

דוח מעבדה 1

בקורס מעבדה לבינה מלאכותית

הגשת:

מוחמד סח – 209332212

יוסף עאסלה - 207344573

חלק א':

סעיף 1:

מוצע ה-FITNESS של האוכלוסיה ושל סטיית התקן מהממוצע:

הוספנו קוד לפונקציה calc_fitness הנתונה :

```
void calc_fitness(ga_vector &population, int method)
{
    if (method == 0) // if method == 0 normal method else Bullseye method
    {
        string target = GA_TARGET;
        int tsize = target.size();
        unsigned int fitness;
        // average and devi parameters
        float average = 0;
        float devi = 0;

        for (int i = 0; i < GA_POPSIZE; i++) {
            fitness = 0;
            for (int j = 0; j < tsize; j++) {
                fitness += abs(int(population[i].str[j] - target[j]));
            }
            population[i].fitness = fitness;
            //summing fitness
            average += fitness;
        }
        //calculate average
        average = average / GA_POPSIZE;
        for (int i = 0; i < GA_POPSIZE; i++)
            devi += pow(population[i].fitness - average, 2);
        devi = sqrt(devi / GA_POPSIZE);
        //updating average and deviation
        for (int i = 0; i < GA_POPSIZE; i++)
        {
            population[i].average = (int)average;
            population[i].devi = (int)devi;
        }
    }
    else // Bullseye
        BullsEye(population);
}
```

והוספנו גם average ,devi(deviation) ל ga_struct.

סעיף 2:

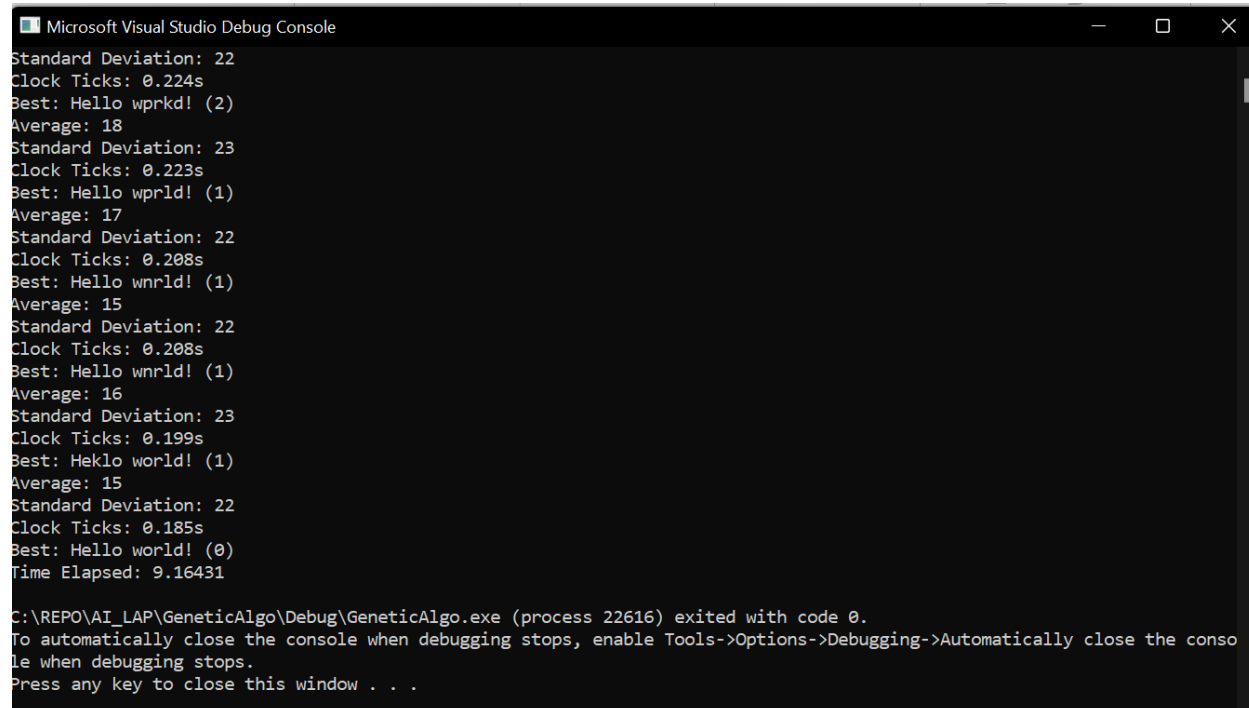
זמן ריצה, clock ticks ו זמן ריצה אבסולוטי:

