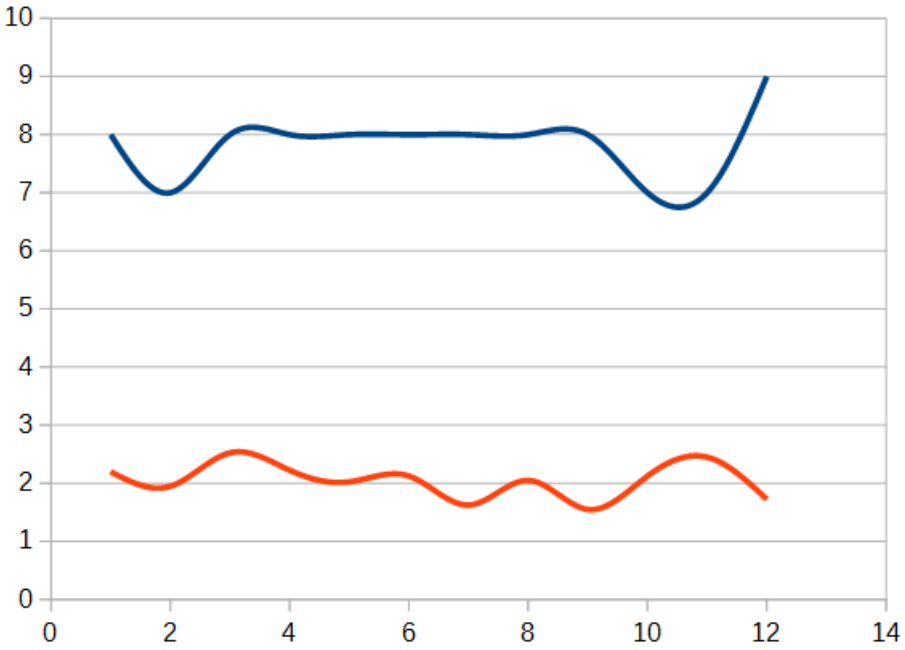
```cpp
#include "stdafx.h"
#include <iostream>
#include <vector>
#include <ctime>
#include <fstream>

using namespace std;

int mark(int number, vector<vector<bool>> chrome);
vector<vector<bool>> selection(vector<vector<bool>> generation, vector<int> marks, int
number);
vector<vector<bool>> evolution(vector<vector<bool>> par, unsigned number_of_chr, unsigned
number_of_gen);
void mutation(vector<vector<bool>> &generation, unsigned procent);
void vector_output(vector<vector<bool>> vec);


int main()
{
	srand(time(0));
	vector<vector<vector<bool>>> library;
	ofstream fout, stat;
	fout.open("D:\\test\\test.txt");
	fout.close();
	fout.open("D:\\test\\test.txt");
	stat.open("D:\\tets\\test2.txt");

	int size = 64; //размер хромосомы
	int num = 40; //количество хромосом в поколении
	int par = 10; //количество родителей

	vector<vector<bool>> generation;
	for (int i = 0; i < num; i++) {
		vector<bool> temp;
		for (int j = 0; j < size; j++) {
			temp.push_back(rand() % 2);
		}
		generation.push_back(temp);
	}
	library.push_back(generation);
	cout << "generation 0:" << endl;
	vector_output(generation);
	int score = 1;
	while (true) {
		vector<int> weight;
		int counter = 0;
		vector<vector<vector<bool>>> groups;
		groups.resize(4);
		for (int i = 0; i < num; i++) {
			if (i % 10 == 0 && i != 0) counter++;
			groups[counter].push_back(generation[i]);
		}
		for (int i = 0; i < groups.size(); i++) {
			for (int j = 0; j < groups[i].size(); j++) {
				weight.push_back(mark(j, groups[i]));
			}
		}

		int max = 0, min = 9;
		double mid = 0;
		for (int i = 0; i < weight.size(); i++) {
			if (weight[i] > max) max = weight[i];
			if (weight[i] < min) min = weight[i];
			mid += weight[i];
		}
```

```
            mid /= weight.size();
            fout << score << '   ' << max << ' ' << mid << ' ' << min << '\n';
            if (max == 9) break;

            vector<vector<bool>> parents;
            parents = selection(generation, weight, par);
            generation.clear();
            generation = evolution(parents, num, size);
            mutation(generation, 5);
            library.push_back(generation);
            cout << "generation " << score << ":" << endl;
            vector_output(generation);
            score++;
            if (score == 10000) break;
        }
    return 0;
}

int mark(int number, vector<vector<bool>> chrome) {
        int result = 0;
        for (int k = 0; k < chrome.size(); k++) {
            if (k == number) break;
            int res1 = 0, res2 = 0;
