VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Kartézské Genetické Programování v Evolučním umění

Cartesian Genetic Programming in Evolutionary Art

2015 Marek Jelínek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Marek Jelínek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Kartézské Genetické Programování v Evolučním umění
Cartesian Genetic Programming in Evolutionary Art

Zásady pro vypracování:

Práce bude zaměřena na Kartézské Genetické Programování a jeho využití v evolučním umění.

Cíl práce:
1. Popis Kartézského Genetického Programování, srovnání s dalšími evolučními technikami, výhody a a vhodné využití.
2. Seznámení se s technikou evolučního umění, popis algoritmů vhodných pro evoluční umění a srovnání těchto algoritmů s Kartézským Genetickým Programováním.
3. Implementace vlastní aplikace produkující evoluční umění založeném na Kartézském Genetickém programování. Tato aplikace bude zaměřena především na generování a evoluci obrázků s vlastním ohodnocovacím systémem řízeným uživatelem.
4. Dokumentace a popis struktury aplikace.

Seznam doporučené odborné literatury:

[1] Matthew Lewis : Evolutionary visual art and design, The Art of Artificial Evolution, Springer, 2008, ISBN: 978-3-540-72876-4
[2] Julian Francis Miller, Simon L. Harding: Cartesian genetic programming, Genetic and Evolutionary Computation Conference, 2009, ISBN: 978-1-60558-505-5
[3] Laurence Ashmore, JulianFrancisMiller : Evolutionary Art with Cartesian Genetic Programming, EvoMusArt2004, 2004

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Tomáš Buriánek**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Tomáša Buriánka a uviedol som všetky literárne zdroje a publikácie, z ktorých som čerpal.

V Ostrave dňa 27.4.2015

. . . . . .

Marek Jelínek

Poďakovanie

Ďakujem môjmu konzultantovi Ing. Tomášovi Buriánkovi za rady a pripomienky k bakalárskej práci.

Abstrakt

Táto práca sa zaoberá evolučným umením so zameraním na využitie karteziánskeho genetického programovania. Cieľom práce je zhodnotiť využitie karteziánskeho genetického programovania v evolučnom umení a jeho porovnanie s ostatnými evolučnými technikami. V práci sme sa zamerali na problémy evolučného umenia, analýzu rôznych evolučných algoritmov a možnosti reprezentácie generovaného umenia formou obrázkov. Zhodnotíme výhody a nevýhody evolučného umenia a jednotlivých evolučných algoritmov v porovnaní s karteziánskym genetickým programovaním. Pre overenie a testovanie implementujeme aplikáciu generujúcu evolučné umenie na základe interaktívnych evolučných výpočtov formou obrázkov.

Kľúčové slová: Evolučné umenie, Umelá evolúcia, Generatívne umenie, Genetické programovanie, Karteziánske genetické programovanie, Genetické algoritmy, Počítačové učenie, Genetické operátory, Umelý výber, Mutácia, Kríženie, Počítačové videnie, Interaktívne evolučné výpočty, Kansei, Estetické hodnotenie počítačom

Abstract

This thesis deals with evolutionary art with focus on the use of cartesian genetic programming. The aim of this work is to evaluate the use of cartesian genetic programming in evolutionary art and its comparison with other evolutionary techniques. In this thesis we focused on problems of evolutionary art, analysis of different evolutionary algorithms and possibilities of representation of generated art in form of images. We evaluate advantages and disadvantages of evolutionary art and different forms of evolutionary algorithms compared to cartesian genetic programming. For verification and testing purposes we implement application generating evolutionary art based on interactive evolutionary computations in the form of images.

Keywords: Evolutionary art, Artificial evolution, Generative art, Genetic programming, Cartesian genetic programming, Genetic algorithms, Machine learning, Artificial selection, Genetic operators, Mutation, Crossover, Computer Vision, Interactive Evolutionary Computations, Kansei, Computational Aesthetic Evaluation

Zoznam použitých skratiek a symbolov

EU	–	Evolučné umenie
GU	–	Generatívne umenie
EA	–	Evolučný algoritmus
GA	–	Genetický algoritmus
ES	–	Evolučné stratégie
EP	–	Evolučné programovanie
GP	–	Genetické programovanie
CGP	–	Karteziánske genetické programovanie
RVCGP	–	Real-Valued kartezianske genetické programovanie
EMG	–	Elektromyografia
EGG	–	Elektroencefalografia
IEC	–	Interaktívne evolučné výpočty
CAE	–	Estetické hodnotenie počítačom

Obsah

Zoznam tabuliek

Zoznam obrázkov

Zoznam výpisov zdrojového kódu

1 Úvod

Ľudská rasa je viac technologicky vyspelá ako kedykoľvek predtým. Vďaka počítačovému videniu sa nám podarilo vyvinúť autá, ktoré dokážu samy riadiť úžasné megapixelové kamery, ktoré majú automatické zaostrovanie, detekciu tvárí a poskytujú detaily, ktoré ďaleko prekračujú možnosti ľudského oka.

Aj keď sa počítačové videnie čoraz viac stáva súčasťou nášho každodenného života, zatiaľ ani tie najpokročilejšie technológie nedokážu plne nahradiť ľudské videnie. Počítačové videnie sa stalo veľmi dobrým nástrojom pre rôzne analýzy a rozpoznávanie objektov v obrázkoch a to dokonca oveľa presnejšie a rýchlejšie ako kedy dokázal človek. Kamerou zachytávame svetelné lúče, ktoré ukladáme do dvojrozmerného poľa čísel, pixelov. Samotné pixle zachytené kamerou tvoria obrázok a nenesú hlbší význam, pocity, príbeh alebo emócie, tak ako ich môže vnímať človek. U ľudí videnie začína s očami, ale v skutočnosti prebieha v mozgu, ktorý obrázok vníma ako zmes pocitov a skúseností.

Naša spoločnosť kladie čoraz vačší dôraz nielen na kvalitu a funkcionalitu produktov a služieb, ale aj na ich prevedenie, vizuálnu stránku a dizajn. Aj keď je funkcionalita hlavným parametrom väčšiny produktov, výzor a prevedenie u človeka hrá podobne dôležitú úlohu. Pri umení je to úplne naopak, kde je estetický pôžitok vždy a v akejkoľvek podobe na prvom mieste.

Mnoho produktov sa v dnešnej dobe navrhuje za pomoci počítačov, kde systém generuje návrhy a vyhodnocuje ich na základe rôznych vlastností ako napríklad veľkosti produktu, ceny výroby a pod. Počítače sú pri takýchto návrhoch veľmi úspešné, problém však nastáva, ak sú vlastnosti produktu pre počítač neohodnotiteľné. A počítače ako stroje bez emócií majú veľký problém identifikovať a ohodnotiť práve estetickú stránku, ktorá je hlavným aspektom každého umeleckého diela.

