Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Umelá Inteligencia

Zadanie č.2

a) Zenová záhrada

Akademický rok 2022/2023

Meno: Ján Ágh
Cvičiaci: Ing. Martin Komák, PhD.

Dátum: 13.11.2022
Počet strán: 17

Obsah

1 Stručný opis problematiky a zadania

Cieľom úlohy bolo navrhnúť program využívajúci genetický algoritmus na vyriešenie problému Zenovej záhradky. Spomínaný problém spočíva v tom, že v záhradke vyplnenej pieskom a rôznymi nepohyblivými prekážkami je úlohou mnícha, aby v piesku pomocou hrablí vytvoril súvislé pásy siahajúce od jedného okraja záhradky k druhému bez toho, aby sa ocitol v slepej uličke. Samotná záhradka je ohraničená okrajom, po ktorom sa mních môže ľubovoľne presúvať a zvoliť si začiatočnú pozíciu vstupu do záhradky. Ak sa mních ocitne v slepej uličke, nastáva koniec hry. Problém sa považuje za vyriešený, keď sa mníchovi podarí pásmi pokryť celú záhradku, prípadne jej čo najväčšiu časť.

Druhou úlohou mnícha počas prechodmi záhradkou je zbieranie popadaného lístia. V hre sa nachádzajú tri druhy listov – žlté, oranžové a červené – a mních ich môže zbierať len vo vopred stanovenom poradí, t.j. najprv musí pozbierať žlté, potom oranžové a nakoniec červené lístie. Listy, ktoré v danom momente mních nemôže pozbierať sa správajú rovnako, ako prekážky a musí ich obísť. Cieľ hry ostáva rovnaký – úlohou mnícha je pokryť pásmi čo najväčšiu časť záhradky.

2 Navrhnuté riešenie

2.1 Použité programové prostriedky

Na riešenie úlohy bol použitý programovací jazyk C# v spojení s .NET framework verzie 6.0.300. Riešenie bolo vyvíjané v prostredí Visual Studio Code verzie 1.72.1 a pri jeho vývoji boli využité základné aj pokročilejšie objektovo orientované princípy.

2.2 Organizácia projektu

Implementácia sa nachádza v adresári s názvom "*src*". Tento adresár obsahuje hlavné súbory *Program.cs* a *EulerHorse.csproj*, slúžiace na spustenie vykonávania programu, a štvoricu ďalších adresárov *inputs, constants, logic* a *models*.

Adresár *inputs* obsahuje rozsiahly počet vstupných *.txt* súborov predstavujúcich jednotlivé rozloženia objetkov v záhradke. Každý z týchto súborov pozostáva z horizontálnych a vertikálnych súradníc kameňov a listov, ktoré sa v danej záhradke nachádzajú.

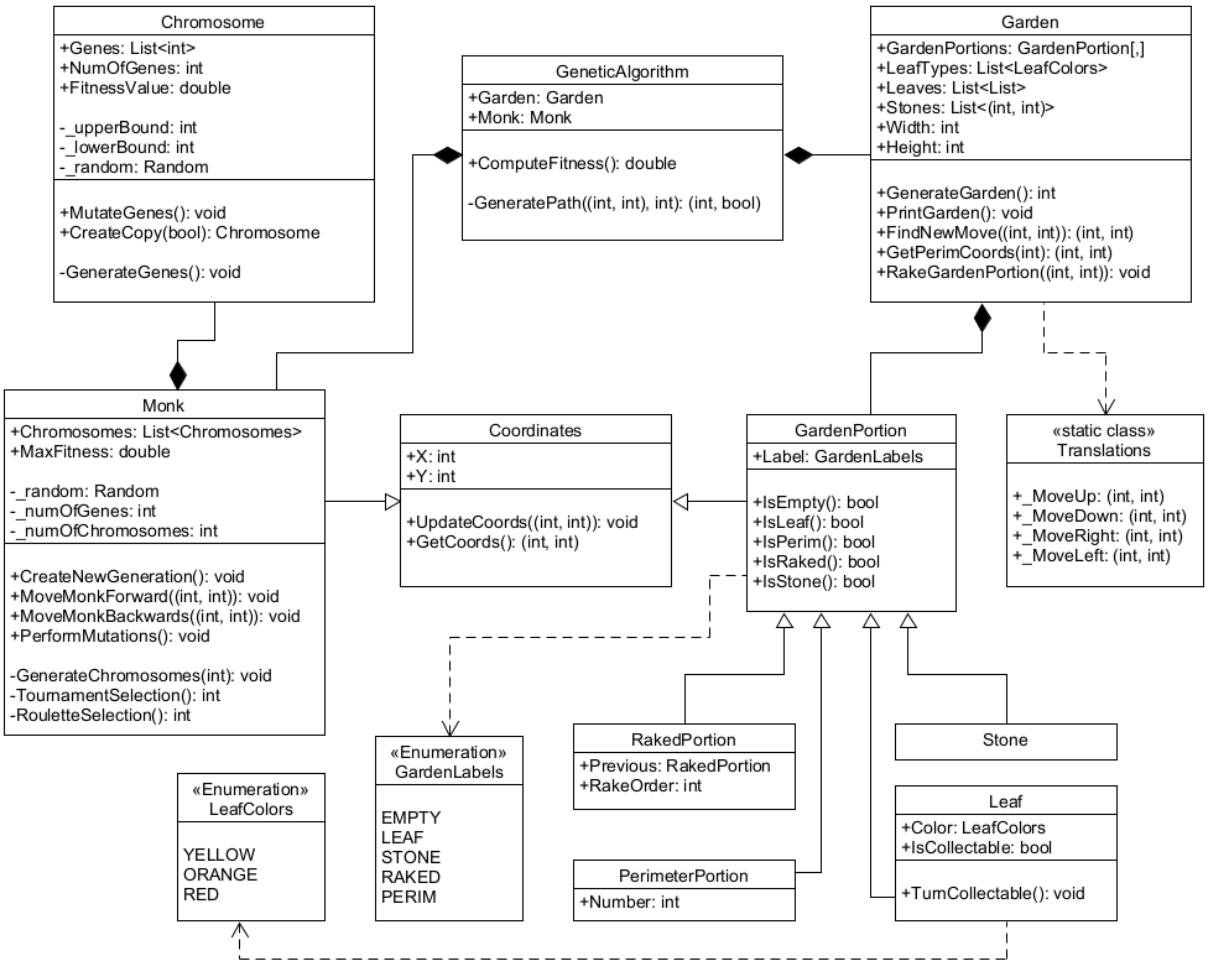
Adresár *constants* obsahuje celkovo tri súbory - *Translations.cs* so všetkými štyrmi dovolenými pohybmi mnícha (hore, dole, doprava a doľava), *GardenLabels.cs* s enumeráciou jednotlivých druhov políčok v záhradke (prázdne políčko, list, kameň, pohrabané políčko a okraj záhradky) a *LeafColors.cs* s enumeráciou troch druhov listov (žlté, oranžové a červené).

Kľúčovým adresárom je *logic*, kde sa nachádza implementácia používateľského rozhrania, hlavný cyklus vykonávania programu a taktiež statické triedy s pomocnými funkcionalitami. Spomínaný hlavný cyklus je umiestnený v súbore *GeneticAlgorithm.cs* vrámci metódy *ComputeFitness()*. Implementáciu používateľského rozhrania je možné nájsť v súbore *UserInterface.cs*. V statickej triede *Validation.cs* sú umiestnené metódy na validáciu pohybu mnícha a zistenie, či jeho súčasná pozícia nie je prekážka, ďalšia statická trieda *Converters.cs* je naplnená pomocnými metódami rôznych druhov, ako napríklad klonovanie objektov, získanie súradníc ľubovoľného okrajového políčka záhradky a mnoho ďalších, a trieda *Counter* je využívaná na meranie času počas vykonávania algoritmu.

