CodeMap

Google Maps for Source Code

version 0.3

Yoann Padioleau
pad@fb.com

April 29, 2014

Copyright © 2010-2012 Facebook
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3.

Contents

1 Introduction

1.1 Motivations

```
(*
 * 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.
 *
 * 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.
 *
 * maxim of information visualization:
 *  - show the data
 *  - show comparisons
 *)


can copy also what is inside Treemap.tex.nw
```

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

whole program visualization. To actually see the architecture.

%It also abstract away like PofFS how you organize your information.
If have one big test file or many small test files for instance,
then it does not matter, you can see all of the information at once
as it was a giant tiling plane.
So poffs is a kind of a precursor to codemap :)
```

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.

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:

2 ⟨*Facebook copyright* 2⟩≡

```
(* Yoann Padioleau
 *
 * Copyright (C) 2010-2012 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

```
put cool scenario of use. cool question and how can get visual answer to
those questions!
```

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.

3 ⟨*interface_doc* 3⟩≡

```
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).

  "
```

2.2 Viewing Pfff itself

2.3 Generic semantic visual feedback

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

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

%sgrep connection ?

% see semantic info like  takeArgByRef or use of globals or ContainDynCall
%  or TODO dead func,
```

3 Implementation Overview

```
% say that model/view/controller

 * 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

 * There are also 'draw' which is called by the 'paint'. 'Paint' creates
 * the cairo context and adjust the scaling if needed and then call
 * 'draw'

 * then there are cairo layers.

 *
 * Assumes the treemap contains the absolute paths to existing files/dirs.


concepts:
user vs device
see Cairo/Gtk appendix

macro vs micro level
```

```
Plan for following sections.
Dependencies diagram and Tabular.
 - main
 - treemap
 - semantic info and parser
 - assembling layers
 - gtk/cairo
```

4 Main

5 ⟨*main flags* 5⟩≡

```
(* on Macos lion, X11 resizes the window to a smaller size so
 * no point in starting with a big screen_size :(
 *)
let screen_size = ref 1
let legend = ref true

(* you can also put this in your ~/gtkrc-2.0
 * gtk-icon-theme-name = "Tango"
 * gtk-theme-name = "Murrine-Gray"
 * gtk-font-name = "DejaVu Sans 16"
 *)

(* if not specified, codemap will try to use files in the current directory *)
let db_file    = ref (None: Common.filename option)
let graph_file = ref (None: Common.filename option)
let layer_file = ref (None: Common.filename option)
let layer_dir  = ref (None: Common.dirname option)

(* See also Gui.synchronous_actions *)
let test_mode = ref (None: string option)
```

6 ⟨*main_action()* 6⟩≡

```
let main_action xs =
  set_gc ();
  Logger.log Config_pfff.logger "codemap" None;

  (* this used to be done by linking with gtkInit.cmo, but better like this *)
  let _locale = GtkMain.Main.init () in
  pr2 (spf "Using Cairo version: %s" Cairo.compile_time_version_string);

  let root = Common2.common_prefix_of_files_or_dirs xs in
  pr2 (spf "Using root = %s" root);

  let async_model = Async.async_make () in
```

```
let layers =
  match !layer_file, !layer_dir, xs with
  | Some file, _, _ ->
      [Layer_code.load_layer file]
  | None, Some dir, _ | None, None, [dir] ->
      layers_in_dir dir +> List.map Layer_code.load_layer
  | _ -> []
in
let layers_with_index =
  Layer_code.build_index_of_layers ~root
    (match !layer_file, layers with
    | Some _, [layer] -> [layer, true]
    | _ -> layers +> List.map (fun x -> x, false)
    )
in

let db_file =
  match !db_file, xs with
  | Some file, _ -> Some file
  | None, [dir] ->
    let candidates = [
        Filename.concat dir Database_code.default_db_name;
        Filename.concat dir Database_code.default_db_name ^ ".json";
    ] in
    (try
      Some (candidates +> List.find (fun file -> Sys.file_exists file))
    with Not_found -> None
    )
    | _ -> None
in
db_file +> Common.do_option (fun db ->
  pr2 (spf "Using pfff light db: %s" db)
);
let graph_file =
  match !graph_file, xs with
  | Some file, _ -> Some file
  | None, [dir] ->
    let candidates = [
        Filename.concat dir Graph_code.default_filename;
    ] in
    (try
      Some (candidates +> List.find (fun file -> Sys.file_exists file))
    with Not_found -> None
    )
  | _ -> None
```

```
in
graph_file +> Common.do_option (fun db ->
  pr2 (spf "Using graphcode: %s" db)
);
let skip_file = !skip_file ||| Filename.concat root "skip_list.txt" in
let skip_list =
  if Sys.file_exists skip_file
  then begin
    pr2 (spf "Using skip file: %s" skip_file);
    Skip_code.load skip_file
  end
  else []
in
let filter_files_skip_list = Skip_code.filter_files skip_list root in
let filter_file = (fun file ->
  !filter file &&
  (skip_list = [] || filter_files_skip_list [file] <> []))
in

let treemap_func = treemap_generator ~filter_file in
let dw = Model.init_drawing  treemap_func layers_with_index xs root in

(* This can require lots of stack. Make sure to have ulimit -s 40000 *)
Thread.create (fun () ->
  (* heavy computation are not *fairly* scheduled apparently by the OCaml
   * runtime, so let's do the heavy computation in another process
   * and here just have the thread waiting for it to be done.
   * This thread used to cause some Bus error on MacOS but now that
   * we use invoke and do the job in another process things seems better :)
   *)
  let job () = build_model root db_file graph_file in
  let res = Parallel.invoke job () () in
  Async.async_set res async_model;
) ()
+> ignore;

let w = { Model.
  dw;
  dw_stack = ref [dw];
  model = async_model;
  treemap_func;
  current_node = None;
  current_node_selected = None;
  current_entity = None;
  settings = { Model.
    (* todo: too fuzzy for now *)
```

```
      draw_summary = false;
      draw_searched_rectangles = true;
    };
    root_orig = root;
  }
  in

  View2.mk_gui  ~screen_size:!screen_size ~legend:!legend !test_mode w
```

5 The Model

5.1 The code database

8 ⟨*type model* 8⟩≡

```
(* filename below should be in readable path format *)
type model = {
  (* for translating the absolute filenames in tr_label in readable so
   * one can access the node in the model for a tr_rectangle
   *)
  root: Common.dirname;

  db: Database_code.database option;
  ⟨model fields hook 12c⟩

  (* for microlevel use/def information *)
  g: Graph_code.graph option;
  (* for macrolevel use/def information, only for Dir and File *)
  hfile_deps_of_node: (Graph_code.node, Common.filename deps) Hashtbl.t;
  (* we used to store line information there, but the file may have changed *)
  hentities_of_file: (Common.filename, Graph_code.node list) Hashtbl.t;
 }
```

9a ⟨*build_model* 9a⟩≡

```
(* this is currently called in the background *)
let build_model2 root dbfile_opt graphfile_opt =

  let db_opt = dbfile_opt +> Common.map_opt Database_code.load_database in
  (* todo: and skip_list?
   * less: opti by factorizing the 'find' with treemap_generator?
  *)
  let files =
    Common.files_of_dir_or_files_no_vcs_nofilter [root] +> List.filter !filter
  in
  let hentities = Model_database_code.hentities root db_opt in
  let all_entities = Model_database_code.all_entities ~root files db_opt in
```

```
  let big_grep_idx = Completion2.build_completion_defs_index all_entities in

  let g_opt = graphfile_opt +> Common.map_opt Graph_code.load in
  let hfile_deps_of_node, hentities_of_file =
    match g_opt with
    | None -> Hashtbl.create 0, Hashtbl.create 0
    | Some g ->
      let a = Model_graph_code.build_filedeps_of_dir_or_file g in
      let b = Model_graph_code.build_entities_of_file g in
      let b = Model_graph_code.add_headers_files_entities_of_file root b in
      a, Common.hash_of_list b
  in

  let model = { Model.
        root = root;
        db = db_opt;
        hentities; big_grep_idx;
        g =  g_opt;
        hfile_deps_of_node; hentities_of_file;
  }
  in
  model

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

5.2 The treemap generator

9b ⟨*treemap_generator* 9b⟩≡

```
(* this is called each time we go in a new directory (or set of dirs) *)
let treemap_generator ~filter_file =
 fun paths ->
  let treemap = Treemap_pl.code_treemap ~filter_file paths in
  let algo = Treemap.Ordered Treemap.PivotByMiddle in
  let big_borders = !Flag.boost_label_size in
  let rects = Treemap.render_treemap ~algo ~big_borders treemap in
  Common.pr2 (spf "%d rectangles to draw" (List.length rects));
  rects
```

5.3 The final model

10 ⟨*type drawing* 10⟩≡

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

```
 *)
type drawing = {

  (* Macrolevel. In user coordinates from 0 to T.xy_ratio for 'x' and 0 to 1
   * for 'y'. Assumes the treemap contains absolute paths (tr.tr_label).
   *)
  treemap: Treemap.treemap_rendering;
  (* Microlevel. When we render content at the microlevel, we then need to
   * know to which line corresponds a position and vice versa.
   *)
  microlevel: (Treemap.treemap_rectangle, microlevel) Hashtbl.t;

  (* generated from dw.treemap, contains readable path relative to model.root *)
  readable_file_to_rect:
    (Common.filename, Treemap.treemap_rectangle) Hashtbl.t;
  (* coupling: = List.length treemap *)
  nb_rects: int;
  (* Used to display readable paths. When fully zoomed it's a filename. *)
  current_root: Common.path;

  mutable layers: Layer_code.layers_with_index;

  ⟨fields drawing query stuff 51b⟩

  ⟨fields drawing main view 60⟩

  ⟨fields drawing viewport 49i⟩

  ⟨fields drawing minimap 49e⟩
}
  ⟨type settings 78a⟩
```

5.4 The microlevel specific model

11a ⟨type context 11a⟩≡

```
(* a slice of drawing used in the drawing functions *)
type context = {
  model2: model Async.t;
  nb_rects_on_screen: int;
  grep_query: (Common.filename, line) Hashtbl.t;
  layers_microlevel:
   (Common.filename, (int, Simple_color.emacs_color) Hashtbl.t) Hashtbl.t;
}
```

5.5 On filenames

```
quite tricky. Also had pbs with our testing code. Relative path
are convenient but bad in code.

- relative
- absolute
- readable

need readable in files so can reuse (for code light database and layers,
see section X and Y later)
```

11b ⟨*readable_to_absolute_filename_under_root sig* 11b⟩≡

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

11c ⟨*actual_root_of_db sig* 11c⟩≡

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

11d ⟨*readable_to_absolute_filename_under_root* 11d⟩≡

```
(* 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 readable_to_absolute_filename_under_root ~root filename =

  (* the root may be a filename *)
  let root_dir =
    if Common2.is_directory root then root
    else Filename.dirname root
  in
```

```
  let root_and_parents =
    Common2.inits_of_absolute_dir root_dir +> List.rev
  in
  try
    root_and_parents +> Common2.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)
```

12a ⟨*actual_root_of_db* 12a⟩≡

```
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)
```

5.6 Misc

12b ⟨*hentities sig* 12b⟩≡

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

12c ⟨*model fields hook* 12c⟩≡

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

13a ⟨*hentities()* 13a⟩≡

```
(* 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
```

13b ⟨*init_drawing sig* 13b⟩≡

```
val init_drawing :
  ?width:int ->
  ?height:int ->
  (Common.path list -> Treemap.treemap_rendering) ->
  Layer_code.layers_with_index ->
  Common.path list ->
  Common.dirname (* root *) ->
  drawing
```

13c ⟨*init_drawing()* 13c⟩≡

```
(* This is a first guess. The first configure ev will force a resize. *)
let init_drawing   ?(width = 600) ?(height = 600) func layers paths root =
  let paths = List.map Common2.relative_to_absolute paths in
  let current_root = Common2.common_prefix_of_files_or_dirs paths in
  let treemap =
    Common.profile_code "Visual.building the treemap" (fun () -> func paths) in
  let readable_file_to_rect =
    treemap +> Common.map_filter (fun rect ->
      if not rect.T.tr_is_node
      then
        let file  = rect.T.tr_label in
        let readable = Common.readable ~root file in
        Some (readable, rect)
      else None
    ) +> Common.hash_of_list
```

```
in
{
  treemap;
  nb_rects = List.length treemap;
  current_root;
  readable_file_to_rect;
  microlevel = Hashtbl.create 0;
  layers;
  current_query = "";
  current_searched_rectangles = [];
  current_grep_query = Hashtbl.create 0;
  width; height;
  base    = new_surface ~alpha:false ~width ~height;
  overlay = new_surface ~alpha:true ~width ~height;
}
```

6 The UI

6.1 Overall organisation

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

14 ⟨*mk_gui sig* 14⟩≡

```
val mk_gui :
  screen_size:int -> legend:bool -> 'b option (* test *) -> Model2.world ->
  unit
```

15a ⟨*view globals* 15a⟩≡

```
(* 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.
 *)
```

6.2 Menu

6.3 Toolbar

6.4 Main view

6.5 Status bar

6.6 Misc

15b ⟨*mk_gui()* 15b⟩≡

```
let mk_gui ~screen_size ~legend test_mode w =
  let width, height, minimap_hpos, minimap_vpos =
    Style.windows_params screen_size in

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

  let statusbar = GMisc.statusbar () in
  let ctx = statusbar#new_context "main" in
  Controller._statusbar_addtext := (fun s -> ctx#push s +> ignore);

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

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

  win#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:win#add () in
  let hbox = GPack.hbox ~packing:(vbox#pack ~expand:false ~fill:false) () in
```

```
    (*--------------------------------------------------------------------*)
    (* Menu *)
    (*--------------------------------------------------------------------*)
    hbox#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;

      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 () ->
          !Controller._go_back w;
        ) +> ignore;

        fc#add_item "_Go to example" ~key:K._E ~callback:(fun () ->
          let model = Async.async_get w.model in
          match w.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 =
                    Model_database_code.readable_to_absolute_filename_under_root
                      file ~root:w.dw.current_root in
                  w.current_entity <- Some e;
                  !Controller._go_dirs_or_file w [final_file];
              )
```

```
    | _ -> failwith "no entity currently selected or no db"
  ) +> ignore;
  fc#add_item "_Quit" ~key:K._Q ~callback:quit +> 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 = Ui_search.dialog_search_def w.model in
    res +> Common.do_option (fun s ->
      let root = w.root_orig in
      let matching_files = Ui_search.run_grep_query ~root s in
      let files = matching_files +> List.map fst +> Common2.uniq in
      let current_grep_query =
        Some (Common.hash_of_list matching_files)
      in
      !Controller._go_dirs_or_file ~current_grep_query w files
    );
  ) +> ignore;

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

    let res = Ui_search.dialog_search_def w.model in
    res +> Common.do_option (fun s ->
      let root = w.dw.current_root in
      let matching_files = Ui_search.run_tbgs_query ~root s in
      let files = matching_files +> List.map fst +> Common2.uniq in
      let current_grep_query =
        Some (Common.hash_of_list matching_files)
      in
      !Controller._go_dirs_or_file ~current_grep_query w files
    );
  ) +> ignore;

);
factory#add_submenu "_Layers" +> (fun menu ->
  let layers =
    w.dw.layers.Layer_code.layers +> List.map (fun (layer, active) ->
      (layer.Layer_code.title, active, (fun b ->
        if b then
          Ui_layers.choose_layer ~root:w.root_orig
            (Some layer.Layer_code.title) w;
      ))
```

```
    )
  in
  (* todo: again, make the architecture a layer so less special cases *)
  let entries = [`R (
       ("Architecture", true, (fun _b ->
         Ui_layers.choose_layer ~root:w.root_orig None w;
       ))::
       layers)
  ]
  in
  GToolbox.build_menu menu ~entries
);

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 = w.dw.current_root in
    let _old_dw = Common2.pop2 w.dw_stack in
    !Controller._go_dirs_or_file w [current_root];
  ) +> ignore;

  fc#add_item "Reset entity" ~callback:(fun () ->
    w.current_node_selected <- None;
    let cr_overlay = Cairo.create w.dw.overlay in
    CairoH.clear cr_overlay;
    !Controller._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 Help.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 *)
    (*----------------------------------------------------------------------*)
    hbox#pack ~padding:10 (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 = Async.async_get w.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
      (Model_database_code.readable_to_absolute_filename_under_root
         ~root:w.dw.current_root)
  in

  pr2 (spf "e= %s, final_paths= %s" str(Common.join "|" final_paths));
  w.current_entity <- Some e;
  Async.async_get_opt w.model +> Common.do_option (fun model ->
    model.g +> Common.do_option (fun g ->
      w.current_node_selected <-
        Model_graph_code.node_of_entity e g
    )
  );
  !Controller._go_dirs_or_file w final_paths;
  true
)
~callback_changed:(fun str ->
  w.dw.current_query <- str;
  w.dw.current_searched_rectangles <- [];

  if w.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 = w.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
        w.dw.current_searched_rectangles <- res;

      end;
      let cr_overlay = Cairo.create w.dw.overlay in
      CairoH.clear cr_overlay;
      View_overlays.draw_searched_rectangles ~dw:w.dw;
      !Controller._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 () ->
    !Controller._go_back w;
  )
));

tb#insert_widget (G.mk (GButton.button ~stock:`GO_UP) (fun b ->
  b#connect#clicked ~callback:(fun () ->
    let current_root = w.dw.current_root in
    !Controller._go_dirs_or_file w [Common2.dirname current_root];
  )
));

tb#insert_widget (G.mk (GButton.button ~stock:`GOTO_TOP) (fun b ->
  b#connect#clicked ~callback:(fun () ->
    let top = Common2.list_last !(w.dw_stack) in
    (* put 2 in the stack because _go_back will popup one *)
    w.dw_stack := [top; top];
    !Controller._go_back w;
  )
));
));
```

