```c
#include <gtk/gtk.h>
#include <math.h>
#include <time.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>

#include "sim.h"
#include "afield.h"
#include "graph.h"
#include "ui_afield.h"
#include "ui_graph.h"
#include "darea.h"
#include "status.h"
#include "timer.h"
#include "util.h"
/*
#include ”sound.h”
*/
#if HAVE_CONFIG_H
#include <config.h>
#endif

#include "gettext.h"
#define _(String) gettext (String)
#define N_(String) gettext_noop (String)

static void resume_sim(GtkWidget *button, MyTimer *timer);
static void pause_sim(GtkWidget *button, MyTimer *timer);
static void stop_sim(GtkWidget *button, gint *quit);

static gdouble calc_duration(gint number, gdouble thalf);
static gdouble exp_growth(gdouble t, SimData *data);
static gint decay_real(gdouble t, gint n, gdouble thalf, gsl_rng *rand);

static gint decay_stat(gdouble t, gint n, gint n0, gdouble thalf);

void sim_decay(GtkWidget *button_start, gsl_rng *rand)
{
    GtkWidget *top, **darea, *button_stop,
               *spin_number, **spin_htime,
               **label_atom, *label_time;
    gdouble t, thalf, tstart, tnext, told, tstep, tloop;

    CoordSystem *coord;
    Graph **graph;
    Point *point, *old_point;

    MyTimer *timer;
    gint quit;
    gint number, pos, state, decays, i, a, b;
    AtomField *afield;

    gulong *sig_darea[N_DAREAS];

    SimData *sdata;
    GraphFunc *gf;

    top = gtk_widget_get_toplevel(button_start);

    /* holt ein paar gespeicherte Widgets äöü */
    button_stop = g_object_get_data(G_OBJECT(top), "button_stop");
    spin_number = g_object_get_data(G_OBJECT(top), "spin_number");
    spin_htime = g_object_get_data(G_OBJECT(top), "spin_htime");
    darea = g_object_get_data(G_OBJECT(top), "darea");
    label_atom = g_object_get_data(G_OBJECT(top), "label_atom");
    label_time = g_object_get_data(G_OBJECT(top), "label_time");

    /* ersetzt den Startbutton durch den Pausebutton */
    g_signal_handlers_block_by_func(G_OBJECT(button_start),
                                    (gpointer) sim_decay, rand);
    gtk_button_set_label(GTK_BUTTON(button_start), _("pause"));
    gtk_button_leave(GTK_BUTTON(button_start));

    /* bereitet den Stopbutton vor  */
    quit = 0;
    gtk_widget_set_sensitive(button_stop, TRUE);
    g_signal_connect(G_OBJECT(button_stop), "clicked",
                       G_CALLBACK(stop_sim), &quit);

    /* holt die Eingaben des Nutzers von den Spinbutton */
    number = gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(spin_number))
              ;
    for (i = 0; i < ATOM_STATES-1; i++)
        thalf[i] = gtk_spin_button_get_value(GTK_SPIN_BUTTON(spin_htime[i])
                     );

    /* packt die Eingaben in die SimData Struktur */
    sdata = (SimData *) g_malloc(sizeof(SimData));
    sdata->atoms[0] = number;
    sdata->states = 3;
    sdata->atoms[1] = 0;
    sdata->atoms[2] = 0;
    sdata->thalf[0] = thalf[0];
    sdata->thalf[1] = thalf[1];

    gf = (GraphFunc *) g_malloc(sizeof(GraphFunc));
    gf->func = exp_growth;
    gf->data = sdata;

    afield = afield_new(number, (darea + 1)->allocation.width,
              (darea + 1)->allocation.height);
    afield_randomize(afield, rand);

    sig_darea[0] = g_signal_connect(G_OBJECT(darea[0]),
    "configure_event", G_CALLBACK(afield_resize), afield);

    tstep = 0.003;
    pos = 0;

    update_status_atoms(darea[0], sdata->atoms);
    update_status_time(darea[0], 0.0);
    darea_clear(darea[0]);
    afield_draw(darea[0], afield);

    gdk_window_get_size((darea + 1)->window, &a, &b);
    coord = coord_system_new((darea + 1)->allocation.width,
                            (darea + 1)->allocation.height,
                            0, calc_duration(number, thalf),
                            0, number);

    darea_clear(darea[1]);
```

```c
    coord_system_draw(darea[1], coord);

    graph_draw_func(graph_func, darea[1], coord);

    graph = g_malloc(2 * sizeof(Graph *));
    graph[0] = graph_new(0);
    coord->graphs = graph;

    sig_darea[1] = g_signal_connect(G_OBJECT(darea[1]), "configure_event",
                                    G_CALLBACK(graph_resize), coord);

    while (g_main_iteration(FALSE));

    timer = timer_new();
    g_signal_connect(G_OBJECT(button), "clicked",
                     G_CALLBACK(pause_sim), timer);
