

Umelá inteligencia  
Zen záhrada - Simulované žihanie  
Jakub Šimko  
AIS ID: 103146  
2021/2022

Cvičiaci: Ing. Ivan Kapustík  
Čas cvičení: Streda 13:00

## Obsah

1. Zadanie úlohy.....	4
2. Úvod.....	6
3. Gény.....	7
3.1. Štartovacie pozície.....	7
3.2. Rozhodovanie pri kolízii .....	8
3.3. Maximálny počet génov .....	9
4. Mních / Záhradník.....	9
4.1. Získanie fitness – hrabanie záhrady.....	10
4.1.1. Cyklus hrabania.....	11
5. Hľadanie susedov .....	12
5.1. Prvý variant.....	12
5.1.1. Prehodenie pozícií dvoch génov.....	13
5.1.2. Inverzia rozhodovania dvoch náhodných génov .....	13
5.2. Druhý variant.....	14
6. Simulované žihanie.....	15
6.1. Úprava kvôli testovaniu.....	16
7. Testovanie a výsledky pre rôzne nastavenia.....	16
7.1. Testovanie – nastavenie 1.....	17
7.1.1. Testovanie nastavenia s náhodnými chromozómami.....	17
7.1.2. Testovanie nastavenia s určenými chromozómami.....	17
7.2. Testovanie – nastavenie 2.....	18
7.2.1. Testovanie nastavenia s náhodnými chromozómami.....	18
7.2.2. Testovanie nastavenia s určenými chromozómami.....	19
7.3. Porovnanie nastavení .....	19

Meno: Jakub Šimko  
AIS ID: 103146

7.3.1. Pribeh fitness .....	20
8. Záver .....	21
8.1. Dolad'ovanie a zlepšovanie.....	21

## Obrázky

Obrázok 1: Reprezentácia štartovacích pozícií .....	7
Obrázok 2: Pohyb pri štarte na pozíciách 1, 9 a 6 (nezadefinované správanie pri kolízii) .....	7
Obrázok 3: Gén je záporný - mních odbočil do ľava .....	8
Obrázok 4: Gén nie je záporný - mních odbočil do prava .....	8
Obrázok 5: Funkcia na výpočet maximálneho počtu génov .....	9
Obrázok 6: Trieda záhradníka .....	9
Obrázok 7: Trieda políčka - uchováva informácie o pohybe a pozícií .....	9
Obrázok 8: Implementácia hrabania na základe chromozómu - skrytý cyklus hrabania .....	10
Obrázok 9: Určenie pozície prvého políčka a smeru pohybu na základe štartovacej pozície .....	10
Obrázok 10: Cyklus hrabania vo funkcii <code>rake_garden()</code> .....	11
Obrázok 11: Návratové hodnoty funkcie <code>rake_garden()</code> .....	11
Obrázok 12: Nastavenie hľadania susedov .....	12
Obrázok 13: Implementácia prvého variatnu - a), b) .....	12
Obrázok 14: Ukážka realizácie prvého variantu - a) .....	13
Obrázok 15: Prvá ukážka realizácie prvého variantu – b) .....	13
Obrázok 16: Druhá ukážka realizácie prvého variantu – b) .....	13
Obrázok 17: Ukážka realizácie druhého variantu .....	14
Obrázok 18: Aktuálna implementácia druhé variantu .....	14
Obrázok 19: Príklad nastavenia simulovaného žihania .....	15
Obrázok 20: Implementácia simulovaného žihania .....	15
Obrázok 21: Nastavenie 1 pre simulované žihanie .....	16
Obrázok 22: Výsledok testovania s náhodnými chromozónmi – nastavenie 1 .....	17
Obrázok 23: Určené začiatkové/prvé chromozómy .....	17
Obrázok 24: Výsledok testovania so zadanými chromozónmi pre nastavenie 1 – pokus č. 1 .....	17
Obrázok 25: Výsledok testovania so zadanými chromozónmi pre nastavenie 1 - pokus č. 2 .....	18
Obrázok 26: nastavenie 2 .....	18
Obrázok 27: Výsledok testovania s náhodnými zač. chromozónmi – nastavenie 2 .....	18
Obrázok 28: Výsledok testovania so zadanými chromozónmi pre nastavenie 2 – pokus č. 1 .....	19
Obrázok 29: Výsledok testovania so zadanými chromozónmi pre nastavenie 2 - pokus č. 2 .....	19
Obrázok 30: Výsledok testovania so zadanými chromozónmi pre nastavenie 2 - pokus č. 3 .....	19

## 1. Zadanie úlohy

### Zenová záhrada

#### Zadanie č. 3a

##### Úloha

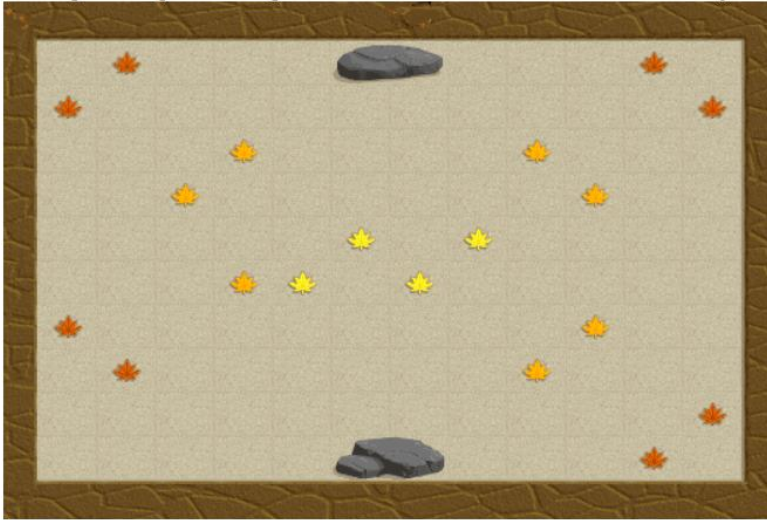
Zenová záhrada je plocha vysypaná hrubším pieskom (drobnými kamienkami). Obsahuje však aj nepohyblivé väčšie objekty, ako napríklad kamene, sochy, konštrukcie, samorasty. Mních má upraviť piesok v záhradke pomocou hrablí tak, že vzniknú pásy ako na nasledujúcom obrázku.



