Slovenská Technická Univerzita v Bratislave

Fakulta informatiky a informačných technológií

Zenová záhrada

Zadanie 3a – Evolučný algoritmus

Predmet: Umelá inteligencia

Obdobie: Letný semester 2019/2020

Cvičiaci: Ing. Ivan Kapustík

Študent: Bc. František Gič

Obsah

Zadanie

Zenová záhradka 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, prípade maximálny možný počet políčok. Výstupom je pokrytie danej záhrady prechodmi mnícha.

Uvedenú úlohu riešte pomocou evolučného algoritmu. Maximálny počet génov nesmie presiahnuť polovicu obvodu záhrady plus počet kameňov, v našom príklade podľa prvého obrázku 12+10+6=28. Fitnes je určená počtom pohrabaných políčok. Výstupom je matica, znázorňujúca cesty mnícha. Je potrebné, aby program zvládal aspoň záhradku podľa prvého obrázku, ale vstupom môže byť v princípe ľubovoľná mapa.

Implementácia

K vypracovaniu tohto zadania boli evolučný algoritmus s krížením a mutáciou. Programoval som v jazyku *JavaScript* , lokálnom environmente - *Node.js* s použitím supersetu *Typescript* pre striktné otypovanie.

Inštalácia

Prerekvizity:

- Node.js (https://www.nodejs.org/)
- Node package manager (https://www.npmjs.com/)

V root adresári spustite nasledovné príkazy:

```
npm install
```

A následne, pre každú transpiláciu typescriptového kódu na javascript a spustenie kódu v node.js:

```
npm run dev
```

Reprezentácia údajov

Moje riešenie pozostáva z dvoch údajových typov – tried, `Monk` a `Garden`.

`Garden`, ako už anglický preklad napovedá je reprezentácia stavu riešenia.

Obsahuje celočíselné informácie o veľkosti grafu `x` a `y` – záhrady, reprezentovanej dvojrozmerným poľom `char` (v JavaScripte `string`).

Taktiež, obsahuje počet kameňov (prekážok), pre správne vypočítanie maximálnej veľkosti genómu.

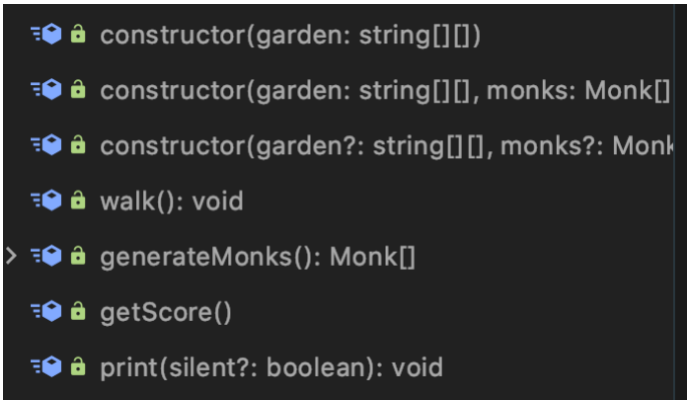
Samotné gény sú reprezentované ako pole typu `Monk`.

```
export default class Garden {
    public x: number;
    public y: number;
    public rocks: number;
    public score: number = 0;
    public monks: Monk[] = [];
    public garden: string[][];
```

Obrázok 1: Vlastnosti triedy Garden

Medzi metódy triedy `Garden` *patria nasledovné:*

- `walk()` – metóda vezme všetky gény daného stavu záhrady a aplikuje ich na graf záhrady (zapíše jednotlivé číselné reprezentácie génov a vygeneruje dvojrozmerné pole záhrady, v ktorej tieto hodnoty sú už vpísané)
- `generateMonks()` – metóda, tvoriaca všetky možné vstupy do záhrady, vracajúca náhodne poprehadzované pole maximálneho počtu génov (v zadaní udané ako polovica obvodu + počet kameňov)
- `getScore()` – metóda vracajúca fitness hodnotu daného stavu (počet pohrabaných políčok vydelené počtom polí) od 0 po 1
- `print()` - metóda vypisujúca farebne aktuálny stav záhradky do konzoly