השתמשו בספריות chrono ו ctime, הוספנו ל main function:

```
using clock = std::chrono::system_clock;
using sec = std::chrono::duration<double>;
```

```
clock_t end = std::clock();
float toatl_time = (float)(end - start) / CLOCKS_PER_SEC;
numOfGenerations++;
cout << "Average: " << (*population)[0].average << std::endl;
cout << "Standard deviation: " << (*population)[0].devi << std::endl;
cout << "Clock Ticks: " << toatl_time << "s" << std::endl;
}
const sec duration = clock::now() - before;
cout << "Time Elapsed: " << duration.count() << std::endl;
return 0;
}
```

דוגמא להרצת קוד:



```
Microsoft Visual Studio Debug Console

Standard Deviation: 22
Clock Ticks: 0.224s
Best: Hello wprkd! (2)
Average: 18
Standard Deviation: 23
Clock Ticks: 0.223s
Best: Hello wprld! (1)
Average: 17
Standard Deviation: 22
Clock Ticks: 0.208s
Best: Hello wnrlld! (1)
Average: 15
Standard Deviation: 22
Clock Ticks: 0.208s
Best: Hello wnrlld! (1)
Average: 16
Standard Deviation: 23
Clock Ticks: 0.199s
Best: Heklo world! (1)
Average: 15
Standard Deviation: 22
Clock Ticks: 0.185s
Best: Hello world! (0)
Time Elapsed: 9.16431

C:\REPO\AI_LAP\GeneticAlgo\Debug\GeneticAlgo.exe (process 22616) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

סעיף 4:

אופרטורים לשיחלוף:

הוספנו קטע קוד ל `mate function` שקיבלנו:

מה שקבלנו הוא ה `one point` :

```
if(PointOp == 0) // one point
{
    spos = rand() % tsize;

    buffer[i].str = population[i1].str.substr(0, spos) +
        population[i2].str.substr(spos, tsize - spos);
}
```

הוספנו את ה `two point` :

```
if (PointOp == 1) // two point
{
    int curr_max, curr_min;
    spos = rand() % tsize;
    spos2 = rand() % tsize;
    curr_max = std::max(spos, spos2);
    curr_min = std::min(spos, spos2);
    str1 = population[i1].str.substr(0, curr_min);
    str2 = population[i2].str.substr(curr_min, curr_max - curr_min);
    str3 = population[i1].str.substr(curr_max, tsize - curr_max);
    buffer[i].str = str1 + str2 + str3;
}
```

וגם ה `uniform` :

