

```

1 #include <gtk/gtk.h>
2 #include <math.h>
3 #include <time.h>
4 #include <gsl/gsl_rng.h>
5 #include <gsl/gsl_randist.h>
6
7 #include "sim.h"
8 #include "afield.h"
9 #include "graph.h"
10 #include "ui_afield.h"
11 #include "ui_graph.h"
12 #include "darea.h"
13 #include "status.h"
14 #include "timer.h"
15 #include "util.h"
16 /*
17 #include "sound.h"
18 */
19 #if HAVE_CONFIG_H
20 #include <config.h>
21 #endif
22
23 #include "gettext.h"
24 #define _(String) gettext (String)
25 #define N_(String) gettext_noop (String)
26
27 static void resume_sim(GtkWidget *button, MyTimer *timer);
28 static void pause_sim(GtkWidget *button, MyTimer *timer);
29 static void stop_sim(GtkWidget *button, gint *quit);
30
31 static gdouble calc_duration(gint number, gdouble thalf);
32 static gdouble exp_growth(gdouble t, SimData *data);
33 static gint decay_real(gdouble t, gint n, gdouble thalf, gsl_rng *rand);
34
35 static gint decay_stat(gdouble t, gint n, gint n0, gdouble thalf);
36
37 void sim_decay(GtkWidget *button_start, gsl_rng *rand)
38 {
39     GtkWidget *top, **darea, *button_stop,
40             *spin_number, **spin_hptime,
41             **label_atom, *label_time;
42     gdouble t, thalf, tstart, tnext, told, tstep, tloop;
43
44     CoordSystem *coord;
45     Graph **graph;
46     Point *point, *old_point;
47
48     MyTimer *timer;
49     gint quit;
50     gint number, pos, state, decays, i, a, b;
51     AtomField *afield;
52
53     gulong *sig_darea[N_DAREAS];
54
55     SimData *sdata;
56     GraphFunc *gf;
57
58     top = gtk_widget_get_toplevel(button_start);
59
60     /* holt ein paar gespeicherte Widgets äöü */

```

```

61 button_stop = g_object_get_data(G_OBJECT(top), "button_stop");
62 spin_number = g_object_get_data(G_OBJECT(top), "spin_number");
63 spin_hptime = g_object_get_data(G_OBJECT(top), "spin_hptime");
64 darea = g_object_get_data(G_OBJECT(top), "darea");
65 label_atom = g_object_get_data(G_OBJECT(top), "label_atom");
66 label_time = g_object_get_data(G_OBJECT(top), "label_time");
67
68 /* ersetzt den Startbutton durch den Pausebutton */
69 g_signal_handlers_block_by_func(G_OBJECT(button_start),
70                                (gpointer) sim_decay, rand);
71 gtk_button_set_label(GTK_BUTTON(button_start), _("pause"));
72 gtk_button_leave(GTK_BUTTON(button_start));
73
74 /* bereitet den Stopbutton vor */
75 quit = 0;
76 gtk_widget_set_sensitive(button_stop, TRUE);
77 g_signal_connect(G_OBJECT(button_stop), "clicked",
78                 G_CALLBACK(stop_sim), &quit);
79
80 /* holt die Eingaben des Nutzers von den Spinbutton */
81 number = gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(spin_number))
82 ;
83 for (i = 0; i < ATOM_STATES-1; i++)
84     thalf[i] = gtk_spin_button_get_value(GTK_SPIN_BUTTON(spin_hptime[i])
85 );
86
87 /* packt die Eingaben in die SimData Struktur */
88 sdata = (SimData *) g_malloc(sizeof(SimData));
89 sdata->atoms[0] = number;
90 sdata->states = 3;
91 sdata->atoms[1] = 0;
92 sdata->atoms[2] = 0;
93 sdata->thalf[0] = thalf[0];
94 sdata->thalf[1] = thalf[1];
95
96 gf = (GraphFunc *) g_malloc(sizeof(GraphFunc));
97 gf->func = exp_growth;
98 gf->data = sdata;
99
100 afield = afield_new(number, (darea + 1)->allocation.width,
101                     (darea + 1)->allocation.height);
102 afield_randomize(afield, rand);
103
104 sig_darea[0] = g_signal_connect(G_OBJECT(darea[0]),
105 "configure_event", G_CALLBACK(afield_resize), afield);
106
107 tstep = 0.003;
108 pos = 0;
109
110 update_status_atoms(darea[0], sdata->atoms);
111 update_status_time(darea[0], 0.0);
112 darea_clear(darea[0]);
113 afield_draw(darea[0], afield);
114
115 gdk_window_get_size((darea + 1)->window, &a, &b);
116 coord = coord_system_new((darea + 1)->allocation.width,
117                           (darea + 1)->allocation.height,
118                           0, calc_duration(number, thalf),
119                           0, number);
120
121 darea_clear(darea[1]);

