------------------------------------*)
  vbox#pack (G.mk (GMenu.menu_bar) (fun m ->

    let factory = new GMenu.factory m in

    factory#add_submenu "_File" +> (fun menu ->
      let fc = new GMenu.factory menu ~accel_group in

      (* todo? open Db ? *)

      fc#add_item "_Open stuff from db" ~key:K._O ~callback:(fun () ->
        ();
      ) +> ignore;
      fc#add_separator () +> ignore;

      fc#add_item "_Quit" ~key:K._Q ~callback:quit;
    ) +> ignore;

    factory#add_submenu "_Edit" +> (fun menu ->
      GToolbox.build_menu menu ~entries:[
        `S;
      ];
    ) +> ignore;
```

```
factory#add_submenu "_Move" +> (fun menu ->
  let fc = new GMenu.factory menu ~accel_group in

  (* todo? open Db ? *)
  fc#add_item "_Go back" ~key:K._B ~callback:(fun () ->
    go_back dw;
  ) +> ignore;

  fc#add_item "_Go to example" ~key:K._E ~callback:(fun () ->
    let model = !dw.model in
    let model = Model2.async_get model in
    match !dw.current_entity, model.db with
    | Some e, Some db ->
        (match e.Db.e_good_examples_of_use with
        | [] -> failwith "no good examples of use for this entity"
        | x::xs ->
            let e = db.Db.entities.(x) in
            let file = e.Db.e_file in

            let final_file =
              Model2.readable_to_absolute_filename_under_root file
                ~root:!dw.root in

            go_dirs_or_file ~current_entity:(Some e) dw [final_file];
        )
    | _ -> failwith "no entity currently selected or no db"
  ) +> ignore ;
);

factory#add_submenu "_Search" +> (fun menu ->
  let fc = new GMenu.factory menu ~accel_group in

  (* todo? open Db ? *)
  fc#add_item "_Git grep" ~key:K._G ~callback:(fun () ->

    let res = dialog_search_def !dw.model in
    res +> Common.do_option (fun s ->
      let root = !dw.root in
      let matching_files = run_grep_query ~root s in
      let files = matching_files +> List.map fst +> Common.uniq in
      let current_grep_query =
        Some (Common.hash_of_list matching_files)
      in
      go_dirs_or_file ~current_grep_query dw files
    );
  ) +> ignore;
```

```
  fc#add_item "_Tbgs query" ~key:K._T ~callback:(fun () ->

    let res = dialog_search_def !dw.model in
    res +> Common.do_option (fun s ->
      let root = !dw.root in
      let matching_files = run_tbgs_query ~root s in
      let files = matching_files +> List.map fst +> Common.uniq in
      let current_grep_query =
        Some (Common.hash_of_list matching_files)
      in
      go_dirs_or_file ~current_grep_query dw files
    );
  ) +> ignore;

);

factory#add_submenu "_Misc" +> (fun menu ->
  let fc = new GMenu.factory menu ~accel_group in

  (* todo? open Db ? *)

  fc#add_item "_Refresh" ~key:K._R ~callback:(fun () ->
    let current_root = !dw.root in
    let _old_dw = Common.pop2 dw_stack in
    (* have to disable the AST caching.
     * todo? disable all entries in the cache ?
     *)
    if Common.is_file current_root
    then Parsing2.disable_file_in_cache current_root;

    go_dirs_or_file dw [current_root];

  ) +> ignore;

  fc#add_item "_Zoom" ~key:K._Z ~callback:(fun () ->
    !dw.in_zoom_incruste <- not (!dw.in_zoom_incruste);
    !_refresh_da();
  ) +> ignore;

);

factory#add_submenu "_Help" +> (fun menu ->
  let fc = new GMenu.factory menu ~accel_group in

  fc#add_item "_Interface" ~key:K._H ~callback:(fun () ->
```

```
          G.dialog_text interface_doc "Help"
        ) +> ignore;

        fc#add_item "_Legend" ~key:K._L ~callback:(fun () ->
          raise Todo
        ) +> ignore;

        fc#add_item "_Help on Pfff" ~callback:(fun () ->
            G.dialog_text "Read\nthe\nsource\n\ndude" "Help"
        ) +> ignore;
        fc#add_separator () +> ignore;
        fc#add_item "About" ~callback:(fun () ->
            G.dialog_text "Brought to you by pad\nwith love" "About"
        ) +> ignore;
      );


    ));

    (*---------------------------------------------------------------------*)
    (* toolbar *)
    (*---------------------------------------------------------------------*)

    vbox#pack (G.mk (GButton.toolbar) (fun tb ->

(*
      tb#insert_widget (G.mk (GButton.button ~stock:`OPEN) (fun b ->
        b#connect#clicked ~callback:(fun () ->
          pr2 "OPEN";
        );
      ));
      tb#insert_widget (G.mk (GButton.button ~stock:`SAVE) (fun b ->
        b#connect#clicked ~callback:(fun () ->
          pr2 "SAVE";
        );
      ));
      tb#insert_space ();
      tb#insert_button ~text:"SAVE THIS" ~callback:(fun () ->
        pr2 "SAVE THIS";
      ) () +> ignore;
      tb#insert_space ();

*)
      let idx = (fun () ->
        let model = Model2.async_get model in
        model.Model2.big_grep_idx
```

```
)
in

let entry =
  Completion2.my_entry_completion_eff
   ~callback_selected:(fun entry str file e ->
    (* pb is that we may have run the visualizer on a subdir
     * of what is mentionned in the database code. We have
     * then to find the real root.
     *)
    entry#set_text "";

    let readable_paths =
      (* hack to handle multidirs *)
      match e.Db.e_kind with
      | Database_code.MultiDirs ->
          (* hack: coupling: with mk_multi_dirs_entity *)
          Common.split "|" e.Db.e_file
      | _ ->
          [e.Db.e_file]
    in

    let final_paths =
      readable_paths +> List.map
        (Model2.readable_to_absolute_filename_under_root ~root:!dw.root)
    in

    pr2 (spf "e= %s, final_paths= %s" str(Common.join "|" final_paths));
    go_dirs_or_file ~current_entity:(Some e) dw final_paths;
    true
  )
  ~callback_changed:(fun str ->
    !dw.current_query <- str;
    !dw.current_searched_rectangles <- [];

    if !dw.settings.draw_searched_rectangles
    then begin
      (* better to compute once the set of matching rectangles
       * cos doing it each time in motify would incur too much
       * calls to ==~
       *)
      let minimum_length = 3 in

      if String.length str > minimum_length then begin

        let rects = !dw.treemap in
```

```
        let re_opt =
          try Some (Str.regexp (".*" ^ str))
         (* can raise exn when have bad or not yet complete regexp *)
          with _ -> None
        in
        let res =
          match re_opt with
          | None -> []
          | Some re ->
              rects +> List.filter (fun r ->
                let label = r.T.tr_label +> String.lowercase in
                label =~ re
              )
        in
        !dw.current_searched_rectangles <- res;

      end;
      let cr_overlay = Cairo.create !dw.overlay in
      CairoH.clear cr_overlay;
      draw_searched_rectangles ~cr_overlay ~dw:!dw;
      !_refresh_da();
    end
  )
  idx
  ;

in


tb#insert_widget (G.with_label "Search:" entry#coerce);

tb#insert_widget (G.mk (GButton.button ~stock:`GO_BACK) (fun b ->
  b#connect#clicked ~callback:(fun () ->
    go_back dw;
  )
));

tb#insert_widget (G.mk (GButton.button ~stock:`GO_UP) (fun b ->
  b#connect#clicked ~callback:(fun () ->
    let current_root = !dw.root in
    go_dirs_or_file dw [Common.dirname current_root];
  )
));

tb#insert_widget (G.mk (GButton.button ~stock:`GOTO_TOP) (fun b ->
  b#connect#clicked ~callback:(fun () ->
```

```
      dw_stack := [Common.last !dw_stack];
      go_back dw;

    )
  ));

));

(*----------------------------------------------------------------------*)
(* main view *)
(*----------------------------------------------------------------------*)

let hpane = GPack.paned `HORIZONTAL
  ~packing:(vbox#pack ~expand:true ~fill:true) () in

let da = GMisc.drawing_area
  ~packing:(hpane#add1) () in
da#misc#set_double_buffered false;

let vpane = GPack.paned `VERTICAL
  ~packing:(hpane#add2) () in
hpane#set_position minimap_hpos;


let da2 = GMisc.drawing_area
  ~packing:(vpane#add1) () in
da2#misc#set_double_buffered false;


let da3 = GMisc.drawing_area
  ~packing:(vpane#add2) () in
vpane#set_position minimap_vpos;

da#misc#set_can_focus true ;
da#event#add [ `KEY_PRESS;
`BUTTON_MOTION; `POINTER_MOTION;
`BUTTON_PRESS; `BUTTON_RELEASE ];

da2#misc#set_can_focus true ;
da2#event#add [ `KEY_PRESS;
             (* weird, but because even if didn't say
              * POINTER_MOTION here, the minimap still
              * gets an event for mouse over :(
              *)
`BUTTON_MOTION; `POINTER_MOTION;
`BUTTON_PRESS; `BUTTON_RELEASE ];
```

```
    da#event#connect#expose ~callback:(expose da dw) +> ignore;
    da#event#connect#configure ~callback:(configure da dw) +> ignore;

    da3#event#connect#expose ~callback:(expose_legend da3 dw) +> ignore;

(*
    da2#event#connect#expose ~callback:(expose_minimap da2 dw) +> ignore;
    da2#event#connect#configure ~callback:(configure_minimap da2 dw) +> ignore;
*)

    da#event#connect#button_press   (button_action da dw) +> ignore;
    da#event#connect#button_release (button_action da dw) +> ignore;
    da#event#connect#motion_notify  (motion_notify (da,da2) dw) +> ignore;

(*
    da2#event#connect#button_press
      (button_action_minimap (da,da2) dw) +> ignore;
    da2#event#connect#button_release
      (button_action_minimap (da, da2) dw) +> ignore;
    da2#event#connect#motion_notify
      (motion_notify_minimap (da,da2) dw) +> ignore;
*)

    _refresh_da := (fun () ->
      GtkBase.Widget.queue_draw da#as_widget;
    );


(*
    da#event#connect#key_press ~callback:(key_pressed da dw);
*)

    (*-----------------------------------------------------------------------*)
    (* status bar *)
    (*-----------------------------------------------------------------------*)
    (* the statusbar widget is defined in beginning of this file because *)
    vbox#pack (*~from: `END*) statusbar#coerce;

  (*  )); *)

  (*-------------------------------------------------------------------------*)
  (* End *)
  (*-------------------------------------------------------------------------*)
```

```
  (* Controller._before_quit_all_func +> Common.push2 Model.close_model; *)

  GtkSignal.user_handler := (fun exn ->
    pr2 "fucking callback";
    (* old: before 3.11: Features.Backtrace.print(); *)
    let s = Printexc.get_backtrace () in
    pr2 s;
    let pb = "pb: " ^ string_of_exn exn in
    G.dialog_text ~text:pb ~title:"pb";
    raise exn
  );

  (* TODO: should do that on 'da', not 'w'
  w#event#connect#key_press ~callback:(key_pressed (da,da2) dw) +> ignore;
  *)

(*
  w#event#connect#key_press ~callback:(fun ev ->
    let k = GdkEvent.Key.keyval ev in
    (match k with
    | _ when k = Char.code 'q' ->
        quit();
        true
    | _ -> false
    )
  );
*)

  w#event#connect#delete    ~callback:(fun _  -> quit(); true) +> ignore;
  w#connect#destroy         ~callback:(fun () -> quit(); ) +> ignore;
  w#show ();

  (* test *)
  test_mode +> Common.do_option (fun s ->
    (* View_test.do_command s model *)
    ()
  );

  (* GMain.Idle.add ~prio: 1000 (idle dw) +> ignore; *)

  (* This can require lots of stack. Make sure to have ulimit -s 40000.
   * This thread also cause some Bus error on MacOS :(
   * so have to use Timeout instead when on the Mac
   *)
  (if CairoH.is_old_cairo()
  then
```

```
    Thread.create (fun () ->
      Model2.async_set (build_model root dbfile_opt) model;
    ) ()
    +> ignore
   else
    Model2.async_set (build_model root dbfile_opt) model;
   (*
    GMain.Timeout.add ~ms:2000 ~callback:(fun () ->
      Model2.async_set (build_model root dbfile_opt) model;
      false
    ) +> ignore
   *)
  );
  GtkThread.main ();
  ()
```

# 7 Treemap View

29 ⟨*paint* 29⟩≡ (94b)

```
let context_of_drawing dw =
  { Draw.
    nb_rects_on_screen = dw.nb_rects;
    model = dw.model;
    settings = dw.settings;
    current_grep_query = dw.current_grep_query;
  }

let paint_content_maybe_rect ~user_rect dw rect =
  let cr = Cairo_lablgtk.create dw.pm#pixmap in
  zoom_pan_scale_map cr dw;

  let context = context_of_drawing dw in

  Draw.draw_treemap_rectangle_content_maybe ~cr  ~clipping:user_rect  ~context
    rect;

  (* have to redraw the label *)
  Draw.draw_treemap_rectangle_label_maybe ~cr ~zoom:dw.zoom ~color:"black"
    rect;
  ()

(* todo: deadlock:  M.locked (fun () ->  ) dw.M.model.M.m *)
let lazy_paint ~user_rect dw () =
  pr2 "Lazy Paint";
  let start = Unix.gettimeofday () in
```

```
  while Unix.gettimeofday () - start < 0.3 do
    match !current_rects_to_draw with
    | [] -> ()
    | x::xs ->
        current_rects_to_draw := xs;
        pr2 (spf "Drawing: %s" (x.T.tr_label));
        paint_content_maybe_rect ~user_rect dw x;
  done;
  !_refresh_da ();
  if !current_rects_to_draw = []
  then false
  else true


let paint2 dw =

  !paint_content_maybe_refresher +> Common.do_option (fun x ->
    GMain.Idle.remove x;
  );
  current_rects_to_draw := [];

  let cr = Cairo_lablgtk.create dw.pm#pixmap in
  dw.pm#rectangle
    ~x:0 ~y:0
    ~width:dw.width ~height:dw.height
    ~filled:true () ;

  pr2 (spf "paint, with zoom = %f, xtrans = %f, ytrans = %f"
          dw.zoom dw.xtrans dw.ytrans);
  let user_rect = device_to_user_area dw in
  pr2 (F.s_of_rectangle user_rect);

  zoom_pan_scale_map cr dw;

  let rects = dw.treemap in
  let nb_rects = dw.nb_rects in

  (* phase 1, draw the rectangles *)
  rects +> List.iter (Draw.draw_treemap_rectangle ~cr);