```
if (PointOp == -1) // uniform
{
    string new_str;
    new_str.erase();
    for (int j = 0; j < tsize; j++)
    {
        spos = rand() % 2;
        if (spos == 0)
            new_str += population[i1].str.substr(j, 1);
        else
            new_str += population[i2].str.substr(j, 1);
    }
    buffer[i].str = new_str;
}
if (rand() < GA_MUTATION) mutate(buffer[i]);
}
```

סעיף 5:

היוריסטיקה נוספת "בול פגיעה":

הוספנו היוריסטיקה חדשה "בול פגיעה" שהיא נותנת משקל נוסף על ניחוש נכון:

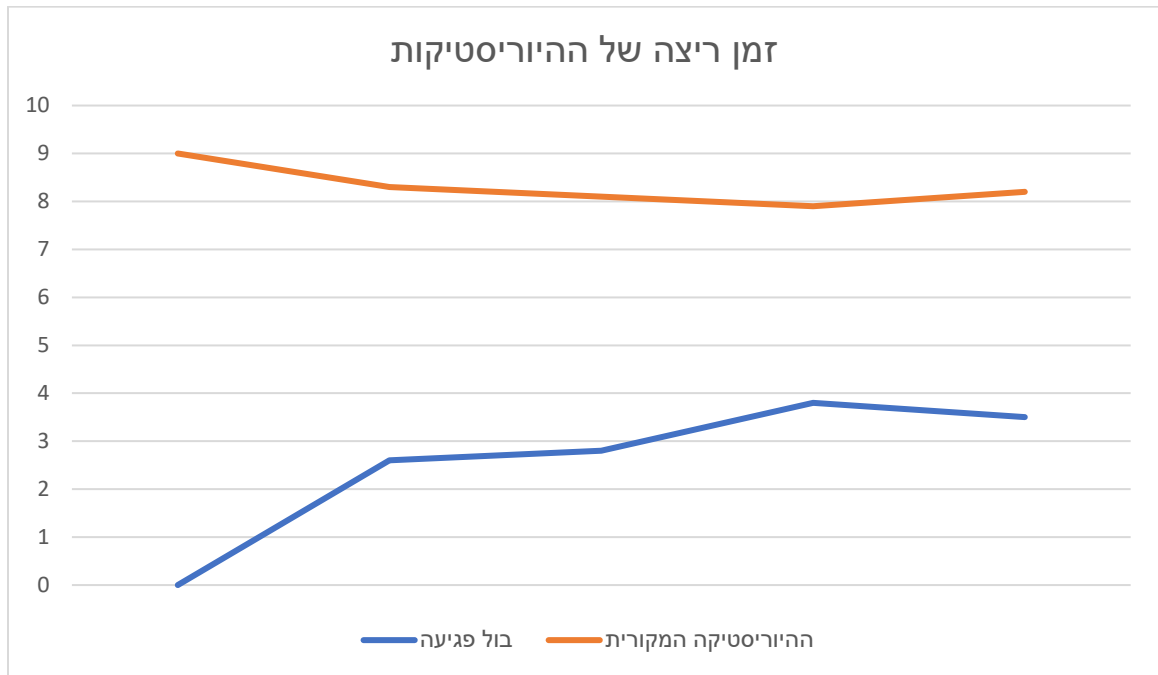
```
//same struction as normal method
string target = GA_TARGET;
int tsize = target.size();
unsigned int fitness;
float average = 0, devi = 0;
for (int i = 0; i < GA_POPSIZE; i++)
{
    fitness = tsize * 10;
    for(int k = 0 ;k < tsize; k++)