            for (int i = 0; i < chrome[0].size(); i++) {
                if (chrome[number][i] == false && chrome[k][i] == false) {  //правила
                    res1 += 3; res2 += 3;
                }
                else if (chrome[number][i] == false && chrome[k][i] == true) {
                    res1 += 0; res2 += 5;
                }
                else if (chrome[number][i] == true && chrome[k][i] == false) {
                    res1 += 5; res2 += 0;
                }
                else if (chrome[number][i] == true && chrome[k][i] == true) {
                    res1 += 2; res2 += 2;
                }
            }
            if (res1 > res2) result++;
        }
        return result;
}

vector<vector<bool>> selection(vector<vector<bool>> generation, vector<int> marks, int
number) {
        vector<vector<bool>> result;
        for (int i = 0; i < number; i++) {
            int temp = marks[0];
            int point = 0;
            for (int j = 1; j < marks.size(); j++) {
                if (marks[j] > temp) {
                    temp = marks[j];
                    point = j;
                }
            }
            result.push_back(generation[point]);
            temp = marks[marks.size() - 1];
            marks[marks.size() - 1] = point;
            marks[point] = temp;
            marks.pop_back();
            generation[point] = generation[generation.size() - 1];
            generation.pop_back();
        }
        return result;
}
```

```
vector<vector<bool>> evolution(vector<vector<bool>> par, unsigned number_of_chr, unsigned
number_of_gen) {
	vector<vector<bool>> result;
	for (int i = 0; i < number_of_chr; i++) {
		unsigned mother = rand() % par.size();
		unsigned father;
		while (true) {
			father = rand() % par.size();
			if (mother != father) break;
		}
		unsigned point = rand() % number_of_gen;
		vector<bool> child;
		for (int j = 0; j < number_of_gen; j++) {
			if (j <= point) child.push_back(par[mother][j]);
			else child.push_back(par[father][j]);
		}
		result.push_back(child);
	}
	return result;
}

void mutation(vector<vector<bool>> &generation, unsigned procent) {
	for (int i = 0; i < generation.size(); i++) {
		if (rand() % 100 + 1 >= procent) {
			int j = rand() % generation[i].size();
			if (generation[i][j] == 0) generation[i][j] = 1;
			else generation[i][j] = 0;
		}
	}
}

void vector_output(vector<vector<bool>> vec) {
	for (int i = 0; i < vec.size(); i++) {
		for (int j = 0; j < vec[i].size(); j++) {
			cout << vec[i][j];
		}
		cout << "   ";
		if (i % 2 == 0 && i != 0) cout << endl;
	}
	cout << endl;
}
```



