Umelá inteligencia

Zadanie č.2a – Zenová záhrada

Tomáš Brček

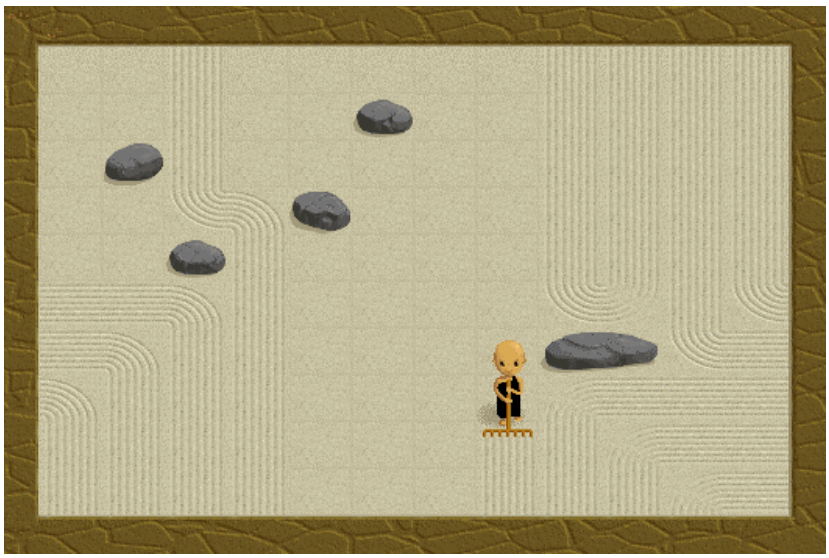
FIIT STU, Bratislava

2023

Obsah

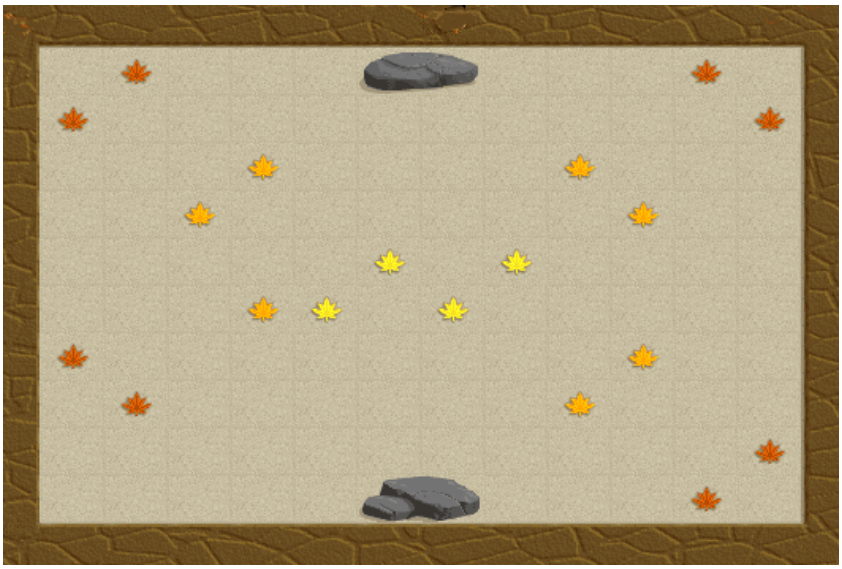
Opis úlohy

Zenová záhradka je plocha, kde hrubý piesok (jemné kamienky) tvorí podklad, avšak obsahuje aj nepohyblivé väčšie prvky, ako sú kamene. Mních je povinný usporiadať piesok v záhradke pomocou hrablí tak, aby vytvoril pruhy podobné tým na priloženom obrázku.



Tieto pruhy môžu ísť iba vodorovne alebo zvislo a nikdy nie diagonálne. Mních začína vždy na okraji záhrady a vytvára rovný pruh piesku až k druhému okraju alebo kým nenarazí na prekážku. Na okraji záhrady (mimo nej) môže chodiť, ako sa chce. No ak sa stretne s prekážkou, ako je kameň alebo už upravený piesok, musí sa otočiť, ak je to možné. V prípade, že má voľný priestor na otočenie vľavo alebo vpravo, môže si vybrať smer. Ak je voľný iba jeden smer, otočí sa týmto smerom. Ak nemá žiadnu možnosť otočenia, hra končí. Úspešnou hrou sa považuje situácia, v ktorej mních dokáže na základe uvedených pravidiel pohrabať celú záhradu, prípade maximálny možný počet políčok. Výsledkom hry je usporiadanie záhrady podľa mníchových prechodov.

Túto úlohu je možné ešte zložitejšie rozšíriť tým, že mních bude zároveň zbierať spadnuté listy. Listy musí zbierať v určenom poradí, začínajúc žltými, potom pomarančovými a nakoniec červenými. Listy, na ktoré ešte nemá povolené siahať, budú predstavovať pevné prekážky.



Genetický algoritmus

Na riešenie tohto zadania som použil klasický genetický algoritmus.