    {
        //if correct asseption
        if (population[i].str[k] == target[k])
            // bonus for correct asseption
            fitness -= 10;
        else
        {
            for (int k = 0; k < tsize; k++)
            {
                // if correct asseption but wrong place
                if(population[i].str[k] == target[k])
                {
                    fitness--;
                    break;
                }
            }
        }
    }
    population[i].fitness = fitness;
    average += fitness;
}
```

סעיף 6:

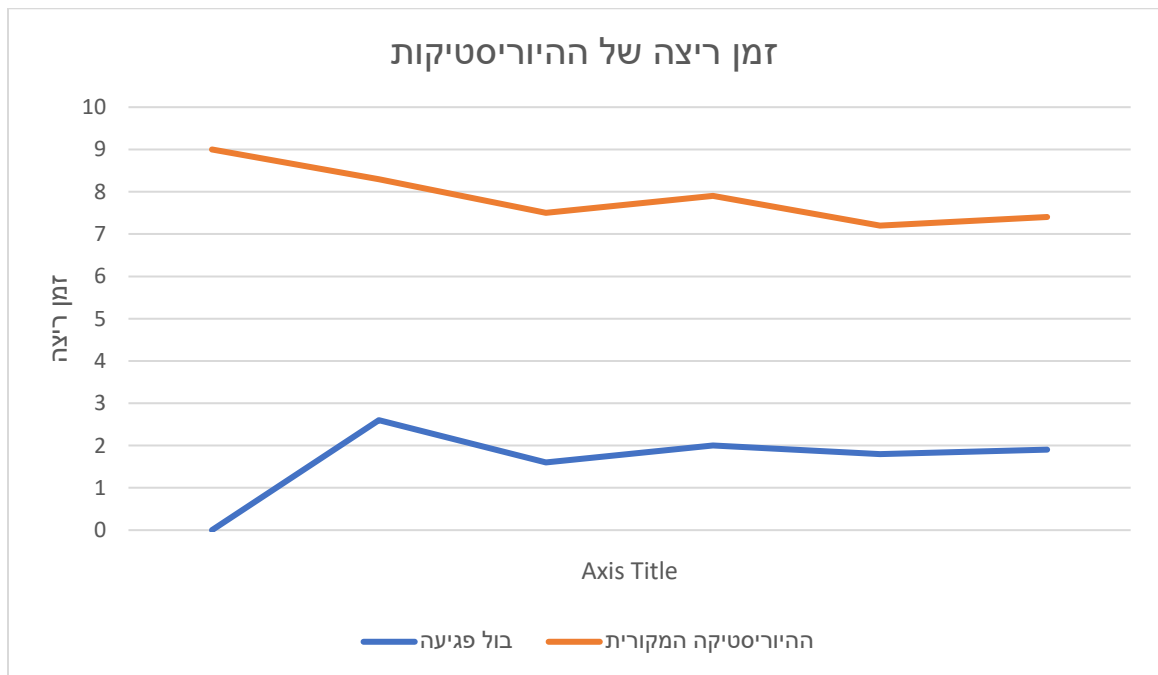
השוואה בין ההיוריסטיקות:

הרצנו את הקוד מספר פעמים, כמו שרואים בגרף היוריסטיקת "בול פגיעה" יותר מהירה וזה בגלל שהיא נותנת משקל נוסף לחיזוי נכון.

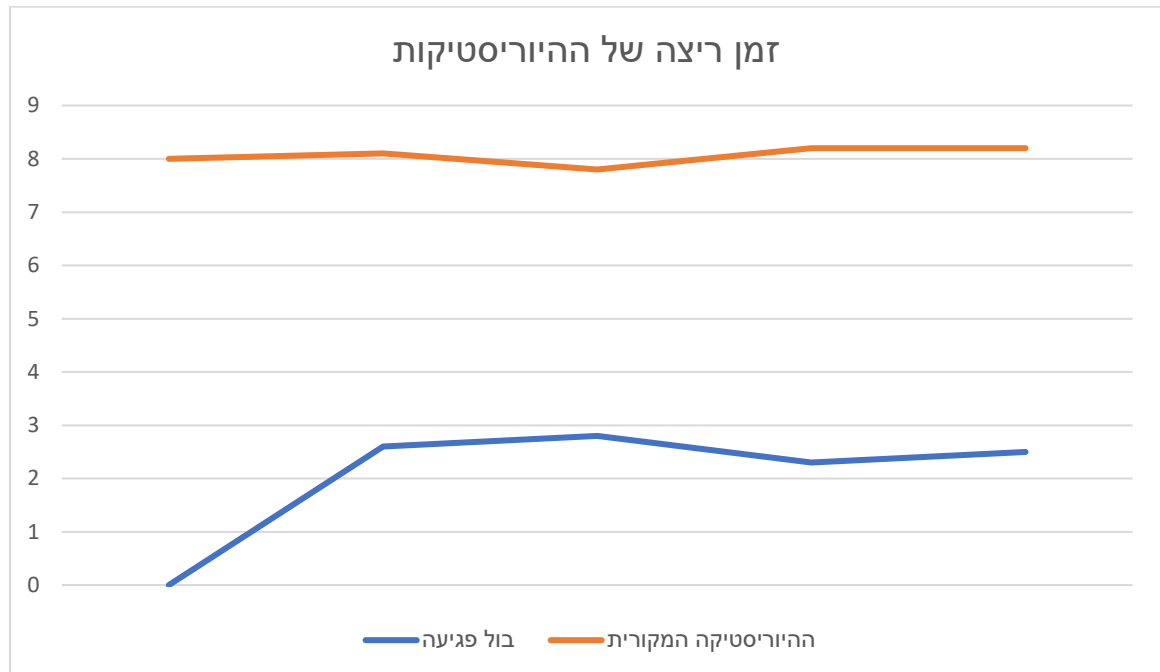