  (* phase 2, draw the labels, if have enough space *)
  rects +> List.iter
    (Draw.draw_treemap_rectangle_label_maybe ~cr ~zoom:dw.zoom ~color:"black");

  (* phase 3, draw the content, if have enough space *)
  if not dw.in_dragging && nb_rects < !Flag.threshold_nb_rects_draw_content
```

```
    (* draw_content_maybe calls nblines which is quite expensive so
     * want to limit it *)
    then begin
      current_rects_to_draw := rects;
      paint_content_maybe_refresher :=
        Some (GMain.Idle.add ~prio:3000 (lazy_paint ~user_rect dw));
    end;

    (* also clear the overlay *)
    let cr_overlay = Cairo.create dw.overlay in
    CairoH.clear cr_overlay;

    ()

  let paint dw =
    Common.profile_code2 "View.paint" (fun () -> paint2 dw)
```

## 7.1 Treemap Principles

## 7.2 Color code

31a ⟨*expose_legend* 31a⟩≡ (94b)

```
  let expose_legend da dw_ref ev =

    let gwin = da#misc#window in
    let cr = Cairo_lablgtk.create gwin in
    paint_legend ~cr;
    true
```

31b ⟨*paint_legend* 31b⟩≡ (94b)

```
  let paint_legend ~cr =

    Cairo.select_font_face cr "serif"
      Cairo.FONT_SLANT_NORMAL Cairo.FONT_WEIGHT_NORMAL;
    let size = 25.  in

    Cairo.set_font_size cr (size * 0.6);

    Cairo.set_source_rgba cr 0. 0. 0.    1.0;

    let archis = Archi_code.source_archi_list in

    let grouped_archis = archis +> Common.group_by_mapped_key (fun archi ->
      let color = Treemap_pl.color_of_source_archi archi in
      (* I tend to favor the darker variant of the color in treemap_pl.ml hence
       * the 3 below
```

```
   *)
  let color = color ^ "3" in
  color
)
in


grouped_archis +> Common.index_list_1 +> List.iter (fun ((color,kinds), i) ->

  let x = 10. in

  let y = float_of_int i * size in

  let w = size in
  let h = size in

  CairoH.fill_rectangle ~cr ~color ~x ~y ~w ~h ();

  let s =
    kinds +> List.map Archi_code.s_of_source_archi +> Common.join ", " in

  Cairo.set_source_rgba cr 0. 0. 0.    1.0;
  Cairo.move_to cr (x + size * 2.) (y + size * 0.8);
  Cairo.show_text cr s;
);
()
```

## 7.3 Drawing

32a ⟨*device_to_user_area* 32a⟩≡ (94b)

```
(* still needed ? reuse helper functions above ? *)
let device_to_user_area dw =
  with_map dw (fun cr ->

    let device_point = { Cairo. x = 0.0; y = 0.0 } in
    let user_point1 = Cairo.device_to_user cr device_point in
    let device_point = { Cairo.x = float_of_int dw.width;
                         Cairo.y = float_of_int dw.height; } in
    let user_point2 = Cairo.device_to_user cr device_point in

    { F.p = CairoH.cairo_point_to_point user_point1;
      F.q = CairoH.cairo_point_to_point user_point2;
    }
  )
```

32b ⟨*draw_treemap_rectangle sig* 32b⟩≡ (98a)

```
val draw_treemap_rectangle :
  cr:Cairo.t ->
  ?color:Simple_color.emacs_color option ->
  ?alpha:float ->
  Treemap.treemap_rectangle ->
  unit
```

33a ⟨*draw_treemap_rectangle() 33a*⟩≡ (98b)

```
let draw_treemap_rectangle2 ~cr ?(color=None) ?(alpha=1.) rect =
  let r = rect.T.tr_rect in

  (let (r,g,b) =
    let (r,g,b) = rect.T.tr_color +> Color.rgb_of_color +> Color.rgbf_of_rgb in
    match color with
    | None -> (r,g,b)
    | Some c ->
        let (r2,g2,b2) = c +> Color.rgbf_of_string in
        (r2 + r / 20., g2 + g / 20., b2 + b / 20.)
  in
  Cairo.set_source_rgba cr r g b (alpha);
  );

  Cairo.move_to cr r.p.x r.p.y;
  Cairo.line_to cr r.q.x r.p.y;
  Cairo.line_to cr r.q.x r.q.y;
  Cairo.line_to cr r.p.x r.q.y;
  Cairo.fill cr;
  ()

let draw_treemap_rectangle ~cr ?color ?alpha a =
  Common.profile_code "View.draw_treemap_rectangle" (fun () ->
    draw_treemap_rectangle2 ~cr ?color ?alpha a)
```

33b ⟨*draw_treemap_rectangle_label_maybe sig 33b*⟩≡ (98a)

```
val draw_treemap_rectangle_label_maybe :
  cr:Cairo.t ->
  zoom:float ->
  color:Simple_color.emacs_color ->
  Treemap.treemap_rectangle ->
  unit
```

33c ⟨*draw_treemap_rectangle_label_maybe 33c*⟩≡ (98b)

```
let _hmemo_text_extent = Hashtbl.create 101

(* This can be quite cpu intensive. CairoH.text_extents is quite slow
 * so you must avoid calling it too much. A simple optimisation
```

```
 * when the treemap is big is to avoid trying to draw labels
 * that are too tiny already.
 *)
let rec draw_treemap_rectangle_label_maybe2 ~cr ~zoom ~color rect =
  if !Flag.disable_fonts then ()
  else begin

  let lbl = rect.T.tr_label in
  let base = Filename.basename lbl in
  (* old: Common.is_directory_eff lbl *)
  let is_directory = rect.T.tr_is_node in
  let txt =
    if is_directory
    then base ^ "/"
    else base
  in

  Cairo.select_font_face cr "serif"
    Cairo.FONT_SLANT_NORMAL Cairo.FONT_WEIGHT_BOLD;
  let font_size, minus_alpha =
    (match rect.T.tr_depth with
    | 1 -> 0.1, 0.8
    | 2 -> 0.05, 0.2
    | 3 -> 0.03, 0.4
    | 4 -> 0.02, 0.6
    | 5 -> 0.01, 0.7
    | 6 -> 0.005, 0.8
    | _ -> 0.003, 0.9
    )
  in
  let font_size = font_size / (zoom) (* use zoom factor inversely *) in
  let alpha = 1. - (minus_alpha / zoom) in

    try_draw_label
      ~font_size_orig:font_size
      ~color ~alpha
      ~cr ~rect txt
  end


and try_draw_label ~font_size_orig ~color ~alpha ~cr ~rect txt =

  let r = rect.T.tr_rect in

  let w = F.rect_width r in
  let h = F.rect_height r in
```

```
let is_file =
  (* old: try Common.is_file_eff rect.T.tr_label with _ -> false *)
  not rect.T.tr_is_node
in

(let (r,g,b) = color +> Color.rgbf_of_string in
 Cairo.set_source_rgba cr r g b alpha;
);

let rec aux ~font_size ~step =

  (* opti: this avoid lots of computation *)
  let font_size_real = CairoH.user_to_device_font_size cr font_size in

  if font_size_real < !Flag.threshold_draw_label_font_size_real
  then ()
  else begin

   CairoH.set_font_size cr font_size;

   (* opti:
    *  was let extent = CairoH.text_extents cr txt
    *)
   let th, base_tw =
     Common.memoized _hmemo_text_extent (font_size, font_size_real) (fun ()->
       (* peh because it exercises the spectrum of high letters *)
       let extent = CairoH.text_extents cr "peh" in
       let tw = extent.Cairo.text_width in
       let th = extent.Cairo.text_height in
       th, tw
     )
   in
   let tw = float_of_int (String.length txt) * base_tw / 3. in


   (* will try first horizontally at a certain size, then
    * diagonally at a certain size, and if can't then reduce
    * the font_size (up to a certain limit) and try again
    * (first horizontally, then diagonally).
    *)
   match step with
   | 1 | 4 | 7 | 10 when tw < w && th < h && rect.T.tr_depth > 1 ->
         (* see http://cairographics.org/tutorial/#L3showtext
          * for the logic behind the placement of the text
          *)
```

```
      let x = r.p.x + w / 2.0 - (tw / 2.0) in
      let y = r.p.y + h / 2.0 + (th / 2.0) in

      Cairo.move_to cr x y;
      CairoH.show_text cr txt;

| 2 | 5 | 8 | 11 when
      tw < sqrt (w * w + h * h) &&
      th < min w h &&
        rect.T.tr_depth > 1 ->
      (* todo: try vertically ... *)

    (* you have to draw on a paper to understand this code below ... *)

    let tangent = h / w in
    let angle = atan tangent in

    (* right now we don't handle the fact that the text itself has
     * a height but below the x and y positions are just the
     * place of the very bottom of the first letter. In some way we trace a
     * line below the text in the diagonal of the rectangle. This means all
     * the text is on the top of the diagonal. It should be in the middle
     * of the diagonal.
     * As a first fix we can artificially augment the angle ... ugly
     *)
    let angle =
      min (angle + angle / 10.) (Math.pi / 2.)
    in

    (* I love basic math *)
    let x = r.p.x + w / 2.0 - (cos angle * (tw / 2.0)) in
    let y = r.p.y + h / 2.0 - (sin angle * (tw / 2.0)) in
    Cairo.move_to cr x y;


    Cairo.rotate cr ~angle:angle;
    CairoH.show_text cr txt;
    Cairo.rotate cr ~angle:(-. angle);

| 3 ->
    (* I am ok to go down to 70% *)
    let font_size = font_size_orig * 0.7 in
    aux ~step:4 ~font_size
| 6 ->
    (* I am ok to go down to 50% of original *)
```

```
            let font_size = font_size_orig * 0.5 in
            aux ~step:7 ~font_size

        | 9 ->
            (* I am ok to go down to 30% of original for file only  *)
            if is_file
            then
              let font_size = font_size_orig * 0.25 in
              aux ~step:10 ~font_size
            else ()

        (* this case is taken only for the first cases (1, 2, ..) when the
         * associated 'when' expression is false
         *)
        | n ->
            if n >= 12
            then ()
            else aux ~step:(Pervasives.(+) n 1) ~font_size
      end
    in
    aux ~font_size:font_size_orig ~step:1

  let draw_treemap_rectangle_label_maybe ~cr ~zoom ~color rect =
    Common.profile_code "View.draw_label_maybe" (fun () ->
      draw_treemap_rectangle_label_maybe2 ~cr ~zoom ~color rect)
```

# 8 Source View

## 8.1 Principles

## 8.2 Color code

## 8.3 Drawing

37a ⟨*type draw_content_layout* 37a⟩≡ (98b)

```
type draw_content_layout = {
  font_size: float;
  split_nb_columns: float;
  w_per_column:float;
  space_per_line: float;
}
```

37b ⟨*type context* 37b⟩≡ (98)

```
(* a slice of Model2.drawing *)
type context = {
  model: Model2.model Model2.async;
```

```
    settings:Model2.settings;
    nb_rects_on_screen: int;
    current_grep_query: (Common.filename, int) Hashtbl.t;
  }
```

38a ⟨*draw_treemap_rectangle_content_maybe sig* 38a⟩≡ (98a)

```
  val draw_treemap_rectangle_content_maybe :
    cr:Cairo.t ->
    clipping:Figures.rectangle ->
    context:context ->
    Treemap.treemap_rectangle ->
    unit
```

38b ⟨*final_font_size_when_multiplier* 38b⟩≡ (98b)

```
  let final_font_size_when_multiplier
      ~multiplier ~size_font_multiplier_multiplier
      ~font_size ~font_size_real
     =
    let size_font_multiplier = multiplier in

    let font_size_adjusted =
      if size_font_multiplier = 1.
      then font_size
      else
        max
         (font_size * size_font_multiplier * size_font_multiplier_multiplier)
         (font_size * 1.5)
    in

    let final_font_size =
      Common.borne ~min:font_size ~max:(font_size * 30.) font_size_adjusted
    in
    final_font_size
```

38c ⟨*final_font_size_of_categ* 38c⟩≡ (98b)

```
  let final_font_size_of_categ ~font_size ~font_size_real categ =

    let multiplier = Style.size_font_multiplier_of_categ ~font_size_real categ in
    (* as we zoom in, we don't want to be as big, and as
     * we zoom out we want to be bigger
     *)
    let size_font_multiplier_multiplier =
       (*- 0.2 * font_size_real + 2. *)
      match font_size_real with
      | n when n < 3. -> 2.
      | n when n < 8. -> 1.5
```

```
    | n when n < 10. -> 1.
    | _ -> 0.5
  in

  final_font_size_when_multiplier
    ~multiplier
    ~size_font_multiplier_multiplier
    ~font_size
    ~font_size_real
```

39a ⟨*font_size_when_have_x_columns* 39a⟩≡ (98b)

```
let font_size_when_have_x_columns ~nblines ~nbcolumns ~w ~h ~with_n_columns =
  let size_x = (w / with_n_columns) / nbcolumns in
  let size_y = (h / (nblines / with_n_columns)) in

  let min_font = min size_x size_y in
  min_font
```

39b ⟨*optimal_nb_columns* 39b⟩≡ (98b)

```
(* Given a file with nblines and nbcolumns (usually 80) and
 * a rectangle of w width and h height, what is the optimal
 * number of columns. The principle is to start at 1 column
 * and see if by adding columns we can have a bigger font.
 * We try to maximize the font_size.
 *)
let optimal_nb_columns ~nblines ~nbcolumns ~w ~h =

  let rec aux current_font_size current_nb_columns =
    let min_font = font_size_when_have_x_columns
      ~nblines ~nbcolumns ~w ~h ~with_n_columns:current_nb_columns
    in
    if min_font > current_font_size
    then aux min_font (current_nb_columns + 1.)
    else
      (* regression, then go back on step *)
      current_nb_columns - 1.
  in
  aux 0.0   1.
```

39c ⟨*draw_column_bars* 39c⟩≡ (98b)

```
let draw_column_bars2 ~cr ~split_nb_columns ~font_size ~w_per_column rect =
  let r = rect.T.tr_rect in
  for i = 1 to int_of_float (split_nb_columns - 1.) do
    let i = float_of_int i in

    Cairo.set_source_rgba cr 0.0 0.0 1. 0.2;
```

```
    let font_size_real = CairoH.user_to_device_font_size cr font_size in
    let width =
      if font_size_real > 5.
      then  (font_size / 10.)
      else font_size
    in
    Cairo.set_line_width cr width;

    Cairo.move_to cr (r.p.x + w_per_column * i) r.p.y;
    Cairo.line_to cr (r.p.x + w_per_column * i) r.q.y;
    Cairo.stroke cr ;
  done
let draw_column_bars ~cr ~split_nb_columns ~font_size ~w_per_column rect =
  Common.profile_code "View.draw_bars" (fun () ->
    draw_column_bars2 ~cr ~split_nb_columns ~font_size ~w_per_column rect)
```

40 ⟨*draw_content* 40⟩≡ (98b)