Na začiatku, po načítaní rozmerov záhrady a pozícií kameňov, sa vytvorí prvá generácia jedincov s náhodne nastavenými génmi.

```
#funkcia generujúca náhodné gény pre chromozómy prvej generácie
def generate_genes(self):
    number_of_genes = self.garden.rows + self.garden.cols       #počet týchto génov sa rovná polovici obvodu záhrady
    genes = []
    used_numbers = []       #pole na kontrolu, či sme už také číslo použili
    i = 0
    while i < number_of_genes:
        o = 2*self.garden.rows + 2*self.garden.cols     #obvod záhrady
        pos = randint(0, o-1)           #náhodná pozícia na obvode záhrady
        #ak sa takýto gén ešte nevyskytuje, vytvorí sa a uloží sa medzi gény daného chromozómu
        if pos not in used_numbers:
            genes.append(Gene(pos, self.garden.rows, self.garden.cols))
            used_numbers.append(pos)
            i += 1

    return genes
```

Následne sa pre každého jedinca (chromozóm) vytvorí matica prechodov mnícha, a uloží sa do atribútov daného chromozómu. Po vytvorení cesty funkciou *make_path()* v triede *Chromosome* sa zavolá fitness funkcia *fitness_func()*, ktorá na základe matice prechodov mnícha pridelí fitness hodnotu, t.j. počet pohrabaných políčok záhrady.

```
#funkcia, ktorá zisťuje fitness hodnotu daného chromozómu, teda počet pohrabaných políčok
def fitness_func(self):
    empty = 0
    for i in range(self.garden.rows):
        for j in range(self.garden.cols):
            if self.garden.garden[i][j] == 0 or self.garden.garden[i][j] in ["Z", "P", "C"]:
                empty += 1
    return self.garden.rows * self.garden.cols - len(self.garden.rocks) - empty
```

Na základe tohto ohodnotenia sa vyberú jedince na tvorbu novej generácie – kríženie a takto vytvorení jedinci môžu s určitou pravdepodobnosťou aj mutovať.

Výber jedincov, kríženie aj mutácie sú opísané v podkapitolách nižšie.

Vytvorí sa nová generácia a to sa vykonáva dokola, až kým sa nepodarí pokryť všetky políčka alebo sa dosiahne stanovený počet nových generácií (1000 generácii).

Pohyb mnícha

Mních začína vždy na okraji záhradky a ťahá rovný pás až po druhý okraj alebo po prekážku. Mimo záhradky môže chodiť ako chce. Ak však príde k prekážke – kameňu alebo už pohrabanému piesku – musí sa otočiť, ak má kam. Ak má voľné smery vľavo aj vpravo, je jeho vec, kam sa otočí. Ak má voľný len jeden smer, otočí sa tam. Ak sa nemá kam otočiť, je koniec hry.

Výber jedincov na kríženie

```python
#ruletový výber
if selectionNum < 0.33:
    total = sum(item[0] for item in chromosomes)
    randNum1 = randint(0, total)
    randNum2 = randint(0, total)
    for_total = 0
    x = 0
    for chrom in chromosomes:
        x += 1
        for_total += chrom[0]
        if(for_total >= randNum1) and parent1 == None:
            parent1 = chrom[1]
        if(for_total >= randNum2) and parent2 == None:
            parent2 = chrom[1]
    new_generation.append(crossing(parent1, parent2, deepcopy(garden)))

#turnajový výber rodičov
elif 0.33 < selectionNum < 0.66:
    parent1 = None
    parent2 = None
    sublist = sample(chromosomes, 10)
    sublist.sort(key=lambda a: a[0], reverse=True)
    parent1 = sublist[0][1]
    parent2 = sublist[1][1]

    new_generation.append(crossing(parent1, parent2, deepcopy(garden)))

#výber podľa poradia
else:
    subtract = 1
    total = ((1 + len(chromosomes))*len(chromosomes))//2        #vypocet suctu n clenov aritmetickej postupnosti
    randNum1 = randint(0, total)
    randNum2 = randint(0, total)
    for i in range(len(chromosomes), 0, -1):
        randNum1 -= subtract
        randNum2 -= subtract
        subtract += 1
        if(randNum1 <= 0) and parent1 == None:
            parent1 = chromosomes[i-1][1]
        if(randNum2 <= 0) and parent2 == None:
            parent2 = chromosomes[i-1][1]
    new_generation.append(crossing(parent1, parent2, deepcopy(garden)))
```

Na obrázku vyššie môžeme vidieť časť kódu z funkcie *generate_gen(),* ktorá slúži na generovanie nasledujúcej generácie.

Na výber rodičov som použil až 3 metódy výberu, aby som dosiahol čo najrozmanitejšie obsadenie chromozómov v každej generácií. Každá z nich nastáva s pravdepodobnosťou 33%.

Metódy výberu, ktoré som použil:

- **ruletový výber**
  - dáva pravdepodobnosť každému jedincovi v populácii na základe jeho hodnoty fitness, s vyšším hodnotením fitness má jedinec vyššiu pravdepodobnosť byť vybraný ako rodič pre reprodukciu
- **turnajový výber**
  - turnajový výber vyberá náhodne 10 jedincov z populácie a následne vyberá jedincov s najvyšším hodnotením fitness z tejto podmnožiny ako rodičov/
- **výber podľa poradia**
  - vykonáva sa na základe poradia jedincov v populácii po zoradení podľa ich hodnoty fitness

Kríženie

```python
#funkcia zabezpečujúca kríženie dvoch chromozómov
def crossing(chromosome1, chromosome2, garden):
    #vygenerovanie náhodných čísel
    mutateNum = random()
    crossProb = random()
    newChrom = Chromosome(deepcopy(garden))     #nový chromozóm
    pivotPoint = randrange(len(chromosome1.genes))              #pivotný prvok pre gény
    pivotOrient = randrange(len(chromosome1.orient_genes))      #pivotný prvok pre orientačné gény
    #ak je číslo crossProb medzi 0,4 a 0,8 nastane toto kríženie
    if 0.4 < crossProb < 0.8:
        #vyberajú sa náhodné gény postupne od obidvoch rodičov
        newChrom.genes = []
        for i in range(len(chromosome1.genes)):
            newChrom.genes.append(choice((chromosome1.genes[i], chromosome2.genes[i])))
        for i in range(len(chromosome1.orient_genes)):
            newChrom.orient_genes.append(choice((chromosome1.orient_genes[i], chromosome2.orient_genes[i])))
    elif crossProb <= 0.4:
        #jedna časť sa berie od rodiča1 a druhá časť od rodiča2
        newChrom.genes = chromosome1.genes[:pivotPoint] + chromosome2.genes[pivotPoint:]
        newChrom.orient_genes = chromosome1.orient_genes[:pivotOrient] + chromosome2.orient_genes[pivotOrient:]
    else:
        #žiadne kríženie, berú sa všetky gény od jedného rodiča
        newChrom.genes = choice((chromosome1.genes, chromosome2.genes))
        newChrom.orient_genes = choice((chromosome1.orient_genes, chromosome2.orient_genes))

    #pravdepodobnosť, že pre daný chromozóm môže nastať mutácia je 50%
    if mutateNum < 0.5:
        mutation(newChrom)

    return newChrom
```