Posledným a zároveň nemenej dôležitým adresárom je *models* skladajúci sa z trojice podadresárov – *garden* s triedami týkajúcimi sa samotnej reprezentácie záhradky (*Garden.cs*) a reprezentácie jednotlivých typov políčok v nej (*Stone.cs, Leaf.cs, PerimPortion.cs, RakedPortion.cs* a *GardenPortion.cs*), *monk* s triedami predstavujúcimi samotného mnícha, jednotlivé chromozómy s génmi a umožňujúcimi vytváranie nových generácií jedincov (*Monk.cs* a *Chromosome.cs*) a *general*, ktorý obsahuje jedinú, ale zato nesmierne dôležitú triedu na uchovávanie súradníc všetkých entít v hre (*Coordinates.cs*).

2.3 UML diagram jednotlivých tried

Na obr. 2.1 sa nachádza UML diagram najdôležitejších spolupracujúcich tried algoritmu spolu s jednotlivými úrovňami dedenia, kompozície a agregácie.



Obr. 2.1 UML diagram najdôležitejších tried algoritmu

2.4 Podrobný opis vlastností algoritmu

Návrh aj konkrétna realizácia algoritmu pozostávajú z množiny relevantných krokov, zohrávajúcich nezanedbateľnú úlohu v súvislosti s vyhľadávaním najlepšieho možného riešenia. Spomenuté kroky sa neustále vykonávajú v nasledujúcom poradí:

- *Vytvorenie prvotnej generácie jedincov.*
- *Ohodnotenie jedincov prvotnej generácie podľa fitness.*
- *Vytvorenie jednej z n nových generácií podľa jedincov predchádzajúcej generácie.*
- *Ohodnotenie jedincov uvedenej generácie podľa fitness.*
- *Návrat k bodu č.3 v prípade, ak ešte nebol dosiahnutý konečný počet (n) generácií.*

2.4.1 Návrh vlastností génov

Gény v kontexte genetického algoritmu predstavujú primárne jednotky rozhodovania. Tvoria hlavnú zložku vytvorených jedincov (chromozómov) a určujú ich fundamentálne vlastnosti, na základe ktorých sa jednotliví jedinci od seba odlišujú. Sú taktiež integrálnou súčasťou exaktného určenia fitness hodnoty zvoleného jedinca.

Z pohľadu riešenej problematiky bolo logickým rozhodnutím, aby jednotlivé gény jedinca symbolizovali konkrétnu lokalitu, nachádzajúcu sa na periférii záhradky, odkiaľ bude mníchovi umožnené vstúpiť do záhradky. Numerický rozsah hodnôt predstavujúcich gény bol stanovený na interval od 0 po 100 vrátane, pričom podľa tejto hodnoty sa prostredníctvom pomocnej metódy statickej triedy *Converters* vydedukujú konkrétne súradnice vstupu.

2.4.2 Tvorba počiatočnej generácie jedincov

Základom kompletnej množiny genetických algoritmov je prvotná populácia jedincov obsahujúcich gény s automaticky vygenerovanými hodnotami. Uvedený krok je dôvodom nederministickosti genetických algoritmov – každá iterácia riešenia problematiky má totižto tendenciu objaviť alternatívny výsledok v porovnaní s predchádzajúcimi.

Zhotovenie počiatočnej generácie sa uskutoční okamžite po vytvorení objetku mnícha (triedy *Monk*), tá totižto vo svojom konštruktore automaticky vykoná privátnu metódu *GenerateChromosomes()* (obr. 2.2) na naplnenie zoznamu jedincov, pričom každý jedinec vo svojom vlastnom konštruktore spustí vykonávanie privátnej metódy *GenerateGenes()* (obr. 2.3), pomocou ktorej sa vygeneruje zoznam génov s náhodne nastavenými hodnotami.

```
1 reference
private void GenerateChromosomes(int genes)
{
    for (int count = 0; count < _numOfChromosomes; count++)
    {
        Chromosomes.Add(new Chromosome(true, genes));
    }
}
```

Obr. 2.2 Metóda GenerateChromosomes() triedy Monk

```
1 reference
private void GenerateGenes()
{
    for (int count = 0; count < NumOfGenes; count++)
    {
        Genes.Add(_random.Next(_lowerBound, _upperBound + 1));
    }
}
```

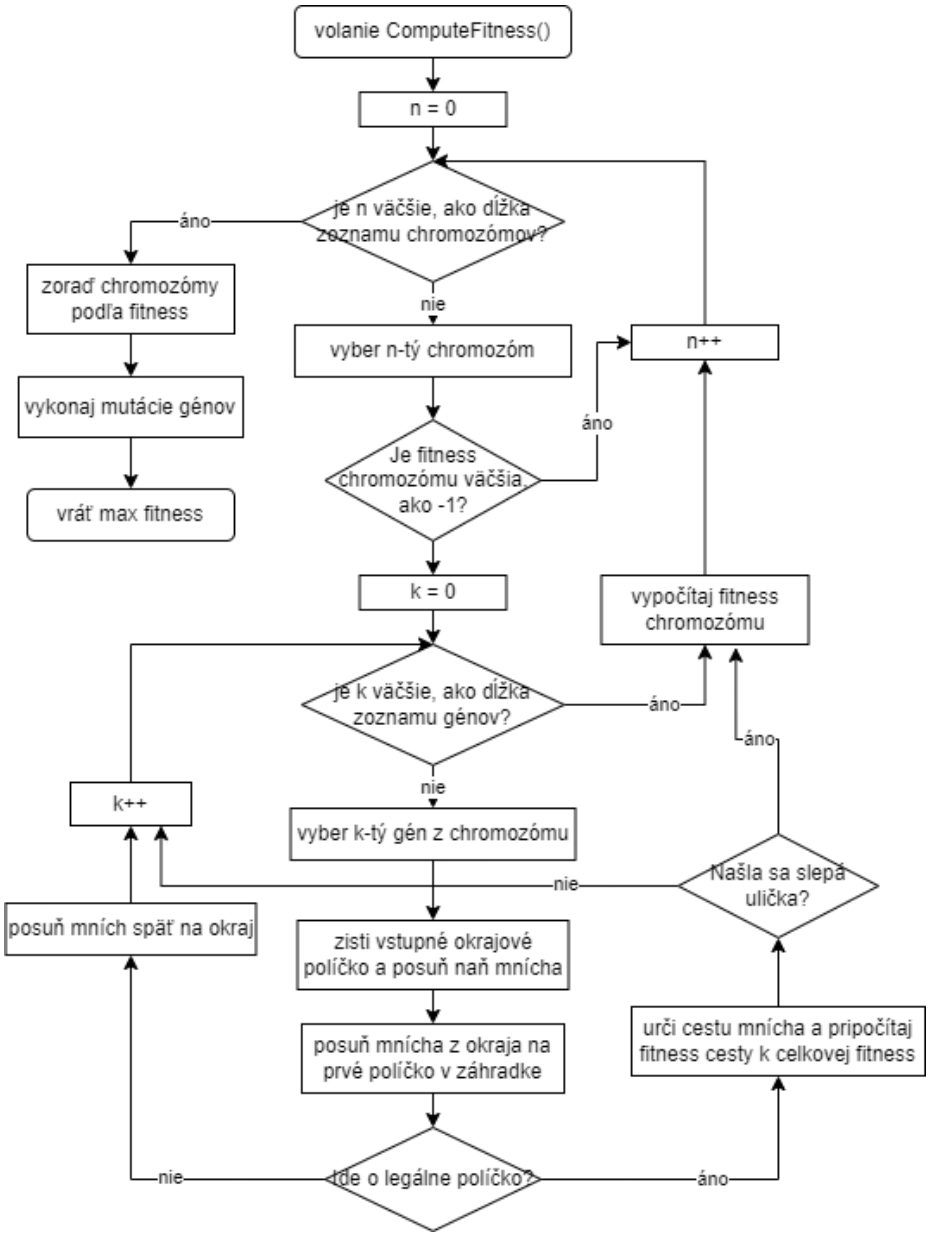
Obr. 2.3 Metóda GenerateGenes() triedy Chromosome