```
let draw_content2 ~cr ~layout ~context ~nblines ~file rect =

  let font_size = layout.font_size in
  let split_nb_columns = layout.split_nb_columns in
  let w_per_column  = layout.w_per_column in
  let space_per_line = layout.space_per_line in

  let r = rect.T.tr_rect in
  let font_size_real = CairoH.user_to_device_font_size cr font_size in


  (* highlighting grep-like queries *)
  let matching_lines =
    try Hashtbl.find_all context.current_grep_query file
    with Not_found -> []
  in
  let hmatching_lines = Common.hashset_of_list matching_lines in
  let is_matching_line i =
    Hashtbl.mem hmatching_lines i
  in
  pr2_gen matching_lines;

  let line = ref 1 in

  let nblines_per_column =
    (nblines / split_nb_columns) +> ceil +> int_of_float in

  (* ugly *)
```

```
text_with_user_pos := [];

(match FT.file_type_of_file file with
| (
    FT.PL (FT.Web (FT.Php _))
  | FT.PL (FT.Web (FT.Js _))
  | FT.PL (FT.ML _)
  | FT.PL (FT.Cplusplus | FT.C)
  | FT.PL (FT.Thrift)
  ) ->

  let column = ref 0 in
  let line_in_column = ref 1 in

  let x = r.p.x + (float_of_int !column) * w_per_column in
  let y = r.p.y + (space_per_line * (float_of_int !line_in_column)) in

  Cairo.move_to cr x y;

  let model = Model2.async_get context.model in
  let entities = model.Model2.hentities in

  let tokens_with_categ =
    Parsing.tokens_with_categ_of_file file entities
  in
  if font_size_real > Style.threshold_draw_dark_background_font_size_real
  then begin
    let alpha =
      match context.nb_rects_on_screen with
      | n when n <= 2 -> 0.8
      | n when n <= 10 -> 0.6
      | _ -> 0.3
    in
    draw_treemap_rectangle ~cr ~color:(Some "DarkSlateGray") ~alpha rect;
    CairoH.draw_rectangle_bis ~cr ~color:(rect.T.tr_color)
      ~line_width:font_size rect.T.tr_rect;
  end;

  tokens_with_categ +> List.iter (fun (s, categ, filepos) ->
    let attrs =
      match categ with
      | None -> Highlight_code.info_of_category Highlight_code.Normal
      | Some categ -> Highlight_code.info_of_category categ
    in

    let final_font_size =
```

```
  final_font_size_of_categ ~font_size ~font_size_real categ in
let final_font_size =
  if is_matching_line !line
  then final_font_size * 1.
  else final_font_size
in

let _alpha_adjust =
  let ratio = final_font_size / font_size in
  match ratio with
  | _ when ratio > 4. -> 0.
  | _ when ratio > 2. -> 0.3
  | _ when ratio >= 1. -> 0.5
  | _ -> 0.3
in

Cairo.set_font_size cr final_font_size;

let final_font_size_real =
  CairoH.user_to_device_font_size cr final_font_size in

attrs |> List.iter (fun attr ->
  match attr with
  | `FOREGROUND s ->
      let (r,g,b) = Color.rgbf_of_string s in
      (* this seems needed only on old version of Cario, or at least
       * on the cairo I am using under Linux. Under Mac I don't need
       * this; I put alpha = 1. for everything and the rendering
       * is fine.
       *)
      let alpha =
        if CairoH.is_old_cairo () then
          match () with
          | _ when final_font_size_real < 1. -> 0.2
          | _ when final_font_size_real < 3. -> 0.4
          | _ when final_font_size_real < 5. -> 0.9

          | _ when final_font_size_real < 8.
                -> 1. (* TODO - alpha_adjust, do that only when not in
                         fully zoomed mode *)

          | _ -> 1.
        else 1.
      in
      Cairo.set_source_rgba cr r g b alpha;
      | _ -> ()
```

```
    );

    let xs = Common.lines_with_nl_either s in

    xs +> List.iter (function
    | Left s ->
        let pt = Cairo.get_current_point cr in
        Common.push2 (s, filepos, pt) text_with_user_pos;

        CairoH.show_text cr s
    | Right () ->

        incr line_in_column;
        incr line;

        if !line_in_column > nblines_per_column
        then begin
          incr column;
          line_in_column := 1;
        end;

        let x = r.p.x + (float_of_int !column) * w_per_column in
        let y = r.p.y + (space_per_line * (float_of_int !line_in_column)) in

        (* must be done before the move_to below ! *)
        if is_matching_line !line
        then begin
          CairoH.fill_rectangle ~cr
            ~alpha:0.5
            ~color:"magenta"
            ~x
            ~y:(y - space_per_line)
            ~w:w_per_column
            ~h:(space_per_line * 3.)
            ()
        end;

        Cairo.move_to cr x y;


    );
  )
| FT.PL _ | FT.Text _ ->
 (* This was causing some "out_of_memory" cairo error on linux. Not
  * sure why.
  *)
```

```
    Cairo.set_font_size cr font_size ;
    Cairo.set_source_rgba cr 0.0 0.0 0.0 0.9;

    let xs = Common.cat file in
    let xxs = Common.pack_safe nblines_per_column xs in

    (* I start at 0 for the column because the x displacement
     * is null at the beginning, but at 1 for the line because
     * the y displacement must be more than 0 at the
     * beginning
     *)
    Common.index_list_0 xxs +> List.iter (fun (xs, column) ->
      Common.index_list_1 xs +> List.iter (fun (s, line_in_column) ->

        let x = r.p.x + (float_of_int column) * w_per_column in
        let y = r.p.y + (space_per_line * (float_of_int line_in_column)) in

        Cairo.move_to cr x y;
        CairoH.show_text cr s;

        incr line;
      );
    );
        ()
  | _ ->
        ()
  )

let draw_content ~cr ~layout ~context ~nblines ~file rect =
  Common.profile_code "View.draw_content" (fun () ->
    draw_content2 ~cr ~layout ~context ~nblines ~file rect)
```

44 ⟨*draw_treemap_rectangle_content_maybe* 44⟩≡ (98b)

```
let draw_treemap_rectangle_content_maybe2 ~cr ~clipping ~context rect  =
  let r = rect.T.tr_rect in
  let file = rect.T.tr_label in

  if intersection_rectangles r clipping = None
  then (* pr2 ("not drawing: " ^ file) *) ()
  else begin

  let w = F.rect_width r in
  let h = F.rect_height r in

  (* pr2_gen (dw.path, file); *)
```

```
let fullpath = file in

(* if the file is not textual, or contain weird characters, then
 * it confuses cairo which then can confuse computation done in gtk
 * idle callbacks
 *)
if Common.lfile_exists_eff fullpath && File_type.is_textual_file fullpath
then begin
  let font_size_estimate = h / 100. in
  let font_size_real_estimate =
    CairoH.user_to_device_font_size cr font_size_estimate in
  if font_size_real_estimate > 0.4
  then begin

  (* Common.nblines_with_wc was really slow. fork sucks.
   * alternative: we could store the nblines of a file in the db but
   * we would need a fast absolute_to_readable then.
   *)
  let nblines = Common.nblines_eff fullpath +> float_of_int in
  (* assume our code follow certain conventions. Could infer from file.
   * we should put 80, but a font is higher than large, so
   * I manually readjust things. todo: should readjust something
   * else.
   *)

  let nbcolumns = 41.0 in

  let split_nb_columns =
    optimal_nb_columns ~nblines ~nbcolumns ~h ~w
  in
  let font_size =
    font_size_when_have_x_columns ~nblines ~nbcolumns ~h ~w
      ~with_n_columns:split_nb_columns
  in
  let w_per_column = w / split_nb_columns in

  let space_per_line = font_size in

  (* draw the columns bars *)
  draw_column_bars ~cr ~split_nb_columns ~font_size ~w_per_column rect;


  (* todo: does not work :(
  let font_option = Cairo.Font_Options.make [`ANTIALIAS_SUBPIXEL] in

  (try
```

```
      Cairo.set_font_options cr font_option;
    with exn ->
      let status = Cairo.status cr in
      let s2 = Cairo.string_of_status status in
      failwith s2;
    );
    *)
    Cairo.select_font_face cr "serif"
      Cairo.FONT_SLANT_NORMAL Cairo.FONT_WEIGHT_NORMAL;

    let font_size_real = CairoH.user_to_device_font_size cr font_size in
    (*pr2 (spf "file: %s, font_size_real = %f" file font_size_real);*)

    let layout = {
      font_size = font_size;
      split_nb_columns = split_nb_columns;
      w_per_column = w_per_column;
      space_per_line = space_per_line;
    }
    in

    if font_size_real > !Flag.threshold_draw_content_font_size_real
       && not (is_big_file_with_few_lines ~nblines fullpath)
       && nblines < !Flag.threshold_draw_content_nblines
    then draw_content ~cr ~layout ~context ~nblines ~file:fullpath rect
    else
     if context.settings.draw_summary
     then
       draw_summary_content ~cr ~layout ~context ~nblines ~file:fullpath  rect
    end
  end
  end
let draw_treemap_rectangle_content_maybe ~cr ~clipping ~context rect =
  Common.profile_code "View.draw_content_maybe" (fun () ->
    draw_treemap_rectangle_content_maybe2 ~cr ~clipping ~context rect)
```

# 9 Mixed View

46 ⟨*draw_summary_content* 46⟩≡ (98b)

```
(* todo: should base this on the current max in the current view.
 * Also bad that can not compare function use by just looking
 * at their size :(
*)
let threshold_print_summary_entity_users = 10
```

```
let draw_summary_content2 ~cr ~layout ~context ~nblines ~file rect =

  let w_per_column  = layout.w_per_column in

  let r = rect.T.tr_rect in
  let file = rect.T.tr_label in

  let model = Model2.async_get context.model in

  let files_entities = model.Model2.hfiles_entities in
  let entities =
    try Hashtbl.find files_entities file
    with Not_found -> []
  in
  let entities =
    entities
    +> Common.take_safe 5
    +> Common.filter (fun e ->
      e.Db.e_number_external_users > threshold_print_summary_entity_users
    )
  in

  let _w = F.rect_width r in
  let h = F.rect_height r in

  (* todo: bad to use w cos its give advantage to huge file.
   * w_per_column on the opposite is rougly the same on a screen.
   *)
  let font_size = w_per_column / 2. in

  let font_size_real = CairoH.user_to_device_font_size cr font_size in

  let space_per_line_summary =
    h / 6.
  in
  entities +> Common.index_list_1 +> List.iter (fun (e, i) ->

    let nb_use = e.Db.e_number_external_users in
    (* todo: move this func elsewhere, in database_code ?  *)
    let use_arity = Parsing.use_arity_of_use_count nb_use in

    (* would like to reuse code used when drawing content but
     * I've copy paste to allow specifics anamorphic config
     *)
    let font_size_multiplier =
      (* should reuse Style2.mulitplier_use ? *)
```

```
    match use_arity with
    | HC.HugeUse when nb_use > 1000 -> 4.
    | HC.HugeUse -> 3.
    | HC.LotsOfUse -> 2.
    | HC.MultiUse -> 1.1
    | HC.SomeUse | HC.UniqueUse -> 0.7
    | HC.NoUse -> 0.5
  in
  let size_font_multiplier_multiplier =
    (*- 0.2 * font_size_real + 2. *)
    match font_size_real with
    | n when n < 3. -> 2.
    | n when n < 8. -> 1.5
    | n when n < 10. -> 1.
    | _ -> 0.5
  in

  let final_font_size =
    final_font_size_when_multiplier
      ~multiplier:font_size_multiplier
      ~size_font_multiplier_multiplier
      ~font_size
      ~font_size_real
  in

(* TODO use the appropriate color for entity *)
  (let (r,g,b) =
    0.2, 0.0, 0.0
  in
  let alpha = 0.5 in
  Cairo.set_source_rgba cr r g b alpha;
  );
  Cairo.set_font_size cr final_font_size;

  let x = r.p.x in
  let y = r.p.y + (space_per_line_summary * (float_of_int i)) in

  let str = e.Db.e_name in
  Cairo.move_to cr x y;
  CairoH.show_text cr str;
  let str = spf "(%d)" e.Db.e_number_external_users in
  CairoH.show_text cr str;
);
()
```

```
let draw_summary_content
 ~cr ~layout ~context ~nblines ~file rect =
 Common.profile_code "View.draw_summary_content" (fun () ->
   draw_summary_content2 ~cr ~layout ~context ~nblines ~file rect)
```

49a ⟨*hfiles_and_top_entities sig* 49a⟩≡ (91c)

```
val hfiles_and_top_entities :
  Common.path -> Database_code.database option ->
  (string, Database_code.entity list) Hashtbl.t
```

49b ⟨*hfiles_and_top_entities()* 49b⟩≡ (92)

```
let hfiles_and_top_entities
 root db_opt =

  let hfiles = Hashtbl.create 1001 in

  db_opt +> Common.do_option (fun db ->
    let ksorted =
      Db.build_top_k_sorted_entities_per_file ~k:5 db.Db.entities in
    let actual_root = actual_root_of_db ~root db in

    Hashtbl.iter (fun k v ->
      let k' = Filename.concat actual_root k in
      Hashtbl.add hfiles k' v
    ) ksorted
  );
  hfiles
```

49c ⟨*model fields hook* 14a⟩+≡ (11c) ◁14a 64b▷

```
  hfiles_entities : (Common.filename, Database_code.entity list) Hashtbl.t;
```

# 10 Navigation

49d ⟨*key_pressed* 49d⟩≡ (94b)

```
let key_pressed (da, da2) dw_ref ev =
  let dw = !dw_ref in

  pr2 ("key pressed");

  (* this is in device coordinate, so no need to take into account the zoom *)
  let _delta_move = float dw.width /. 16. in
  let delta_move_user = 0.1 in (* TODO *)

  let delta_zoom = 1.3 in
```

```
let b =
  (match GdkEvent.Key.keyval ev with
  | k when k = K._Left ->
      dw.xtrans <- dw.xtrans +. delta_move_user;
      (* todo opti: *)
      paint dw;
      true
  | k when k = K._Right ->
      dw.xtrans <- dw.xtrans -. delta_move_user;
      (* todo opti: *)
      paint dw;
      true

  | k when k = K._Up ->
      dw.ytrans <- dw.ytrans +. delta_move_user;
      (* todo opti: *)
      paint dw;
      true
  | k when k = K._Down ->
      dw.ytrans <- dw.ytrans -. delta_move_user;
      (* todo opti: *)
      paint dw;
      true

  | k when k = K._plus ->
      dw.zoom <- dw.zoom /. delta_zoom;
      (* can't optimize here, have to paint *)
      paint dw;
      true
  | k when k = K._minus ->
      dw.zoom <- dw.zoom *. delta_zoom;
      (* can't optimize here, have to paint *)
      paint dw;
      true




  | k when k = K._z ->
      dw.in_zoom_incruste <- not (dw.in_zoom_incruste);
      true

  | k when k = K._b ->
      go_back dw_ref;
      true
```

```
      | k when k = K._e ->
          raise Todo

      | k when k = K._q ->
          GMain.quit () ; false

      | _ -> false
      )
    in
    if b then begin
      GtkBase.Widget.queue_draw da#as_widget;
      GtkBase.Widget.queue_draw da2#as_widget;
    end;
    b
```

51 ⟨*find_filepos_in_rectangle_at_user_point* 51⟩≡ (94b)