V prvej kapitole tejto práce sa budeme snažiť teoreticky uviesť a vysvetliť oblasti evolučného umenia a evolučných algoritmov. Prevažne sa zameriame na aktuálnu situáciu evolučného umenia a predstavíme najväčšie výzvy a nedostatky jeho použitia ako systémov evolučných výpočtov. Uvedieme problematiku evolučných algoritmov a predstavíme niektoré ich používané formy. Hlavne sa zameriame na karteziánske genetické programovanie a jeho využitie v evolučnom umení.

Druhá kapitola je plne venovaná návrhu, implementácii a testovaniu aplikácie. Aplikácia bude generovať obrázky na základe princípov evolučného umenia. Ako evolučný algoritmus bude použité karteziánske genetické programovanie a pokúsime sa aj o implementáciu novšej verzie karteziánskeho genetického programovania, ktorá má zvačšiť neutralitu genotypu a umožniť implementovanie efektívnejšieho genetického operátora kríženia.

2 Teoretická časť

Poznámka 2.1 V práci môžete natrafiť na rôzne termíny prebrané z genetiky, biológie a iných vedných odvetví. Berte prosím na vedomie, že všetky tieto termíny sú brané v oblasti umelej inteligencie a ich význam a úloha môže byť, viac či menej, pozmenená a nemusia presne zodpovedať definíciám z pôvodných vedných oblastí.

2.1 Evolučné umenie

Evolučné umenie je jedným zo systémov umelej evolúcie, ktorý spadá pod generatívne umenie. Rozdielom od klasického generatívneho umenia je proces generovania umenia, ktorý je založený na evolučných princípoch. Evolučné výpočtové techniky sú metódy pre prehľadávanie veľkých viacrozmerných množín riešení pre optimálne výsledky. Plne automatizovaný evolučný systém môže prehľadávať veľmi veľké populácie po mnoho generácií. Najväčšou prekážkou evolučného umenia je nutnosť implementácie CAE, čiže implementácie vyhodnocovacej funkcie, ktorá by automaticky ohodnocovala riešenia na základe ich estetickej hodnoty. [6, 7].

Aj keď bolo o implementáciu CAE mnoho pokusov, je to zatiaľ vzdialene nevyriešený problém. Použité CAE a hybridné systémy, ktoré mali aspoň čiastočný úspech používajú metódy, ktoré sú všeobecne svojrázne a pomerne špecifické pre dané médium, štýl a umelecký cieľ. Napríklad Sanders, Gero a Greenfield skúmali vývoj `agent-based` evolučného systému. Jaskowski et al., Fornari et al. a Ciesielski et al. popisovali systémy s automatickým ohodnocovaním, založenom na forme merania odchýliek od referencie vzoru. McDermott et al. používal kombináciu percepčných meraní, spektrálnych analýz a porovnávanie vzorky po vzorke na nízkej úrovni pre vyvinutie syntetizéru hlasu relatívneho k zvukovej vzorke. [4]

CAE sa s veľkej časti prelína s `Kansei engineering-om` 2.2, ktorému som venoval vlastnú sekciu (sekcia 2.2), ktorá popisuje odbor zaobereajúci sa dizajnom na základe `Kansei`.

Doteraz najlepšou alternatívou CAE je použitie interaktívnych evolučných výpočtov. V IEC je do procesu ohodnocovania zapojený človek, ktorý úplne alebo aspoň sčasti zasahuje do ohodnocovania riešení. Táto metóda je však veľkým znehodnotením efektivity evolučného algoritmu, kde je efektivita viacmenej lineárna s presnosťou a rýchlosťou ohodnocovania riešení človekom. Preto sa pri IEC kladie najväčší dôraz na zefektívnenie interaktívneho ohodnocovania. Ohodnocovanie sa pri rôznych problémoch môže podstatne líšiť a to často vedie k nutnému individuálnemu prístupu zlepšenia kvality ohodnocovacieho systému pre užívateľa. Pri evolúcii obrázkov na základe estetických parametrov môžu ohodnocovanie zlepšiť nasledovné úpravy EA:

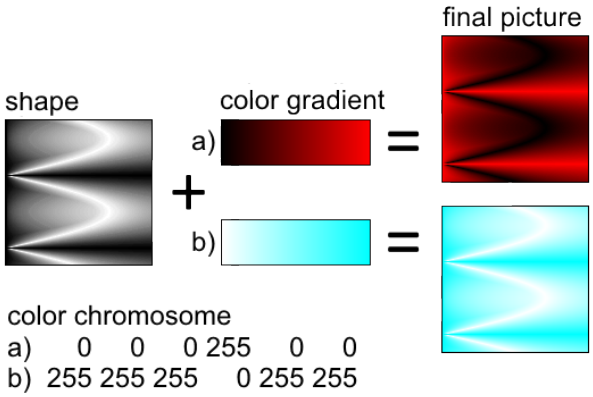
- Človek sa oproti počítaču môže začať nudiť a to hlavne pri monotónnej činnosti. Preto by bolo dobré, aby užívateľ čo najskôr videl známky pokroku. Tomu môže napomôcť zmenšenie veľkosti populácie a sprístupnenie upravovania konvergencie užívateľovi.

- Počet viditeľných obrázkov, ich veľkosť a kvalita by mala byť v rovnováhe, aby užívateľ vedel čo najrýchlejšie určiť estetickú hodnotu obrázkov.
- Ohodnotenie by malo byť pre užívateľa jednoznačné. Oproti CAE, kde môže systém ohodnocovať veľmi detailne sa užívateľ presnosťou až tak zdržovať nemôže a situáciám ako "No, mohlo by to byť za 8 bodov, ale možno je to bližšie ku 9-tim, hmm...", by sme sa mali vyhnúť.
- Ak máme možnosť rozpoznať zlé riešenie, malo by sa automaticky odfiltrovať a užívateľa tým už nezdržovať. Naopak, ak vieme rozpoznať potenciálne dobré riešenie, môže sa užívateľovi prioritne navrhnúť. A to napríklad formou zoradenia od potenciálne najlepšieho po potenciálne najhoršie riešenie.
- V evolučných algoritmoch, hlavne pri použití CGP 2.6, sa vďaka neutralite môže často stať, že sú viaceré riešenia identické alebo veľmi podobné. Obrázky síce môžu byť rovnaké, ale genotyp môže obsahovať zmeny, ktoré sa na obrázku môžu prejaviť v neskorších generáciách. Preto je dôležité zachovať obe riešenia, ale užívateľovi by sme mali prezentovať iba jedno, kde sa jeho ohodnotenie automaticky pripíše obom.