2.4.3 Kalkulácia hodnoty fitness

Fitness hodnota jedincov by sa dala považovať za najdôležitejší parameter genetických algoritmov ako takých, podľa nej sa totižto selektujú jedinci na automatický presun do novej

generácie, odstraňujú nedostatočne vyhovujúci jedinci a vyhľadáva sa globálne najlepšie ohodnotený jedinec, ktorý prestavuje akceptované riešenie problematiky vo zvolenej kvantite zkonštruovaných generácií.

Nástroj na výpočet fitness hodnoty konkrétneho jedinca predstavuje metóda *ComputeFitness()* triedy *GeneticAlgorithm.* Spomenutá metóda iteruje nad celkovou súčasnou populáciou jedincov a určí ich fitness hodnotu v jednom volaní. Na obr. 2.4 sa nachádza jej detailný vývojový diagram.



Obr. 2.4 Vývojový diagram metódy procesu určovania fitness hodnôt

V samotnej kalkulácii zohráva signifikantnú úlohu rozsiahly počet parametrov. Jednoznačne najdôležitejšou z nich je počet úspešne pokrytých políčok záhradky, ktorý k celkovému fitness skóre prispieva celou jednotkou (1.0) za každú pokrytú lokalitu. Ďalším v rade je početnosť potrebných vstupov mnícha do záhradky (t.j. počet prechodov záhradkou), pričom pri výpočte sa uplatňuje nepriama úmernosť (čím menej vykonaných vstupov, tým

vyššia fitness hodnota jedinca). Táto skutočnosť prispieva k celkovej fitness hodnote sumou *1.0 / (pocet_vstupov * 10)* a pomáha odlíšiť od seba dobré riešenie od lepšieho. Posledným, ale nemenej dôležitým parametrom sú numerické označenia jednotlivých lokalít na periférii záhradky predstavujúcich vstupné súradnice mnícha, ktoré celkovú fitness hodnotu navyšujú o sumu ekvivalentnú *sucet_numerickych_oznaceni * 0.00001*. Po realizácii výpočtov sa výsledná fitness hodnota pred uložením zaokrúhli, konkrétne presnosťou 5 desatinných miest. Obr. 2.5 obsahuje kompletný proces výpočtu fitness hodnoty jedinca.

```
entity.FitnessValue = Math.Round(portionFitness + 1.0 / (double) (order * 10) + perimSum * 0.00001, 5);
```

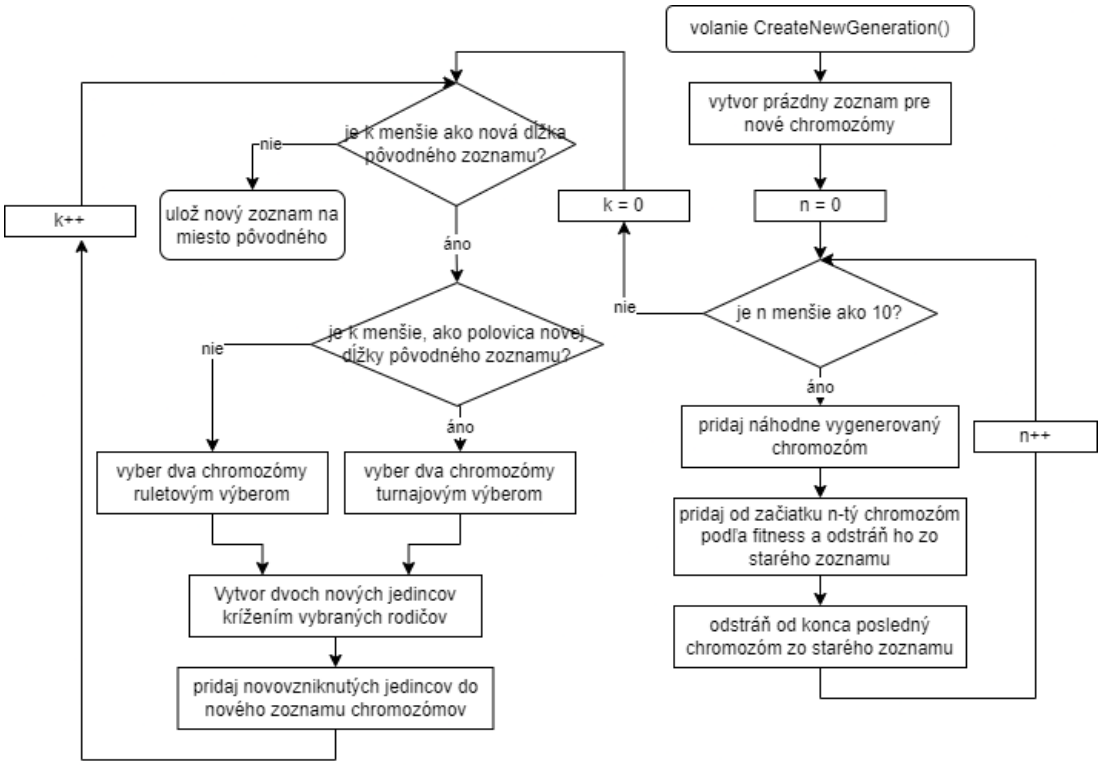
Obr. 2.5 Vzorec na kalkuláciu celkovej fitness hodnoty zvoleného jedinca

2.4.4 Tvorba n-tej generácie jedincov

Princípiálnym cieľom genetického algoritmu je nájdenie čo najlepšieho riešenia pre danú problematiku, a to systematickou tvorbou nových generácií jedincov (riešení). Na uskutočnenie tejto požiadavky je kľúčová prvotná generácia, ktorej ekzaktný postup zhotovenia je opísaný v podkapitole 2.4.2. Dôležitým pravidlom je, aby nová generácia vznikla vždy na základe bezprostredne predchádzajúcej generácie, a to kombináciou nasledujúcich možností:

- *Okamžitý presun istých jedincov s najlepšou fitness hodnotou do novej generácie.*
- *Kríženie dvoch náhodne zvolených jedincov zámenou niektorých ich génov.*
- *Mutácia, kde sa čiastočne upraví hodnota náhodne zvolených génov vrámci jedinca.*

V tejto implementácii genetického algoritmu sa využívajú všetky tri vyššie spomenuté stratégie aj ich kombinácie v čiastočne pozmenenej podobe. Vývojový diagram metódy *CreateNewGeneration()* je zobrazený na obr. 2.6.



