SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta informatiky a informačných technológií

Ilkovičova 2, 842 16 Bratislava 4

# ZADANIE 3 – Zenova záhradka (Simulované žíhanie)

Meno a priezvisko: Marko Stahovec

Dátum vypracovania: 17.10.2021

Cvičiaci: Ing. Boris Slíž

# 1. Zadanie

Našou úlohou bolo implementovať algoritmus **simulovaného žíhania** v ľubovoľnom jazyku pre problém **Zenovej záhradky**. Simulované žíhanie je algoritmus založený na **probabilistickej technike aproximácie globálneho optima**.
Zenova záhradka je problém, v ktorom je cieľom pohrabanie celej záhradky mníchom s jednoduchými pravidlami: mních vstupuje do záhradky z okraja a vystupuje taktiež okrajom a pohybuje sa stále rovno, no ak narazí na prekážku alebo už pohrabané políčko, otáča sa doľava alebo doprava podľa možností. Na okraji sa môže mních pohybovať ľubovoľne, t.j. môže začať úkon hrabania z ľubovoľného hraničného políčka, ktoré je validné.
V našom prípade je teda **globálne optimum** stav, v ktorom **mních pohrabal celú záhradku na čo najmenej ťahov.**

# 2. Opis algoritmu

Môj algoritmus, rovnako ako stochastický hill-climbing, **modifikuje minimálnym spôsobom jedno riešenie a prehľadáva jeho okolitý blízky priestor, pokiaľ nenájde lokálne maximum**. Rozdiel je v akceptácii riešenia, keďže moja implementácia simulovaného žíhania **môže akceptovať aj horšie riešenia s cieľom opustiť lokálne maximum.**

Pravdepodobnosť akceptácie horšieho riešenia je pevne spätá s parametrom teploty, ktorá sa postupným behom programu kontinuálne znižuje. Z toho vyplýva, že program akceptuje **horšie riešenia častejšie na začiatku behu programu**, vďaka čomu môže opustiť lokálne maximá a vyhľadať “vrchol” s globálnym maximom a následne sa po ňom vyšplhať.

## 2.1 Dôležité triedy a funkcie

### 2.1.1 Triedy

```
class Monk:
    def __init__(self, fitness, complete_rakes, correct_finish, random_decisions, border_tiles):
        self.fitness = fitness
        self.complete_rakes = complete_rakes
        self.correct_finish = correct_finish
        self.random_decisions = random_decisions
        self.border_tiles = border_tiles
```

Trieda Monk predstavuje mnícha aj s jeho vlastnými atribútmi na jednoduchú a zrozumiteľnú prácu s jeho údajmi.

- **fitness** - hlavný parameter ohodnotenia. Predstavuje počet políčok, ktorý sa podaril pohrabať danému mníchovi.
- **complete_rakes** - premenná, v ktorej sa udržuje počet ťahov, ktoré sa mníchovi podarilo vykonať. Platí, že ak je riešenie kompletné, tak čím menší počet complete_rakes, tým je to riešenie kvalitnejšie.
- **correct_finish** - vo svojej podstate iba boolean hodnota, v ktorej je uložená pravdivostná hodnota faktu, či sa mníchovi podarila pohrabať celá mapa alebo nie.
- **random_decisions** - pole náhodných rozhodnutí, ktoré sa presúva z mnícha na mnícha s cieľom dodržať čo najväčšiu “susednosť” dvoch mníchov. Obsahuje hodnoty v intervale <0;1> dátového typu float. (Ak je aktuálne prvé rozhodnutie väčšie ako 0,5, tak choď doprava a presuň dané rozhodnotie na koniec poľa)
- **border_tiles** - zoznam hraničných vstupných políčok, cez ktoré môže mních validne vstúpiť do záhradky. Formát jedného prvku poľa vyzerá ako: [0, 3, 1], kde prvé dve čísla predstavujú súradnice vstupného políčka a tretie číslo predstavuje smer vstupu (0 -> smer doprava, 1 -> smerom dole, 2 -> smer doľava, 3 -> smer hore).

### 2.1.2 Hlavné funkcie

```
def simulated_annealing(main_garden, dimensions):
```

je **hlavná algoritmická funkcia**, v ktorej parameter main_garden predstavuje počiatočnú mapu bez akýchkoľvek pohybov a pole dimensions, čo je len pole s dvomi hodnotami, ktoré predstavujú dĺžku a šírku záhrady. Z tejto funkcie sa pre každé riešenie volá funkcia:

```
def rake_wrapper(start_garden, new_monk, max_fitness):
```

ktorá je tzv. placeholder, kde sa **inicializujú premenné, cyklí možnými vstupnými políčkami a vyhodnocujú výsledky pre funkciu**:

```
def rake_garden(garden, position, turn, monk):
```

kde sa **vykonávajú** všetky **pohyby** pre aktuálneho mnícha. Táto funkcia úzko spolupracuje s ďalšími podpornými, ktoré vyhodnocujú, či je políčko validné, či je možné urobiť úkrok do strán pri strete s prekážkou apod. Jej návratovými hodnotami sú aktuálne vykonávané **poradové číslo ťahu** a **počet políčok**, ktoré sa mníchovi podarilo pohrabať.
Spomínané poradové číslo ťahu sa nachádza v premennej **turn**, ktorá sa inkrementuje po vyhrabaní jednej cesty a návrate z funkcie rake_garden(). Všetky tieto inkrementácie a vyberanie vstupných políčok sa realizuje vo funkcii o jeden stupeň vyššie, ktorá po vyskúšaní všetkých možností (alebo zaseknutí sa mnícha) vracia výslednú záhradku ako jedno riešenie.
Kvalita tohto riešenia je vyhodnotená už vo funkcii **simulated_annealing()**.

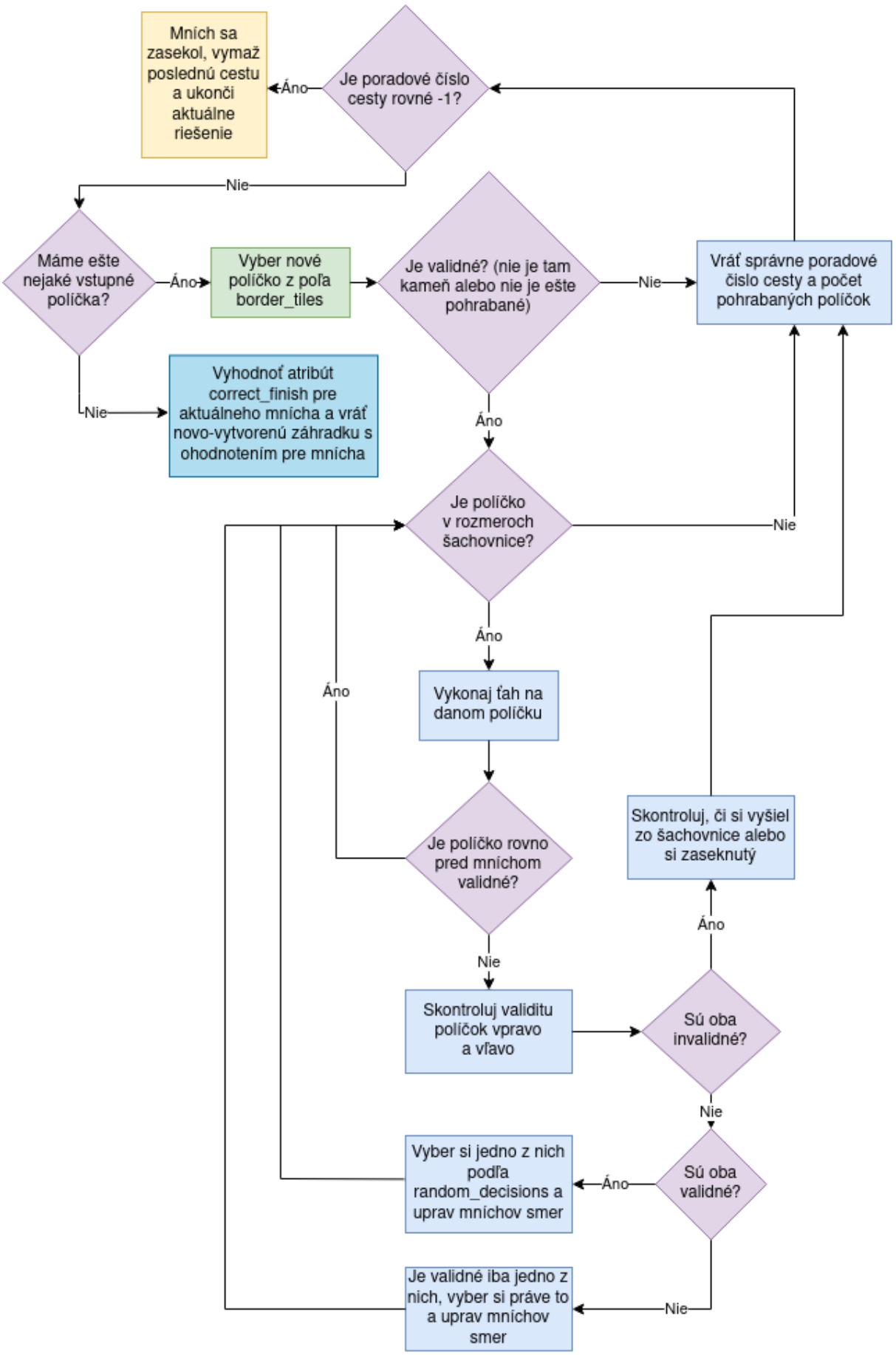
## 2.2 Ako sa generuje jedno riešenie

### 2.2.1 Flow-chart

Proces generovania jedného riešenia je vykreslený vo flow-charte nižšie.

Legenda pre bunky flow-chartu:

- **zelené** - konacie a zároveň počiatočné
- **žlté** - vyhodnocovacie s chybou
- **modré** - vyhodnocovacie bez chyby
- **fialové** - rozhodovacie
- **svetlo-modré** - konacie

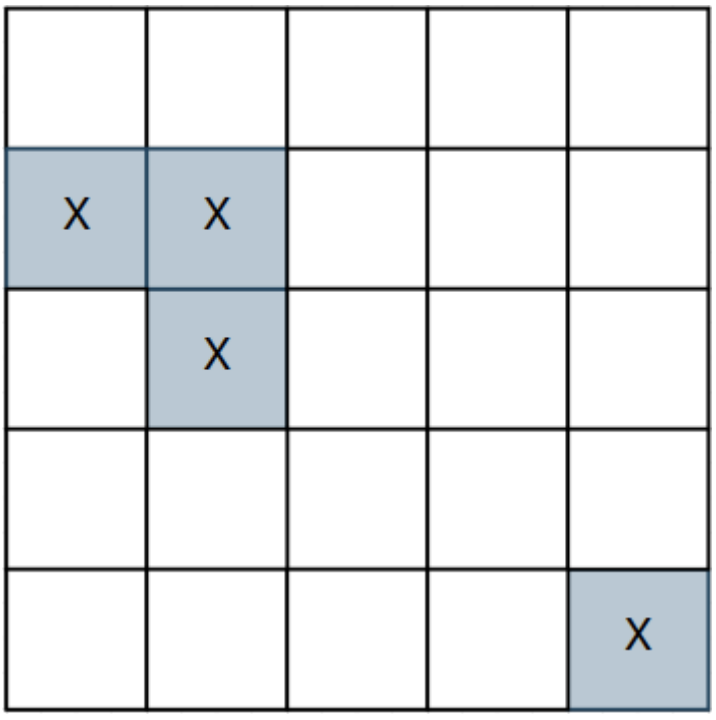

Je poradové číslo cesty rovné -1?
Áno
Mních sa zasekol, vymaž poslednú cestu a ukonči aktuálne riešenie
Nie
Máme ešte nejaké vstupné políčka?
Áno
Vyber nové políčko z poľa border_tiles
Je validné? (nie je tam kameň alebo nie je ešte pohrabané)
Nie
Vráť správne poradové číslo cesty a počet pohrabaných políčok
Nie
Vyhodnoť atribút correct_finish pre aktuálneho mnícha a vráť novo-vytvorenú záhradku s ohodnotením pre mnícha
Áno
Je políčko v rozmeroch šachovnice?
Nie
Áno
Vykonaj ťah na danom políčku
Áno
Je políčko rovno pred mníchom validné?
Nie
Skontroluj validitu políčok vpravo a vľavo
Sú oba invalidné?
Áno
Skontroluj, či si vyšiel zo šachovnice alebo si zaseknutý
Nie
Sú oba validné?
Áno
Vyber si jedno z nich podľa random_decisions a uprav mníchov smer
Nie
Je validné iba jedno z nich, vyber si práve to a uprav mníchov smer

### 2.2.2 Grafický opis jedného riešenia

Jedno riešenie sa začína inicializáciou inštancie triedy Monk a jej parametrov. Ak ide o mnícha, ktorý je inicializačný, tak sa mu do jeho atribútov **random_decisions** a **border_tiles** vygenerujú vhodné zoznamy cez ďalšie pomocné funkcie. Tieto polia budú prekopírované do ďalšieho suseda a následne sa v nich vykoná **len jedna zmena**, aby sme zaručili, že ďalší mních bude naozaj susedom inicializačného.

*Pozn. pole random_decisions bude mať v sebe vždy (3 + počet kameňov) hodnôt. Konštanta 3 pripočítaná k počtu kameňov zaručuje, že ak by v danej záhrade neboli nijaké kamene, tak mních by stále mal nejaké rozhodnutia, s ktorými by mohol pracovať ďalej.*

Príklad ako sa generuje jedno riešenie bude zobrazené na tejto mape (map5.txt):




Po vygenerovaní takejto mapy nasleduje výber nastavení od používateľa, kde si môže zadefinovať rôzne parametre pre efektivitu a hĺbku prehľadávania pre program (tieto parametre budú vysvetlené v časti 2.3). Z tých sa rozpočítajú potrebné hodnoty pre teplotu, hraničnú teplotu apod.

Nasleduje **inicializácia prvého mnícha**.
Príklad atribútov prvého mnícha po inicializácii:

- **monk.fitness** = 0 (zatiaľ nič nepohrabal, tak jeho ohodnotenie je 0)
- **monk.complete_rakes** = 0
- **monk.correct.finish** = 0 (zatiaľ nenašiel kompletné riešenie)
- **monk.random_decisions** = [0.21, 0.64, 0.63, 0.78, 0.96, 0.68, 0.3]
- **monk.border_tiles** = [[4, 0, 0], [0, 4, 1], [4, 1, 3], [0, 3, 1], [2, 4, 2], [2, 0, 0], [1, 4, 2], [4, 2, 3], [0, 4, 2], [0, 0, 0], [0, 1, 1], [0, 0, 1], [4, 3, 3], [3, 0, 0], [4, 0, 3], [0, 2, 1], [3, 4, 2]]

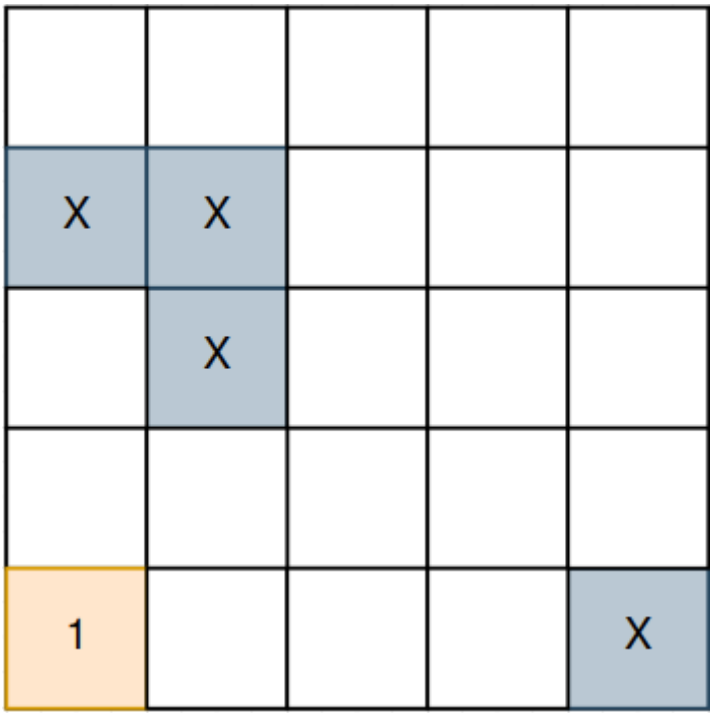
Po inicializácii takýchto parametrov nasleduje vstup do funkcie rake_wrapper(main_garden, new_monk, max_fitness), do ktorej sa posiela **počiatočná záhradka** z obrázku vyššie, **nový mních** s príkladnými atribútmi taktiež vyššie a hodnota **max_fitness**, ktorá predstavuje maximálne možné (najlepšie) ohodnotenie pre danú konkrétnu záhradku. V našom prípade je táto hodnota rovná 21, čo sme dostali násobkom x-ovho a y-ovho rozmeru mapy a odpočítaním počtu kameňov.

Na úvod vytvorí funkcia rake_wrapper() identickú kópiu pôvodnej mapy, nainicializuje premennú turn na 0, ktorá bude obsahovať počet ťahov vykonaných našim mníchom a prejde do hlavného cyklu hrabania.

Syntax cyklu:

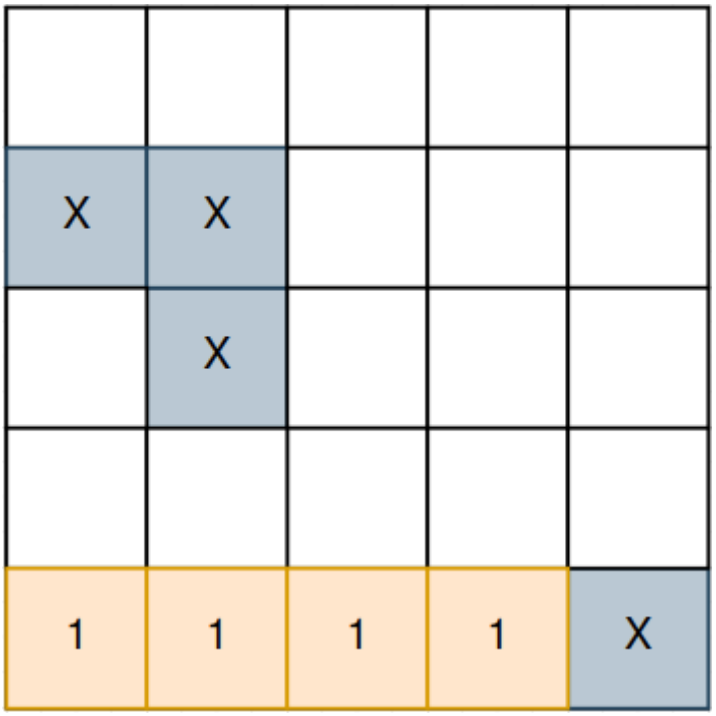
```
for tile in new_monk.border_tiles:
```

Tento cyklus prechádza **všetkými možnými vstupnými políčkami** v poradí, v akom boli inicializované pre nášho mnícha, a teda prvým vstupom bude políčko o súradniciach **[4, 0] a smerom 0**, čo je v mojom riešení smer vpravo. Vstup je zobrazený nižšie na obrázku:



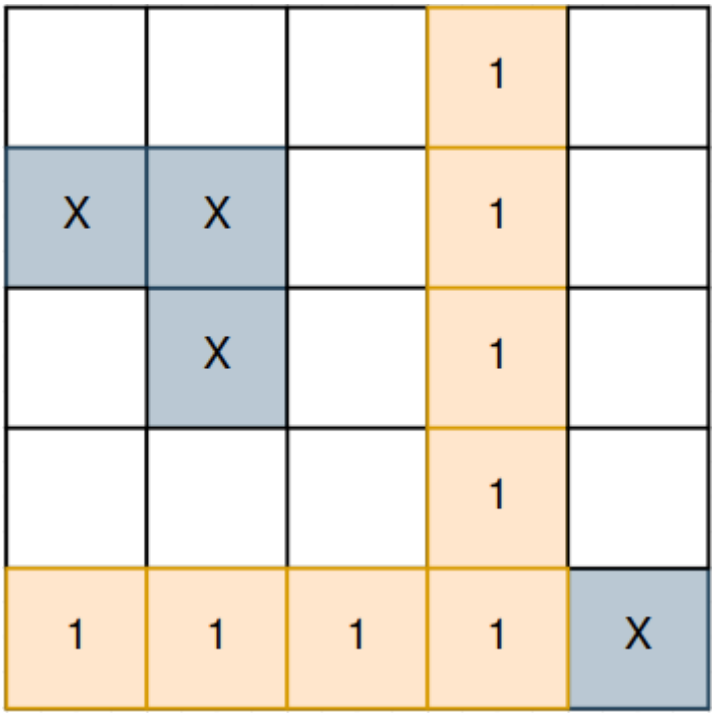

Keďže na danom políčku nebola **nijaká prekážka**, vstup bol mníchovi **umožnený**. Je dobré poznamenať, že ak by bol na krajnom políčku kameň, dané vstupné políčko by nebolo v zozname atribútu border_tiles, keďže tento vstup by za žiadnych okolností nebol validný.

Keďže toto vstupné políčko z border_tiles bolo [4, 0, 0], tretí záznam nám hovorí, že mních vstúpil **nasmerovaný vpravo**, a preto ďalšie políčko, na ktoré vstúpi, bude napravo od vstupného. Malý progres je zobrazený nižšie:




Mních podľa pravidiel postupoval **vždy len rovno**, až pokiaľ nenarazil na prekážku. Môj algoritmus pracuje tak, že pri takejto situácie vyhodnocuje políčka, na ktoré by vstúpil, ak by sa rozhodol zabočiť vpravo aj vľavo. Pri kontrole políčka vpravo zistí, že políčko **vpravo** nie je korektné, **no môže ním vystúpiť**, ale pri ohľade na **ľavé** políčko usúdi, že to je **absolútne korektné**, a preto ním bude pokračovať. Premenná is_right_correct bude teda False a is_left_correct True.

Po takomto rozhodnutí **pokračuje** mních opäť **priamo**, keďže iba to je legálna možnosť pre problém Zenovej záhradky.




V tejto situácii algoritmus zistí, že ďalší ťah vpred by bol ***out of bounds***, kvôli čomu vyhodnotí pohyb vpred ako korektné opustenie záhradky, a preto sa vráti späť do funkcie rake_wrapper() s hodnotami:

- **turn** = 1
- **tiles_raked** = 8

K atribútu **fitness** (monk.fitness) sa **pripočíta hodnota** premennej tiles_raked, vďaka čomu sa priebežne aktualizuje ohodnotenie pre aktuálneho mnícha.
Na záver sa vyhodnotí návratová hodnota z premennej turn:

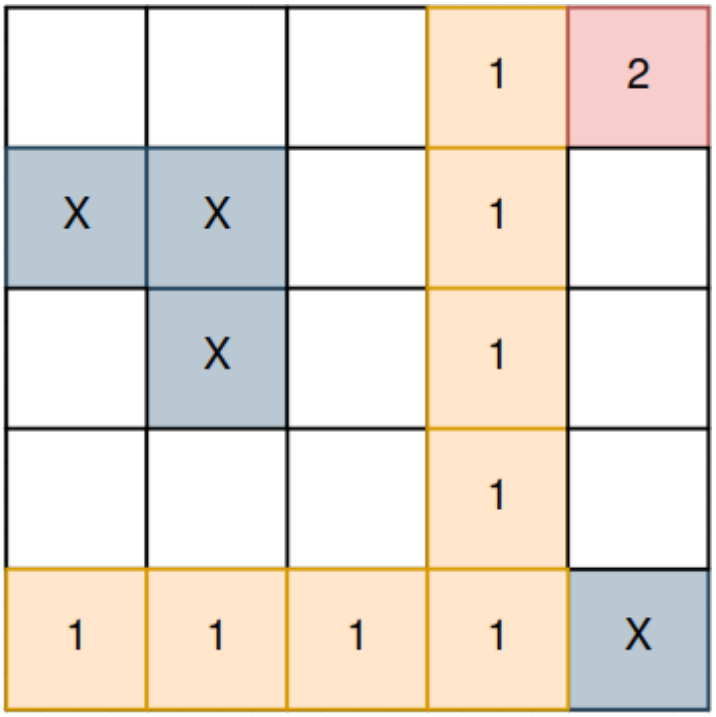
- **turn != -1** znamená, že vykonaná cesta bola validná, a preto sa môže inkrementovať atribút mnícha complete_rakes.
- **turn == -1** znamená, že sa mních zasekol vnútri záhradky, a preto sa cyklus hľadania riešenia **končí**.

Aktuálny stav atribútov mnícha:

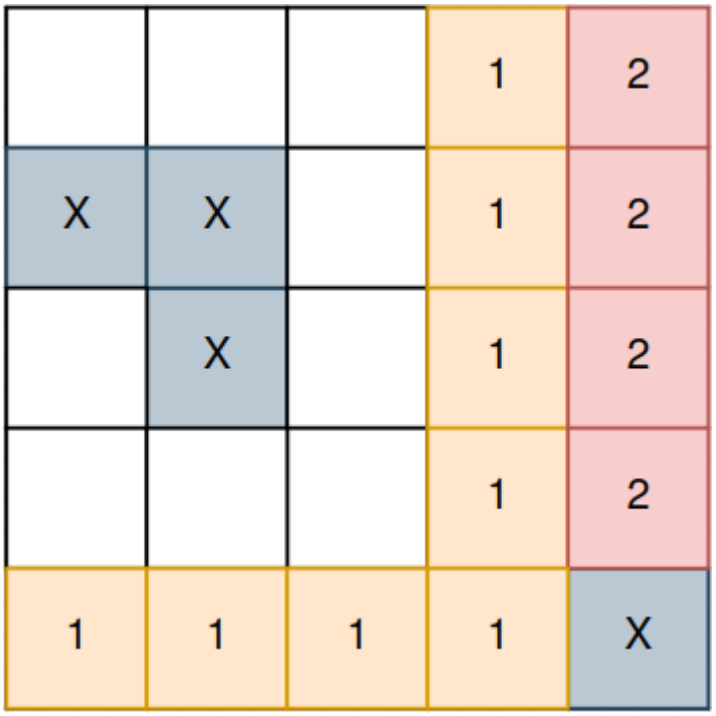
- **monk.fitness** = 8 (pripočítané tiles_raked)
- **monk.complete_rakes** = 1 (1 korektné hrabanie)
- **monk.correct.finish** = 0

- **monk.random_decisions** = [0.21, 0.64, 0.63, 0.78, 0.96, 0.68, 0.3] (nezmenené, lebo sa mních ešte nemusel rozhodovať)
- **monk.border_tiles** = [[4, 0, 0], **[0, 4, 1]**, [4, 1, 3], [0, 3, 1], [2, 4, 2], [2, 0, 0], [1, 4, 2], [4, 2, 3], [0, 4, 2], [0, 0, 0], [0, 1, 1], [0, 0, 1], [4, 3, 3], [3, 0, 0], [4, 0, 3], [0, 2, 1], [3, 4, 2]] (tento zoznam sa počas celého behu nezmení, no v ďalšom cykle sa vyberie zvýraznené políčko)

Vstup do záhradky pre **[0, 4, 1]**




Mních vstúpil do záhrady z pozície [0, 4] a bude pokračovať smerom dole [1] až kým nenarazí na prekážku.



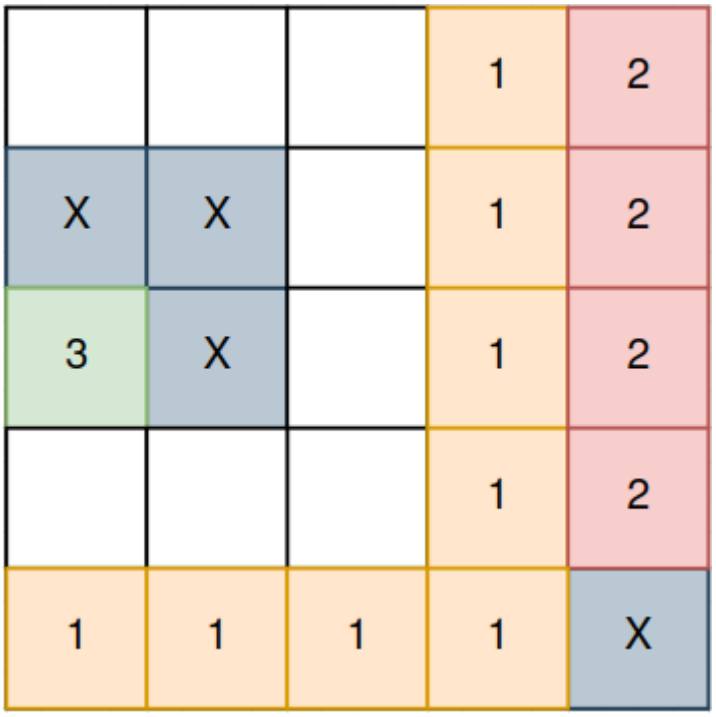

V tejto chvíli narazil na kameň na súradnici [3, 4] a premenné is_left_correct a is_right_correct budú evaluované ako False.
V takom prípade dochádza na podmienku, overenie, **či jedno z odbočení nevychádza z mapy**, čo sa vyhodnocuje ako korektný výstup. V tomto prípade ide o pohyb vľavo, ktorým sa mních validne dostane von z mapy.

Stav atribútov:

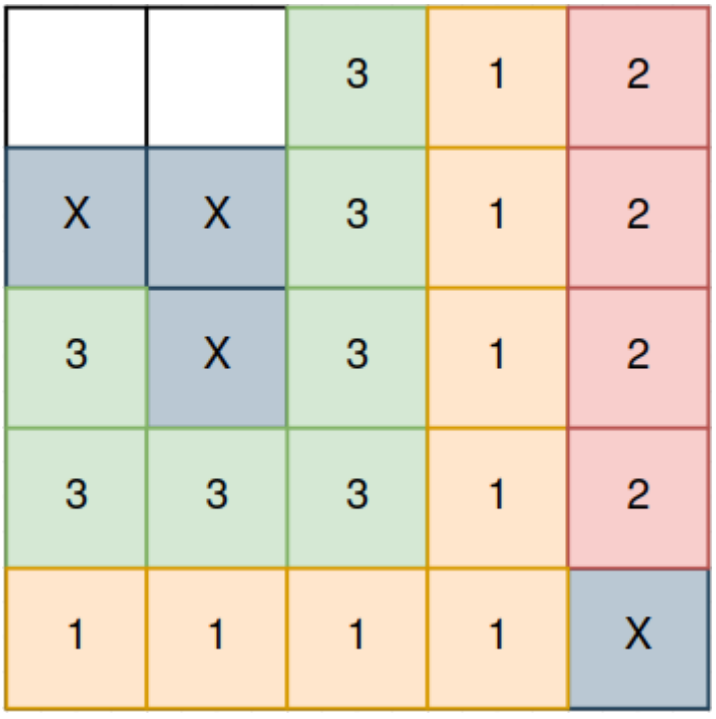
- **monk.fitness** = 12
- **monk.complete_rakes** = 2
- **monk.correct.finish** = 0
- **monk.random_decisions** = [0.21, 0.64, 0.63, 0.78, 0.96, 0.68, 0.3] (nezmenené, lebo sa mních ešte nemusel rozhodovať)
- **monk.border_tiles** = [[4, 0, 0], [0, 4, 1], **[4, 1, 3]**, **[0, 3, 1]**, **[2, 4, 2]**, [2, 0, 0], [1, 4, 2], [4, 2, 3], [0, 4, 2], [0, 0, 0], [0, 1, 1], [0, 0, 1], [4, 3, 3], [3, 0, 0], [4, 0, 3], [0, 2, 1], [3, 4, 2]]

Nasleduje vyhodnocovanie vyššie zvýraznených vstupných políčok. Tie budú všetky vyhodnotené ako ***unrakeable***, keďže sú blokované predchádzajúcimi ťahmi, a preto nevykonajú **nijakú zmenu** v záhradke.

Vstup pre **[2, 0, 0]**:



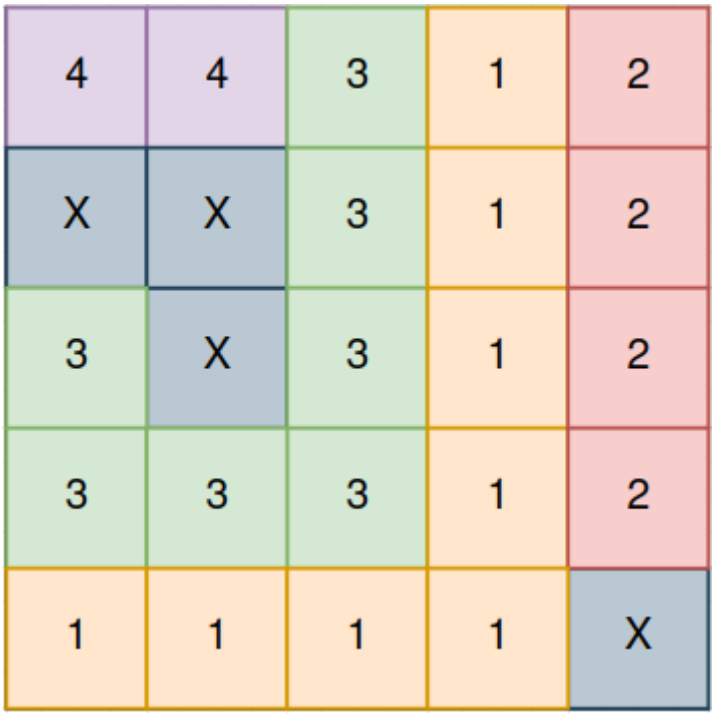

Následne mních vykoná niekoľko odbočovaní, konkrétne na políčku [2, 0] kvôli kameňom na [2, 1] a [1, 0] zabočí vpravo, na [3, 0] kvôli [4, 0] vľavo a na súradniciach [3, 2] opäť vľavo kvôli políčku [3, 3].



Stav atribútov:

- **monk.fitness** = 19
- **monk.complete_rakes** = 3
- **monk.correct.finish** = 0
- **monk.random_decisions** = [0.21, 0.64, 0.63, 0.78, 0.96, 0.68, 0.3]
- **monk.border_tiles** = [[4, 0, 0], [0, 4, 1], [4, 1, 3], [0, 3, 1], [2, 4, 2], [2, 0, 0], **[1, 4, 2], [4, 2, 3], [0, 4, 2],** [0, 0, 0], [0, 1, 1], [0, 0, 1], [4, 3, 3], [3, 0, 0], [4, 0, 3], [0, 2, 1], [3, 4, 2]]

Zvýraznené ťahy budú odignorované, lebo sú **blokované**. Nasledujúci validný vstup začne s **[0, 0, 0]**, odkiaľ bude pokračovať priamo doprava až kým nenarazí na pohrabané políčko, kde sa otočí vľavo a opustí záhradu.




Stav atribútov:

- **monk.fitness** = 21
- **monk.complete_rakes** = 4
- **monk.correct.finish** = 0
- **monk.random_decisions** = [0.21, 0.64, 0.63, 0.78, 0.96, 0.68, 0.3]
- **monk.border_tiles** = [[4, 0, 0], [0, 4, 1], [4, 1, 3], [0, 3, 1], [2, 4, 2], [2, 0, 0], [1, 4, 2], [4, 2, 3], [0, 4, 2], [0, 0, 0], **[0, 1, 1], [0, 0, 1], [4, 3, 3], [3, 0, 0], [4, 0, 3], [0, 2, 1], [3, 4, 2]]**

Na záver sa zvyšné vstupné políčka **odignorujú**, keďže ani jedno už nie je validné a máme hotové kompletné riešenie. Po výstupe z funkcie rake_wrapper() mapa ostane nezmenená a atribúty budú vyzerať nasledovne:

- **monk.fitness** = 21
- **monk.complete_rakes** = 4
- **monk.correct.finish** = 1
- **monk.random_decisions** = [0.21, 0.64, 0.63, 0.78, 0.96, 0.68, 0.3]
- **monk.border_tiles** = [[4, 0, 0], [0, 4, 1], [4, 1, 3], [0, 3, 1], [2, 4, 2], [2, 0, 0], [1, 4, 2], [4, 2, 3], [0, 4, 2], [0, 0, 0], [0, 1, 1], [0, 0, 1], [4, 3, 3], [3, 0, 0], [4, 0, 3], [0, 2, 1], [3, 4, 2]]

Toto riešenie je jedno z najlepších možných pre danú mapu, čo znamená, že sme našli **potenciálne globálne optimum** pri prvej iterácii.

*Pozn. V tomto riešení nedošlo k zmenám v atribúte random_decisions, keďže mních nenarazil na políčko, na ktorom mohol validne odbočiť vľavo aj vpravo. Ak by na také políčko narazil, atribút by sa zmenil nasledovne:*

- ***monk.random_decisions** = [0.21, 0.64, 0.63, 0.78, 0.96, 0.68, 0.3]*

---

- ***monk.random_decisions** = [0.64, 0.63, 0.78, 0.96, 0.68, 0.3, 0.21]*

*kde prvé rozhodnutie by bolo premiestnené na koniec poľa.*

## 2.3 Ako sa vyberajú riešenia

### 2.3.1 Flow-chart

Princíp vyberania riešení v hlavnej funkcii **simulated_annealing()** je zobrazený vo flow-charte nižšie, no ešte pred samotným flow-chartom je potrebné vedieť informácie o dôležitých premenných, ktoré sa v hlavnej funkcii používajú:

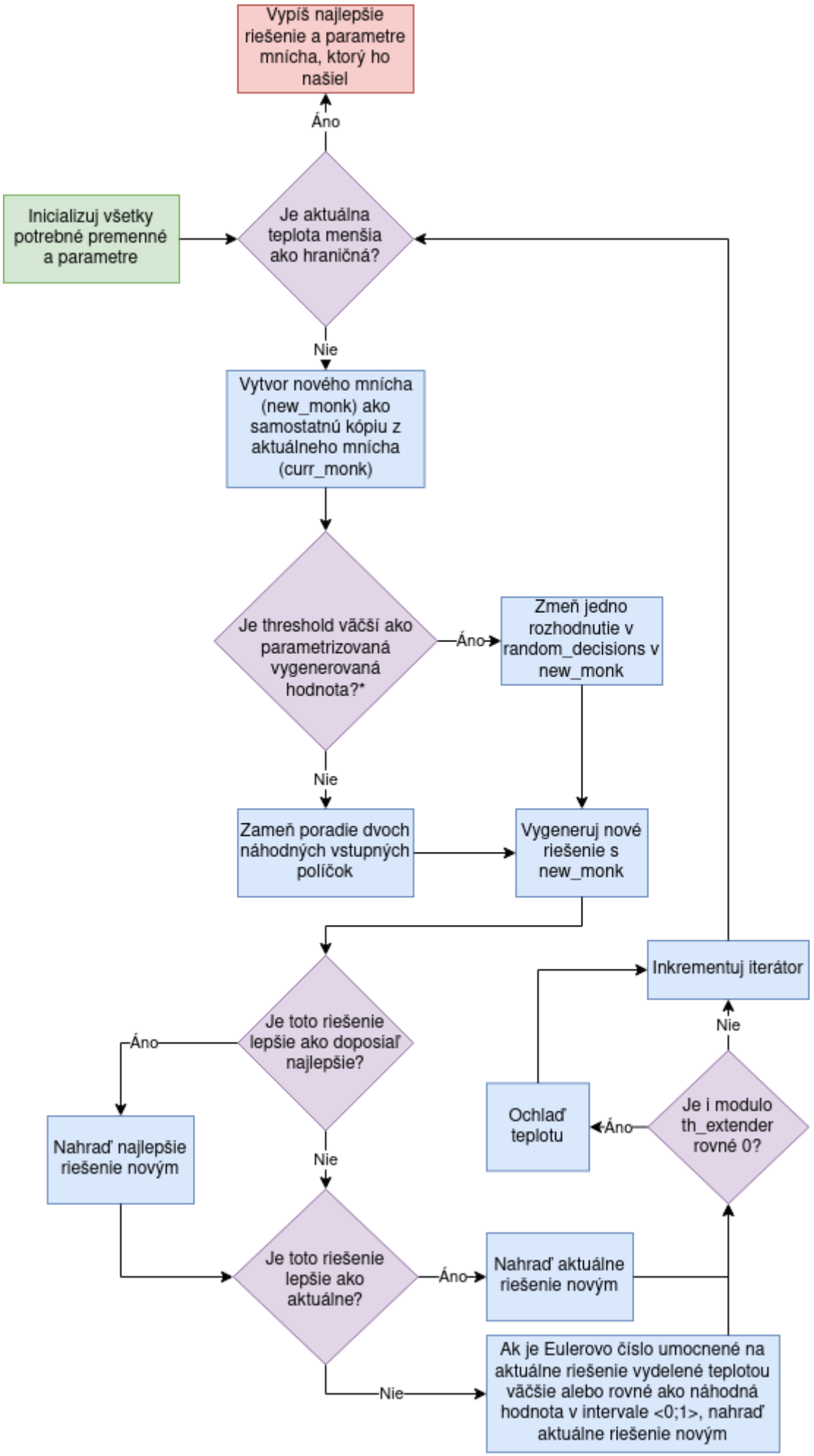
- **main_garden** - hlavná mapa bez akýchkoľvek ťahov, ktorá bola načítaná zo súboru alebo náhodne vygenerovaná
- **dimensions** - rozmery main_garden
- **initial_temperature** - počiatočná teplota, ktorá bola zvolená používateľom alebo bola načítaná základná hodnota (30)
- **th_extender** - skratka pre threshold_extender, kde je celočíselná hodnota, ktorá znižuje hodnotu thresholdu a periódu ochladzovania, čím zaručuje pomalšie ochladzovanie a väčší počet iterácií
- **threshold** - hraničná hodnota teploty, ktorá určuje termináciu behu algoritmu
- **max_fitness** - maximálne možné ohodnotenie pre main_garden
- **temperature** - teplota, podľa ktorej sa určuje šanca pre akceptáciu horšieho riešenia
- **i** - iterátor na určenie počtu iterácií
- **best_diff** - rozdiel ohodnotení medzi novým a najlepším riešením
- **curr_diff** - rozdiel ohodnotení medzi novým a aktuálnym riešením

Dôležité inštancie tried:

- **best_monk** - doposiaľ najlepšie nájdené riešenie vôbec, ktoré sa po dosiahnutí thresholdu vypíše
- **curr_monk** - aktuálne najlepšie riešenie, ktoré sa líši od best_monk len tým, že sa doň môže dostať aj horšie riešenie
- **new_monk** - najnovšie riešenie

Legenda pre bunky flow-chartu vyberania riešení:

- **zelené** - inicializačné
- **červené** - ukončovacie
- **svetlo-modré** - vykonávacie
- **fialové** - rozhodovacie


Vypíš najlepšie riešenie a parametre mnícha, ktorý ho našiel
Áno
Inicializuj všetky potrebné premenné a parametre
Je aktuálna teplota menšia ako hraničná?
Nie
Vytvor nového mnícha (new_monk) ako samostatnú kópiu z aktuálneho mnícha (curr_monk)
Je threshold väčší ako parametrizovaná vygenerovaná hodnota?*
Áno
Zmeň jedno rozhodnutie v random_decisions v new_monk
Nie
Zameň poradie dvoch náhodných vstupných políčok
Vygeneruj nové riešenie s new_monk
Inkrementuj iterátor
Nie
Je toto riešenie lepšie ako doposiaľ najlepšie?
Áno
Nahraď najlepšie riešenie novým
Nie
Ochlaď teplotu
Áno
Je i modulo th_extender rovné 0?
Je toto riešenie lepšie ako aktuálne?
Áno
Nahraď aktuálne riešenie novým
Nie
Ak je Eulerovo číslo umocnené na aktuálne riešenie vydelené teplotou väčšie alebo rovné ako náhodná hodnota v intervale <0;1>, nahraď aktuálne riešenie novým

** Pozn. toto je parametrizovaná vygenerovaná hodnota:*

```
uniform(0, max(new_monk.random_decisions) + threshold)
```

*ktorá v danej podmienke zabezpečuje náhodné rozhodnutie, ktoré je ovplyvnené parametrom. Podmienka je vyhodnotená ako pravdivá v približne 10-15% prípadoch.*

V zelenej bunke flow-chartu, ktorá je inicializačná, sa načítajú hodnoty pre nasledovné parametre:

- **initial_temperature**, **th_extender**, **threshold**, **max_fitness**, **starter_tiles**, **random_decisions**, best_monk, best_garden, current_garden, current_monk, temperature, i

Oranžovou zvýraznené parametre sú konštanty, ktoré sa počas behu programu nezmenia ani raz.
Modrou zvýraznené premenné sa hneď pri inicializácii “naplnia” do inštancií best_monk a curr_monk, ktoré si ich budú v algoritme meniť na úrovni svojich vlastných atribútov.

### 2.3.2 Cyklus vyberania riešení

Ako bolo vyššie spomenuté pri definícii dôležitých parametrov, **threshold** je hraničná hodnota teploty, ktorá určuje terminániu behu algoritmu. Ak sa teplota, ktorá sa kontinuálne ochladzuje, dostane pod úroveň hodnoty v premennej threshold, program končí cyklus vyberania riešení a prechádza na výpis najlepšieho nájdeného riešenia, ktoré sa nachádza v premenných **best_monk** a **best_garden**.

Samotný **predpis cyklu** vyberania riešení vyzerá nasledovne:

```
while temperature > threshold:
```

a **ochladzovanie teploty**, ktoré v konečnom stave spôsobí terináciu programu pri prekročení podmienke v cykle, vyzerá takto:

```
if i % th_extender == 0:
    temperature = temperature * (0.9992 ** (i ** threshold))
```

V tejto podmienke je aj jedna z úloh **th_extenderu**, ktorý, ako z názvu vyplýva, **posúva threshold do nižších hodnôt**, t.j. zabezpečuje viac iterácií. Jeho druhá funkcia je pri samotnom výpočte threshold, kde sa výpočet delí th_extenderom, vďaka čomu je finálna hodnota thresholdu **nižšia**, ako by bola bez tohto delenia.

Cyklus vyberania riešení začína vytvorením **exaktnej kópie z aktuálneho mnícha** do novej inštancie, ktorá sa volá **new_monk**.
Za tým nasleduje úkon minimálnej zmeny, resp. **vytvorenie suseda** k aktuálnemu mníchovi. To sa docieli buď zmenou jedného rozhodnutia v poli new_monk.random_decisions, alebo výmenou poradia dvoch vstupných políčok v atribúte new_monk.border_tiles.
S takým susedom nasleduje **generovanie riešenia**, ktorého proces je popísaný v časti 2.2.

Po nájdení riešenia sa **evaluuje jeho fitness** v porovnaní s doposiaľ najlepším riešením odčítaním hodnoty fitness a correct_finish. Vo všeobecnosti platí takáto schéma:

- Ak je súčet new_monk.fitness a new_monk.correct_finish väčší ako súčet best_monk.fitness a best_monk.correct_finish, našlo sa nové najlepšie riešenie, a preto sa hodnoty v inštancii best_monk aktualizujú podľa atribútov nového mnícha (new_monk)
- Ak je súčet new_monk.fitness a new_monk.correct_finish rovný súčtu best_monk.fitness a best_monk.correct_finish, nasleduje porovnanie hodnôt v atribúte complete_rakes. Vtedy platí, že ak má new_monk nižšiu hodnotu complete_rakes ako best_monk, našli sme lepšie riešenie a aktualizujú sa hodnoty v inštancii best_monk.
- Ak je súčet iný od predchádzajúcich možností, pokračuje sa ďalej.

Nasleduje **porovnanie nového riešenia s aktuálnym riešením**, kde platia rovnaké podmienky (namiesto best_monk sa porovnáva s curr_monk) s jednou obmenou v tretej z nich.

- Ak je súčet new_monk.fitness a new_monk.correct_finish menší ako súčet current_monk.fitness a current_monk.correct_finish, akceptujeme riešenie len vtedy, **ak je Eulerovo číslo umocnené na aktuálne riešenie vydelené teplotou väčšie alebo rovné ako náhodná hodnota v intervale <0;1>.**

Po porovnaní s aktuálnym riešením nasleduje už len prípadné **ochladenie teploty a inkrementácia iterátora**. Tieto dva príkazy ukončujú celý beh jedného cyklu, ktorý pri novej iterácii opäť začína porovnaním, či je temperature nižšia ako stanovený threshold.

# 3. Používateľské rozhranie

Používateľské rozhranie je jednoduché - orientované len na operácie v **konzole**.
Ihneď po spustení programu sa objaví nasledovná správa:

```
------------------------------------------------

* Zen's garden (Simulated annealing)
* Author: Marko Stahovec

------------------------------------------------
Load map from file? [y/n]:
```

Používateľ je ponúknutý prvou z možností, ktorá hovorí, či chce **načítavať mapu z externého súboru** alebo nie. Pri výbere možnosti áno je používateľ ponúknutý ďalšou možnosťou:

```
------------------------------------------------

* Zen's garden (Simulated annealing)
* Author: Marko Stahovec

------------------------------------------------
Load map from file? [y/n]: y
File name (ex. 'map1'):
```

aby zadal názov súboru, z ktorého chce načítavať. Následne mu je poskytnutá ponuka, v ktorej si môže zvoliť, **či chce nastaviť parametre ručne** alebo nie.

```
------------------------------------------------

* Zen's garden (Simulated annealing)
* Author: Marko Stahovec

------------------------------------------------
Load map from file? [y/n]: y
File name (ex. 'map1'): map4
Set annealing settings manually? [y/n]:
```

Pri zvolení kladnej odpovede je mu ponúknutá **možnosť výberu konkrétnej hodnoty pre premennú temperature a th_extender**, ktorými kontroluje komplexitu vykonávania tohto algoritmu.

```
------------------------------------------------

* Zen's garden (Simulated annealing)
* Author: Marko Stahovec

------------------------------------------------
Load map from file? [y/n]: y
File name (ex. 'map1'): map4
Set annealing settings manually? [y/n]: y
Choose temperature [2-100]: 25
Choose threshold extender (default is 4) [1-100]: 5
```

Následne sa algoritmus spustí aj s výpisom počiatočnej mapy. Ukážkový výstup vyzerá nasledovne:

```
Set annealing settings manually? [y/n]: y
Choose temperature [2-100]: 25
Choose threshold extender (default is 4) [1-100]: 5
-----------------------------------------------

***************** INITIAL_MAP *****************

-----------------------------------------------

0   0   0   0   0
X   X   0   0   0
0   X   0   0   0
0   0   0   0   0
0   0   0   0   X

-----------------------------------------------

***************** FINAL_MAP *****************

-----------------------------------------------

4   4   2   1   3
X   X   2   1   3
2   X   2   1   3
2   2   2   1   3
1   1   1   1   X


NO. OF ITERATIONS: 17525
FINAL FITNESS: 21
MAX FITNESS: 21
COMPLETE RAKES: 4
FINISHED CORRECTLY: YES

end
```

kde používateľ okrem výslednej mapy dostane aj **krátku štatistiku o mníchovi a počte iterácií**, ktoré program vykonal so zadanými parametrami.
Postup pri iných odpovediach je analogický a priamočiary, ak sa používateľ rozhodne pre **nenačítavanie** zo súboru, zadá počet riadkov a stĺpcov a hustotu kameňov cez hyperparameter rock_density. Tieto požiadavky spracuje a vytvorí požadovanú mapu, napr:

```
------------------------------------------------

* Zen's garden (Simulated annealing)
* Author: Marko Stahovec

------------------------------------------------
Load map from file? [y/n]: n
Choose no. of columns: 6
Choose no. of rows: 6
Select rock density [1-5]: 2
Set annealing settings manually? [y/n]: n
------------------------------------------------

****************** INITIAL_MAP ******************

------------------------------------------------

0   0   0   0   0   0
0   0   X   0   0   0
0   0   0   0   0   0
0   0   0   0   0   0
0   0   0   0   0   0
0   0   0   0   0   0
```

```
-----------------------------------------------

***************** FINAL_MAP *****************

-----------------------------------------------

3   4   4   4   3   2
3   4   X   4   3   2
3   4   4   4   3   2
3   3   3   3   3   2
2   2   2   2   2   2
1   1   1   1   1   1


NO. OF ITERATIONS: 17697
FINAL FITNESS: 35
MAX FITNESS: 35
COMPLETE RAKES: 4
FINISHED CORRECTLY: YES

end
```

# 4. Testovanie

Táto časť je vyhradená na testovanie a zhrnutie poznatkov z neho.
Testovanie prebehlo na **nasledovných 3 mapách**:

Mapa č.1 (rozmer 6x6, počet kameňov 4) (map30.txt):

```
0   0   0   0   0   0
X   X   0   0   0   0
0   X   0   0   0   0
0   0   0   0   0   0
0   0   0   0   X   0
0   0   0   0   0   0
```

Mapa č.2 (rozmer 10x10, počet kameňov 6) (map31.txt):

```
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
0   0   0   0   X   0   0   0   0   0
0   0   X   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   X   0   0
0   0   0   0   0   0   0   0   0   X
0   0   0   0   0   0   0   0   0   0
X   0   0   0   0   0   0   0   0   0
X   0   0   0   0   0   0   0   0   0
```

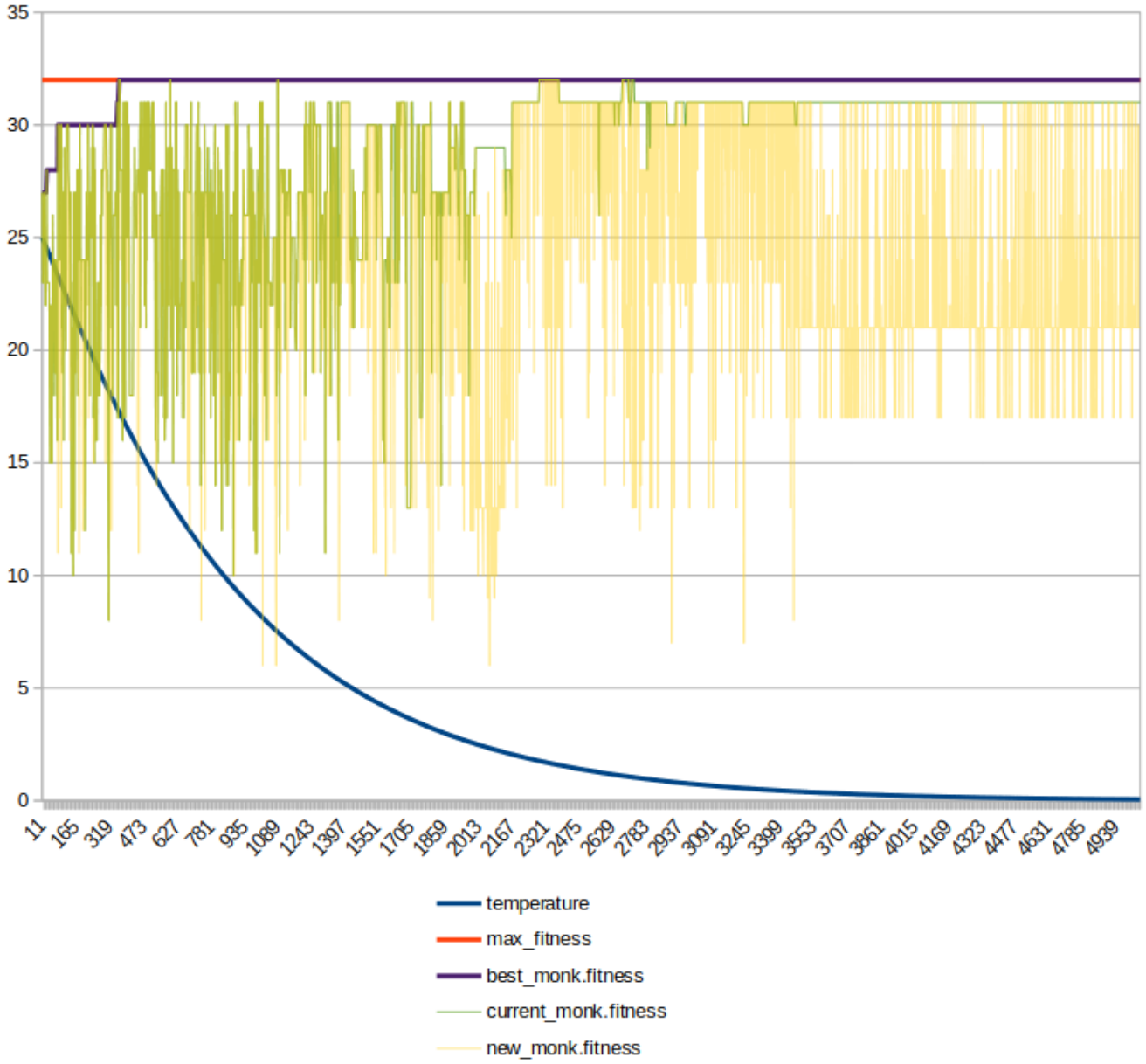
Mapa č.3 (rozmer 14x14, počet kameňov 22) (map32.txt):

```
----------------------------------------------

0   0   0   0   0   0   X   0   0   0   0   0   0   0
0   0   0   0   0   0   X   0   0   0   0   0   0   0
0   0   0   0   0   0   X   0   0   0   0   0   0   0
0   0   0   0   0   0   X   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   X   X   0   0   0
0   0   0   0   0   0   0   X   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   X   X   X   0   0   0   0   0
0   0   X   X   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   X   X   X   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   X   0   0
0   0   0   0   0   0   0   0   0   0   0   X   0   0
0   0   0   0   0   0   0   0   0   0   0   X   0   0
0   0   0   X   X   0   0   0   0   0   0   X   0   X

----------------------------------------------
```

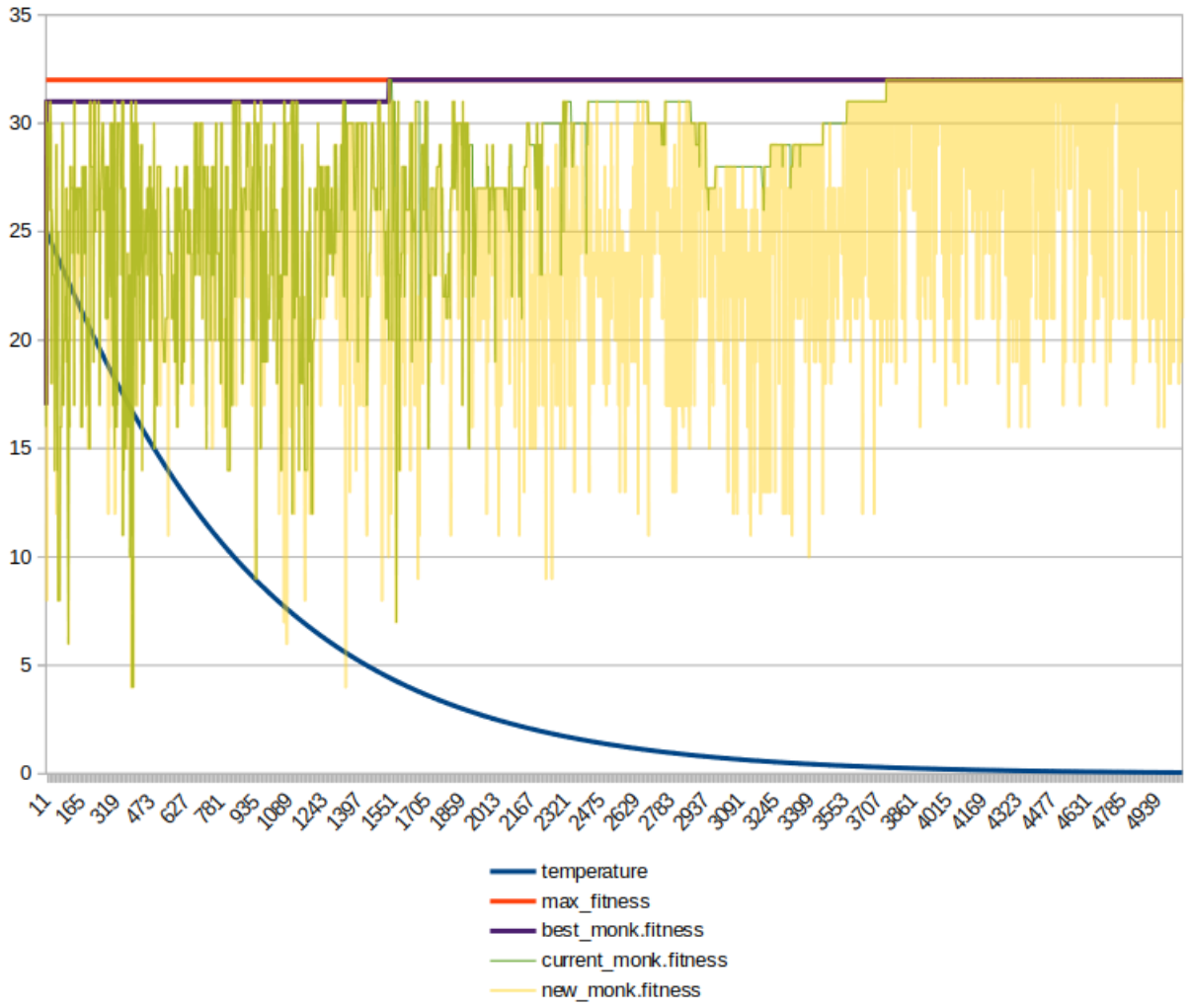
## 4.1 Testovanie konvergencie

Ako prvým objektom testovania som si zvolil ukážku **konvergencie** na rôznych mapách s **počiatočnou teplotou 25** a **minimálnym th_extenderom** z dôvodu čitateľnosti grafu. Na osi **x** je hodnota **iterátoru** a na osi **y** je ohodnotenie **fitness**, resp. pre temperature je to jej hodnota.

Z grafu vidíme veľmi rýchlu konvergenciu k hodnote max_fitness, čo je cieľom tohto algoritmu. Následne program bežal ďalej a hľadal rôzne ďalšie riešenia, ktoré by mohli vylepšiť to najlepšie z hľadiska počtu ťahov, čo v danom grafe zobrazené nie je, no takýto prístup je aj tak korektný, keďže nechávame algoritmus “**vychladnút**” až po jeho hranicu.



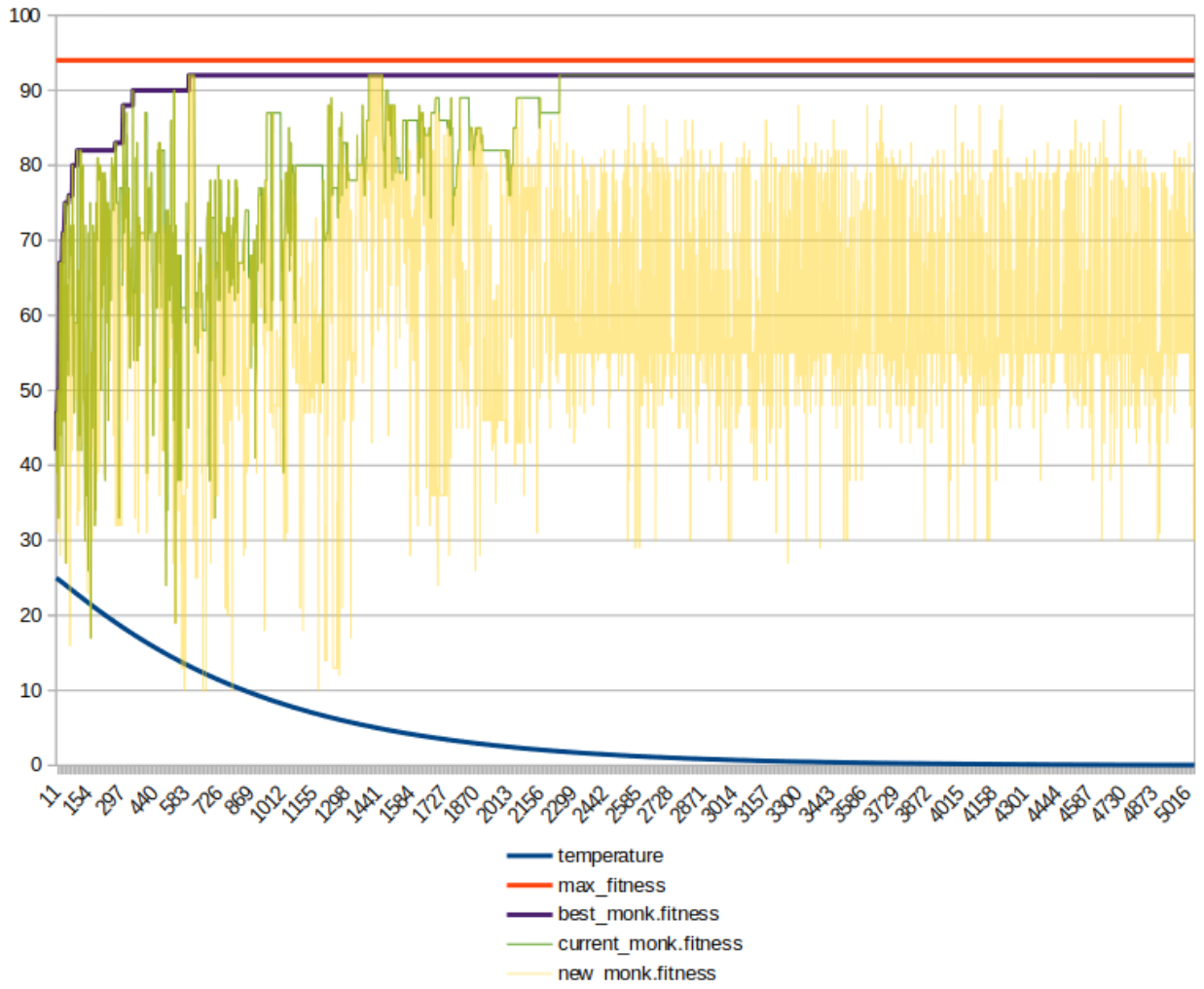

V 2. behu pre tú istú mapu možno pozorovať rovnakú tendenciu pre ohodnotenie best_monk.fitness. Rovnako **konverguje aj current_monk.fitness**, ktorý kvôli znižujúcej sa teplote prestane akceptovať horšie riešenia ku koncu behu programu.



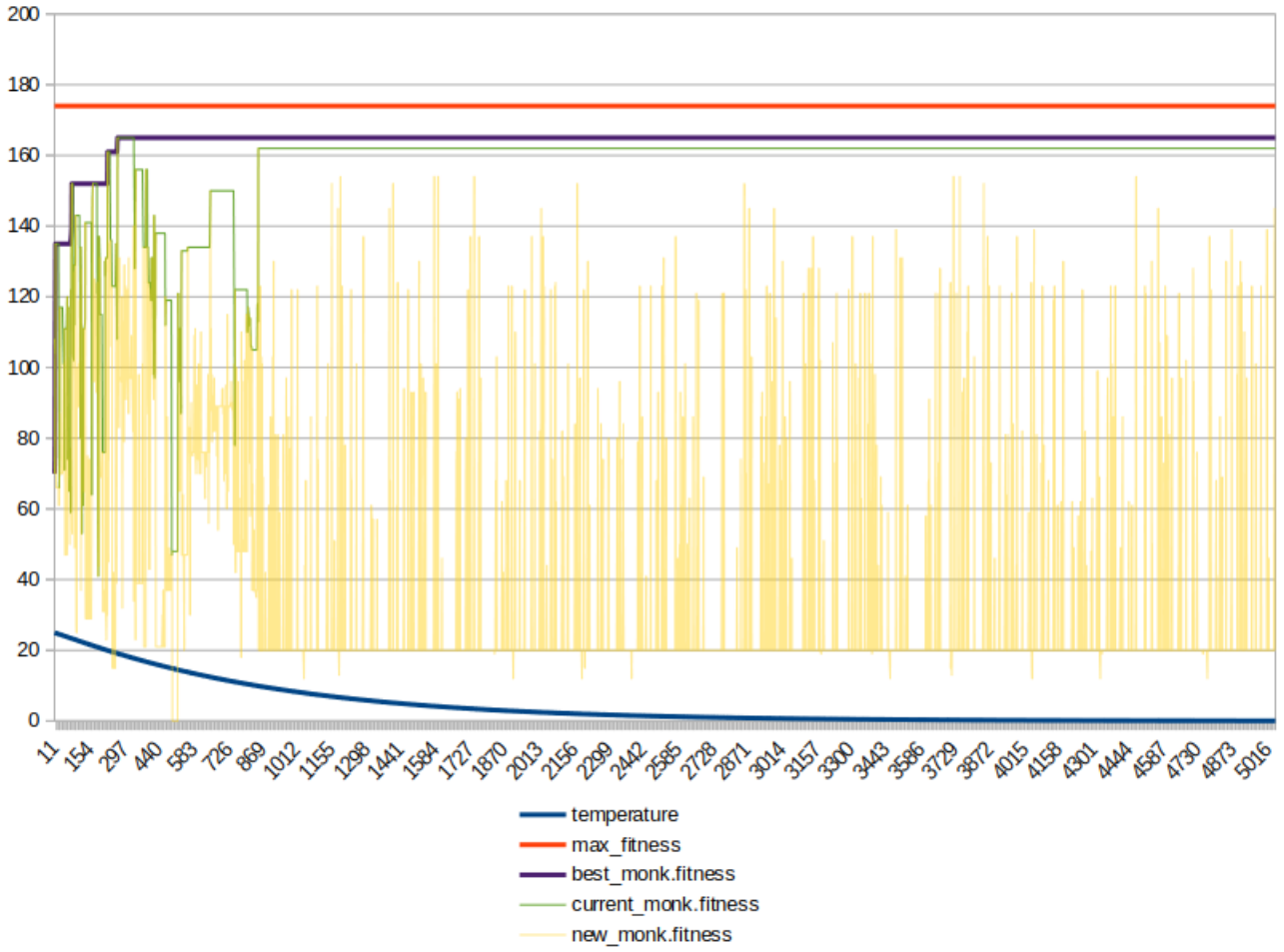

Pre mapu č.2, ktorá má rozmery 10x10, bolo vykonané rovnaké testovanie, v ktorom je možné pozorovať konvergenciu rôznych fitnessov a teploty. Pri väčšom rozmere boli očakávané miernejšie konvergencie, resp. graduálnejšie približovanie sa maximálnej hodnote fitness až neskôr.

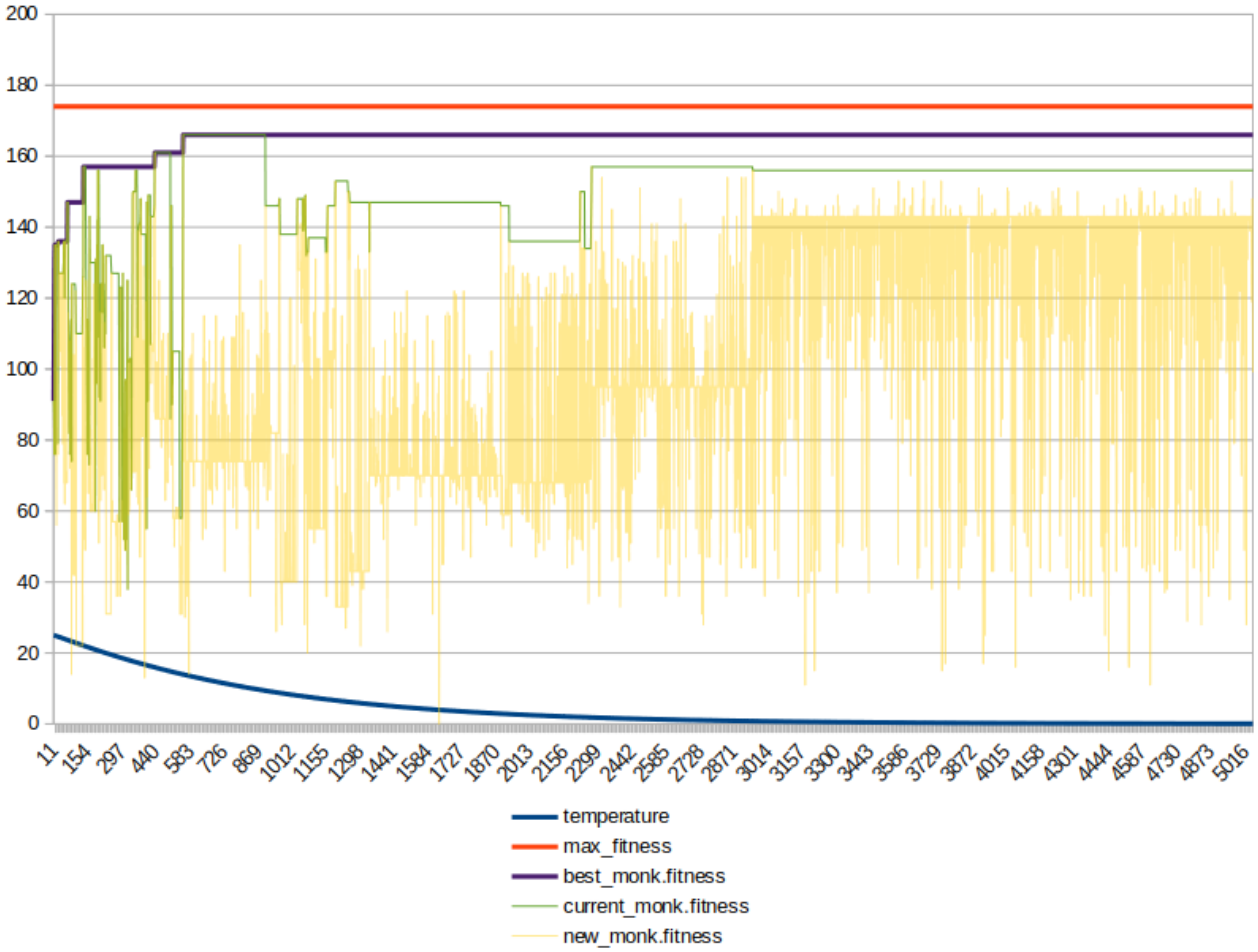
Tento predpoklad sa zároveň aj potvrdil, keďže v prvom behu algoritmus našiel **kompletné riešenie až v druhej polovici iterácií** a v druhom dokonca ani nenašiel, keďže sa podľa všetkého **zasekol v lokálnom maxime**, čo je možno odčítať z hodnôt current_monk.fitness, keďže tie vôbec nefluktuovali do nižších rozmedzí počas mnohých iterácií. Fakt, že sa riešenie nenašlo, je spôsobený tým, že **parametre teploty a th_extenderu sú príliš nízke**, no na ukážku konvergencie parametrov primerané.


100
90
80
70
60
50
40
30
20
10
0
11 143 275 407 539 671 803 935 1067 1199 1331 1463 1595 1727 1859 1991 2123 2255 2387 2519 2651 2783 2915 3047 3179 3311 3443 3575 3707 3839 3971 4103 4235 4367 4499 4631 4763 4895 5027
temperature
max_fitness
best_monk.fitness
current_monk.fitness
new_monk.fitness


100
90
80
70
60
50
40
30
20
10
0
11 154 297 440 583 726 869 1012 1155 1298 1441 1584 1727 1870 2013 2156 2299 2442 2585 2728 2871 3014 3157 3300 3443 3586 3729 3872 4015 4158 4301 4444 4587 4730 4873 5016
temperature
max_fitness
best_monk.fitness
current_monk.fitness
new_monk.fitness

Pre mapu č.3 s rozmermi 14x14 boli vykonané identické testy konvergencie atribútov. Ako možno pozorovať na oboch grafoch, tak **riešenie nájdené nebolo** ani v jednom prípade. Fitness najlepšieho riešenia sa postupne menil, keďže sa nachádzali lepšie a lepšie riešenia, no globálne maximum nebolo dosiahnuté, keďže hyperparametre boli nastavené na príliš nízke hodnoty.






Z daných grafov je možné usúdiť, že sa fitness mení korektným, resp. očakávaným spôsobom. Ďalšie fakty, ktoré sme zistili sú tie, že veľkosť šachovnice vplýva na zložitosť a že nízke hodnoty hyperparametrov majú tendenciu zaseknúť sa v lokálnom optime.

## 4.2 Testovanie rôznych hyperparametrov

Testovanie s rôznymi hyperparametrami má za cieľ ukázať dôležitosť správneho nastavenia teploty a th_extenderu a ich dôsledky na **najlepšie riešenia**.
Toto testovanie je rozdelené do štyroch podčastí, v ktorých sa testuje vždy iná kombinácia hyperparametrov:

- **nízka temperature a nízky th_extender**

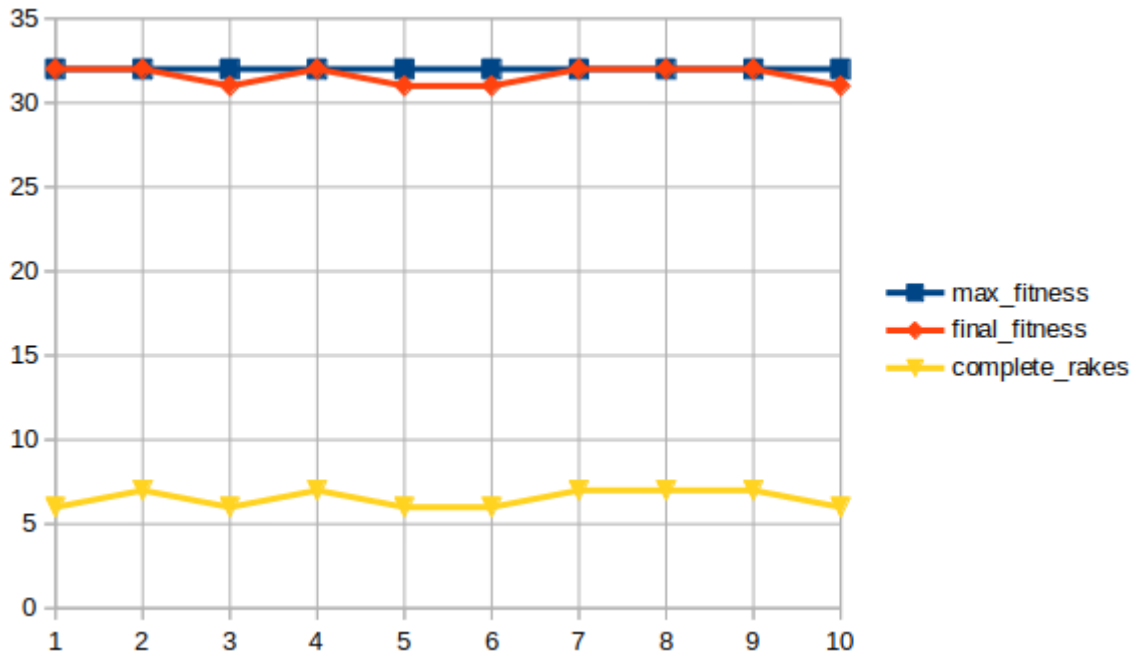
- **vysoká temperature a nízky th_extender**
- **nízka temperature a vysoký th_extender**
- **vysoká temperature a vysoký th_extender**

Všetky tieto podčasti budú otestované na 3 mapách, ktoré sú identické ako tie z predchádzajúcej časti.
*Pozn. Všetky hodnoty v nasledujúcich tabuľkách predstavujú hodnoty najlepšieho mnícha po ukončení programu.*

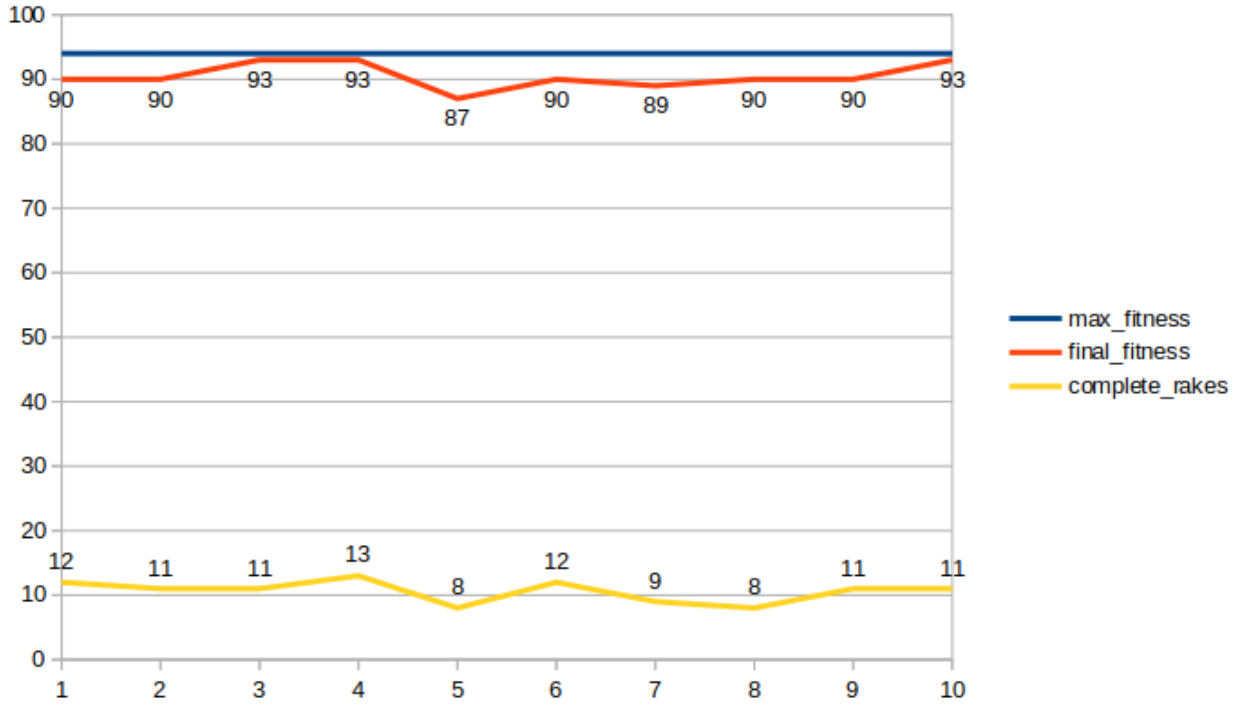
### 4.2.1 Nízka temperature a nízky th_extender

- **temperature** = 5
- **th_extender** = 2

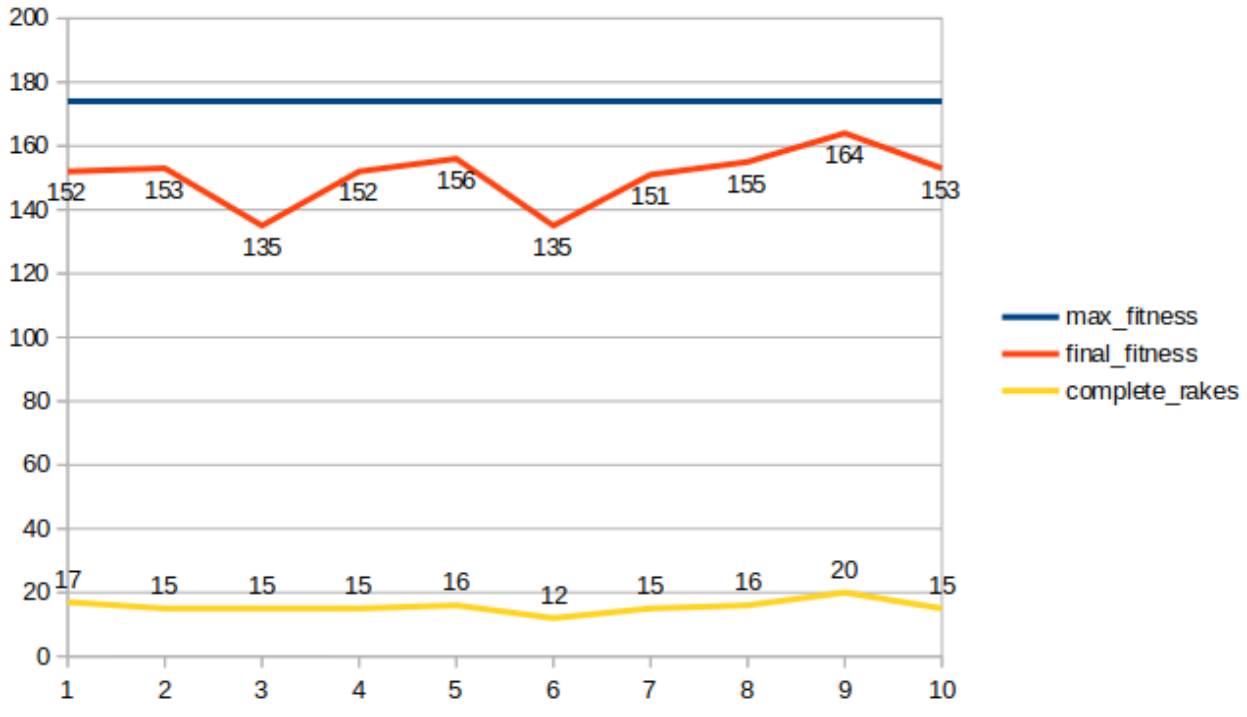
Počet iterácií pre danú kombináciu je vždy rovný **2130**.
Mapa č.1 (rozmer 6x6, počet kameňov 4) (map30.txt):



Mapa č.2 (rozmer 10x10, počet kameňov 6) (map31.txt):




Mapa č.3 (rozmer 14x14, počet kameňov 22) (map32.txt):

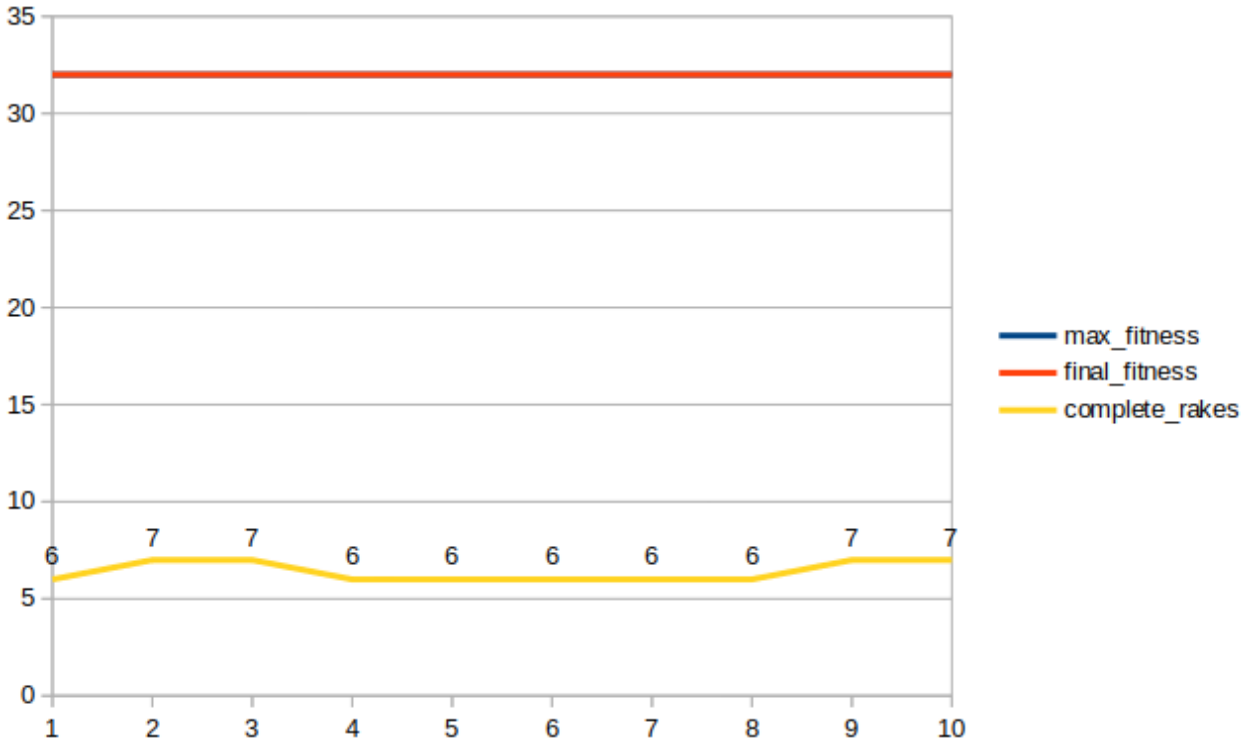


Ako možno odčítať z grafov, príliš nízke hodnoty hyperparametrov sú dostačujúce iba pre **veľmi malé rozmery** záhrad. Pri komplexných mapách to už zostáva len na náhodu, či sa nájde dobré riešenie na začiatku hľadania alebo nie.

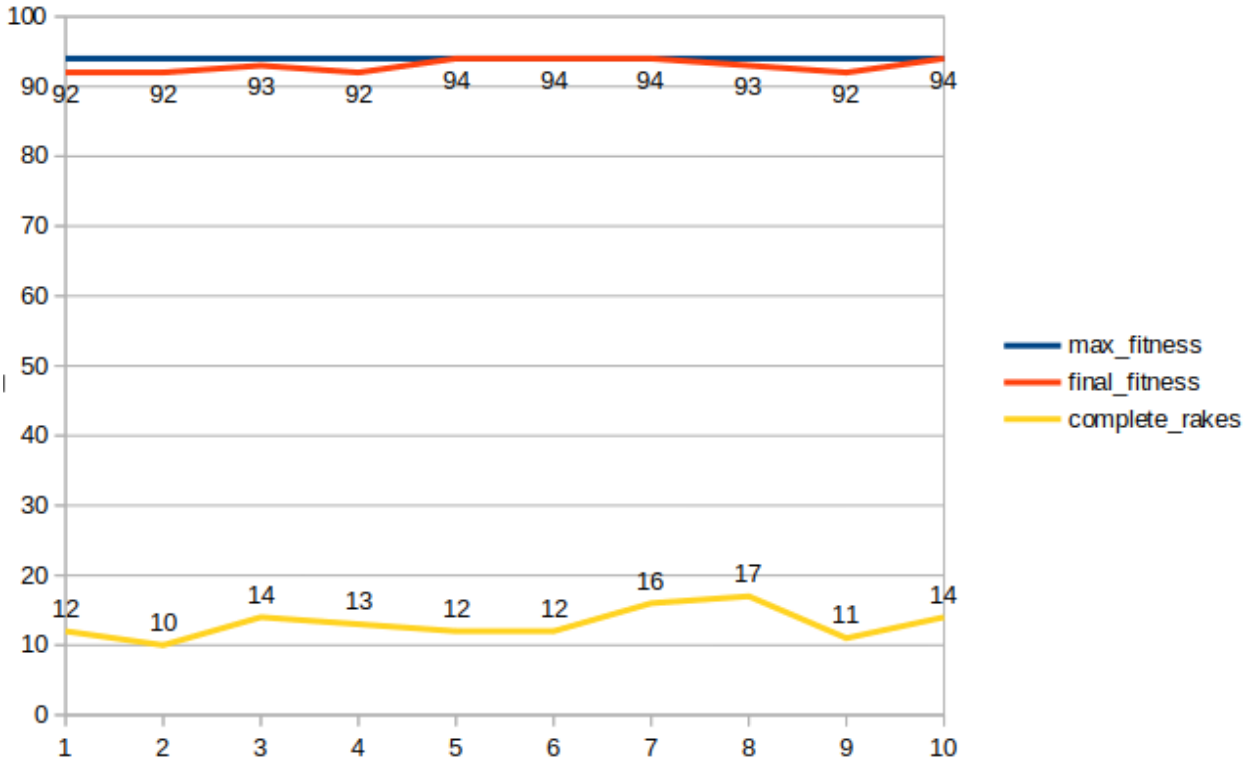
### 4.2.2 Vysoká temperature a nízky th_extender

- **temperature** = 50
- **th_extender** = 2

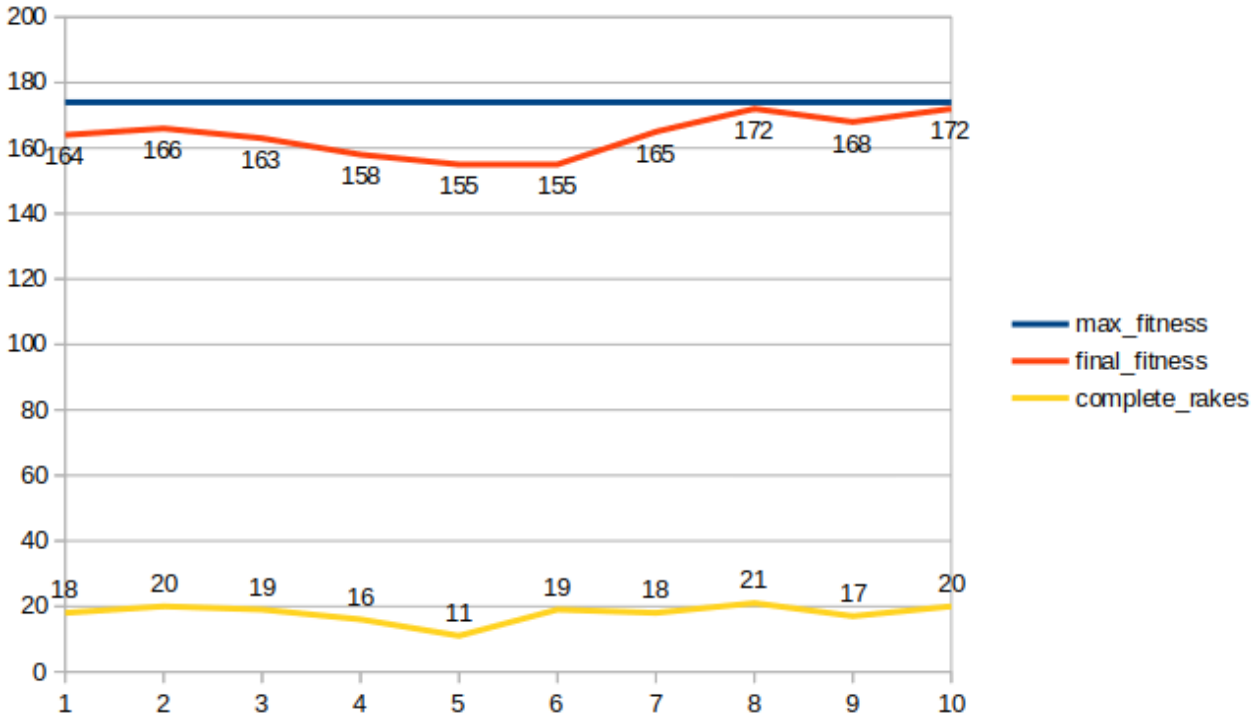
Počet iterácií pre danú kombináciu je vždy rovný **9007**.
Mapa č.1 (rozmer 6x6, počet kameňov 4) (map30.txt):




Mapa č.2 (rozmer 10x10, počet kameňov 6) (map31.txt):




Mapa č.3 (rozmer 14x14, počet kameňov 22) (map32.txt):



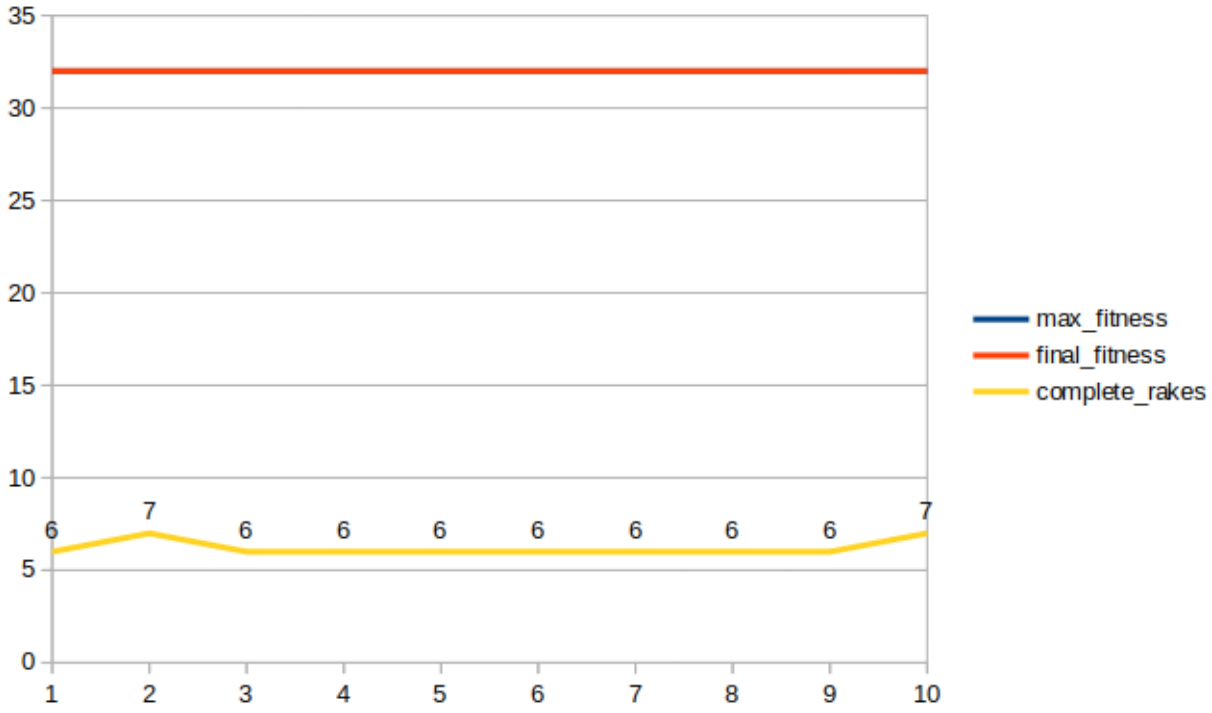

Teplota o hodnote 50 zaručuje dostatočné “**shufflovanie**” rôznych riešení na začiatku algoritmu, vďaka čomu vie program s väčšou pravdepodobnosťou

identifikovať vrchol s **globálnym optimom**. Pri mape o rozmeroch 10x10 sú už tieto parametre takmer dostačujúce, no predpokladáme, že program by potreboval viac iterácií na to, aby mohol použiť svoje hill-climbing schopnosti na výstup až k cieľu. Práve na to slúži hyperparameter th_extender, ktorého hodnoty sú zvýšené v nasledujúcej sekcii.

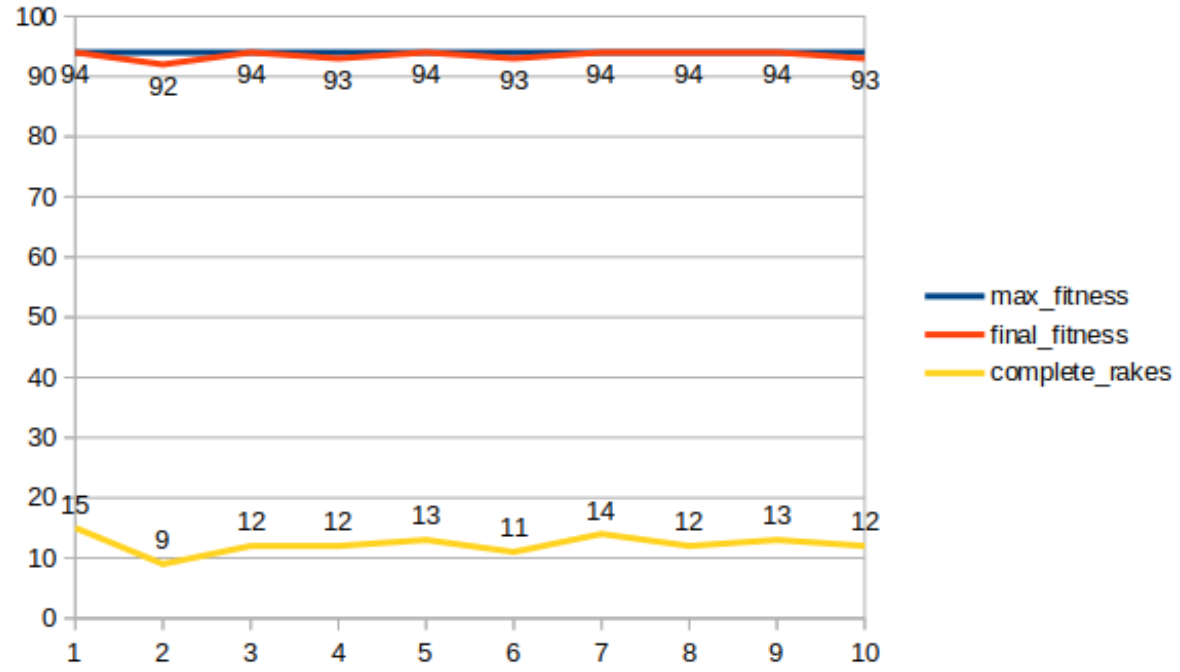
### 4.2.3 Nízka temperature a vysoký th_extender

- **temperature** = 5
- **th_extender** = 100

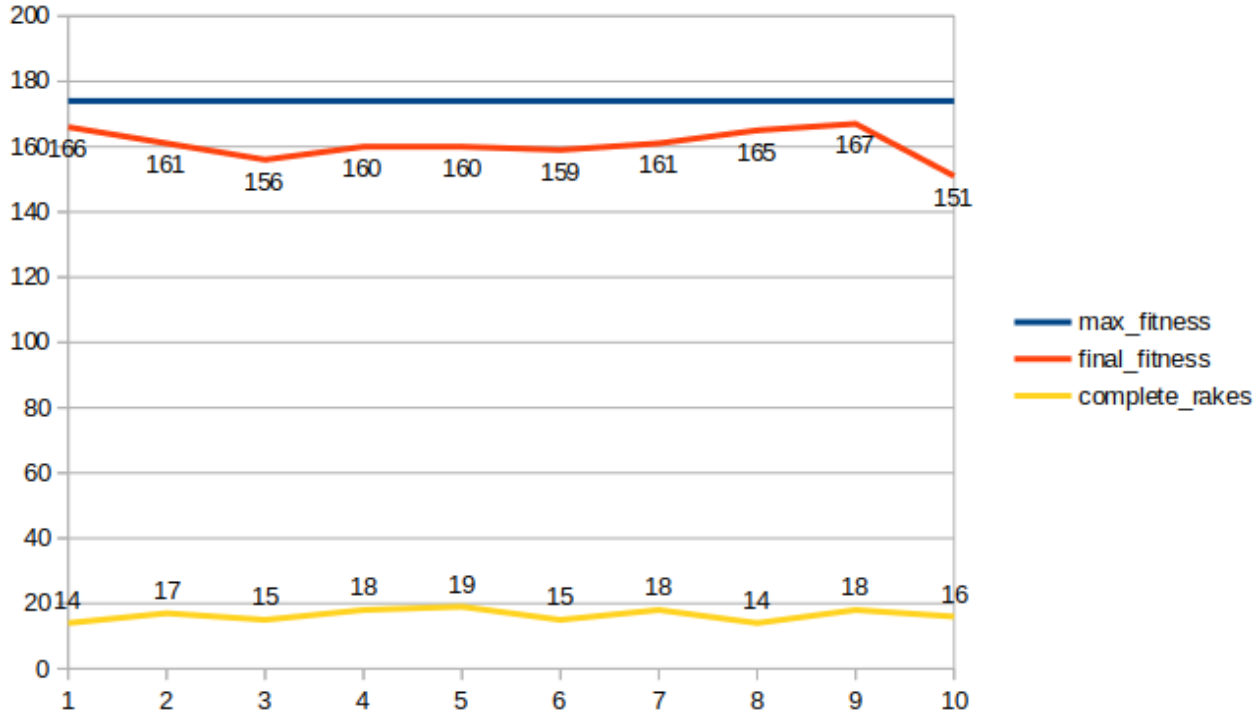
Počet iterácií pre danú kombináciu je vždy rovný **93431**.
Mapa č.1 (rozmer 6x6, počet kameňov 4) (map30.txt):




Mapa č.2 (rozmer 10x10, počet kameňov 6) (map31.txt):




Mapa č.3 (rozmer 14x14, počet kameňov 22) (map32.txt):



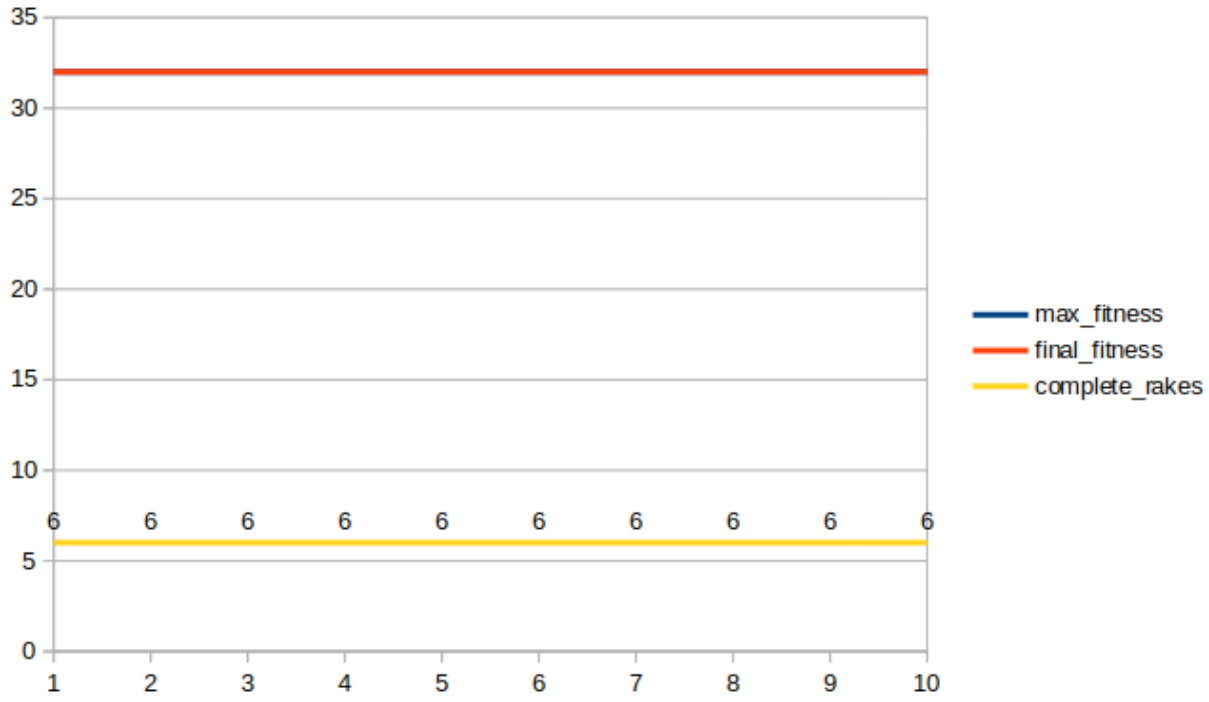

Na grafoch pozorujeme viditeľné zlepšenie zvýšením hodnoty parametra th_extender. Program dosahuje obdobné výsledky ako v sekcii 4.2.2, no v priemere

**nachádzal lepšie riešenia pri menších mapách** ako pri väčších. Môžeme teda očakávať, že zvýšením oboch parametrov dostaneme najlepšie výsledky.

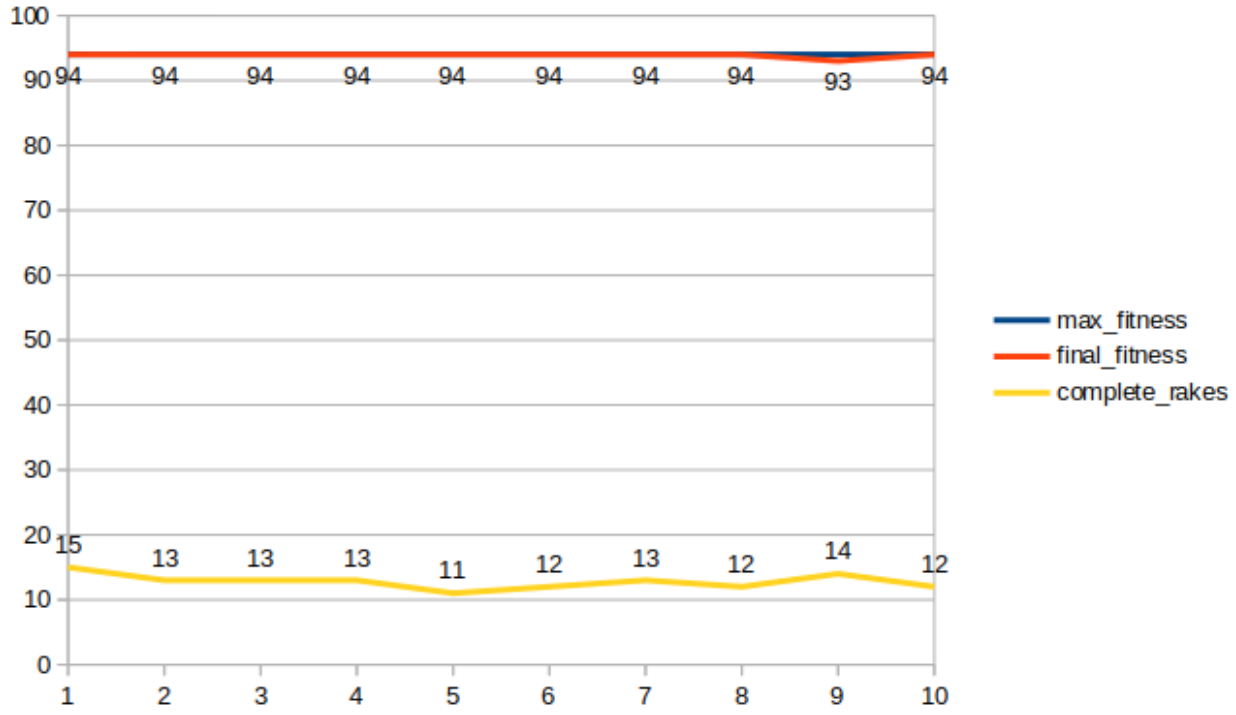
### 4.2.4 Vysoká temperature a vysoký th_extender

- **temperature** = 50
- **th_extender** = 100

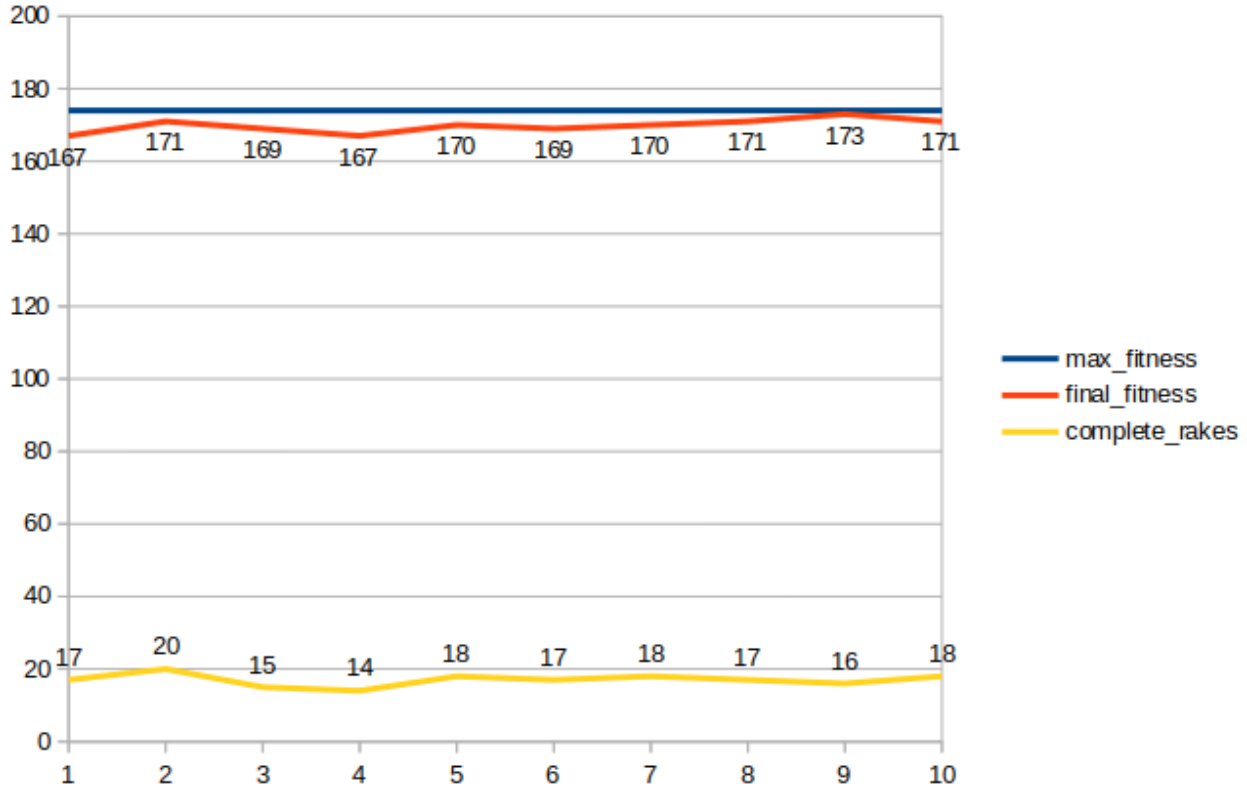
Počet iterácií pre danú kombináciu je vždy rovný **149941**.
Mapa č.1 (rozmer 6x6, počet kameňov 4) (map30.txt):




Mapa č.2 (rozmer 10x10, počet kameňov 6) (map31.txt):




Mapa č.3 (rozmer 14x14, počet kameňov 22) (map32.txt):



Teraz už vieme s istotou povedať, že maximalizácia hyperparametrov naozaj pomáha k nachádzaniu lepších riešení. Je dôležité poznamenať, že je možné, že mapa č.3 **nemá kompletné riešenie**, keďže isté kombinácie kameňov to môžu veľmi ľahko zabezpečiť.
Vylučujúc tento fakt možno povedať, že **algoritmus funguje korektne** a pre rôzne mapy s dostatočným počtom vhodne uložených kameňov nájde riešenie takmer vždy. Takisto je schopný ho vylepšovať aj na úrovni počtu ťahov, nielen v rámci atribútu fitness.

# 5. Možné vylepšenia a záver

Medzi možné vylepšenia patria nasledovné:

- **epochy** - buď ako nový parameter alebo vstavane priamo. Epochami by sme docielili to, že algoritmus by našiel **N riešení** simulovaným žíhaním osobitne a na záver by vybral to najlepšie z daných N možností. Tým by sa jeho úspešnosť ešte o zopár percent zlepšila.

- **generovanie máp**, ktoré zaručene **obsahujú kompletné riešenie** - pridanie algoritmu, ktorý by bol schopný generovať mapy, ktoré majú riešenie, čo by si program overil **paralelným Threadom** v pozadí, ktorý by na danej mape vykonal algoritmus simulovaného žíhania s dostatočne nastavenými hyperparametrami. Ak by záhradu nebolo možné pohrabať s maximálnym fitnessom, algoritmus by pridal kamene na miesta, ktoré nepohrabal, a následne by skúsil vyriešiť daný problém znova.

- **efektívnejšia práca s border_tiles** - konkrétne aby program vedel počas vykonávania legálnych ťahov v poli **paralelne vyhadzovať hraničné políčka** so zoznamu border_tiles, ktoré po vykonaní ťahu už zaručene nebudú korektné. Tak by program nemusel cykliť políčkami, o ktorých sa už predom vie, že nebudú validné. To isté sa dá spraviť v prípade, ak už mních dosiahne max_fitness - t.j. opustiť funkciu okamžite a neprehľadávať ďalšie hraničné políčka.

Myslím si, že moje riešenie je **mimoriadne kvalitné**, keďže obsahuje aj funkcionality, ktoré neboli explicitne zadané, no priamo napomáhajú fungovaniu programu. Algoritmus funguje **spoľahlivo** a správa sa **predvídavo**, čo naznačuje celkovú **korektnosť** mojej implementácie.