Ohodnotenie od užívateľa môže byť získavané aj nepriamo 2.2. Napríklad v práci Simsa Galapagosa je ohodnotením diela strávený čas užívateľa jeho prezeraním [4].

2.1.1 Vylepšovanie špecifických vlastností obrázku

V štandardnom generatívnom umení sa obrázky generujú často iba náhodne, bez možnosti zachovania, či vylepšenia špecifických vlastností obrázku. Niektoré implementácie generatívneho umenia preto oddelili vlastnosti obrázku, napríklad farbu a tvar a tak umožnili zachovanie alebo zamieňanie týchto vlastností.



Obrázek 1: Vytvorenie obrázku z dvoch chromozómov.

V evolučnom umení sa vďaka genetickým operátorom vlastnosti prenášajú z generácie na generáciu implicitne. O tom, ktoré vlastnosti budú zmenené ale rozhodovať priamo nemôžme, pretože sú zakódované v genotype a rôzne gény môžu zasahovať do rôznych vlastností. Ak by sme chceli mať väčšiu kontrolu nad špecifickými vlastnosťami, tak ako pri oddelení vlastností v generatívnom umení, môžeme implementáciu EA upraviť a genotyp jedinca rozdeliť na niekoľko chromozómov. Každý chromozóm bude reprezentovať inú vlastnosť, napríklad na obrázku (obrázok 1) jeden chromozóm reprezentuje tvar obrázku v podobe počítačového programu a druhý chromozóm reprezentuje farebnú schému v podobe lineárneho gradientu dvoch farieb. Pri použití evolučného algoritmu GA (sekcia 2.8.1) môžu byť tieto farby v chromozóme uložené priamo ako hodnoty fariebných zložiek `r`, `g` a `b`.

2.1.2 Z programu k obrázku

Pri vytváraní obrázkov v evolučnom umení pomocou počítačového programu jedinca máme viacero možností. Obrázok môžeme vytvoriť pomocou pixlov, kde pre každý pixel na pozícii `x` a `y` vyhodnotíme príslušné hodnoty farebných zložiek `r`, `g` a `b`. Program teda akceptuje súradnice pixelu, ktoré spracuje a vráti farbu pixelu na týchto súradniciach. Pseudo algoritmus vytvárajúci obrázok touto metódou (algoritmus 1).

INPUT: p is program representation of an individual.
OUTPUT: Bitmap 256x256.

```
 1: FUNCTION ConstructPicture(p)
 2: BEGIN
 3:    picture  = BITMAP(256,256)
 4:    x = 0
 5:    y = 0
 6:    WHILE y < 256 DO
 7:    BEGIN
 8:       WHILE x < 256 DO
 7:       BEGIN
 9:          rgb = p(x,y)
10:          picture[y,x].r  = rgb[0]
11:          picture[y,x].g = rgb[1]
12:          picture[y,x].b = rgb[2]
13:          x = x + 1
14:       END
15:       y = y + 1
16:    END
17:    RETURN picture
18: END
```

Výpis 1: Algoritmus pre vytvorenie obrázku na základe farieb a súradníc pixlov.

Farby pixlov obrázku môžeme reprezentovať v `RGB`, `HSV` a môžeme pridať priehľadnosť. Vytváranie obrázkov však môže nadobudnúť veľa foriem a nemusia byť založené iba na súradniciach a farbách pixlov obrázku. V evolučnom umení sa ešte často používajú iterujúce funkcie pre tvorbu fraktálov alebo atraktorov. Napríklad Scott Draves vyvinul systém `Electronic Sheep`, ktorý animuje a vyvíja takzvané fraktálové plamene.[4]

Iterujúca funkcia je typ funkcie, ktorá začne od určitej počiatočnej hodnoty a jej výsledok je potom použitý ako jej vstup (rovnica 1). Tento proces sa nazýva `iterácia`, ktorá je potom opakovaná. Výsledky iterácií funkcie potom môžu byť využité pre tvorbu obrázku.

$$\begin{aligned} x_0 &= initial\_value \\ x_{n+1} &= f(x_n) \end{aligned} \tag{1}$$

2.1.3 Normalizácia hodnôt

Pri niektorých metódach sa objavuje problém, pretože vstupy a výstupy programu väčšinou reprezentujú dva odlišné atribúty. Vstupy programu môžu nadobudnúť iba hodnoty v podmnožine hodnôt výstupov. Napríklad pri metóde vytvárania obrázku z pixlov vstupy reprezentujú súradnice pixlu obrázku a výstupy reprezentujú farebnú zložku `RGB` tohto pixlu. Problémom teda je, že hodnoty zložiek `RGB` môžu byť iba v rozmedzí [0, 255] a hodnoty súradníc pixlov sa menia v závislosti od veľkosti obrázka. Z tohto dôvodu sa táto metóda dá použiť iba pre obrázky s veľkosťou do 256x256 pixlov. Väčšie rozlíšenia použiť nemôžeme, pretože hodnoty výstupov `r`, `g` a `b` by mohli prekročiť hodnotu 255.

Jedným riešením tohto problému je normalizovanie vstupov a výstupov programu. Ak špecifikujeme minimálnu hodnotu 0 a maximálnu hodnotu 1, súradnice `x` a `y` potom môžeme pre vstup do programu previesť na príslušné hodnoty v rozmedzí [0,1] nasledovne:

$$\begin{aligned} x_{normalized} &= x/image\_width \\ y_{normalized} &= y/image\_height \end{aligned} \tag{2}$$

A naopak, výsledné normalizované hodnoty farebných zložiek `r`, `g` a `b` z programu môžeme previesť na hodnoty [0,255] nasledovne:

$$\begin{aligned} r &= floor(r_{normalized} * 255) \\ g &= floor(g_{normalized} * 255) \\ b &= floor(b_{normalized} * 255) \end{aligned} \tag{3}$$

2.2 Kansei engineering

`Kansei engineering` je citové alebo emocionálne inžinierstvo, zaoberajúce sa ohodnotením a dizajnovaním produktov na základe `Kansei`. Počiatky `Kansei engineering`-u zaznamenávame už od roku 1970, keď Nagamachi prezentoval metódu vývoja emocio- nálnych produktov, ktorú nazval `emotional engineering`. Nagamachi v roku 1989 založil techniku `Kansei engineering` a definoval ju ako techniku pre prekladanie ľudského `Kansei` do dizajnérskych elementov produktu. Dnes je `Kansei engineering` uznávaná interdisciplinárna metodika dizajnovania produktov, ktorá zasahuje do humanitných, spoločenských a prírodných vied. [11]

Parametere `Kansei` sú pre špecifický produkt najčastejšie získavané pomocou nasledujúcich metód[11, 12]:

- **Slovne**: Slovné vyjadrenie pocitov.
- **Správanie**: Chovanie a akcie ľudí.
- **Mimika**: Výrazy tváre alebo tela.
- **Fyziologické reakcie**: Tep srdca, EMG, EEG a pod.