Pásy môžu ísť len vodorovne alebo zvislo, nikdy nie šikmo. Začína vždy na okraji záhradky a ťahá rovný pás až po druhý okraj alebo po prekážku. Na okraji – 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. Úspešná hra je taká, v ktorej mních dokáže za daných pravidiel pohrabať celú záhradu, prípadne maximálny možný počet políčok. Výstupom je pokrytie danej záhrady prechodmi mnicha. Pokrytie zodpovedajúce presne prvému obrázku (priebežný stav) je napríklad takéto:

0	0	1	0	0	0	0	0	10	10	8	9
0	0	1	0	0	K	0	0	10	10	8	9
0	K	1	0	0	0	0	0	10	10	8	9
0	0	1	1	K	0	0	0	10	10	8	9
0	0	K	1	0	0	0	0	10	10	8	9
2	2	2	1	0	0	0	0	10	10	8	9
3	3	2	1	0	0	0	0	K	K	8	8
4	3	2	1	0	0	0	0	5	5	5	5
4	3	2	1	0	0	0	11	5	6	6	6
4	3	2	1	0	0	0	11	5	6	7	7

Úlohu je možné rozšíriť tak, že mních navyše zbiera popadané listie. Listy musí zbierať v poradí: najprv žlté, potom pomarančové a nakoniec červené. Příklad vidno na obrázku nižšie. Listy, ktoré zatiaľ nemôže zbierať, predstavujú pevnú prekážku. Je potrebné primerane upraviť fitness funkciu. Za takto rozšírenú úlohu je možné získať navyše jeden bonusový bod.



### Zadanie

Uvedenú úlohu riešte pomocou evolučného algoritmu. (Je možné použiť aj ďalšie algoritmy, ako sú uvedené v probléme obchodného cestujúceho.) Maximálny počet génov nesmie presiahnuť polovicu obvodu záhrady plus počet kameňov, v našom prípade podľa prvého obrázku  $12+10+6=28$ . Fitness je určená počtom pohrabaných políčok. Výstupom je matica, znázorňujúca cesty mnicha. Je potrebné, aby program zvládol aspoň záhradku podľa prvého obrázku, ale vstupom môže byť v princípe ľubovoľná mapa.

### Náčrt algoritmu

Jedná sa o klasický genetický algoritmus, takže na začiatku, po načítaní rozmerov záhrady a pozícií kameňov, sa vytvorí prvá generácia jedincov s náhodne nastavenými génmi. Potom sa všetky jedince ohodnotia, teda pre každého jedinca sa vytvorí matica s prechodmi mnicha a zistí sa, koľko políčok sa podarilo pokryť. Na základe ohodnotenia sa vyberú jedince na tvorbu novej generácie – križenie a takto vytvorení jedinci môžu s určitou pravdepodobnosťou aj mutovať. Vytvorí sa tak nová generácia a to sa vykonáva dokola, až kým sa nepodari pokryť všetky políčka alebo sa dosiahne stanovený počet nových generácií.

Pre gény je najvhodnejšie, keď reprezentujú priamo miesto na obvode záhrady, kde mních vstúpi (ak môže) a začne hrabať. Zvyšné gény môžu reprezentovať rozhodnutia mnicha, či sa pri najbližšej možnosti voľby dá vpravo alebo vľavo.

Vstupom sú rozmery záhrady a súradnice kameňov, výstupom je mapa pohrabanej záhrady, podobne ako na druhom obrázku. (Kamene môžu byť napríklad -1.)

Použijete aspoň dve rôzne metódy výberu jedincov.

Tu je [ukážka](#), ako sa mení pravdepodobnosť výberu zvoleného jedinca od počtu jedincov v turnaji. Pri dvoch jedincoch je závislosť lineárna (tak ako pri selekcii ohodnotením). Viac ako troch jedincov v turnaji zvyčajne nepoužívame, lebo je príliš malá šanca, že sa vyberie nejaký jedinec zo slabšej polovice generácie.

### Dokumentácia

Dokumentácia musí obsahovať konkrétny použitý algoritmus (nie len náčrt algoritmu, ako v zadaní), podrobný opis vlastností použitých génov, opis ako sa pohybuje a rozhoduje mních, spôsob tvorby novej generácie a možnosti nastavenia parametrov. Dôležitou časťou dokumentácie je zhodnotenie vlastností vytvoreného systému a porovnanie dosahovaných výsledkov pre viacero nastavení parametrov. Vývoj fitness je vhodné zobrazit' grafom (stredná hodnota, maximálna). Dokumentácia by mala tiež obsahovať opis vylepšovania, dolad'ovania riešenia.

## 2. Úvod

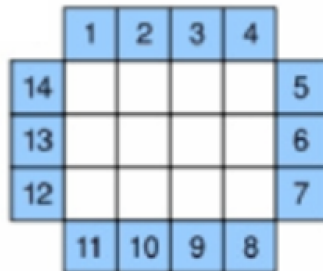
Úlohou je implementovať funkčné riešenie pre problém zen záhrady využitím algoritmu simulovaného žihania. Daný problém som rozdelil na viacero menších častí:

1. Ako zadefinovať gény
  - a. Štartovacia pozícia
  - b. Rozhodovanie
2. Ako odsimulovať hrabanie daného mnícha
3. Ako získať ohodnotenie daného pohrabania
4. Ako hľadať susedov
5. Implementácia simulovaného žihania
6. Správne nastavenia simulovaného žihania a hľadania susedov

### 3. Gény

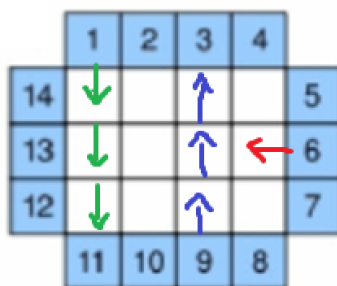
Za najzložitejšiu časť tohto zadania považujem tú v ktorej som musel vymyslieť ako reprezentovať jednotlivé gény. Po dlhšom uvažovaní som dospel k tomu, že gén musí v sebe niesť informáciu o štartovacej pozícii a informáciu o tom ako sa mních rozhodne ak dôjde ku kolízii.