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

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

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


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

let da3 = GMisc.drawing_area () in
vpane#set_position minimap_vpos;
vpane#add2 da3#coerce;


if legend then hpane#add2 vpane#coerce;

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

(* subtle: do not change those callbacks to get a dw; you need to
 * pass a w! Indee if you do ~callback:(expose da w.dw)
 * and an event changes w.dw (e.g. a resize of the window)
 * then the expose event will still expose the old drawing.
 *)
da#event#connect#expose ~callback:(expose da w) +> ignore;
da#event#connect#configure ~callback:(configure w) +> ignore;
da3#event#connect#expose ~callback:(expose_legend da3 w) +> ignore;

da#event#connect#button_press
  (View_mainmap.button_action w) +> ignore;
da#event#connect#button_release
  (View_mainmap.button_action w) +> ignore;
da#event#connect#motion_notify
  (View_overlays.motion_notify w) +> ignore;

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

```
    Controller._refresh_legend := (fun () ->
      GtkBase.Widget.queue_draw da3#as_widget;
    );
    Controller._go_back := Ui_navigation.go_back;
    Controller._go_dirs_or_file := Ui_navigation.go_dirs_or_file;
    Controller.hook_finish_paint := (fun () ->
      View_overlays.hook_finish_paint w
    );

    (*----------------------------------------------------------------------*)
    (* 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 ^ "\n" ^ s 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
    )
```

```
  );
*)

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

  (* test *)
  test_mode +> Common.do_option (fun _s ->
    (* View_test.do_command s model *)
    ()
  );
  (* Gui.gmain_idle_add ~prio: 1000 (idle dw) +> ignore; *)
  GtkThread.main ();
  ()
```

7 The Controller

8 Macrolevel View: The Treemap

8.1 Principles

8.2 Drawing

24 ⟨*device_to_user_area* 24⟩≡

```
(* 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;
    }
  )
```

25a ⟨*draw_treemap_rectangle sig* 25a⟩≡

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

```
    Treemap.treemap_rectangle ->
    unit
```

25b ⟨*draw_treemap_rectangle()* 25b⟩≡

```
  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)
```

8.3 Labels

25c ⟨*draw_treemap_rectangle_label_maybe sig* 25c⟩≡

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

26 ⟨*draw_treemap_rectangle_label_maybe* 26⟩≡

```
  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.
 *
 * note that this function is also called when we mouse over a rectangle
 * in which case we redraw the label in a different color
 *
 * design: good to have a color different for dir and files.
 *
 * design: could decide to give different colors to dirs depending on its
 * depth, like red for toplevel dir, green, and so on, like I do for
 * my code sections, but feel simpler to just have one.
 * The rectangles will already have different colors and in the end
 * the depth does not have that much meaning in projects. For instance
 * in java code there are lots of nested directories (org/apache/...),
 * in some projects there is always an intermediate src/ directory;
 * each software have different conventions.
 *)
let rec draw_treemap_rectangle_label_maybe2 ~cr ~zoom ?(color=None) 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
  let color =
    match color with
    | None ->
        if is_directory
        then "NavyBlue"
        else "black"
    | Some c -> c
  in

  Cairo.select_font_face cr "serif"
    Cairo.FONT_SLANT_NORMAL Cairo.FONT_WEIGHT_BOLD;

  let font_size, minus_alpha =
    if not !Flag.boost_label_size
    then
     (match rect.T.tr_depth with
```

```
    | 1 -> 0.1, 0.8
    | 2 -> 0.05, 0.2
    | 3 -> 0.03, 0.4
    | 4 -> 0.02, 0.5
    | 5 -> 0.02, 0.65
    | 6 -> 0.02, 0.7
    | _ -> 0.02, 0.8
     )
    else
    (match rect.T.tr_depth with
    | 1 -> 0.1, 0.8
    | 2 -> 0.06, 0.2
    | 3 -> 0.045, 0.3
    | 4 -> 0.041, 0.35
    | 5 -> 0.04, 0.4
    | 6 -> 0.03, 0.45
    | _ -> 0.02, 0.5
    )
  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 =

(* ugly: sometimes labels are too big. Should provide a way to
 * shorten them.
 * let txt =
 * if true
 * then if txt =~ "org.eclipse.\\(.*\\)" then Common.matched1 txt else txt
 * else txt
 * in
 *)
(*
  let txt =
  if true
  then if txt =~ "[^-]*-\\(.*\\)" then Common.matched1 txt else txt
  else txt
  in
*)
```

```
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' actually means "the toplevel entries" as
       * the root is at depth=1
       *)
      if rect.T.tr_depth <= 2 then begin
        (let (r,g,b) = Color.rgbf_of_string "red" in
         Cairo.set_source_rgba cr r g b alpha;
        );
        Cairo.move_to cr x y;
        CairoH.show_text cr (String.sub txt 0 1);
      end


| 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);

    if rect.T.tr_depth <= 2 then begin
      (let (r,g,b) = Color.rgbf_of_string "red" in
       Cairo.set_source_rgba cr r g b alpha;
      );
      Cairo.move_to cr x y;
      Cairo.rotate cr ~angle:angle;
      CairoH.show_text cr (String.sub txt 0 1);
      Cairo.rotate cr ~angle:(-. angle);
    end;




| 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)
```

8.4 Color code

This is kind of interfile "aspect"

31a ⟨*paint_legend* 31a⟩≡

```
  (* todo: make the architecture a layer so no need for special case *)
  let draw_legend ~cr =

    let archis = Archi_code.source_archi_list in
    let grouped_archis = archis +> Common.group_by_mapped_key (fun archi ->
      (* I tend to favor the darker variant of the color in treemap_pl.ml hence
       * the 3 below
       *)
      Treemap_pl.color_of_source_archi archi ^ "3"
    )
    in
    let grouped_archis = grouped_archis +> List.map (fun (color, kinds) ->
      color, kinds +> List.map Archi_code.s_of_source_archi +> Common.join ", "
    ) in
    draw_legend_of_color_string_pairs ~cr grouped_archis
```

31b ⟨*expose_legend* 31b⟩≡

```
  let expose_legend da w _ev =
    let dw = w.dw in
    let cr = Cairo_lablgtk.create da#misc#window in

    (* todo: make the architecture a layer so no need for special case *)
    (if not (Layer_code.has_active_layers dw.layers)
    then Draw_legend.draw_legend ~cr
    else Draw_legend.draw_legend_layer ~cr dw.layers
    );
    true
```

9 Microlevel View: Source Thumbnails

9.1 Principles

9.2 Drawing

32a ⟨*type draw_content_layout* 32a⟩≡

32b ⟨*draw_treemap_rectangle_content_maybe sig* 32b⟩≡

```
(* will render (maybe) the file content of treemap_rectangle.tr_label *)
val draw_treemap_rectangle_content_maybe:
  Cairo.t ->
  Figures.rectangle ->
  Model2.context ->
  Treemap.treemap_rectangle ->
  Model2.microlevel option
```

9.3 Entities size

32c ⟨*final_font_size_when_multiplier* 32c⟩≡

```
let final_font_size_when_multiplier
    ~multiplier ~size_font_multiplier_multiplier
    ~font_size ~font_size_real
   =
  ignore(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 =
    Common2.borne ~min:font_size ~max:(font_size * 30.) font_size_adjusted
  in
  final_font_size
```

33a ⟨*final_font_size_of_categ* 33a⟩≡

```
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 multiplier =
   (*- 0.2 * font_size_real + 2. *)
  match font_size_real with
  | n when n < 3. -> 2.0 * multiplier
  | n when n < 8. -> 0.8 * multiplier
  | n when n < 10. -> 0.7 * multiplier
  | _ -> 0.5 * multiplier
in
Common2.borne ~min:font_size ~max:(font_size * 30.) (font_size * multiplier)
```

9.4 Column layout

33b ⟨*font_size_when_have_x_columns* 33b⟩≡

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

33c ⟨*optimal_nb_columns* 33c⟩≡

```
(* 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 ~chars_per_column ~w ~h =

  let rec aux current_font_size current_nb_columns =
    let min_font = font_size_when_have_x_columns
      ~nblines ~chars_per_column ~w ~h ~with_n_columns:current_nb_columns
    in
    if min_font > current_font_size
    then aux min_font (current_nb_columns + 1.)
    (* regression, then go back on step *)
    else current_nb_columns - 1.
  in
  aux 0.0 1.0
```

34a ⟨*draw_column_bars* 34a⟩≡

```
let draw_column_bars2 cr layout r =
  for i = 1 to int_of_float (layout.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 layout.lfont_size in
    let width =
      if font_size_real > 5.
      then layout.lfont_size / 10.
      else layout.lfont_size
    in
    Cairo.set_line_width cr width;

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

9.5 Content rendering

34b ⟨*draw_content* 34b⟩≡

```
let draw_content2 cr layout context tr =

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

  let font_size = layout.lfont_size in
  let font_size_real = CairoH.user_to_device_font_size cr font_size in

  if font_size_real > Style.threshold_draw_dark_background_font_size_real
  then begin

    (* erase what was done at the macrolevel *)
    if Hashtbl.length context.layers_microlevel > 0 then begin
      Draw_macrolevel.draw_treemap_rectangle ~cr ~color:(Some "white")
        ~alpha:1.0 tr;
    end;

    let alpha =
      match context.nb_rects_on_screen with
      | n when n <= 1 -> 0.95
      | n when n <= 2 -> 0.8
      | n when n <= 10 -> 0.6
      | _ -> 0.3
    in
    (* unset when used when debugging the layering display *)
```

```
  (* if Hashtbl.length context.layers_microlevel = 0 *)

  Draw_macrolevel.draw_treemap_rectangle ~cr ~color:(Some "DarkSlateGray")
    ~alpha tr;
  (* draw a thin rectangle with aspect color *)
  CairoH.draw_rectangle_bis ~cr ~color:(tr.T.tr_color)
    ~line_width:(font_size / 2.) tr.T.tr_rect;
end;

(* highlighting layers (and grep-like queries at one point) *)
let hmatching_lines =
  try Hashtbl.find context.layers_microlevel file
  with Not_found -> Hashtbl.create 0
in
(* todo: make sgrep_query a form of layer *)
let matching_grep_lines =
  try Hashtbl.find_all context.grep_query file
  with Not_found -> []
in
matching_grep_lines +> List.iter (fun line ->
  let (Line iline) = line in
  Hashtbl.add hmatching_lines (iline+..1) "purple"
);

(* the important function call, getting the decorated content *)
let glyphs_opt =
  glyphs_of_file ~font_size ~font_size_real context.model2 file in

glyphs_opt +> Common.do_option (fun glyphs ->
  glyphs +> Array.iteri (fun line_0_indexed _glyph ->
    let lc = line_to_line_in_column (Line line_0_indexed) layout in
    let x, y = line_in_column_to_bottom_pos lc r layout in
    Cairo.move_to cr x y;

    glyphs.(line_0_indexed) +> List.iter (fun glyph ->
      let pos = Cairo.get_current_point cr in
      glyph.pos <- pos;
      Cairo.set_font_size cr glyph.M.font_size;
      let (r,g,b) = Color.rgbf_of_string glyph.color in
      Cairo.set_source_rgba cr r g b 1.;
      CairoH.show_text cr glyph.M.str;
    );

    (* hmatching_lines comes from layer_microlevel which is 1-index based *)
    let line = line_0_indexed +.. 1 in
    (match Common2.hfind_option line hmatching_lines with
```

```
      | None -> ()
      | Some color ->
        CairoH.fill_rectangle ~cr
          ~alpha:0.25
          ~color
          ~x
          (* 'y' is from line_in_column_to_bottom_pos() so it's a bottom pos
           * todo? do we want to show 3 lines centered, hence the * 2 below?
           *)
          ~y:(y - (layout.height_per_line (* * 2. *)) )
          ~w:layout.width_per_column
          ~h:(layout.height_per_line (* * 3. *))
          ()
      );
    );
  );

  { line_to_rectangle =
      (fun line -> line_to_rectangle line r layout);
    point_to_line =
      (fun pt -> point_to_line pt r layout);
    layout;
    container = tr;
    content = glyphs_opt;
    defs = (match glyphs_opt with None -> [] | Some x -> defs_of_glyphs x);
  }

let draw_content cr layout context tr =
  Common.profile_code "View.draw_content" (fun () ->
    draw_content2 cr layout context tr)
```

36 ⟨*draw_treemap_rectangle_content_maybe* 36⟩≡

```
let draw_treemap_rectangle_content_maybe2 cr clipping context tr  =
  let r = tr.T.tr_rect in

  if F.intersection_rectangles r clipping = None
  then (* pr2 ("not drawing: " ^ file) *) None
  else begin
    let file = tr.T.tr_label 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 Common2.lfile_exists_eff file && File_type.is_textual_file file
    then begin
```

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

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. Forking sucks.
  * alt: we could store the nblines of a file in the db.
  *)
  let nblines = Common2.nblines_eff file +> 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 readjusted.
  *)
  let chars_per_column = 41.0 in

  let split_nb_columns =
    optimal_nb_columns ~nblines ~chars_per_column ~h ~w in
  let font_size =
    font_size_when_have_x_columns ~nblines ~chars_per_column ~h ~w
      ~with_n_columns:split_nb_columns in

  let layout = {
    nblines;
    lfont_size = font_size;
    split_nb_columns;
    width_per_column = w / split_nb_columns;
    height_per_line = font_size;
    nblines_per_column = (nblines / split_nb_columns) +> ceil;
  }
  in
  draw_column_bars cr layout r;

  Cairo.select_font_face cr Style.font_text
    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);*)

  if font_size_real > !Flag.threshold_draw_content_font_size_real
      && not (is_big_file_with_few_lines ~nblines file)
      && nblines < !Flag.threshold_draw_content_nblines
  then Some (draw_content cr layout context tr)
  else None
```

```
      end
      else None
    end
    else None
  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)
```

9.6 Color code

10 Macro+Micro View

10.1 Painting

38 ⟨*paint* 38⟩≡

```
let paint_content_maybe_rect ~user_rect dw model rect =
 with_map dw (fun cr ->
   let context = M.context_of_drawing dw model in
   let clipping = user_rect in
   let microlevel_opt =
     Draw_microlevel.draw_treemap_rectangle_content_maybe
       cr clipping context rect in
   microlevel_opt +> Common.do_option (fun microlevel ->
     Hashtbl.replace dw.microlevel rect microlevel
   );
  (* have to redraw the label *)
   Draw_labels.draw_treemap_rectangle_label_maybe
     ~cr ~zoom:1.0 ~color:None rect;
 )

(* todo: deadlock:  M.locked (fun () ->  ) dw.M.model.M.m *)
let lazy_paint user_rect dw model () =
  pr2 "Lazy Paint";
  let start = Unix.gettimeofday () in
  while Unix.gettimeofday () - start < 0.3 do
    match !Ctl.current_rects_to_draw with
    | [] -> ()
    | x::xs ->
        Ctl.current_rects_to_draw := xs;
        pr2 (spf "Drawing: %s" (x.T.tr_label));
        paint_content_maybe_rect ~user_rect dw model x;
  done;
  !Ctl._refresh_da ();
  if !Ctl.current_rects_to_draw = []
```

```
  then begin
    !Ctl.hook_finish_paint ();
    !Ctl._refresh_da ();
    false
  end
  (* call me again *)
  else true

let paint2 dw model =
  pr2 (spf "paint");

  !Ctl.paint_content_maybe_refresher +> Common.do_option GMain.Idle.remove;
  Ctl.current_rects_to_draw := [];

  let user_rect = device_to_user_area dw in
  pr2 (F.s_of_rectangle user_rect);
  let nb_rects = dw.nb_rects in
  let rects = dw.treemap in

  with_map dw (fun cr ->

    (if not (Layer_code.has_active_layers dw.layers)
     (* phase 1, draw the rectangles *)
     then rects +> List.iter (Draw_macrolevel.draw_treemap_rectangle ~cr)
     else rects +> List.iter (Draw_macrolevel.draw_trect_using_layers
                                ~cr dw.layers)
    );

    (* phase 2, draw the labels, if have enough space *)
    rects +> List.iter (Draw_labels.draw_treemap_rectangle_label_maybe
                          ~cr ~zoom:1.0  ~color:None);
  );

  (* phase 3, draw the content, if have enough space *)
  if nb_rects < !Flag.threshold_nb_rects_draw_content
  (* draw_content_maybe calls nblines which is quite expensive so
   * want to limit it *)
  then begin
    Ctl.current_rects_to_draw := rects;
    Ctl.paint_content_maybe_refresher :=
        Some (Gui.gmain_idle_add ~prio:3000 (lazy_paint user_rect dw model));
  end

let paint a b =
  Common.profile_code "View.paint" (fun () -> paint2 a b)
```

10.2 Mixed view

40a ⟨*draw_summary_content* 40a⟩≡

40b ⟨*hfiles_and_top_entities sig* 40b⟩≡
```
val hfiles_and_top_entities :
  Common.path -> Database_code.database option ->
  (Common.filename, Database_code.entity list) Hashtbl.t
```

40c ⟨*hfiles_and_top_entities()* 40c⟩≡
```
(* used in the summary mixed mode *)
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
```

40d ⟨*model fields hook* 12c⟩+≡

11 Layers

11.1 Micro-level

```
This is kind of intra-file "aspect"
```

11.2 Macro-level

```
This is kind of inter-file "aspect". See also
the regular archi_code treemap which is also
about aspects.
```

11.3 UI

41a ⟨*ui_layers.mli* 41a⟩≡
```
val choose_layer:
  root:Common.dirname ->
  string option (* layer title we want *) ->
```

```
    Model2.world ->
    unit
```

