#ifndef PARAMETER\_VALUES\_H

#define PARAMETER\_VALUES\_H

#include <array>

#include <bitset>

struct Cell\_params{

using reaction\_norm = std::array<double,2>;

constexpr static reaction\_norm init\_reprod{0.0,0.0};

constexpr static reaction\_norm init\_repair{0.0,0.0};

// proportion resources for foraging: 1-reprod-repair

constexpr static reaction\_norm init\_harvest{1.0,1.0}; // (relative) effort into harvesting foraged resources

constexpr static reaction\_norm init\_damalloc{0.0,0.0}; // damage equally distributed at first

constexpr static std::array<reaction\_norm,4> g\_init{init\_reprod,init\_repair,init\_harvest,init\_damalloc};

// mutations

constexpr static double mu{0.05};

constexpr static double mu\_v{0.001}; // viability loci 0->1

constexpr static double pr\_back{0.1}; // rel. prob 1->0: abs pr =mu\_v\*pr\_back

constexpr static double mu\_t{0}; // for damage transmission locus

constexpr static double sd\_mu{0.5};

constexpr static double mu\_h{0.01}; // for harvest (division of resources between cells)

// forage parameters

// for harvest (division of resources between cells)

// forage parameters

constexpr static double forage\_max{6.0};

constexpr static double q\_f{1}; // half-max

constexpr static bool type\_2{true}; // else (partially) convex type III

// repro params

// rate at which Pr(repro) approaches 1 with amount of resources 1-exp(-b\*res)

constexpr static double b\_r{0.8};

// mortality parameters

// adult cell mortality = c\*(d/d0)+(1-c)\*(d/d0)^4

// constexpr static double d0a{1000.0};

constexpr static double c{0.2};

// adult mortality when solitary for a bit:

constexpr static double m0{0};

// delta damage (lambda of exponential distribution; mean=1/dd)

constexpr static double dd{2.0};

// damage repair rate constant

constexpr static double b\_re{0.5};

// nr of bi-allelic loci

constexpr static unsigned n\_v\_loci{16};

constexpr static std::bitset<n\_v\_loci> null\_v\_genotype = 0;

// extra mort round 1-exp(-b\_svg\*I(1))

constexpr static double **b\_svg{0.08};** // here

// max damage to which max-damage allele corresponds

constexpr static double d\_max{4.0};

};

struct Pop\_params{

constexpr static unsigned long size{1000000};

// cell resources in generation 0

constexpr static double init\_resources{3.0};

};

// nr of traits to calc stats for

constexpr static unsigned n\_traits{45};

constexpr static unsigned long n\_timesteps{5000};

constexpr static unsigned skip{10};

**Cell.cpp**

#include "Cell.h"

void Cell::mutate(){

for (size\_t i=0; i<2; ++i) // repro & repair loci

for (size\_t j=0; j<2; ++j) if (ru()<p.mu){

g[i][j] += rnorm(p.sd\_mu);

// g[i][1]=g[i][0];

}

// harvest

for (size\_t j=0; j<2; ++j) if (ru()<p.mu\_h){

g[2][j] += rnorm(p.sd\_mu);

clip0\_epsilon(g[2][j]);

//g[2][1]=g[2][0];

}

// damage transmission locus

for (size\_t j=0; j<2; ++j) if (ru()<p.mu\_t){

g[3][j] += rnorm(p.sd\_mu);

//g[3][1]=g[3][0];

}

// mutate viability loci

// biased: mostly 0->1 (as long as pr\_back<1)

for (size\_t i=0; i<p.n\_v\_loci; ++i)

if (ru()<p.mu\_v) {

if (vg[i]==0) vg[i].flip();

else if (ru()<p.pr\_back) vg[i].flip();

}

}

**Cell.h**

#ifndef CELL\_H

#define CELL\_H

#include "parameter\_values.h"

#include "utils.h"

#include "randomnumbers.h"

class Cell{

// Parameters for Cells

inline static Cell\_params p{};

// Define aliases

using reaction\_norm = std::array<double, 2>; // 2 cell types

using genotype = std::array<reaction\_norm, 4>; // repro, repair (forage=1-repro-repair), harvest, damage transmitted

using v\_genotype = std::bitset<p.n\_v\_loci>;

genotype g{p.g\_init};

unsigned type{0};

double resources{0.0};

double damage{0.0};

v\_genotype vg{p.null\_v\_genotype};

public:

// constructor with type

Cell(const unsigned t) : type{t} {}

// regular copy constructor

Cell(const Cell& c) :

g{c.g},

type{c.type},

resources{c.resources},

damage{c.damage},

vg{c.vg}

{}

// Special copy constructor used in reproduction (from mother cell c)

// NB: resources are mom's investment in repro

Cell(const Cell& c, const double&& damage) :

g{c.g},

type{0},

resources{c.resources\*logist3(c.g[0][c.type],c.g[1][c.type],0)},

damage{std::move(damage)},

vg{c.vg}

{}

// move constructor required because copy constructor defined

// noexcept required or vector will use copy constructors instead

Cell(Cell&& c) noexcept :

g{c.g},

type{c.type},

resources{c.resources},

damage{c.damage},

vg{c.vg}

{}

// copy assignment operator

Cell& operator=(const Cell& rhs){

if (this != &rhs){

g=rhs.g;

type=rhs.type;

resources=rhs.resources;

damage=rhs.damage;

vg=rhs.vg;

}

return \*this;

}

// NB: move assignment disabled

void set\_type(const unsigned t) { type = t; }

void set\_resources(const double r) { resources = r; }

double forage\_returns(const bool concave) const {

double alloc\_forage{ resources \* logist3(g[0][type],g[1][type],2) };

if (concave) return p.forage\_max \* alloc\_forage/(alloc\_forage + p.q\_f);

double af\_sqr = alloc\_forage\*alloc\_forage;

return p.forage\_max \* af\_sqr/(af\_sqr + p.q\_f\*p.q\_f);

}

double get\_harvest() const { return g[2][type]; }

// convert damage to index for viability loci vector

size\_t d2i(const double d) const {

if (d>p.d\_max) return p.n\_v\_loci-1;

else return static\_cast<size\_t>(floor(d\*(p.n\_v\_loci-1)/p.d\_max));

}

bool dies\_ad() const {

// double x{damage/p.d0a};

// double p1{p.c\*x+(1-p.c)\*pow(x,4)};

double p1{ 0.0 };

double p2{1-exp(-p.b\_svg\*vg[d2i(damage)])};

double pr\_dead{p1+(1-p1)\*p2};

return ru() < pr\_dead;

}

bool dies\_sol() const { return ru() < p.m0; }

bool reproduces() const { return ru() < 1.0-exp(-p.b\_r\*logist3(g[0][type],g[1][type],0)\*resources); }

// bool reproduces() const { return ru() < 1.0/(1+exp(-p.b\_r\*logist3(g[0][type],g[1][type],0)\*resources+5)); }

void mutate();

void accumulate\_damage() { damage += rexp(p.dd); } // could make less stochastic

//void accumulate\_damage() { damage += p.dd; }

// additive repair (was multiplicative: damage\*=exp(-...)

void repair\_damage() { damage \*= exp(-p.b\_re\*logist3(g[0][type],g[1][type],1)\*resources); }

unsigned get\_type() const { return type; }

genotype get\_g() const { return g; }

v\_genotype get\_vg() const { return vg; }

double get\_damage() const { return damage; }

void set\_damage(const double d) { damage = d; }

double damage\_transmitted() const { return damage\*logist(g[3][type]); }

double get\_resources() const { return resources; }

};

#endif // CELL\_H