Parametre Kansei sú poväčšine percepčné a ten istý produkt môže u každého vyvolať rôzne pocity a emócie. Presné ohodnotenie preto zahrňuje veľké množstvo parametrov, ktoré sú väčšinou nelineárne. Pre tieto prípady boli na modelovanie systémov Kansei často použité neurónové siete, fuzzy logika alebo genetické algoritmy. [12]

2.3 Umelá evolúcia

Termín evolúcia je vnímaná v mnohých významoch. Všeobecne ju poznáme ako vývoj, technický pokrok, zmena niečoho na niečo nové, lepšie. Evolúcia sa dá hlavne spozorovať u živých, biologických foriem vďaka prítomnosti DNA. Ak sa na pojem evolúcie pozrieme z vedeckého hľadiska, ide skôr o potvrdenie proporcionality teórie pre určité udalosti.

Základné princípy evolúcie poprvé popísal Charles Darwin vo svojich teóriách, ktoré sa postupne rozširovali už od roku 1838. Tie nám poskytujú vysvetlenie a predpovedanie prirodzene vyskytujúcich sa fenoménov. Taktiež sa pokúša vysvetliť chod prirodzeného sveta. Dnes najviac uznávanou kolekciou evolučných teórií je neodarwinova paradigma alebo neodarvinizmus, ktorých havné princípy sú[5]:

- **Variabilita** - jedinci sa vplyvom prostredia a času menia.
- **Prirodzený výber** - niektorí jedinci v danom prostredí obstoja lepšie ako ostatní.
- **Dedičnosť** - vlastnosti, ktoré jedinec získal vo vývoji sa prenášajú na jeho potomstvo.

Richard Dawkins neskôr poukázal na to, že evolúcia nie je obmedzená iba na živé organizmy a teoreticky k nej môže dôjsť všade tam, kde je prítomná informácia. V odvetví evolučných výpočtov sa spomínané princípy evolúcie snažíme aplikovať formou počítačového algoritmu, takzvaného evolučného algoritmu 2.4, ktorý simuluje takúto evolúciu.

Umelá evolúcia je napodobenina biologickej evolúcie, pri ktorej človek zasahuje do procesu prirodzenej selekcie. Jedným z prvých pomenovaní tejto selekcie je umelá selekcia, ale najnovšie sa môžeme stretnúť s pomenovaním selective breeding. [5, 3]

Pri umelej selekcii, narozdiel od prirodzenej selekcie, výber jedincov neprebieha na základe ich schopnosti prežitia a reprodukovania sa v danom prostredí, ale na základe človekom špecifikovaných vlastností tohto jedinca. Výnimkou by bol systém, ktorý by mal procesy selekcie a reprodukcie implementované v samotnom prostredí. Čiže prežitie a reprodukcia jedinca by bola závislá od samotnej interakcie tohto jedinca

s prostredím. Aj keď sú pravidlá a vlastnosti prostredia definované človekom, pravidlá prirodzenej selekcie nie sú porušené. Príkladom prirodzenej selekcie môže byť automat Game of Life vymyslený J. H. Conwayom v roku 1970. [3, 5]

Úlohou selekcie v evolúcii je v prvom rade zabrániť strate dobrého genetického materiálu pri prechode medzi generáciami uprednostňovaním prežitia a reprodukovania sa lepších jedincov v populácii. V druhom rade by mala selekcia udržovať divergenciu a výber jedincov striedať a kombinovať pre ich väčšiu rozmanitosť. [3]

2.4 Evolučný algoritmus

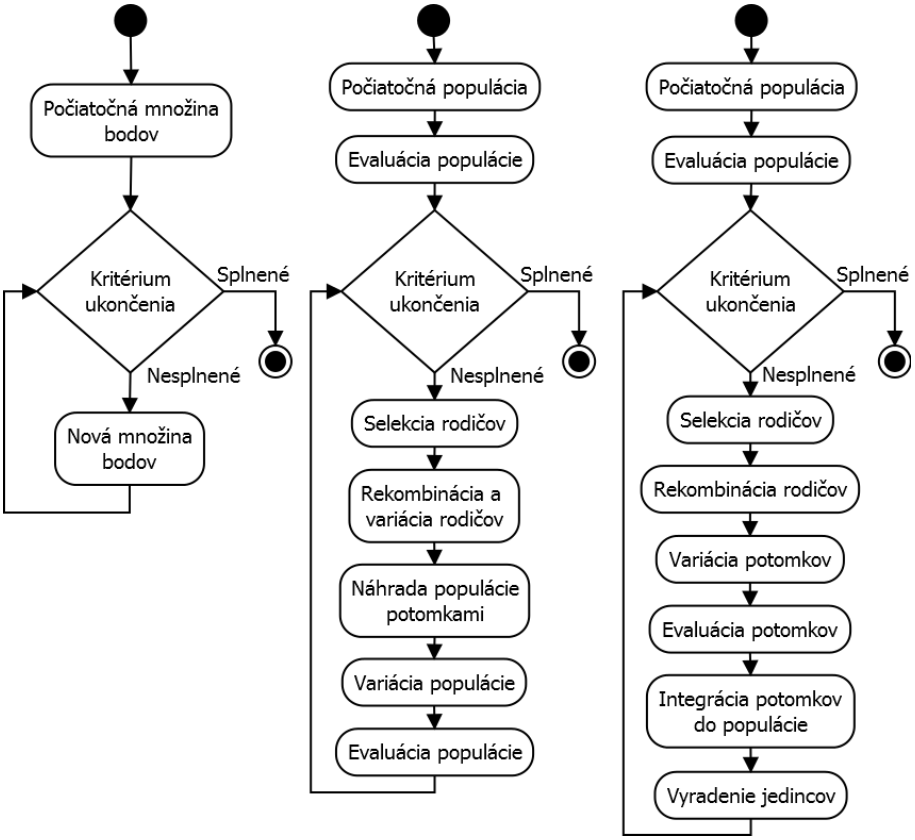
Evolučný algoritmus (EA) je termín, zastrešujúci počítačovo založené systémy, slúžiace na riešenie problémov, ktorých kľúčový dizajn a implementácia výpočtového modelu spočíva na evolučných procesoch. EA doteraz nadobudli mnoho foriem, no všetky zdieľajú rovnaký konceptuálny základ simulácie evolúcie štruktúr jedincov prostredníctvom procesu seleckie 2.3, variácie 2.5.6 a reprodukcie. [6, 7, 8]