41b ⟨*ui_layers.ml* 41b⟩≡

```
  ⟨Facebook copyright 2⟩
  open Common

  open Model2
  module M = Model2
  module Controller = Controller2

  module L = Layer_code

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

  let choose_layer ~root layer_title_opt w =
    pr2 "choose_layer()";
    let dw = w.dw in

    let original_layers = dw.M.layers.L.layers +> List.map fst in
    let layers_idx =
      Layer_code.build_index_of_layers
        ~root
        (original_layers +> List.map (fun layer ->
          layer,
          match layer_title_opt with
          | None -> false
          | Some title -> title =$= layer.L.title
        ))
    in
    w.dw <-
      Model2.init_drawing
        ~width:dw.width
        ~height:dw.height
        w.treemap_func
        layers_idx
        [root]
        root;
    View_mainmap.paint w.dw w.model;
    !Controller._refresh_da ();
    !Controller._refresh_legend ();
    ()
```

12 Navigation

42a ⟨*key_pressed* 42a⟩≡

42b ⟨*find_filepos_in_rectangle_at_user_point* 42b⟩≡

42c ⟨*text_with_user_pos sig* 42c⟩≡

42d ⟨*button_action* 42d⟩≡

```
let button_action w ev =
  let dw = w.dw in

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

  match GdkEvent.get_type ev with
  | `BUTTON_PRESS ->
      let button = GdkEvent.Button.button ev in
      let state = GdkEvent.Button.state ev in
      pr2 (spf "button %d pressed" button);
      (match button with
      | 1 ->
        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);
          let line =
            M.find_line_in_rectangle_at_user_point user r dw ||| (Line 0)
          in
          Editor_connection.open_file_in_current_editor ~file ~line;
        );
        true

      | 3 ->
        r_opt +> Common.do_option (fun (tr, _, _r_englobing) ->
          (* actually file_or_dir *)
          let file = tr.T.tr_label in

          if not (Gdk.Convert.test_modifier `SHIFT state)
          then !Ctl._go_dirs_or_file w [file]
```

```
else begin

let model = Async.async_get w.model in

(* similar to View_overlays.motion.refresher *)
let line_opt =
  M.find_line_in_rectangle_at_user_point user tr dw in
let glyph_opt =
  M.find_glyph_in_rectangle_at_user_point user tr dw in
let entity_def_opt =
  line_opt >>= (fun line ->
    M.find_def_entity_at_line_opt line tr dw model) in
let entity_use_opt =
  line_opt >>= (fun line ->
  glyph_opt >>= (fun glyph ->
    M.find_use_entity_at_line_and_glyph_opt line glyph tr dw model))
in
let entity_opt =
  match entity_use_opt, entity_def_opt with
  (* priority to use *)
  | Some e, Some _ -> Some e
  | Some e, _ | _, Some e -> Some e
  | _ -> None
in
let n = entity_opt ||| M.node_of_rect tr model in

let uses, users = M.deps_readable_files_of_node n model in

let paths_of_readables xs =
  xs
  +> List.sort Pervasives.compare
  +> Common2.uniq
  (* todo: tfidf to filter files like common2.ml *)
  +> Common.exclude (fun readable ->
      readable =~ "commons/common2.ml"
      (*readable =~ "external/.*"  *)
  )
  +> List.map (fun s -> Filename.concat model.root s)
  (* less: print a warning when does not exist? *)
  +> List.filter Sys.file_exists
in
let readable = Common.readable ~root:model.root file in
let readable =
  match entity_opt with
  | None -> readable
  | Some n ->
```

```
      let g = Common2.some model.g in
      try Graph_code.file_of_node n g
      with Not_found -> readable
in
let entries = [
  `I ("go to file", (fun () ->
    !Ctl._go_dirs_or_file w (paths_of_readables [readable]);));
  `I ("deps inout", (fun () ->
    w.current_node_selected <- entity_opt;
    !Ctl._go_dirs_or_file w (paths_of_readables
                               (uses @ users @ [readable]))));
  `I ("deps in (users)", (fun () ->
    w.current_node_selected <- entity_opt;
    !Ctl._go_dirs_or_file w (paths_of_readables (users@[readable]))));
  `I ("deps out (uses)", (fun () ->
    w.current_node_selected <- entity_opt;
    !Ctl._go_dirs_or_file w (paths_of_readables (uses@[readable]))));
  `I ("info file", (fun () ->
    let microlevel = Hashtbl.find dw.microlevel tr in
    let defs = microlevel.defs in
    let buf = Buffer.create 100 in
    let prx s = Buffer.add_string buf (s ^ "\n") in
    prx "short defs";
    defs +> List.iter (fun (_line, short_node) ->
        prx (" " ^ Graph_code.string_of_node short_node)
    );
    Gui.dialog_text ~text:(Buffer.contents buf) ~title:"Info file";
  ));
] in
let entries =
  entries @
  (match entity_opt with
  | None -> []
  | Some n ->
      let g = Common2.some model.g in
      [`I ("info entity", (fun () ->
        let users = Graph_code.pred n (Graph_code.Use) g in
        let str =
          ([Graph_code.string_of_node n] @
          (users +> List.map Graph_code.string_of_node )
          ) +> Common.join "\n"
        in
        Gui.dialog_text ~text:str ~title:"Info entity";
      ));
       `I ("goto def", (fun () ->
         w.current_node_selected <- entity_opt;
```

```
                  let dest = Graph_code.file_of_node n g in
                  !Ctl._go_dirs_or_file w (paths_of_readables [dest])
                ));
              ]
          )
        in
        GToolbox.popup_menu ~entries ~button:3
          ~time:(GtkMain.Main.get_current_event_time());
        end
      );
      true
    | _ -> false
    )
| `BUTTON_RELEASE ->
    let button = GdkEvent.Button.button ev in
    pr2 (spf "button %d released" button);
    (match button with
    | 1 -> 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
      !Ctl._go_dirs_or_file w [path];
    );
    true
| _ -> false
```

12.1 Zooming in a directory

45 ⟨*go_dirs_or_file* 45⟩≡

```
let go_dirs_or_file ?(current_grep_query=None) w paths =
  let root = Common2.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 = w.dw in
  !Controller._set_title (Controller.title_of_path root);

  Common.push w.dw w.dw_stack;
  w.dw <-
```

```
    Model2.init_drawing
      ~width:dw.width
      ~height:dw.height
      w.treemap_func
      dw.layers
      paths
      w.root_orig;
  (match current_grep_query with
  | Some h -> w.dw.current_grep_query <- h;
  (* wants to propagate the query so when right-click the query
   * is still there  *)
  | None -> w.dw.current_grep_query <- dw.current_grep_query;
  );
  View_overlays.paint_initial w.dw;
  View_mainmap.paint w.dw w.model;
  !Controller._refresh_da ();
  ()
```

46a ⟨*find_rectangle_at_user_point sig* 46a⟩≡

```
  val find_rectangle_at_user_point :
    Cairo.point -> drawing ->
    (Treemap.treemap_rectangle * (* most precise *)
     Treemap.treemap_rectangle list * (* englobbing ones *)
     Treemap.treemap_rectangle (* top one *)
    ) option
```

46b ⟨*find_rectangle_at_user_point()* 46b⟩≡

```
  (* alt: we could use Cairo_bigarray and the pixel trick below if
   * it takes too long to detect which rectangle is under the cursor.
   * We could also sort the rectangles ... or have some kind of BSP.
   * Add in model:
   *   mutable pm_color_trick: GDraw.pixmap;
   *   mutable pm_color_trick_info: (string) array.
   *
   * current solution: just find pixel by iterating over all the rectangles
   * and check if it's inside.
   *)
  let find_rectangle_at_user_point2 user dw =
    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)
      (* opti: this should be far faster by using a quad tree to represent
       * the treemap
       *)
      +> Common.sort_by_val_highfirst
      +> List.map fst
     in
     match matching_rects with
     | [] -> None
     | [x] -> Some (x, [], x)
     | _ -> Some (Common2.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)
```

12.2 Zooming in multiple directories

```
see multi dir search
```

12.3 Zooming in a file

12.4 Opening a file in an external editor

47a ⟨*editor_connection.mli* 47a⟩≡

```
  val open_file_in_current_editor: file:string -> line:Model2.line -> unit
```

47b ⟨*emacs configuration* 47b⟩≡

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

48a ⟨*open_file_in_current_editor()* 48a⟩≡

```
let open_file_in_current_editor ~file ~line =
  let (Model2.Line line) = line in
  (* emacs line numbers start at 1 *)
  let line = line + 1 in
  run_emacsclient ~file ~line
```

12.5 Going back

48b ⟨*go_back* 48b⟩≡

```
let go_back w =
  (* reset also the motion notifier ? less needed because
   * the next motion will reset it
   *)
  !Controller.paint_content_maybe_refresher +> Common.do_option (fun x ->
    GMain.Idle.remove x;
  );
  let old_dw = Common2.pop2 w.dw_stack in
  w.dw <- old_dw;

  let path = w.dw.current_root in
  !Controller._set_title (Controller.title_of_path path);
  !Controller._refresh_da();
  ()
```

12.6 Magnifying glass

48c ⟨*idle* 48c⟩≡

12.7 Minimap

48d ⟨*motion_notify_minimap* 48d⟩≡

48e ⟨*button_action_minimap* 48e⟩≡

49a ⟨*paint_minimap* 49a⟩≡

49b ⟨*expose_minimap* 49b⟩≡

49c ⟨*configure_minimap* 49c⟩≡

49d ⟨*with_minimap* 49d⟩≡

49e ⟨*fields drawing minimap* 49e⟩≡

49f ⟨*scale_minimap* 49f⟩≡

12.8 Fine-grained pan and zoom

49g ⟨*with_map* 49g⟩≡
```
let with_map dw f =
  let cr = Cairo.create dw.base in
  zoom_pan_scale_map cr dw;
  f cr
```

49h ⟨*zoom_pan_scale_map* 49h⟩≡
```
let zoom_pan_scale_map cr dw =
  Cairo.scale cr
    (float_of_int dw.width / T.xy_ratio)
    (float_of_int dw.height)
  ;
  (* I first scale and then translate as the xtrans are in user coordinates *)
  Cairo.translate cr 0.0 0.0;
  ()
```

49i ⟨*fields drawing viewport* 49i⟩≡
```
(* viewport, device coordinates *)
mutable width: int;
mutable height: int;
```

13 Search

49j ⟨*dialog_search_def* 49j⟩≡
```
let dialog_search_def model =
  let idx = (fun () ->
    let model = Async.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
```

50 ⟨*run_grep_query* 50⟩≡

```
  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, (M.Line (lineno - 1))
      else
        failwith ("wrong git grep line: " ^ s)
    ) in
    xs
```

51a ⟨*run_tbgs_query* 51a⟩≡

```
  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, (M.Line (lineno - 1))
      else
        failwith ("wrong tbgs line: " ^ s)
    ) in
    xs
```

51b ⟨*fields drawing query stuff* 51b⟩≡

```
  (* queries *)
  mutable current_query: string;
  mutable current_searched_rectangles: Treemap.treemap_rectangle list;
  mutable current_grep_query: (Common.filename, line) Hashtbl.t;
```

13.1 Definition search

```
%tags-like

% php manual integration!
```

51c ⟨*all_entities sig* 51c⟩≡

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

13.1.1 Completion building

51d ⟨*completion2.mli* 51d⟩≡

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

52a ⟨*build_completion_defs_index* 52a⟩≡

```
  (* 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
```

13.1.2 Completion window

52b ⟨*completion2.mli* 51d⟩+≡

```
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
```

52c ⟨*model fields hook* 12c⟩+≡

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

```
%tags-like

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

52d ⟨*all_entities* 52d⟩≡

```
(* To get completion for functions/class/methods/files/directories.
 *
 * We pass the root in addition to the db_opt because sometimes we
 * don't have a db but we still want to 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 ~root files db_opt =
  match db_opt with
  | None ->
      let db = Database_code.files_and_dirs_database_from_files ~root files in
      Database_code.files_and_dirs_and_sorted_entities_for_completion
        ~threshold_too_many_entities:!Flag.threshold_too_many_entities
        db
```

```
  | 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);

      (* the db passed might be just about .ml files but we could be
       * called on a directory with non .ml files that we would
       * still want to quicky jump too hence the need to include
       * other regular files and dirs
       *)
      let db2 = Database_code.files_and_dirs_database_from_files ~root files in
      let db = Database_code.merge_databases db db2 in

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

53 ⟨*completion2.ml* 53⟩≡

```
⟨Facebook copyright 2⟩
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* 52a⟩

```
(*****************************************************************************)
(* 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
   * todo? class vs interface ?
   *)
  | Db.Class -> `CONNECT
  | Db.Module -> `DISCONNECT
  | Db.Package -> `DIRECTORY
  | Db.Type -> `PROPERTIES
  | Db.Constant -> `CONNECT
  | Db.Global -> `MEDIA_RECORD
  | Db.Method -> `CONVERT

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

  (* todo *)
  | Db.ClassConstant -> `CONNECT
  | Db.Field -> `CONNECT
  | Db.Macro -> `CONNECT
  | Db.Exception -> `CONNECT
  | Db.Constructor -> `CONNECT
  | Db.Prototype -> `CONNECT
```

```
  | Db.GlobalExtern -> `CONNECT

  | (Db.TopStmts | Db.Other _ ) -> raise Todo


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
    (* if the string is too long, we will not see the other properties *)
    let final_name =
      try (String.sub name 0 30) ^ "..."
      with Invalid_argument _ -> 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 = final_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_code "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 = { Common2.l = -1; c = -1 };
    e_number_external_users = 0;
    e_good_examples_of_use = [];
    e_properties = [];
}

let my_entry_completion_eff2 ~callback_selected ~callback_changed fn_idx =

  let entry = GEdit.entry ~width:500 () 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
          (G.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
```

13.2 Use Search, aka Visual grep

```
%cscope-like
% see also layers
```

13.3 Directory search

```
%multi dirs
```

13.4 Multi-directories

13.5 Example search

```
%test search

% pleac integration!
```

14 Overlays

14.1 Cairo overlays

```
aka layers
similar but different from the  layers in prevision section.
Here low level graphics layers.
```

60 ⟨*fields drawing main view* 60⟩≡

```
(* device coordinates *)
(* first cairo layer, for heavy computation e.g. the treemap and content*)
mutable base: [ `Any ] Cairo.surface;
(* second cairo layer, when move the mouse *)
mutable overlay: [ `Any ] Cairo.surface;
(* todo? third cairo layer? for animations and time related graphics such
 * as tooltips, glowing rectangles, etc?
 *)
```

14.2 Rectangle overlay

61a ⟨*draw_rectangle_overlay* 61a⟩≡

```
let draw_englobing_rectangles_overlay ~dw (r, middle, r_englobing) =
 with_overlay dw (fun cr_overlay ->
  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_labels.draw_treemap_rectangle_label_maybe
```

```
    ~cr:cr_overlay ~color:(Some "red") ~zoom:1.0 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_labels.draw_treemap_rectangle_label_maybe
      ~cr:cr_overlay ~color:(Some color) ~zoom:1.0 r;
  );
 )
```

14.3 Label overlay

61b ⟨*draw_label_overlay* 61b⟩≡

```
 (* assumes cr_overlay has not been zoom_pan_scale *)
 let draw_label_overlay ~cr_overlay ~x ~y txt =

   Cairo.select_font_face cr_overlay "serif"
     Cairo.FONT_SLANT_NORMAL Cairo.FONT_WEIGHT_NORMAL;
   Cairo.set_font_size cr_overlay Style.font_size_filename_cursor;

   let extent = CairoH.text_extents cr_overlay txt in
   let tw = extent.Cairo.text_width in
   let th = extent.Cairo.text_height in

   let refx = x - tw / 2. in
   let refy = y in

   CairoH.fill_rectangle ~cr:cr_overlay
     ~x:(refx + extent.Cairo.x_bearing) ~y:(refy + extent.Cairo.y_bearing)
     ~w:tw ~h:(th * 1.2)
     ~color:"black"
     ~alpha:0.5
     ();

   Cairo.move_to cr_overlay refx refy;
   Cairo.set_source_rgba cr_overlay 1. 1. 1.    1.0;
   CairoH.show_text cr_overlay txt;
   ()
```

14.4 Searched files overlay

62a ⟨*draw_searched_rectangles* 62a⟩≡

```
let draw_searched_rectangles ~dw =
 with_overlay dw (fun cr_overlay ->
  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
   *)
 )
```

14.5 Zoom overlay

62b ⟨*zoomed_surface_of_rectangle* 62b⟩≡

14.6 Assembling overlays

62c ⟨*motion_refresher* 62c⟩≡

```
let motion_refresher ev w =
  paint_initial w.dw;
  hook_finish_paint w;
  let dw = w.dw in
  let cr_overlay = Cairo.create dw.overlay in

  (* some similarity with View_mainmap.button_action handler *)
  let x, y = GdkEvent.Motion.x ev, GdkEvent.Motion.y ev in
  let pt = { Cairo. x = x; y = y } in
  let user = View_mainmap.with_map dw (fun cr -> Cairo.device_to_user cr pt) in
  let r_opt = M.find_rectangle_at_user_point user dw in

  r_opt +> Common.do_option (fun (tr, middle, r_englobing) ->
    (* coupling: similar code in right click handler in View_mainmap *)
    let line_opt =
      M.find_line_in_rectangle_at_user_point user tr dw in
    let glyph_opt =
```