עבור uniform point operation :



עבור 1 point operation :



עבור 2 point operation:



סעיף 7:

point operations ו mate function הם החלק האחראי על ה EXPLORATION, מפנה שבחלק זה אנחנו "מערבבים" גינים ו מקבלים גינים חדשים.

פונקצית elitism היא החלק האחראי על ה EXPLOITATION מפנה שבחלק זה אנחנו מעבירים אחוז מסויים מהגינים הכי טובים לדור הבא.

סעיף 8:

PSO ALGORITHM

הוספנו מחלקה חדשה PSO שתומכת ב particles

השתמשנו בנוסחה שלמדנו בקורס המבוא :

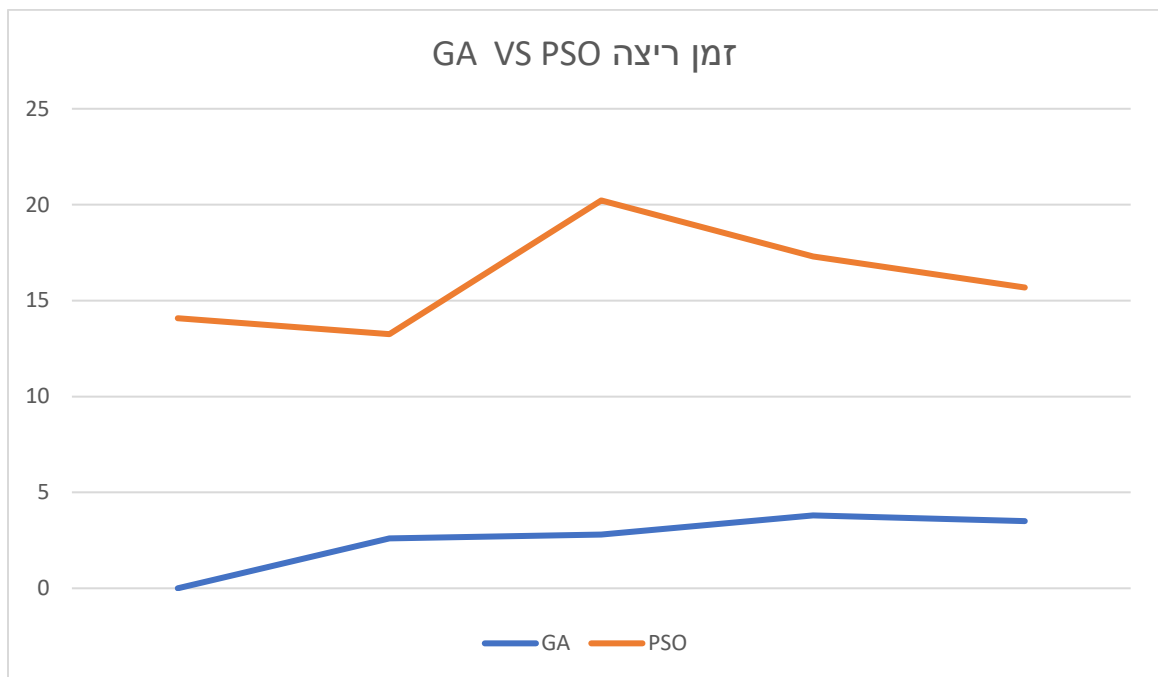
$$\vec{X}_i^{t+1} = \vec{X}_i^t + \vec{V}_i^{t+1}$$
$$\vec{V}_i^{t+1} = \underbrace{w\vec{V}_i^t}_{\text{Inertia}} + \underbrace{c_1 r_1 (\vec{P}_i^t - \vec{X}_i^t)}_{\text{Cognitive component}} + \underbrace{c_2 r_2 (\vec{G}^t - \vec{X}_i^t)}_{\text{Social component}}$$

