The following was taken from the header files as of 19 July 2012. The classes are embedded in a tree-like fashion, i.e. **class ancestors** contains the most general data and explicitly includes **class genus** which includes **class species** and **class population** at the same level, and so on. The level of inclusion is indicated by the number of tabs.

**class ancestors**

public:

ancestors(void);

ancestors(int buffanc);

~ancestors(void);

//ANCESTORS INTERACTIONS

void setTempPopulation(population tp);

void addToTheAncestors(population oldGen);

void removePopulation(vector<population>::iterator byeGen);

void resetTheAncestors();

void replacePopulation(int disone, population\* indi);

population assortativeMating(landscape\* babylaw, population\* oldPop,

CRandomMersenne\* raNuGe);

population cleaving(landscape\* babylaw, population\* oldPop,

CRandomMersenne\* raNuGe);

//ITERATOR ACCESS

vector<population>::iterator beginOfTheAncestors();

vector<population>::iterator endOfTheAncestors();

//THE ANCESTORS ACCESS

population getThisPopulationOfTheAncestors(int thisPopulation);

vector<population> getTheAncestors();

int intSizeOfTheAncestors() const;

int getPopulationSize(int k) const;

int getParent1s(int k) const;

int getParent2s(int k) const;

int getNeighbor1s(int k) const;

int getNeighbor2s(int k) const;

double getMutabilities(int k) const;

double getPhenotypeXs(int k) const;

double getPhenotypeYs(int k) const;

void doubleCheck(population\* P) const;

private:

vector<population> TheAncestors;

vector<int> PopulationSize;

vector<int> Parent1s;

vector<int> Parent2s;

vector<int> Neighbor1s;

vector<int> Neighbor2s;

vector<double> Mutabilities;

vector<double> PhenotypeXs;

vector<double> PhenotypeYs;

public:

void addToTheNations(genus oldGenus);

void removeGenus(vector<genus>::iterator byeGenus);

void resetTheNations();

void replaceGenus(int disone, genus\* genulogy);

genus getThisGenusOfTheNations(int thisGenus);

//ITERATOR ACCESS

vector<genus>::iterator beginOfTheNations();

vector<genus>::iterator endOfTheNations();

//THE ANCESTORS ACCESS

vector<genus> getTheNations();

int intSizeOfTheNations() const;

private:

vector<genus> TheNations;

public:

void setSaveCommands(char\* popc, char\* phnc, char\* parc, char\* nghc,

char\* mutc, char\* genc, char\* spsc, char\* spCc, char\* spdc,

char\* spic, char\* sptc, char\* expc);

char\* getIMeanMutabilityCommand() const;

char\* getPopulationCommand() const;

char\* getPhenotypeCommand() const;

char\* getParentCommand() const;

char\* getNeighborCommand() const;

char\* getMutabilityCommand() const;

char\* getGenusCommand() const;

char\* getSpeciesDetailCommand() const;

char\* getSpeciesIndivsCommand() const;

bool checkCommand(char\* command) const;

int checkCommand() const;

void writeAncestors(int simu, int gens);

private:

char\* Popc; //population

char\* Phnc; //phenotype

char\* Parc; //parent

char\* Nghc; //neighbor

char\* Mutc; //mutability

char\* Genc; //genus

char\* Spsc; //species size

char\* SpCc; //species centroid

char\* Spdc; //species density

char\* Spic; //species indivs (identities)

char\* Sptc; //speciation type (parent2 neighbor2)

char\* Expc; //experiment type

**class genus**  
 public:

genus();

~genus();

void NateSpeciation(population\* populous); //nearest and second nearest

void AdamSpeciation(population\* populous); //nearest only

void Centroids(population\* theIndivs);

void Densities(population\* theIndivs);

vector<species>::iterator beginOfTheGenus();

vector<species>::iterator endOfTheGenus();

int intSizeOfTheGenus() const;

vector<species> getTheGenus() const;

species getThisSpeciesOfTheGenus(int thisSpecies) const;

int getSpeciesPlaceInTheGenus(

vector<species>::iterator theyreHere);//?????????????

