Pseudocode for MAGE (specific)

/*Comments

In MAGE, Sim class is where starting population size N is created, individuals acquire mutations, individuals mate and pass mutations to offspring, N most fit offspring become the new population mating repeats for desired number of generations.

Comments end*/

```
Create Sim;

If args.length is greater than 0

        Try

                Read configuration file, store values for parameters within Sim object, s;

        Catch IOException ioe

                Call method out.println with "Unable to read from configuration file \";

        ENDTRY

        Call method Stats.init     // initiate the only random number generator of MAGE;
```

[Step 1: DominanceWeighted determines the fitness calculation for each individual]

```
        If s.isDominanceWeighted_ is true

                Read DominanceWeighted file;

                Set individual Fitness = Math.max(mFitness[j], pFitness[j])*maxCoeff + Math.min(mFitness[j], pFitness[j])*minCoeff +
(mFitness[j]+pFitness[j])/2*meanCoeff;

        Else
```

Read DominanceWeighted coefficient ‘w’;

Set individual fitness = (PaternalFitness + MaternalFitness)*w;

ENDIF

[Step 2: Introduce mutations into MAGE]

Read mutation file;

Store mutation file (startingBase, endingBase, probability);

[Step 3: Introduce codon table into MAGE]

Read CodonTable file;

Set optimalTriplet for each amino acid; // optimalTriplet stands for highest frequency codon for amino acid;

Set worstTriplet for each amino acid; // worstTriplet stands for least frequency codon for amino acid;

[Step4: Truncated Sequences]

If s.isTruncated is true

Read the ‘max_size’ from configuration file;

Read the ‘chromosome_filename’ from configuration file;

Truncate chromosome sequence based on max_size;

Else

Read the ‘chromosome_filename’ from configuration file;

Input chromosome sequence;

ENDIF

[Step 5: Blocks for the individual]

Read the ‘exon_filename’ from configuration file;

Get the gene’s name for the gene;

Get the starting position for each exon in the gene;

Get the ending position for each exon in the gene;

Get the phrase for each exon;

[Step 6: MRI for the individual]

Read the ‘MRI_filename’ from configuration file;

If it is GC rich in this region Then

Mark this MRI_ region as ‘1’;

Else

Mark this MRI_ region as ‘-1’;

ENDIF

[Step 7: Create individuals]

Create the packedBlocks based on starting and ending indexes for each exon;

Set blockNumber = 0;

While x < sequence size

While x< sequence size AND position x in the sequence is in exon Then

Set x = x +1;

Put this exon in the same block;

ENDWHILE

Truncated the sequence based on starting and ending indexes for each exon;

Store the information pb = PackedBlock(isSeqStored_, seq, blockNumber, startIdx, state);

Add the block (pb) into pc;

Set blockNumber = blockNumber + 1;

ENDWHILE *I don't understand this whole loop*

Create the first individual adam_ = Individual (pc, pc); //(pc, pc) one for maternal, one for paternal

[Step 8: Set Mutation rates for individual]

Try

Get crossoverMultiplier_ from configuration file;

Catch Exception e

Set crossoverMultiplier_ to 1.0f

ENDTRY

[Step 9: Introduce genetic distance into MAGE]

Read the genetic distance file from 'crossover_filename';

Store the data in a hashtable hm (baseposition, centiMorgan);

[Step 10: Establish fitness table for MAGE]

If it is the generation number is = 0 Then

Create a table;

Fill the table with fitness for each 'ATCG' nucleotide based on the probability and fitness value according to the selection coefficient curve;

ENDIF

[Step11: Run the MAGE]

If generation number = 0 Then

Create a whole population with same individuals adams_;

Else

Read the parameter from the backup file;

Create the Results.csv

Create the TmpResults.csv

For gen = (genNum_ + 1) to number of generations

If gen = 1 Then

Output (gen, mean_fitness_env_, polymorphic_sites, fixed_mutations);

ENDIF

ENDFOR *what does this for loop accomplish?*

```
        Call method sim.matingThreads( gen);    // sequences of individuals mutate and calculate the fitness, number of polymorphic sites, fixed mutations;

        Set Meanfitness_ = Meanfitness_ / population size;

        If remainder of gen divide outputFrequency_ = 0 Then

                Output the parameters (gen, mean_fitness_env_, polymorphic_sites, fixed_mutations);

        ENDIF

        If remainder of gen divide backupFrequency_ = 0 Then

                Create a consensus sequence;

                Create a backup file;

        ENDIF

        If gen = number of Gen Then

                Output the parameters (gen, mean_fitness_env_, polymorphic_sites, fixed_mutations);

                Remove TmpResults.csv;

        ENDIF

Else

        Call method System.out.println with "Usage: Gem <configfile.in>"
```

Step1: DominanceWeighted determines the fitness calculation for each individual

/*Comments

Dominance dominance filename has been read into Sim object, s, from config file;

The dominance method is going to deal with the fitness for each individual in the MAGE project. The MAGE provides two Dominance_methods: one is weighted the other one is file specific. Each individual contains the fitness for nucleotide from the paternal chromosome and maternal chromosome.

For weighted method, the fitness calculation for an individual is based on the weight that is given. For example, assumes weight is 0.5, then the fitness calculated will be: Individual fitness = (PaternalFitness + MaternalFitness)*0.5.