Obr. 2.6 Vývojový diagram metódy tvorby novej generácie jedincov

Na začiatku tvorby novej generácie jedincov je nevyhnutné, aby mala metóda *CreateNewGeneration()* k dispozícii zoradený zoznam jedincov predchádzajúcej generácie podľa ich fitness hodnoty. Jedna generácia obsahuje presne 80 jedincov, z toho:

- *10 najlepších jedincov je okamžite premiestnených z predchádzajúcej generácie.*
- *Kvôli zachovaniu potrebnej diverzity je náhodne vytvorených 10 nových jedincov.*
- *Vznikne 30 nových jedincov krížením (rodičia sú zvolení turnajovým výberom).*
- *Vznikne ďalších 30 nových jedincov krížením (rodičia sú zvolení ruletovým výberom).*

Za zmienku stojí aj skutočnosť, že ešte pred procesom generovania je z predchádzajúcej generácie selektovaných a odstránených 10 jedincov s najhoršou fitness hodnotou z dôvodu obmedzenia možnosti kríženia a šírenia menej vhodných riešení.

Počas turnajového výberu rodičov (obr. 2.7) sa z predchádzajúcej generácie náhodne zvolia dvaja jedinci, ktorí budú predmetom turnaja. Vyhráva jedinec s vyššou fitness hodnotou a stáva sa z neho prvý rodič. Identickým spôsobom sa zvolí aj druhý rodič.

```
2 references
private int TournamentSelection()
{
    var participants = new List<int>(2);

    for (int counter = 0; counter < 2; counter++)
    {
        participants.Add(_random.Next(0, _numOfChromosomes - 20));
    }

    int tournamentWinner = participants[0];

    foreach (int index in participants)
    {
        if (Chromosomes[index].FitnessValue > Chromosomes[tournamentWinner].FitnessValue) {
            tournamentWinner = index;
        }
    }
    return tournamentWinner;
}
```

Obr. 2.7 Metóda TournamentSelection() triedy Monk

Ruletový výber vytvára populáciu 5 náhodných jedincov, ktorým je priradená hodnota pravedpodobnosti ich zvolenia, pričom spomenutá hodnota je proporcionálna k ich relatívnej fitness hodnote. Je nevyhnutné, aby súčet hodnôt pravedpodobnosti zvolenia všetkých jedincov bol rovný jednej (1.0). Následne sa náhodne generuje desatinná numerická hodnota v rozsahu od 0 do 1 a na pozíciu rodiča je zvolený jedinec, ktorého interval pravedpodobnosti obsahuje túto generovanú hodnotu. Identickým spôsobom sa zvolí aj druhý rodič.

Akonáhle sa uskutočnil výber rodičov, nasleduje vykonanie metódy *PerformCrossover()* triedy *Monk*. Ako už názov metódy napovie, pomocou nej sa uskutoční kríženie rodičov a vytvorenie dvoch nových potomkov, ktorí sa následne stanú súčasťou novej generácie jedincov. Počas kríženia sa najprv vytvoria kópie rodičovských objektov a zvolí sa interval génov na kríženie, pričom začiatočná a konečná hranica intervalu sú volené náhodne. Nasleduje klasická výmena intervalu génov medzi jedincami a vzniknú dvaja potomkovia.

```
2 references
private int RouletteSelection()
{
    var participants = new List<int>(5);
    var probabilities = new List<double>(5);

    double totalFitness = 0, tempFitness = 0;

    for (int counter = 0; counter < 5; counter++)
    {
        participants.Add(_random.Next(0, _numOfChromosomes - 20));
        totalFitness += Chromosomes[participants[counter]].FitnessValue;
    }

    int rouletteWinner = participants[0];

    participants.ForEach(index => probabilities.Add(Math.Round(Chromosomes[index].FitnessValue / totalFitness, 4)));

    double selection = Math.Round(_random.NextDouble(), 4);

    for (int counter = 0; counter < 5; counter++)
    {
        tempFitness += probabilities[counter];

        if (selection <= tempFitness) {
            rouletteWinner = participants[counter];
            break;
        }
    }
    return rouletteWinner;
}
```

Obr. 2.8 Metóda RouletteSelection() triedy Monk

```
1 reference
private (Chromosome first, Chromosome second) PerformCrossover(int first, int second)
{
    var firstEntity = Chromosomes[first].CreateCopy(false);
    var secondEntity = Chromosomes[second].CreateCopy(false);

    var crossoverBegin = _random.Next(0, _numOfGenes / 2);
    var crossoverEnd = _random.Next(_numOfGenes / 2, _numOfGenes);

    for (int counter = crossoverBegin; counter <= crossoverEnd; counter++)
    {
        var temp = firstEntity.Genes[counter];
        firstEntity.Genes[counter] = secondEntity.Genes[counter];
        secondEntity.Genes[counter] = temp;
    }
    return (firstEntity, secondEntity);
}
```

Obr. 2.9 Metóda PerformCrossover() triedy Monk

2.4.5 Pohyb a rozhodovanie mnícha

Okamžite po vstupe do záhradky je mníchovi priradený jeden zo štyroch možných druhov pohybov zadefinovaných ako konštanty v statickej triede *Translations*, pričom konkrétny pohyb záleží od skutočnosti, na ktorej strane záhradky sa vstupná lokalita mnícha nachádza. Na výber druhu pohybu slúži metóda *GetMove()* (obr. 2.10) statickej triedy *Converters*. Pohyb sa vždy volí logicky (ak napríklad mních vstupuje z pravej strany, jeho počiatočný pohyb bude doľava, ak vstupuje zhora, jeho počiatočný pohyb bude smerom dole).

Mních sa pohybuje zvoleným smerom dovtedy, kým nenatrafí na prekážku (kameň, už pohrabané políčko alebo list, ktorý ešte nie je možné pozbierať) alebo úspešne neukončí svoj ťah na druhej strane záhradky. Ak objaví prekážku a existuje ešte smer, kam by mohol pokračovať vo svojej ceste, pomocou metódy *FindNewMove()* triedy *Garden* sa zvolí nový smer pohybu (ak existuje viacero dovolených smerov, náhodne sa zvolí jeden z nich) a mních sa bude týmto smerom hýbať, kým znova nenatrafí na prekážku alebo úspešne neskončí svoj ťah. Jeho ťah sa končí neúspechom, ak sa ocitne v slepej uličke. V tomto momente je cyklus tvorby ciest podľa génov súčasného jedinca prerušený a nasleduje ďalší jedinec.

```
1 reference
public static (int, int) GetMove(int width, int height, (int X, int Y) coords)
{
    if (coords.Y == 0) {
        return Translations._MoveDown;
    }
    else if (coords.Y == height - 1) {
        return Translations._MoveUp;
    }
    else if (coords.X == 0) {
        return Translations._MoveRight;
    }
    else if (coords.X == width - 1) {
        return Translations._MoveLeft;
    }
    return (0, 0);
}
```