#### 3.1. Štartovacie pozície



Obrázok 1: Reprezentácia štartovacích pozícií

Štartovacie pozície som sa rozhodol reprezentovať tak ako sú znázornené na obrázku č. 1. To znamená, že štartovacie políčko číslo 1 označuje začiatok hrabania na políčko s indexmi 0,0 a mních v prípade tejto začiatočnej pozície hrabe smerom dole. Štartovacia pozícia 14 reprezentuje rovnaké začiatočné políčko ale hrabanie mnícha bude prebiehať smerom do prava. To vlastne znamená, že v prípade štartovacej pozície č. 1 vchádza mních do záhrady zhora v prípade č. 14 vchádza do záhrady z ľava.



Obrázok 2: Pohyb pri štarte na pozíciách 1, 9 a 6 (nezadefinované správanie pri kolízii)

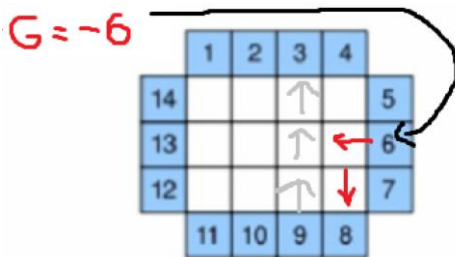
Zadefinovaním štartovacích pozícií je vyriešená prvá časť implementácie génov.



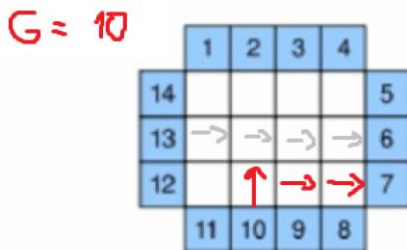
### 3.2. Rozhodovanie pri kolízii

Ako ďalšie je nutné vymyslieť ako bude v danom géne uložená informácií o tom ako sa pri kolízií mních rozhodne. Ak dôjde ku kolízii a mních ma na výber dva nové smery hrabania, tak si vyberie podľa génu. Ak je na výber len jedna možnosť tak sa pre ňu rozhodne – nezáleží na géne. V prípade kedy není na výber žiadny nový smer tak sa mních zasekol.

Vzhľadom na to, že chromozóm je pole celých čísel (int), tak som sa rozhodol v jednotlivých génoch reprezentovať informáciu o rozhodnutí nového smeru pri kolízii znamienkami + a -. Keď dôjde ku kolízii a daný gén je záporný tak z pohľadu mnícha, začne hrabať smerom vľavo v opačnom prípade začína hrabať smerom doprava. Logika tohto správania je implementovaná vo funkcii `decide_direction()`.



Obrázok 3: Gén je záporný - mních odbočil do ľava



Obrázok 4: Gén nie je záporný - mních odbočil do prava

Príklad reprezentácie náhodného chromozómu pre záhradu zo zadania:

[37, -15, 31, -12, -36, 16, -35, -8, 26, -11, -38, 7, 34, -2, -18, -17, 28, -21, -25, -40, -10, 14, 19, 33, 4, 27, -42, -3]

### 3.3. Maximálny počet génov

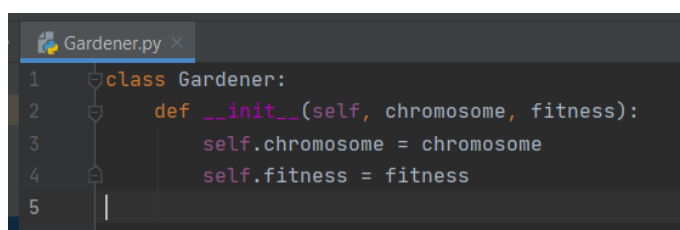
Maximálny počet génov je určený v zadaní ako polovica obvodu záhrady + počet kameňov.

```
# maximalny pocet genov
def get_max_genome(rows, columns, rocks_count):
    return int(get_perimeter(rows, columns) / 2 + rocks_count)
```

Obrázok 5: Funkcia na výpočet maximálneho počtu génov

## 4. Mních / Záhradník

Záhradník je reprezentovaný chromozómom zo sekcie 3 a ohodnotením – fitness.



```
Gardener.py
1 class Gardener:
2     def __init__(self, chromosome, fitness):
3         self.chromosome = chromosome
4         self.fitness = fitness
5
```

Obrázok 6: Trieda záhradníka

Mních v sebe neuchováva informácií o tom, ktorým smerom aktuálne hrabe alebo na akom políčku (súradniciach) sa aktuálne nachádza. Tieto informácie sú uložené v triede Tile, ktorá je využívaná len pri hraní vo funkcii **rake\_garden()**.

```
class Tile:
    def __init__(self, row, column, dir_row, dir_col, previous):
        self.row = row
        self.column = column

        # 1 -> pohyb/hrabanie smerom DOLE
        # -1 -> pohyb/hrabanie smerom HORE
        # 0 -> nehybe sa v tomto smere
        self.dir_row = dir_row

        # 1 -> pohyb/hrabanie smerom DOPRAVA
        # -1 -> pohyb/hrabanie smerom DOLAVA
        # 0 -> nehybe sa v tomto smere
        self.dir_col = dir_col

        # prechadzajúce políčko
        self.previous = previous
```

Obrázok 7: Trieda políčka - uchováva informácie o pohybe a pozícii

#### 4.1. Získanie fitness – hrabanie záhrady

Na to aby sme dokázali určiť fitness mnícha tak je nutné aby nad záhradou vykonal hrabanie – musíme odsimulovať jeho riešenie. Na to slúži funkcia **rake\_garden()**. Na začiatku funkcie sa vytvorí kópia záhrady aby sa hrabanie neuložilo do pôvodnej záhrady. Následné sa pre každý gén vykoná hrabanie.

```
def rake_garden(chromosome, garden, get_solution):
    move_flag = 0
    raked_garden = get_copy_of_map(garden)

    for i in range(0, len(chromosome)):
        # print(f"Pokus o hrabanie č. {i+1}")
        curr_tile = get_direction(chromosome[i], garden.rows, garden.columns)

        # test ci moze vstupit do zahrady
        if raked_garden[curr_tile.row][curr_tile.column] == NOT_RAKED_SAND:
            # print(f"Číslo hrabania: {move_flag+1}")

            move_flag += 1
            # pokiaľ zahradník nevyšiel zo zahrady tak hrabe
            while in_garden_bounds(curr_tile.row, curr_tile.column, garden.rows, garden.columns):...

    if get_solution:
        return raked_garden

    return get_fitness(raked_garden, garden.rows, garden.columns)
```