```
  (* Cannot move this to model.ml because we abuse the drawing function
   * and the Draw.text_with_user_pos 100.
   *)
  let find_filepos_in_rectangle_at_user_point user_pt dw r =
    (* ugly, but if use dw.pm#pixmap directly then it has
     * weird side effects like darking the label
     *)
    let sur =
      Cairo.surface_create_similar (CairoH.surface_of_pixmap dw.pm)
        Cairo.CONTENT_COLOR_ALPHA dw.width dw.height
    in
    let cr = Cairo.create sur in

    zoom_pan_scale_map cr dw;
    let user_rect = device_to_user_area dw in

    let context = context_of_drawing dw in
    let context = { context with Draw.nb_rects_on_screen = 1 } in

    (* does side effect on Draw.text_with_user_pos *)
    Draw.draw_treemap_rectangle_content_maybe ~cr ~clipping:user_rect ~context r;
    let xs = !Draw.text_with_user_pos in

    let scores = xs
      +> List.map (fun (s, filepos, pt) ->
        (s, filepos), CairoH.distance_points user_pt pt
      )
      +> Common.sort_by_val_lowfirst
      +> List.map fst
    in
```

```
    (match scores with
    | [] ->
        pr2 ("no filepos found");
        None
    | (s, filepos)::xs ->
        pr2 (spf "closest point is: %s at %d:%d"
                s filepos.Common.l filepos.Common.c);
        Some filepos
    )
```

52 ⟨*button_action* 52⟩≡ (94b)

```
  let button_action da dw_ref ev =
    let dw = !dw_ref in

    let pt = { Cairo. x = GdkEvent.Button.x ev; y = GdkEvent.Button.y ev;} in
    let user = with_map dw (fun cr -> Cairo.device_to_user cr pt) in

    let r_opt = Model2.find_rectangle_at_user_point dw user in

    match GdkEvent.get_type ev with
    | `BUTTON_PRESS ->

        let button = GdkEvent.Button.button ev in
        pr2 (spf "button %d pressed" button);

        (match button with
        | 1 ->

  (* DISABLED FOR NOW
            dw.drag_pt <- {
              Cairo.x = GdkEvent.Button.x ev;
              Cairo.y = GdkEvent.Button.y ev;
            };
            dw.in_dragging <- true;
  *)

            r_opt +> Common.do_option (fun (r, _, _r_englobing) ->
              let file = r.T.tr_label in
              pr2 (spf "clicking on %s" file);
            );
            true
        | 2 ->
            r_opt +> Common.do_option (fun (r, _, _r_englobing) ->
              let file = r.T.tr_label in
              pr2 (spf "opening %s" file);
```

```
          match find_filepos_in_rectangle_at_user_point user dw r with
          | None ->
              Editor_connection.open_file_in_current_editor ~file ~line:0;
          | Some (fpos) ->
              Editor_connection.open_file_in_current_editor ~file
                ~line:fpos.Common.l;
        );
        true

    | 3 ->

    r_opt +> Common.do_option (fun (r, _, _r_englobing) ->
      let path = r.T.tr_label in

      go_dirs_or_file dw_ref [path];
    );
    true
    | _ -> false
    )
| `BUTTON_RELEASE ->

    let button = GdkEvent.Button.button ev in
    pr2 (spf "button %d released" button);

    (match button with
    | 1 ->
        dw.in_dragging <- false;

        GtkBase.Widget.queue_draw da#as_widget;
        true
    | _ -> false
    )

| `TWO_BUTTON_PRESS ->
    pr2 ("double click");

    r_opt +> Common.do_option (fun (_r, _, r_englobing) ->
      let path = r_englobing.T.tr_label in
      go_dirs_or_file dw_ref [path];
    );

    true
| _ -> false
```

## 10.1 Zooming In

54 ⟨*go_dirs_or_file* 54⟩≡ (94b)

```
let go_dirs_or_file ?(current_entity=None) ?(current_grep_query=None)
  dw_ref paths =

  let root = Common.common_prefix_of_files_or_dirs paths in
  pr2 (spf "zooming in %s" (Common.join "|" paths));

  (* reset the painter ? not needed because will call draw below
   * which will reset it
   *)

  let dw = !dw_ref in
  !_set_title (title_of_path root);

  (* the release event will arrive later on the new dw so
   * has to first set the context good for the old dw
   *)
  !dw_ref.in_dragging <- false;

  Common.push2 !dw_ref dw_stack;

  dw_ref :=
    Model2.init_drawing
      ~width:dw.width
      ~height:dw.height
      ~width_minimap:dw.width_minimap
      ~height_minimap:dw.height_minimap
      dw.treemap_func
      dw.model
      paths;
  !dw_ref.current_entity <- current_entity;
  (match current_grep_query with
  | Some h ->
      !dw_ref.current_grep_query <- h;
  | None ->
      (* wants to propagate the query so when right-click the query
       * is still there
       *)
      !dw_ref.current_grep_query <- dw.current_grep_query;
  );
  paint !dw_ref;
  !_refresh_da ();
  ()
```

55a ⟨*find_rectangle_at_user_point sig* 55a⟩≡ (91c)

```
val find_rectangle_at_user_point :
  drawing ->
  Cairo.point ->
  (Treemap.treemap_rectangle *
   Treemap.treemap_rectangle list *
   Treemap.treemap_rectangle
  )
  option
```

55b ⟨*find_rectangle_at_user_point()* 55b⟩≡ (92)

```
(* alt: could use Cairo_bigarray and the pixel trick if
 * it takes too long to detect which rectangle is under the cursor.
 * coud also sort the rectangles ... or have some kind of BSP.
 *)
let find_rectangle_at_user_point2 dw user =
  let user = CairoH.cairo_point_to_point user in

  let rects = dw.treemap in
  if List.length rects = 1
  then
    (* we are fully zommed, this treemap will have tr_depth = 1 but we return
     * it *)
    let x = List.hd rects in
    Some (x, [], x)
  else
   let matching_rects = rects
    +> List.filter (fun r ->
      F.point_is_in_rectangle user r.T.tr_rect
      && r.T.tr_depth > 1
    )
    +> List.map (fun r -> r, r.T.tr_depth)
    +> Common.sort_by_val_highfirst
    +> List.map fst
   in
   match matching_rects with
   | [] -> None
   | [x] -> Some (x, [], x)
   | _ -> Some (Common.head_middle_tail matching_rects)


let find_rectangle_at_user_point a b =
  Common.profile_code "Model.find_rectangle_at_point" (fun () ->
    find_rectangle_at_user_point2 a b)
```

## 10.2 Zooming Fully

## 10.3 Editor Connection

56a ⟨*editor_connection.mli* 56a⟩≡
```
val open_file_in_current_editor : file:string -> line:int -> unit
```

56b ⟨*emacs configuration* 56b⟩≡ (102a)
```
let emacsclient_path_mac =
  "/home/pad/Dropbox/apps/Emacs.app/Contents/MacOS/bin/emacsclient"

let emacsclient_path = "emacsclient"

(* you need to have done a M-x server-start first *)
let run_emacsclient ~file ~line =
  Common.command2 (spf "%s -n %s" emacsclient_path file);
  Common.command2 (spf
    "%s -e '(with-current-buffer (window-buffer (selected-window)) (goto-line %d))'"
    emacsclient_path line);
  ()
```

56c ⟨*open_file_in_current_editor()* 56c⟩≡ (102a)
```
let open_file_in_current_editor ~file ~line =
  run_emacsclient ~file ~line
```

56d ⟨*text_with_user_pos sig* 56d⟩≡ (98a)
```
(* ugly: used when middle-clicking on the drawing area to know
 * how to translate a point into a filepos so that we can open
 * the file at the right position.
 *)
val text_with_user_pos :
  (string * Common.filepos * Cairo.point) Common.stack ref
```

## 10.4 Going Back

56e ⟨*go_back* 56e⟩≡ (94b)
```
let go_back dw_ref =

  (* reset also the motion notifier ? less needed because
   * the next motion will reset it
   *)
  !paint_content_maybe_refresher +> Common.do_option (fun x ->
    GMain.Idle.remove x;
  );

  let old_dw = Common.pop2 dw_stack in
  dw_ref := old_dw;
```

```
let path = !dw_ref.root in
!_set_title (title_of_path path);

!_refresh_da();
()
```

## 10.5 Incruste

57a ⟨*idle* 57a⟩≡ (94b)

```
let idle dw () =
  let dw = !dw in

  (*pr2 "idle";*)
  !current_r +> Common.do_option (fun r ->
    (* will compute and cache *)
    if dw.in_zoom_incruste
    then zoomed_surface_of_rectangle dw r +> ignore;
  );
  true
```

## 10.6 Minimap

57b ⟨*motion_notify_minimap* 57b⟩≡ (94b)

```
let motion_notify_minimap (da, da2) dw ev =
  let dw = !dw in

  let x = GdkEvent.Motion.x ev in
  let y = GdkEvent.Motion.y ev in
  pr2 ("motion minimap");

  if dw.in_dragging then begin

    let deltax = x -. dw.drag_pt_minimap.Cairo.x in
    let deltay = y -. dw.drag_pt_minimap.Cairo.y in

    let deltax_user =
      with_minimap dw (fun cr -> CairoH.device_to_user_distance_x cr deltax)
    in
    let deltay_user =
      with_minimap dw (fun cr -> CairoH.device_to_user_distance_y cr deltay)
    in
    dw.xtrans <- dw.xtrans -. deltax_user;
    dw.ytrans <- dw.ytrans -. deltay_user;
```

```
    (* pr2_gen (deltax, deltay); *)
    dw.drag_pt_minimap <- { Cairo.x = x ; Cairo.y = y } ;

    (* TODO: opti, should not recompute when just move! *)
    paint dw;
    GtkBase.Widget.queue_draw da#as_widget;
    GtkBase.Widget.queue_draw da2#as_widget;

    true
  end else begin
    true
  end
```

58a ⟨*button_action_minimap* 58a⟩≡ (94b)

```
let button_action_minimap (da,da2) dw ev =
  let dw = !dw in

  match GdkEvent.get_type ev with
  | `BUTTON_PRESS ->
      pr2 ("button pressed minimap");

      dw.drag_pt_minimap <- {
        Cairo.x = GdkEvent.Button.x ev ;
        Cairo.y = GdkEvent.Button.y ev
      };
      dw.in_dragging <- true;

      true

  | `BUTTON_RELEASE ->
      pr2 ("button released minimap");
      dw.in_dragging <- false;

      (* TODO: opti *)
      paint dw;

      GtkBase.Widget.queue_draw da#as_widget;
      GtkBase.Widget.queue_draw da2#as_widget;
      true
  | _ -> false
```

58b ⟨*paint_minimap* 58b⟩≡ (94b)

```
let paint_minimap2 dw =
  let cr = Cairo_lablgtk.create dw.pm_minimap#pixmap in
  dw.pm_minimap#rectangle
```

```
    ~x:0 ~y:0
    ~width:dw.width_minimap ~height:dw.height_minimap
    ~filled:true () ;

  scale_minimap cr dw;

  let rects = dw.treemap in
  (* draw the rectangles *)
  rects +> List.iter (Draw.draw_treemap_rectangle ~cr);

  (* draw the labels, if have enough space *)
  rects +> List.iter
    (Draw.draw_treemap_rectangle_label_maybe ~cr ~zoom:1.0 ~color:"black");

  (* draw the zoom rectangle *)
  let user_rect = device_to_user_area dw in
  CairoH.draw_rectangle_figure ~cr ~color:"white" user_rect;
  ()


let paint_minimap dw =
  Common.profile_code2 "View.paint minimap" (fun () -> paint_minimap2 dw)
```

59a ⟨*expose_minimap* 59a⟩≡ (94b)

```
let expose_minimap da dw_ref ev =
  let dw = !dw_ref in

  (* todo? opti? don't paint if not needed ? *)
  (*paint_minimap dw;*)

  let area = GdkEvent.Expose.area ev in
  let x = GR.x area in
  let y = GR.y area in
  let width = GR.width area in
  let height = GR.height area in

  let gwin = da#misc#window in
  let d = new GDraw.drawable gwin in

  Common.profile_code2 "View.put_pixmap mini" (fun () ->
    d#put_pixmap ~x ~y ~xsrc:x ~ysrc:y ~width ~height
      dw.pm_minimap#pixmap;
  );
  true
```

59b ⟨*configure_minimap* 59b⟩≡ (94b)

```
let configure_minimap da2 dw_ref ev =
  let dw = !dw_ref in

  let w = GdkEvent.Configure.width ev in
  let h = GdkEvent.Configure.height ev in
  dw.width_minimap <- w;
  dw.height_minimap <- h;
  dw.pm_minimap <- new_pixmap dw.width_minimap dw.height_minimap;
  true
```

60a ⟨*with_minimap* 60a⟩≡ (94b)

```
let with_minimap dw f =
  let cr = Cairo_lablgtk.create dw.pm_minimap#pixmap in
  scale_minimap cr dw;
  f cr
```

60b ⟨*fields drawing minimap* 60b⟩≡ (13a)

```
  (* minimap *)
  mutable pm_minimap: GDraw.pixmap;
  mutable width_minimap: int;
  mutable height_minimap: int;

  mutable drag_pt_minimap: Cairo.point;
```

60c ⟨*scale_minimap* 60c⟩≡ (94b)

```
let scale_minimap cr dw =
  (* no zoom, no pan, no clippnig *)
  Cairo.translate cr 0.0 0.0;
  Cairo.scale cr
    (1.0 * (float_of_int dw.width_minimap / T.xy_ratio))
    (1.0 * (float_of_int dw.height_minimap))
```

## 10.7 Fine Grained Pan and Zoom

60d ⟨*with_map* 60d⟩≡ (94b)

```
let with_map dw f =
  let cr = Cairo_lablgtk.create dw.pm#pixmap in
  zoom_pan_scale_map cr dw;
  f cr
```

60e ⟨*zoom_pan_scale_map* 60e⟩≡ (94b)