Po výbere dvoch chromozómov nastáva kríženie. V mojom riešení som použil 2 metódy kríženia, pričom každá z nich nastáva s pravdepodobnosťou 40%:

1. **<u>náhodné gény od rodičov</u>**
   - postupne sa prechádzajú gény a pre každú dvojicu sa náhodne vyberie jeden z nich
2. **<u>na základe pivotného prvku</u>**
   - jedna časť génov (od začiatku po pozíciu pivotného prvku) sa berie od jedného rodiča
   - druhá časť génov (od pozície pivotného prvku po koniec) sa berie od druhého rodiča

S pravdepodobnosťou 20% nedochádza ku kríženiu a náhodne sa vyberú gény jedného z rodičov.

Pre každý chromozóm môže s pravdepodobnosťou 50% nastať aj **mutácia**.

Mutácie

```python
#funkcia zabezpečujúca mutácie génov
def mutation(chromosome):
    #for-cyklus prechádza všetky gény
    for i in range(len(chromosome.genes)):
        number = random()       #vygeneruje náhodné číslo
        #s pravdepodobnosťou 10% bude daný gén mutovať
        if number < 0.1:
            o = 2*chromosome.garden.rows + 2*chromosome.garden.cols
            pos = randint(0, o-1)       #vygeneruje sa nová náhodná pozícia
            chromosome.genes[i] = Gene(pos, chromosome.garden.rows, chromosome.garden.cols)

    #for-cyklus prechádza všetky gény otáčania
    for i in range(len(chromosome.orient_genes)):
        number = random()
        #s pravdepodobnosťou 0,5% bude daný gén mutovať
        if number < 0.005:
            if chromosome.orient_genes[i] == "r":
                chromosome.orient_genes[i] = "l"
            else:
                chromosome.orient_genes[i] = "r"
```

Postupne sa prechádzajú gény chromozómu a s pravdepodobnosťou 10% môže každý z génov zmutovať. To znamená, že sa vygeneruje nové náhodné číslo na obvode záhrady.

Podobne je to aj pre orientačné gény, ktoré hovoria o tom, kam sa má mních otočiť v prípade, že došlo ku kolízii a mních má na výber. Tu je pravdepodobnosť mutácie len 0,5%. Ak gén zmutuje, zmení sa na opačný smer otočenia.

Triedy

V mojom riešení som implementoval nasledujúce triedy:

1. **Garden** - trieda reprezentujúca záhradu, ktorú má mních svojimi prechodmi pohrabať

```
class Garden:
    #konštruktor triedy Garden
    def __init__(self, rows, cols, rocks, leafs):
        self.rows = rows                #počet riadkov
        self.cols = cols                #počet stĺpcov
        self.rocks = rocks              #pole obsahujúce súradnice kameňov v záhrade
        self.leafs = leafs              #dvojrozmerné pole obsahujúce súradnice listov, prvé pole - žlté, druhé pole - pomarančové, tretie - červené listy
        #ak zadáme pole listov, spočítajú sa počty jednotlivých farieb listov
        if len(leafs) != 0:
            self.yellow = len(self.leafs[0])
            self.orange = len(self.leafs[1])
            self.red = len(self.leafs[2])
        #inak sú to nuly
        else:
            self.yellow = 0
            self.orange = 0
            self.red = 0
        self.garden = self.generate_garden()        #vygenerovanie záhrady spolu s kameňmi a listami
```

- atribúty:
  - *rows* – počet riadkov
  - *cols* – počet stĺpcov
  - *rocks* – zoznam súradníc, na ktorých sa nachádzajú skaly
  - *leafs* – dvojrozmerné pole obsahujúce súradnice listov
  - *yellow* – počet žltých listov
  - *orange* – počet oranžových listov
  - *red* – počet červených listov
  - *garden* – matica reprezentujúca záhradu
- metódy:
  - *generate_garden()* – slúži na vygenerovanie záhrady s kameňmi a listami
  - *print_garden()* – vypísanie záhrady do terminálu

2. **Gene** - trieda reprezentujúca jednotlivé gény, teda miesta, z ktorých bude mních vychádzať

```
class Gene:
    #konštruktor triedy Gene
    def __init__(self, number, rows, cols):
        self.number = number
        self.rows = rows
        self.cols = cols
        self.pos, self.mov = self.get_position()
```

- atribúty:
  - *number* – číslo reprezentujúce pozíciu na okraji záhrady
  - *rows* – počet riadkov
  - *cols* – počet stĺpcov
  - *pos* – dvojica, v ktorej sú uložené súradnice začiatku, odkiaľ má mních hrabať
  - *mov* – smer, v akom sa mních pohybuje
- metódy:
  - *get_position()* – z atribútu number, získa počiatočné súradnice a smer pohybu