    tstart = tnext = told = timer_elapsed(timer);

    while(sdata->atoms[0] > 0 && (!quit)) {
        t = timer_elapsed(timer) - tstart;
        if (t >= tnext) {
            tloop = t - told;
            told = t;
            for (state = 0; state < sdata->states - 1; state++) {
                decays = decay_real(tloop, sdata->atoms[state], thalf,
                         rand);
                if (decays > 0) {
                    sdata->atoms[state + 0] -= decays;
                    sdata->atoms[state + 1] += decays;

                    update_status_atoms(darea[0], sdata->atoms);

                    for (i = 0; i < decays; i++) {
                        (af->coords + pos)->state = 1;
                        draw_atom(darea[0],
                                  (afield->coords + pos),
                                  afield->wide);
                        pos++;
                    }
                }

                point = point_alloc(t, sdata->atoms[0]);
                if (graph->points != NULL) {
                    old_point = graph->points->data;
                    graph_draw_line(darea[1], coord,
                                    old_point->x, old_point->y,
                                    point->x, point->y, 0);
                }
                graph_add(graph, point);
            }

            tnext += tstep;
        }
        update_status_time(darea[0], t);

        while (gtk_events_pending())
            gtk_main_iteration();
/*          while (g_main_iteration(FALSE)); */
    }

    g_signal_handlers_disconnect_matched(G_OBJECT(button_stop),
                                            G_SIGNAL_MATCH_FUNC,
                                            0,
                                            0,
                                            NULL,
                                            (gpointer) stop_sim,
                                            NULL);

    gtk_widget_set_sensitive(button_stop, FALSE);

    if (timer_is_running(timer))
        g_signal_handlers_disconnect_matched(G_OBJECT(button_start),
                                                G_SIGNAL_MATCH_FUNC,
                                                0, 0, NULL,
                                                (gpointer) pause_sim,
                                                NULL);
    else
        g_signal_handlers_disconnect_matched(G_OBJECT(button_start),
                 G_SIGNAL_MATCH_FUNC, 0, 0, NULL, (gpointer) resume_sim,
                                                     NULL);

    g_signal_handlers_unblock_by_func(G_OBJECT(button_start),
                                         (gpointer) sim_decay, rand);
    gtk_button_set_label(GTK_BUTTON(button_start), _("start"));

    g_signal_handler_disconnect(G_OBJECT(darea[0]), sig_darea[0]);
    g_signal_handler_disconnect(G_OBJECT(darea[1]), sig_darea[1]);

    timer_free(timer);

    afield_free(afield);

    coord_system_free(coord);
}

static void resume_sim(GtkWidget *button, MyTimer *timer)
{
    timer_start(timer);
    g_signal_handlers_disconnect_by_func(G_OBJECT(button),
    (gpointer) resume_sim, timer);
    g_signal_connect(G_OBJECT(button), "clicked",
                      G_CALLBACK(pause_sim), timer);
    gtk_button_set_label(GTK_BUTTON(button), _("pause"));
}

static void pause_sim(GtkWidget *button, MyTimer *timer)
{
    timer_stop(timer);
    g_signal_handlers_disconnect_by_func(G_OBJECT(button),
             (gpointer) pause_sim, timer);
    g_signal_connect(G_OBJECT(button), "clicked",
                      G_CALLBACK(resume_sim), timer);
    gtk_button_set_label(GTK_BUTTON(button), _("resume"));
}

static void stop_sim(GtkWidget *button, gint *quit)
{
    *quit = 1;
}

static gdouble calc_duration(gint number, gdouble thalf)
{
    return -thalf * log2(1.0 / number) + thalf;
}

static gdouble exp_growth(gdouble t, SimData *data)
{
    return (gint) (data->atoms[0] * pow(0.5, (t / data->thalf[0])) + 0.5);
}

static gint decay_stat(gdouble t, gint n, gint n0, gdouble thalf)
{
    return n - (gint) ((n0 * pow(0.5, t / thalf)) + 0.5);
}

static gint decay_real(gdouble t, gint n, gdouble thalf, gsl_rng *rand)
{
/*      return (gsl_ran_binomial(rand, (1.0 - pow(0.5, (t / thalf))), n)); */
   return gsl_ran_poisson(rand, ((1.0 - pow(0.5, (t / thalf)))) * n);
}
```