```
let zoom_pan_scale_map cr dw =
  Cairo.scale cr
    (dw.zoom * (float_of_int dw.width / T.xy_ratio))
    (dw.zoom * (float_of_int dw.height))
  ;
```

```
(* I first scale and then translate as the xtrans are in user coordinates *)
Cairo.translate cr dw.xtrans dw.ytrans;
(* TODO clipping  Cairo.rectangle cr ~x:dw.xtrans ~y: *)
()
```

61a ⟨*fields drawing viewport* 61a⟩≡ (13a)

```
(* viewport, device coordinates *)
mutable width: int;
mutable height: int;

mutable zoom: float;

(* in user coordinates *)
mutable xtrans: float;
mutable ytrans: float;

mutable drag_pt: Cairo.point;
mutable in_dragging: bool;

mutable in_zoom_incruste: bool;
```

# 11 Search

61b ⟨*dialog_search_def* 61b⟩≡ (94b)

```
let dialog_search_def model =
  let idx = (fun () ->
    let model = Model2.async_get model in
    model.Model2.big_grep_idx
  )
  in
  let entry =
    Completion2.my_entry_completion_eff
      ~callback_selected:(fun entry str file e ->
        true
      )
      ~callback_changed:(fun str ->
        ()
      )
      idx
  in

  let res =
    G.dialog_ask_generic ~title:""
      (fun vbox ->
        vbox#pack (G.with_label "search:" entry#coerce);
```

```
      )
      (fun () ->
        let text = entry#text in
        pr2 text;
        text
      )
  in
  res +> Common.do_option (fun s ->
    pr2 ("selected: " ^ s);
  );
  res
```

62a ⟨*run_grep_query* 62a⟩≡ (94b)

```
let run_grep_query ~root s =
  (* --cached so faster ? use -w ?
   * -I means no search for binary files
   * -n to show also line number
   *)
  let git_grep_options =
    "-I -n"
  in
  let cmd =
    spf "cd %s; git grep %s %s" root git_grep_options s
  in
  let xs = Common.cmd_to_list cmd in
  let xs = xs +> List.map (fun s ->
    if s =~ "\\([^:]*\\):\\([0-9]+\\):.*"
    then
      let (filename, lineno) = Common.matched2 s in
      let lineno = s_to_i lineno in
      let fullpath = Filename.concat root filename in
      fullpath, lineno
    else
      failwith ("wrong git grep line: " ^ s)
  ) in
  xs
```

62b ⟨*run_tbgs_query* 62b⟩≡ (94b)

```
let run_tbgs_query ~root s =
  let cmd =
    spf "cd %s; tbgs --stripdir %s" root s
  in
  let xs = Common.cmd_to_list cmd in
  let xs = xs +> List.map (fun s ->
    if s =~ "\\([^:]*\\):\\([0-9]+\\):.*"
    then
```

```
        let (filename, lineno) = Common.matched2 s in
        let lineno = s_to_i lineno in
        let fullpath = Filename.concat root filename in
        fullpath, lineno
      else
        failwith ("wrong tbgs line: " ^ s)
  ) in
  xs
```

63a ⟨*fields drawing query stuff* 63a⟩≡ (13a)

```
  (* queries *)
  mutable current_query: string;
  mutable current_searched_rectangles: Treemap.treemap_rectangle list;
  mutable current_entity: Database_code.entity option;
  mutable current_grep_query :
    (Common.filename, int) Hashtbl.t;
```

## 11.1 Definition Search

```
%tags-like

% php manual integration!
```

63b ⟨*all_entities sig* 63b⟩≡ (91c)

```
(* Will generate extra entities for files, dirs, and also generate
 * an extra entity when have a fullname that is not empty
 *)
val all_entities :
  Database_code.database option ->
  Database_code.entity list
```

## 11.2 Completion Building

63c ⟨*completion2.mli* 63c⟩≡ 64a ▷

```
val build_completion_defs_index :
  Database_code.entity list -> Big_grep.index
```

63d ⟨*build_completion_defs_index* 63d⟩≡ (65)

```
(* I was previously using a prefix-clustering optimisation but it
 * does not allow suffix search. Moreover it was still slow so
 * big_grep is just simpler and better.
 *)

let build_completion_defs_index all_entities =
  (* todo? compute stuff in background ?
```

```
 *  Thread.create (fun () ->
 * while(true) do
 * Thread.delay 4.0;
 * pr2 "thread1";
 * done
 * ) ();
 *)
BG.build_index all_entities
```

## 11.3 Completion Window

64a ⟨*completion2.mli* 63c⟩+≡ ◁63c

```
val my_entry_completion_eff :
  callback_selected:
   (GEdit.entry -> string -> string -> Database_code.entity -> bool) ->
  callback_changed:(string -> unit) ->
  (unit -> Big_grep.index) ->
  GEdit.entry
```

64b ⟨*model fields hook* 14a⟩+≡ (11c) ◁49c

```
    big_grep_idx: Big_grep.index;
```

```
%tags-like

% for func/class/methods/ files and dirs!
```

64c ⟨*all_entities* 64c⟩≡ (92)

```
(* We want to provide completion not only for functions/class/methods
 * but also for files and directory themselves.
 *
 * We should maybe pass files_and_dirs in addition to the db_opt
 * because sometimes we don't have a db but we could still provide
 * completion for the dirs and files.
 *
 * todo: what do do when the root of the db is not the root
 * of the treemap ?
 *)
let all_entities db_opt =
  match db_opt with
  | None -> []
  | Some db ->
      let nb_entities = Array.length db.Db.entities in
      let nb_files = List.length db.Db.files in
      pr2 (spf "We got %d entities in %d files" nb_entities nb_files);
```

```
      Database_code.files_and_dirs_and_sorted_entities_for_completion
        ~threshold_too_many_entities:!Flag.threshold_too_many_entities
        db
```

65 ⟨*completion2.ml* 65⟩≡

```
⟨Facebook copyright 4⟩

open Common

module G = Gui

module Db = Database_code
module BG = Big_grep

module Flag = Flag_visual

(* to optimize the completion by using a specialized fast ocaml-based model *)
open Custom_list_generic

(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)

(*
 * Gtk is quite "fragile". You change what looks to be an innocent line
 * and then suddenly your performance goes down or you get some
 * gtk warnings at runtime. So take care when changing this file.
 *)

(*****************************************************************************)
(* Helpers *)
(*****************************************************************************)

let is_prefix2 s1 s2 =
  (String.length s1 <= String.length s2) &&
  (String.sub s2 0 (String.length s1) = s1)
let is_prefix a b =
  Common.profile_code "Completion.is_prefix" (fun () -> is_prefix2 a b)


(*****************************************************************************)
(*  *)
(*****************************************************************************)

⟨build_completion_defs_index 63d⟩
```

```
(*****************************************************************************)
(* Model *)
(*****************************************************************************)

let icon_of_kind kind has_test =
  match kind with
  | Db.Function ->
      if has_test then `YES else `NO

  (* TODO: do different symbols for unit tested class and methods ?
   * or add another column in completion popup
   *)
  | Db.Class -> `CONNECT
  | Db.Module -> `DISCONNECT

  | Db.Type -> `PROPERTIES

  | Db.Constant -> `CONNECT
  | Db.Global -> `MEDIA_RECORD

  | Db.Method -> `CONVERT
  | Db.StaticMethod -> `EXECUTE

  | Db.File -> `FILE
  | Db.Dir -> `DIRECTORY
  | Db.MultiDirs -> `QUIT


module L=struct
  type t = {
    mutable entity: Database_code.entity;
    mutable text: string;
    mutable file: string;
    mutable count: string;
    mutable kind: string;
    mutable icon: GtkStock.id;
  }

  (** The columns in our custom model *)
  let column_list = new GTree.column_list ;;
  let col_full = (column_list#add Gobject.Data.caml: t GTree.column);;

  let col_text = column_list#add Gobject.Data.string;;
  let col_file = column_list#add Gobject.Data.string;;
  let col_count = column_list#add Gobject.Data.string;;
```

```
  let col_kind = column_list#add Gobject.Data.string;;
  let col_icon = column_list#add GtkStock.conv;;

  let custom_value _ t ~column =
    match column with
    | 0 -> (* col_full *) `CAML (Obj.repr t)

    | 1 -> (* col_text *) `STRING (Some t.text)
    | 2 -> (* col_file *) `STRING (Some t.file)
    | 3 -> (* col_count *) `STRING (Some t.count)
    | 4 -> (* col_kind *) `STRING (Some t.kind)

    (* pad: big hack, they use STRING to present stockid in gtkStock.ml *)
    | 5 -> (* col_icon *) `STRING (Some (GtkStock.convert_id t.icon))
    | _ -> assert false

end

module MODEL=MAKE(L)

let model_of_list_pair_string_with_icon2 query xs =

  let custom_list = MODEL.custom_list () in

  pr2 (spf "Size of model = %d" (List.length xs));
  xs +> List.iter (fun e ->
    let kind = e.Db.e_kind in

    let has_unit_test =
      List.length e.Db.e_good_examples_of_use >= 1
    in
    let name = e.Db.e_name in
    custom_list#insert {L.
      entity = e;

      (* had originally an ugly hack where we would artificially create
       * a text2 field with always set to query. Indeed
       * gtk seems to be confused if the column referenced
       * by set_text_column contains a string that is not matching
       * the current query. So here we were building this fake text entry.
       * In fact as explained on the pygtk entry of entry_completion
       * you don't have to use set_text_column if you provide
       * your own set_match_func, which we do.
       * Maybe we should just not use Entrycompletion at all and build
       * our own popup.
       *)
```

```
      text = name;


      file = e.Db.e_file;
      count = i_to_s (e.Db.e_number_external_users);
      kind = Db.string_of_entity_kind kind;
      icon = icon_of_kind kind has_unit_test;
    };
  );
  (custom_list :> GTree.model)


let model_of_list_pair_string_with_icon query a =
  Common.profile_code2 "Completion2.model_of_list" (fun () ->
    model_of_list_pair_string_with_icon2 query a
  )

let model_col_of_prefix prefix_or_suffix idx =
  let xs =
    BG.top_n_search
      ~top_n:!Flag.top_n
      ~query:prefix_or_suffix idx
  in
  model_of_list_pair_string_with_icon prefix_or_suffix xs

(*****************************************************************************)
(* Main entry point *)
(*****************************************************************************)

let add_renderer (completion : GEdit.entry_completion) =

  let renderer =
    GTree.cell_renderer_pixbuf [ `STOCK_SIZE `BUTTON ] in
  completion#pack (renderer :> GTree.cell_renderer);
  completion#add_attribute (renderer :> GTree.cell_renderer)
    "stock_id" L.col_icon;

  let renderer = GTree.cell_renderer_text [] in
  completion#pack (renderer :> GTree.cell_renderer);
  completion#add_attribute (renderer :> GTree.cell_renderer)
    "text" L.col_text;

  let renderer = GTree.cell_renderer_text [] in
  completion#pack (renderer :> GTree.cell_renderer);
  completion#add_attribute (renderer :> GTree.cell_renderer)
    "text" L.col_count;
```

```
  let renderer = GTree.cell_renderer_text [] in
  completion#pack (renderer :> GTree.cell_renderer);
  completion#add_attribute (renderer :> GTree.cell_renderer)
    "text" L.col_file;

  (* can omit this:
   *    completion#set_text_column L.col_text2;
   *
   * but then must define a set_match_func otherwise will never
   * see a popup
   *)
  ()

let fake_entity = {Database_code.
     e_name = "foobar";
     e_fullname = "";
     e_file = "foo.php";
     e_kind = Db.Function;
     e_pos = { Common.l = -1; Common.c = -1 };
     e_number_external_users = 0;
     e_good_examples_of_use = [];
}

let my_entry_completion_eff2 ~callback_selected ~callback_changed fn_idx =

  let entry = GEdit.entry ~width:900 () in
  let completion = GEdit.entry_completion () in
  entry#set_completion completion;

  let xs = [ fake_entity ] in
  let model_dumb = model_of_list_pair_string_with_icon "foo" xs in
  let model = ref (model_dumb) in

  add_renderer completion;
  completion#set_model (!model :> GTree.model);

  (* we don't use the builtin gtk completion mechanism as we
   * recompute the model each time using big_grep so where
   * we just always return true. Moreover the builtin gtk
   * function would do a is_prefix check between the row
   * and the current query which in our case would fail when
   * we use the suffix-search ability of big_grep.
   *)
  completion#set_match_func (fun key row ->
    true
```

```
);
completion#set_minimum_key_length 2;

completion#connect#match_selected (fun model_filter row ->
   (* note: the code below does not work; the row is relative to the
    * model_filter.
    * let str = !model#get ~row ~column:col1 in
    * let file = !model#get ~row ~column:col2 in
    *)

    let str =
      model_filter#child_model#get
        ~row:(model_filter#convert_iter_to_child_iter row)
        ~column:L.col_text
    in
    let file =
      model_filter#child_model#get
        ~row:(model_filter#convert_iter_to_child_iter row)
        ~column:L.col_file
    in
    let t =
      model_filter#child_model#get
        ~row:(model_filter#convert_iter_to_child_iter row)
        ~column:L.col_full
    in
    callback_selected entry str file t.L.entity
) +> ignore;

let current_timeout = ref None in

entry#connect#changed (fun () ->
  let s = entry#text in
  pr2 s;
  if s <> "" then begin
    !current_timeout +> Common.do_option (fun x ->
      GMain.Timeout.remove x;
    );
    current_timeout :=
      Some
        (GMain.Timeout.add ~ms:250
         ~callback:(fun _ ->

          pr2 "changing model";
          let idx = fn_idx () in
          model := model_col_of_prefix s idx;
          completion#set_model (!model :> GTree.model);
```

```
            callback_changed s;
            false
          ));
    end
    else callback_changed s
  ) +> ignore;

  (* return the entry so someone can hook another signal *)
  entry

let my_entry_completion_eff ~callback_selected ~callback_changed x =
  my_entry_completion_eff2 ~callback_selected ~callback_changed x
```

### 11.4 Use Search, aka Visual grep

```
%cscope-like
```

### 11.5 Directory search

```
%multi dirs
```

### 11.6 Example search

```
%test search

% pleac integration!
```

## 12 Assembling Views

71 ⟨*motion_refresher* 71⟩≡ (94b)