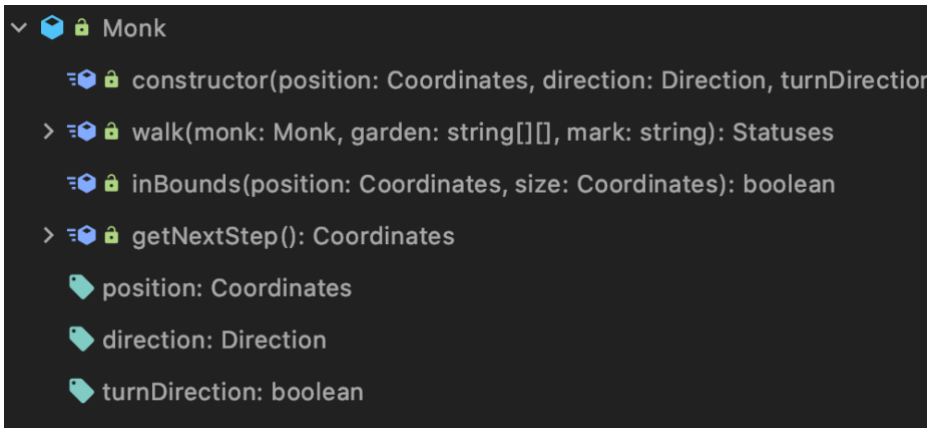


Obrázok 2: Metódy triedy Garden

Trieda `Monk` reprezentuje v mojom riešení samotný gén, chromozóm. Počet génov v jedincovi je teda x + y + počet kameňov.

Jednotlivý gén uchováva v sebe **nasledujúce informácie:**

- `position` – pozícia, reprezentovaná typom `Coordinates` – x a y
- `direction` – numerická hodnota reprezentujúca smer ktorým daný mních (monk) smeruje (0 – Hore, 1 – Vpravo, 2 – Dolu, 3 – Vľavo)
- `turnDirection` – atribút typu `boolean`, reprezentujúci voľbu smeru v prípade narazenia do prekážky



Obrázok 3: Hierarchia triedy Monk

Medzi **metódy** patria:

- `walk()` – rekurzívna metóda volajúca sa v triede `Garden`, snažiaca sa prejsť z pozície v danom smere, s daným rozhodovaním ku okraju mapy. Vracia statusy – `TURN`, `SUCCESS` a `FAIL`. V prípade že vráti `SUCCESS`, zaznačí sa dané

políčko do mapky. V prípade že narazí, vráti stacku `TURN`, a vynorí sa z rekurzie o jeden stupeň. V tomto prípade sa skúsi otočiť na základe atribútu `turnDirection`. Ak sa ani v tomto smere nedokáže otočiť a nedokáže sa tak vrátiť ku okraju záhrady, vráti status `FAIL` a tento gén sa teda „neaktivuje“ – existuje v genotype, ale neprejaví sa na danej záhradke.

- `inBounds()` – metóda vracajúca pravdivostnú hodnotu, či sa dané koordináty nachádzajú v záhradke. Táto metóda sa hlavne používa na zistenie, či už daný Monk (mníchova prechádzka) skončila úspechom – či gén je aktivovaný (vráti sa na políčko mimo záhradky)

- `getNextStep()` – metóda vracajúca koordináty nasledujúceho kroku na základe aktuálnych koordinátov a smeru, využívaná v rekurzívnej metóde `walk()` pre prechod záhradkou.

Prechádzka mnícha

Pre každý z génov (monkov) záhrady je zavolaná nasledujúca metóda triedy `Monk`. Daná funkcia rekurzívne prezerá aktuálne políčko mnícha. V prípade že je mních na koordinátoch mimo mapu, znamená to, že úspešne opustil záhradku.

V prípade, že políčko nie je prázdne, vynorí sa z rekurzie so statusom `TURN`. Inak, pokračuje ďalej v rekurzií a sleduje návratová hodnotu. V prípade že je rekurzia úspešná, zaznačí na dané políčko svoju značku (zvyčajne poradie prechádzky).

V prípade, ak návratová hodnota je `TURN`, zmení svoj smer na základe aktuálneho smeru a rozhodovacieho faktoru `turnDirection` (atribút kópie mnícha – nikdy neupravujeme samotného mnícha).

Ak sa rekurzia vrátila s hodnotou FAIL, znamená to že daný gén zlyhal a vo výslednej záhradke nebude „aktivovaný“ – prejavený. V genetike a reálnom živote je toto nazvané ako recesívny gén.

```
walk(monk: Monk, garden: string[][], mark: string): Statuses {
    // Whether the monk could exit the garden
    if (
        !monk.inBounds(monk.position, {
            x: garden.length,
            y: garden[0].length,
        })
    )
        return Statuses.SUCCESS;

    // If hit rock or another pathway
    if (garden[monk.position.x][monk.position.y] !== '0') {
        return Statuses.TURN;
    }
```

```
for (let i = 0; i < 2; i++) {
    // Nest into next step
    const result: Statuses = monk.walk(
        new Monk(monk.getNextStep(), monk.direction, monk.turnDirection),
        garden,
        mark
    );

    // Mark the current pathway
    if (result === Statuses.SUCCESS) {
        garden[monk.position.x][monk.position.y] = mark;
        return Statuses.SUCCESS;
    }
    // Toggle direction
    else if (result === Statuses.TURN) {
        if (monk.direction === Direction.UP || monk.direction === Direction.DOWN) {
            monk.direction = monk.turnDirection ? Direction.LEFT : Direction.RIGHT;
        } else {
            monk.direction = monk.turnDirection ? Direction.UP : Direction.DOWN;
        }
    }
    // Cannot mark the pathway, this gene failed
    else return Statuses.FAIL;
}
```

