

```
1 #include "PCH.h"
2 std::mt19937 mt{ std::random_device{}() };
3
4 GA::GA(int chromosomeDim, int populationDim, double crossoverProb, int      ↗
    randomSelectionChance, int maxGenerations, int numPrelimRuns, int      ↗
    maxPrelimGenerations, double mutationProb, int crossoverType, bool    ↗
    computeStatistics)
5 {
6     this->randomSelectionChance = randomSelectionChance;
7     this->crossoverType = crossoverType;
8     this->chromosomeDim = chromosomeDim;
9     this->populationDim = populationDim;
10    this->computeStatistics = computeStatistics;
11
12    this->chromosomes = std::vector<Chromosome*>(populationDim);
13    this->chromNextGen = std::vector<Chromosome*>(populationDim);
14    this->prelimChrom = std::vector<Chromosome*>(populationDim);
15    this->genAvgDeviation = std::vector<double>(maxGenerations);
16    this->genAvgFitness = std::vector<double>(maxGenerations);
17
18    this->crossoverProb = crossoverProb;
19    this->maxGenerations = maxGenerations;
20    this->numPrelimRuns = numPrelimRuns;
21    this->maxPrelimGenerations = maxPrelimGenerations;
22    this->mutationProb = mutationProb;
23 }
24
25 double GA::getAvgDeviation(int iGeneration)
26 {
27     return (this->genAvgDeviation[iGeneration]);
28 }
29
30 double GA::getAvgFitness(int iGeneration)
31 {
32     return (this->genAvgFitness[iGeneration]);
33 }
34
35 double GA::getMutationProb()
36 {
37     return mutationProb;
38 }
39
40 int GA::getMaxGenerations()
41 {
42     return maxGenerations;
43 }
44
45 int GA::getNumPrelimRuns()
46 {
47     return numPrelimRuns;
48 }
49
50 int GA::getMaxPrelimGenerations()
51 {
52     return maxPrelimGenerations;
53 }
```

```
54
55 int GA::getRandomSelectionChance()
56 {
57     return randomSelectionChance;
58 }
59
60 double GA::getCrossoverProb()
61 {
62     return crossoverProb;
63 }
64
65 int GA::getChromosomeDim()
66 {
67     return chromosomeDim;
68 }
69
70 int GA::getPopulationDim()
71 {
72     return populationDim;
73 }
74
75 int GA::getCrossoverType()
76 {
77     return crossoverType;
78 }
79
80 bool GA::getComputeStatistics()
81 {
82     return computeStatistics;
83 }
84
85 Chromosome * GA::getFittestChromosome()
86 {
87     return (this->chromosomes[bestFitnessChromIndex]);
88 }
89
90 double GA::getFittestChromosomesFitness()
91 {
92     return (this->chromosomes[bestFitnessChromIndex]->fitness);
93 }
94
95 int GA::getRandom(int upperBound)
96 {
97     std::uniform_int_distribution<int> dist(0, upperBound);
98     return dist(mt);
99 }
100
101 double GA::getRandom(double upperBound)
102 {
103     std::uniform_real_distribution<double> dist(0.0, upperBound);
104     return dist(mt);
105 }
106
107
108
109 int GA::evolve()
```

```
110 {
111     int iGen;
112     int iPrelimChrom, iPrelimChromToUsePerRun;
113     std::cout << "CPGene start" << std::endl;
114
115     if (numPrelimRuns > 0)
116     {
117         iPrelimChrom = 0;
118         iPrelimChromToUsePerRun = populationDim / numPrelimRuns;
119         for (int iPrelimRuns = 1; iPrelimRuns <= numPrelimRuns; iPrelimRuns++)
120         {
121             iGen = 0;
122             initPopulation();
123             while (iGen < maxPrelimGenerations)
124             {
125                 std::cout << iPrelimRuns << " z " << numPrelimRuns << "      ↗
126                     Uruchomien wstepnych. Pokolenie: " << (iGen + 1) << " z " << ↗
127                     maxPrelimGenerations << std::endl;
128
129                 computeFitnessRankings();
130                 doGeneticMating();
131                 copyNextGenToThisGen();
132
133                 if (computeStatistics == true)
134                 {
135                     this->genAvgDeviation[iGen] = getAvgDeviationAmongChroms ↗
136                     ();
137                     this->genAvgFitness[iGen] = getAvgFitness();
138                 }