```
  M.find_glyph_in_rectangle_at_user_point user tr dw in

let entity_def_opt =
  Async.async_get_opt w.model >>= (fun model ->
  line_opt >>= (fun line ->
    M.find_def_entity_at_line_opt line tr dw model)) in
let entity_use_opt =
  Async.async_get_opt w.model >>= (fun model ->
  line_opt >>= (fun line ->
  glyph_opt >>= (fun glyph ->
    M.find_use_entity_at_line_and_glyph_opt line glyph tr dw model)))
in
let entity_opt =
  match entity_use_opt, entity_def_opt with
  (* priority to use *)
  | Some e, Some _ -> Some e
  | Some e, _ | _, Some e -> Some e
  | _ -> None
in

let statusbar_txt =
  tr.T.tr_label ^
  (match line_opt with None -> "" | Some (Line i) ->
    spf ":%d" i) ^
  (match glyph_opt with None -> "" | Some glyph ->
    spf "[%s]" glyph.str) ^
  (match entity_def_opt with None -> "" | Some n ->
    spf "(%s)" (Graph_code.string_of_node n)) ^
  (match entity_use_opt with None -> "" | Some n ->
    spf "{%s}" (Graph_code.string_of_node n))
in
!Controller._statusbar_addtext statusbar_txt;

(match line_opt with
| None ->
    let label_txt = readable_txt_for_label tr.T.tr_label dw.current_root in
    draw_label_overlay ~cr_overlay ~x ~y label_txt
| Some line ->
    let microlevel = Hashtbl.find dw.microlevel tr in
    draw_magnify_line_overlay_maybe ~honor_color:true dw line microlevel
);

draw_englobing_rectangles_overlay ~dw (tr, middle, r_englobing);
Async.async_get_opt w.model +> Common.do_option (fun model ->
  draw_deps_files tr dw model;
  entity_opt +> Common.do_option (fun _n -> w.current_node <- None);
```

```
      entity_def_opt+>Common.do_option (fun n -> draw_deps_entities n dw model);
      entity_use_opt+>Common.do_option (fun n -> draw_deps_entities n dw model);
    );

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

    !Controller.current_tooltip_refresher
    +>Common.do_option GMain.Timeout.remove;
    Controller.current_tooltip_refresher :=
      Some (Gui.gmain_timeout_add ~ms:1000 ~callback:(fun _ ->
        Async.async_get_opt w.model +> Common.do_option (fun model ->
          match entity_opt, model.g with
          | Some node, Some g ->
            draw_tooltip ~cr_overlay ~x ~y node g;
            !Controller._refresh_da ();
          | _ -> ()
          ;
        );
        (* do not run again *)
        false
      ));

    Controller.current_r := Some tr;
  );
  !Controller._refresh_da ();
  false


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

  (* The motion code now takes time, so it's better do run it when the user
   * has finished moving his mouse, hence the use of gmain_idle_add below.
   *)
  !Controller.current_motion_refresher +> Common.do_option GMain.Idle.remove;
  Controller.current_motion_refresher :=
    Some (Gui.gmain_idle_add ~prio:200 (fun () -> motion_refresher ev w));
  true
```

15 Final Rendering

15.1 The big picture

15.2 The configure event

65a ⟨*configure* 65a⟩≡

```
let configure2_bis w ev =
  let dw = w.dw in
  let width = GdkEvent.Configure.width ev in
  let height = GdkEvent.Configure.height ev in
  dw.width <- width;
  dw.height <- height;
  dw.base <- Model2.new_surface ~alpha:false ~width ~height;
  dw.overlay <- Model2.new_surface ~alpha:true ~width ~height;
  View_mainmap.paint dw w.model;
  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 =
  (* 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

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

15.3 The expose event

65b ⟨*assemble_layers* 65b⟩≡

```
(* Composing the "layers". See cairo/tests/knockout.ml example.
 * Each move of the cursor will call assemble_layers which does all
 * those pixels copying (which is very fast).
 *
 * The final target is the actual gtk window which is represented by cr_final.
 * We copy the pixels from the pixmap dw.pm on the window. Then
 * we copy the pixels from the pixmap dw.overlay on the window
 * getting the final result.
 *)
```

```
let assemble_layers cr_final dw =
  let surface_src = dw.base 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;
  ()
```

66a ⟨*expose* 66a⟩≡

```
(* opti: don't 'paint dw;' painting is the computation
 * heavy function. expose() just copy the "canvas" layers.
 *)
let expose2 da w _ev =
  let dw = w.dw in
  let gwin = da#misc#window in
  let cr = Cairo_lablgtk.create gwin in
  assemble_layers cr dw;
  true

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

15.4 Trace: clicking a directory

diagram where see events, functions, draw vs paint vs overlays.

15.5 Trace: moving the mouse

diagram where see events, functions, draw vs paint vs overlays.

16 Language Modes

16.1 Parsing

66b ⟨*parsing2.mli* 66b⟩≡

```
(* internally memoize the parsing part in _hmemo_file *)
val tokens_with_categ_of_file:
  Common.filename ->
  (string, Database_code.entity) Hashtbl.t ->
  (string * Highlight_code.category option * Common2.filepos) list
```

```
(* helpers *)
val use_arity_of_use_count:
  int -> Highlight_code.use_arity
```

16.2 Generic semantic visual feedback

67 ⟨*parsing2.ml* 67⟩≡

```
⟨Facebook copyright 2⟩
open Common

module FT = File_type
module PI = Parse_info
module HC = Highlight_code
module Db = Database_code
module Flag = Flag_visual

open Highlight_code

(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)
(*
 * The main entry point of this module is tokens_with_categ_of_file
 * which is called in Draw_microlevel to "render" the content of a file.
 *)

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

(* This type is needed if we want to use a single hashtbl to memoize
 * all the parsed file.
 *)
type ast =
  | ML  of Parse_ml.program_and_tokens
  | Hs  of Parse_hs.program2

  | Html of Parse_html.program2
  | Js  of Parse_js.program_and_tokens
  | Php of Parse_php.program_with_comments

  | Opa of Parse_opa.program_with_tokens
```

```
  | Cpp of Parse_cpp.program2

  | Csharp of Parse_csharp.program_and_tokens
  | Java of Parse_java.program_and_tokens

  | Lisp of Parse_lisp.program2
  | Erlang of Parse_erlang.program2

  | Python of Parse_python.program2

  | Noweb of Parse_nw.program2

  (* less? | Org of Org_mode.org ? *)

let _hmemo_file = Hashtbl.create 101

(* with directories with many files, this is useful *)
let parse_cache parse_in extract file =
  Common.profile_code "View.parse_cache" (fun () ->
    let mtime = Common2.filemtime file in
    let recompute =
      if Hashtbl.mem _hmemo_file file
      then
        let (oldmtime, _ast) = Hashtbl.find _hmemo_file file in
        mtime > oldmtime
      else true
    in
    let ast =
      if recompute
      then begin
        let ast = parse_in file in
        Hashtbl.replace _hmemo_file file (mtime, ast);
        ast
      end
      else Hashtbl.find _hmemo_file file +> snd
    in
    extract ast
  )
(*****************************************************************************)
(* 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 =
  match Db.entity_kind_of_highlight_category_def categ 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 *)
        Database_code.matching_def_short_kind_kind e_kind e.Db.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
          if Database_code.is_entity_def_category categ
          then HC.rewrap_arity_def2_category arity categ
          else 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

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

```
(* Helpers *)
(*****************************************************************************)
type ('ast, 'token) for_helper = {
  parse: (Common.filename -> ('ast * 'token list) list);
  highlight_visit:(tag_hook:(Parse_info.info -> HC.category -> unit) ->
                   Highlight_code.highlighter_preferences ->
                   'ast * 'token list -> unit);
  info_of_tok:('token -> Parse_info.info);
}

let tokens_with_categ_of_file_helper
  {parse; highlight_visit; info_of_tok} file prefs hentities =

  if !Flag.verbose_visual then pr2 (spf "Parsing: %s" file);
  let ast2 = parse file in

  if !Flag.verbose_visual then pr2 (spf "Highlighting: %s" file);
  (* todo: ast2 should not be a list, should just be (ast, toks)
   * but right now only a few parsers will satisfy this interface
   *)
  ast2 +> List.map (fun (ast, toks) ->

    let h = Hashtbl.create 101 in

    (* computing the token attributes *)
    highlight_visit ~tag_hook:(fun info categ -> Hashtbl.add h info categ)
      prefs (ast, toks);

    (* getting the text *)
    toks +> Common.map_filter (fun tok ->
      let info = info_of_tok tok in
      let s = PI.str_of_info info in
      if not (PI.is_origintok info)
      then None
      else
        let categ = Common2.hfind_option info h +> Common2.fmap (fun categ ->
          rewrite_categ_using_entities s categ file hentities
        ) in
        Some (s, categ,{ Common2.l = PI.line_of_info info; c = PI.col_of_info info; })
    )) +> List.flatten

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

(* coupling: right now if you add a language here, you need to whitelist it
```

```
 * also in draw_microlevel.draw_contents2.
 *)
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 _)) ->
      tokens_with_categ_of_file_helper
        { parse = (parse_cache (fun file ->
          Common.save_excursion Flag_parsing_php.error_recovery true (fun () ->
            let ((ast, toks), _stat) = Parse_php.parse file in
            (* todo: use database_light if given? we could so that
             * variables are better annotated.
             * note that database_light will be passed in
             * rewrite_categ_using_entities() at least.
             *)
            let find_entity = None in
            (* work by side effect on ast2 too *)
            (try
            Check_variables_php.check_and_annotate_program
              find_entity
              ast
             with Ast_php.TodoNamespace _ | Common.Impossible -> ()
            );
            Php ((ast, toks))
          ))
         (function Php (ast, toks) -> [ast, toks] | _ -> raise Impossible));
         highlight_visit = (fun ~tag_hook prefs (ast, toks) ->
          Highlight_php.visit_program ~tag:tag_hook prefs hentities
            (ast, toks)
         );
         info_of_tok = Token_helpers_php.info_of_tok;
        }
        file prefs hentities

  | FT.PL (FT.ML _) ->
      tokens_with_categ_of_file_helper
        { parse = (parse_cache (fun file ->
           Common.save_excursion Flag_parsing_ml.error_recovery true (fun()->
             ML (Parse_ml.parse file +> fst))
         )
         (function
         | ML (astopt, toks) ->
             let ast = astopt ||| [] in
             [ast, toks]
```

```
       | _ -> raise Impossible));
      highlight_visit = (fun ~tag_hook prefs (ast, toks) ->
        Highlight_ml.visit_program ~tag_hook prefs (ast, toks));
      info_of_tok = Token_helpers_ml.info_of_tok;
      }
      file prefs hentities

| FT.PL (FT.Haskell _) ->
    tokens_with_categ_of_file_helper
      { parse = (parse_cache
       (fun file -> Hs (Parse_hs.parse file +> fst))
       (function Hs x -> x | _ -> raise Impossible));
      highlight_visit = (fun ~tag_hook prefs (ast, toks) ->
        Highlight_hs.visit_toplevel ~tag_hook prefs (ast, toks));
      info_of_tok = Parser_hs.info_of_tok;
      }
      file prefs hentities

| FT.PL (FT.Python) ->
    tokens_with_categ_of_file_helper
      { parse = (parse_cache
       (fun file -> Python (Parse_python.parse file +> fst))
       (function Python x -> x | _ -> raise Impossible));
      highlight_visit = (fun ~tag_hook prefs (ast, toks) ->
        Highlight_python.visit_toplevel ~tag_hook prefs (ast, toks));
      info_of_tok = Token_helpers_python.info_of_tok;
      }
      file prefs hentities

| FT.PL (FT.Csharp) ->
    tokens_with_categ_of_file_helper
      { parse = (parse_cache
       (fun file -> Csharp (Parse_csharp.parse file +> fst))
       (function Csharp (ast, toks) -> [ast, toks] | _ -> raise Impossible));
      highlight_visit = (fun ~tag_hook prefs (ast, toks) ->
        Highlight_csharp.visit_program ~tag_hook prefs (ast, toks));
      info_of_tok = Token_helpers_csharp.info_of_tok;
      }
      file prefs hentities

| FT.PL (FT.Opa) ->
    tokens_with_categ_of_file_helper
      { parse = (parse_cache
       (fun file -> Opa (Parse_opa.parse_just_tokens file))
       (function
       | Opa (ast, toks) -> [ast, toks]
```

```
       | _ -> raise Impossible));
      highlight_visit = Highlight_opa.visit_toplevel;
      info_of_tok = Token_helpers_opa.info_of_tok;
      }
      file prefs hentities

| FT.PL (FT.Erlang) ->
    tokens_with_categ_of_file_helper
      { parse = (parse_cache
       (fun file -> Erlang (Parse_erlang.parse file +> fst))
       (function Erlang x -> x | _ -> raise Impossible));
      highlight_visit = Highlight_erlang.visit_toplevel;
      info_of_tok = Token_helpers_erlang.info_of_tok;
      }
      file prefs hentities

| FT.PL (FT.Java) ->
    tokens_with_categ_of_file_helper
      { parse = (parse_cache
       (fun file -> Java (Parse_java.parse file +> fst))
        (function
        | Java (ast, toks) -> [Common2.some ast, (toks)]
        | _ -> raise Impossible));
      highlight_visit = Highlight_java.visit_toplevel;
      info_of_tok = Token_helpers_java.info_of_tok;
      }
      file prefs hentities

| FT.PL (FT.Lisp _) ->
    tokens_with_categ_of_file_helper
      { parse = (parse_cache
       (fun file -> Lisp (Parse_lisp.parse file +> fst))
       (function Lisp x -> x | _ -> raise Impossible));
      highlight_visit = Highlight_lisp.visit_toplevel;
      info_of_tok = Parser_lisp.info_of_tok;
      }
      file prefs hentities

| FT.Text ("nw" | "tex" | "texi" | "web") ->
    tokens_with_categ_of_file_helper
      { parse = (parse_cache
       (fun file -> Noweb (Parse_nw.parse file +> fst))
       (function Noweb x -> x | _ -> raise Impossible));
      highlight_visit = Highlight_nw.visit_toplevel;
      info_of_tok = Token_helpers_nw.info_of_tok;
      }
```

```
      file prefs hentities

  | FT.PL (FT.Cplusplus _ | FT.C _ | FT.Thrift | FT.ObjectiveC _) ->
      tokens_with_categ_of_file_helper
        { parse = (parse_cache
         (fun file ->
           let (ast2, _stat) = Parse_cpp.parse file in
           let ast = Parse_cpp.program_of_program2 ast2 in
           (* work by side effect on ast2 too *)
           Check_variables_cpp.check_and_annotate_program
             ast;
           Cpp ast2
         )
         (function Cpp x -> x | _ -> raise Impossible));
        highlight_visit = Highlight_cpp.visit_toplevel;
        info_of_tok = Token_helpers_cpp.info_of_tok;
        }
        file prefs hentities

  | FT.PL (FT.Web (FT.Js)) ->
      tokens_with_categ_of_file_helper
        { parse = (parse_cache
          (fun file ->
            Common.save_excursion Flag_parsing_js.error_recovery true (fun () ->
              Js (Parse_js.parse file +> fst))
          )
         (function
         | Js (astopt, toks) ->
             let ast = astopt ||| [] in
             [ast, toks]
         | _ -> raise Impossible
         ));
        highlight_visit = Highlight_js.visit_program;
(* TODO?
          let s = Token_helpers_js.str_of_tok tok in
          Ast_js.remove_quotes_if_present s
*)
        info_of_tok = Token_helpers_js.info_of_tok;
        }
        file prefs hentities

  | FT.PL (FT.Web (FT.Html)) ->
      tokens_with_categ_of_file_helper
        { parse = (parse_cache
          (fun file -> Html (Parse_html.parse file))
          (function
```

```
          | Html (ast, toks) -> [ast, toks]
          | _ -> raise Impossible));
        highlight_visit = Highlight_html.visit_toplevel;
        info_of_tok = Token_helpers_html.info_of_tok;
        }
        file prefs hentities

  | FT.Text ("org") ->
      let org = Org_mode.parse file in
      Org_mode.highlight org

  (* ugly, hardcoded, should instead look at the head of the file for a
   * # -*- org   indication.
   * very pad and code-overlay specific.
   *)
  | FT.Text ("txt") when Common2.basename file =$= "info.txt" ->
      let org = Org_mode.parse file in
      Org_mode.highlight org

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

16.3 OCaml

16.4 Tex/Latex/NoWeb

16.5 PHP

16.6 Javascript

16.7 C/C++ and variants

16.8 Haskell

16.9 Lisp/Scheme

17 Optimisations

17.1 Threads, Idle, Timeouts

75 ⟨*type async* 75⟩≡

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

76a ⟨*async functions sig* 76a⟩≡

```
val async_make: unit -> 'a t
val async_get: 'a t -> 'a
val async_set: 'a -> 'a t -> unit
val async_ready: 'a t -> bool
val async_get_opt: 'a t -> 'a option

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

76b ⟨*async functions* 76b⟩≡

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

let with_lock 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
  with_lock (fun () -> go a) a.m

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

let async_ready a =
  (* actually I don't think you need the lock *)
  with_lock (fun () ->
```

```
      match !(a.v) with
      | Some _ -> true
      | None -> false
    ) a.m

  let async_get_opt a =
    if async_ready a
    then Some (async_get a)
    else None
```

18 Configuration

77 ⟨*options* 77⟩≡

```
    "-screen_size", Arg.Set_int screen_size,
    " <int> (1 = small, 2 = big)";
    "-ss", Arg.Set_int screen_size,
    " <int> alias for -screen_size";
    "-no_legend", Arg.Clear legend,
    " do not display the legend";

    "-symlinks", Arg.Unit (fun () -> Treemap.follow_symlinks := true;),
    " follow symlinks";
    "-no_symlinks", Arg.Unit (fun () -> Treemap.follow_symlinks := false),
    " do not follow symlinks";

    "-with_graph", Arg.String (fun s -> graph_file := Some s),
    " <graph_file> dependency semantic information";
    "-with_db", Arg.String (fun s -> db_file := Some s),
    " <db_file> generic semantic information";
    "-with_layer", Arg.String (fun s -> layer_file := Some s),
    " <layer_file>";
    "-with_layers", Arg.String (fun s -> layer_dir := Some s),
    " <dir_with_layers>";