```

```

122 coord_system_draw(darea[1], coord);
123
124 graph_draw_func(graph_func, darea[1], coord);
125
126 graph = g_malloc(2 * sizeof(Graph *));
127 graph[0] = graph_new(0);
128 coord->graphs = graph;
129
130 sig_darea[1] = g_signal_connect(G_OBJECT(darea[1]), "configure_event",
131                                G_CALLBACK(graph_resize), coord);
132
133 while (g_main_iteration(FALSE));
134
135 timer = timer_new();
136 g_signal_connect(G_OBJECT(button), "clicked",
137                 G_CALLBACK(pause_sim), timer);
138 tstart = tnext = told = timer_elapsed(timer);
139
140 while(sdata->atoms[0] > 0 && (!quit)) {
141     t = timer_elapsed(timer) - tstart;
142     if (t >= tnext) {
143         tloop = t - told;
144         told = t;
145         for (state = 0; state < sdata->states - 1; state++) {
146             decays = decay_real(tloop, sdata->atoms[state], thalf,
147                                rand);
148             if (decays > 0) {
149                 sdata->atoms[state + 0] -= decays;
150                 sdata->atoms[state + 1] += decays;
151
152                 update_status_atoms(darea[0], sdata->atoms);
153
154                 for (i = 0; i < decays; i++) {
155                     (af->coords + pos)->state = 1;
156                     draw_atom(darea[0],
157                              (afield->coords + pos),
158                              afield->wide);
159                     pos++;
160                 }
161             }
162
163             point = point_alloc(t, sdata->atoms[0]);
164             if (graph->points != NULL) {
165                 old_point = graph->points->data;
166                 graph_draw_line(darea[1], coord,
167                                old_point->x, old_point->y,
168                                point->x, point->y, 0);
169             }
170             graph_add(graph, point);
171         }
172
173         tnext += tstep;
174     }
175     update_status_time(darea[0], t);
176
177     while (gtk_events_pending())
178         gtk_main_iteration();
179     /* while (g_main_iteration(FALSE)); */
180 }
181
182 g_signal_handlers_disconnect_matched(G_OBJECT(button_stop),

```

```

183                                     G_SIGNAL_MATCH_FUNC,
184                                     0,
185                                     0,
186                                     NULL,
187                                     (gpointer) stop_sim,
188                                     NULL);
189
190     gtk_widget_set_sensitive(button_stop, FALSE);
191
192     if (timer_is_running(timer))
193         g_signal_handlers_disconnect_matched(G_OBJECT(button_start),
194                                             G_SIGNAL_MATCH_FUNC,
195                                             0, 0, NULL,
196                                             (gpointer) pause_sim,
197                                             NULL);
198     else
199         g_signal_handlers_disconnect_matched(G_OBJECT(button_start),
200                                             G_SIGNAL_MATCH_FUNC, 0, 0, NULL, (gpointer) resume_sim,
201                                             NULL);
202
203     g_signal_handlers_unblock_by_func(G_OBJECT(button_start),
204                                       (gpointer) sim_decay, rand);
205     gtk_button_set_label(GTK_BUTTON(button_start), _("start"));
206
207     g_signal_handler_disconnect(G_OBJECT(darea[0]), sig_darea[0]);
208     g_signal_handler_disconnect(G_OBJECT(darea[1]), sig_darea[1]);
209
210     timer_free(timer);
211
212     afield_free(afield);
213
214     coord_system_free(coord);
215 }
216
217 static void resume_sim(GtkWidget *button, MyTimer *timer)
218 {
219     timer_start(timer);
220     g_signal_handlers_disconnect_by_func(G_OBJECT(button),
221                                         (gpointer) resume_sim, timer);
222     g_signal_connect(G_OBJECT(button), "clicked",
223                     G_CALLBACK(pause_sim), timer);
224     gtk_button_set_label(GTK_BUTTON(button), _("pause"));
225 }
226
227 static void pause_sim(GtkWidget *button, MyTimer *timer)
228 {
229     timer_stop(timer);
230     g_signal_handlers_disconnect_by_func(G_OBJECT(button),
231                                         (gpointer) pause_sim, timer);
232     g_signal_connect(G_OBJECT(button), "clicked",
233                     G_CALLBACK(resume_sim), timer);
234     gtk_button_set_label(GTK_BUTTON(button), _("resume"));
235 }
236
237 static void stop_sim(GtkWidget *button, gint *quit)
238 {
239     *quit = 1;
240 }
241
242 static gdouble calc_duration(gint number, gdouble thalf)
243 {

```

```

244     return -thalf * log2(1.0 / number) + thalf;
245 }
246
247 static gdouble exp_growth(gdouble t, SimData *data)
248 {
249     return (gint) (data->atoms[0] * pow(0.5, (t / data->thalf[0])) + 0.5);
250 }
251
252 static gint decay_stat(gdouble t, gint n, gint n0, gdouble thalf)
253 {
254     return n - (gint) ((n0 * pow(0.5, t / thalf)) + 0.5);
255 }
256
257 static gint decay_real(gdouble t, gint n, gdouble thalf, gsl_rng *rand)
258 {
259     /* return (gsl_ran_binomial(rand, (1.0 - pow(0.5, (t / thalf))), n)); */
260     return gsl_ran_poisson(rand, ((1.0 - pow(0.5, (t / thalf)))) * n);
261 }

```