

```

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_h_time,
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 */

```

```

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_htime = g_object_get_data(G_OBJECT(top), "spin_htime");
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 for (i = 0; i < ATOM_STATES-1; i++)
83     thalf[i] = gtk_spin_button_get_value(GTK_SPIN_BUTTON(spin_htime[i]));
84
85 /* packt die Eingaben in die SimData Struktur */
86 sdata = (SimData *) g_malloc(sizeof(SimData));
87 sdata->atoms[0] = number;
88 sdata->states = 3;
89 sdata->atoms[1] = 0;
90 sdata->atoms[2] = 0;
91 sdata->thalf[0] = thalf[0];
92 sdata->thalf[1] = thalf[1];
93
94 gf = (GraphFunc *) g_malloc(sizeof(GraphFunc));
95 gf->func = exp_growth;
96 gf->data = sdata;
97
98 afield = afield_new(number, (darea + 1)->allocation.width,
99                      (darea + 1)->allocation.height);
100 afield_randomize(afield, rand);
101
102 sig_darea[0] = g_signal_connect(G_OBJECT(darea[0]), "configure_event",
103                                 G_CALLBACK(afield_resize), afield);
104
105 tstep = 0.003;
106 pos = 0;
107
108 update_status_atoms(darea[0], sdata->atoms);
109 update_status_time(darea[0], 0.0);
110 darea_clear(darea[0]);
111 afield_draw(darea[0], afield);
112
113 gdk_window_get_size((darea + 1)->window, &a, &b);
114 coord = coord_system_new((darea + 1)->allocation.width,
115                           (darea + 1)->allocation.height,
116                           0, calc_duration(number, thalf),
117                           0, number);
118
119 darea_clear(darea[1]);
120 coord_system_draw(darea[1], coord);
121

```

```

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

```

```

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

```

```

243 }
244
245 static gdouble calc_duration(gint number, gdouble thalf)
246 {
247     return -thalf * log2(1.0 / number) + thalf;
248 }
249
250 static gdouble exp_growth(gdouble t, SimData *data)
251 {
252     return (gint) (data->atoms[0] * pow(0.5, (t / data->thalf[0])) + 0.5);
253 }
254
255 static gint decay_stat(gdouble t, gint n, gint n0, gdouble thalf)
256 {
257     return n - (gint) ((n0 * pow(0.5, t / thalf)) + 0.5);
258 }
259
260 static gint decay_real(gdouble t, gint n, gdouble thalf, gsl_rng *rand)
261 {
262     /* return (gsl_rng_binomial(rand, (1.0 - pow(0.5, (t / thalf))), n)); */
263     return gsl_rng_poisson(rand, ((1.0 - pow(0.5, (t / thalf)))) * n);
264 }
265

```