3. **Chromosome** - trieda reprezentujúca jednotlivé chromozómy, teda jedincov z populácie

```
class Chromosome:
    #konštruktor triedy Chromosome
    def __init__(self, garden, first=False):
        self.garden = garden
        self.fitness = 0
        #ak je to chromozóm prvej generácie, tak generujeme gény náhodne
        if first:
            self.genes = self.generate_genes()                  #gény, ako sa bude mních pohybovať
            self.orient_genes = self.gen_orient_genes()         #gény, kam sa mních otočí pri možnosti voľby
        else:
            self.genes = []
            self.orient_genes = []

    def __str__(self):
        result = "("
        for gene in self.genes:
            result += f"{str(gene.number)} "
        return result[:-1]+")"
```

- atribúty:
  - *garden* – záhrada, ktorú má mních pohrabať
  - *fitness* – fitness hodnota chromozómu
  - *genes* – zoznam génov
  - *orient_genes* – zoznam orientačných génov, kam sa má mních otočiť, v prípade kolízie
- metódy:
  - *generate_genes()* - funkcia generujúca náhodné gény pre chromozómy prvej generácie
  - *gen_orient_genes()* - funkcia generujúca gény na rozhodovanie, kam sa má mních otočiť v prípade voľby
  - *make_move()* - metóda zabezpečujúca pohyb
  - *check_move()* - funkcia kontrolujúca, či sa mních môže pohnúť na nasledujúce políčko
  - *check_leaf()* - funkcia zabezpečujúca zbieranie listov a kontrolu, či sa na dané políčko môže mních presunúť
  - *find_dir()* - funkcia, ktorá zisťuje kam sa má mních otočiť v prípade kolízie
  - *make_path()* - funkcia zabezpečujúca prejdenie záhrady
  - *fitness_func()* - funkcia, ktorá zisťuje fitness hodnotu daného chromozómu, teda počet pohrabaných políčok

Testovanie

```python
#funkcia, ktorá rieši danú záhradu
def test_garden(garden, whole_garden):
    generation = []
    best_fitness = 0
    genNum = 0

    #jednu populáciu tvorí 60 chromozómov
    for i in range(60):
        ch = Chromosome(deepcopy(garden), True)
        generation.append(ch)
    try:
        #kým sa nedosiahne riešenie, alebo neprejde 1000 generácii
        while best_fitness != whole_garden and genNum != 1000:
            fitnesses = []
            for ch in generation:
                ch.garden = ch.make_path()
                ch.fitness = ch.fitness_func()
                fitnesses.append((ch.fitness, ch))

            fitnesses.sort(key=lambda a: a[0], reverse=True)        #zoradenia podľa fitness
            best_fitness = fitnesses[0][0]

            #dosiahnuté riešenie - ukončenie cyklu
            if(best_fitness == whole_garden):
                break

            generation = generate_gen(fitnesses, deepcopy(garden))

            #po každých 100 generáciách sa vypisuje stav
            if genNum%100 == 0:
                print(genNum, " -> ", best_fitness, "/", whole_garden)

            genNum += 1

        print("Generation:", genNum)
        fitnesses[0][1].garden.print_garden()
    except KeyboardInterrupt:
        print("Generation:", genNum)
        fitnesses[0][1].garden.print_garden()
```

Na vyriešenie záhrady slúži funkcia *test_garden()*, ktorú môžete vidieť na obrázku vľavo.

Najskôr vytvorí prvú generáciu **60** náhodných **chromozómov.**

Následne pre ne vygeneruje cestu mnícha a ohodnotí každý z chromozómov. Ohodnotené chromozómy následne usporiada podľa hodnoty fitness a pošle ich do funkcie *generate_gen()*, ktorej úlohou je vytvorenie novej generácie.

Tento postup opakuje, kým nepríde k riešeniu, alebo nebol dosiahnutý nastavený počet generácii, v tomto prípade **1000 generácii**.

Na otestovanie môjho riešenia som si vytvoril 4 záhrady rôznych náročností, na ktorých skúsim mojim algoritmom nájsť riešenie.

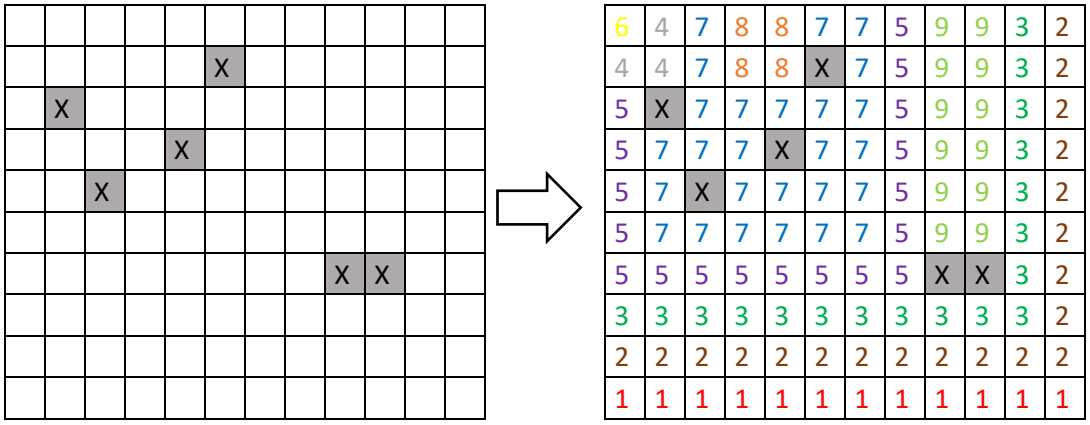
Jednotlivé záhrady budem testovať s rôznymi hodnotami parametrov, budem meniť pravdepodobnosti kríženia, výberu rodičov, a tiež aj pravdepodobnosť mutácii.