139                 iGen++;
140             }
141             computeFitnessRankings();
142             int iNumPrelimSaved = 0;
143             for (int i = 0; i < populationDim && iNumPrelimSaved < ↗
144                 iPrelimChromToUsePerRun; i++)
145             {
146                 if (this->chromosomes[i]->fitnessRank >= populationDim - ↗
147                     iPrelimChromToUsePerRun)
148                 {
149                     this->prelimChrom[iPrelimChrom + iNumPrelimSaved]- ↗
150                     >copyChromGenes(this->chromosomes[i]);
151                     iNumPrelimSaved++;
152                 }
153             }
154             iPrelimChrom += iNumPrelimSaved;
155         }
156         for (int i = 0; i < iPrelimChrom; i++)
157         {
158             this->chromosomes[i]->copyChromGenes(this->prelimChrom[i]);
159         }
160         std::cout << "Populacja wstepna (po generacji wstepnej):" << ↗
161         std::endl;
162     }
163     else
164     {
165         std::cout << "Populacja wstepna (bez generacji wstepnej):" << ↗
```

```

        std::endl;
159     }
160
161     addChromosomesToLog(0, 10);
162
163     iGen = 0;
164     while (iGen < maxGenerations)
165     {
166         computeFitnessRankings();
167         doGeneticMating();
168         copyNextGenToThisGen();
169         //std::cout << "Gen: " << iGen << std::endl;
170
171         if (computeStatistics == true)
172         {
173             this->genAvgDeviation[iGen] = getAvgDeviationAmongChroms();
174             this->genAvgFitness[iGen] = getAvgFitness();
175         }
176
177         iGen++;
178     }
179     std::cout << "Gen: " << (iGen) << " Avg Fitness = " << this->genAvgFitness &
180     [iGen-1] << " Avg DEV = " << this->genAvgDeviation[iGen-1] << std::endl;
181     addChromosomesToLog(iGen, 10);
182     computeFitnessRankings();
183     std::cout << "Najlepszy chromosom: ";
184     std::cout << this->chromosomes[this->bestFitnessChromIndex]->getGenesAsStr &
185     () << " Fitness = " << std::fixed << std::setprecision(8) << this->
186     >chromosomes[this->bestFitnessChromIndex]->fitness << std::endl;
187
188     std::cout << "CPGene stop" << std::endl;
189     return (iGen);
190 }
191
192 double GA::getAvgFitness()
193 {
194     double rSumFitness = 0.0;
195
196     for (int i = 0; i < populationDim; i++)
197     {
198         rSumFitness += this->chromosomes[i]->fitness;
199     }
200     return (rSumFitness / populationDim);
201 }
202
203 void GA::selectTwoParents(std::vector<int>& indexParents)
204 {
205     int indexParent1 = indexParents[0];
206     int indexParent2 = indexParents[1];
207     bool bFound = false;
208     int index;
209
210     while (bFound == false)

```

```
211     {
212         index = getRandom(populationDim-1);
213
214         if (randomSelectionChance > getRandom(100))
215         {
216             indexParent1 = index;
217             bFound = true;
218         }
219         else
220         {
221
222             if (this->chromosomes[index]->fitnessRank + 1 > getRandom      ↗
                (populationDim-1))
223             {
224                 indexParent1 = index;
225                 bFound = true;
226             }
227         }
228     }
229     bFound = false;
230     while (bFound == false)
231     {
232         index = getRandom(populationDim-1);
233
234         if (randomSelectionChance > getRandom(100))
235         {
236             if (index != indexParent1)
237             {
238                 indexParent2 = index;
239                 bFound = true;
240             }
241         }
242         else
243         {
244
245             if ((index != indexParent1) && (this->chromosomes[index]-      ↗
                >fitnessRank + 1 > getRandom(populationDim-1)))
246             {
247                 indexParent2 = index;
248                 bFound = true;