Another way is to provide the project with a dominance file. Below is an example for a simple dominance file:

frequency mean max min

0.875 0 1 0

0.125 0 0 1

According to the input file, the calculation for the individual fitness would be Math.max(mFitness[j], pFitness[j])*maxCoeff + Math.min(mFitness[j], pFitness[j])*minCoeff + (mFitness[j]+pFitness[j])/2*meanCoeff;

Comments end */

Try

Initialise result = config.get with "dominance_method"

If result equals to "weight"

Set isDominanceWeighted_ to true;

Else

```
            Set isDominanceWeighted_ to false;
        ENDIF
Catch Exception e
        Set isDominanceWeighted_ to false;
ENDTRY
Return isDominanceWeighted_;
If s.isDominanceWeighted is true
        Set dominanceFilename_ = config.get with "dominance_filename";
        Return dominanceFilename_;
        Try
                Create BufferedReader in = read dominanceFilename line by line;
                While in.ready
                        Set line = in.readLine
                        If character at 0 position for line is equal to '#'
                                Continue;
                        ENDIF
                        Initialise probability;
                        Initialise mean;
```

```
                    Initialize max;
                    Initialize min;
                    Create DomStruct dp (probability, mean, max, min);
                    Add dp into dominanceProbabilities_;
                ENDWHILE
            Catch IOException e
                Call method e.printStackTrace;
            ENDTRY
Else
            Try
                Set dominanceWeight_;
            Catch Exception e
                Set dominanceWeight_ to 0.0f;
            ENDTRY
            Return dominanceWeight_;
            Set dominanceWeight_ to dominanceWeight
ENDIF
```

Step2: Introduce mutations into MAGE

/*Comments

Mutation filename has been read into Sim object, s, from config file. Mutation filename contains the probabilities of different substitutions. Transitions are more common than transversions. Below is an example input:

A A 0

A T 0.039

A C 0.044

A G 0.147

Above it shows the mutation happen in A -> T, C, G and the corresponding probability. It is also possible input the triplet nucleotide mutation as an input file.

GGG GCG 0.00034041817131

CCC CGC 0.00034205000276

CTA CGA 0.00036670878907

TAG TCG 0.00036698076098

AGG ACG 0.00041104021005

CCT CGT 0.00042617997959

CTG CGG 0.00048302210832

Comments end*/

Set substitutionFilename_ = config.get"mutation_filename";

```
Return substitutionFilename_;

Create Vector mutationProbabilities_;

Try

        Initialise in = read “substitutionFilename_”;

        While in is ready

                Set line = in.readLine;

                Create array tokens stores line split based on “\t”;

                Initialise startingBase to position 0 in tokens;

                Initialize endingBase to position 1 in tokens;

                Initialize probability to position 2 in tokens;

                Initialise mp (startingBase, endingBase, probability);

                Call method mutationProbabilities_.add with mp;

        ENDWHILE

Catch IOException e

         Call method e.printStackTrace;

ENDTRY
```

Step3: Introduce codon table into MAGE

/* Comments

Each triplet nucleotide in the coding region stands for one amino acid. MAGE contains the frequency for each codon in the project. For example, TCT -> 15.2, TCA -> 12.2, TCG -> 4.4, TCC ->17.2code for the amino acid serine (S). TCC has the highest frequency 17.2 and thus is assigned as optimalTriplet. TCG has the lowest frequency 4.4 and thus is assigned as worstTriplet. MAGE would like to mutate more into the optimalTriplet based on its frequency.

Below is the format for the input file in MAGE for codon table:

TTT F 0.46 17.6 (714298) *TCT S 0.19* 15.2 (618711) *TAT Y 0.44* 12.2 (495699) *TGT C 0.46* 10.6 (430311)

The italicized letters are stored into MAGE.