Obrázok 8: Implementácia hrabania na základe chromozómu - skrytý cyklus hrabania

Na základe začiatkovej pozície získa zač. políčko a smer hrabania. Následne sa skontroluje vo funkcii **rake\_garden()** či je možné vstúpiť do záhrady na toto políčko.

```
def get_direction(gene_number, rows, columns):
    if gene_number < 0:
        gene_number = gene_number * -1

    # začiatok v hornej časti zahrady -> pohyb smerom DOLE
    if gene_number <= columns:
        tile = Tile(0, gene_number-1, 1, 0, None)
    # začiatok v pravej časti zahrady -> pohyb smerom DOLAVA
    elif columns < gene_number <= columns + rows:
        tile = Tile(gene_number-columns-1, columns-1, 0, -1, None)
    # začiatok v dolnej časti zahrady -> pohyb smerom HORE
    elif columns + rows < gene_number <= columns*2 + rows:
        tile = Tile(rows-1, columns*2 + rows - gene_number, -1, 0, None)
    # začiatok v lavej časti zahrady -> pohyb smerom DOPRAVA
    else:
        tile = Tile(get_perimeter(rows, columns) - gene_number, 0, 0, 1, None)

    return tile
```

Obrázok 9: Určenie pozície prvého políčka a smeru pohybu na základe štartovacej pozície

#### 4.1.1. Cyklus hrabania

Komentáre v kóde jednoducho vysvetľujú logiku hrabania. Najdôležitejšou funkciou tohto cyklu je funkcia **decide\_direction()**, ktorej správanie je vysvetlené v sekcii 3.2.

```
while in_garden_bounds(curr_tile.row, curr_tile.column, garden.rows, garden.columns):
    # kolízia s kamenom - zaahradník musí vykonať rozhodnutie zmeny smeru
    if raked_garden[curr_tile.row][curr_tile.column] != NOT_RAKED_SAND:
        # print("NARAZIL SOM")
        prev_tile = curr_tile.previous
        curr_tile = decide_direction(Garden(raked_garden, garden.rows, garden.columns, garden.rock_count), curr_tile.previous, chromosome[i])

    # zahradník uviazol / zasekol sa
    if curr_tile is None:
        # print("Zasekol som sa, mažem svoje kroky")
        # vrátiť sa späť kroky tohto tahu
        move_flag -= 1
        while prev_tile is not None:
            raked_garden[prev_tile.row][prev_tile.column] = 0
            prev_tile = prev_tile.previous
        break

    # po zmene smeru sme vyšli von zo záhrady -> koniec hrabania v tomto tahu
    elif not in_garden_bounds(curr_tile.row, curr_tile.column, garden.rows, garden.columns):
        # print("Zmenou som vyšiel von zo záhrady")
        break

    raked_garden[curr_tile.row][curr_tile.column] = move_flag
    new_tile = Tile(curr_tile.row + curr_tile.dir_row, curr_tile.column + curr_tile.dir_col,
                    curr_tile.dir_row, curr_tile.dir_col, curr_tile)
    curr_tile = new_tile
```

Obrázok 10: Cyklus hrabania vo funkcii `rake_garden()`

Na konci funkcia vráti fitness daného mnícha alebo pohrabanú záhradu. Záleží od tretieho argumentu funkcie (`get_solution`).

```
# ak je funkcia zavolaná s umyslom získať pohrabanú záhradu a nie fitness mnícha
if get_solution:
    return raked_garden

return get_fitness(raked_garden, garden.rows, garden.columns)
```

Obrázok 11: Návrátové hodnoty funkcie `rake_garden()`

## 5. Hľadanie susedov

V rámci hľadania susedov som postupne vylepšoval funkciu `get_neighbour()`. V programe sú implementované tri spôsoby hľadania suseda.

```
# nastavenia pre vytvorenie suseda
VARIANT2_CHANCE: Final[int] = 2 # 20% šanca vykonania variantu 2
SWAP_POSITIONS_GENES_CHANCE: Final[int] = 5 # 50% šanca na vykonanie variantu 1a) / 1b)
```

Obrázok 12: Nastavenie hľadania susedov

### 5.1. Prvý variant

Šanca na vykonanie prvého variantu oproti variantu dva je 80%. Šanca na vykonanie typu 4.1.1 oproti 4.1.2 je 50%. Dôvod prečo som sa rozhodol len pre 20% šancu na vykonanie druhého variantu je ten, že výber/generovanie suseda je náhodné a susedia nie sú unikátny. Ak by šanca na hľadanie pomocou druhého variantu bola vysoká, tak by sa program teoreticky mohol dostať len pomocou pár iterácií do úplne odlišného stavu a to by znamenalo, že neprebíha lokálne vylepšovanie ale skôr náhodne hľadanie.

```
# 1. variant na najdenie suseda:
#     a) vymena pozicii genov
#     b) inverzia v rozhodovaní genu
# 2. variant na najdenie suseda:
#     a) Vyberie sa 1 nahodny gen ktory bude v chromozome nahradeny inym

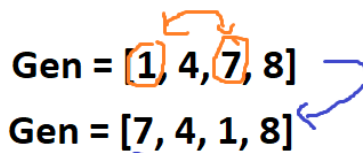
# 90% šanca na vykonanie variantu 1
variant1_chance = random.randint(1, 10)
if variant1_chance > VARIANT2_CHANCE:
    # vybere sa nahodne cislo podla ktoreho sa urci ci pri tomto hladani susedov sa bude swapovat start gen
    # alebo sa obrati/zmeni rozhodovanie pri naraze (+- znak)
    swap_decision_genes_chance = random.randint(1, 10)

    # inverznu sa rozhodovania genov - 50% šanca
    if swap_decision_genes_chance > SWAP_POSITIONS_GENES_CHANCE:
        changed_chromosome[rand1] = changed_chromosome[rand1] * -1
        changed_chromosome[rand2] = changed_chromosome[rand2] * -1
    # prehodia sa pozicie genov
    else:
        tmp = changed_chromosome[rand1]
        changed_chromosome[rand1] = changed_chromosome[rand2]
        changed_chromosome[rand2] = tmp
```

Obrázok 13: Implementácia prvého variantu - a), b)

#### 5.1.1. Prehodenie pozícií dvoch génov

Pri prvej implementácii algoritmu program hľadal nových susedov len prehodením 2 náhodných génov.