Evolučné algoritmy sa začali využívať hlavne ako vyhľadávacie algoritmy pre optimizačné problémy. Radia sa síce medzi stochastické vyhľadávacie algoritmy [10], ale našli uplatnenie aj v generatívnom umení a počítačovom učení. EA sa poväčšine využívajú na také problémy, ktoré sa nedajú, alebo nie tak ľahko dajú, vyriešiť inými technikami. Evolučné algoritmy nie vždy vedia nájsť optimálne riešenie pre daný problém, ale často nájdu aspoň dobré riešenie, ak nejaké existuje[6]. Preto sa využívajú napríklad na problémy, do ktorých zasahuje veľké množstvo parametrov, alebo ak presné riešenie vyžaduje priveľký výpočtový výkon a takmer optimálne riešenie je postačujúce. [6, 8] Klasické prehľadávacie techniky ako lineárne prehľadávanie , deterministický hill climbing, či čisto náhodné prehľadávania, boli pri aplikácii na nelineárne problémy všeobecne neuspokojivé, hlavne pri problémoch so stachostickými, prechodnými alebo chaotickými komponentami.[3]

Ďalšou silnou stránkou EA je robustná adaptácia. Ako ani reálny svet nie je statický, tak ani niektoré problémy, ktoré sa snažíme optimalizovať. Pri malých zmenách v prostredí nemusíme začať optimalizáciu od začiatku, ale môžeme použiť už predošle optimalizovaných jedincov, ktorí by sa mali veľmi rýchlo adaptovať týmto zmenám. To sa dá využívať aj v reálnom čase a vyhodnocovaciu funkciu počas evolúcie upravovať a upresňovať. To umožnuje optimalizáciu aj pri veľkej nepresnosti vyhodnocovacej funkcie (aproximácia fitness 2.5.2). [3].

2.5 Štruktúra evolučného algoritmu

Štruktúra evolučných algoritmov sa v mnohých prácach môže značne líšiť, ale nemenným základom štruktúry je model vyhľadávacieho algoritmu (obrázok 2).



Obrázek 2: Štruktúra vyhľadávacieho algoritmu, jedno a viac populačného EA [8, 9, 10]

2.5.1 Počiatočná populácia

Počiatočná populácia tvorí prvú generáciu jedincov. Títo jedinci sú väčšinou náhodne generovaní, niekedy sa do tejto populácie umelo vkladajú jedinci pre urýchlenie a nasmerovanie konvergencie ku konkrétnemu riešeniu.

Počet jedincov populácie je predom definovaný. V niektorých prípadoch sa v ďalších generáciách môže veľkosť populácie zväčšiť kvôli veľkej rozmanitosti dobrých riešení, kedy by mohlo dôjsť k strate potenciálne dobrého riešenia.

2.5.2 Evaluácia populácie

V tomto kroku každého jedinca vystavujeme prostrediu a hodnotíme jeho výkon v podobe hodnoty `fitness`. Pre ohodnotenie výkonu jedinca používame `fitness funkciu` a sadu vopred definovaných testov, ktorými podrobujeme každého jedinca. Táto fitness funkcia musí byť čo najpresnejšia požadovanému cieľu a čas strávený jej výpočtom musí byť čo najmenší, pretože pre získanie použiteľného výsledku u netriviálnych problémoch musí táto funkcia pre každé riešenie iterovať mnohokrát.

`Aproximácia fitness` sa používa ak sa `fitness funkcia` nepodarí optimalizovať na požadovaný výkon, alebo nemá dostačujúcu presnosť. Táto metóda drasticky redukuje počet iterácií, ale často vedie ku konvergencii do lokálneho optima. V niektorých prípadoch sa používa IEC, v ktorej sám dizajnér nahradzuje, alebo doplňuje `fitness funkciu`. Jedná sa hlavne o prípady, kedy človek na rozdiel od počítača dokáže ľahko a rýchlo ohodnotiť výkon jedinca. Napríklad pri návrhu plošnej dosky fitness funkcia hodnotí cenu použitých súčiastok a človek hodnotí estetickú stránku plošnej dosky.

2.5.3 Kritérium ukončenia

Kritérium ukončenia je podmienka rozhodujúca o ukončení cyklu evolúcie evolučného algoritmu. Primárne kritérium ukončenia najčastejšie býva dosiahnutie optimálneho riešenia problému v podobe hodnoty fitness. Nie vždy sa však podarí dosiahnuť tento výsledok, preto je nutné implementovať aj sekundárne kritérium.

Sekundárne kritérium môže sledovať strávený čas od začiatku spustenia evolúcie, počet cyklov evolučného algoritmu alebo počet cyklov bez zlepšenia výsledkov. Taktiež je dobré a pri použití IEC nutné implementovať možnosť ukončenia evolúcie užívateľom.

2.5.4 Selekcia

Procesy selekcie 2.3 sú závislé na ohodnotení výkonu 2.5.2 jednotlivej štruktúry pri interakcii s definovaným prostredím. Selekcia zabezpečuje výber jedincov z aktuálnej populácie pre vytvorenie novej generácie pomocou genetických operátorov. Hlavným problémom selekcie je dobré udržanie selekčného tlaku a to presne tak, aby EA neskonvergoval na prvé riešenie, ale preskúmal aj iné riešenia, aj keď zozačiatku nemusia byť veľmi dobré. Ide o vyváženie globálneho prehľadávania neznámych regiónov a lokálneho prehľadávania okolia doposiaľ najlepších riešení. Zvýšením selekčného tlaku strácame divergenciu, potlačujeme výber slabších jedincov, čím im nedávame šancu vyvinúť sa a zameriavame sa iba na najlepších jedincov. To vedie ku rýchlej, ale často unáhlenej konvergencii do prvého, nie až tak dobrého riešenia. [3, 10]

Pred samotnou selekciou sa môže niektorým jedincom úplne zamedziť dostať sa do selekčného procesu. V niektorých implementáciách IEC užívateľ nehrá iba rolu ohodnocovania jedincov, ale priamo nahradzuje selekciu a výber jedincov, je takto priamo závislý od užívateľa.