```
//Particle Position update
for (int j = 0; j < tsize; j++) {
    //based on equation that we leanded in class
    double r1 = (double)rand() / (RAND_MAX);
    double r2 = (double)rand() / (RAND_MAX);
    double inertia = W * particle_vector[i].get_velocity()[j];
    double cognitive = C1 * r1 * (particle_vector[i].get_localBest()[j] - particle_vector[i].get_str()[j]);
    double social = C2 * r2 * (globalBest[j] - particle_vector[i].get_str()[j]);
    double ics = inertia + cognitive + social;
    myVelocity += ics;
    myStr += particle_vector[i].get_str()[j] + myVelocity[j];
}
```


סעיף 9:

נשתמש בהיוריסטיקה "פול פגיעה" ו ב 1 point operation בהשוואה זו:

כמו שאנחנו רואים בגרף יש הבדל גדול בין זמן הריצה ו מספר ה איטרציות בין שני האלגוריתמים. האלגוריתם הגנטי יותר מהיר ו יעיל לכן הוא מועדף יותר לשימוש.



חלק ב:

סעיף 1: הוספנו selectparent שהיא אחראית על בחירת התמיכה בשיטות הבחירה

```
#define parenttype 4 //parent type

int* selectParents(ga_vector &population)
{
    int esize = static_cast<int>(GA_POPSIZE * GA_ELITRATE);
    int numOfParents = 2 * (GA_POPSIZE - esize);
    int *parents = new int[numOfParents];
    //sorted numbers
    int *points = new int[numOfParents];
    //calculating new fitness
    int* newFitness = new int[GA_POPSIZE];
    for (int i = 0; i < GA_POPSIZE; i++) {
        newFitness[i] = ((-1)*(population[i].fitness) + population[GA_POPSIZE - 1].fitness);
    }
    //total fitness
    long totalFitness = 0;
    for (int i = 0; i < GA_POPSIZE; i++) {
        totalFitness += newFitness[i];
    }
    //parent type:
    if (parenttype == 1)
    {
        //RWS + Scaling
        Scaling(population);
        for (int i = 0; i < numOfParents; i++) {
            *(points + i) = rand() % (totalFitness + 1);
        }
        //sorting
        sort(points, points + numOfParents);
        //calling RWS
        parents = RWS(population, points, newFitness);
    }
    if (parenttype == 2)
    {
        // SUS method
        parents = SUS(population, totalFitness, newFitness);
    }
    if (parenttype == 3)
    {
        // Tournament selection
        parents = Tournament(population);
    }
    return parents;
}
```

RWS:

```
//
int* RWS(ga_vector &population, int *points, int* newFitness)
{
    int esize = static_cast<int>(GA_POPSIZE * GA_ELITRATE);
    int numOfParents = 2 * (GA_POPSIZE - esize);
    //parent
    int *parents = new int[numOfParents];
    int* sumFitness = new int[GA_POPSIZE];
    sumFitness[0] = newFitness[0];
    // summing fitness
    for (int i = 1; i < GA_POPSIZE; i++) {
        sumFitness[i] = sumFitness[i - 1] + newFitness[i];
    }
    //parent select
    for (int i = 0; i < numOfParents; i++) {
        int j = 0;
        while (sumFitness[j] < *(points + i) && j < GA_POPSIZE) {
            j++;
        }
        if (j >= GA_POPSIZE)
            j = GA_POPSIZE - 1;
        *(parents + i) = j;
    }
    return parents;
}
```

SCALING:

השתמשנו בנוסחה שלמדנו בהרצאה $a*f+b$, קבענו:

$a=0.2$, $b= 10$

```
//*****
void Scaling(ga_vector &population)
{
    //based on lecture scale = a*f+b a=0.2 , b= 10
    for (int i = 0; i < GA_POPSIZE; i++) {
        population[i].fitness = static_cast<unsigned int>(0.2 * population[i].fitness + 10); // linear transformation
    }
}
```

SUS:

השתמשנו באלגוריתם שפותר בעזרת RWS

```

//*****
//SUS function

int* SUS(ga_vector &population, long totalFitness, int* newFitness)
{
    int esize = static_cast<int>(GA_POPSIZE * GA_ELITRATE);
    int numOfParents = 2 * (GA_POPSIZE - esize);
    int *parents = new int[numOfParents];
    // randnum is list of random integers
    int *randnum = new int[numOfParents];
    for (int i = 0; i < numOfParents; i++) {
        *(randnum + i) = rand() % (totalFitness / numOfParents + 1) + (i * totalFitness / numOfParents);
    }
    return RWS(population, randnum, newFitness);
}

```

TOURNAMENT:

השתמשנו באלגוריתם שלמדנו בהרצאה: נבחר מספר מועמדים לטורניר
psize נתחיל עם המועמד הראשון ו אם נמצא מועמד יותר טוב בתהליך
נמשיך אותו את הטורניר.

```

//*****
#define Psize 7 // Tournament size

int* Tournament(ga_vector &population)
{
    int esize = static_cast<int>(GA_POPSIZE * GA_ELITRATE);
    int numOfParents = 2 * (GA_POPSIZE - esize);
    //parents
    int *parents = new int[numOfParents];
    //players of the tournament
    int* players = new int[Psize];
    //loop
    for (int i = 0; i < numOfParents; i++) {
        for (int j = 0; j < Psize; j++) {
            //random players
            players[j] = rand() % GA_POPSIZE;
        }
        //playing the tournament
        int win = players[0];
        for (int s = 0; s < Psize; s++) {
            if (population[players[i]].fitness < population[win].fitness)
                win = players[i];
        }
        //parent = the winner player
        parents[i] = win;
    }
    return parents;
}

```

סעיף 2:

הוספנו ל GA_struct את ה age של כל citizen

```
struct ga_struct
{
    string str;                // the string
    unsigned int fitness;      // its fitness
    int age;
    int average = 0;
    int devi = 0;
};
```

כמו כן הוספנו ב define את הגיל המקסימלי ו המינימאלי

```
#define MAX_AGE 20           // Max age of a citizen
#define Min_AGE 1           // min age of a citizen
```

הגדרנו פינקציה שמעדכנת את הגיל

```
// for this we added age in ga struct
void updateAge(ga_vector &population, int i)
{
    population[i].age += 1;
}

//*****
```

סעיף 3:

עשינו שינוי על ה nq_struct כך שיהיה לנו מערך באורך מספר המלכות שמהוות פרמוטציה כלשהו, כלומר לא יהיו שתי מלכות באותה עמודה או שורה.