    "-filter", Arg.String (fun s -> filter := List.assoc s filters;),
     spf " filter certain files (available = %s)"
      (filters +> List.map fst +> Common.join ", ");
    "-extra_filter", Arg.String (fun s -> Flag.extra_filter := Some s),
    " ";
    "-skip_list", Arg.String (fun s -> skip_file := Some s),
    " <file> skip files or directories";

    "-with_info", Arg.String (fun _s -> ()),
    " obsolete\n"; (* for codemap_www in engshare/admin/scripts *)
```

```
  "-ft", Arg.Set_float Flag.threshold_draw_content_font_size_real,
  " <float> threshold to draw content";
  "-boost_lbl", Arg.Set Flag.boost_label_size,
  " boost size of labels";
  "-no_boost_lbl", Arg.Clear Flag.boost_label_size,
  " do not boost labels\n";

(*-------------------------------------------------------------------------*)
(* debugging helpers *)
(*-------------------------------------------------------------------------*)

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

  "-verbose", Arg.Set Flag.verbose_visual,
  " ";
  "-debug_gc", Arg.Set Flag.debug_gc,
  " ";
  "-debug_handlers", Arg.Set Gui.synchronous_actions,
  " ";
  (* "-disable_ancient", Arg.Clear Flag.use_ancient, " "; *)
  "-disable_fonts", Arg.Set Flag.disable_fonts,
  " ";
```

78a ⟨*type settings* 78a⟩≡

78b ⟨*style2.mli* 78b⟩≡

```
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

val font_size_filename_cursor: float

val font_text: string
```

79a ⟨*windows_params()* 79a⟩≡

```
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
```

79b ⟨*size_font_multiplier_of_categ()* 79b⟩≡

```
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

    (* entities defs *)

    | Some (SH.Class SH.Def2 use) -> 5. *. multiplier_use use
    | Some (SH.Module SH.Def) -> 5.
    | Some (SH.Function (SH.Def2 use)) -> 3.5 *. multiplier_use use
    | Some (SH.TypeDef SH.Def) -> 5.
    | Some (SH.Global (SH.Def2 use)) -> 3. *. multiplier_use use
    | Some (SH.FunctionDecl use) -> 2.5 *. multiplier_use use
    | Some (SH.Macro (SH.Def2 use)) -> 2. *. multiplier_use use
    | Some (SH.Constant (SH.Def2 use)) -> 2. *. multiplier_use use
    | Some (SH.Method (SH.Def2 use)) -> 3.5 *. multiplier_use use
    | Some (SH.StaticMethod (SH.Def2 use)) -> 3.5 *. multiplier_use use
    | Some (SH.Field (SH.Def2 use)) -> 1.7 *. multiplier_use use
    | Some (SH.Constructor(SH.Def2 (use))) -> 1.2 *. multiplier_use use
```

```
    | Some (SH.GrammarRule) -> 2.5

    (* entities uses *)
    | Some (SH.Global (SH.Use2 _)) when font_size_real > 7.
         -> 1.5
(*
    | Some (SH.Method (SH.Use2 _)) when font_size_real > 7.
         -> 1.2
*)

    (* "literate programming" *)
    | Some (SH.CommentSection0) -> 5.
    | Some (SH.CommentSection1) -> 3.
    | Some (SH.CommentSection2) -> 2.0
    | Some (SH.CommentSection3) -> 1.2
    | Some (SH.CommentSection4) -> 1.1
    | Some (SH.CommentEstet) -> 1.0
    | Some (SH.CommentCopyright) -> 0.5

    | Some (SH.CommentSyncweb) -> 1.

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

    (* semantic visual feedback *)

    | Some (SH.BadSmell) -> 2.5

    (* ocaml *)
    | Some (SH.UseOfRef) -> 2.

    (* php, C, etc *)
    | Some (SH.PointerCall) -> 3.
    | Some (SH.ParameterRef) -> 2.
    | Some (SH.CallByRef) -> 3.

    (* misc *)
    | Some (SH.Local (SH.Def)) -> 1.2

    | _ ->
        (* the cases above should have covered all the cases *)
        categ +> Common.do_option (fun categ ->
          if Database_code.is_entity_def_category categ
          then failwith "You should update size_font_multiplier_of_categ";
```

```
  );


  1.
```

81a ⟨*zoom_factor_incruste_mode* 81a⟩≡

```
(* 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 *)
```

81b ⟨*threshold_draw_dark_background_font_size_real* 81b⟩≡

```
(* CONFIG *)
let threshold_draw_dark_background_font_size_real = 5.
```

81c ⟨*flag_visual.ml* 81c⟩≡

```
let verbose_visual = ref false

(* It 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.
 * I've recently raised this number because
 * when too low it's just too much noise on the screen.
 * Let's draw the content when you can actually read things and
 * when things don't overlap too much.
 *)
let threshold_draw_content_font_size_real = ref
  2.5

(* big and auto-generated files can take too much time to render *)
let threshold_draw_content_nblines =
  ref 25000.

let threshold_draw_label_font_size_real = ref
  10.

let threshold_nb_rects_draw_content = ref 2500

let threshold_too_many_entities = ref 600000

let top_n = ref 100

let boost_label_size = ref false

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 *)
```

19 Other Features

82a ⟨*visual_commitid() action* 82a⟩≡

```
let test_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
```

82b ⟨*actions* 82b⟩≡

```
    "-test_loc", " ",
    Common.mk_action_1_arg (test_loc);
    "-test_cairo", " ",
    Common.mk_action_0_arg (test_cairo);
    "-test_commitid", " <id>",
    Common.mk_action_1_arg (test_visual_commitid);
    "-test_treemap_dirs", " <id>",
    Common.mk_action_0_arg (test_treemap_dirs);
```

20 Related Work

```
http://www.haskell.org/haskellwiki/Yi

http://www.cse.chalmers.se/~bernardy/FunctionalIncrementalParsing.pdf
"Thanks for the links. The paper is a very interesting reading indeed.
Its main focus is on incrementality (not reparsing the whole buffer at
every keystroke). I'm not so sure how important it is in the context
of the current discussion though: I guess that with an efficient
```

parsing technology and modern computers, parsing even a big buffer at every keystroke should be fast enough. Trivial optimizations like storing the internal state of the parser at some point could also be used if needed. I'm more concerned about the error recovery aspect; the paper suggests the use of annotated error recovery rules, but writing them for a grammar like OCaml's does not seem an easy task at all." - frish

21 Conclusion

Hope you like it.

A Gtk

B Cairo

83a ⟨*new_pixmap sig* 83a⟩≡

```
val new_surface:
  alpha:bool -> width:int -> height:int -> [ `Any ] Cairo.surface
```

83b ⟨*new_pixmap()* 83b⟩≡

```
let new_surface ~alpha ~width ~height =
  let drawable = GDraw.pixmap ~width:1 ~height:1 () in
  drawable#set_foreground `WHITE;
  drawable#rectangle ~x:0 ~y:0 ~width:1 ~height:1 ~filled:true ();

  let cr = Cairo_lablgtk.create drawable#pixmap in
  let surface = Cairo.get_target cr in
  Cairo.surface_create_similar surface
    (if alpha
    then Cairo.CONTENT_COLOR_ALPHA
    else Cairo.CONTENT_COLOR
    ) width height
```

84 ⟨*cairo helpers functions sig* 84⟩≡

```
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 -> string
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
```

85 ⟨*cairo helpers functions* 85⟩≡

```
(* !does side effect on the (mutable) string! *)
let prepare_string s =
  match s with
  | _ when s =~ re_space -> s ^ s (* double it *)
  | _ when s =~ re_tab ->
      Str.global_replace (Str.regexp "\t") "        " 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'
      else
        (* still useful now that have re_tab case above? *)
        if c = '\t'
        then String.set s i ' '
      else ()
    done;
    s

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
    let s' = prepare_string s in
    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_extents cr s =
  Common.profile_code "CairoH.cairo_text_extent" (fun () ->
    (*if s = ""  then fake_text_extents else *)
    Cairo.text_extents cr s
  )

(* just wrap it here so that we can profile it *)
let set_font_size cr font_size =
  Common.profile_code "CairoH.set_font_size" (fun () ->
```

```
    Cairo.set_font_size cr font_size
  )

(*****************************************************************************)
(* Distance conversion *)
(*****************************************************************************)

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

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

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

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

(* 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

(*****************************************************************************)
(* Drawing *)
(*****************************************************************************)

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

C Extra Code

C.1 main_codemap.ml

88 ⟨*main_codemap.ml* 88⟩≡

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

module Flag = Flag_visual
module FT = File_type

module Model = Model2

(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)
(*
 * This is the main entry point of codemap, a semantic source code visualizer
 * using treemaps and code thumbnails. The focus here is code understanding
 * not editing, so for instance even if features like autocompletion are
```

```
* great for editing, they are not really helpful for understanding an existing
* codebase. What can help is completion to help navigate and go from one
* place to another, and this is one of the feature of this tool.
*
* requirements:
*  - get a bird's eye view of all the code (hence treemaps)
*  - get a bird's eye view of a file (hence code thumbnails)
*  - better syntax highlighting than Emacs, using real parsers so
*    we can colorize differently identifiers (a function vs a field vs
*    a constant)
*  - important code should be bigger. Just like in google maps
*    the important roads are more visible. So need some sort of
*    global analysis.
*  - show the data (the source code), but also show the relations
*    (hence codegraph integration)
*  - look at the code through different views (hence layers)
*
* history:
*  - saw Aspect Browser while working on aspects as an intern at IRISA
*  - work on Poffs and idea of visualizing the same code through
*    different views
*  - talked about mixing sgrep/spatch with code visualization,
*    highlighting with a certain color different architecture aspects
*    of the Linux kernel (influenced by work on aspect browser)
*  - talked about fancy code visualizer while at cleanmake with YY,
*    spiros, etc.
*  - saw SeeSoft code visualizer while doing some bibliographic work
*  - saw code thumbnails by MSR, and Rob Deline
*  - saw treemap of Linux kernel by fekete => idea of mixing
*    tree-map+code-thumbnails+seesoft = codemap
*  - saw talk at CC about improving javadoc by putting in bigger fonts
*    really often used API functions => idea of light db and semantic
*    visual feedback
*  - read hierarchical edge bundling paper and its d3 implementation to
*    visualize on top of a treemap the call graph
*
* related work:
*  - racket IDE (was called DrScheme before), had arrows long time ago
*    between occurences of a variable and its definition
*  - http://peaker.github.io/lamdu/, but focused more on AST pretty printing
*  - light table, interesting visualization slice but now focused more
*    on live programming a la Bret Victor
*  - http://www.kickstarter.com/projects/296054304/zeta-code, mostly focused
*    on code relations, so related more to codegraph
*  - textmate, meh
*  - sublime, has thumbnails, but people don't really care about it
```

```
 *  - http://www.hello2morrow.com/products/sotoarc
 *  - http://scg.unibe.ch/codemap
 *  - http://scg.unibe.ch/wiki/projects/rbcrawler, class blueprint
 *  - moose http://youtu.be/yvXm9LC17vk at 14min
 *  - http://redotheweb.com/CodeFlower/
 *  - code swarm, visualize git history, focused on people more than code
 *    https://code.google.com/p/gource/
 *    http://artzub.com/ghv/#repo=d3&climit=100&user=mbostock
 *  - http://www.codetrails.com/ctrlflow, smarter completion by infering
 *    importance of method (like I do, by #times this entity is globally used)
 *
 * features of IDE we do want (e.g. see the list at http://xamarin.com/studio):
 *  - smart syntax highlighting
 *  - go to definition (=~ TAGS, light db and search bar completion provides it)
 *  - code navigation (directory, files, also "hypertext" go to def/uses)
 *  - find uses (funcs, classes, TODO tricky for methods in dynamic languages)
 *  - code tooltip, hover on use of an entity to display information about
 *    it (#uses, TODO: type/args, comments, code, age, methods, etc)
 *  - unified search (files, entities, TODO but also content)
 *  - debugger? it helps understand code so a coverage layer or TODO live
 *    coverage tracing would be nice (as in tracegl)
 *  - source control? extract age, number of authors, churn information in
 *    layers
 *
 * features of IDE we care less about:
 *  - folding/outline? thumbnails make this less important
 *  - auto completion? no. One nice thing of autocomplete though is that
 *    it proposes all the possible methods of an object, the overriden
 *    as well as not overriden parent methods.  We don't want autocomplete
 *    but we want the ability to understand a class by TODO "inlining" parent
 *    methods that are relevant to understand the local code of the class
 *    (e.g. the short command of Eiffel)
 *  - refactoring? no
 *  - UI designer? no
 *  - deploy assistant, cloud assistant? no
 *)

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

⟨*main flags* 5⟩

```
(* see filters below, which files we are interested in *)
let filter = ref (fun _file -> true)
let skip_file  = ref (None: Common.filename option)
```

```
(* less: a config file: GtkMain.Rc.add_default_file "/.../pfff_browser.rc"; *)

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

(*****************************************************************************)
(* Shortcuts *)
(*****************************************************************************)

let filters = [
  (* pad specific, ocaml related *)
  "pfff", (fun file ->
    match FT.file_type_of_file file with
    | FT.PL (
       (FT.ML _) | FT.Makefile | FT.Opa | FT.Prolog _ | FT.Web (FT.Php _)) ->
        (* todo: should be done in file_type_of_file *)
        not (FT.is_syncweb_obj_file file)
        && not (
                (* file =~ ".*commons/" || *)
                (* file =~ ".*external/" || *)
                file =~ ".*_build/")
    | _ -> false
  );
  "ocaml", (fun file ->
    match File_type.file_type_of_file file with
    | FT.PL (FT.ML _) | FT.PL (FT.Makefile)  ->
      (* todo: should be done in file_type_of_file *)
      not (File_type.is_syncweb_obj_file file)
    | _ -> false
  );
  "mli", (fun file ->
    match File_type.file_type_of_file file with
    | FT.PL (FT.ML "mli") | FT.PL (FT.Makefile)   ->
    (* todo: should be done in file_type_of_file *)
      not (File_type.is_syncweb_obj_file file) &&
      not (file =~ ".*/commons/")
    | _ -> false
  );
  "nw", (fun file ->
    match FT.file_type_of_file file with
    | FT.Text "nw" -> true | _ -> false
  );

  (* other languages *)
  "php", (fun file ->
    match File_type.file_type_of_file file with
```

```
    | FT.PL (FT.Web (FT.Php _)) -> true  | _ -> false
  );
  "js", (fun file ->
    match File_type.file_type_of_file file with
    | FT.PL (FT.Web (FT.Js)) -> true  | _ -> false
  );

  "cpp", (let x = ref false in (fun file ->
    Common2.once x (fun () -> Parse_cpp.init_defs !Flag_parsing_cpp.macros_h);
    match FT.file_type_of_file file with
    | FT.PL (FT.C _ | FT.Cplusplus _) -> true
    | FT.PL FT.Asm -> true
    | _ -> false
  ));

  (* exotic languages *)
  "opa", (fun file ->
    match FT.file_type_of_file file with
    | FT.PL (FT.Opa) (* | FT.PL (FT.ML _) *)  -> true
(*    | FT.PL (FT.Web (_)) -> true *)
    | _ -> false
  );
]

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

let set_gc () =
  if !Flag.debug_gc
  then Gc.set { (Gc.get()) with Gc.verbose = 0x01F };
  (* only relevant in bytecode, in native the stacklimit is the os stacklimit *)
  Gc.set {(Gc.get ()) with Gc.stack_limit = 1000 * 1024 * 1024};
  (* see www.elehack.net/michael/blog/2010/06/ocaml-memory-tuning *)
  Gc.set { (Gc.get()) with Gc.minor_heap_size = 4_000_000 };
  Gc.set { (Gc.get()) with Gc.major_heap_increment = 8_000_000 };
  Gc.set { (Gc.get()) with Gc.space_overhead = 300 };
  ()

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

⟨*treemap_generator* 9b⟩

⟨*build_model* 9a⟩

```
(* could also to parse all json files and filter the one which do not parse *)
let layers_in_dir dir =
  Common2.readdir_to_file_list dir +> Common.map_filter (fun file ->
    if file =~ "layer.*json"
    then Some (Filename.concat dir file)
    else None
  )

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

⟨*main_action()* 6⟩

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

(* related work: http://cloc.sourceforge.net/ but have skip list
 * and archi_code_lexer.mll which lower the important of some files?
 *)
let test_loc root =
  let root = Common.realpath root in
  let skip_file = !skip_file ||| Filename.concat root "skip_list.txt" in
  let skip_list =
    if Sys.file_exists skip_file
    then begin
      pr2 (spf "Using skip file: %s" skip_file);
      Skip_code.load skip_file
    end
    else []
  in
  let filter_files_skip_list = Skip_code.filter_files skip_list root in
  let filter_file = (fun file ->
    !filter file && (skip_list = [] || filter_files_skip_list [file] <> []))
  in
  let treemap = Treemap_pl.code_treemap ~filter_file [root] in

  let res = ref [] in
  let rec aux tree =
    match tree with
    | Common2.Node (_dir, xs) ->
        List.iter aux xs
    | Common2.Leaf (leaf, _) ->
        let file = leaf.Treemap.label in
```

```
        let size = leaf.Treemap.size in
        let unix_size = (Common2.unix_stat_eff file).Unix.st_size in
        if unix_size > 0
        then begin
          let multiplier = (float_of_int size /. float_of_int unix_size) in
          let multiplier = min multiplier 1.0 in
          let loc = Common2.nblines_with_wc file in
          Common.push ((Common.readable ~root file),
                       (float_of_int loc *. multiplier)) res;
        end
  in
  aux treemap;
  let total = !res +> List.map snd +> List.map int_of_float  +> Common2.sum in
  pr2 (spf "LOC = %d (%d files)" total (List.length !res));
  let topx = 30 in
  pr2 (spf "Top %d:" topx);
  !res +> Common.sort_by_val_highfirst +> Common.take_safe topx
  +>  List.iter (fun (file, f) ->
      pr2 (spf "%-40s: %d" file (int_of_float f))
  )


let test_treemap_dirs () =
  let paths =
    ["commons/common.ml"; "h_visualization"; "code_graph"]
    +> List.map Common.realpath in
  let paths = List.sort String.compare paths in
  let tree =
    paths +> Treemap.tree_of_dirs_or_files
      ~filter_dir:Lib_vcs.filter_vcs_dir
      ~filter_file:(fun file -> file =~ ".*\\.ml")
      ~file_hook:(fun _file -> 10)
  in
  pr2_gen tree