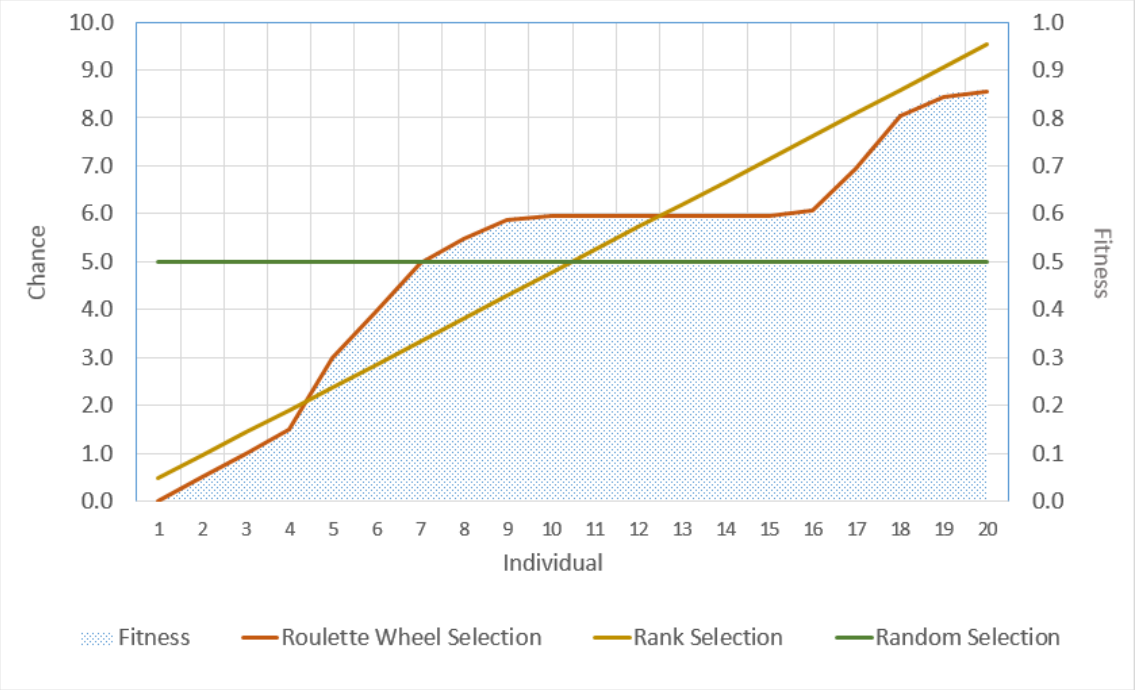
Existuje viacero metód a druhov pre implementáciu selekcie, niektoré používané sú [10]:

- **Elitism** sama o sebe netvorí metódu selekcie, ale zaručuje prežitie najlepších jedincov populácie. Často sa používa v kombinácii s `roulette wheel selection`, `rank selection` a `tournament selection`, kde je šanca, že sa aj ten najlepší jedinec môže stratiť.

- **Truncation selection** rozdeľuje populáciu na dve skupiny. Prvá skupina, ktorá zahrňuje lepších jedincov sa automaticky prenáša do novej populácie. Druhá polovica

je zahodená a nahradená potomkami prvej polovice. Táto metóda je veľmi jednoduchá a výpočtovo nenáročná. Používa sa najmä pri veľmi veľkých populáciách.

- **Steady-state selection** je podobná ako `truncation selection`. U `truncation selection` je väčšinou viac ako polovica populácie nahradená potomkami, kdežto u `steady-state selection` je nahradených iba zopár najhorších jedincov.
- **Roulette wheel selection, rank selection, random selection** priraďuje každému jedincovi populácie percentuálnu šancu na výber. U `roulette wheel selection` je šanca lineárna s hodnotou fitness, kdežto u `rank selection` je šanca lineárna s umiestnením jedinca (obrázok 3).



Obrázek 3: Porovnanie selekcií založených na percentuálnej šanci.

- **Tournament selection** je selekcia pri ktorej môžeme, ľahko, jedným parametrom upravovať selekčný tlak a tak zrýchliť alebo spomaliť konvergenciu EA. Je to jedna z najideálnejších selekčných metód pre IEC, pretože užívateľovi poskytuje väčšiu kontrolu nad EA. Nastaviteľný parameter `k` reprezentuje veľkosť turnamentu. Ak veľkosť turnamentu nastavíme na `k=1`, tak dostávame ekvivalentnú šancu na výber pre každého jedinca podobne ako u `random selection` a postupným zvyšovaním tejto hodnoty sa zvyšuje šanca na výber na základe hodnoty `fitness`. Pseudo algoritmus (algoritmus 2) implementácie `tournament selection`.

INPUT: P is zero−indexed array where $P = p_0, p_1 ... p_{(n-1)}$, n is size of array P, $n > 0$, p_m is individual at index m and k is Tournament size where $k > 0$.
OUTPUT: Individual.

```
 1: FUNCTION TournamentSelection(P, k)
 2: BEGIN
 3:    best_individual = P[random(0, length(P)−1]
 4:    i = 1
 5:    WHILE i < k DO
 6:    BEGIN
 7:       opponent = P[random(0, length(P)−1]
 8:       IF opponent.fitness >= best_individual.fitness THEN
 9:       BEGIN
10:          best_individual = opponent
11:       END
12:       i = i + 1
13:    END
14:    RETURN best_individual
15: END
```

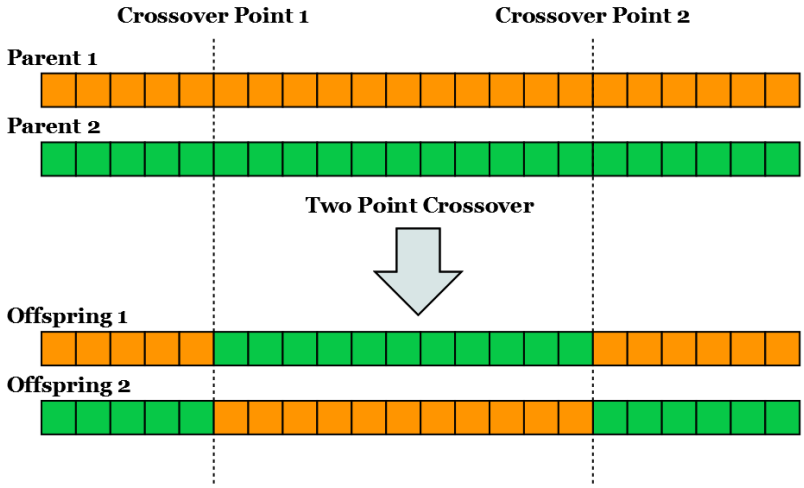
Výpis 2: Pseudo Turnament selection algoritmus

- **Boltzmann selection** je selekcia, ktorá automaticky upravuje selekčný tlak tak, aby udržiaval veľkú divergenciu. Pri tejto selekcii sa málokedy stáva, že problém skonverguje do lokálneho optima.