Obrázok 4: Metóda prechádzky mnícha (génu) cez záhradku

Algoritmus

Genetický algoritmus algoritmus v prvom kroku vygeneruje počiatočnú populáciu. V našom prípade sa generuje `startPopulation` jedincov – inštancí triedy `Garden`.
Tieto inštancie majú vo vstupe pole záhradky z externého súboru – parameter input. Následne, v konštruktori triedy `Garden` sa nastaví samotná reprezentácia záhrady, vygenerujú sa náhodní mníchovia (gény) pomocou už vyššie spomenutej metódy `generateMonks()` , vygeneruje sa graf záhrady a vypočíta sa hodnota *fitness* daného stavu záhrady.
Poznámka: `generateMonks()` *generuje* mníchov po celom obvode záhradky.

```
// Generate first starting population
for (let i = 0; i < startPopulation; i++) {
    population.push(new Garden(input));
}
```

```
constructor(garden: string[][]);
constructor(garden: string[][], monks: Monk[]);
constructor(garden?: string[][], monks?: Monk[]) {
    // Deep copy of array
    this.garden = cloneDeep<string[][]>(garden!);

    // Set dimensions
    this.x = this.garden.length;
    this.y = this.garden[0].length;

    // Count rocks
    this.rocks = 0;
    for (let i = 0; i < this.garden.length; i++) {
        for (let j = 0; j < this.garden[0].length; j++) {
            if (this.garden[i][j] == 'K') this.rocks++;
        }
    }

    // Generate the pathways
    this.monks = monks ?? this.generateMonks();
    this.walk();
    this.score = this.getScore();
}
```

Obrázok 5: Generácia počiatočnej populácie

Nasledovne, pokračuje while cyklom kým sa nenaplní jedna z podmienok – buď skončí algoritmus po kroku, kde sa skóre (fitness) najlepšieho z generácie rovná 1 – čiže je nájdené optimálne riešenie, alebo sa vykoná maximálny počet krát (maximálny počet generácií).

V tomto cykle vytvárame novú generáciu. Ako prvý krok si ponecháme najlepšieho jedinca z predchádzajúcej generácie, a vložíme ho do novej generácie.
Následne prechádzame vnútorný cyklus, kde prechádzame zvyšný počet jedincov v generácií. Náhodným výberom alebo algoritmom turnaja vyberieme dvoch rodičov pre každého jedinca a ideme krížiť.

Kríženie

Postup pri krížení je úplne náhodný a pozostáva zo 4 variantov z ktorých sa vykoná iba jeden.

- Prvý variant je výber monkov (génov) rodiča náhodou – pri každom z génov sa hodí kockou, a rozhodne, či daný gén bude od rodiča 1 alebo rodiča 2
- Druhý variant je polovica génov (konsekutívne) od prvého rodiča, a druhá polovica od druhého rodiča
- Tretím variantom je výber génov na striedačku – rodič 1 a rodič 2
- Najmenej pravdepodobným je kópia všetkých génov od rodiča 1

Mutácia

Po krížení nastáva mutácia. Mutácia je dôležitým prvkom genetických algoritmov – dostáva generáciu z lokálneho maxima.
V mojom riešení mutujem 5timi rôznymi spôsobmi s určitou pravdepodobnosťou mutácie pre každý gén.
Varianty mutácie:

- *Mutácia nahradenia génu novým génom* – vytvorí sa náhodný gén a nahradí sa ním pôvodný gén

- *Výmena rozhodovacieho atribútu* – génu sa zmení informácia o rozhodovaní v prípade narazenia, gén sa rozhodne opačným spôsobom
- *Posunutie súradnice štartovania génu* – génu sa pripočíta číslo 1 k súradnici štartovania (podľa pozície štartovania) a vydelí sa so zvyškom veľkosti záhradky v danej osi
- *Vynásobenie súradnice štartovania génu* – génu a jeho súradnica sa vynásobí dvomi, vydelí zvyškom osi záhradky
- *Prehodenie génov* – gény sa preusporiadajú, vznikne tak nová postupnosť, niektoré gény sa tak stanú recesívne a iné dominantné

Nová generácia

Po ukončení kríženia a mutácie sa tak nový jedinec s novou (možno aj starou) sadou génov (mníchov) priradí do novej generácie.
Po vytvorení novej generácie sa stará generácia nahradí novou a kolobeh sa opakuje.