void replaceSpecies(int deezGuys, species daGuys);

void resetGenus();

private://used in NateSpeciation

void DiscoveredANewSpecies(species newSpp);

species combineSpecies(species these, species those);

void removeSpecies(

vector<species>::iterator away);

private:

vector<species> TheGenus;

**class species**  
 public:

species();

~species();

void addToSpecies(int broMontana);

void setCentroid(double cX, double cY);

void setDensity(double density);

void resetSpecies();

vector<int>::iterator beginOfTheSpecies();

vector<int>::iterator endOfTheSpecies();

int getThisIndivOfTheSpecies(int individ) const;

int intSizeOfTheSpecies() const;

vector<int> getTheSpecies() const;

double getCentroidX() const;

double getCentroidY() const;

double getDensity() const;

private:

double CentroidX;

double CentroidY;

double Density;

vector<int> TheSpecies;

**class population**  
 **class vector**

public:

population();

~population();

int capacityPop();

void reservePop(int buffpop);

//ALTER THE POPULATION COMPOSITION

void addToPopulation(indiv newIndiv);

vector<indiv>::iterator removeIndivFromThePopulation(vector<indiv>::iterator byeIndiv);

void resetThePopulation();

void countIndiv();

//SET INDIV INFORMATION AMONG THE POPULATION

void setNeighbors(landscape\* land);

void replaceIndiv(int disone, indiv\* indi);

indiv getThisIndivOfThePopulation(int thisIndiv);

indiv getNeighbor1OfThisIndiv(indiv\* div1);

indiv getNeighbor2OfThisIndiv(indiv\* div2);

//ITERATOR ACCESS

vector<indiv>::iterator beginOfThePopulation();

vector<indiv>::iterator endOfThePopulation();

//THE POPULATION ACCESS

vector<indiv> getThePopulation();

int intSizeOfThePopulation() const;

//CALCULATIONS

double meanMutability();

int uniqueMutability();

private:

vector<indiv> ThePopulation;

int ThePopulationSize;

//Death Dances

public:

void RussianRoulette(CRandomMersenne\* dice); //carries out random chance kills

void CliffJumpers(landscape\* cliffs); //carries out killing indiv's that go off the landscape

void NicheControl(CRandomMersenne\* R);//landscape\* resources);

void setOverPopulationLimit(double limit);

void setDeathChance(double unfortunate);

int getOPKillCount();

int getRRKillCount();

int getCJKillCount();

int getKillCount();

double getOverPopulationLimit();

double getDeathChance();

private:

void adjustIdentity(int rmvd, vector<indiv>::iterator fixIt);

/\*double getDefaultRange(landscape\* smallWorld);

vector<indiv> OverPopulated(int tempi, vector<indiv> findAgain,

vector<indiv>::iterator currIt);\*/

void NeighborsRestored(vector<indiv> redoThese, double dR);

bool checkIdentity(int id);

double OverPopulationLimit; //overpop = 0.25 in Matlab

double DeathChance; //death\_max = 0.70 in Matlab

int KilledOP;

int KilledRR;

int KilledCJ;

int Killed;

//CHECKS

public:

void goodNeighbors(vector<indiv>::iterator ref);

**class indiv**  
 public:

indiv();

~indiv();

void setBaby(int babeid, int par1, int par2,

double babeX, double babeY, double parMR);

void setBaby(int babeid, int par1,

double babeX, double babeY, double parMR);

//void setIndiv(int ident, double ex, double why, int cede, landscape \*carto);

void setIndiv(int ident, landscape \*carto, double randomX,

double randomY, double rmma);

//INDIV'S IDENTITY

public:

void setIdentity(int thisis); //set indiv identity

int getIdentity() const; //get indiv identity among it's generation

private:

int Identity; //indiv's identity

//PARENT IDENTITIES

public:

//set indiv's (type 1) parent

void setParent1Identity(int parent1id);

//set indiv's (type 2) parents

void setParent2Identity(int parent2id);

//get indiv's parent 1 identity

int getParent1Identity() const;

//get indiv's parent 2 identity

int getParent2Identity() const;

private:

//indiv's mutation rate determining parent identity

int Parent1Identity;