Obrázok 14: Ukážka realizácie prvého variantu - a)

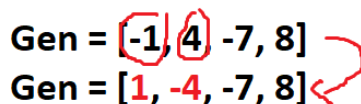
Počet susedov sa pri tejto metóde rovná:

$$\sum_{i=1}^n n - i$$

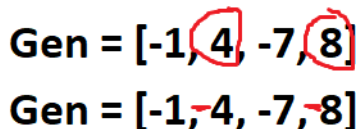
Pre počet génov ( $n$ ) = 4 je týmto hľadaním program schopný nájsť sedem ( $3 + 2 + 1$ ) unikátnych susedov. S výsledkami program využitím tejto jedinej metódy hľadania som nebol spokojný a preto som sa rozhodol implementovať variant b) hľadania susedov.

#### 5.1.2. Inverzia rozhodovania dvoch náhodných génov

Implementácia tohto spôsobu je veľmi podobná 4.1.1. Vyberú sa 2 náhodné gény a zmenia sa ich znamienka na opačné.



Obrázok 15: Prvá ukážka realizácie prvého variantu – b)



Obrázok 16: Druhá ukážka realizácie prvého variantu – b)

Počet susedov sa po pridaní tohto spôsobu zvýši o:

$$\sum_{i=1}^n n - i$$

## 5.2. Druhý variant

Druhý variant spočíva vo výbere náhodného génu a jeho nahradení génom, ktorý sa v chromozóme nenachádza. Rozhodujúca je štartovacia pozícia génu v prípade nového génu 29 sa gén nevymení s náhodným ak sa v chromozóme už nachádza gén -29.

Gen = [1, 4, 7, 8]  
Gen = [6, 4, 7, 8]

Obrázok 17: Ukážka realizácie druhého variantu

Pridaním druhej varianty hľadania susedov sa ich počet zvýši o:

$$x * n$$

o = obvod záhrady

n = počet génov v chromozóme

x = o - n (počet génov ktoré nie sú v chromozóme)

```
else:
    range_numbers = []

    # vsetky mozne geny
    for i in range(1, get_perimeter(garden.rows, garden.columns) + 1):
        range_numbers.append(i)

    existing_gene = True
    while existing_gene:
        existing_gene = False

        # ak nahodne vybrany gen uz je v chromozome tak sa cyklus zopakuje
        # aby sa našiel na vymenu taky gen ktorý v chromozome nie je
        rand_gene_num = random.randint(0, get_perimeter(garden.rows, garden.columns)-1)
        for i in range(0, get_max_genome(garden.rows, garden.columns, garden.rocks_count)-1):
            if range_numbers[rand_gene_num] == changed_chromosome[i] \
                or range_numbers[rand_gene_num] == changed_chromosome[i] * -1:
                existing_gene = True
                break

        if existing_gene:
            continue

        # vymeni nahodny gen za novy
        rand3 = random.randint(0, get_max_genome(garden.rows, garden.columns, garden.rocks_count) - 1)
        changed_chromosome[rand3] = range_numbers[rand_gene_num]
        if random.randint(1, 10) > 5:
            changed_chromosome[rand3] = changed_chromosome[rand3] * -1
```

Obrázok 18: Aktuálna implementácia druhého variantu

Druhý variant som najprv implementoval len ako mutáciu, šanca mutácie sa zvyšovala keď sa program zasekol v lokálnom maxime. Po dlhšom uvažovaní som ale program zmenil aby sa toto hľadanie susedov vykonávalo s určitou šancou vždy, pri každom hľadaní susedov. Aktuálne je v programe nastavená šanca na výber suseda touto metódou na 20%.

## 6. Simulované žihanie

Simulované žihanie je pravdepodobnostná optimalizačná metóda prehľadávania stavového priestoru založená na simulácii žihania ocele. Pri prehľadávaní stavového priestoru môže ľahko dôjsť ku uviaznutiu v lokálnom maxime. V simulovanom žihani sa tomu snažíme zabrániť tým, že robíme aj zmeny k horšiemu. Na základe šance si algoritmus vyberie aj suseda, ktorý má horšie ohodnotenie ako aktuálny stav. Tieto zmeny nastávajú hlavne zo začiatku a vďaka nim sa môžeme dostať z lokálneho maxima. Šanca zmeny závisí na teplote. Čím väčšia teplota tým väčšia šanca na vykonanie zmeny. Pri nízkej teplote sa simulované žihanie začne správať ako lačný algoritmus. Simulované žihanie pracuje len s jedným kandidátnym riešením. V programe je simulované žihanie možné kontrolovať troma konštantami – začiatočnou teplotou, znižovaním teploty a dĺžkou jednej fázy.

```
START_TEMPERATURE: Final[int] = 3600  
TEMPERATURE_DECREASE: Final[int] = -3  
PHASE_LENGTH: Final[int] = 120
```

Obrázok 19: Príklad nastavenia simulovaného žihania

Algoritmus simulovaného žihania je implementovaný vo funkcii *simulated\_annealing()*.

```
331 def simulated_annealing(start_garden, first_gardener):  
332     current = best_gardener = first_gardener  
333     counter = 0  
334     for t in range(START_TEMPERATURE, 0, TEMPERATURE_DECREASE):  
335         counter += 1  
336         for i in range(0, PHASE_LENGTH):  
337             new = get_neighbour(current, start_garden)  
338  
339             # našlo sa riešenie  
340             if new.fitness == (start_garden.rows * start_garden.columns - start_garden.rocks_count):  
341                 print(f"\nPôčet iterácií: {counter * PHASE_LENGTH}")  
342                 return new, counter*PHASE_LENGTH  
343             # priebežne si uklada najlepšího zahradníka v prípade že program nenajde riešenie  
344             elif new.fitness > best_gardener.fitness:  
345                 best_gardener = new  
346  
347             # našiel sa lepší sused / bol prijatý horší  
348             fitness_diff = current.fitness - new.fitness  
349             if new.fitness > current.fitness or random.uniform(0, 1) < math.exp(-fitness_diff / t):  
350                 current = new  
351  
352     print(f"\nPôčet iterácií: {counter*PHASE_LENGTH}")  
353     return best_gardener, counter*PHASE_LENGTH
```

Obrázok 20: Implementácia simulovaného žihania

V prípade kedy sa ohodnotenie aktuálneho mnícha rovná ohodnoteniu riešenia tak funkcia vráti riešenie. V prípade kedy sa riešenie nenájde, tak funkcia vráti najlepšieho nájdeného mnícha. Algoritmus prejde do suseda ak je jeho ohodnotenie lepšie ako ohodnotenie aktuálneho stavu alebo v prípade pravdepodobnostného prijatia nižšieho ako 100% - náhodne číslo v rozmedzí 0 až 1 má nižšiu hodnotu ako  $e^{\frac{-\Delta \text{fitness}}{\text{teplota}}}$ .