Obr. 2.10 Metóda GetMove() statickej triedy Converters

2.5 Štruktúra vstupných súborov

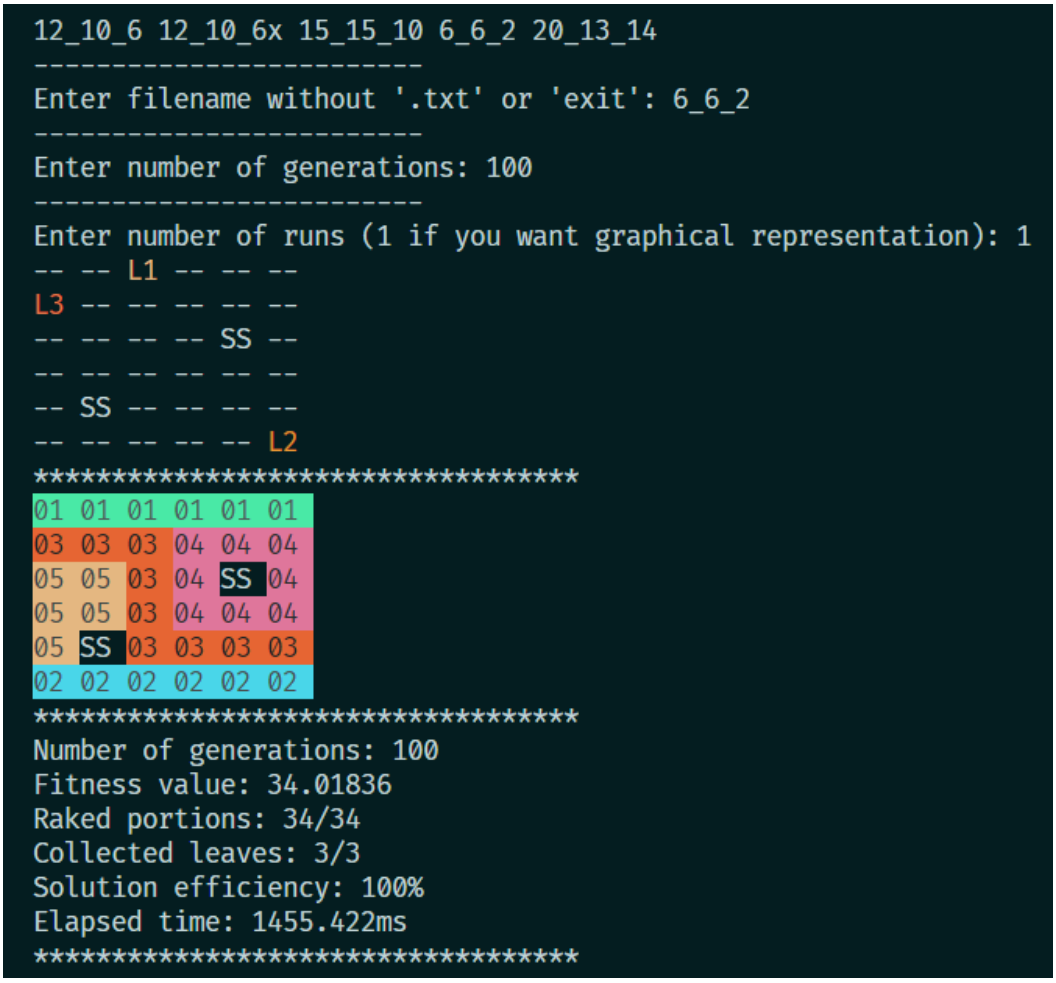
Vstupné rozloženia jednotlivých prvkov v záhradke, ako aj horizontálne a vertikálne rozmery záhradky sú uložené v súboroch formátu *.txt* vrámci adresára *inputs*. Každý vstupný súbor má názov rovnakého formátu: *cislo1_cislo2_cislo3.txt*, kde *cislo1* označuje šírku záhradky, *cislo2* reprezentuje výšku záhradky a *cislo3* udáva počet kameňov nachádzajúcich sa v záhradke. Vnútorná štruktúra súborov je rovnako pomerne jednoznačná: na začiatku súboru sa nachádzajú dvojice v tvare *cislo1_cislo2* predstavujúce súradnice jednotlivých kameňov v záhradke, kde *cislo1* udáva horizontálnu a *cislo2* vertikálnu súradnicu kameňa. Nasleduje znak „-“ oddeľujúci kamene od listov. Záznamy listov sú identické so záznamami kameňov, s jediným rozdielom, a to tretím údajom poukazujúcim na farbu daného listu (*Y* pre žlté, *O* pre oranžové a *R* pre červené listy).

```
1   5_3
2   2_5
3   -
4   3_1_Y
5   6_6_O
6   1_2_R
```

Obr. 2.11 Ukážka štruktúry vstupného súboru 6_6_2.txt

2.6 Používateľské rozhranie programu

Riadenie vykonávania programu sa uskutočňuje zásluhou elementárneho používateľského rozhrania v konzole (obr. 2.12). Používateľ v ňom má možnosť si zvoliť vstupný súbor zadaním presného mena, bez prípony *.txt*, zadefinovať výsledný počet vytvorených generácií riešení a taktiež zvoliť počet behov programu. Ak si používateľ vyberie iba jeden beh, po zadaní spomenutých údajov program vykreslí počiatočný vzhľad záhradky s pozíciami kameňov a listov a začne hľadať riešenie. Akonáhle sa vytvorí posledná očakávaná generácia, program vykreslí nájdené najlepšie riešenie spolu s doplňujúcimi údajmi – maximálnu fitness hodnotu, počet pokrytých políčok z celkového počtu, percentuálnu mieru úspešnosti a počet úspešne pozbieraných listov. Program sa bude vykonávať dovtedy, kým používateľ pri výzve zadať názov súboru nevpíše „exit“.



Obr. 2.12 Používateľské rozhranie programu a vykreslenie nájdeného riešenia

3 Testovanie algoritmu

S cieľom poukázať na správnu funkčnosť navrhnutého riešenia bol vykonaný rozmanitý počet testov, pričom aj samotný program je navrhnutý spôsobom, aby používateľ dokázal identickým druhom testov podrobiť algoritmus aj sám.

3.1 Spôsob testovania

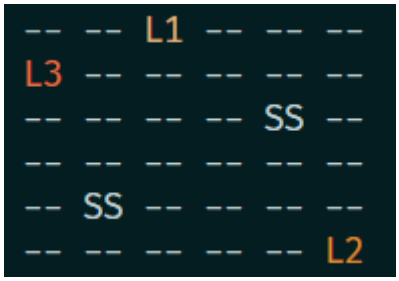
Na realizáciu testovania boli využité vstupné *.txt* súbory so súradnicami reprezentujúcimi rozmiestnenie prvkov v záhradke. Boli vykonané dva fundamentálne druhy testov – test s jedným behom programu, ktorého hlavnou úlohou je poskytnúť používateľovi optimálne riešenie zvoleného problému, a test s viacerými behmi programu, kde sa kladie dôraz hlavne na analýzu funkčnosti a efektivity algoritmu ako takého.

3.1.1 Testovanie s jedným behom programu

V prípade testovania s jedným behom programu sa používateľovi po zhotovení poslednej generácie jedincov vykreslí v konzole výsledná štruktúra záhradky spolu s relevantnými údajmi, medzi ktorý patrí počet vytvorených generácií, maximálna fitness hodnota, počet pokrytých políčok z počtu všetkých možných, počet pozbieraných listov (ak vstupná záhradka obsahuje listy) a efektivitu nájdeného riešenia, vyjadrenú v percentách. Okrem toho sa vytvorí aj záznam maximálnej fitness hodnoty každej generácie jedincov v súbore *fitness.txt*, ktorý je možné použiť napríklad na zhotovenie grafu súvislosti vývoja fitness hodnoty od počtu generácií.