249             }
250         }
251     }
252     indexParents[0] = indexParent1;
253     indexParents[1] = indexParent2;
254 }
255
256 int GA::getFitnessRank(double fitness)
257 {
258     int fitnessRank = -1;
259     for (int i = 0; i < populationDim; i++)
260     {
261         if (fitness >= this->chromosomes[i]->fitness)
262         {
263             fitnessRank++;
264         }
265     }
```

```
265     }
266
267     return (fitnessRank);
268     return (fitnessRank);
269 }
270
271 void GA::computeFitnessRankings()
272 {
273     for (int i = 0; i < populationDim; i++)
274     {
275         this->chromosomes[i]->fitness = getFitness(i);
276     }
277
278     for (int i = 0; i < populationDim; i++)
279     {
280         this->chromosomes[i]->fitnessRank = getFitnessRank(this->chromosomes  ↗
281             [i]->fitness);
282     }
283
284     double rBestFitnessVal;
285     double rWorstFitnessVal;
286     for (int i = 0; i < populationDim; i++)
287     {
288         if (this->chromosomes[i]->fitnessRank == populationDim - 1)
289         {
290             rBestFitnessVal = this->chromosomes[i]->fitness;
291             this->bestFitnessChromIndex = i;
292         }
293         if (this->chromosomes[i]->fitnessRank == 0)
294         {
295             rWorstFitnessVal = this->chromosomes[i]->fitness;
296             this->worstFitnessChromIndex = i;
297         }
298     }
299
300 void GA::doGeneticMating()
301 {
302     int iCnt, iRandom;
303     int indexParent1 = -1, indexParent2 = -1;
304     Chromosome *Chrom1, *Chrom2;
305
306     iCnt = 0;
307
308     //elityzm
309     this->chromNextGen[iCnt]->copyChromGenes(this->chromosomes[this-  ↗
310         >bestFitnessChromIndex]);
311     iCnt++;
312     this->chromNextGen[iCnt]->copyChromGenes(this->chromosomes[this-  ↗
313         >bestFitnessChromIndex]);
314     iCnt++;
315
316     if (dynamic_cast<GAString*>(this) != nullptr)
317     {
318         Chrom1 = new ChromChars(chromosomeDim);
319         Chrom2 = new ChromChars(chromosomeDim);
320     }
321 }
```

```
318     }
319     else if (dynamic_cast<GAFloat*>(this) != nullptr)
320     {
321         Chrom1 = new ChromFloat(chromosomeDim);
322         Chrom2 = new ChromFloat(chromosomeDim);
323     }
324     else
325     {
326         Chrom1 = nullptr;
327         Chrom2 = nullptr;
328     }
329
330     do
331     {
332         std::vector<int> indexes = { indexParent1, indexParent2 };
333         selectTwoParents(indexes);
334         indexParent1 = indexes[0];
335         indexParent2 = indexes[1];
336
337         Chrom1->copyChromGenes(this->chromosomes[indexParent1]);
338         Chrom2->copyChromGenes(this->chromosomes[indexParent2]);
339
340         if (getRandom(1.0) < crossoverProb)
341         {
342             if (this->crossoverType == Crossover::ctOnePoint)
343             {
344                 doOnePtCrossover(Chrom1, Chrom2);
345             }
346             else if (this->crossoverType == Crossover::ctTwoPoint)
347             {
348                 doTwoPtCrossover(Chrom1, Chrom2);
349             }
350             else if (this->crossoverType == Crossover::ctUniform)
351             {
352                 doUniformCrossover(Chrom1, Chrom2);
353             }
354             else if (this->crossoverType == Crossover::ctRoulette)
355             {
356                 iRandom = getRandom(3);
357                 if (iRandom < 1)
358                 {
359                     doOnePtCrossover(Chrom1, Chrom2);
360                 }
361                 else if (iRandom < 2)
362                 {
363                     doTwoPtCrossover(Chrom1, Chrom2);
364                 }
365                 else
366                 {
367                     doUniformCrossover(Chrom1, Chrom2);
368                 }
369             }
370
371             this->chromNextGen[iCnt]->copyChromGenes(Chrom1);