### 6.1. Úprava kvôli testovaniu

Kvôli testovaniu som do funkcie pre simulované žihanie pridal ďalšie premenné a funkcionality na zaznamenávanie priebehu fitness v jednotlivých fázach. Pridané časti sú v zdrojovom kóde okomentované aby bol rozdiel oproti obrázku č. 20 jednoznačný. Program po každom vykonaní algoritmu vygeneruje xls súbor obsahujúci informácie o priebehu fitness v danej teplote. Ak prebieha viac ako 1 test tak sa tento súbor vždy nahrádza a na konci sú uložené len výsledky z posledného vykonania algoritmu.

## 7. Testovanie a výsledky pre rôzne nastavenia

```
# nastavenia simulovaného žihania
START_TEMPERATURE: Final[int] = 3600
TEMPERATURE_DECREASE: Final[int] = -3
PHASE_LENGTH: Final[int] = 690

# nastavenia pre vytvorenie suseda
VARIANT2_CHANCE: Final[int] = 2 # 20% šanca vykonania variantu 2
SWAP_POSITIONS_GENES_CHANCE: Final[int] = 5 # 50% šanca na vykonanie variantu 1a) / 1b)
```

Obrázok 21: Nastavenie 1 pre simulované žihanie

Pri určovaní dĺžky fázy som sa odrážal hlavne od výpočtu koľko unikátnych susedov môže daný stav v mojom programe mať, v prípade, že sa žiadny so susedov nevybral až dokiaľ sa neprejdú všetky.

$$2 * (\sum_{i=1}^n n - i) + x * n = 2(120) + 448 = 688$$

Ku všetkým hodnotám som sa ale viac menej ajtak dopracoval len formou testovania a hľadaním optimálneho nastavenia. To isté platí aj pre nastavené hodnoty v hľadaní suseda. Pri hľadaní optimálneho nastavenia som sa viac sústredil na čo najvyššiu úspešnosť než na rýchlosť vykonania. Nastavenie z obrázka 21 dosahuje pre záhradu zo zadania 100% úspešnosť ale nie je najrýchlejšie.

### 7.1. Testovanie – nastavenie 1

Pre nastavenie z obrázku 21 bolo vykonaných 200+ testov, všetky skončili s úspešným nájdením riešenia (100% úspešnosť). Priemerná dĺžka prehľadávania pre toto nastavenie sa pohybuje okolo 20 sekúnd a odvíja sa hlavne od začiatočného chromozómu. Samozrejme je možné považovať toto nastavenie za pomalé a radšej zvoliť nastavenie s nižšou úspešnosťou ale rýchlejším vykonaním. To už ale záleží od toho čo považujeme za optimálne nastavenie.

#### 7.1.1. Testovanie nastavenia s náhodnými chromozómami

```
Počet testov: 50
Počet úspešných hľadání: 50
Počet neúspešných hľadání: 0
Priemerne ohodnotenie neúspešneho riešenia: 0
Priemerna dĺžka prehľadávania: 22.9596875s
```

Obrázok 22: Výsledok testovania s náhodnými chromozómami – nastavenie 1

#### 7.1.2. Testovanie nastavenia s určenými chromozómami

```
testing_chromosomes = [
  [-34, 2, -14, -29, 37, 20, -8, -40, 38, -42, 24, -31, 33, 22, -21, -41, -17, 18, -9, 15, -23, 4, 43, 13, 30, -10, -32, -35],
  [-42, -36, 11, 35, 38, 28, 12, 24, -33, 25, 8, 6, 29, 14, -34, 40, -15, -23, 16, 4, -26, 17, -30, 32, -39, -10, -27, 31],
  [44, 38, -14, -40, 43, -29, 9, 4, 6, -7, -21, -25, 2, 18, 41, 15, -35, -30, -22, -1, 19, -39, 13, 27, 28, -3, -17, 37],
  [-15, 37, -14, -30, -1, -40, -9, -38, 17, 16, -23, -2, -7, 31, 35, 34, 39, -19, -3, 36, -32, 26, 28, -43, 8, 21, -24, -4],
  [-28, 37, -3, -7, -16, -12, -14, -19, -38, -25, -43, 30, 40, 39, 27, 34, 13, -41, -35, -17, -15, 11, 10, -1, 20, -21, -31, 23],
  [-1, 31, -18, -35, 16, 41, 3, -6, -23, -28, 24, 34, 42, -17, -2, -32, -19, -26, 11, -14, -21, -12, -40, -36, 13, 9, -25, -44],
  [-4, -13, -2, -32, -15, 35, -22, 21, 10, 39, -3, 18, 23, 20, -8, -17, 25, 7, 36, -14, 12, -42, 34, 26, 9, -6, 11, -37],
  [-31, -42, -44, 21, -27, 37, 32, 3, 15, 22, 14, -17, -24, 30, -5, -16, -19, 26, -38, 13, -18, -1, -34, 28, 12, 9, -43, 29],
  [-23, 40, -10, 26, -25, 41, -15, 44, 9, 42, -17, -3, 14, 43, 2, 38, 12, 39, -19, 34, 5, 4, -13, 11, -30, 37, 32, 18],
  [10, 42, -7, -14, -39, 11, -31, -26, 44, 18, -35, -16, 43, -1, -36, -29, 9, 32, 5, -28, -2, 22, -24, -40, 27, -17, -19, -38]
]
```