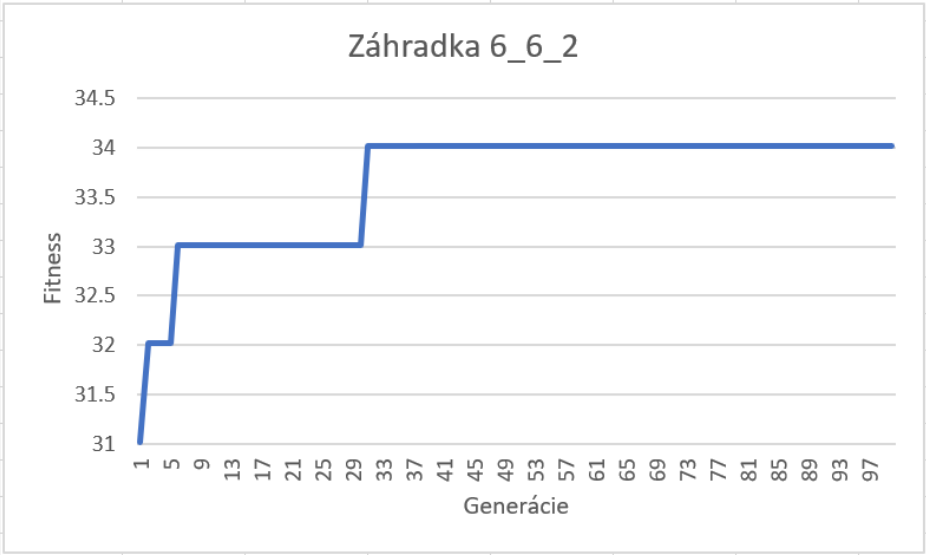
Vstupný súbor 6_6_2

Uvedený súbor, ako to už z jeho názvu vyplýva, obsahuje záhradku rozmerov 6x6 s dvoma prekážkami – kameňmi. Okrem nich sa v nej nachádzajú 3 listy, jeden z každej dostupnej farby. Záhradka obsahuje 34 zaplniteľných políčok.



Obr. 3.1 Počiatočný vzhľad záhradky 6_6_2

Na obr. 3.2 je zachytený vývoj maximálnej fitness hodnoty pre 100 vytvorených generácií. Z dostupných skutočností vyplýva, že algoritmus je schopný s pomocou relatívne nízkeho počtu generácií nájsť najlepšie možné riešenie (v tomto príklade sa mníchovi podarilo pohrabať všetkých 34 políčok už v 31. generácii).



Obr. 3.2 Vývoj maximálnej fitness hodnoty záhradky 6_6_2 pri 100 generáciách

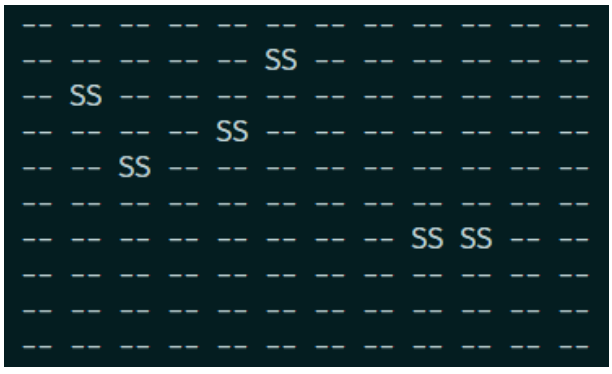
Údaje v tabuľke 3.1 poukazujú na rovnaký výsledok, ako graf na obr. 3.2. Mníchovi sa podarí pokryť všetky dostupné políčka medzi 10. a 50. generáciou jedincov, vo zvyšných generáciách sa maximálna fitness hodnota už iba doľaďuje.

6_6_2	generácie	max fitness	políčka	listy	úspešnosť	čas
1	1	32.01349	32/34	3/3	94.12%	35.888ms
2	10	33.01406	33/34	3/3	97.06%	142.794ms
3	50	34.01655	34/34	3/3	100%	617.045ms
4	100	34.01848	34/34	3/3	100%	1218.654ms

Tab. 3.1 Odmerané a zistené údaje pre záhradku 6_6_2

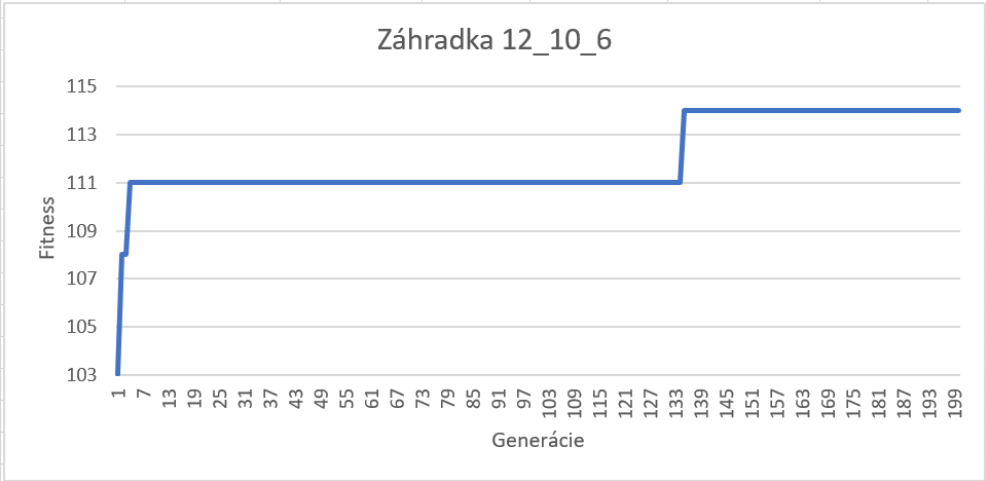
Vstupný súbor 12_10_6

Tento vstupný súbor obsahuje identickú záhradku, aká je zobrazená na webovom sídle predmetu. Má rozmery 12x10 a 114 zaplniteľných políčok, obsahuje 6 prekážok a žiadne listy.



Obr. 3.3 Počiatočný vzhľad záhradky 12_10_6

Na obr. 3.4 je zachytený vývoj maximálnej fitness hodnoty pre 200 vytvorených generácií. Z dostupných skutočností vyplýva, že menej komplikované záhradky stredných veľkostí potrebujú približne 130-150 generácií na nájdenia najlepšieho možného riešenia (v tomto prípade sa mníchovi podarilo pokryť všetkých 114 políčok v 136. generácii).



Obr. 3.4 Vývoj maximálnej fitness hodnoty záhradky 12_10_6 pri 200 generáciách

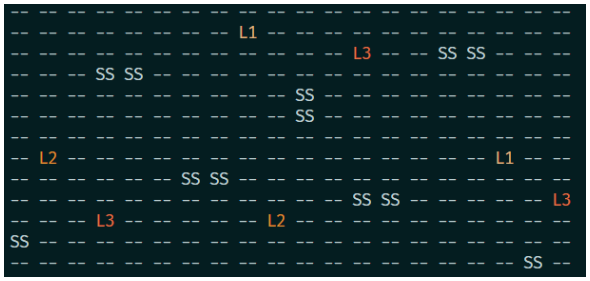
Údaje v tabuľke 3.2 poukazujú na neustále vylepšovanie najvyhovujúcejšieho riešenia s narastajúcim počtom generácií, pričom nárast, ako to je možné pozorovať aj na obr. 3.4, je strmejší pri nižších počtoch generácií a postupne sa spomalí, akonáhle sa fitness hodnota začne blížiť k optimálnemu riešeniu.