```
(* todo: deadclock   M.locked (fun () ->    ) dw.M.model.m *)
let motion_refresher ev dw () =
  let cr_overlay = Cairo.create dw.overlay in
  CairoH.clear cr_overlay;

  let x = GdkEvent.Motion.x ev in
  let y = GdkEvent.Motion.y ev in

  let pt = { Cairo. x = GdkEvent.Motion.x ev; y = GdkEvent.Motion.y ev;} in
  let user = with_map dw (fun cr -> Cairo.device_to_user cr pt) in

  let r_opt = find_rectangle_at_user_point dw user in
```

```
  r_opt +> Common.do_option (fun (r, middle, r_englobing) ->
    let txt = r.T.tr_label in
    statusbar_addtext txt;

    draw_label_overlay ~cr_overlay ~dw ~x ~y r;
    draw_rectangle_overlay ~cr_overlay ~dw (r, middle, r_englobing);

    if dw.settings.draw_searched_rectangles;
    then
        draw_searched_rectangles ~cr_overlay ~dw;

    current_r := Some r;

    (* it has been computed, use it then *)
    if Hashtbl.mem _hmemo_surface (r.T.tr_label, dw.zoom) &&
      dw.in_zoom_incruste
    then
      draw_zoomed_overlay ~cr_overlay ~user ~dw ~x ~y r;

  );
  !_refresh_da ();
  false


let motion_notify (da, da2) dw ev =

  !current_motion_refresher +> Common.do_option (fun x ->
    GMain.Idle.remove x;
  );

  let dw = !dw in

  let x = GdkEvent.Motion.x ev in
  let y = GdkEvent.Motion.y ev in
  pr2 (spf "motion: %f, %f" x y);

  if dw.in_dragging then begin

    let deltax = x -. dw.drag_pt.Cairo.x in
    let deltay = y -. dw.drag_pt.Cairo.y in

    let deltax_user =
      with_map dw (fun cr -> CairoH.device_to_user_distance_x cr deltax)
    in
    let deltay_user =
      with_map dw (fun cr -> CairoH.device_to_user_distance_y cr deltay)
```

```
  in

  dw.xtrans <- dw.xtrans +. deltax_user;
  dw.ytrans <- dw.ytrans +. deltay_user;

  dw.drag_pt <- { Cairo.x = x ; Cairo.y = y } ;

  GtkBase.Widget.queue_draw da#as_widget;
  GtkBase.Widget.queue_draw da2#as_widget;

  true
end else begin
  current_motion_refresher :=
    Some (GMain.Idle.add ~prio:100 (motion_refresher ev dw));
  true
end
```

73a ⟨*draw_searched_rectangles* 73a⟩≡ (94b)

```
let draw_searched_rectangles ~cr_overlay ~dw =
  Cairo.save cr_overlay;
  zoom_pan_scale_map cr_overlay dw;

  dw.current_searched_rectangles +> List.iter (fun r ->
    CairoH.draw_rectangle_figure ~cr:cr_overlay
      ~color:"yellow" r.T.tr_rect
  );
  (*
   * would also like to draw not matching rectangles
   * bug the following code is too slow on huge treemaps.
   * Probably because it is doing lots of drawing and alpha
   * computation.
   *
   * old:
   * let color = Some "grey3" in
   * Draw.draw_treemap_rectangle ~cr:cr_overlay
   * ~color ~alpha:0.3
   * r
   *)
  Cairo.restore cr_overlay;
  ()
```

73b ⟨*zoomed_surface_of_rectangle* 73b⟩≡ (94b)

```
let _hmemo_surface = Hashtbl.create 101
let zoomed_surface_of_rectangle dw r =
  Common.memoized _hmemo_surface (r.T.tr_label, dw.zoom) (fun () ->
```

```
let user_rect = device_to_user_area dw in

let sur =
  Cairo.surface_create_similar (CairoH.surface_of_pixmap dw.pm)
    (* subtle: can not use dw.width or dw.height here because at
     * the zoom level we will proceed, the whole file would probably not
     * feel on the full screen. If it does not fit, then having a too
     * small surface mean parts of the rendering of the file will not
     * be stored.
     *)
    Cairo.CONTENT_COLOR_ALPHA 9000 9000;
in
let cr = Cairo.create sur in

(* simplify the drawing context, draw on 0 x 0 a rectangle that itself
 * starts at 0 x 0
 *)
let dw' = { dw with
  zoom = dw.zoom * Style.zoom_factor_incruste_mode; (* CONFIG *)
  xtrans = 0.; ytrans = 0.;
}
in
zoom_pan_scale_map cr dw';
(* a normalized rectangle that starts at 0 x 0 *)
let rect = r.T.tr_rect in
let rect' = {
  F.p = { F. x = 0.; y = 0.;};
  F.q = { F. x = F.rect_width rect; y = F.rect_height rect;};
}
in
let r' = { r with T.tr_rect = rect' } in

let user_width = F.rect_width rect in
let user_height = F.rect_height rect in

let device_width  = CairoH.user_to_device_distance_x cr user_width in
let device_height = CairoH.user_to_device_distance_y cr user_height in
(* now have on the surface the same thing we would have got if we had
 * zoomed a lot.
 *)

let context = context_of_drawing dw in
let context = { context with Draw.nb_rects_on_screen = 1 } in

Draw.draw_treemap_rectangle ~cr ~alpha:0.9 r';
Draw.draw_treemap_rectangle_content_maybe ~cr ~context ~clipping:user_rect r';
```

```
  sur, device_width, device_height
  )



let draw_zoomed_overlay ~cr_overlay ~user ~dw ~x ~y r =

  let percent_x =
    (user.Cairo.x - r.T.tr_rect.p.F.x) / F.rect_width r.T.tr_rect in
  let percent_y =
    (user.Cairo.y - r.T.tr_rect.p.F.y) / F.rect_height r.T.tr_rect in

  let zoomed_surface, zoomed_device_width, zoomed_device_height =
    zoomed_surface_of_rectangle dw r
  in
  Cairo.set_operator cr_overlay Cairo.OPERATOR_OVER;
  (* old:
     Cairo.set_source_surface cr_overlay zoomed_surface (x - 100.) (y - 100.);
     Cairo.paint cr_overlay;
  *)
  (* see http://cairographics.org/FAQ/#paint_from_a_surface *)
  let dest_x = (x + 20.) in
  let dest_y = (y + 20.) in
  let width = float_of_int dw.width / 2.5 in
  let height = float_of_int dw.height / 2.5 in
  let source_x =
    Common.borne
      ~min:0. ~max:(zoomed_device_width - width)
      ((percent_x * zoomed_device_width) - 140.)
  in
  let source_y =
    Common.borne
      ~min:0. ~max:(zoomed_device_height - height)
      ((percent_y * zoomed_device_height) - 30.)
  in

  pr2 (spf "at x%%= %.3f, y%% = %.3f, zoom_w = %.3f, zoom_h = %.3f"
          percent_x percent_y
          zoomed_device_width
          zoomed_device_height
  );

  Cairo.set_source_surface cr_overlay zoomed_surface
    (dest_x -. source_x) (dest_y -. source_y);
  Cairo.rectangle cr_overlay dest_x dest_y width height;
  Cairo.fill cr_overlay;
```

```
  ()
```

76a ⟨*assemble_layers* 76a⟩≡ (94b)

```
  (* Composing the "layers". See cairo/tests/knockout.ml example.
   * Each move of the cursor will call assemble_layers which does all
   * those pixels copying but this is fast enough.
   *)
  let assemble_layers cr_final dw ~width ~height =

    let surface_src = CairoH.surface_of_pixmap dw.pm in

    Cairo.set_operator cr_final Cairo.OPERATOR_OVER;
    Cairo.set_source_surface cr_final surface_src 0. 0.;
    Cairo.paint cr_final;

    Cairo.set_operator cr_final Cairo.OPERATOR_OVER;
    Cairo.set_source_surface cr_final dw.overlay 0. 0.;
    Cairo.paint cr_final;
    ()
```

76b ⟨*expose* 76b⟩≡ (94b)

```
  let expose2 da dw_ref ev =
    let dw = !dw_ref in

    (* opti: don't 'paint dw;' if not needed! painting is the computation
     * heavy function. expose just copy the "canvas" layers
     *)

    (* todo? equivalent to
     *  let allocation = d_area#misc#allocation in
     * allocation.Gtk.width allocation.Gtk.height
     * ?
     *)
    let area = GdkEvent.Expose.area ev in
    let width = GR.width area +> float_of_int in
    let height = GR.height area +> float_of_int in
    (* todo? use ? it can optimise things ? *)
    let _x = GR.x area in
    let _y = GR.y area in

    let gwin = da#misc#window in
    let cr = Cairo_lablgtk.create gwin in
    assemble_layers cr dw ~width ~height;
    (* old:
    Common.profile_code "View.put_pixmap" (fun () ->
      let d = new GDraw.drawable gwin in
```

```
    d#put_pixmap ~x ~y ~xsrc:x ~ysrc:y ~width ~height dw.pm#pixmap;
  );
  *)
  true

let expose a b c =
  Common.profile_code2 "View.expose" (fun () -> expose2 a b c)
```

77a ⟨*configure* 77a⟩≡ (94b)

```
let configure2_bis da dw_ref ev =
  let dw = !dw_ref in

  let w = GdkEvent.Configure.width ev in
  let h = GdkEvent.Configure.height ev in

  dw.width <- w;
  dw.height <- h;
  dw.pm <- Model2.new_pixmap dw.width dw.height;
  let cr_src = Cairo_lablgtk.create dw.pm#pixmap in
  let sur_src = Cairo.get_target cr_src in
  dw.overlay <-
    Cairo.surface_create_similar sur_src
    Cairo.CONTENT_COLOR_ALPHA w h;

  paint dw;
  true

(* ugly: for some unknown reason configure get called twice at
 * the beginning of the program
 *)
let first_call = ref true
let configure2 a b c =
  (* should probably do is_old_gtk() *)
  if !first_call && CairoH.is_old_cairo ()
  then begin first_call := false; true end
  else
    configure2_bis a b c

let configure a b c =
  Common.profile_code2 "View.configure" (fun () -> configure2 a b c)
```

77b ⟨*draw_label_overlay* 77b⟩≡ (94b)

```
let draw_label_overlay ~cr_overlay ~dw ~x ~y r =

  let txt = r.T.tr_label in
```

```
let readable_txt =
  if dw.root = txt (* when we are fully zoomed on one file *)
  then "root"
  else
    Common.filename_without_leading_path dw.root txt in

Cairo.select_font_face cr_overlay "serif"
  Cairo.FONT_SLANT_NORMAL Cairo.FONT_WEIGHT_NORMAL;
Cairo.set_source_rgba cr_overlay 1. 1. 1.    1.0;
Cairo.set_font_size cr_overlay 40.;

let extent = CairoH.text_extents cr_overlay readable_txt in
let tw = extent.Cairo.text_width in
let _th = extent.Cairo.text_height in


Cairo.move_to cr_overlay (x - tw / 2.) (y);
CairoH.show_text cr_overlay readable_txt;

(*
Cairo.set_source_rgb cr_overlay 0.3 0.3 0.3;
Cairo.move_to cr_overlay x y;
Cairo.line_to cr_overlay (x + 10.) (y + 10.);
Cairo.stroke cr_overlay;
*)
()
```

## 12.1 Layering

78 ⟨*draw_rectangle_overlay* 78⟩≡ (94b)

```
let draw_rectangle_overlay ~cr_overlay ~dw (r, middle, r_englobing) =
  Cairo.save cr_overlay;
  zoom_pan_scale_map cr_overlay dw;
  CairoH.draw_rectangle_figure ~cr:cr_overlay ~color:"white" r.T.tr_rect;

  CairoH.draw_rectangle_figure
    ~cr:cr_overlay ~color:"blue" r_englobing.T.tr_rect;
  Draw.draw_treemap_rectangle_label_maybe
    ~cr:cr_overlay ~color:"blue" ~zoom:dw.zoom r_englobing;

  middle +> Common.index_list_1 +> List.iter (fun (r, i) ->
    let color =
      match i with
      | 1 -> "grey70"
      | 2 -> "grey40"
```

```
      | _ -> spf "grey%d" (max 1 (50 -.. (i *.. 10)))
    in
    CairoH.draw_rectangle_figure
      ~cr:cr_overlay ~color r.T.tr_rect;
    Draw.draw_treemap_rectangle_label_maybe
      ~cr:cr_overlay ~color ~zoom:dw.zoom r;
  );

  Cairo.restore cr_overlay;
  ()
```

### 12.2 Assembling

79a ⟨*fields drawing main view* 79a⟩≡ (13a)

```
  (* device coordinates *)
  mutable pm: GDraw.pixmap;
  (* todo: going from a point to the enclosing rectangle via pixel color
     trick. Kind of ugly.
     mutable pm_color_trick: GDraw.pixmap;
     mutable pm_color_trick_info: (string) array.
     alternative: just find pixel by iterating over all the rectangles
      and check if he's inside
  *)
  mutable overlay: [ `Any ] Cairo.surface;
```

## 13 Language Modes

79b ⟨*parsing2.mli* 79b⟩≡

```
val tokens_with_categ_of_file :
  Common.filename ->
  (string, Database_code.entity) Hashtbl.t ->
  (string * Highlight_code.category option * Common.filepos) list

val use_arity_of_use_count : int -> Highlight_code.use_arity

type ast =
    Php of Parse_php.program2
  | ML of Parse_ml.program2
  | Cpp of Parse_cpp.program2
  | Js of Parse_js.program2
val _hmemo_file : (Common.filename, ast) Hashtbl.t
val disable_file_in_cache :
  Common.filename -> unit
```

80 ⟨*parsing2.ml* 80⟩≡

⟨*Facebook copyright* 4⟩

```
open Common

module FT = File_type

module HC = Highlight_code
module Db = Database_code

open Highlight_code

(*****************************************************************************)
(* Parsing helpers *)
(*****************************************************************************)

(* This type is needed if we want to use a single hashtbl to memoize
 * all the parsed file. Having a single hash helps for
 * disable_file_in_cache below.
 *)
type ast =
  | Php of Parse_php.program2
  | ML  of Parse_ml.program2
  | Cpp of Parse_cpp.program2
  | Js  of Parse_js.program2

let _hmemo_file = Hashtbl.create 101

let parse_php2 file =
  Common.memoized _hmemo_file file (fun () ->
    let (ast2, stat) = Parse_php.parse file in
    let ast = Parse_php.program_of_program2 ast2 in
    (* work by side effect on ast2 too *)
    Check_variables_php.check_and_annotate_program
      ast;
    Php ast2
  )
let parse_php_cache a =
  Common.profile_code "View.parse_php_cache" (fun () ->
    match parse_php2 a with | Php a -> a | _ -> raise Impossible
  )

let parse_ml2 file =
  Common.memoized _hmemo_file file (fun () ->
    ML (Parse_ml.parse file +> fst))
let parse_ml_cache a =
```