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

⟨*visual_commitid() action* 82a⟩

```
let width = 500
let height = 500
```

```
let test_draw cr =
  (* [0,0][1,1] world scaled to a width x height screen *)
  Cairo.scale cr (float_of_int width) (float_of_int height);

  Cairo.set_source_rgba cr ~red:0.5 ~green:0.5 ~blue:0.5 ~alpha:0.5;
  Cairo.set_line_width cr 0.001;

  Cairo.move_to cr 0.5 0.5;
  Cairo.line_to cr 0.6 0.6;
  Cairo.stroke cr;

  Cairo.select_font_face cr "serif"
    Cairo.FONT_SLANT_NORMAL Cairo.FONT_WEIGHT_BOLD;
  Cairo.set_font_size cr 0.1;
  Cairo.move_to cr 0.1 0.1;
  Cairo.show_text cr "THIS IS SOME TEXT";
  Cairo.move_to cr 0.1 0.2;
  Cairo.show_text cr "THIS IS SOME TEXT";
  Cairo.set_font_size cr 0.05;
  Cairo.move_to cr 0.1 0.3;
  Cairo.show_text cr "THIS IS SOME TEXT";

  Cairo.set_source_rgb cr ~red:0.1 ~green:0.1 ~blue:0.1;
  Cairo.move_to cr 0.1 0.1;
  Cairo.line_to cr 0.1 0.2;
  Cairo.stroke cr;

  let start = ref 0.0 in

  for _i = 0 to 3 do
    let end_ = !start +. 0.5 in
    Cairo.arc cr ~xc:0.5 ~yc:0.5 ~radius:0.3 ~angle1:!start
      ~angle2:end_;
    Cairo.stroke cr;
    start := end_;
  done;

  ()

let test_cairo () =
  let _locale = GtkMain.Main.init () in
  let w = GWindow.window ~title:"test" () in
  (w#connect#destroy GMain.quit) +> ignore;
  let px = GDraw.pixmap ~width ~height ~window:w () in
  px#set_foreground `WHITE;
  px#rectangle ~x:0 ~y:0 ~width ~height ~filled:true ();
```

```
  let cr = Cairo_lablgtk.create px#pixmap in
  test_draw cr;
  (GMisc.pixmap px ~packing:w#add ()) +> ignore;
  w#show ();
  GMain.main()

(*---------------------------------------------------------------------------*)
(* the command line flags *)
(*---------------------------------------------------------------------------*)
let extra_actions () = [
 ⟨actions 82b⟩
]

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

let all_actions () =
 extra_actions()@
 []

let options () = [
  ⟨options 77⟩
  ] @
  Common.options_of_actions action (all_actions()) @
  Common2.cmdline_flags_devel () @
  [
  "-version",   Arg.Unit (fun () ->
    pr2 (spf "CodeMap version: %s" Config_pfff.version);
    exit 0;
  ),
    " guess what";
  ]

(*****************************************************************************)
(* The main entry point *)
(*****************************************************************************)
let main () =

  let usage_msg =
    spf "Usage: %s [options] <file or dir> \nDoc: %s\nOptions:"
      (Filename.basename Sys.argv.(0))
      "https://github.com/facebook/pfff/wiki/Codemap"
  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 _ =
  Common.main_boilerplate (fun () ->
    main ()
  )
```

C.2 flag_visual.ml

C.3 model_graph_code.mli

97 ⟨*model_graph_code.mli* 97⟩≡

```
val build_filedeps_of_dir_or_file:
  Graph_code.graph ->
  (Graph_code.node, Common.filename list * Common.filename list) Hashtbl.t

(* the nodes are sorted by line numbers *)
val build_entities_of_file:
  Graph_code.graph ->
  (Common.filename, Graph_code.node list) Common.assoc
```

```
val add_headers_files_entities_of_file:
  Common.dirname ->
  (Common.filename, Graph_code.node list) Common.assoc ->
  (Common.filename, Graph_code.node list) Common.assoc

val node_of_entity:
  Database_code.entity -> Graph_code.graph -> Graph_code.node option
```

C.4 model_database_code.mli

98a ⟨*model_database_code.mli* 98a⟩≡

⟨*hentities sig* 12b⟩

⟨*hfiles_and_top_entities sig* 40b⟩

⟨*all_entities sig* 51c⟩

⟨*actual_root_of_db sig* 11c⟩

⟨*readable_to_absolute_filename_under_root sig* 11b⟩

C.5 model_graph_code.ml

98b ⟨*model_graph_code.ml* 98b⟩≡

```
⟨Facebook copyright 2⟩
open Common

module G = Graph_code
module E = Database_code

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

(*****************************************************************************)
(* Helpers *)
(*****************************************************************************)
let is_prefix prefix str =
  (* todo: have better than that? *)
  try
    String.sub str 0 (String.length prefix) =$= prefix
  with Invalid_argument _ -> false
```

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

let build_filedeps_of_dir_or_file g =
  (* we use the 'find_all' property of those hashes *)
  let huses = Hashtbl.create 101 in
  let husers = Hashtbl.create 101 in

  let halready = Hashtbl.create 101 in

  g +> G.iter_use_edges (fun n1 n2 ->
    try
      let file1 = G.file_of_node n1 g in
      let file2 = G.file_of_node n2 g in
      (* file to file deps *)
      if file1 <> file2 && not (Hashtbl.mem halready (file1, file2)) then begin
        Hashtbl.replace halready (file1, file2) true;
        Hashtbl.add huses (file1, E.File) file2;
        Hashtbl.add husers (file2, E.File) file1;
      end;
      (* dir to file deps *)
      (* e.g. if a/b/foo.c -> a/c/bar.c then need to add
       * a/b -> a/c/bar.c, but not a/ -> a/c/bar.c cos of is_prefix
       * a/c <- a/b/foo.c, but not a/ <- a/b/foo.c cos of is_prefix
       *)
      let dirs_n1 = Common2.inits_of_relative_dir file1 in
      let dirs_n2 = Common2.inits_of_relative_dir file2 in
      dirs_n1 +> List.iter (fun dir ->
        if not (is_prefix dir file2)
        then Hashtbl.add huses (dir, E.Dir) file2;
      );
      dirs_n2 +> List.iter (fun dir ->
        if not (is_prefix dir file1)
        then Hashtbl.add husers (dir, E.Dir) file1;
      );

    with Not_found -> ()
  );
  let hres = Hashtbl.create 101 in
  let keys = Common2.union_set (Common2.hkeys huses) (Common2.hkeys husers) in
  keys +> List.iter (fun k ->
    let uses = try Hashtbl.find_all huses k with Not_found -> [] in
    let users = try Hashtbl.find_all husers k with Not_found -> [] in
    (* todo: have to do uniq? if add in hash multiple times with same value,
```

```
     * then get multiple bindings?
     *)
    Hashtbl.add hres k (uses, users)
  );
  hres

let build_entities_of_file g =

  (* we use the 'find_all' property of those hashes *)
  let h = Hashtbl.create 101 in

  g +> G.iter_nodes (fun n ->
    try
      let info = G.nodeinfo n g in
      let file = info.G.pos.Parse_info.file in
      (* old: let line = info.G.pos.Parse_info.line in *)
      Hashtbl.add h file n;
    with Not_found -> ()
  );
  Common2.hkeys h +> List.map (fun k ->
    let xs = Hashtbl.find_all h k in
    k, xs
  )

(* Codegraph does not currently handle very well header files. It's because
 * an header contain entities that are considered DUPE of their
 * corresponding entity in the source file. Right now we skip such
 * headers (e.g. .mli) in codegraph. The code below is here to
 * adjust codegraph deficiencies by artificially associate the
 * nodes in the source file to also the header file so one can
 * hover a function signature in a .mli and get its uses, users, etc.
 *
 * ugly: fix that in codegraph instead?
 *)
let add_headers_files_entities_of_file root xs =
  let headers =
    xs +> Common.map_filter (fun (file, xs) ->
      let (d,b,e) = Common2.dbe_of_filename_noext_ok file in
      match e with
      | "ml" ->
        let header_readable = Common2.filename_of_dbe (d,b,"mli") in
        let header = Filename.concat root header_readable in
        if Sys.file_exists header
        (* todo: we add too many defs here, a mli can actually restrict
         * the set of exported functions, but because we use such
         * information mostly when hovering over entities in a .mli,
```

```
          * this should be fine.
          *)
         then Some (header_readable, xs)
         else None
       | _ -> None
    )
  in
  headers @ xs

let node_of_entity e g =
  let fullname =
    match e.E.e_fullname with
    | "" -> e.E.e_name
    | s -> s
  in
  let node = (fullname, e.E.e_kind) in
  if G.has_node node g
  then Some node
  else None
```

C.6 model_database_code.ml

101 ⟨*model_database_code.ml* 101⟩≡

```
⟨Facebook copyright 2⟩
open Common

module Flag = Flag_visual
module Db = Database_code

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

⟨readable_to_absolute_filename_under_root 11d⟩

⟨actual_root_of_db 12a⟩

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

⟨hentities() 13a⟩

⟨hfiles_and_top_entities() 40c⟩
```

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

⟨*all_entities* 52d⟩

C.7 model2.mli

102 ⟨*model2.mli* 102⟩≡

```
⟨type model 8⟩
and 'a deps = 'a list (* uses *) * 'a list (* users *)


type macrolevel = Treemap.treemap_rendering



type microlevel = {
  point_to_line: Cairo.point -> line;
  line_to_rectangle: line -> Figures.rectangle;
  layout: layout;
  container: Treemap.treemap_rectangle;
  (* the lines of the files, 0-based indexed line, see line type below *)
  content: (glyph list) array option;
  (* defs based on highlighters categories *)
  defs: (line * short_node) list;
}
  (* 0-indexed line number, which is different from most tools, but
   * programs prefer 0-based index
   *)
  and line = Line of int

  and layout = {
    lfont_size: float;
    split_nb_columns: float; (* int *)
    width_per_column:float;
    height_per_line: float;
    nblines: float; (* int *)
    nblines_per_column: float; (* int *)
  }

  and glyph = {
```

```
    str: string;
    categ: Highlight_code.category option;
    font_size: float;
    color: Simple_color.emacs_color;
    mutable pos: Cairo.point;
  }

 (* Note that I don't use G.node because the string below is not fully
  * qualified so one must use match_short_vs_node when comparing with nodes.
  *)
  and short_node = (string * Database_code.entity_kind)
```

⟨*type drawing* 10⟩

```
type world = {
  mutable dw: drawing;
  dw_stack: drawing Common.stack ref;

  (* computed lazily, semantic information about the code *)
  model: model Async.t;
  root_orig: Common.dirname;
  (* to compute a new treemap based on user's action *)
  treemap_func: Common.path list -> Treemap.treemap_rendering;
  (* misc settings, not really used for now *)
  settings: settings;

  mutable current_node: Graph_code.node option;
  mutable current_node_selected: Graph_code.node option;
  mutable current_entity: Database_code.entity option;
}
   and settings = {
     mutable draw_summary: bool;
     mutable draw_searched_rectangles: bool;
   }
```

⟨*type context* 11a⟩

```
val context_of_drawing: drawing -> model Async.t -> context
```

⟨*init_drawing sig* 13b⟩

⟨*new_pixmap sig* 83a⟩

```
(* point -> rectangle -> line -> glyph -> entity *)
```

⟨*find_rectangle_at_user_point sig* 46a⟩

```
val find_line_in_rectangle_at_user_point:
  Cairo.point -> Treemap.treemap_rectangle -> drawing -> line option
val find_glyph_in_rectangle_at_user_point:
  Cairo.point -> Treemap.treemap_rectangle -> drawing -> glyph option

(* graph code integration *)

val find_def_entity_at_line_opt:
  line -> Treemap.treemap_rectangle -> drawing -> model ->
  Graph_code.node option
val find_use_entity_at_line_and_glyph_opt:
  line -> glyph -> Treemap.treemap_rectangle -> drawing -> model ->
  Graph_code.node option

(* macrolevel deps *)
val node_of_rect:
  Treemap.treemap_rectangle -> model -> Graph_code.node

val deps_readable_files_of_node:
  Graph_code.node -> model ->
  Common.filename (* readable *) deps

val deps_rects_of_rect:
  Treemap.treemap_rectangle -> drawing -> model ->
  Treemap.treemap_rectangle deps

(* microlevel deps *)
val deps_nodes_of_node_clipped:
  Graph_code.node -> drawing -> model ->
  (Graph_code.node * line * microlevel) deps

(* line highlight *)
val line_and_microlevel_of_node_opt:
  Graph_code.node -> drawing -> model ->
  (Graph_code.node * line * microlevel) option

val lines_where_used_node:
  Graph_code.node -> line -> microlevel -> line list
```

C.8 model2.ml

105 ⟨*model2.ml* 105⟩≡
⟨*Facebook copyright* 2⟩

```
open Common

module Flag = Flag_visual
module CairoH = Cairo_helpers
module F = Figures
module T = Treemap
module E = Database_code

(*****************************************************************************)
(* The code model *)
(*****************************************************************************)

⟨type model 8⟩
and 'a deps = 'a list (* uses *) * 'a list (* users *)

(*****************************************************************************)
(* The drawing model *)
(*****************************************************************************)

type macrolevel = Treemap.treemap_rendering

(*
 * We use different sources to provide fine-grained semantic visualization:
 * - the source code itself, lexed and parsed in parsing2.ml with language
 *   specific parsers, with the ASTs and tokens stored in a global cache
 * - a language agnostic 'glyph list array' computed from the AST and tokens
 *   by the language specific highlighter
 * - a language agnostic fuzzy defs identification based on the category of
 *   the glyphs (but containing only "short nodes")
 * - the graph code computed for the whole project, usually not up to
 *   date with the most recent modifications, but containing useful
 *   global information such as the precise set of uses and users of an entity
 * - the light database (but could be replaced by the graph code)
 *
 * We try to match specific glyphs to the right entity, then use
 * the graph code to find users (and uses) of this entity, and then going
 * from those entities to their corresponding glyph in this file or another
 * file in the whole treemap.
 *)

type microlevel = {
  point_to_line: Cairo.point -> line;
  line_to_rectangle: line -> Figures.rectangle;
  layout: layout;
  container: Treemap.treemap_rectangle;
  content: (glyph list) array option;
```

```
  (* sorted list of entities by line, defs based on highlighter *)
  defs: (line * short_node) list;
}
 (* 0-indexed line number, which is different from most tools, but
  * programs prefer 0-based index
  *)
  and line = Line of int
 (* Note that I don't use G.node because the string below is not fully
  * qualified so one must use match_short_vs_node when comparing with nodes.
  *)
  and short_node = (string * Database_code.entity_kind)
  and glyph = {
    str: string;
    categ: Highlight_code.category option;
    font_size: float;
    color: Simple_color.emacs_color;
    (* the lower left position, before calling Cairo.show_text str *)
    mutable pos: Cairo.point;
  }
  and layout = {
    lfont_size: float;
    split_nb_columns: float; (* int *)
    width_per_column:float;
    height_per_line: float;
    nblines: float; (* int *)
    nblines_per_column: float; (* int *)
  }
```

⟨*type drawing* 10⟩

```
(*****************************************************************************)
(* The world *)
(*****************************************************************************)
type world = {
  mutable dw: drawing;
  dw_stack: drawing stack ref;

  (* computed lazily, semantic information about the code *)
  model: model Async.t;

  root_orig: Common.dirname;

  (* to compute a new treemap based on user's action *)
  treemap_func: Common.path list -> Treemap.treemap_rendering;
```

```
  (* misc settings, not really used for now *)
  settings: settings;

  mutable current_node: Graph_code.node option;
  mutable current_node_selected: Graph_code.node option;
  mutable current_entity: Database_code.entity option;
}
   and settings = {
     mutable draw_summary: bool;
     mutable draw_searched_rectangles: bool;
   }

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

⟨*new_pixmap()* 83b⟩

⟨*init_drawing()* 13c⟩

```
(*****************************************************************************)
(* The drawing context *)
(*****************************************************************************)
```

⟨*type context* 11a⟩

```
let context_of_drawing dw model = {
  nb_rects_on_screen = dw.nb_rects;
  model2 = model;
  grep_query = dw.current_grep_query;
  layers_microlevel = dw.layers.Layer_code.micro_index;
}

(*****************************************************************************)
(* Point -> (rectangle, line, glyph, entity) *)
(*****************************************************************************)
```