12_10_6	generácie	max fitness	políčka	listy	úspešnosť	čas
1	1	101.0141	101/114	0/0	88.60%	63.967ms
2	10	107.01547	107/114	0/0	93.86%	193.99ms
3	50	111.01314	111/114	0/0	97.37%	730.702ms
4	100	112.01709	112/114	0/0	98.25%	1204.98ms
5	200	114.01706	114/114	0/0	100%	2448.209ms

Tab. 3.2 Odmerané a zistené údaje pre záhradku 12_10_6

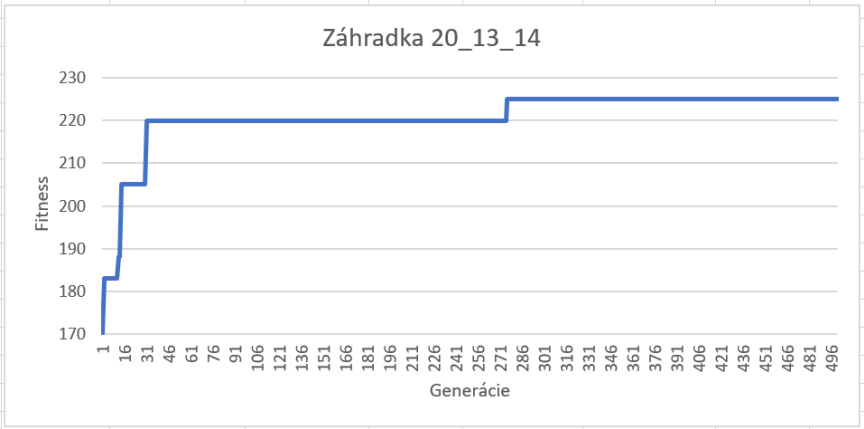
Vstupný súbor 20_13_14

Vo vyššie uvedenom súbore sídli záhradka rozmerov 20x13 s 248 zaplniteľnými políčkami, 14 kameňmi a siedmimi listami rôznych farieb.



Obr. 3.5 Počiatočný vzhľad záhradky 20_13_14

Na obr. 3.5 je zachytený vývoj maximálnej fitness hodnoty pre 500 vytvorených generácií. Z dostupných skutočností vyplýva, že pri záhradkách väčších rozmerov naplnených kameňmi a rôznymi listami nestačí ani uvedený počet generácií, aby sa našlo najideálnejšie riešenie (mních pri 500 generáciách pokryl 225 políčok z 248 možných).



Obr. 3.6 Vývoj maximálnej fitness hodnoty záhradky 20_13_14 pri 500 generáciách

Údaje v tabuľke 3.3 poukazujú na skutočnosť, že pri rozsiahlych a komplikovaných záhradkách je extrémne náročné nájsť dokonalé riešenie problému, vo väčšine prípadov nemožné. Treba sa preto uspokojiť s dostatočne vhodným riešením, ktorý spĺňa používateľom stanovené kritériá.

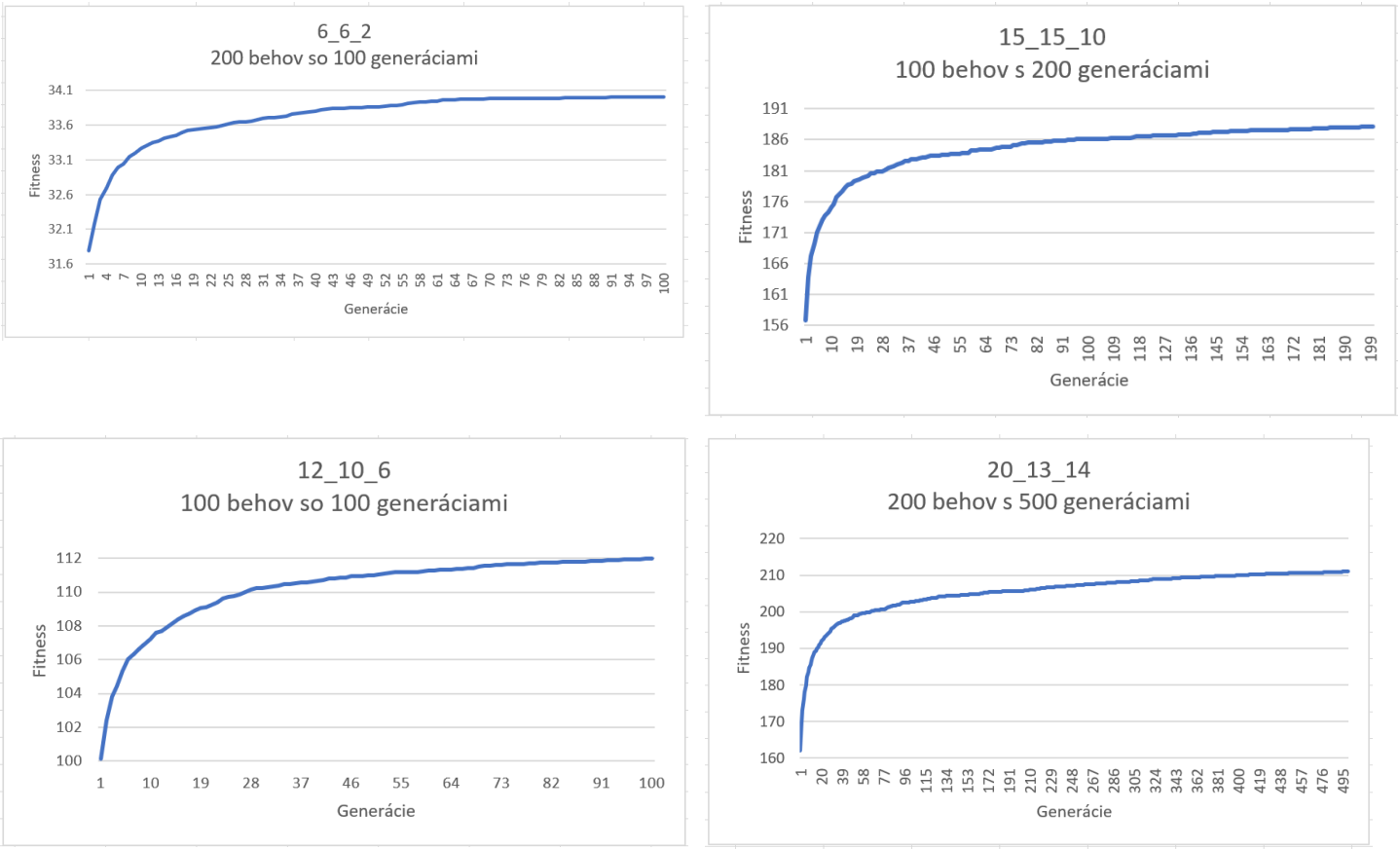
20_13_14	generácie	max fitness	políčka	listy	úspešnosť	čas
1	1	144.01197	144/248	1/7	58.06%	49.427ms
2	100	205.01993	205/248	5/7	82.66%	1171.374ms
3	200	213.01544	213/248	5/7	85.89%	2287.719ms
4	300	216.01988	216/248	6/7	87.10%	3409.117ms
5	500	225.02105	225/248	6/7	90.73%	5696.599ms

Tab. 3.3 Odmerané a zistené údaje pre záhradku 20_13_14

3.1.2 Testovanie s viacerými behmi programu

Tento druh testovania slúži na analýzu funkčnosti a efektivity navrhutého genetického algoritmu, pričom počas jeho behu sa používateľovi v konzole zobrazuje iba informácia týkajúca sa toho, v ktorom behu sa program práve nachádza. Výstupom uvedeného testovania je aritmetický priemer maximálnych fitness hodnôt všetkých generácií, pričom výpočet prebieha nasledovne:

- *Pri prvom behu sa do zoznamu uložia maximálne fitness hodnoty z každej generácie*
- *Pri každom ďalšom behu sa k hodnotám v zozname pripočítajú nové maximálne fitness hodnoty z každej zodpovedajúcej generácie*
- *Po skončení všetkých behov sa vypočíta aritmetický priemer maximálnych fitness hodnôt pre každú generáciu a hodnoty sa zaznamenajú v súbore fitness.txt*