Comments end*/

```
Set codonTableFilename_ = config.get "codon_table";

Return codonTableFilename_;

Initiate tmpCodonUsage as an empty hashtable;

Initiate the familyCharacters (amino acids) as a new vector;

Initiate the tripletFrequency as a hashtable;

Initiate the tripletFamilies_ as a hashtable;

Initiate optimalTriplet_ as a hashtable;

Initiate worstTriplet_ as a hashtable;

Try

        Set ctr = 0;
```

```
Scan the codonTableFilename_ into in;
While in is ready
        Set str = in.next;
        If remainder of ctr/5 = 0
                Set the ‘triplet’ = str;
        ENDIF
        If remainder of ctr/5 = 1
                Set ‘letter’ = str;
                If the tmpCodonUsage does not contain ‘letter’
                        Add this codon into tmpCodonUsage as the key;
                        Add this codon into familyCharacters;
                ENDIF
        ENDIF
        If remainder of ctr/5 = 2
                Set the ‘frequency’ = str;
                Add the ‘triplet’ as the value for key ‘letter’;
                Put the (triplet, frequency) as key and value into tripletFrequency;
        ENDIF
```

```
            Ctr = Ctr + 1;
        ENDWHILE
        Call method in.close;
Catch IOException e
        Call method e.printStackTrace;
ENDTRY
Done reading codon table from file;
Iterator<Character> it = familyCharacters.iterator ();
While it.hasNext
        Set familyCharacter = it.next;
        Put the values of the tmpCOdonUsage into the tripletIterator using the familyCharacter as the Key; [one F : TTT]
        Read all the values from the tripletIterator one by one;
        While tripletIterator.hasNext
            Set triplet = tripletIterator.next;
            Initiate a new Vector synonomousTriplets;
            Put the values of the tmpCodonUsage into the synonymousTripletIterator using the familyCharacter as the Key;
            While synonomousTripletIterator.hasNext
                Set synonomousTriplet = synonomousTripletIterator.next;
```

```
                    If character at position 0 for synonomousTriplet is equal to character at position 0 for triplet
                        Call method synonomousTriplets.add with synonomousTriplet;
                    ENDIF
                ENDWHILE
                Call method tripletFamilies_.put with triplet, synonomousTriplets; // {TTT=[TTT, TTC]}}    TTT is the key;
        ENDWHILE
ENDWHILE
// find optimal triplet for the codon
Iterator<String> tripletIterator = tripletFamilies_.keySet ().iterator ();
// iterate through all codons
While tripletIterator.hasNext
        Set triplet = tripletIterator.next;
        Put (Triplet, Triplet) into the optimalTriplet_;
        Iterate the Values of tripletFamilies for Key 'Triplet' into optimalTripletIterator;
        While optimalTripletIterator.hasNext
                Set optimalTriplet = optimalTripletIterator.next;
                If tripletFrequency_.get with optimalTriplet is greater than tripletFrequency_.get with optimalTriplet_.get triplet
                        Call method optimalTriplet_.put with triplet, optimalTriplet; //create the {TTT,TTC}  {TTG,TTC} TTC is the highest
frequency
```

```
                ENDIF
        ENDWHILE
ENDWHILE
Set tripletIterator = tripletFamilies_.keySet;
While tripletIterator.hasNext
        Set triplet = tripletIterator.next;
        Put (Triplet, Triplet) into the worstTriplet_;
        Iterate the Values of tripletFamilies for Key 'Triplet' into worstTripletIterator;
        While worstTripletIterator.hasNext
                Set worstTriplet = worstTripletIterator.next;
                If tripletFrequency_.get with worstTriplet is less than tripletFrequency_.get with worstTriplet_.get triplet
                        Call method worstTriplet_.put with triplet, worstTriplet;
                ENDIF
        ENDWHILE
ENDWHILE
```

Step4: Truncated sequences

/*Comments

Truncation option for simulation is set in config file. If truncation is used, only use the first maxSize nucleotides to make a chromosome. E.g. create a chromosome from the first 1000 nucleotides in the FASTA file truncation is useful for local debugging. We use a 27Mbp sequence for our simulation which takes a lot of memory. Truncating the sequence is useful for debugging locally, where not enough memory may be available to use the entire sequence. Creator now also accepts number of environments as argument.

Comments end*/

Set default value truncated_ = false;

Try

    Set truncatedSize_ = config.get ("max_size");

    Set truncated_ = true;

Catch Exception e

    Set truncatedSize_ to 2147483647;

ENDTRY

Return truncated_;

If truncated_ is true

    Set the 'max_size' = mazSize; //max_size: the input setting for the length of sequence

    Try

        Create BufferedReader in = read 'chromosome filename' line one by one;

        Set counter = 0;

```
            While in.ready
                Set line = in.readLine;
                If the first character of line is not equal to '>'
                    If counter + length of Line > maxSize_
                        sbSeq = sbSeq join with the sequence (maxSize_ - counter);
                        break;
                    ENDIF
                    Call method sbSeq.append with line;
                    Counter = counter + length of Line;
                    Set the chrSize_ = maxSize;
                ENDIF
            ENDWHILE
            Call method in.close;
            Set strChr_ = sbSeq.toString;
        Catch IOException ioe
            Call method ioe.printStackTrace;
        ENDTRY
Else
```

Set ‘max_size’ = the length of the sequence file;

Do the same thing as the if truncated = true

ENDIF

Step5: Blocks for the individual

/*Comments

Reads exon positions from a .txt file. Exon positions are used to determine starting and ending position of each gene. The entire chromosome contig sequence is broken into locus blocks. The blocks are generally alternating intergenic-genic-intergenic-genic etc.

For the codon region, phase 0 means that the exon begins with a complete codon. Phase 1 means that the first two bases (5-prime) of the exon; constitute the second and third bases of an interrupted codon. Phase 2 means the first base of the exon consists of the last base of an interrupted codon.

Below is the example of input file for the exon:

It contains contig information, plus/minus strand, starting position and ending position, genes name and phrase of exons.