```
  Common.profile_code "View.parse_ml_cache" (fun () ->
    match parse_ml2 a with | ML a -> a | _ -> raise Impossible
  )


let parse_cpp2 file =
  Common.memoized _hmemo_file file (fun () ->
    Cpp (Parse_cpp.parse_tokens file +> fst))
let parse_cpp_cache a =
  Common.profile_code "View.parse_ml_cache" (fun () ->
    match parse_cpp2 a with | Cpp a -> a | _ -> raise Impossible
  )

let parse_js2 file =
  Common.memoized _hmemo_file file (fun () ->
    Js (Parse_js.parse file +> fst))
let parse_js_cache a =
  Common.profile_code "View.parse_ml_cache" (fun () ->
    match parse_js2 a with | Js a -> a | _ -> raise Impossible
  )



let disable_file_in_cache file =
  Hashtbl.remove _hmemo_file file


(*****************************************************************************)
(* Semantic ehancement *)
(*****************************************************************************)

let use_arity_of_use_count n =
  match () with
  (* note that because my PHP object analysis have some threshold
   * on the number of callers (see threshold_callers_indirect_db)
   * the number for HugeUse can not be more than this one otherwise
   * you will miss some cases
   *)
  | _ when n >= 100 -> HugeUse

  | _ when n > 20 -> LotsOfUse
  | _ when n >= 10  -> MultiUse
  | _ when n >= 2 -> SomeUse
  | _ when n = 1 -> UniqueUse
  | _ -> NoUse
```

```
let rewrite_categ_using_entities s categ file entities =

  let e_kind_opt =
    try Some (Db.entity_kind_of_highlight_category categ)
    with _ -> None
  in
  match e_kind_opt with
  | None -> categ
  | Some e_kind ->


   let entities =
    Hashtbl.find_all entities s +> List.filter (fun e ->
      (* we could have the full www dbcode but run the treemap on
       * a subdir in which case the root will not be the same.
       * It's a good approximation to just look at the basename.
       * The only false positive we will get if another file,
       * with the same name happened to also define entities
       * with the same name, which would be rare.
       *
       * update: TODO use Model2.readable_to_absolute_filename_under_root ?
       *)
      Filename.basename e.Db.e_file = Filename.basename file &&
      (* some file have both a function and class with the same name *)
      e.Db.e_kind = e_kind
    )
  in
  match entities with
  | [] -> categ
  | [e] ->
      let use_cnt = e.Db.e_number_external_users in
      let arity = use_arity_of_use_count use_cnt in
      (* coupling: you must also say in style2 to take into account
       * the arity for the category
       *)
      (match categ with
      | Function (Def2 _) ->
          Function (Def2 arity)
      | FunctionDecl _ ->
          FunctionDecl (arity)
      | Class (Def2 _) ->
          Class (Def2 arity)
      | Method (Def2 _) ->
          Method (Def2 arity)
      | StaticMethod (Def2 _) ->
          StaticMethod (Def2 arity)
```

```
      | _ -> categ
      )
  | x::y::xs ->
      (* TODO: handle __construct directly *)
      if not (List.mem s ["__construct"])
      then
        pr2_once (spf "multi def found for %s in %s" s file);
      categ

(*****************************************************************************)
(* Main entry point *)
(*****************************************************************************)

(* coupling: right now if you add a language here, you need to whitelist it
 * also in draw2.draw_contents2.
 *
 * todo: ugly, lots of repetitive code. If factorize code in
 * parse_info.ml can at least factorize some of the Ast_xxx.str_of_xxx.
 *)
let tokens_with_categ_of_file file hentities =
  let ftype = FT.file_type_of_file file in
  let prefs = Highlight_code.default_highlighter_preferences in


  match ftype with
  | FT.PL (FT.Web (FT.Php _)) ->
      let h = Hashtbl.create 101 in

      let ast2 = parse_php_cache file in
      ast2 +> List.map (fun (ast, (_str, toks)) ->
        (* computing the token attributes *)
        Highlight_php.visit_toplevel
          ~tag:(fun info categ -> Hashtbl.add h info categ)
          ~maybe_add_has_type_icon:(fun a b c -> ())
          prefs
          None
          (ast, toks)
        ;

        (* getting the text *)
        toks |> Common.map_filter (fun tok ->
          let info = Token_helpers_php.info_of_tok tok in
          let s = Token_helpers_php.str_of_tok tok in

          if not (Ast_php.is_origintok info)
          then None
```

```
        else
          let categ = Common.hfind_option info h in
          let categ = categ +> Common.fmap (fun categ ->
              rewrite_categ_using_entities s categ file hentities
            )
          in
          Some (s, categ,
               { l = Ast_php.line_of_info info;
                 c = Ast_php.col_of_info info;
               })
      )
    ) +> List.flatten

| FT.PL (FT.ML _) ->
    let h = Hashtbl.create 101 in

    let ast2 = parse_ml_cache file in
    ast2 +> List.map (fun (ast, (_str, toks)) ->
      (* computing the token attributes *)
      Highlight_ml.visit_toplevel
        ~tag_hook:(fun info categ -> Hashtbl.add h info categ)
        prefs
        (ast, toks)
      ;

      (* getting the text *)
      toks |> Common.map_filter (fun tok ->
        let info = Token_helpers_ml.info_of_tok tok in
        let s = Token_helpers_ml.str_of_tok tok in

        if not (Ast_ml.is_origintok info)
        then None
        else
          let categ = Common.hfind_option info h in
          let categ = categ +> Common.fmap (fun categ ->
              rewrite_categ_using_entities s categ file hentities
            )
          in
          Some (s, categ,
               { l = Ast_ml.line_of_info info;
                 c = Ast_ml.col_of_info info;
               })

      )
    ) +> List.flatten
```

```
| FT.PL (FT.Cplusplus _ | FT.C | FT.Thrift) ->
    let h = Hashtbl.create 101 in

    let ast2 = parse_cpp_cache file in
    ast2 +> List.map (fun (ast, (_str, toks)) ->
      (* computing the token attributes *)
      Highlight_cpp.visit_toplevel
        ~tag_hook:(fun info categ -> Hashtbl.add h info categ)
        prefs
        (ast, toks)
      ;

      (* getting the text *)
      toks |> Common.map_filter (fun tok ->
        let info = Token_helpers_cpp.info_of_tok tok in
        let s = Token_helpers_cpp.str_of_tok tok in

        if not (Ast_cpp.is_origintok info)
        then None
        else
          let categ = Common.hfind_option info h in
          let categ = categ +> Common.fmap (fun categ ->
              rewrite_categ_using_entities s categ file hentities
            )
          in
          Some (s, categ,
               { l = Ast_cpp.line_of_info info;
                 c = Ast_cpp.col_of_info info;
               })

      )
    ) +> List.flatten

| FT.PL (FT.Web (FT.Js _)) ->

    let h = Hashtbl.create 101 in

    let ast2 = parse_js_cache file in
    ast2 +> List.map (fun (ast, (_str, toks)) ->
      (* computing the token attributes *)
      Highlight_js.visit_toplevel
        ~tag_hook:(fun info categ -> Hashtbl.add h info categ)
        prefs
        (ast, toks)
      ;
```

```
        (* getting the text *)
        toks |> Common.map_filter (fun tok ->
          let info = Token_helpers_js.info_of_tok tok in
          let s = Token_helpers_js.str_of_tok tok in
          let s = Ast_js.remove_quotes_if_present s in

          if not (Ast_js.is_origintok info)
          then None
          else
            let categ = Common.hfind_option info h in
            let categ = categ +> Common.fmap (fun categ ->
                rewrite_categ_using_entities s categ file hentities
              )
            in
            Some (s, categ,
                 { l = Ast_js.line_of_info info;
                   c = Ast_js.col_of_info info;
                 })

        )
      ) +> List.flatten



  | _ -> failwith
      "impossible: should be called only when file has good file_kind"
```

### 13.1 OCaml

### 13.2 PHP

### 13.3 C/C++ and variants

### 13.4 Javascript

### 13.5 Tex/Latex/NoWeb

## 14 Optimisations

### 14.1 Threads, Idle, Timeouts

86 ⟨*type async* 86⟩≡ (91c 92)

```
type 'a async = {
  m: Mutex.t;
  c: Condition.t;
  v: 'a option ref;
  }
```

87a ⟨*async functions sig* 87a⟩≡ (91c)

```
val async_get: 'a async -> 'a
val async_make: unit -> 'a async
val async_set: 'a -> 'a async -> unit

val locked: (unit -> 'a) -> Mutex.t -> 'a
```

87b ⟨*async functions* 87b⟩≡ (92)

```
let async_make () = {
  m = Mutex.create ();
  c = Condition.create ();
  v = ref None;
}

let locked f l =
  Mutex.lock l;
  try
    let x = f () in
    Mutex.unlock l;
    x
  with e ->
    Mutex.unlock l;
    raise e

let async_get a =
  let rec go a =
    match !(a.v) with
    | None ->
        pr2 "not yet computed";
        Condition.wait a.c a.m;
        go a
    | Some v -> v
  in
  locked (fun () -> go a) a.m

let async_set v a =
  locked (fun () ->
    a.v := Some v;
    Condition.signal a.c;
  ) a.m
```

# 15 Configuration

87c ⟨*options* 87c⟩≡ (102b)

```
    "-screen_size" , Arg.Set_int screen_size,
```

```
    " <int> 1 = small, 2 = big";
    "-ss" , Arg.Set_int screen_size,
    " alias for -screen_size";
    "-ft", Arg.Set_float Flag.threshold_draw_content_font_size_real,
    " ";

    "-filter", Arg.String (fun s -> Flag.extra_filter := Some s),
    " ";
    "-with_info", Arg.String (fun s -> db_file := Some s),
    " ";

    "-test" , Arg.String (fun s -> test_mode := Some s),
    " <str> execute an internal script";
    "-proto" , Arg.Set proto,
    " ";

    "-verbose" , Arg.Set Flag.verbose_visual,
    " ";
    "-debug_gc", Arg.Set Flag.debug_gc,
    " ";
   "-disable_ancient", Arg.Clear Flag.use_ancient,
    " ";
   "-enable_ancient", Arg.Set Flag.use_ancient,
    " ";
   "-disable_fonts", Arg.Set Flag.disable_fonts,
    " ";
```

88a ⟨*type settings* 88a⟩≡ (13a)

```
and settings = {
  mutable draw_summary: bool;
  mutable draw_searched_rectangles: bool;
}
```

88b ⟨*style2.mli* 88b⟩≡

```
val windows_params : int -> int * int * int * int

val size_font_multiplier_of_categ :
  font_size_real:float -> Highlight_code.category option -> float

val threshold_draw_dark_background_font_size_real : float

val zoom_factor_incruste_mode : float
```

88c ⟨*windows_params()* 88c⟩≡ (101a)

```
let windows_params screen_size =
```

```
  let width, height, minimap_hpos, minimap_vpos =
    match screen_size with
    | 1 ->
        1350, 800, 1100, 150
    | 2 ->
        2560, 1580, 2350, 100 (* was 2200 and 280 *)
    | 3 ->
        7000, 4000, 6900, 100
    | 4 ->
        16000, 9000, 15900, 100
    | 5 ->
        20000, 12000, 19900, 100
    | 6 ->
        25000, 15000, 24900, 100
    | _ ->
        failwith "not valid screen_size"
  in
  width, height, minimap_hpos, minimap_vpos
```

89 ⟨*size_font_multiplier_of_categ()* 89⟩≡ (101a)

```
  let multiplier_use x =
    match x with
    | SH.HugeUse -> 3.3
    | SH.LotsOfUse -> 2.7
    | SH.MultiUse -> 2.1
    | SH.SomeUse -> 1.7
    | SH.UniqueUse -> 1.3
    | SH.NoUse -> 0.9

  let size_font_multiplier_of_categ ~font_size_real categ =
      match categ with
      | Some (SH.Class SH.Def2 use) -> 6. *. multiplier_use use
      | Some (SH.Module SH.Def) -> 6.
      | Some (SH.Function (SH.Def2 use)) -> 3.5 *. multiplier_use use
      | Some (SH.TypeDef SH.Def) -> 6.
      | Some (SH.Global (SH.Def2 _)) -> 3.
      | Some (SH.FunctionDecl use) -> 2.5 *. multiplier_use use
      | Some (SH.Macro (SH.Def2 _)) -> 2.
      | Some (SH.MacroVar (SH.Def2 _)) -> 2.
      | Some (SH.Method (SH.Def2 use)) -> 3.5 *. multiplier_use use
      | Some (SH.StaticMethod (SH.Def2 use)) -> 3.5 *. multiplier_use use


      | Some (SH.Method (SH.Use2 _)) when font_size_real > 7.
            -> 1.5
```

```
    | Some (SH.CommentSection0) -> 5.
    | Some (SH.CommentSection1) -> 4.
    | Some (SH.CommentSection2) -> 3.5
    | Some (SH.CommentEstet) -> 1.0
    | Some (SH.CommentCopyright) -> 0.5

  (*
    | Some (SH.Comment) when font_size_real > 7.
        -> 1.5
  *)

    | Some (SH.BadSmell) -> 2.5
    | Some (SH.UseOfRef) -> 2.

    | _ -> 1.
```

90a ⟨*zoom_factor_incruste_mode* 90a⟩≡ (101a)

```
  (* TODO: should be automatically computed. Should have instead a
   * wanted_real_font_size_when_incruste_mode = 9.
   *)
  let zoom_factor_incruste_mode = 10. (* was 18 *)
```

90b ⟨*threshold_draw_dark_background_font_size_real* 90b⟩≡ (101a)

```
  (* CONFIG *)
  let threshold_draw_dark_background_font_size_real = 1.
```

90c ⟨*flag_visual.ml* 90c⟩≡

```
  let verbose_visual = ref false

  (* was 0.4, but on linux the anti-aliasing seems to not be as good
   * as on mac (possibly because I have only an old cairo lib on my
   * Linux machine
   *)
  let threshold_draw_content_font_size_real = ref
    0.6

  (* big and auto-generated files can take too much time to render *)
  let threshold_draw_content_nblines =
    ref 10000.

  let threshold_draw_label_font_size_real = ref
    10.

  let threshold_nb_rects_draw_content = ref 2500
```

```
let threshold_too_many_entities = ref 300000

let top_n = ref 100

let debug_gc = ref false

(* Ancient does not interact well with hashtbl and ocaml polymorphic
 * equality and hash. Have to use a functorized hashtbl which sucks.
 *)
let use_ancient = ref false

let disable_fonts = ref false

let extra_filter = ref (None: string option) (* regexp *)
```

# 16 Other Features

91a ⟨*visual_commitid() action* 91a⟩≡ (102b)

```
let visual_commitid id =
  let files = Common.cmd_to_list
    (spf "git show --pretty=\"format:\" --name-only %s"
        id)
    (* not sure why git adds an extra empty line at the beginning but we
     * have to filter it
     *)
    +> Common.exclude Common.null_string
  in
  pr2_gen files;
  main_action files
```

91b ⟨*actions* 91b⟩≡ (102b)