2.5.5 Rekombinácia

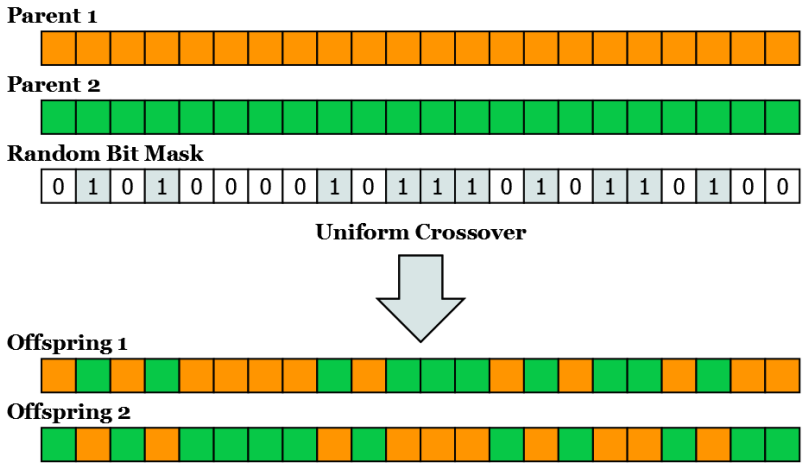
Rekombinácia je jedným z genetických operátorov evolučných algoritmov a často sa nazýva `crossover`. Jej účelom je vytvorenie nového jedinca kombináciou dvoch alebo viacerých jedincov. Jej cieľom je vytvorenie potomka, ktorý bude mať niektoré vlastnosti zdedené od jedného a niektoré od druhého rodiča. Niektoré implementácie EA rekombináciu nepoužívajú, buď z toho dôvodu, že to neumožnuje štruktúra alebo z dôvodu, že jej použitie zhoršilo efektivitu EA. U foriem GP2.8.3 a EP2.8.2 sa kvôli stromovej štruktúre rekombinácia nemôže aplikovať priamo na genotyp a aplikuje sa až na fenotyp a to formou výmeny celého podstromu (obrázok 9).

- **Single point crossover:** Zvolí sa jeden `crossover point`, ktorý rozdelí genotypy oboch jedincov a pravú alebo ľavú stranu si zamenia. Takto vzniknú dvaja noví jedinci.
- **Two point crossover:** Zvolia sa dva `crossover point`-y podobne ako u `single point crossover`, tu nám ale vzniknú tri časti. V tomto prípade zamieňame strednú časť (obrázok 4).
- **Multipoint crossover:** Zvolí sa `n crossover point`-ov, ktoré genotypy rozdelia na `n + 1` časti. Zamieňame každú párnu, alebo nepárnu časť.



Obrázek 4: Two Point Crossover.

- **Uniform crossover:** Pri `uniform corssover` náhodne zamieňame jednotlivé gény jedincov pomocou bitovej masky (obrázok 5).



Obrázek 5: Uniform Crossover.

- **Arithmetic crossover**: Táto metóda kríženia sa využíva hlavne pri evolučných algoritmoch ako GA. Kríženie spočíva v použití aritmetických funkcií na jednotlivé gény. Napríklad hodnota génu potomka je priemernou hodnotou rodičovských génov.

2.5.6 Variácia

Pri variácii dochádza ku zmene genetickej informácie jedinca. Takáto zmena sa nazýva mutácia a v genetike je považovaná za chybu genetickej informácie.

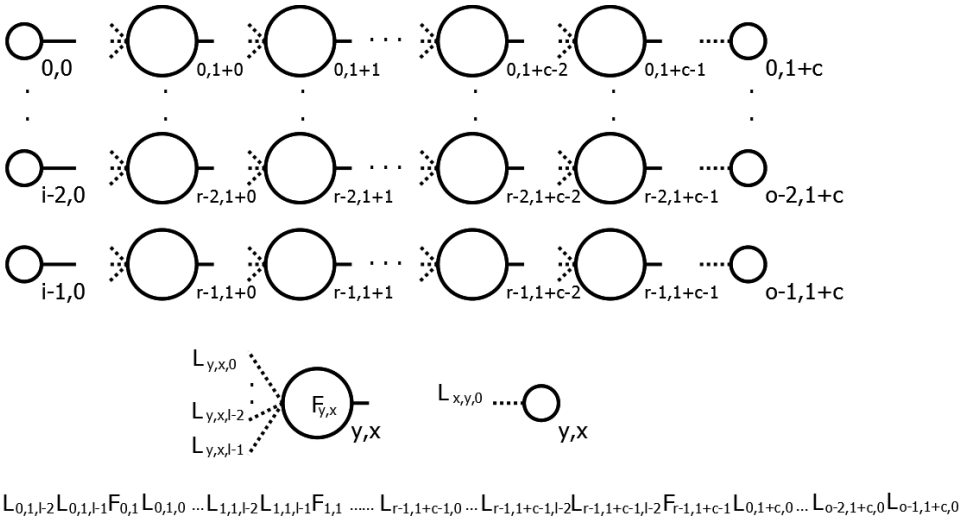
Efekty takejto mutácie môžu byť rôzne:

- **Prospešné:** zmutovaný jedinec má lepšie vlastnosti.
- **Škodlivé:** zmutovaný jedinec má horšie vlastnosti.
- **Neutrálne:** mutácia nemala priamy vplyv na vlastnosti jedinca. Zmena genetickej informácie sa môže prejaviť na jeho potomkoch.

2.6 Karteziánske genetické programovanie

Karteziánske genetické programovanie bolo pôvodne vyvinuté Julianom F. Millerom a Petrom Thomsonom pre návrh digitálnych obvodov. Nazýva sa `karteziánske`, pretože reprezentuje program použitím dvojrozmernej mriežky vrcholov, ktoré sú adresované v karteziánskom súradnicovom systéme[1]. Predchodcom CGP je genetické programovanie 2.8.3, ktoré je veľmi podobné a líši sa v reprezentácii programu [3].