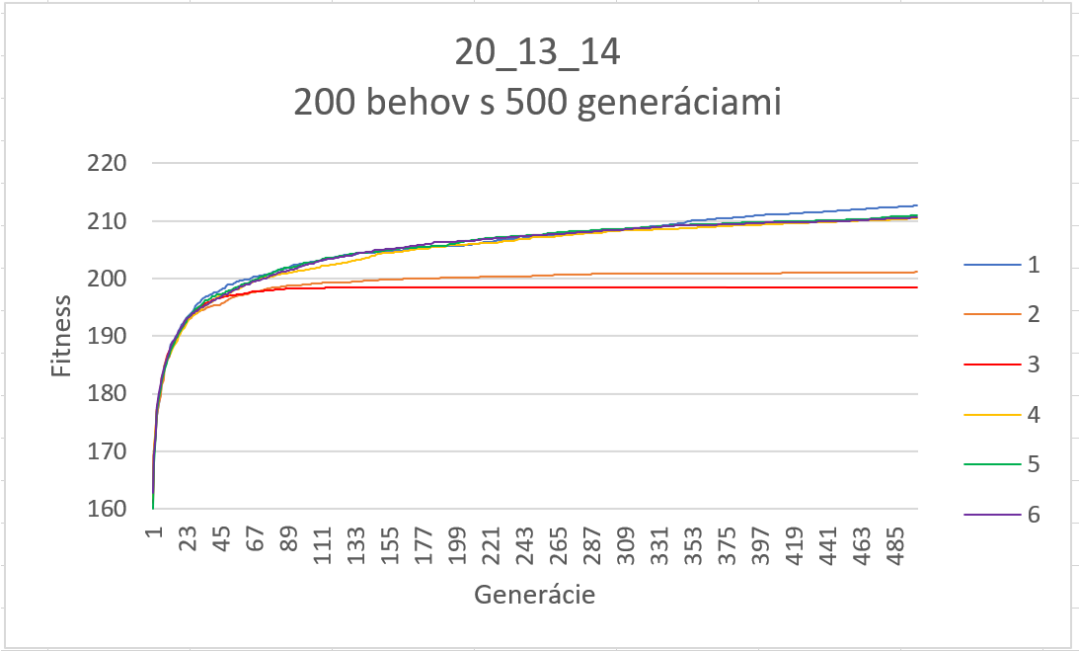
Obr. 3.7 Grafy pre 4 vybrané záhradky zobrazujúce vývoj maximálnej fitness hodnoty pre každú vytvorenú generáciu

Z grafov zobrazených na obr. 3.7 vyplýva niekoľko zaujímavých skutočností. Graf každej zo zvolených záhradiek konverguje v približne rovnakej pozícii, pričom pri priblížení sa k najlepšiemu riešeniu (alebo riešeniu blízkemu najlepšiemu) sa nárast fitness hodnoty spomalí, až sa nakoniec úplne zastaví. Pri každom uvedenom grafe sa rovnováha medzi dostatočne optimálnou fitness hodnotou a časom vykonávania algoritmu (použitými výpočtovými prostriedkami potrebnými na vykonávania algoritmu) nachádza v intervale medzi 60. a 100. generáciou. Pri skorších generáciách fitness hodnota ešte nie je dostatočne vyhovujúca, pri neskorších, naopak, fitness hodnota narastá príliš pomaly v porovnaní s veľkým počtom využitých výpočtových prostriedkov.

3.2 Porovnanie algoritmov s rôznymi parametrami

Pri každom genetickom algoritme existuje nespočetne veľa možností optimalizácie, začínajúc správnou voľbou použitých génov a končiac optimálnou kombináciou spôsobov tvorby nových jedincov pri zhotovovaní budúcich generácií. S cieľom nájsť algoritmus, ktorý dokáže nájsť najlepšie riešenia pre rôznorodé typy záhradiek, bolo vyskúšaných celkovo 6 spôsobov tvorby novej generácie, pričo niektoré sa líšili iba v menších detailoch. Rovnako sa otestovalo, či majú mutácie pozitívny alebo negatívny vplyv na vývoj maximálnej fitness hodnoty. Jednotlivé spôsoby sa nachádzajú v zozname nižšie:

1) *10 najlepších jedincov je prekopírovaných, je vytvorených 10 nových náhodných jedincov, pridá sa 30 jedincov krížením pomocou turnajového výberu, 30 jedincov krížením pomocou ruletového výberu a vykonajú sa mutácie.*
2) *10 najlepších jedincov je prekopírovaných, pridá sa 35 jedincov krížením pomocou turnajového výberu a 35 jedincov krížením pomocou ruletového výberu.*
3) *Všetkých 80 jedincov je vytvorených krížením (polovica ruletovým a polovica turnajovým krížením).*
4) *10 najlepších jedincov je prekopírovaných, je vytvorených 10 nových náhodných jedincov a zvyšných 60 jedincov vznikne krížením pomocou turnajového výberu.*
5) *10 najlepších jedincov je prekopírovaných, je vytvorených 10 nových náhodných jedincov a zvyšných 60 jedincov vznikne krížením pomocou ruletového výberu.*
6) *Rovnako ako prvý bod, ale neaplikujú sa žiadne mutácie.*



Obr. 3.8 Graf zobrazujúci 6 rôznych spôsobov tvorby novej generácie jedincov

Zistené skutočnosti v sebe obsahujú mnoho cenných poznatkov. Jednoznačne najhorším riešením je, keď sú všetci jedinci vytvorení krížením (č.3). Toto riešenie síce v počiatočnej časti vykonávania algoritmu pomerne narastá, ale okolo 20. generácie sa spomalí a úplne zastaví. Medzi horšie riešenia patrí aj to, ktoré v sebe neobsahuje žiadnych novovytvorených náhodných jedincov (č.2). Naopak, pomocou č.6 sa zistilo, že mutácia v kontexte tejto problematiky nezohráva príliš rozsiahlu úlohu, bez nej je totižto nárast fitness hodnoty porovnateľný s inými spôsobmi generovania.

Po vykonaní testu sa tiež zistilo, že pri vyšších počtoch generácií neexistujú badateľné zmeny medzi ruletovým (č.5) a turnajovým výberom (č.4). Isté rozdiely sa objavujú iba v prvej polovici generácií, kde ruletový výber podáva o niečo lepší výkon.

Najlepší výkon podáva spôsob č.1, ktorý je z tohto dôvodu aj základom implementácie algoritmu. Po celý čas vykonávania programu narastá rýchlejšie, ako väčšina iných spôsobov, a nakoniec dosiahne najlepšie riešenie.

3.3 Možnosti rozšírenia a optimalizácie riešenia

V súčasnom algoritme sa pri kalkulácii fitness hodnoty každého jedinca generuje nové dvojdimenzionálne pole záhradky a napĺňa objektmi. Táto skutočnosť pri vyššom počte behov (nad 100) spolu s extenzívnym počtom generácií (nad 300) vo veľkej miere spomaľuje vykonávanie programu. Z hľadiska optimalizácie by bolo vhodné, keby objekty jednotlivých druhov políčok záhradky (kamene, listy, prázdne políčka…) boli nahradené niektorým zo základných dátových typov (napríklad *string*, ktorý by obsahoval informáciu o type políčka).

Pomerne priamočiarym spôsobom rozšírenia programu by bolo pridanej externej knižnice na automatické generovanie grafov. Súčasný program je schopný len uložiť výstupné údaje do súboru typu *.txt*, kde ich má používateľ k dispozícii, ale ak potrebuje zhotoviť graf, musí na to využiť niektorý z na to špecializovaných softvérových výrobkov.

Posledným spôsobom rozšírenia programu by bolo umožnenie používateľovi manuálne zadávať rozmery záhradky a súradnice prekážok z používateľského zadania.