Testovanie

Bohužiaľ, moje riešenie nebolo možné vygenerovať z nejakého dôvodu správnu postupnosť, najbližšie som sa dostal viackrát, k hranici **~0.9 fitness**.

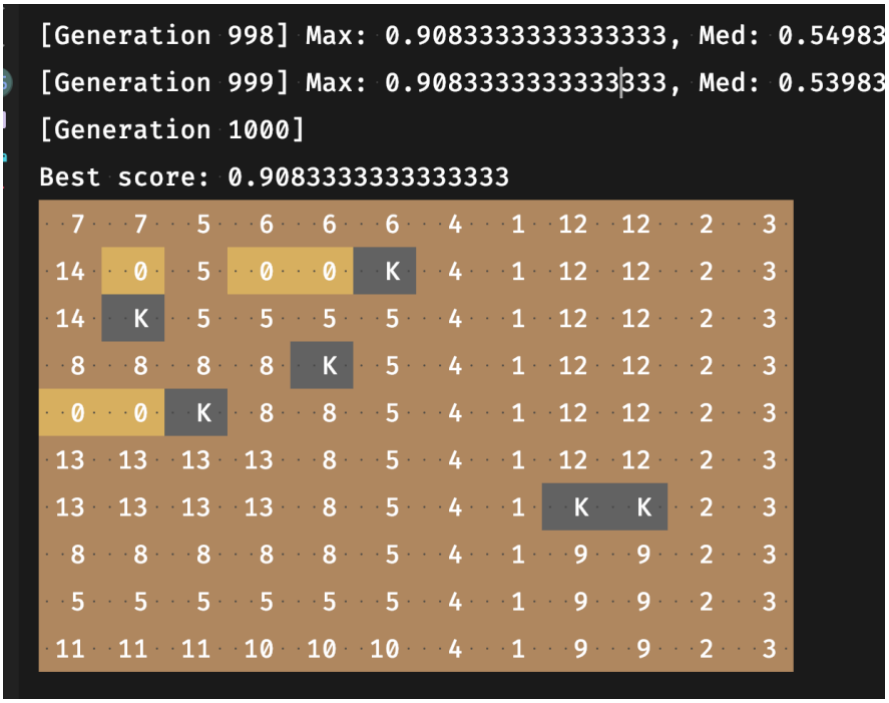
Moje riešenie je variabilné, je možné špecifikovať vlastný graf záhrady zo súboru, veľkosť populácie aj maximálny počet opakovaní (populácií).

Riešenie obsahuje zaujímavý výpis do konzole, ktorý je prehľadný.