	Náhodné gény [%]	Pivotný prvok [%]	Bez kríženia [%]	Ruletový výber [%]	Turnajový výber [%]	Výber podľa poradia [%]	Mutácie génov [%]
Test 1	40	40	20	33,3	33,3	33,3	10
Test 2	50	30	20	30	50	20	5
Test 3	30	50	20	50	30	20	8
Test 4	20	30	50	30	20	50	10

Na nasledujúcich prípadoch otestujem, všetky pripravené záhrady a na záver porovnám výsledky.

Záhrada 1

Jedno z nájdených riešení:



Test č. 1:

- počet generácii: 89
- čas potrebný na nájdenie riešenia: 8.588s

Test č.2:

- počet generácii: 52
- čas potrebný na nájdenie riešenia: 4.469s

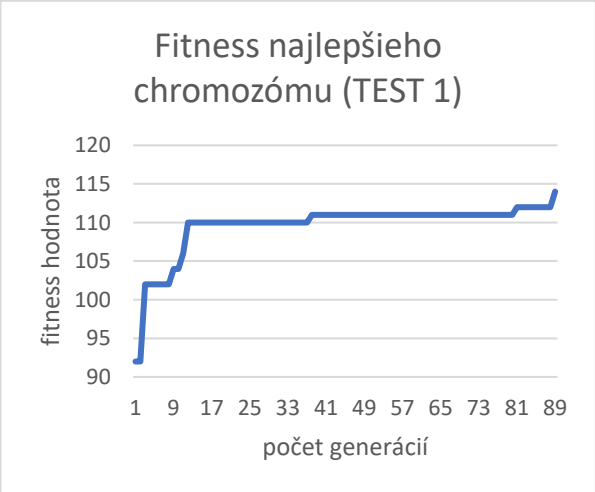
Test č.3:

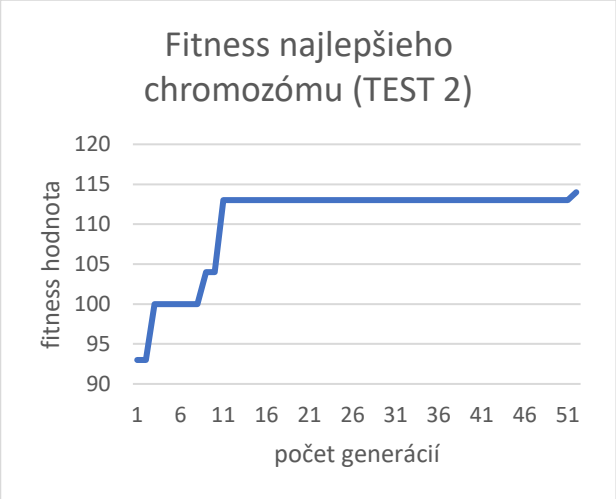
- počet generácii: 48
- čas potrebný na nájdenie riešenia: 4.382s

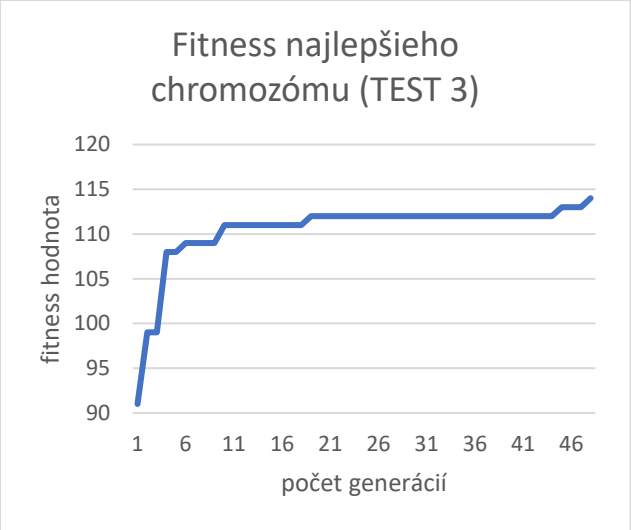
Test č.4:

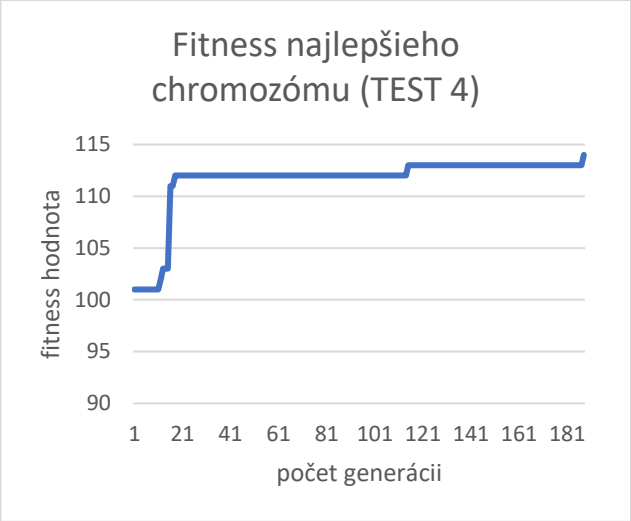
- počet generácii: 188
- čas potrebný na nájdenie riešenia: 18.9s

Grafy závislosti fitness hodnoty od počtu generácii:







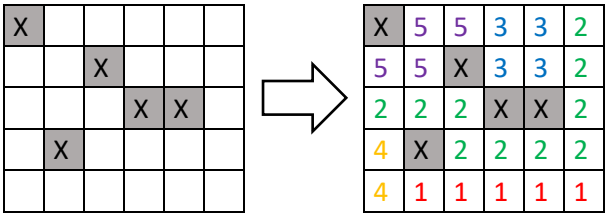


Zhodnotenie

Prvá záhrada predstavuje záhradu, ktorá bola zobrazená aj v zadaní úlohy. Na základe testov, ktoré som si stanovil môžeme vidieť, že najlepšie dopadol **TEST č.3**. Z grafov vyššie môžeme vidieť ako sa vyvíjala fitness hodnota najlepšieho chromozómu z každej generácie. Táto hodnota nikdy neklesá, čo je zabezpečené tým, že najlepší jedinec z každej generácie sa presúva aj do nasledujúcej generácie.

Záhrada 2

Jedno z nájdených riešení:



Test č. 1:

- počet generácii: 5
- čas potrebný na nájdenie riešenia: 0,118s

Test č.2:

- počet generácii: 50
- čas potrebný na nájdenie riešenia: 1,598s

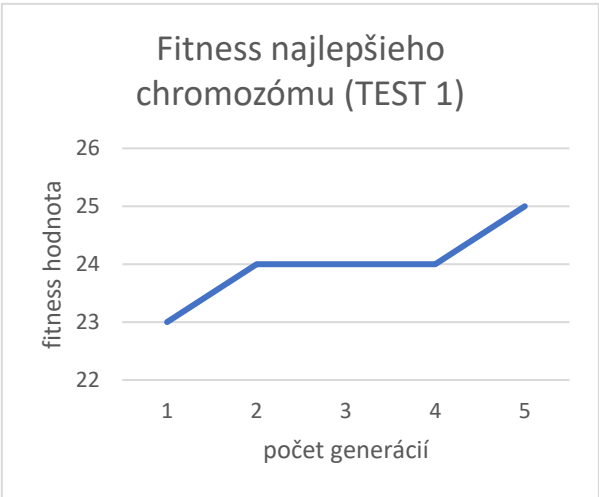
Test č.3:

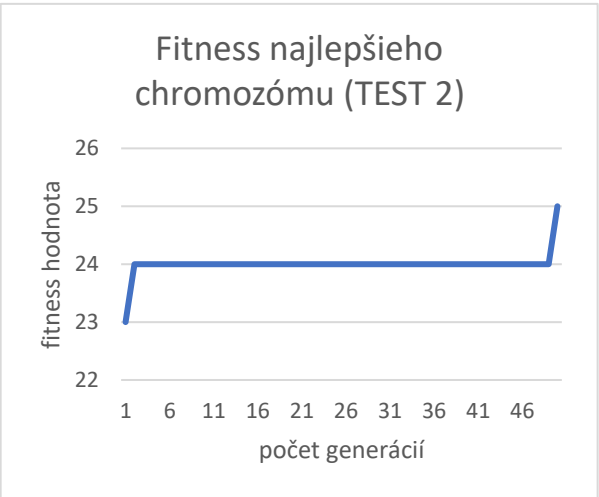
- počet generácii: 7
- čas potrebný na nájdenie riešenia: 0,187s

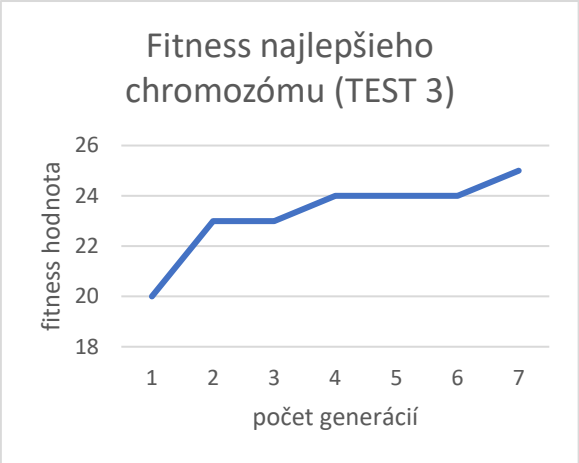
Test č.4:

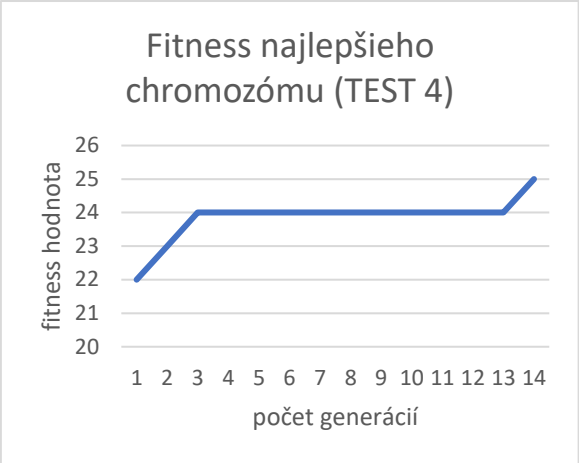
- počet generácii: 14
- čas potrebný na nájdenie riešenia: 0,419s

Grafy závislosti fitness hodnoty od počtu generácii:







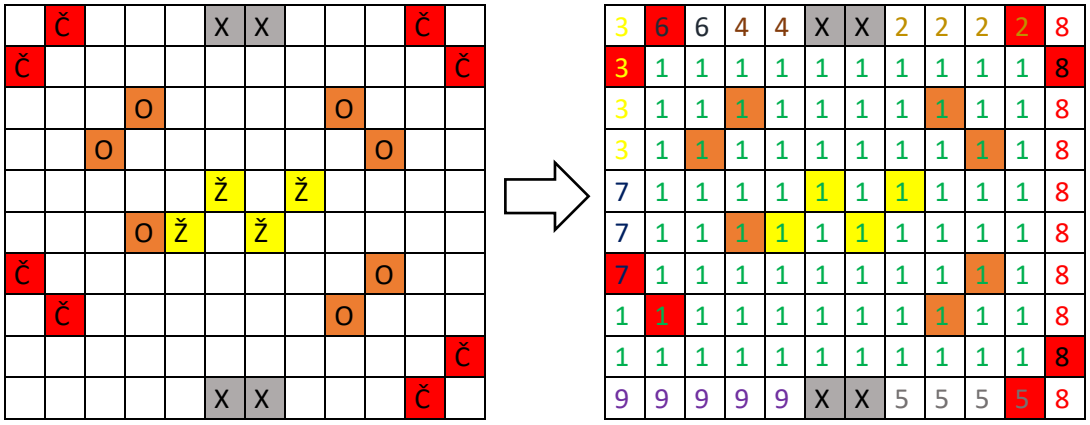


Zhodnotenie

Druhá záhrada predstavuje jednoduchšiu záhradu s niekoľkými kameňmi. Na základe testov, ktoré som si stanovil môžeme vidieť, že najlepšie dopadol **TEST č.1**, kedy sa podarilo riešenie nájsť už po 5 generáciách. Z grafov vyššie môžeme vidieť ako sa vyvíjala fitness hodnota najlepšieho chromozómu z každej generácie. Táto hodnota nikdy neklesá, čo je zabezpečené tým, že najlepší jedinec z každej generácie sa presúva aj do nasledujúcej generácie.

Záhrada 3

Jedno z nájdených riešení:



Test č. 1:

- počet generácii: 17
- čas potrebný na nájdenie riešenia: 1,386s

Test č.2:

- počet generácii: 59
- čas potrebný na nájdenie riešenia: 7,081s

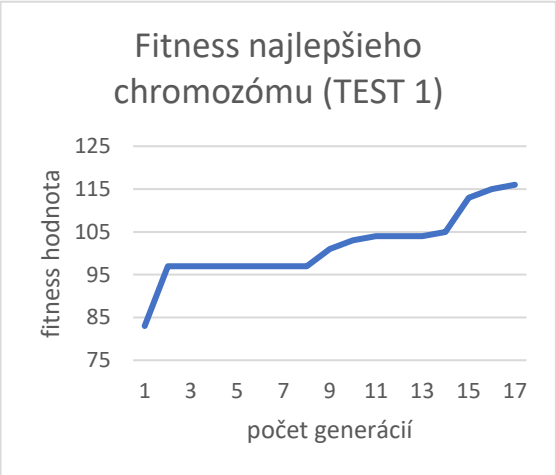
Test č.3:

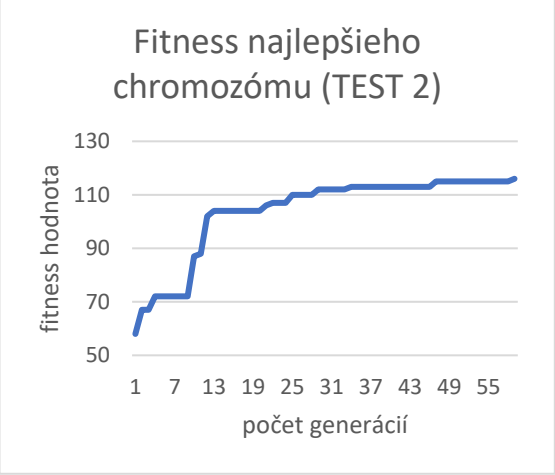
- počet generácii: 16
- čas potrebný na nájdenie riešenia: 1,234s

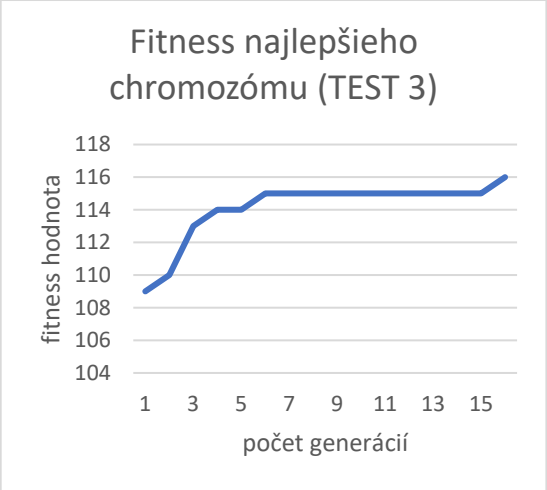
Test č.4:

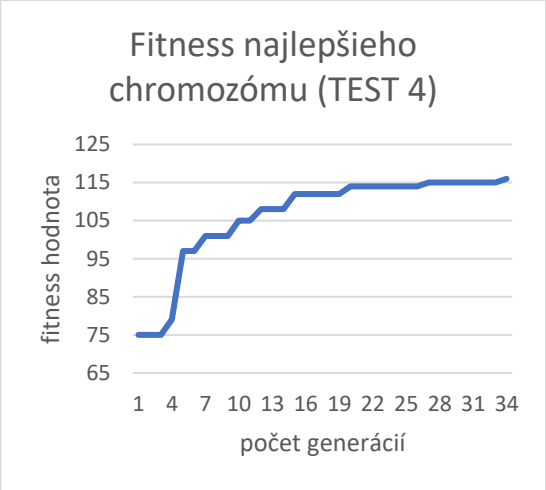
- počet generácii: 34
- čas potrebný na nájdenie riešenia: 3,531s

Grafy závislosti fitness hodnoty od počtu generácii:







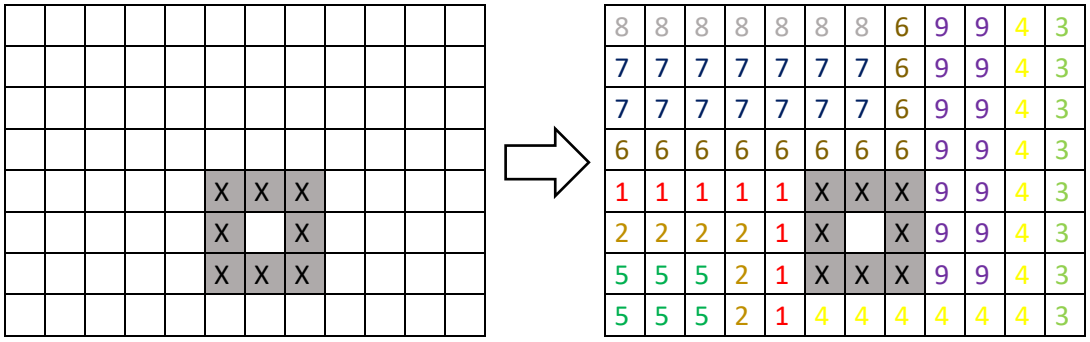


Zhodnotenie

Tretia záhrada predstavuje záhradu s listami, ktorá bola zobrazená aj v zadaní úlohy. Na základe testov, ktoré som si stanovil môžeme vidieť, že najlepšie dopadol **TEST č.3**. Z grafov vyššie môžeme vidieť ako sa vyvíjala fitness hodnota najlepšieho chromozómu z každej generácie. Táto hodnota nikdy neklesá, čo je zabezpečené tým, že najlepší jedinec z každej generácie sa presúva aj do nasledujúcej generácie.

Záhrada 4

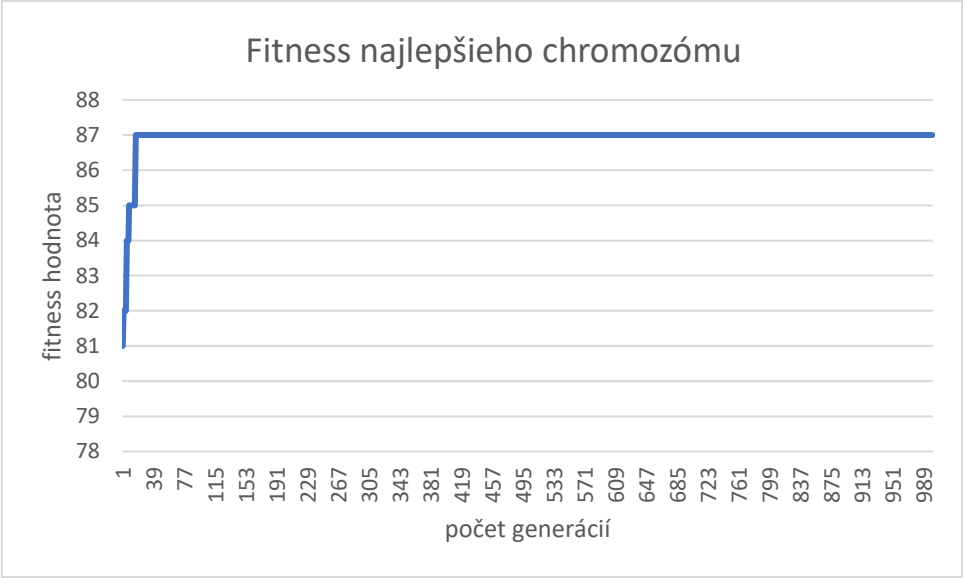
Jedno z nájdených riešení:



Ako môžeme vidieť, táto záhrada nemá riešenie, keďže jedno políčko je zo všetkých strán obklopené kameňmi.

Ani jednému z mojich testov sa teda nepodarí nájsť riešenie, a teda prejdú všetkých 1000 generácií a vrátia najlepšie nájdené riešenie, teda také, v ktorom sa podarilo pohrabať, čo najviac políčok. Jedno z takýchto riešení môžeme vidieť aj vyššie.

Grafy závislosti fitness hodnoty od počtu generácii vyzerá pre všetky testy približne rovnako. Vždy sa rýchlo dostanú na hodnotu fitness 87, ktorá je maximálna možná, a túto si udržiavajú až do konca:



Zhodnotenie

Štvrtá záhrada predstavuje záhrad, ktorá nemá riešenie, a teda žiadnemu z testov sa riešenie nepodarí nájsť. Všetky testy prejdú nastavený počet generácií, v mojom prípade 1000 a vrátia riešenie s maximálnym počtom pohrabaných políčok. Pri takejto záhrade nemá význam porovnávať jednotlivé testovacie scenáre, keďže sa žiadnemu z nich nepodarilo nájsť rešenie.

Zhodnotenie

Zenová záhradka predstavuje zaujímavú výzvu pre vytvorenie efektívneho algoritmu, ktorý by pomohol mníchovi pohrabať piesok tak, aby vytvoril pásy, a zároveň obišiel prekážky.

Evolučný algoritmus pomáha optimalizovať pohyb mnícha a dosiahnuť čo najlepší výsledok vytvorením genetických riešení. Použitie fitness funkcie na hodnotenie kvality riešení je dôležité pre určenie úspešnosti. Maximálny počet génov je obmedzený a to zabezpečuje, že mních neobsiahne viac než polovicu obvodu záhrady a pritom musí prekonať prekážky.

Napriek tomu, že evolučné algoritmy sú silným nástrojom pre riešenie tejto úlohy, je dôležité zdôrazniť, že ich úspech závisí na správnom nastavení parametrov.

Pri rôznych parametroch sa čas a aj počet potrebných generácií môže veľmi líšiť. Taktiež je algoritmus ovplyvnený začiatočným nastavením génov, ktoré sú náhodné. V prípade, že algoritmus začal zle, môže mu trvať oveľa dlhšie, kým príde k výsledku.

Možnosti vylepšenia:

- lepšie ako náhodné generovanie počiatočných génov
- lepšie nastavenie parametrov
- efektívnejší výber rodičov na kríženie
- a podobne