⟨*find_rectangle_at_user_point()* 46b⟩

```
let find_line_in_rectangle_at_user_point user r dw =
  try
    let microlevel = Hashtbl.find dw.microlevel r in
    let line = microlevel.point_to_line user in
    Some line
  with Not_found -> None
```

```
let find_glyph_in_rectangle_at_user_point user r dw =
  find_line_in_rectangle_at_user_point user r dw >>= (fun line ->
    let microlevel = Hashtbl.find dw.microlevel r in
    microlevel.content >>= (fun glyphs ->
      let (Line line) = line in
      if line >= Array.length glyphs
      then None
      else
        let glyphs = glyphs.(line) in
        (* find the best one *)
        glyphs +> List.rev +> Common.find_opt (fun glyph ->
          let pos = glyph.pos in
          user.Cairo.x >= pos.Cairo.x
        )
    )
  )

(*****************************************************************************)
(* Graph code integration *)
(*****************************************************************************)

let match_short_vs_node (str, short_kind) node =
  Graph_code.shortname_of_node node =$= str &&
  Database_code.matching_def_short_kind_kind short_kind (snd node)

(* when in a file we have both the prototype (forward decl) and
 * the def, we prefer the def.
 *)
let rank_entity_kind = function
  | E.Function | E.Global -> 3
  | E.Prototype | E.GlobalExtern -> 1
  | _ -> 2

(* We used to just look in hentities_of_file for the line mentioned
 * in the graph_code database, but the file may have changed so better
 * instead to rely on microlevel.defs.
 *)
let find_def_entity_at_line_opt line tr dw model =
  let file = tr.T.tr_label in
  let readable = Common.readable ~root:model.root file in
  try
    let nodes = Hashtbl.find model.hentities_of_file readable in
    let microlevel = Hashtbl.find dw.microlevel tr in
    let short_node = List.assoc line microlevel.defs in
    (* try to match the possible shortname str with a fully qualified node
```

```
    *)
    nodes +> Common.map_filter (fun node ->
      if match_short_vs_node short_node node
      then Some node
      else None
    ) +> List.map (fun (s, kind) -> ((s, kind), rank_entity_kind kind))
      +> Common.sort_by_val_highfirst
      +> List.hd +> fst +> (fun x -> Some x)
  with Not_found | Failure "hd" -> None

let find_use_entity_at_line_and_glyph_opt line glyph tr dw model =
  model.g >>= (fun g ->
    (* find enclosing def line *)
    let microlevel = Hashtbl.find dw.microlevel tr in
    (* try because maybe have no enclosing defs *)
    try
      let (line_def, _shortnode) =
        microlevel.defs +> List.rev +> List.find (fun (line2, _shortnode) ->
          line >= line2
        )
      in
      find_def_entity_at_line_opt line_def tr dw model >>= (fun node ->
        let uses = Graph_code.succ node Graph_code.Use g in
        uses +> Common.find_opt (fun node ->
          let s = Graph_code.shortname_of_node node in
          let categ =  glyph.categ ||| Highlight_code.Normal in
          glyph.str =$= s &&
          Database_code.matching_use_categ_kind categ (snd node)
        )
      )
    with Not_found -> None
  )

let node_of_rect tr model =
  let file = tr.Treemap.tr_label in
  let readable = Common.readable ~root:model.root file in
  let kind = if tr.Treemap.tr_is_node then E.Dir else E.File in
  readable, kind


let deps_readable_files_of_node node model =
  match node, model.g with
  | (_, (E.Dir | E.File)), _ ->
      (* opti: can't use g for that *)
      (try Hashtbl.find model.hfile_deps_of_node node with Not_found -> [], [])
  | _, None -> [], []
```

```
  | _, Some g ->
      let succ = Graph_code.succ node Graph_code.Use g in
      let pred = Graph_code.pred node Graph_code.Use g in
      succ +> Common.map_filter (fun n ->
        try Some (Graph_code.file_of_node n g) with Not_found -> None
      ),
      pred +> Common.map_filter (fun n ->
        try Some (Graph_code.file_of_node n g) with Not_found -> None
      )

let deps_rects_of_rect tr dw model =
  let node = node_of_rect tr model in
  let uses, users = deps_readable_files_of_node node model in
  uses +> Common.map_filter (fun file ->
    Common2.optionise (fun () -> Hashtbl.find dw.readable_file_to_rect file)
  ),
  users +> Common.map_filter (fun file ->
    Common2.optionise (fun () ->Hashtbl.find dw.readable_file_to_rect file)
  )

let line_and_microlevel_of_node_opt n dw model =
  model.g >>= (fun g ->
    try
      let file = Graph_code.file_of_node n g in
      (* rectangles not on the screen will be automatically "clipped"
       * as this may raise Not_found
       *)
      let rect = Hashtbl.find dw.readable_file_to_rect file in
      let microlevel = Hashtbl.find dw.microlevel rect in
      let line = microlevel.defs +> List.find (fun (_line, snode) ->
        match_short_vs_node snode n
      ) +> fst in
      Some (n, line, microlevel)
    with Not_found -> None
  )

let uses_or_users_of_node node dw fsucc model =
  match model.g with
  | None -> []
  | Some g ->
    let succ = fsucc node Graph_code.Use g in
    succ +> Common.map_filter (fun n ->
      line_and_microlevel_of_node_opt n dw model
    )

let deps_nodes_of_node_clipped node dw model =
```

```
  uses_or_users_of_node node dw Graph_code.succ model,
  uses_or_users_of_node node dw Graph_code.pred model


let lines_where_used_node node startl microlevel =
  let s = Graph_code.shortname_of_node node in
  let s =
    (* ugly: see Graph_code_clang.new_str_if_defs() where we rename dupes *)
    match s with
    | _ when (s =~ "\\(.*\\)__[0-9]+$") -> Common.matched1 s
    | _ when (s =~ "^\\$\\(.*\\)") -> Common.matched1 s
    | _ -> s
  in

  let (Line startl) = startl in
  match microlevel.content with
  | None -> []
  | Some glypys ->
    let res = ref [] in
    (* todo: should be from startl to endl (the start of the next entity) *)
    for line = startl to Array.length glypys - 1 do
      let xs = glypys.(line) in
      if xs +> List.exists (fun glyph ->
        let categ =  glyph.categ ||| Highlight_code.Normal in
        glyph.str =$= s &&
        Database_code.matching_use_categ_kind categ (snd node)
      )
      then Common.push (Line line) res
    done;
    !res
```

C.9 view2.mli

111 ⟨*view2.mli* 111⟩≡
⟨*mk_gui sig* 14⟩

C.10 view2.ml

112 ⟨*view2.ml* 112⟩≡
⟨*Facebook copyright* 2⟩

```
open Common2
open Common

module G = Gui
```

```
module K = GdkKeysyms
module GR = Gdk.Rectangle
module F = Figures
module T = Treemap
module CairoH = Cairo_helpers
open Model2 (* for the fields *)
module M = Model2
module Controller = Controller2
module Flag = Flag_visual
module Style = Style2
module Db = Database_code

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

(*****************************************************************************)
(* Wrappers *)
(*****************************************************************************)
let pr2, _pr2_once = Common2.mk_pr2_wrappers Flag.verbose_visual

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

⟨*view globals* 15a⟩

```
(*****************************************************************************)
(* Final view rendering *)
(*****************************************************************************)

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

⟨*assemble_layers* 65b⟩

⟨*expose* 66a⟩

⟨*configure* 65a⟩

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

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

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

⟨*mk_gui()* 15b⟩

C.11 controller2.mli

113 ⟨*controller2.mli* 113⟩≡

```
val _refresh_da: (unit -> unit) ref
val _refresh_legend: (unit -> unit) ref

val _go_back: (Model2.world -> unit) ref
val _go_dirs_or_file:
  (?current_grep_query: (string, Model2.line) Hashtbl.t option ->
    Model2.world -> Common.path list -> unit
  ) ref

val _statusbar_addtext: (string -> unit) ref
val _set_title: (string -> unit) ref

val current_rects_to_draw:
  (Treemap.treemap_rectangle list) ref
val current_r:
  Treemap.treemap_rectangle option ref
val hook_finish_paint: (unit -> unit) ref

val paint_content_maybe_refresher:
  GMain.Idle.id option ref
val current_motion_refresher:
  GMain.Idle.id option ref
val current_tooltip_refresher:
  GMain.Timeout.id option ref

val title_of_path: string -> string
```

C.12 controller2.ml

114 ⟨*controller2.ml* 114⟩≡
⟨*Facebook copyright* 2⟩

```
(* refresh drawing area *)
let _refresh_da = ref (fun () ->
  failwith "_refresh_da not defined"
)
let _refresh_legend = ref (fun () ->
  failwith "_refresh_legend not defined"
)

let current_rects_to_draw = ref []
let hook_finish_paint = ref (fun () ->
  ()
)

let current_r = ref None

let paint_content_maybe_refresher = ref None
let current_motion_refresher = ref None
let current_tooltip_refresher = ref None

let _go_back = ref (fun _w ->
  failwith "_go_back not defined"
)

let _go_dirs_or_file = ref
 (fun ?(current_grep_query=None)  _dw_ref _paths ->
   ignore current_grep_query;
  failwith "_go_dirs_or_file not defined"
)

let _statusbar_addtext = ref (fun _s ->
  failwith "_statusbar_addtext not defined"
)
let _set_title = ref (fun _s ->
  failwith "_set_title not defined"
)

let title_of_path s = "CodeMap: " ^ s
```

C.13 help.mli

115a ⟨*help.mli* 115a⟩≡

```
val interface_doc: string
```

C.14 help.ml

115b ⟨*help.ml* 115b⟩≡

⟨*interface_doc* 3⟩

C.15 draw_macrolevel.mli

115c ⟨*draw_macrolevel.mli* 115c⟩≡

⟨*draw_treemap_rectangle sig* 25a⟩

```
val draw_trect_using_layers:
  cr:Cairo.t ->
  Layer_code.layers_with_index ->
  Treemap.treemap_rectangle ->
  unit
```

C.16 draw_macrolevel.ml

115d ⟨*draw_macrolevel.ml* 115d⟩≡

⟨*Facebook copyright* 2⟩

```
open Common
open Common2.ArithFloatInfix

open Figures (* for the fields *)
module T = Treemap
module Color = Simple_color

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

(*****************************************************************************)
(* Drawing a treemap rectangle *)
(*****************************************************************************)
```

⟨*draw_treemap_rectangle()* 25b⟩

```
(*****************************************************************************)
(* Layers macrolevel *)
(*****************************************************************************)

(* How should we draw layer information at the macro level ?
 *
 *  - fill the rectangle with the color of one layer ?
 *  - separate equally among layers ?
 *  - draw on top of the existing archi color ?
 *  - draw circles instead of rectangle so have quantitative information too
 *    (like I was doing when display git related commit information).
 *
 * It is maybe good to not draw on top of the existing archi_code color.
 * Too many colors kill colors. Also we can not convey quantitative
 * information by coloring with full rectangles (instead of the random
 * circles trick) but for some layers like security it is probably better.
 * Don't care so much about how many bad calls; care really about
 * number of files with bad calls in them.
 *
 * So for now we just fill rectangles with colors from the layer and
 * when a file matches multiple layers we split the rectangle in equal
 * parts.
 *)

let draw_trect_using_layers ~cr layers_with_index rect =
  (* don't use archi_code color. Just black and white *)
  let is_file = not rect.T.tr_is_node in
  let color = if is_file then "white" else "black" in
  draw_treemap_rectangle ~cr ~color:(Some color) rect;

  if is_file then begin
    let file = rect.T.tr_label in

    let color_info =
      try Hashtbl.find layers_with_index.Layer_code.macro_index file
      with Not_found -> []
    in
    (* What to draw? TODO a splitted rectangle? *)
    let sorted = Common2.sort_by_key_highfirst color_info in
    (match sorted with
    | [] -> ()
    | (_float, color)::_rest ->
      draw_treemap_rectangle ~cr ~color:(Some color) rect;
    );
  end
```

⟨*draw_summary_content* 40a⟩

C.17 draw_microlevel.mli

117a ⟨*draw_microlevel.mli* 117a⟩≡

⟨*draw_treemap_rectangle_content_maybe sig* 32b⟩

⟨*text_with_user_pos sig* 42c⟩

```
val draw_magnify_line:
  ?honor_color:bool ->
  Cairo.t -> Model2.line -> Model2.microlevel -> unit
```

C.18 draw_microlevel.ml

117b ⟨*draw_microlevel.ml* 117b⟩≡

```
⟨Facebook copyright 2⟩
open Common
open Common2.ArithFloatInfix

open Figures (* for the fields *)
open Model2 (* for the fields *)
module F = Figures
module M = Model2
module T = Treemap
module Color = Simple_color
module CairoH = Cairo_helpers
module HC = Highlight_code
module Flag = Flag_visual
module Style = Style2
module FT = File_type
module Parsing = Parsing2

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

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

(* There are many different coordinates relevant to the lines of a file:
 *  - line number in the file
```

```
 *  - column and line in column for the file when rendered in multiple columns
 *  - x,y position relative to the current treemap rectangle
 *  - x,y position on the screen in normalized coordinates
 *  - x,y position on the screen in pixels
 * We have many functions below to go from one to the other.
 *
 * note: some types below could be 'int' but it's more convenient to have
 * everything as a float because arithmetic with OCaml sucks when have
 * multiple numeric types.
 *
 * Below line numbers starts at 0, not at 1 as in emacs.
 *)

type _line = Model2.line

type line_in_column = {
  column: float; (* int *)
  line_in_column: float; (* int *)
}

type _pos = float (* x *) * float (* y *)

type _point = Cairo.point
```

⟨*type draw_content_layout* 32a⟩

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

let is_big_file_with_few_lines ~nblines file =
  nblines < 20. && Common2.filesize_eff file > 4000

(* coupling: with parsing2.ml, todo move in parsing2.ml? *)
let use_fancy_highlighting file =
  match FT.file_type_of_file file with
  | ( FT.PL (FT.Web (FT.Php _))
    | FT.PL (FT.Web (FT.Js))
    | FT.PL (FT.Web (FT.Html))
    | FT.PL (FT.ML _)
    | FT.PL (FT.Cplusplus _ | FT.C _ | FT.ObjectiveC _)
    | FT.PL (FT.Thrift)
    | FT.Text ("nw" | "tex"  | "texi" | "web" | "org")
    | FT.PL (FT.Lisp _)
    | FT.PL (FT.Haskell _)
    | FT.PL (FT.Python)
```

```
    | FT.PL (FT.Csharp)
    | FT.PL (FT.Java)
    (*    | FT.PL (FT.Prolog _) *)
    | FT.PL (FT.Erlang)
    | FT.PL (FT.Opa)
    ) -> true
  | (FT.Text "txt") when Common2.basename file =$= "info.txt" -> true
  | _ -> false

(*****************************************************************************)
(* Coordinate conversion *)
(*****************************************************************************)

let line_in_column_to_bottom_pos lc r layout =
  let x = r.p.x + (lc.column * layout.width_per_column) in
  (* to draw text in cairo we need to be one line below, hence the +1
   * as y goes down but the text is drawn above
   *)
  let y = r.p.y + ((lc.line_in_column + 1.) * layout.height_per_line) in
  x, y

let line_to_line_in_column line layout =
  let (Line line) = line in
  let line = float_of_int line in
  let column = floor (line / layout.nblines_per_column) in
  let line_in_column =
    line - (column * layout.nblines_per_column) in
  { column; line_in_column }

let line_to_rectangle line r layout =
  let lc = line_to_line_in_column line layout in
  (* this is the bottom pos, so we need to substract height_per_line
   * if we want to draw above the bottom pos
   *)
  let x, y = line_in_column_to_bottom_pos lc r layout in
  { p = { x;
          y = y - layout.height_per_line };
    q = { x = x + layout.width_per_column;
          y = y + 0.2 * layout.height_per_line };
  }

let point_to_line pt r layout =
  let x = pt.Cairo.x - r.p.x in
  let y = pt.Cairo.y - r.p.y in
  let line_in_column = floor (y / layout.height_per_line) in
  let column = floor (x / layout.width_per_column) in
```

```
  Line ((column * layout.nblines_per_column + line_in_column) +> int_of_float)

(*****************************************************************************)
(* Content properties *)
(*****************************************************************************)

(* Anamorphic entities *)
```
⟨*final_font_size_of_categ* 33a⟩

```
let color_of_categ categ =
  let attrs =
    match categ with
    | None ->       Highlight_code.info_of_category Highlight_code.Normal
    | Some categ -> Highlight_code.info_of_category categ
  in
  attrs +> Common.find_some (fun attr ->
    match attr with
    | `FOREGROUND s
    | `BACKGROUND s (* todo: should really draw the background of the text *)
      ->
        Some (s)
    | _ -> None
  )

let glyphs_of_file ~font_size ~font_size_real model_async file
  : (glyph list) array option =

  (* real position is set later in draw_content *)
  let pos = { Cairo.x = 0.; y = 0. } in

  match FT.file_type_of_file file with
  | _ when use_fancy_highlighting file ->

    let entities =
      match Async.async_get_opt model_async with
      | Some model -> model.Model2.hentities
      | None -> Hashtbl.create 0
    in

    (* if you have some cache in tokens_with_categ_of_file, then it
     * must be invalidated when a file has changed on the disk, otherwise
     * we can get some Array out of bound exceptions as the number of lines
     * returned by nblines_eff may be different
     *)
    let nblines = Common2.nblines_eff file in
    let arr = Array.create nblines [] in
```

```
    let tokens_with_categ = Parsing.tokens_with_categ_of_file file entities in

    let line = ref 0 in
    let acc = ref [] in
    (try
     tokens_with_categ +> List.iter (fun (s, categ, _filepos) ->
      let final_font_size =
        final_font_size_of_categ ~font_size ~font_size_real categ in
      let color =
        color_of_categ categ in

      let xs = Common2.lines_with_nl_either s in
      xs +> List.iter (function
      | Common2.Left str ->
          Common.push { M. str; font_size=final_font_size; color; categ;pos }
            acc;
      | Common2.Right () ->
          arr.(!line) <- List.rev !acc;
          acc := [];
          incr line;
      )
    );
    if !acc <> []
    then arr.(!line) <- List.rev !acc;
    Some arr
   with Invalid_argument("index out of bounds") ->
     failwith (spf "try on %s, nblines = %d, line = %d" file nblines !line)
    )

  | FT.PL _ | FT.Text _ ->
      Common.cat file
      +> List.map (fun str ->
        [{ M.str; font_size; color = "black"; categ=None; pos }])
      +> Array.of_list
      +> (fun x -> Some x)

  | _ -> None

let defs_of_glyphs glyphs =
  let res = ref [] in
  glyphs +> Array.iteri (fun line_0_indexed glyphs ->
    glyphs +> List.iter (fun glyph ->
      glyph.categ +> Common.do_option (fun categ ->
        Database_code.entity_kind_of_highlight_category_def categ
        +> Common.do_option (fun kind ->
              Common.push (Line line_0_indexed, (glyph.str, kind)) res
```