```
    "-commitid", " <id>",
    Common.mk_action_1_arg (visual_commitid);
```

# 17 Conclusion

Hope you like it.

# A Extra Code

## A.1 model2.mli

91c ⟨*model2.mli* 91c⟩≡

⟨*type async* 86⟩
⟨*async functions sig* 87a⟩

⟨*type model* 11c⟩

⟨*type drawing* 13a⟩

⟨*init_drawing sig* 14c⟩

⟨*new_pixmap sig* 6a⟩

⟨*find_rectangle_at_user_point sig* 55a⟩

⟨*hentities sig* 13b⟩

⟨*hfiles_and_top_entities sig* 49a⟩

⟨*all_entities sig* 63b⟩

⟨*readable_to_absolute_filename_under_root sig* 16a⟩

⟨*actual_root_of_db sig* 16b⟩

## A.2 model2.ml

92 ⟨*model2.ml* 92⟩≡
⟨*Facebook copyright* 4⟩

```
open Common

module CairoH = Cairo_helpers

module F = Figures
module T = Treemap

module Db = Database_code

module Flag = Flag_visual

(*****************************************************************************)
```

```
(* Types *)
(*****************************************************************************)
```

⟨*type model* 11c⟩

⟨*type async* 86⟩

⟨*async functions* 87b⟩

```
(*****************************************************************************)
(* The drawing area *)
(*****************************************************************************)
```

⟨*type drawing* 13a⟩

⟨*new_pixmap()* 6b⟩

⟨*init_drawing()* 14d⟩

```
(*****************************************************************************)
(* POINT -> treemap info *)
(*****************************************************************************)
```

⟨*find_rectangle_at_user_point()* 55b⟩

```
(*****************************************************************************)
(* Filenames *)
(*****************************************************************************)
```

⟨*readable_to_absolute_filename_under_root* 16c⟩

⟨*actual_root_of_db* 17a⟩

```
(*****************************************************************************)
(* Entities info *)
(*****************************************************************************)
```

⟨*hentities()* 14b⟩

⟨*hfiles_and_top_entities()* 49b⟩

```
(*****************************************************************************)
(* Completion data *)
(*****************************************************************************)
```

⟨*all_entities* 64c⟩

## A.3 view2.mli

94a ⟨*view2.mli* 94a⟩≡
⟨*mk_gui sig* 17b⟩

## A.4 view2.ml

94b ⟨*view2.ml* 94b⟩≡

⟨*Facebook copyright* 4⟩

```
open Common

module G = Gui
module K = GdkKeysyms
module GR = Gdk.Rectangle

module F = Figures
module T = Treemap

module CairoH = Cairo_helpers

open Figures (* for the fields *)
open Model2 (* for the fields *)
module M = Model2

(* floats are the norm in graphics *)
open Common.ArithFloatInfix

module Style = Style2
module Draw = Draw2

module Flag = Flag_visual

module Db = Database_code

(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)

(*****************************************************************************)
(* Wrappers *)
```

```
(*****************************************************************************)
let pr2, pr2_once = Common.mk_pr2_wrappers Flag.verbose_visual

(*****************************************************************************)
(* Globals *)
(*****************************************************************************)
```

⟨*view globals* 18⟩

```
(*****************************************************************************)
(* Scaling *)
(*****************************************************************************)
```

⟨*zoom_pan_scale_map* 60e⟩

⟨*scale_minimap* 60c⟩

⟨*with_map* 60d⟩

⟨*with_minimap* 60a⟩

⟨*device_to_user_area* 32a⟩

```
(*****************************************************************************)
(* Painting *)
(*****************************************************************************)
```

⟨*paint* 29⟩

⟨*paint_minimap* 58b⟩

⟨*paint_legend* 31b⟩

```
(*****************************************************************************)
(* Overlays *)
(*****************************************************************************)

(* ---------------------------------------------------------------------- *)
(* The current filename *)
(* ---------------------------------------------------------------------- *)
```

⟨*draw_label_overlay* 77b⟩

```
(* ---------------------------------------------------------------------- *)
(* The current rectangles *)
(* ---------------------------------------------------------------------- *)
```

⟨*draw_rectangle_overlay* 78⟩
```
(* ------------------------------------------------------------------------ *)
(* The selected rectangles *)
(* ------------------------------------------------------------------------ *)
```

⟨*draw_searched_rectangles* 73a⟩

```
(* ------------------------------------------------------------------------ *)
(* The magnifying glass *)
(* ------------------------------------------------------------------------ *)
```

⟨*zoomed_surface_of_rectangle* 73b⟩

```
(****************************************************************************)
(* Layering *)
(****************************************************************************)

(* ------------------------------------------------------------------------ *)
(* The main-map *)
(* ------------------------------------------------------------------------ *)
```

⟨*assemble_layers* 76a⟩

⟨*expose* 76b⟩

⟨*configure* 77a⟩

```
(* ------------------------------------------------------------------------ *)
(* The mini-map *)
(* ------------------------------------------------------------------------ *)
```

⟨*expose_minimap* 59a⟩

⟨*configure_minimap* 59b⟩

```
(* ------------------------------------------------------------------------ *)
(* The legend *)
(* ------------------------------------------------------------------------ *)
```
⟨*expose_legend* 31a⟩

```
(****************************************************************************)
(* Events *)
(****************************************************************************)

(* ------------------------------------------------------------------------ *)
(* Navigation *)
```

```
(* ------------------------------------------------------------------------ *)
```

⟨*go_back* 56e⟩

⟨*go_dirs_or_file* 54⟩

```
(* ------------------------------------------------------------------------ *)
(* Search *)
(* ------------------------------------------------------------------------ *)
```

⟨*dialog_search_def* 61b⟩

⟨*run_grep_query* 62a⟩

⟨*run_tbgs_query* 62b⟩

```
(* ------------------------------------------------------------------------ *)
(* The main map *)
(* ------------------------------------------------------------------------ *)
```

⟨*key_pressed* 49d⟩

⟨*find_filepos_in_rectangle_at_user_point* 51⟩

⟨*button_action* 52⟩

⟨*motion_refresher* 71⟩

⟨*idle* 57a⟩

```
(* ------------------------------------------------------------------------ *)
(* The mini-map *)
(* ------------------------------------------------------------------------ *)
```

⟨*motion_notify_minimap* 57b⟩

⟨*button_action_minimap* 58a⟩

```
(****************************************************************************)
(* Heavy Computation *)
(****************************************************************************)
```
⟨*treemap_generator* 12a⟩

⟨*build_model* 12b⟩
```
(****************************************************************************)
```

```
(* The main UI *)
(*****************************************************************************)
```

⟨*mk_gui()* 19⟩

## A.5 draw2.mli

98a ⟨*draw2.mli* 98a⟩≡

⟨*type context* 37b⟩

⟨*draw_treemap_rectangle sig* 32b⟩

⟨*draw_treemap_rectangle_content_maybe sig* 38a⟩

⟨*draw_treemap_rectangle_label_maybe sig* 33b⟩

⟨*text_with_user_pos sig* 56d⟩

## A.6 draw2.ml

98b ⟨*draw2.ml* 98b⟩≡
⟨*Facebook copyright* 4⟩

```
open Common

(*
 * Floats are the norm in graphics.
 * note: with ocaml 3.12 could also use the Float.(...) local open extension
 *)
open Common.ArithFloatInfix

module Color = Simple_color

open Figures (* for the fields *)
open Model2 (* for the fields *)

module Style = Style2
module FT = File_type
module Parsing = Parsing2

module Flag = Flag_visual
```

```
module HC = Highlight_code

module Db = Database_code

module CairoH = Cairo_helpers
module T = Treemap

module F = Figures

(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)

(* ugly *)

let text_with_user_pos = ref []

(*****************************************************************************)
(* Types *)
(*****************************************************************************)
```

⟨*type draw_content_layout* 37a⟩

⟨*type context* 37b⟩

```
(*****************************************************************************)
(* Helpers *)
(*****************************************************************************)

let is_big_file_with_few_lines ~nblines fullpath =
  nblines < 20. &&
  Common.filesize_eff fullpath > 4000

(*****************************************************************************)
(* Basics *)
(*****************************************************************************)
```

⟨*draw_treemap_rectangle()* 33a⟩

```
(*****************************************************************************)
(* Color of entity *)
(*****************************************************************************)

(*****************************************************************************)
(* Anamorphic entities *)
(*****************************************************************************)
```

⟨*final_font_size_when_multiplier* 38b⟩

⟨*final_font_size_of_categ* 38c⟩

```
(*****************************************************************************)
(* Columns *)
(*****************************************************************************)
```

⟨*font_size_when_have_x_columns* 39a⟩

⟨*optimal_nb_columns* 39b⟩

⟨*draw_column_bars* 39c⟩

```
(*****************************************************************************)
(* File Summary *)
(*****************************************************************************)
```

⟨*draw_summary_content* 46⟩

```
(*****************************************************************************)
(* File Content *)
(*****************************************************************************)
```

⟨*draw_content* 40⟩

⟨*draw_treemap_rectangle_content_maybe* 44⟩

```
(*****************************************************************************)
(* Label *)
(*****************************************************************************)
```

⟨*draw_treemap_rectangle_label_maybe* 33c⟩

## A.7 parsing2.mli

## A.8 parsing2.ml

## A.9 completion2.mli

## A.10 completion2.ml

## A.11 style2.mli

## A.12 style2.ml

101a ⟨*style2.ml* 101a⟩≡
⟨*Facebook copyright* 4⟩

```
open Common

module SH = Highlight_code

module Flag = Flag_visual

(*****************************************************************************)
(* Visual style *)
(*****************************************************************************)
(* see also model2.settings *)
```

⟨*zoom_factor_incruste_mode* 90a⟩

⟨*threshold_draw_dark_background_font_size_real* 90b⟩

⟨*size_font_multiplier_of_categ()* 89⟩

⟨*windows_params()* 88c⟩

## A.13 cairo_helpers.mli

101b ⟨*cairo_helpers.mli* 101b⟩≡

⟨*cairo helpers functions sig* 6c⟩

## A.14 cairo_helpers.ml

101c ⟨*cairo_helpers.ml* 101c⟩≡
⟨*Facebook copyright* 4⟩

```
open Common
```

```
module F = Figures
module Color = Simple_color

open Figures

(* May have to move this in commons/ at some point *)
```

⟨*cairo helpers functions* 7⟩

## A.15 editor_connection.mli

## A.16 editor_connection.ml

102a ⟨*editor_connection.ml* 102a⟩≡

⟨*Facebook copyright* 4⟩

```
open Common

(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)

(*****************************************************************************)
(* Emacs *)
(*****************************************************************************)
```

⟨*emacs configuration* 56b⟩

```
(*****************************************************************************)
(* Vi *)
(*****************************************************************************)

(*****************************************************************************)
(* Wrappers *)
(*****************************************************************************)
```

⟨*open_file_in_current_editor()* 56c⟩

## A.17 main_visual.ml

102b ⟨*main_visual.ml* 102b⟩≡

```
(*
 * Please imagine a long and boring gnu-style copyright notice
 * appearing just here.
```

```
 *)
open Common

module Flag = Flag_visual

(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)

(* *)

(*****************************************************************************)
(* Flags *)
(*****************************************************************************)

⟨main flags 11a⟩

(* action mode *)
let action = ref ""

(*****************************************************************************)
(* Main action *)
(*****************************************************************************)

⟨main_action() 11b⟩

(*****************************************************************************)
(* Extra actions *)
(*****************************************************************************)

⟨visual_commitid() action 91a⟩

(*---------------------------------------------------------------------------*)
(* the command line flags *)
(*---------------------------------------------------------------------------*)
let extra_actions () = [
 ⟨actions 91b⟩
]

(*****************************************************************************)
(* The options *)
(*****************************************************************************)

(* update: try put ocamlgtk related test in widgets/test_widgets.ml, not
 * here. Here it's for ... well it's for nothing I think as it's not really
 * easy to test gui.
```

```
 *)
let all_actions () =
 extra_actions()++
 []

let options () = [
  ⟨options 87c⟩
  ] ++
  Common.options_of_actions action (all_actions()) ++
  Flag_analyze_php.cmdline_flags_verbose () ++
  Common.cmdline_flags_devel () ++
  Common.cmdline_flags_verbose () ++
  [
  "-version",   Arg.Unit (fun () ->
    pr2 (spf "pfff_visual version: %s" Config.version);
    exit 0;
  ),
    "  guess what";
  ]

(*****************************************************************************)
(* The main entry point *)
(*****************************************************************************)
let main () =
  Common_extra.set_link ();

  let usage_msg =
    ("Usage: " ^ basename Sys.argv.(0) ^ " [options] <path> \nOptions are:")
  in
  let args = Common.parse_options (options()) usage_msg Sys.argv in

  (* must be done after Arg.parse, because Common.profile is set by it *)
  Common.profile_code "Main total" (fun () ->

    (match args with
    (* ------------------------------------------------------- *)
    (* actions, useful to debug subpart *)
    (* ------------------------------------------------------- *)
    | xs when List.mem !action (Common.action_list (all_actions())) ->
        Common.do_action !action xs (all_actions())

    | _ when not (Common.null_string !action) ->
        failwith ("unrecognized action or wrong params: " ^ !action)

    (* ------------------------------------------------------- *)
    (* main entry *)
```

```
    (* ------------------------------------------------------- *)

    | (x::xs) ->
        main_action (x::xs)

    (* ------------------------------------------------------- *)
    (* empty entry *)
    (* ------------------------------------------------------- *)
    | _ -> Arg.usage (Arg.align (options())) usage_msg;
    );
  )

(*****************************************************************************)
let _ =
  if Sys.argv +> Array.to_list +> List.exists (fun x -> x ="-debugger")
  then Common.debugger := true;

  Common.finalize
    (fun ()->
      main ()
    )
    (fun()->
      pr2 (Common.profile_diagnostic ());
      Common.erase_temp_files ();
    )
```

### A.18 flag_visual.ml

# B Changelog

# Indexes

# References


[1] Donald Knuth,, *Literate Programming*, http://en.wikipedia.org/wiki/Literate_Program cited page(s) 5

[2] Norman Ramsey, *Noweb*, http://www.cs.tufts.edu/~nr/noweb/ cited page(s) 5

[3] Yoann Padioleau, *Syncweb, literate programming meets unison*, http://padator.org/software/project-syncweb/readme.txt cited page(s) 5

[4] Yoann Padioleau, *Commons Pad OCaml Library*, http://padator.org/docs/Commons.pdf cited page(s)

[5] Wikipedia, *Treemapping*, http://en.wikipedia.org/wiki/Treemapping cited page(s)