```
struct nq_struct
{
    int* board = new int[N];    // The chess size N
    unsigned int fitness;       // The Fitness;
    unsigned int age;
};
```

```

void init_population(nq_vector& population, nq_vector& buffer)
{
    for (int i = 0; i < GA_POPSIZE; i++)
    {
        nq_struct citizen;
        citizen.fitness = 0;
        citizen.age = 1;
        for (int i = 0; i < N; i++) // checking that no more 2 queens in the same slot
            citizen.board[i] = i;
        random_shuffle(citizen.board, citizen.board + N);
        population.push_back(citizen);
    }
    buffer.resize(GA_POPSIZE);
}

```

סעיף 4:

אופרטורי מוטציה:

```

int* swapIndexArea(int* arr, int i, int j)
{
    int temp = *(arr + i);
    *(arr + i) = *(arr + j);
    *(arr + j) = temp;
    return arr;
}

void exchangeMutation(nq_struct& member)
{
    int i = rand() % N;
    int j = rand() % N;
    member.board = swapIndexArea(member.board, i, j);
}

```

```

void insertionMutation(nq_struct& member)
{
    int index_i = rand() % N;
    int temp = index_i;
    int index_j = rand() % N;
    int* myArr = new int[N];
    index_i = min(index_i, index_j);
    index_j = max(temp, index_j);
    for (int i = 0; i < index_i; i++)
        *(myArr + i) = member.board[i];
    temp = *(myArr + index_i);
    for (int i = index_i; i < index_j; i++)
        *(myArr + i) = member.board[i + 1];
    *(myArr + index_j) = temp;
    for (int i = index_j + 1; i < N; i++)
        *(myArr + i) = member.board[i];
    member.board = myArr;
}

```

אופרטורי שחלוף:

```

void PMX(nq_vector& population, nq_struct& memb1, nq_struct& memb2, int ind, int i1, int i2)
{
    for (int i = 0; i < N; i++)
    {
        memb1.board[i] = population[i1].board[i];
        memb2.board[i] = population[i2].board[i];
    }
    int temp1 = memb1.board[ind];
    int temp2 = memb2.board[ind];
    for (int i = 0; i < N; i++) // crossover
    {
        if (memb1.board[i] == temp2)
            memb1.board[i] = temp1;
        if (memb2.board[i] == temp1)
            memb2.board[i] = temp2;
    }
    memb1.board[ind] = temp2;
    memb2.board[ind] = temp1;
}

```

Minimal Conflict Algorithm:

```

void minimalConflict(nq_struct& game)
{
    int* randQueens = new int[N];
    int queen = (rand() % N);
    int prevFitness = game.fitness;
    int prevQueen = game.board[queen];
    for (int i = 0; i < N; i++)
        randQueens[i] = i;
    random_shuffle(randQueens, randQueens + N); //moving queens
    game.board[queen] = randQueens[0];
    game_calc_fitness(game);
    int currentFit = game.fitness;
    int chQueen = randQueens[0];
    for (int i = 1; i < N; i++)
    {
        game.board[queen] = randQueens[i];
        game_calc_fitness(game);
        if (game.fitness < currentFit) // good move
        {
            currentFit = game.fitness;
            chQueen = randQueens[i];
        }
    }
    if (currentFit >= prevFitness) // bad move
    {
        game.board[queen] = prevQueen;
        game_calc_fitness(game);
    }
    game.board[queen] = chQueen;
    game_calc_fitness(game); // updating fitness
}

```

אנחנו נשתמש בפונקציה הזו כדי לתקן את הלוח על ידי הזזת המלכה למקום שבו פחות איומים.

אחרי שהרצנו את הקוד כמה פעמיים, מצאנו שאלגוריתמים ה N Queen עבור genetic algorithm יעיל ומהיר יותר, מכך שאלגוריתם Minimal Conflict עצר ברוב האיטרציות אחרי כניסה ל local opt. לכן עדיף להשתמש ב genetic algorithm.