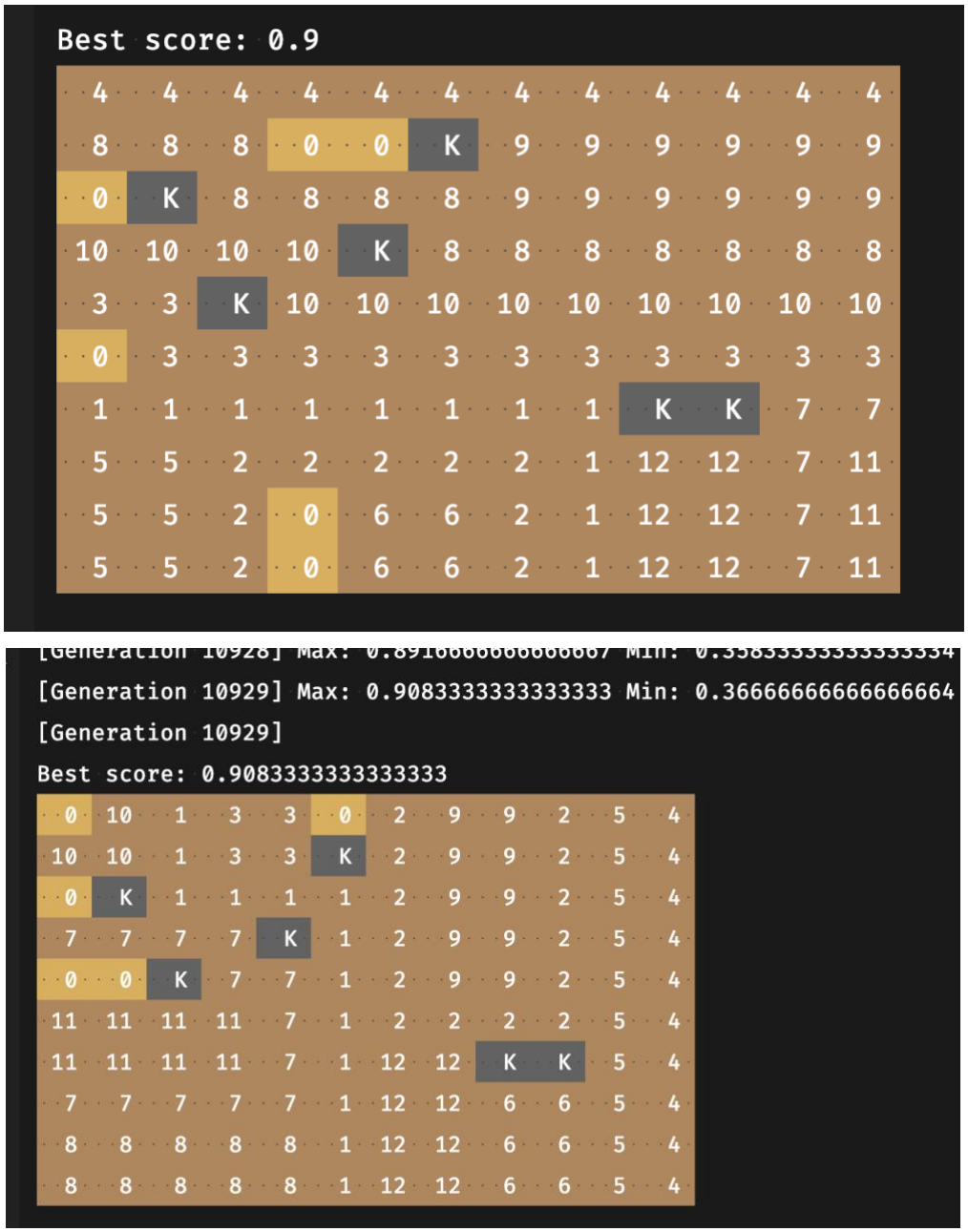
Po zapnutí `DEBUG` módu v riešení je možné vidieť jednotlivé generácie, skóre najlepšieho jedinca, aj skóre najhoršieho jedinca.

Samozrejme, je možnosť upraviť šancu pre mutáciu, a to nastavením parametra `mutationChance` na hodnotu medzi 0 a 1 vrátane.

Najviac sa mi osvedčilo použitie *veľkosti populácie **50*** v kombinácií s *maximálnymi generáciami* **1000**, niekedy **10 000**. Pár krát som použil nekonečný cyklus so zastavením nad 0.9.



Obrázok 6: Populácia 50, 1000 generácií



Obrázok 7: Nekonečný cyklus so zastavením nad 0.9

Zhodnotenie

Toto zadanie ma stálo veľa síl a zopár nervových zrútení. Strávil som nad ním asi 20 hodín v čistom, a viac krát som prerábal logiku samotných prechádzok mnícha. V prvotnom riešení som mal generovanie génov úplne náhodné, ktoré som neskôr zmenil na generovanie z každej pozície jedno, len prehodené.

Následne som mal problémy s odbáčaním mnícha, pôvodne som mal iba polaritu, vertikálnu a horizontálnu.. v prípade horizontálnej, išiel z prava, y súradnica sa znižovala, v prípade vertikálnej sa zvyšovala x súradnica, čiže išiel dolu. To som z dôvodu nízkej úspešnosti prerobil na systém vyššie spomenutých smerov (UP, RIGHT, DOWN, LEFT) s možnosťou `turnDirection` v prípade narazenia.

To mi však stále nepomohlo, a tak som začal generovať mníchov po každej strane záhradky, z ktorých som vzal náhodných maximálny počet génov.

Zopár bugov bolo taktiež skrytých v probléme s najlepším jedincom. Pôvodne som v každej generácií pôvodnú zoradil a vybral posledného, najlepšieho jedinca. Bohužiaľ, sortovanie som časom vyhodil, a tým pádom som si do novej generácie hodil úplne náhodného jedinca, čom mi vyústilo ku kolísaniu maximálnej hodnoty každou generáciou.

Ďalší programátorsky chrobák sa skrýval v posunutom riadku, a z nejakého dôvodu som nasledujúcu generáciu vždy vytváral z x kópií najlepšieho jedinca ☺

Vývoj fitness funkcie generácií z príkladu na obrázku 6: