FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
SLOVENSKÁ TECHNICKÁ UNIVERZITA
Ilkovičova 2, 842 16 Bratislava 4

2022/2023

Peter Bartoš
Umelá inteligencia
Zadanie 2A

Programovacie prostredie: PyCharm (Python)
Čas cvičení: Streda 13:00 – 14:50

Vypracoval: Peter Bartoš
AIS ID: 116143

Obsah

Riešený problém

Zenova záhradka je plocha vysypaná hrubším pieskom. Obsahuje však aj nepohyblivé väčšie objekty, ako napríklad kamene. Mních má upraviť piesok v záhradke pomocou hrablí tak, že vzniknú pásy ako na nasledujúcom obrázku.

Pásy môžu ísť len vodorovne alebo zvislo. 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. 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, prípade maximálny možný počet políčok.

Zadefinovanie záhrady

Zenova záhradka je zadefinovaná ako pole, kde nuly predstavujú prázdne políčka a veľké písmenko K predstavuje kamene, ktoré mních musí obchádzať:

```
zahradka = [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, K, 0, 0, 0, 0, 0, 0],
            [0, K, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, K, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, K, 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, 0, 0, K, K, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

Zadefinovanie mnícha a jeho génov

Za gény jedinca v tomto riešení sa považuje jeho začiatočná x súradnica a začiatočná y súradnica na záhradke. Ďalej sa za gén považuje jeho preferencia otáčania, ktorá je zadefinovaná ako list s hodnotami reťazcov: hore, dole, vpravo, vľavo. Ak mních narazí na prekážku v tomto poradí hodnôt smeru, tak najprv sa pozrie, že či sa nedá ísť smerom hore, následne dole, za tým vľavo a nakoniec vpravo.

Jedinec je v tomto riešení zadefinovaný ako objekt s rozličnými parametrami, ktoré umožňujú uľahčiť jeho pohyb na poli. Premenné x a y sú jeho momentálne súradnice. Ťahanie predstavuje koľko políčok v danom hrabaní hrabe bez toho, aby vyšiel von zo záhrady. Počítadlo počíta koľkokrát už zo záhrady vyšiel a premiestnil sa na iný okraj. Otočenie ukladá jeho momentálny stav otočenia. Záhrada uchováva pole záhrady. Možný ďalší pohyb kontroluje, že či sa mních ešte vie niekde pohnúť. Skóre uchováva fitness. Zadefinovanie jedinca:

```
class Jedinec:
    def __init__(self, gen_x, gen_y, x, y, tahanie, pocitadlo, otocenie,
poradie_otocenia, zahrada, mozny_dalsi_pohyb,
                 skore):
        self.gen_x = gen_x
        self.gen_y = gen_y
        self.x = x
        self.y = y
        self.tahanie = tahanie
        self.pocitadlo = pocitadlo
        self.otocenie = otocenie
        self.poradie_otocenia = poradie_otocenia
        self.zahrada = zahrada
        self.mozny_dalsi_pohyb = mozny_dalsi_pohyb
        self.skore = skore
```

Logika pohybu mnícha

Pre pohyb jedinca sa uskutoční funkcia badanie_jedinca(jedinec), ktorá v cykle vždy kontroluje, že či existuje ďalší možný pohyb pre jedinca. Ak nie, tak skončí a vypíše informácie k danému jedincovi:

```
def badanie_jedinca(jedinec):
    while jedinec.mozny_dalsi_pohyb is True:
        pohyb(jedinec)
    if jedinec.mozny_dalsi_pohyb is False:
        spocitanie_skore(jedinec)
        print("Gen x: " + str(jedinec.gen_x) + " | Gen y: " +
str(jedinec.gen_y) + " | Skore: " + str(jedinec.skore)
              + "\n")
```

Vo funkcii pohyb(jedinec) sa kontroluje otočenie jedinca, čiže smer hrabania a podľa toho sa volajú funkcie na pohyb tým smerom:

```
def pohyb(jedinec):
    if jedinec.otocenie == "vpravo":
        vpravo(jedinec)
    elif jedinec.otocenie == "vlavo":
        vlavo(jedinec)
    elif jedinec.otocenie == "hore":
        hore(jedinec)
    elif jedinec.otocenie == "dole":
        dole(jedinec)
```

Ďalej už keď sa vyberie funkcia na smer pohybu, tak v nej sa kontroluje, že či sa vie týmto smerom mních pohnúť. Najprv sa kontroluje, že či nie je na okraji poľa a má pred smerom pohybu voľný priestor. Ak hej, tak sa posunie. Ak nie, tak sa otočí. Ak je na kraji poľa a vie sa posunúť týmto smerom už len mimo poľa, tak sa určí náhodná pozícia na okraji poľa, odkiaľ bude začínať svoje ďalšie hrabanie.

```
def dole(jedinec):
    x = jedinec.x
    y = jedinec.y
    if jedinec.otocenie == "dole" and x != 9 and jedinec.zahrada[x + 1][y]
== 0:
        jedinec.x += 1
        jedinec.tahanie += 1
    elif jedinec.otocenie == "dole" and x != 9 and jedinec.zahrada[x +
1][y] != 0:
        urcenie_otocenia(jedinec)
    elif jedinec.otocenie == "dole" and x == 9:
        jedinec.pocitadlo += 1
        nahodna_krajna_pozicia(jedinec, 0)
    if jedinec.zahrada[jedinec.x][jedinec.y] == 0:
        jedinec.zahrada[jedinec.x][jedinec.y] = jedinec.pocitadlo
```

Mních môže počas svojho hrabania naraziť na už predtým hrabané miesto alebo kameň a potrebuje sa otočiť. Takýto prípad ošetruje funkcia urcenie_otocenia(jedinec), ktorá si vytvorí list so smermi. Následne v cykle skúša smery, ktorými by sa mních vedel otočiť a mal voľné miesto na hrabanie daným smerom. Ak sa tým smerom jedinec pohnúť nevie, tak sa do listu so smermi uloží táto možnosť na zapamätanie, že dané otočenie je vylúčené. Ak sa tým smerom pohnúť vie, tak sa zavolá funkcia na otočenie mnícha tým smerom. Keď nastane situácia, že sa list so smermi naplní a nie je na žiadnom okraji poľa, tak to indikuje neschopnosť jedinca sa niekde pohnúť. V tomto prípade sa jedincov ďalší pohyb nastaví na nemožný a funkcia sa vráti:

```
def urcenie_otocenia(jedinec):
    smery = []
    for smer in jedinec.poradie_otocenia:
        if smer == "vpravo" and jedinec.y + 1 <= 11:
```

```
            if jedinec.zahrada[jedinec.x][jedinec.y + 1] == 0:
                otocenie_vpravo(jedinec)
            else:
                smery.append(smer)
        elif smer == "vlavo" and jedinec.y - 1 >= 0:
            if jedinec.zahrada[jedinec.x][jedinec.y - 1] == 0:
                otocenie_vlavo(jedinec)
            else:
                smery.append(smer)
        elif smer == "hore" and jedinec.x - 1 >= 0:
            if jedinec.zahrada[jedinec.x - 1][jedinec.y] == 0:
                otocenie_hore(jedinec)
            else:
                smery.append(smer)
        elif smer == "dole" and jedinec.x + 1 <= 9:
            if jedinec.zahrada[jedinec.x + 1][jedinec.y] == 0:
                otocenie_dole(jedinec)
            else:
                smery.append(smer)
        else:
            smery.append(smer)
    if len(smery) == 4:
        if (jedinec.x == 0 or jedinec.x == 9 or jedinec.y == 0 or jedinec.y
== 11) and jedinec.tahanie > 1:
            jedinec.pocitadlo += 1
            nahodna_krajna_pozicia(jedinec, 0)
        else:
            jedinec.mozny_dalsi_pohyb = False
            return
```

Ukážka funkcie otocenie_vpravo(jedinec) pri prípade možného otočenia vpravo:

```
def otocenie_vpravo(jedinec):
    if jedinec.zahrada[jedinec.x][jedinec.y + 1] == 0:
        jedinec.otocenie = "vpravo"
```

V určení otočenia jedinca sa v tomto riešení využíva funkcia na vybratie náhodnej krajnej pozície jedinca, keď sa nemá kam pohnúť a je na niektorom z okrajov poľa. Tu sa náhode určí strana, z ktorej bude jedinec vchádzať do záhradky. Ak je políčko voľné, tak sa aj mních natočí smerom, ktorým bude hrabať. Ak na tejto pozícií sa zistí, že nie je voľné políčko, tak sa funkcia zavolá znova. Funkcia má obmedzenú rekurziu pre prípad zacyklenia, kde sa ďalší možný pohyb nastaví na nemožný a funkcia skončí:

```
def nahodna_krajna_pozicia(jedinec, rekurzia):
    if rekurzia > 800:
        jedinec.mozny_dalsi_pohyb = False
        return
    strana = random.randint(1, 4)
    if strana == 1:  # hore
        jedinec.x = 0
        jedinec.y = random.randint(0, 11)
    elif strana == 2:  # dole
        jedinec.x = 9
        jedinec.y = random.randint(0, 11)
```

```
    elif strana == 3:  # vpravo
        jedinec.x = random.randint(0, 9)
        jedinec.y = 11
    elif strana == 4:  # vlavo
        jedinec.x = random.randint(0, 9)
        jedinec.y = 0

    if jedinec.zahrada[jedinec.x][jedinec.y] != 0:
        nahodna_krajna_pozicia(jedinec, rekurzia + 1)
    else:
        if strana == 1:
            jedinec.otocenie = "dole"
        elif strana == 2:
            jedinec.otocenie = "hore"
        elif strana == 3:
            jedinec.otocenie = "vlavo"
        elif strana == 4:
            jedinec.otocenie = "vpravo"
        jedinec.tahanie = 1
```

Evolučný algoritmus

V tomto riešení sa najprv vygeneruje prvá generácia s náhodne danými génmi. Počet vygenerovaných jedincov závisí od rozmerov záhrady a počtu kameňov:

```
generovanie_mnichov(len(zahradka) + len(zahradka[0]) + pocet_kamenov)
```

Zavolá sa funkcia generovanie_mnichov(pocet), kde v cykle sa vygeneruje dané množstvo mníchov. Vytvorí sa objekt mnícha, kde následne sa určujú gény náhodne, vykreslí sa do záhradky jeho začiatočná pozícia a začne sa hrabanie mnícha (opísané v kapitole „logika pohybu mnícha“). Ďalej sa do listu so skóre uložíme skóre daného mnícha. Do listu s dosial najlepšími mníchmi si uložíme daného jedinca, ak bol dosiaľ najlepší. Po skončení cyklu sa zoradí list so skóre od najväčšieho po najmenšie a štyria najlepší mnísi sa uložia do listu s najlepšími mníchmi z danej generácie (elitarizmus). Ďalších osemnásť najlepších mníchov sa uloží do listu s mníchmi na kríženie. Zvyšní mnísi s najhorším skóre sa uložia do listu s mníchmi na mutáciu:

```
def generovanie_mnichov(pocet):
    skore = []
    for i in range(pocet):
        mnich = Jedinec(-1, -1, -1, -1, 1, 1, "", [],
copy.deepcopy(zahradka), True, 0)
        urcenie_nahodnych_genov(mnich)
        mnich.zahrada[mnich.x][mnich.y] = mnich.pocitadlo
        badanie_jedinca(mnich)
        mnichovia.append(mnich)
        skore.append(mnich.skore)
        if len(dosial_najlepsi) == 0:
            dosial_najlepsi.append(mnich)
        elif mnich.skore > dosial_najlepsi[0].skore and
```

```
len(dosial_najlepsi) != 0:
            dosial_najlepsi.append(mnich)
            dosial_najlepsi.pop(0)
    print("-------------------0-------------------")
    skore.sort(reverse=True)
    najlepsi_po_generaciach.append(skore[0])
    for i in range(len(skore)):
        for jedinec in mnichovia:
            if jedinec.skore == skore[0]:
                elitarizmus.append(jedinec)
                skore.pop(0)
                break
        if len(elitarizmus) == 4:
            break

    for i in range(len(skore)):  # Mnichovia suci na krizenie s dobrymi
genmi
        for jedinec in mnichovia:
            if jedinec.skore == skore[0]:
                mnichovia_na_krizenie.append(jedinec)
                skore.pop(0)
                break
        if len(mnichovia_na_krizenie) == 18:
            break

    skore.sort(reverse=False)
    for i in range(len(skore)):  # Mnichovia suci na uplnu mutaciu, kedze
maju zle geny
        for jedinec in mnichovia:
            if jedinec.skore == skore[0]:
                mnichovia_na_mutaciu.append(jedinec)
                skore.pop(0)
                break
        if len(mnichovia_na_mutaciu) == 6:
            break
```

Ďalej sa v cykle budú krížiť mnísi a vytvárať nové generácie. Počet generácii je určený podľa vstupu:

```
for k in range(pocet_generacii):
    mnichovia = krizenie(k)      # tvorenie dalsich generacii
```

Funkcia krizenie(generacia) je veľmi podobná predchádzajúcej funkcii na generovanie mníchov. Líši sa iba v jej prvej časti. Pripraví sa znova list so skóre ale aj list s novými mníchmi. Do nových mníchov sa pridajú štyria mnísi z listu elitarizmu. Následne sa zavolá funkcia na kríženie mníchov a za tým sa vyčistia listy s mníchmi na kríženie, mutáciu a elitarizmu. Následne sa znova naplní list so skóre novými jedincami:

```
def krizenie(generacia):
    skore = []
    novy_mnichovia = []
    najlepsi_po_generaciach.append(0)
    for jedinec in elitarizmus:
        novy_mnichovia.append(jedinec)
```

```
    krizenie_mnichov(novy_mnichovia, generacia)
    mnichovia_na_krizenie.clear()
    mnichovia_na_mutaciu.clear()
    elitarizmus.clear()
    if najdene_riesenie[0] is True:
        return
    for jedinec in novy_mnichovia:
        skore.append(jedinec.skore)

  print("-------------------" + str(generacia + 1) + "-------------------")
```

Po zavolaní funkcie krizenie_mnichov(novy_mnichovia, generacia) sa v cykle berú dvaja mnísi z listu mníchov na kríženie a následne krížia ich vlastnosti. V párnej iterácii sa zoberie jeden mních a v nepárnej druhý mních. Popri braní druhého mnícha začne proces kríženia. Uložia sa do jedného poľa vlastnosti prvého jedinca a do druhého poľa vlastnosti druhého jedinca. Zavolá sa funkcia kríženia preferencií, kde sa jednotlivé poradia otočenia týchto dvoch jedincov skombinujú a nastavia sa na vybraných mníchov (ide o "recykláciu" objektu). Následne sa zavolá funkcia na skríženie prvých dvoch génov. Po tomto úkone sa rozhodne, že či gény jedincov budú mutovať (desať percentná šanca) a začne sa hrabanie jedincov. Ak je jeden z nich doposiaľ najlepší zo svojej generácie, tak sa uloží do listu najlepšieho zo svojej generácie. Ďalej ak jedno z ich skóre sa zhoduje s maximálnym možným skóre, tak úspešne jeden z nich odhrabal celú záhradu bez vynechania nejakého políčka. Ak je jeden z týchto jedincov doposiaľ najlepší z doposiaľ všetkých generácií, tak sa uloží do listu s doposiaľ najlepším jedincom (v kóde nižšie je kríženie uvedené len pre jedného z dvoch jedincov, lebo presne to isté prebehne aj pre druhého).

```
def krizenie_mnichov(novy_mnichovia, generacia):
    xmnich = None
    for index in range(len(mnichovia_na_krizenie)):
        if index % 2 == 0:
            xmnich = mnichovia_na_krizenie[index]
            continue
        else:
            ymnich = mnichovia_na_krizenie[index]
            stary_x = [xmnich.gen_x, xmnich.gen_y, xmnich.poradie_otocenia]
            stary_y = [ymnich.gen_x, ymnich.gen_y, ymnich.poradie_otocenia]
            x_nove_preferencie = krizenie_preferencii(stary_x[2],
stary_y[2])
            y_nove_preferencie = krizenie_preferencii(stary_y[2],
stary_x[2])
            xmnich.poradie_otocenia = x_nove_preferencie
            ymnich.poradie_otocenia = y_nove_preferencie
            krizenie_prvych_dvoch_genov(stary_x[0], stary_y[1], xmnich)
            if random.randrange(1, 10) == 5:
                mutuj_gen = random.randrange(1, 4)
                if mutuj_gen != 4:
                    mutacia_prvych_dvoch_genov(xmnich)
                else:
                    xmnich.poradie_otocenia = nahodne_otacania()
```

```
urcenie_otocenia(xmnich)
            xmnich.zahrada[xmnich.x][xmnich.y] = xmnich.pocitadlo
            badanie_jedinca(xmnich)
            if xmnich.skore > najlepsi_po_generaciach[generacia + 1]:
                najlepsi_po_generaciach[generacia + 1] = xmnich.skore
            if xmnich.skore == max_skore:
                print("Najdene riesenie…")
                najdene_riesenie.pop(0)
                najdene_riesenie.append(True)
                dosial_najlepsi.clear()
                dosial_najlepsi.append(xmnich)
                vykresli_platno(xmnich.zahrada, xmnich)
                return
            elif xmnich.skore > dosial_najlepsi[0].skore and
len(dosial_najlepsi) != 0:
                dosial_najlepsi.append(copy.deepcopy(xmnich))
                dosial_najlepsi.pop(0)
            novy_mnichovia.append(xmnich)
                mutacia(novy_mnichovia, generacia)
```

Kríženie preferencií prebieha vo funkcii krizenie_preferencii(x_otocenia_povodne, y_otocenia_povodne), kde sa nakopírujú do premenných obidva listy preferencií otočenia zo vstupu a vytvorí sa prázdny list pre nové poradie otočenia pre kríženého jedinca. V cykle sa hodí mincou, že z ktorého génu sa zoberie do listu otočenie a dané otočenie sa z obidvoch nakopírovaných listov vyhodí. Toto sa opakuje až kým nakopírované listy sú prázdne a list preferencií pre kríženého jedinca je plný:

```
def krizenie_preferencii(x_otocenia_povodne, y_otocenia_povodne):
    x_otocenia = copy.deepcopy(x_otocenia_povodne)
    y_otocenia = copy.deepcopy(y_otocenia_povodne)
    poradie_otocenia = []
    pocet_cyklov = len(x_otocenia)
    for i in range(pocet_cyklov):
        hod_mincou = random.randint(0, 1)
        if hod_mincou == 0:
            poradie_otocenia.append(x_otocenia[0])
            otocenie = x_otocenia[0]
            x_otocenia.pop(x_otocenia.index(otocenie))
            y_otocenia.pop(y_otocenia.index(otocenie))
        elif hod_mincou == 1:
            poradie_otocenia.append(y_otocenia[0])
            otocenie = y_otocenia[0]
            x_otocenia.pop(x_otocenia.index(otocenie))
            y_otocenia.pop(y_otocenia.index(otocenie))
    return poradie_otocenia
```

Kríženie prvých dvoch génov vyzerá v tomto riešení tak, že na vstupe funkcia krizenie_prvych_dvoch_genov_primarny_x(gen1, gen2) dostane dva gény, kde ak prvý gén reprezentuje okrajové súradnice záhradky a druhý gén reprezentuje tiež okrajové súradnice záhradky, tak sa pre druhý gén zvolí náhodne číslo medzi 1 a 10, pretože tieto čísla nereprezentujú okraj záhradky. Ak je situácia opačná, že

prvý gén nereprezentuje okraj záhradky, tak ak druhý gén tiež nereprezentuje okraj záhradky, tak sa vyberie číslo 0 alebo 11 pre druhý gén, ktoré reprezentujú okraj záhradky.

```
def krizenie_prvych_dvoch_genov_primarny_x(gen1, gen2):
    if gen1 == 0 or gen1 == 9:
        if gen2 == 0 or gen2 == 11:
            return random.randint(1, 10)
        else:
            return gen2
    else:
        if gen2 == 0 or gen2 == 11:
            return gen2
        else:
            return nahodne_cislo_zo_vstupu(0, 11)
```

Po skončení kríženia nasleduje v algoritme mutácia najhorších jedincov. Vo funkcii mutacia(novy_mnichovia, generacia) sa jednoducho v cykle prechádzajú jedince z listu mníchov na mutáciu a jeden po jednom sa im zadefinujú náhodné gény a následne sa ukladajú do listu nových mníchov:

```
def mutacia(novy_mnichovia, generacia):
    for jedinec in mnichovia_na_mutaciu:
        jedinec.geny(-1, -1, -1, -1, 1, 1, "", [],…)
        urcenie_nahodnych_genov(jedinec)
        jedinec.zahrada[jedinec.x][jedinec.y] = jedinec.pocitadlo
        badanie_jedinca(jedinec)
        if jedinec.skore == max_skore:  # jedinec presiel celu zahradu
        print("Najdene riesenie…")
        novy_mnichovia.append(jedinec)
```

Možnosti nastavenia parametrov

V tomto riešení je možné meniť umiestnenie kameňov v záhradke a záhradku zväčšovať lebo zmenšovať podľa potreby, ale záhradka musí byť veľká aspoň 3x3. Ďalej sa dá určiť, že koľko jedincov z jednotlivej generácie bude mutovať, budú pozostávať do ďalšej generácie (elitarizmus) a koľko sa bude krížiť (počet jedincov na kríženie musí byť párne číslo):

```
pocet_na_mutaciu = 6  # moznost nastavenia parametra
pocet_na_krizenie = 18  # moznost nastavenia parametra
pocet_elitnych = 4  # moznost nastavenia parametra
```

Po spustení programu pre naštartovanie generovania výsledku pre problém je treba napísať príkaz “generate“ alebo stačí aj “g“ a program sa spýta pre počet generácií. Pre “neobmedzený“ počet generácií zadajte pre počet generácií -1. Po zadaní počtu generácií sa začne riešenie problému. Po nájdení riešenia program na plátno vykreslí výslednú záhradu a po zrušení plátna sa znova spýta na príkaz a môže sa generovať znova. Pre zadanie parametrov treba zadať príkaz “params“

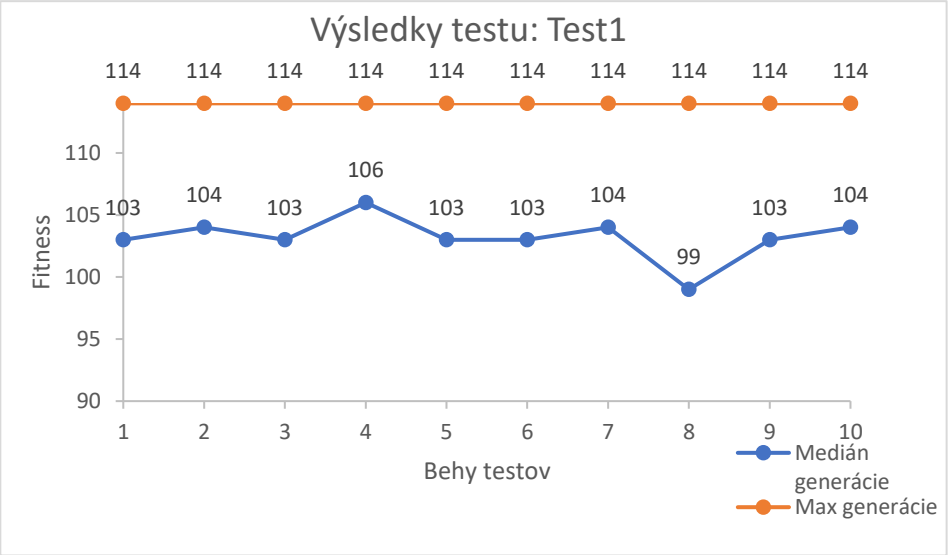
alebo “p“. Dá sa aj krokovať mnícha pre preskúšanie funkcionality príkazom “rake“ alebo “r“. Príkaz “vypis“ alebo “v“ vypíše mnícha s najlepšie ohrabanou záhradou z posledného generovania.

Testovanie

Riešenie bolo otestované na troch rôznych druhoch testovania.

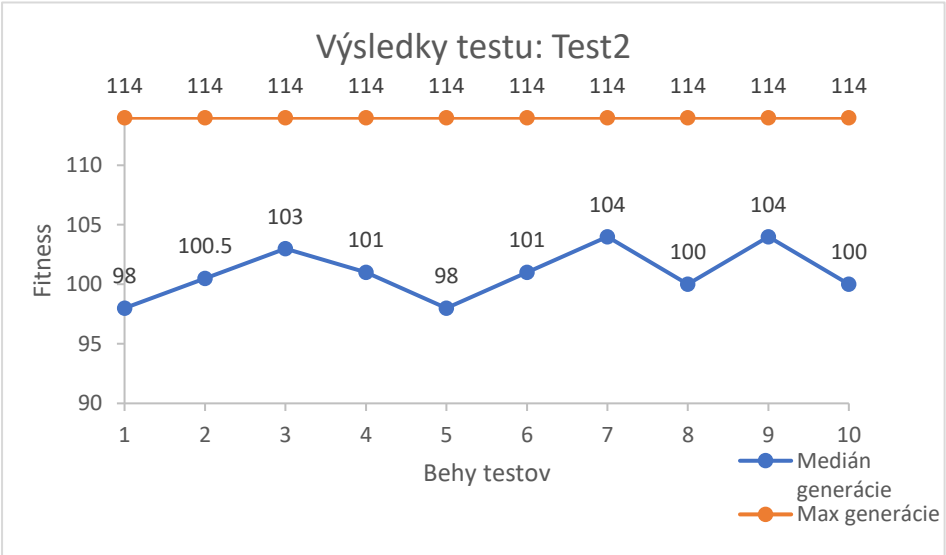
Najprv bolo otestované riešenie bez elitarizmu, čiže sa mnísi mohli len krížiť a mutovať. V každej generácií bolo 28 mníchov. 24 z nich sa krížilo a 4 najhorší mutovali. Priemerný počet generácií bol 46 a priemer mediánov bol 103 po zaokrúhlení. Výsledky boli nasledovné:

Testy	Počet generácii	Typ vzniku výsledného mnícha
1	49	mutácia
2	43	mutácia
3	25	kríženie
4	2	kríženie
5	102	kríženie
6	65	kríženie
7	6	kríženie
8	11	mutácia
9	143	kríženie
10	11	kríženie



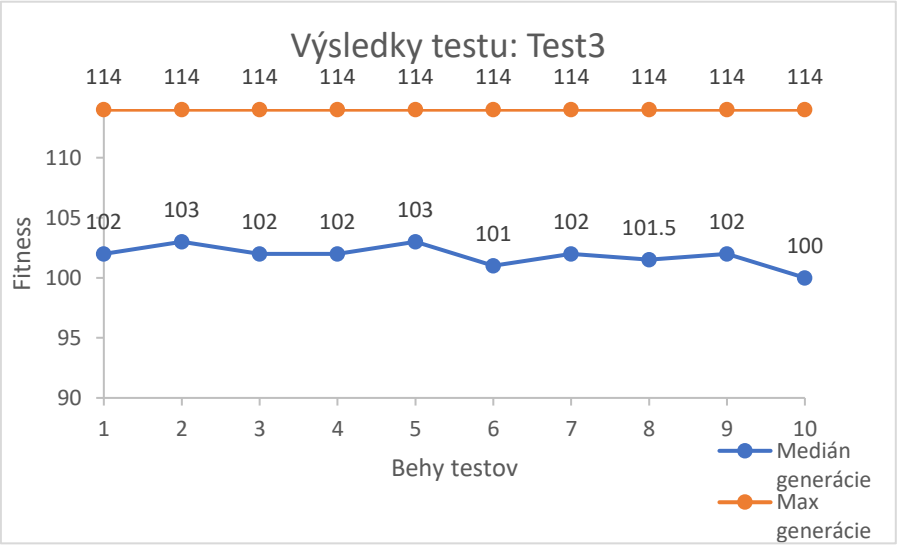
Druhé otestovanie bolo riešené s elitarizmom bez mutácií, kde 10 najlepších mníchov z 28 prežívalo do ďalšej generácie a zvyšných 20 sa krížilo. Priemerný počet generácií bol 159 a priemer mediánov bol 101 po zaokrúhlení. Výsledky boli nasledovné:

Testy	Počet generácii	Typ vzniku výsledného mnícha
1	18	kríženie
2	57	kríženie
3	36	kríženie
4	115	kríženie
5	560	kríženie
6	499	kríženie
7	20	kríženie
8	67	kríženie
9	23	kríženie
10	193	kríženie



Tretie otestovanie bolo riešené so všetkým. Sedem mníchov prežívalo do ďalšej generácie, traja najhorší mutovali a osemnásť sa krížilo. Dokopy 28 v každej generácií. Priemerný počet generácií bol 136 a priemer mediánov bol 102 po zaokrúhlení. Výsledky boli nasledovné:

Testy	Počet generácii	Typ vzniku výsledného mnícha
1	215	kríženie
2	49	kríženie
3	235	kríženie
4	68	mutácia
5	70	kríženie
6	6	kríženie
7	90	kríženie
8	103	kríženie
9	369	kríženie
10	152	kríženie



Zhrnutie

Riešenie daného problému z rôznymi parametrami bolo poriadne otestované.

Prvé otestovanie zamerané viacej na mutácie vyzerá byť najefektívnejšie, avšak pri takom vysokom počte mutácií hrá viacej rolu pravdepodobnosť než evolúcia a preto sa dá klasifikovať ako slabšie v súlade so zadaním.

Druhé otestovanie bolo vysoko zamerané na elitarizmus a kríženie, kde mutácia mníchov s najhorším fitness bola zakázaná a prebiehala iba pri krížení s nízkou pravdepodobnosťou. Výsledky tohto testovania sa nezhoršilo v rámci výsledkov, ale prebiehalo dlhšie a priemerný počet generácií sa strojnásobil kvôli reálnej nutnosti sa krížiť pre nájdenie výsledku. V súlade so zadaním a evolučným algoritmom ako samým je toto otestovanie najlepšie.

Tretie otestovanie zohľadňovalo všetko, čiže elitarizmus, kríženie a aj mutovanie. Poskytlo lepšie výsledky fitness ako otestovanie bez mutácií a aj o niečo lepší priemerný počet generácií, avšak náhoda tu tiež hrá značnú rolu kvôli mutácií mníchov s najhorším fitness.

Otestovanie zamerané iba na elitarizmus a kríženie iba s čiastočným mutovaním pri krížení sa osvedčilo ako najlepšie riešenie tohto problému a svojim priebehom a vlastnosťami najviac spadá pod evolučné algoritmy.