X is iteration

Y is fitness

The blue line is maximum fitness in population.

The orange lane is average fitness in population.

Game examples:

1. From 6th iteration

First chromosome		Second chromosome		WIN
1001111010101011111011010010001 0110111111111111000111111010111111	153	0001011010111011111011000101000 1 1101111111111110000111111010111111	143	1
1001111010101011111011010010001 0110111111111111000111111010111111	159	0001010010111011111011001000000 1 1101111111111111000111111010110111	139	1
1001111010101011111011010010001 0110111111111111000111111010111111	153	0001010010111011111111100010100 01 1101111111111111010000111101111111	143	1
1001111010101011111011010010001 0110111111111111000111111010111111	156	1001010010111011111011001001000 1 0110111111111111000111111001111 1	141	1
1001111010101011111011010010001 0110111111111111000111111010111111	162	1001010010100011111011000101000 1 1101111111101110000111111010111111	137	1
1001111010101011111011010010001 0110111111111111000111111010111111	153	1000110101110110111011000101000 0 1101111111111111010011111010111111	143	1
1001111010101011111011010010001 0110111111111111000111111010111111	153	1000110100110110111111100010100 01 1101111111111111000011111110111111	143	1
1001111010101011111011010010001 0110111111111111000111111010111111	144	1101101100110110111011000101000 1 1101111111111111010011111111111111	149	2
1001111010101011111011010010001 0110111111111111000111111010111111	153	1001101010111011111011001001000 1 1110111111111111010001111011110 1	143	1

2. From last iteration

0100010001101100111010010101100 1101111111111011000011100110010 0	163	1011101000011011101110110011100 1 0110010010100001000011011001110 1	153	1
0100010001101100111010010101100 1101111111111011000011100110010 0	169	1001101011101000110100110010001 1 0011010101000000111101101001011 0	149	1
0100010001101100111010010101100 1101111111111011000011100110010 0	193	0001110000000001000000110110100 00 0010000010011001101110101011100 0	133	1
0100010001101100111010010101100 1101111111111011000011100110010 0	175	0001100001010100110010111010011 0 1010010101010100010011100011101 0	145	1
0100010001101100111010010101100 1101111111111011000011100110010 0	160	0010110110110111101001101011000 1 1111011110000000011001000111110 0	155	1
0100010001101100111010010101100 1101111111111011000011100110010 0	178	0100100000110011001101110101010 0 0101101100000100001110011100101 0	143	1
0100010001101100111010010101100 1101111111111011000011100110010 0	175	1111100110100010001110110101101 0 1001001110010010100100000001100 0	145	1
0100010001101100111010010101100 1101111111111011000011100110010 0	184	0001000000101111001010011101011 0 1000100000010010100110010011010 1	139	1
0100010001101100111010010101100 1101111111111011000011100110010 0	163	0000000010110111101110011010011 1 0101000011011010110101001110111 0	153	1

3. From 1st generation

0101101010100000001101110101001 01 1111100011100110100110000111011 00	135	1111011011011111011101100111101 0 0000011000110101111011101001111 1	175	2
0101101010100000001101110101001 01 1111100011100110100110000111011 00	162	0010111010101111100011111101001 0 0010001110000100110100100110110 0	157	1
0101101010100000001101110101001 01 1111100011100110100110000111011 00	183	0000001010100101100110001011100 1 0101100011000101001010000100001 10	143	1
0101101010100000001101110101001 01 1111100011100110100110000111011 00	165	1110010100110100101001111110000 0 1000011011100001010011011101010 0	155	1
0101101010100000001101110101001 01 1111100011100110100110000111011 00	171	0001000000110001000110111011111 1 0111000000101011010000010011101 1	151	1
0101101010100000001101110101001 01 1111100011100110100110000111011 00	153	0010011100101101010111001110111 1 0100101000010111110110000011011 1	163	2
0101101010100000001101110101001 01 1111100011100110100110000111011 00	171	1100000011001001011101000111011 0 0101010101100011100011010010001 0	151	1
0101101010100000001101110101001 01 1111100011100110100110000111011 00	162	1111101010100110001100010110100 0 1011011000110000010001111010101 1	157	1
0101101010100000001101110101001 01 1111100011100110100110000111011 00	165	1101100001000100010000110110110 1 1111010101001100101100001011011 0	155	1