NT_011512	plus	644421..644941	A26B3_ex1	0
NT_011512	plus	649315..649429	A26B3_ex2	2
NT_011512	plus	649589..649762	A26B3_ex3	0
NT_011512	plus	652541..652647	A26B3_ex4	0
NT_011512	plus	654350..654487	A26B3_ex5	2
NT_011512	plus	656997..657067	A26B3_ex6	2
NT_011512	plus	662574..662644	A26B3_ex7	1
NT_011512	plus	664048..664092	A26B3_ex8	0
NT_011512	plus	665133..665299	A26B3_ex9	0
NT_011512	plus	673707..673830	A26B3_ex10	2
NT_011512	plus	675537..675758	A26B3_ex11	0

Comments end*/

Initiate a hashmap as genesHash_;

Create an array frame_ with capacity [chrSize];

Fill up the frame_ with ‘-1’;

Create an array isExonic_ fill up it with ‘false’;

Try

Initialise in = read the exon file line one by one;

Initialise intStrand = 0;

While in.ready

Set line to in.readLine;

Split ‘In’ based on “\t” and stored in the array tokens;

Split tokens[2] based on “..” and stored in the array positionTokens;

Split tokens[3] based on “-“ and stored in the array geneTokens;

Split geneTokens[1] based on “ex” and stored in the array exonTokens;

Set exonNum = exonTokens[1];

Set geneName = geneTokens[0];

If start position is greater than size OR end position is greater than size

break;

ENDIF

Set phase = tokens[4];

Set strStrand = tokens[1];

```
If strStrand is equals to "plus"
        Set intStrand = 0;
        Set idxStart = starting position;
        Set idxEnd = ending position;
        Truncated the chrSeq based on (starting, ending position);
        Fill the array Frame_[] in according positions with the exon phrase;
        For i=idxStart to idxEnd; i = i +1
            If frame_[i] = 0 AND i <= idxEnd-2
                Set thisTriplet = frame_[i]frame_[i+1]frame_[i+2];
                Fill the array codons_ in position [i], [i+1], [i+2] with thisTriplet;
            ENDIF
            If i = idxStart AND phase = 1
                Get the first nucleotide from the genesHash_;
                Set thisTriplet = frame_[i]frame_[i+1]frame_[i+2];
                Fill the array codons_ in position [i], [i+1],[i+2] with thisTriplet;
            ENDIF
            If i = idxStart AND phase = 2
                Get the first two nucleotides from the genesHash_;
```

```
                    thisTriplet = frame_[i]frame_[i+1]frame_[i+2];
                    Fill the array codons_ in its position [i], [i+1], [i+2] with thisTriplet;
                ENDIF
        Else
            Set intStrand = 1;
            Reverse the sequence;
            Treat the sequence as the above;        // Do the same thing as above for the complementary chromosome
        ENDIF
        If the geneName in the genesHash_ is = null
            Put the genesHash_ as (geneName, g);
        ENDIF
        For I = exon start position to exon end position; i = i +1
            Set isExonic_[i] = true;
            Set numExonicPositions_ = numExonicPositions_ + 1;
        ENDFOR
        Add the current Exon to the genesHah_ with the same name;
    ENDWHILE
Catch Exception e
```

```
        Call method e.printStackTrace;

ENDTRY
```

Step6: MRI for the individual

/*Comments

The MRI file contains locations of G+C rich regions and G+C poor regions;

The MRI example is as follows:

47393 47495 GC_poor NT_011512

50596 50762 GC_poor NT_011512

51432 51533 GC_poor NT_011512

53569 53670 GC_poor NT_011512

53699 53801 GC_poor NT_011512

66759 66875 GC_poor NT_011512

71344 71445 GC_rich NT_011512

71600 71701 GC_rich NT_011512

71852 71971 GC_rich NT_011512

Comments end*/

Set MRI_ = new byte[size]; // size = number of letters in chromosome contig sequence;

Try

    Create BufferedReader in = read MRI File line by line;

    While in is ready

```
                Set line = in.readLine ();
                Split line based on “\t” and put them in array tokens;
                Set start = tokens [0];
                Set end = tokens [1];
                If end is less than size
                        If tokens [3] is equals to "GC_rich"        // mark all positions that are GC rich +1
                                For x = start to end; x = x + 1
                                        MRI_[x] = 1;
                        Else
                                For x = start to end; x = x +1      // mark GC poor positions -1
                                        MRI_[x] = -1;
                                ENDFOR
                        ENDIF               // remaining positions in MRI_ remain 0
                ENDIF
 Catch IOException e
        Call method e.printStackTrace;
ENDTRY
```

Step7: Create individuals

/*Comments

Once the information about the position of CDS/non-CDS regions is processed a single (static) Individual object is constructed within the Creator class. Purpose of Creator class is to assemble the first individual: "adam". This individual contains two copies of the contig specified in the config file. The Creator class is responsible for dividing the contig sequence into multiple blocks (loci), each is called packedBlock. These packedBlocks are stored in the internal arrays of packedChromosome objects. The first individual thus has two packedChromosomes, either from maternal or paternal, each containing many packedBlocks.