Obrázok 23: Určené začiatočné/prvé chromozómy

```
Počet testov: 10
Počet úspešných hľadání: 10
Počet neúspešných hľadání: 0
Priemerne ohodnotenie neúspešneho riešenia: 0
Priemerna dĺžka prehľadávania: 15.525s
```

Obrázok 24: Výsledok testovania so zadanými chromozómami pre nastavenie 1 – pokus č. 1

```
Počet testov: 10  
Počet úspešných hľadání: 10  
Počet neúspešných hľadání: 0  
Priemerne ohodnotenie neúspešného riešenia: 0  
Priemerná dĺžka prehľadávania: 21.9796875s
```

Obrázok 25: Výsledok testovania so zadanými chromozómami pre nastavenie 1 - pokus č. 2

## 7.2. Testovanie – nastavenie 2

```
# nastavenia simulovaného žihania  
START_TEMPERATURE: Final[int] = 1800  
TEMPERATURE_DECREASE: Final[int] = -3  
PHASE_LENGTH: Final[int] = 300
```

Obrázok 26: nastavenie 2

Pri hľadaní nastavenia s ktorým by som porovnal prvotné nastavenie som strávil veľa času vzhľadom na to, že som sa snažil nájsť rovnováhu medzi najrýchlejším vykonaním a najvyššou úspešnosťou. Nakoniec som sa dopracoval ku nastaveniu z obrázka 26. Nižšie hodnoty v nastavení buď dosahovali horšie časy vykonania alebo dosahovali horšiu úspešnosť. Toto nastavenie už naďalej nedosahuje 100% úspešnosť vo všetkých prípadoch testovania ale jeho vykonanie je rýchlejšie.

### 7.2.1. Testovanie nastavenia s náhodnými chromozómami

```
Počet testov: 50  
Počet úspešných hľadání: 48  
Počet neúspešných hľadání: 2  
Priemerne ohodnotenie neúspešného riešenia: 113.0  
Priemerná dĺžka prehľadávania: 19.999375s
```

Obrázok 27: Výsledok testovania s náhodnými zač. chromozómami – nastavenie 2

### 7.2.2. Testovanie nastavenia s určenými chromozómami

Pri tomto testovaní boli využité začiatkové/prvé chromozómy z obrázka 23.

```
Počet testov: 10  
Počet úspešných hľadání: 10  
Počet neúspešných hľadání: 0  
Priemerne ohodnotenie neúspešného riešenia: 0  
Priemerna dĺžka prehľadávania: 9.5625s
```

Obrázok 28: Výsledok testovania so zadanými chromozómami pre nastavenie 2 – pokus č. 1

```
Počet testov: 10  
Počet úspešných hľadání: 10  
Počet neúspešných hľadání: 0  
Priemerne ohodnotenie neúspešného riešenia: 0  
Priemerna dĺžka prehľadávania: 10.965625s
```

Obrázok 29: Výsledok testovania so zadanými chromozómami pre nastavenie 2 - pokus č. 2

```
Počet testov: 10  
Počet úspešných hľadání: 9  
Počet neúspešných hľadání: 1  
Priemerne ohodnotenie neúspešného riešenia: 113.0  
Priemerna dĺžka prehľadávania: 27.73125s
```

Obrázok 30: Výsledok testovania so zadanými chromozómami pre nastavenie 2 - pokus č. 3

### 7.3. Porovnanie nastavení

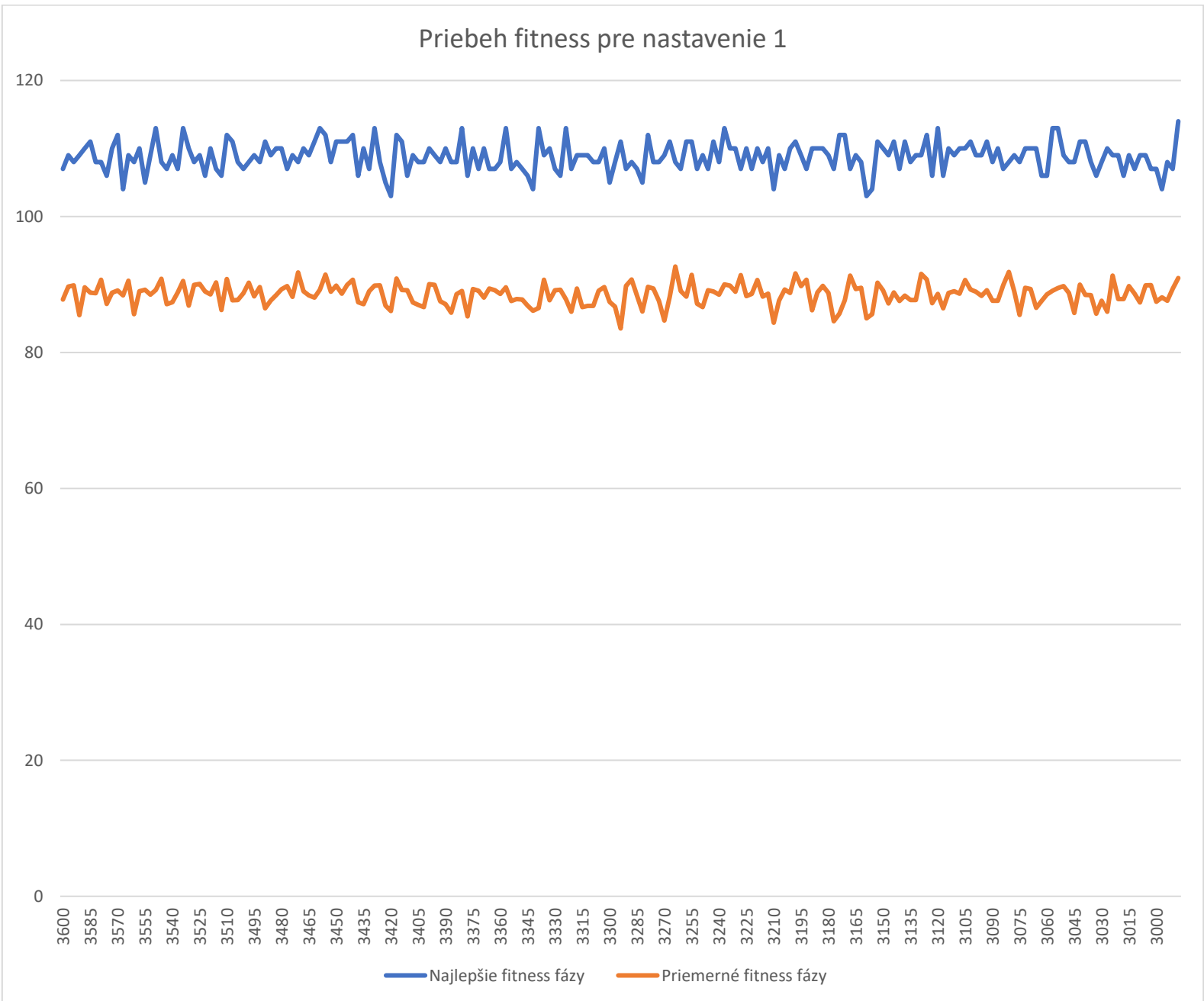
Nastavenie 2 môžeme považovať za rýchlejšie, hlavne ak sa dokáže dopracovať ku riešeniu. Vzhľadom na to, že v prípade tohto nastavenia je začiatková teplota a aj dĺžka fázy nižšia tak sa algoritmus rýchlejšie stane pažravým. Kvôli tomu je aj vyššia šanca, že sa zasekne v lokálnom maxime a to spôsobí, že celkové vykonanie bude dlhšie ako v prípade nastavenia 1.