```
      ))));
  List.rev !res

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

⟨*font_size_when_have_x_columns* 33b⟩

⟨*optimal_nb_columns* 33c⟩

⟨*draw_column_bars* 34a⟩

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

⟨*draw_content* 34b⟩

⟨*draw_treemap_rectangle_content_maybe* 36⟩

```
(*****************************************************************************)
(* Magnifyer Content *)
(*****************************************************************************)

(* alt: digital zoom? good enough? need rendering at better resolution? *)
let draw_magnify_line ?(honor_color=true) cr line microlevel =
  match microlevel.content with
  | None -> ()
  | Some glyphs ->
    let r = microlevel.container.T.tr_rect in
    let layout = microlevel.layout in

    let lc = line_to_line_in_column line layout in
    let x, y = line_in_column_to_bottom_pos lc r layout in
    Cairo.move_to cr x y;

    let (Line iline) = line in
    (* because of the way we layout code in multiple columns with different
     * fonts, we may not use the whole rectangle to draw the content of
     * a file and so the cursor could be far below the last line of
     * the file
     *)
    if iline < Array.length glyphs then begin
```

```
 glyphs.(iline)
 +> (fun xs ->
  match xs with
  | [] -> []
  | x::xs ->
    if x.M.str =~ "[ \t]+" then xs
    else x::xs
)
+> List.iter (fun glyph ->
  (* let font_size = glyph.M.font_size * 3. in *)
  let font_size_real = 15. in
  let font_size = CairoH.device_to_user_size cr font_size_real in
  Cairo.set_font_size cr font_size;
  let color =
    if honor_color
    then glyph.color
    else "wheat"
  in
  let (r,g,b) = Color.rgbf_of_string color in
  let alpha = 1. in
  Cairo.set_source_rgba cr r g b alpha;
  CairoH.show_text cr glyph.M.str;
)
end
```

C.19 draw_labels.mli

123a ⟨*draw_labels.mli* 123a⟩≡

⟨*draw_treemap_rectangle_label_maybe sig* 25c⟩

C.20 draw_label.ml

123b ⟨*draw_labels.ml* 123b⟩≡

```
⟨Facebook copyright 2⟩
open Common
open Common2.ArithFloatInfix

open Figures (* for the fields *)

module Flag = Flag_visual

module T = Treemap
module F = Figures
```

```
module Color = Simple_color

module CairoH = Cairo_helpers

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

⟨*draw_treemap_rectangle_label_maybe* 26⟩

C.21 draw_legend.mli

124a ⟨*draw_legend.mli* 124a⟩≡

```
val draw_legend: cr:Cairo.t -> unit

val draw_legend_layer: cr:Cairo.t -> Layer_code.layers_with_index -> unit
```

C.22 draw_legend.ml

124b ⟨*draw_legend.ml* 124b⟩≡

⟨*Facebook copyright* 2⟩

```
open Common
open Common2.ArithFloatInfix

module CairoH = Cairo_helpers
module L = Layer_code

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

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

let draw_legend_of_color_string_pairs ~cr xs =

  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;
```

```
  xs +> Common.index_list_1 +> List.iter (fun ((color,s), 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 ();
    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;
  )

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

⟨*paint_legend* 31a⟩

```
let draw_legend_layer ~cr layers_idx =
  let pairs =
    layers_idx.L.layers +> Common.map_filter (fun (layer, is_active) ->
      if is_active
      then Some layer.L.kinds
      else None
    ) +> List.flatten +> List.map (fun (a, b) -> (b, a))
  in
  draw_legend_of_color_string_pairs ~cr pairs
```

C.23 view_mainmap.mli

125 ⟨*view_mainmap.mli* 125⟩≡

```
val paint: Model2.drawing -> Model2.model Async.t -> unit

val zoom_pan_scale_map: Cairo.t -> Model2.drawing -> unit

val device_to_user_area: Model2.drawing -> Figures.rectangle

val with_map: Model2.drawing -> (Cairo.t -> 'a) -> 'a

val button_action:
   Model2.world -> GdkEvent.Button.t -> bool
```

C.24 view_mainmap.ml

126 ⟨*view_mainmap.ml* 126⟩≡
⟨*Facebook copyright* 2⟩

```
open Common
(* floats are the norm in graphics *)
open Common2.ArithFloatInfix

open Model2
module CairoH = Cairo_helpers
module F = Figures
module T = Treemap
module Flag = Flag_visual
module M = Model2
module Ctl = Controller2

(*****************************************************************************)
(* Prelude *)
(*****************************************************************************)
(*
 * This module calls Draw_macrolevel and Draw_microlevel and assembles
 * the final "painting" of the code "main map". It is called mainly by
 * View2.configure and Ui_navigation.go_dirs_and_file.
 *
 * Painting is not the last element in the "main map" rendering pipeline.
 * There is also View_overlay which is called mainly when the user
 * moves the mouse which triggers the View_overlay.motion_refresher
 * callback which just add overlays on top of the already drawn (and
 * computationaly expensive) painting done here.
 *)

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

⟨*zoom_pan_scale_map* 49h⟩

⟨*with_map* 49g⟩

⟨*device_to_user_area* 24⟩

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

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

⟨*paint* 38⟩

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

⟨*key_pressed* 42a⟩
⟨*find_filepos_in_rectangle_at_user_point* 42b⟩

⟨*button_action* 42d⟩

C.25 view_minimap.mli

127a ⟨*view_minimap.mli* 127a⟩≡

C.26 view_minimap.ml

127b ⟨*view_minimap.ml* 127b⟩≡
⟨*Facebook copyright* 2⟩

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

⟨*scale_minimap* 49f⟩

⟨*with_minimap* 49d⟩

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

⟨*paint_minimap* 49a⟩

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

⟨*expose_minimap* 49b⟩

⟨*configure_minimap* 49c⟩

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

⟨*motion_notify_minimap* 48d⟩

⟨*button_action_minimap* 48e⟩

C.27 view_overlays.mli

128a ⟨*view_overlays.mli* 128a⟩≡

```
val draw_searched_rectangles:
  dw:Model2.drawing -> unit

val motion_notify:
  Model2.world -> GdkEvent.Motion.t -> bool

val paint_initial:
  Model2.drawing -> unit
val hook_finish_paint:
  Model2.world -> unit
```

C.28 view_overlays.ml

128b ⟨*view_overlays.ml* 128b⟩≡

```
⟨Facebook copyright 2⟩
open Common
(* floats are the norm in graphics *)
open Common2.ArithFloatInfix

open Model2
module F = Figures
module T = Treemap
module CairoH = Cairo_helpers
module M = Model2
module Controller = Controller2
module Style = Style2

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

```
(*
 * This module mainly modifies the dw.overlay cairo surface. It also
 * triggers the refresh_da which triggers itself the expose event
 * which triggers the View2.assemble_layers composition of dw.pm with
 * dw.overlay.
 *)

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

let readable_txt_for_label txt current_root =
  let readable_txt =
    if current_root =$= txt (* when we are fully zoomed on one file *)
    then "root"
    else Common.readable ~root:current_root txt
  in
  if String.length readable_txt > 25
  then
    let dirs = Filename.dirname readable_txt +> Common.split "/" in
    let file = Filename.basename readable_txt in
    spf "%s/.../%s" (List.hd dirs) file
  else readable_txt

let with_overlay dw f =
  let cr_overlay = Cairo.create dw.overlay in
  View_mainmap.zoom_pan_scale_map cr_overlay dw;
  f cr_overlay

(*****************************************************************************)
(* The overlays *)
(*****************************************************************************)

(* ---------------------------------------------------------------------- *)
(* The current filename *)
(* ---------------------------------------------------------------------- *)
⟨draw_label_overlay 61b⟩

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

⟨draw_rectangle_overlay 61a⟩

(* ---------------------------------------------------------------------- *)
(* Uses and users macrolevel *)
```

```
(* ------------------------------------------------------------------- *)
let draw_deps_files tr dw model =
 with_overlay dw (fun cr_overlay ->
   let uses_rect, users_rect = M.deps_rects_of_rect tr dw model in
   (* todo: glowing layer *)
   uses_rect +> List.iter (fun r ->
     CairoH.draw_rectangle_figure ~cr:cr_overlay ~color:"green" r.T.tr_rect;
   );
   users_rect +> List.iter (fun r ->
     CairoH.draw_rectangle_figure ~cr:cr_overlay ~color:"red" r.T.tr_rect;
   )
 )

(* ------------------------------------------------------------------- *)
(* Uses and users microlevel *)
(* ------------------------------------------------------------------- *)
(* todo: better fisheye, with good background color *)
let draw_magnify_line_overlay_maybe ?honor_color dw line microlevel =
  with_overlay dw (fun cr_overlay ->
    let font_size = microlevel.layout.lfont_size in
    let font_size_real = CairoH.user_to_device_font_size cr_overlay font_size in

    (* todo: put in style *)
    if font_size_real < 5.
    then Draw_microlevel.draw_magnify_line
          ?honor_color cr_overlay line microlevel
  )

let draw_deps_entities n dw model =
 with_overlay dw (fun cr_overlay ->

   line_and_microlevel_of_node_opt n dw model
   +> Common.do_option (fun (_n2, line, microlevel) ->
     let rectangle = microlevel.line_to_rectangle line in
     CairoH.draw_rectangle_figure ~cr:cr_overlay ~color:"white" rectangle
   );

   let uses, users = M.deps_nodes_of_node_clipped n dw model in
   uses +> List.iter (fun (_n2, line, microlevel) ->
     let rectangle = microlevel.line_to_rectangle line in
     CairoH.draw_rectangle_figure ~cr:cr_overlay ~color:"green" rectangle;
   );
   users +> List.iter (fun (_n2, line, microlevel) ->
     let rectangle = microlevel.line_to_rectangle line in
     CairoH.draw_rectangle_figure ~cr:cr_overlay ~color:"red" rectangle;
```

```
     let lines_used = M.lines_where_used_node n line microlevel in
     lines_used +> List.iter (fun line ->
       let rectangle = microlevel.line_to_rectangle line in
       CairoH.draw_rectangle_figure ~cr:cr_overlay ~color:"purple" rectangle;

       draw_magnify_line_overlay_maybe ~honor_color:false dw line microlevel;
     );
   );
 )

(* ---------------------------------------------------------------------- *)
(* Tooltip/hovercard current entity *)
(* ---------------------------------------------------------------------- *)
(* assumes cr_overlay has not been zoom_pan_scale *)
let draw_tooltip ~cr_overlay ~x ~y n g =

  let pred = Graph_code.pred n Graph_code.Use g in
  let succ = Graph_code.succ n Graph_code.Use g in
  let files =
    pred
    +> Common.map_filter (fun n ->
        Common2.optionise (fun () -> (Graph_code.file_of_node n g)))
    +> Common.sort +> Common2.uniq
  in
  let str = spf "
 Entity: %s
 #Users: %d (%d different files)
 #Uses: %d
" (Graph_code.string_of_node n)
    (List.length pred) (List.length files)
    (List.length succ)
  in
  let xs = Common2.lines str in

  (* copy paste of draw_label_overlay *)
  Cairo.select_font_face cr_overlay "serif"
    Cairo.FONT_SLANT_NORMAL Cairo.FONT_WEIGHT_NORMAL;
  Cairo.set_font_size cr_overlay Style.font_size_filename_cursor;

  let template = "peh" in
  let max_length =
    xs +> List.map (String.length) +> Common2.maximum +> float_of_int in

  let extent = CairoH.text_extents cr_overlay template in
  let tw = extent.Cairo.text_width * ((max_length / 3.) +> ceil) in
  let th = extent.Cairo.text_height * 1.2 in
```

```
  let nblines = List.length xs +> float_of_int in
  let refx = x - tw / 2. in
  let refy = y - (th * nblines) in

  CairoH.fill_rectangle ~cr:cr_overlay
    ~x:(refx + extent.Cairo.x_bearing) ~y:(refy + extent.Cairo.y_bearing)
    ~w:tw ~h:(th * nblines)
    ~color:"black"
    ~alpha:0.5
    ();

  Cairo.set_source_rgba cr_overlay 1. 1. 1.    1.0;
  xs +> Common.index_list_0 +> List.iter (fun (txt, line) ->
    let line = float_of_int line in
    Cairo.move_to cr_overlay refx (refy + line * th);
    CairoH.show_text cr_overlay txt;
  );
  ()

(* ------------------------------------------------------------------------ *)
(* The selected rectangles *)
(* ------------------------------------------------------------------------ *)
```

⟨*draw_searched_rectangles* 62a⟩

⟨*zoomed_surface_of_rectangle* 62b⟩

```
(*****************************************************************************)
(* Assembling overlays *)
(*****************************************************************************)

let paint_initial dw =
  let cr_overlay = Cairo.create dw.overlay in
  CairoH.clear cr_overlay
  (* can't do draw_deps_entities w.current_node here because
   * of lazy_paint(), the file content will not be ready yet
   *)

(* a bit ugly, but have to because of lazy_paint optimization *)
let hook_finish_paint w =
  (* pr2 "Hook_finish_paint"; *)
  let dw = w.dw in
  w.current_node +> Common.do_option (fun n ->
    Async.async_get_opt w.model +> Common.do_option (fun model ->
      draw_deps_entities n dw model
```

```
    ));
  w.current_node_selected +> Common.do_option (fun n ->
    Async.async_get_opt w.model +> Common.do_option (fun model ->
      draw_deps_entities n dw model
    ))
```

⟨*motion_refresher* 62c⟩

⟨*idle* 48c⟩

C.29 ui_search.mli

133a ⟨*ui_search.mli* 133a⟩≡

```
val dialog_search_def:
  Model2.model Async.t -> string option

val run_grep_query:
  root:string -> string -> (string * Model2.line) list

val run_tbgs_query:
  root:string -> string -> (string * Model2.line) list
```

C.30 ui_search.ml

133b ⟨*ui_search.ml* 133b⟩≡

```
⟨Facebook copyright 2⟩
open Common

module G = Gui
module M = Model2

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

(*
 * todo:
 *  - integrate lfs at some point!
 *  - integrate pof at some point!
 *)
```

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

⟨*dialog_search_def* 49j⟩

⟨*run_grep_query* 50⟩

⟨*run_tbgs_query* 51a⟩

C.31 ui_navigation.mli

134a ⟨*ui_navigation.mli* 134a⟩≡

```
val go_back:
  Model2.world -> unit

val go_dirs_or_file:
  ?current_grep_query:(Common.filename, Model2.line) Hashtbl.t option ->
  Model2.world -> Common.filename list -> unit
```

C.32 ui_navigation.ml

134b ⟨*ui_navigation.ml* 134b⟩≡

⟨*Facebook copyright* 2⟩

```
open Common

open Model2
module G = Gui
module Controller = Controller2

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

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

⟨*go_back* 48b⟩

⟨*go_dirs_or_file* 45⟩

C.33 parsing2.mli

C.34 parsing2.ml

C.35 completion2.mli

C.36 completion2.ml

C.37 style2.mli

C.38 style2.ml

135 ⟨*style2.ml* 135⟩≡

```
⟨Facebook copyright 2⟩
open Common

module SH = Highlight_code
module Flag = Flag_visual

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

⟨zoom_factor_incruste_mode 81a⟩

⟨threshold_draw_dark_background_font_size_real 81b⟩

let font_size_filename_cursor = 30.

(* see also Cairo_helpers.prepare_string which may double the spaces *)
let font_text =
  match 0 with
  | 0 -> "serif"

  | 1 -> "helvetica"
  | 2 -> "courier"
  | 3 -> "arial"
  | 4 -> "consolas"
  | 5 -> "dejavu"
  | 6 -> "terminal"
  | _ -> raise Impossible

⟨size_font_multiplier_of_categ() 79b⟩

⟨windows_params() 79a⟩
```

C.39 editor_connection.mli

C.40 editor_connection.ml

136a ⟨*editor_connection.ml* 136a⟩≡
```
⟨Facebook copyright 2⟩
open Common

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

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

⟨emacs configuration 47b⟩

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

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

⟨open_file_in_current_editor() 48a⟩
```

C.41 async.mli

136b ⟨*async.mli* 136b⟩≡
```
⟨type async 75⟩
⟨async functions sig 76a⟩
```

C.42 async.ml

137a ⟨*async.ml* 137a⟩≡
```
⟨Facebook copyright 2⟩
open Common

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

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

⟨*type async* 75⟩

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

⟨*async functions* 76b⟩

C.43 cairo_helpers.mli

137b ⟨*cairo_helpers.mli* 137b⟩≡

⟨*cairo helpers functions sig* 84⟩

C.44 cairo_helpers.ml

137c ⟨*cairo_helpers.ml* 137c⟩≡

```
⟨Facebook copyright 2⟩
open Common
(* floats are the norm in graphics *)
open Common2.ArithFloatInfix

open Figures
module F = Figures
module Color = Simple_color

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

(*****************************************************************************)
(* Helpers *)
(*****************************************************************************)
let (==~) = Common2.(==~)

(*****************************************************************************)
(* Text related *)
(*****************************************************************************)

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

```
let re_space = Str.regexp "^[ ]+$"
let re_tab = Str.regexp "^[\t]+$"
```

⟨*cairo helpers functions* 85⟩

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

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

D Changelog

Indexes

References

[1] Donald Knuth,, *Literate Programming*, http://en.wikipedia.org/wiki/Literate_Program cited page(s) 3

[2] Norman Ramsey, *Noweb*, http://www.cs.tufts.edu/~nr/noweb/ cited page(s) 3

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

[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)