//indiv's other parent identity

int Parent2Identity;

//NEIGHBOR IDENTITIES AND DISTANCES TO INDIV

public:

//set and get Neighbor's identity

void setNeighbor1Identity(int neighbor1ident);

void setNeighbor2Identity(int neighbor2ident);

int getNeighbor1Identity() const;

int getNeighbor2Identity() const;

//set and get Neighbor's distance from indiv

void setNeighbor1Distance(double dist2Mate);

void setNeighbor2Distance(double dist2ImmSppEdge);

double getNeighbor1Distance() const;

double getNeighbor2Distance() const;

private:

//identity of indiv's nearest neighbor

int Neighbor1Identity;

//identity of indiv's second closest neighbor

int Neighbor2Identity;

double Neighbor1Distance;

double Neighbor2Distance;

//INDIV'S PHENOTYPE LOCATION

public:

//set indiv's phenotype x,y-coordinates - only for initialization of first generation

void setPhenotypeCoordinate(landscape \*land,

double randX, double randY);

double getPhenotypeX() const; //get indiv's phenotype x-coordinate

double getPhenotypeY() const; //get indiv's phenotype y-coordinate

void setPhenotypeX(double phenox);

void setPhenotypeY(double phenoy);

private:

double PhenotypeX; //indiv's phenotype x-coordinate

double PhenotypeY; //indiv's phenotype y-coordinate

//INDIV'S MUTABILITY (BIRTH NOISE)

public:

//void initMutability(double mutr);

void setMutability(double mutatRat); //landscape \*map); //set indiv's mutation rate - only for initialization of first generation

double getMutability() const; //get indiv's mutation rate

private:

double Mutability; //indiv's mutation rate

**class landscape**

public:

landscape();

landscape(int M, int N);

landscape(int M, int N, int num\_gen, int speed, int min, int max);

~landscape();

void setBasicMap(CRandomMersenne\* bRand);

void interp2(int x0, int xf, int y0, int yf);

int getMaxXOfTheLandscape() const;

int getMaxYOfTheLandscape() const;

int getNearestFitnessOfTheLandscape(double indivsX,

double indivsY) const;

double getValueOfTheLandscape(int ecks, int wai) const;

double getMaximumDistance() const;

private:

int X;

int Y;

int MIN;

int MAX;

double MaximumDistance;

double\* TheLandscape;

void setX(int x);

void setY(int y);

void setMaximumDistance();

void setMax(int max);

void setMin(int min);

void setRowForBasicMap(int m, CRandomMersenne\* baserand);

void setTheLandscape(int x, int y, double fit);

int getX() const;

int getY() const;

int getMin() const;

int getMax() const;

//LANDSCAPE SHIFTING

public:

void shiftTheLandscape(CRandomMersenne\* r, int g);

double getValueOfShiftedScape(int ex, int why) const;

int getShift() const;

int getShifted() const;

int getRate() const;

private:

int SHIFT;

int SHIFTED;

int RATE;

double\* ShiftedScape;

void setShiftedScape(int oldx, int oldy, double oldFit);

void setShift(int shift);

void setShifted(int shifted);

void setRate(int rate);

//STORE A CRANDOMMERSENNE FOR THE PROGRAM?

//public:

//void initializeRandom(int seed);

//void setRandom(CRandomMersenne sk);

//CRandomMersenne getRandom();

//private:

//CRandomMersenne SequenceKeeper;

public:

void setSaveLandCommands(char\* fitc, char\* slsc, char\* vlsc, char\* mlsc);

char\* getFitnessCommand() const;

char\* getShiftedCommand() const;

char\* getVaryingCommand() const;

bool checkLandCommand(char\* command) const;

void writeTheLandscape(int simu);

void writeShiftedScape(int simu);

private:

char\* Fitc;

char\* Slsc;

char\* Vlsc;

char\* Mlsc;

**class randomc**

**class general**

#iostream

#ctime

#cstdlib

#cmath

#istream

#fstream

#functional

#algorithm

#sstream

std::cout

std::cin

std::endl

std::ceil

std::ofstream

std::string

std::ostringstream

std::ifstream