## 7.3.1. Priebek fitness

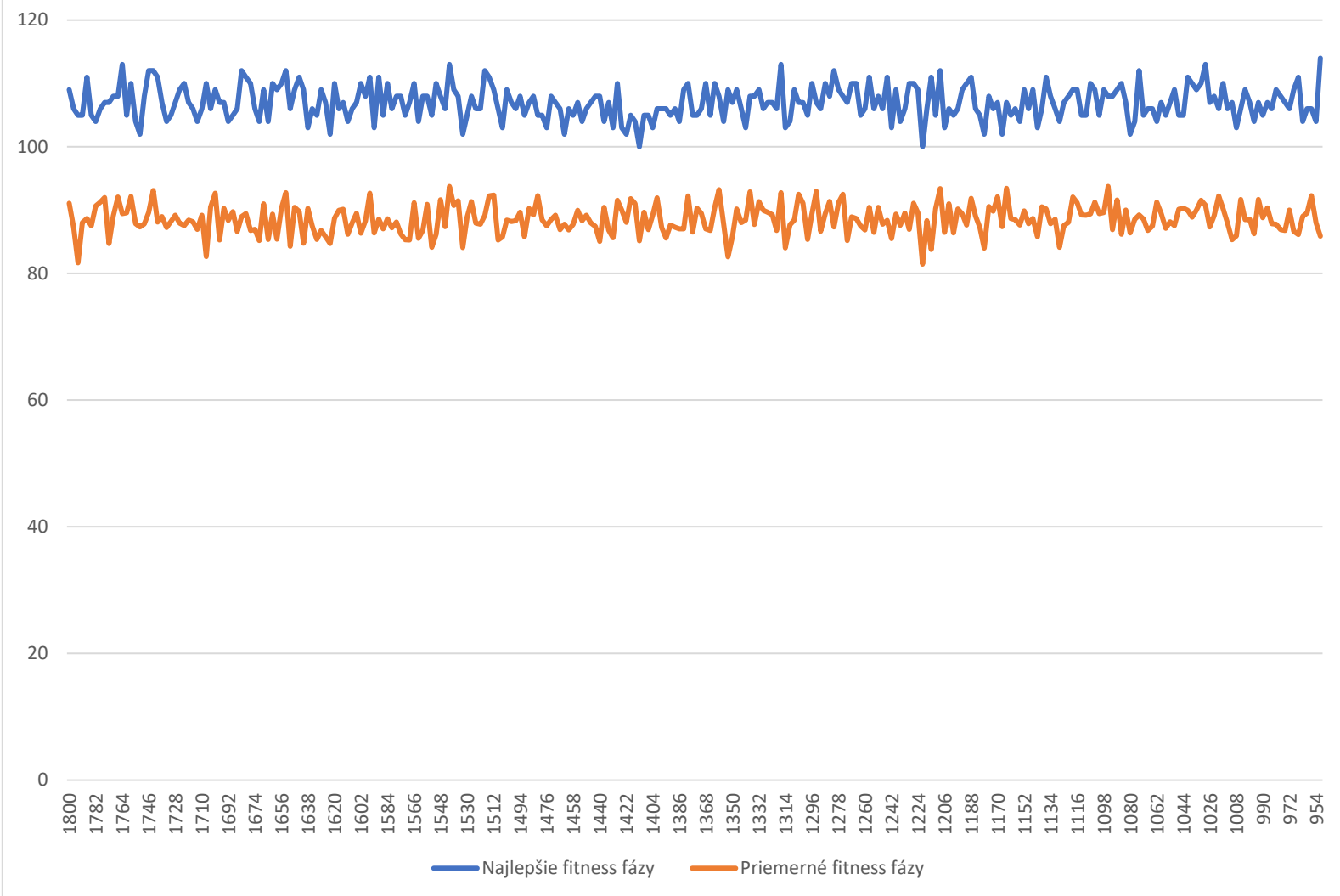
Priebek fitness pre záhradu zo zadania a tento chromozóm:

[-34, 2, -14, -29, 37, 20, -8, -40, 38, -42, 24, -31, 33, 22, -21, -41, -17, 18, -9, 15, -23, 4, 43, 13, 30, -10, -32, -35].

V dolnej časti grafu sa nachádza aktuálna teplota. V rámci tohto testovania sa nastavenie 1 dostalo k riešeniu rýchlejšie. Na grafu si môžeme všimnúť, že čím je teplota nižšia tým menej dochádza ku kolísaniu priemernej fitness.



Priebeh fitness pre nastavenie 2



## 8. Záver

Úspešne som vymyslel a implementoval riešenie pre problém zen záhrady využitím algoritmu simulovaného žihania.

### 8.1. Dolad'ovanie a zlepšovanie

Vo všetkých kapitolách dokumentácie som sa snažil dávať dôraz na to ako som algoritmus a jeho nastavenia pri budovaní dolad'oval a vylepšoval.