Comments end*/

```
Try
        Set maxBlockSize_  = config.get("max_block_size");
Catch Exception e
        Set maxBlockSize_ = 10000000;
ENDTRY
Return maxBlockSize_;
Set x = 0;          startIdx = 0;       endIdx = 0;         blockNumber = 0;
While x < chrSize_
        Set Offset = 0;
        Set intergenicSize = 0;
        While x < chrSize_ AND isCDS_[x] = state          //state equal to false in default;
                Set posBlock_[x] = blockNumber;           //{1111111} stand that they are all in the same block:
```

```
        Set posOffset_[x] = offset;
        Set offset = offset + 1;
        Set x = x + 1;
        Set intergenicSize = intergenicSize + 1;
        If state = false AND intergenicSize = maxBlockSize
            break;                          //maxBlockSize's default value is 10000000;
        ENDIF
ENDWHILE
Set startIdx = endIdx;
Set endIdx = x;
Set seq = get the sequence from strChr_ based on (startIdx, endIdx);
Create a PackedBlock pb (isSeqStored_, seq, blockNumber, startIdx, state);
Input the startIdx into pb;
Input the endIdx into pb;
Add pb into pc as a block;
Set blockNumber = blockNumber + 1;
If x < chrSize_
        Set state = isCDS_[x];          //flip state between true and false;
```

ENDIF

ENDWHILE

Set numBlocks_ = blockNumber;

Input blockNumber into pc;

System.out.println (“Number of blocks/Number of genes/Number of exonic”);

Create the first individual Adam_ with maternal and paternal blocks (pcM,pcP);

Step8: Set Mutation rates for individual

/*Comments

Set the mutation rate for MAGE. Default is 1.

The simulation uses a genetic map obtained from the HapMap consortium to simulate crossover. The mean number of crossovers/bp is somewhat low and therefore deleterious mutations are removed less efficiently than what is possible with a higher rate of crossover. Therefore, a "crossover multiplier" allows one to multiply the genetic distances of the map, in cM, by a constant default = 1.

Comments end*/

```
Try
        Set crossoverMultiplier_ = config.get ("crossover_multiplier");
Catch Exception e
        Set crossoverMultiplier_ = 1.0f;
ENDTRY
Return crossoverMultiplier_;
```

Step9: Introduce genetic distance into MAGE

/*Comments

static init method for the class engine. Crossover reads the genetic map data giving the genetic distance in cM of many sites across the contig under evaluation.

The crossover multiplier file contains two kinds of information. One is the position of the basepair and the other is the genetic maps. In MAGE, if the position of the last nucleotide is not equal to the last nucleotide in the crossover multiplier that provided, it will correct it as shown as the 'realCmdiff' shown in the pseudo code below.

position Genetic_Map(cM)

1 0

27781 0.00118697

28901 0.00147879

32409 0.002526473

34534 0.00334814

34593 0.003367664

54657 0.008971441

54710 0.008985619

56220 0.009373615

56432 0.009427468

59684 0.010248303

60390 0.011043721

Comments end*/

Try

Set linesRead = 0;

Set ‘in’ = Read the geneticMap file line one by one;

While ‘in’ is ready

Set Line = current ‘in’ contents;

Set linesRead = linesRead + 1;

If lineRead = 1

Continue; //Skip the first line for header information;

ENDIF

Split Line by “\t” and put them into array tokens;

Set bp = tokens[0];

Set cM = tokens[1];

Add them into the hashtable hm (bp, cM);

If bp > chromosome size

Break;

ENDIF

ENDWHILE

```
        Put the bp into an array bpPosition_;

        Put the cM into another array cmDistance_;

        //calculate the last position for the genetic distances;

        Set realCmdiff = (laseBpPosition – realEnd)*(cmDistance_[lastIdx]-cmDistance_[lastIdx-1])/(bpPosition_[lastIdx]-bpPosition_[lastIdx-1]);

        Set cmDistance_[lastIdx] = cmDistance_[lastIdx] – realCmDiff;

        Set maxCm_ = cmDistance_[lastIdx];

        Set bpPosition_[lastIdx] = realEnd;
Catch  exception e
        Call method e.printStackTrace;
ENDTRY
```

Step10: Establish fitness table for MAGE

/*Comments

FitnessCalc class contains a matrix of scalar selection coefficients (s-values) for each possible base substitution in the entire contig sequence synonymous mutations (ie: A->A at position 191289) have 0 for s.... The remaining s-values for the matrix are built from random values drawn from two probability distributions: one for CDS regions and one for nonCDS regions. Additionally, an MRI constant may be added to these s vales depending on whether the bp position is inside or outside a GC-rich MRI region See Sim.java for more information.

Four arrays, length of each array will be equal to the length of the sequence used for the chromosome in each Individual. Each array contains selection coefficients for when a base at the given index is mutated to A, C, T or G. For example if Afits_[127254] = -1.21, when the base at index 127254 is mutated to "A" there will be a decrease in fitness of -1.21 for the allele at this position.

Example for the fitness file name for the coefficient curve:

[FitnessValue Frequencies]

-2.499161094 0.000797885

-2.303533611 0.003164303

-2.107906127 0.010767973

-1.912278644 0.031441838

-1.71665116 0.078777083

-1.521023676 0.169359619

-1.325396193 0.312419293

-1.129768709 0.494519964

-0.934141225 0.671657421

-0.738513742 0.782762141

Comments end*/

If this is the 0 generation

Initiate the FitnessCalc class;

System print out (“Initializing fitness table [“);

//The first step is to read the probability

Try

Set in = read the fitness file line one by one;// the fitness file contains the fitness values and their frequencies according to the selection coefficient curve;

Set lineNum = 0;

While ‘in’ is ready

Set ‘Line’ = current line;

Split ‘Line’ based by “\t” and store them in the array tokens;

