FILE: WALKER.F90

```
1  module walker
2
3     use global_env
4     use calendar
5     use cell
6
7     integer, dimension(:,:), allocatable :: walker_pos
8     integer, dimension(:),   allocatable :: walker_col
9
10    integer :: n_walkers, max_walkers
11
12  contains
13
14    subroutine walker_init(max_walkers)
15
16       implicit none
17
18       integer, intent(in) :: max_walkers
19
20       allocate(walker_pos(max_walkers,3))
21       allocate(walker_col(max_walkers))
22
23    end subroutine walker_init
24
25    subroutine walker_predict_event(walker_id)
26    !*****************************************************
27    ! Definition of action
28    !    1 = migration (-x)
29    !    2 = migration (+x)
30    !    3 = migration (-y)
```

```
31      !     4 = migration (+y)
32      !     5 = migration (-z)
33      !     6 = migration (+z)
34      !     7 = membrane association
35      !     8 = membrane desociation
36      !
37      !   walker_action takes care of boundary conditions
38      !*****************************************************
39
40         implicit none
41
42         integer, intent(in) :: walker_id
43
44         integer :: action
45         real(DP) :: tau
46         real(DP) :: g
47
48         if (walker_pos(walker_id,Z)==box_size(Z))  then
49            ! in front of the membrane
50            call random_number(g)
51            tau = -log(g)/(5.0_DP*k_3d+k_on)
52            call random_number(g)
53            if(k_on/(5.0_DP*k_3d+k_on)>g) then
54               action = 7
55            else
56               call random_number(g)
57               action = ceiling(g*5.0_DP)
58            end if
59         else if (walker_pos(walker_id,Z)>box_size(Z))  then
60            ! on the membrane
61            call random_number(g)
```

```
62          tau = -log(g)/(4.0_DP*k_2d+k_off)
63          call random_number(g)
64          if(k_off/(4.0_DP*k_2d+k_off)>g) then
65             action = 8
66          else
67             call random_number(g)
68             action = ceiling(g*4.0_DP)
69          end if
70       else
71          ! in the cytosol
72          call random_number(g)
73          tau = -log(g)/(6.0_DP*k_3d)
74          call random_number(g)
75          action = ceiling(g*6.0_DP)
76       end if
77
78       call calendar_schedule_event(action,walker_id,0,
            time_current+tau)
79
80    end subroutine walker_predict_event
81
82    subroutine walker_action(walker_id,action)
83
84       implicit none
85
86       integer, intent(in) :: walker_id
87       integer, intent(in) :: action
88       integer, dimension(3) :: p ! position
89       integer :: n
90       logical :: occupied
91
```

```
92        p = walker_pos(walker_id,1:3)
93
94        select case (action)
95
96        case(1)
97           p(X) = p(X)-1
98           if(p(X)<1) p(X)=box_size(X)   ! periodic
99
100       case(2)
101          p(X) = p(X)+1
102          if(p(X)>box_size(X)) p(X)=1   ! periodic
103
104       case(3)
105         p(Y) = p(Y)-1
106         if(p(Y)<1) p(Y)=box_size(Y)    ! periodic
107
108       case(4)
109          p(Y) = p(Y)+1
110          if(p(Y)>box_size(Y)) p(Y)=1   ! periodic
111
112       case(5)
113          p(Z) = p(Z)-1
114          if(p(Z)<1) p(Z)=1          ! reflection
115
116       case(6)
117          p(Z) = p(Z)+1
118          if(debug .and. p(Z)>box_size(Z)) then
119             write(fdbg,*) '**walker_action**: Unexpected action.
                 '
120             stop
121          end if
```

```
122
123         case(7)
124            p(Z) = p(Z)+1
125            if(debug .and. p(Z)/=box_size(Z)+1) then
126               write(fdbg,*) '**walker_action**: Unexpected
                     position.'
127               stop
128            end if
129
130         case(8)
131            p(Z) = p(Z)-1
132            if(debug .and. p(Z)/=box_size(Z)) then
133               write(fdbg,*) '**walker_action**: Dissociation
                     failed.'
134               stop
135            end if
136
137         end select
138
139         call cell_find_walkers(cell_coordinates(p),
               cell_local_walkers)
140
141         occupied=.false.
142         n=1
143         do while(cell_local_walkers(n)>0)
144            if(cell_local_walkers(n) /= walker_id .and. all(p==
                  walker_pos(cell_local_walkers(n),:))) then
145               occupied=.true.
146               exit
147            end if
148            n=n+1
```

```
149         end do
150
151         if(.not.occupied) then
152            walker_pos(walker_id,:) = p
153            call cell_crossing(walker_id,p)
154         else
155            n_collisions=n_collisions+1
156         end if
157
158      end subroutine walker_action
159
160   end module walker
```