372             iCnt++;
373             this->chromNextGen[iCnt]->copyChromGenes(Chrom2);
```

```

374         iCnt++;
375     }
376     else
377     {
378         this->chromNextGen[iCnt]->copyChromGenes(Chrom1);
379         iCnt++;
380
381         this->chromNextGen[iCnt]->copyChromGenes(Chrom2);
382         iCnt++;
383     }
384 } while (iCnt < populationDim);
385
386 }
387
388 void GA::copyNextGenToThisGen()
389 {
390     for (int i = 0; i < populationDim; i++)
391     {
392         this->chromosomes[i]->copyChromGenes(this->chromNextGen[i]);
393
394         if (i != this->bestFitnessChromIndex)
395         {
396             if ((i == this->worstFitnessChromIndex) || (getRandom(1.0) <  ↗
                 mutationProb))
397             {
398                 doRandomMutation(i);
399             }
400         }
401     }
402 }
403
404 void GA::addChromosomesToLog(int iGeneration, int iNumChromosomesToDisplay)
405 {
406     std::string sGen, sChrom;
407
408     if (iNumChromosomesToDisplay > this->populationDim)
409     {
410         iNumChromosomesToDisplay = this->chromosomeDim;
411     }
412
413     for (int i = 0; i < iNumChromosomesToDisplay; i++)
414     {
415         this->chromosomes[i]->fitness = getFitness(i);
416         sGen = "" + std::to_string(iGeneration);
417         if (sGen.length() < 2)
418         {
419             sGen = sGen + " ";
420         }
421         sChrom = "" + std::to_string(i);
422         if (sChrom.length() < 2)
423         {
424             sChrom = sChrom + " ";
425         }
426         std::cout << "Gen " << sGen << ": Chrom" << sChrom << " = " << this-  ↗
             << this->chromosomes[i]->getGenesAsStr() << ", fitness = " << std::fixed <<  ↗
             << std::setprecision(8) << this->chromosomes[i]->fitness << std::endl;

```



```
427     }
428 }
429
430
431 long long GA::binaryStrToInt(const std::string & sBinary)
432 {
433     long long digit, iResult = 0;
434
435     int ilen = sBinary.length();
436     for (int i = ilen - 1; i >= 0; i--)
437     {
438         if (sBinary[i] == '1')
439         {
440             digit = 1;
441         }
442         else
443         {
444             digit = 0;
445         }
446         iResult += (digit << (ilen - i - 1));
447     }
448     return (iResult);
449 }
450
451
452 double GA::getAvgDeviationAmongChroms()
453 {
454     int devCnt = 0;
455     for (int iGene = 0; iGene < this->chromosomeDim; iGene++)
456     {
457         if (dynamic_cast<GAString*>(this) != nullptr)
458         {
459             wchar_t bestFitGene = (static_cast<ChromChars*>(this->chromosomes
460 [this->bestFitnessChromIndex]))->getGene(iGene);
461             for (int i = 0; i < this->populationDim; i++)
462             {
463                 wchar_t thisGene = (static_cast<ChromChars*>(this->chromosomes
464 [i]))->getGene(iGene);
465                 if (thisGene != bestFitGene)
466                 {
467                     devCnt++;
468                 }
469             }
470         }
471         else if (dynamic_cast<GAFloat*>(this) != nullptr)
472         {
473             double bestFitGene = (static_cast<ChromFloat*>(this->chromosomes
474 [this->bestFitnessChromIndex]))->getGene(iGene);
475             for (int i = 0; i < populationDim; i++)
476             {
477                 double thisGene = (static_cast<ChromFloat*>(this->chromosomes
478 [i]))->getGene(iGene);
479                 if (thisGene != bestFitGene)
480                 {
481                     devCnt++;
482                 }
483             }
484         }
485     }
486     return (devCnt / (this->populationDim * this->chromosomeDim));
487 }
```

```
479         }
480     }
481     else
482     {
483         //Chromstrings
484     }
485 }
486
487     return (static_cast<double>(devCnt));
488 }
489
490 GA::~~GA()
491 {
492 }
```