Set Fitness2 = tokens[0];

Add the Fitness2 into the fitnessVector;

Set Frequency = tokens[1];

Add the frequency into the frequencyVector;

If lineNum is not = 0

Set frequency1 = the (lineNum-1)th element at frequencyVector;

```
                    Set fitness1 = the (lineNum-1)th element at fitnessVector;
                    Set areaUnderCurve = minimal of (frequency1, frequency2) *absolute value of (fitness2-fitness1);
                    Set areaUnderCurve = areaUnderCurve + absolute value of (.5f*(frequency2-frequency1)*(fitness2-fitness1));
                    Add areaUnderCurve into areaUnderCurveVector;
                    Set totalAreaUnderCurve = totalAreaUnderCurve + areaUnderCurve;
                ENDIF
                Set lineNum = lineNum +1;
            ENDWHILE
            For i =0 to size of areaUnderCurveVector; i = i + 1
                Set fitness1 = element i at fitnessVector;
                Set fitness2 = element (i+1) at fitnessVector;
                Set probability = element i at areaUnderCurveVector divide totalAreaUnderCurve;
                Add (fitness1, fitness2, probability) into the vector fitnessProbabilities_;
            ENDFOR
        Catch IOException e
            Call method e.printStackTrace
        ENDTRY
// put the CDSfitness into the vector      The same way as the fitness and put them into a vector CDSfitnessProbabilities;
```

```
//readNumNonIdentical
        Try
                Set ‘in’ = read the readNumNonIdentical file line one by one;
                While ‘in’ is ready
                        Set ‘Line’ = the current line ‘in’;
                        Split ‘Line’ based on the “\t” and Put them into an array tokens;
                        Set Environment = tokens[0];
                        Set Probability = tokens[1];
                        Create a ProbabilityStruct fp (Environment, Probability);
                        Add fp into a vector numNonIdenticalProbabilies;
                ENDWHILE
        Catch IOException e
                Call method e.printStackTrace;
        ENDTRY
//readProbabilityNonIdentical in the same way as above
        Add them into a vector NonIdenticalProbabilities;
        Initiate the vector Afits_;
        Initiate the vector Cfits_;
```

```
        Initiate the vector Tfits_;

        Initiate the vector Gfits_;
// create the fitness table;
        Set numFits = length of sequence;

        Set fillerVal = 1234567.0;

        For i=0 to numEnv; i = i + 1

                Fill the array for Afits_ in Env (i) with a default value fillerVal;

                Fill the array for Cfits_ in Env (i) with a default value fillerVal;

                Fill the array for Tfits_ in Env (i) with a default value fillerVal;

                Fill the array for Gfits_ in Env (i) with a default value fillerVal;

        ENDFOR

        For x = 0 to numFits; x = x + 1

                Set Character = Uppercase of x position at sequence;

                If (remainder of x /(numFits/10)) = 0

                        Print out ".";

                ENDIF

                If Character is in exon

                        If Character's position at exon = 2
```

```
Set Codon = codon at the position x;
Set familyMut =values of getTripletFamilies by using Codon as key;
Read familyMut line one by one;
While familyMut is ready
        Set synonymousCodon = current contents of familyMut;
        Set mutation = third character for synonymousCodon;
        If mutation = 'A'
                Add 0f for environment 0 position x for Afits_ ;
        ENDIF
        If mutation = 'C'
                Add 0f for environment 0 position x for Cfits_ ;
        ENDIF
        If mutation = 'T'
                Add 0f for environment 0 position x for Tfits_ ;
        ENDIF
        If mutation = 'G'
                Add 0f for environment 0 position x for Gfits_ ;
        ENDIF
```

```
                ENDWHILE
                If Afits_'s element at position x for environment 0 = fillerVal
                        Set Afits_.elementAt (0) [x] = arrange an exon fitness;
                ENDIF
                If Cfits_'s element at position x for environment 0 = fillerVal
                        Set Cfits_.elementAt (0) [x] = arrange an exon fitness;
                ENDIF
                If Tfits_'s element at position x for environment 0 = fillerVal
                        Set Tfits_.elementAt (0) [x] = arrange an exon fitness;
                ENDIF
                If Gfits_'s element at position x for environment 0 = fillerVal
                        Set Gfits_.elementAt (0) [x] = arrange an exon fitness;
                ENDIF
Else
        If Afits_'s elment at position x for environment 0 = fillerVal
                Set Afits_.elementAt (0) [x] = arrange a Non-exon fitness;
        ENDIF
        If Cfits_'s elment at position x for environment 0 = fillerVal
```

```
                Set Cfits_.elementAt (0) [x] = arrange a Non-exon fitness;
        ENDIF
        If Tfits_'s elment at position x for environment 0 = fillerVal
                Set Tfits_.elementAt (0) [x] = arrange a Non-exon fitness;
        ENDIF
        If Gfits_'s elment at position x for environment 0 = fillerVal
                Set Gfits_.elementAt (0) [x] = arrange a Non-exon fitness;
        ENDIF
ENDIF
For i=1 to numEnv; i = i + 1
        Arrange enough set of Afits_ for different environment for the same value;
        Arrange enough set of Afits_ for different environment for the same value;
        Arrange enough set of Afits_ for different environment for the same value;
        Arrange enough set of Afits_ for different environment for the same value;
ENDFOR
For env=0 to numEnv_; env = env + 1
        Write the FitnessTable for "Position \t A \t C \t T \t G" for different environments;
ENDFOR
```

```
            For x =0 to numFits; x = x +1
                Set randDoub = random number generator generate a number;
                If randDoub <= percentIdenticalFitness
                //identical fitness values between different environments account for 80% =  percentIdenticalFitness
                    Set isSameFit = true;
                Else
                    Set isSameFit = false;
                ENDIF
                If it is the same, then fills the fitnessTable as the previous one that creates the table;
                If it is not the same, then run the same procedure as the previous one;
                // the above two ifs do not have any differences, the purpose is to fill up the table;
            ENDFOR
            For env=0 to numEnv_; env = env + 1
                Write the fitnessTable for different environment with Afits_ / Cfits_ / Tfits_ / Gfits_;
            ENDFOR
        ENDFOR
ENDIF
```

Step11: Run the Simulation

```
If generation number (genNum) is equal to 0      // Create vector to hold population
        For pi = 0 to pSize_; pi = pi + 1
                Add (pi, Individual (adam) into population_ vector;
        ENDFOR
Else
        Try
                Read from a backup file;
        Catch Exception e
                Call methods e.printStackTrace ();
        ENDTRY
ENDIF
Create the Results.csv;
Create the TmpResults.csv;
/*Main loop of the simulation FOR EACH generation: get number of mating pairs; FOR EACH mating pair: select a male and a female from population_ Vector; generate number of offspring using Poisson process; FOR EACH offspring generate a gamete for male; generate a gamete for female; create a new Individual: maternal chromosome = female gamete, paternal chromosome = male gamete; add new Individual to nextGeneration_ Vector; contents of nextGeneration_ comprise new population_ Vector*/
For gen = (genNum +1) to nGen_; gen = gen + 1
        If gen is equal to 1        // set up title of columns for .csv output
```

```
        Write the “gen” into tmpOutput;
        Write the “mean_fitness_env” into tmpOutput;
        If isSeqStored_ is true
                Write “\n” into tmpOutput;
                Write “0” into tmpOutput;
                For j = 0 to numEnv_; j = j + 1
                        Write ",0" into tmpOutput;
                ENDFOR
                Write “\n” into tmpOutput;
        Else
                Write the “polymorphic_sites, fixed_mutations, \n” into tmpOutput;
                Write the (“,0”) into tmpOutput;
                Write the “,0,0\n” into tmpOutput;
        ENDIF
ENDIF
If matingType is equal to PAIRING_WITHIN_ENVIRONMENTS
// migration randomly reassign a percentage of individuals to new environments; if environment is = 1, then skip this step;
        For j =0 to (pSize_ * migrationRate_); j = j + 1
```

```
                Set randInd = randomly pick up individual;
                Set randEnv = randomly pick up the environment;
                Set the environment (randEnv) for the individual (randInd);
        ENDFOR
ENDIF
Set sizeNextGeneration = 0;
While sizeNextGeneration is less than (pSize_ * meanNumOffspringPerMatingPair_)      //create enough offsprings
        Set mIdx = 0;
        Set pIdx = 0;
        Set environment = randon number generator generate a environment;
        If matingType_ is equal to PAIRING_WITHIN_ENVIRONMENTS
                Set mIdx = sample from the population vector until a female is found;
                Set pIdx = sample from the population vector until a male is found;
        Else if matingType_ is equal to PAIRING_BETWEEN_ENVIRONMENTS
                Set mIdx = sample from the population vector until a female is found;
                Set pIdx = sample from the population vector until a male is found;
        ENDIF
        If fertilityFitnessDependence_ is true AND gen_ is larger than 1
```

```
        Set numOffspringThisMating = get (fitnessPercentile female + fitnessPercentile male)*randomNumberGenerate;
Else
        Set numOffspringThisMating = randomNumberGenerate;
ENDIF
For x = 0 to numOffspringThisMating; ++x          //multiple offsprings from same pairs get their mutation and fitness
        Set Individual father = population_ (pIdx);
        Set Individual mother = population_(mIdx);
        Set mGamete = mother get the gamete;
        Set pGamete = father get the gamete;
        Create a new individual = (mGamete, pGamete, environment);
        If gen = mutationGen_
                Return;          //default set is mutationGen_ = -1
        ENDIF
        Set mutationGen_ = gen;
        For x = 0 to numMutations; x = x + 1
                Randomly generate an intel (0 or 1);
                If is 0, maternal_ mutate; break;
                If is 1, paternal_ mutate;
```

```
ENDFOR
Create Vector mBlocks = maternal_'s blocks;
Create Vector pBlocks = paternal_'s blocks;
For x =0 to (number of mBlocks); x = x + 1
        Set mBlock = mBlocks for position x;
        Set pBlock = pBlocks for position x;
        Set mFitness = fitness of mBlock;
        Set pFitness = fitness of pBlock;
        Set maxCoeff = mBlock get max Coefficient;
        Set minCoeff = mBlock get min Coefficient;
        Set meanCoeff = mBlock get mean Coefficient;
        For j =0 to numEnv; j = j + 1
                If DominanceWeighted is true    //Calculate the fitness based on DominanceWeighted;
                        If mBlock is CDS
                                Set fitness_[j] = fitness_[j] + abs|mFitness[j] – pFitness[j]|*DominanceWeight + minimal
of (mFitness[j],pFitness[j]);
                        Else
                                Set Fitness_[j] = Fitness_[j]+ (mFitness[j]+pFitness[j])/2;
                        ENDIF
```

```
                    Else

                        Fitness_[j] = Fitness_[j]+ max of (mFitness[j], pFitness[j])*maxCoeff + min(mFitness[j],
                pFitness[j])*minCoeff + (mFitness[j]+pFitness[j])/2*meanCoeff;

                    ENDIF

                ENDFOR

            ENDFOR

        ENDFOR

        If matingType is equal to PAIRING_WITHIN_ENVIRONMENTS

            Add offspring into nextGeneration_;

        Else if matingType_ is equal to PAIRING_BETWEEN_ENVIRONMENTS

            For i=0 to numEnv; i = i + 1

                Add offspring into nextGeneration_ in this environment;

            ENDFOR

        ENDIF

ENDWHILE

If matingType is equal to PAIRING_WITHIN_ENVIRONMENTS

        Set sizeNextGeneration = 0;

        For i=0 to numEnv; i = i + 1

            Set sizeNextGeneration (number of individuals) = number of individuals at environment (i) + sizeNextGeneration;
```

```
            ENDFOR
        Else if matingType_ is equal to PAIRING_BETWEEN_ENVIRONMENTS
            Set sizeNextGeneration = size for nextGeneration at environment 0;
        ENDIF
// sort population using the IndividualComparator class;  Individuals are sorted by fitness in descending order. Therefore, the Individuals at the head of the array are the most fit sorting differs depending on the type of mating and environment selection;
        If matingType is equal to PAIRING_WITHIN_ENVIRONMENTS      //Create the next population_
            Set Env = 0;
            Set Individual tmpInd;
            While size of population_ is less than pSize
// selects the next population by picking an environment at random then taking the most fit individual from that environment placing them in the new population repeats until new population is the correct size
                    If number of individuals for nextGeneration at this environment is equal to 0
                        Continue;
                    ENDIF
                    Set 'It' = individuals at nextGeneration for environment (env);
                    Set tmpId = current individual for 'it';
                    Add tmpId into population_;
                    Remove tmpId from nextGeneration;
```

Set Env = Env + 1

If remainder of env/numEnv_ = 0

env = 0;

ENDIF

ENDWHILE

Else if matingType_ is equal to PAIRING_BETWEEN_ENVIRONMENTS

// in this case the sorting and selection is similar to above except individuals self select to be in the environment where they are most fit so each sorted population for a particular environment contains all individuals;

Set Env =0;

Set Individual tmpInd;

While size for population_ is less than pSize_

Set 'It' = individual in nextGeneration_ at env;

Set tmpId = current individual in 'it';

Set env as the environment for tmpId;

Add tmpId into the population_;

For j =0 to numEnv_; j = j + 1

Remove tmpId from nextGeneration at environment (j);

ENDFOR

Env = Env +1

```
                    If remainder of env/numEnv = 0
                        Set env =0;
                    ENDIF
                ENDWHILE
            ENDIF
            Set fitnessOrder = pSize;
// Individual's order statistic for fitness, order k = kth smallest value e.g. for N = 100 highest fitness -> Order = 100; lowest fitness -> Order = 1. For example, for a population of size 100 highest order statistic for fitness is 100, individual is at index x=0 individual has 99.5% percentile rank for fitness; lowest order statistic is 1, individual is at index x=99 individual has 0.5% percentile rank for fitness
            If fertilityFitnessDependence has been set
                For x = 0 to pSize_; x = x + 1
                    Set fitness percentile for individual x = (fitnessOrder – 0.5)/pSize;
                    Set fitnessOrder = fitnessOrder -1;
                ENDFOR
            ENDIF
            Initiate an array meanFitness;
            Initiate an array envSize with capacity [numEnv];
            Fill up the array envSize with 0;
            For pi = 0 to pSize; ++pi
```

```
        Set indEnv = environment for individual pi;
        Set meanFitness for environment [indEnv] = meanFitness for environment [indEnv] + fitness for individual pi;
        Set envSize[indEnv] = envSize[indEnv] + 1;
ENDFOR
For j=0 to numEnv; j = j + 1
        Set meanFitness[j] = meanFitness[j]/envSize[j];
ENDFOR
If isSeqStored is false
        If remainder of gen/fixationFreq is equal to 0
                Set population_ = FitnessCalc.fixMutations(population_);
        ENDIF
ENDIF
If remainder of gen/outputFrequency_ is equal to 0
         Try
                Output (">Consensus sequence for generation " + gen + "\n");
                Output parameters into backup file;
        Catch IOException e
                Call method e.printStackTrace();
```

```
            ENDTRY
        ENDIF
        If gen is equal to nGen_
            Write the parameters into TmpResults.csv;
            Write the parameters in TmpResults.csv into Results.csv;
            Remove TmpResults.csv
        ENDIF
ENDFOR
```