CGP bolo spočiatku zamerané iba na boolové problémy, ale ukázalo sa, že je to užitočná metóda pre evolúciu programov používajúcich aj iné dátové typy. Existujú podobné systémy ako CGP napríklad `Parallel Distributed Genetic Programming` alebo grafovo založený GP systém `Parallel Algorithm Discovery and Orchestration`. Genotyp je pri CGP reprezentovaný ako reťazec čísel pevnej dĺžky, kde sú čísla mapované do programu s topológiou orientovaného grafu. [1, 2]



Obrázek 6: Karteziánske genetické programovanie - grafová reprezentácia. [3]

Na obrázku (obrázok 6) môžeme vidieť grafovú reprezentáciu. Graf reprezentuje program, ktorý má i vstupných vrcholov, o výstupných vrcholov a $r * c$ funkčných vrcholov.

Hodnoty génov sú v CGP značne obmedzené a pri vytváraní alebo mutovaní genotypu ich musíme dodržať inak by mohol vzniknúť nevalidný program. V sekcii (sekcia

2.7) je prezentovaná nová metóda odvodená od CGP nazvaná RVCGP, ktorá odstraňuje niektoré obmedzenia genotypu.

Gény genotypu reprezentujúce funkciu hodnotou F, ktorá môže nadobudnúť iba platnú adresu v referenčnej tabuľke funkcií (sekcia 2.6). Tabuľka väčšinou býva indexovaná od nuly, preto pre F, kde f je počet funkcií v tabuľke musí platiť

$$0 <= F < f \tag{4}$$

Index funkcie	Definícia funkcie
0	$\begin{cases} x+y & x+y <= 1 \\ 2-(x+y) & x+y > 1 \end{cases}$
1	$\begin{cases} x-y & x-y >= 0 \\ -(x+y) & x-y < 0 \end{cases}$
2	$\begin{cases} x*y & x*y <= 1 \\ 1 & x*y > 1 \end{cases}$
3	x^y
4	$\sin(x*\pi)$
5	$\cos(\frac{x*\pi}{2})$
6	$\frac{\sin(x*\pi)+\sin(y*\pi)}{2}$
7	$\frac{\lvert\sin(2*x*\pi)+\cos(2*y*\pi)\rvert}{2}$
8	$\frac{\lvert\sin(2*x*\pi)+\cos(2*y*\pi)\rvert}{2}$
9	$\frac{\lvert\cos(2*x*\pi)-\cos(2*y*\pi)\rvert}{2}$
10	$\frac{\lvert\cos(3*x*\pi)+\sin(2*y*\pi)\rvert}{2}$
11	$\lvert\sin(50*x*y*\pi)+\sin(2*y*\pi)\rvert$
12	$\lvert\frac{e^{x+y}-e^{-(x+y)}}{e^{x+y}+e^{-(x+y)}}\rvert$
13	$\tan(\frac{\pi*(x+y)}{8})$
14	$\sqrt{\frac{x*x+y*y}{2}}$
15	$\frac{x+y}{2}$
16	$\begin{cases} x & x > y \\ y & x <= y \end{cases}$
17	$\begin{cases} x & x <= y \\ y & x > y \end{cases}$
18	$1-x$
19	$log_2(e-x*(e-1))$
20	0.3

(5)

Tabulka 1: Referenčná tabuľka funkcií pre hodnoty v rozmedzí [0.0,1.0].

Prepojenie vrcholov grafu v CGP je taktiež obmedzené. Graf je orientovaný, preto sú prepojenia možné iba v jednom smere a každý `funkčný` vrchol v stĺpci x musí mať práve

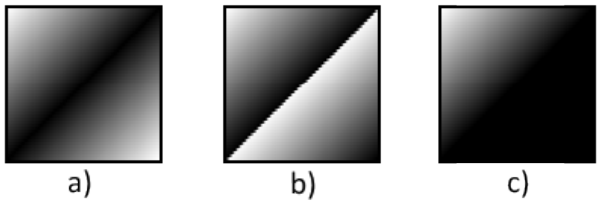
l prepojení s vrcholmi iba v predošlých stĺpcoch, čiže s vrcholmi množiny stĺpcov K, kde $K = \{0, 1, ..., x-2, x-1\}$. Parameter l udáva počet vstupov funkcií v referenčnej tabuľke funkcií2.6 a preto počet prepojení funkčného vrcholu zľava. Výstupné vrcholy určujú z ktorého vrcholu bude použitý výsledok a pre výstupné vrcholy platí to isté, ale hodnota l je vždy rovná jednej. [1]

Uvažujeme o vrchole v stĺpci x a riadku y potom všetky vhodné indexy vrcholov na prepojenie $L_{y,x}$, kde i je počet `vstupných` vrcholov grafu a r je počet riadkov `funkčných` vrcholov grafu, musí platiť[1]:

$$0 <= L_{y,x} < i - 1 + x * (r - 1) \tag{6}$$

Referenčná tabuľka funkcií obsahuje funkcie, ktoré môžu `funkčné` vrcholy reprezentovať. Každá funkcia má k dispozícii definovaný počet vstupov a vo väčšine prípadov sa jedná o dva vstupy, pretože aj väčšina boolových a matematických operácií vyžaduje dva vstupy. Preto budeme ďalej v práci uvažovať už iba o funkciách o dvoch vstupoch. Tieto vstupy môžeme, ale nemusíme využiť. Preto je možné použiť `funkciu` ako konštantu a to ak nevyužieme ani jeden vstup. Vstupy do funkcií by mali byť vždy v definičnom obore a ich výsledok, obor hodnôt, musí byť podmnožinou definičného oboru. [1]

Pri použití boolových operácií nebolo nutné ošetrovať vstupy ani výstupy funkcií, pretože výstupy operácií môžu nadobudnúť iba hodnoty 0 alebo 1. V evolučnom umení nám boolové operácie nebudú postačovať a musíme použiť matematické operácie, ktoré musíme ošetriť, aby výsledok spadal do definovaného rozsahu hodnôt. [1]



Obrázek 7: Efekty ošetrenia výstupu funkcie sčítania.

Uvažujme o operácii sčítania vo funkcii $f(x, y) = x + y$, kde je jej definičným oborom $D(f) = \{[x, y] \in Z : 255 >= x >= 0 \bigwedge 255 >= y >= 0\}$. Aby platilo $R(f) \subseteq D(f)$ musíme funkciu upraviť a na to máme viacero možností:

$$\text{a)} \quad f(x,y) = \begin{cases} x+y & \text{if } x+y <= 255 \\ 510-(x+y) & \text{else} \end{cases} \tag{7}$$

$$\text{b)} \quad f(x,y) = \begin{cases} x+y & \text{if } x+y <= 255 \\ (x+y)-255 & \text{else} \end{cases} \tag{8}$$

$$\text{c)} \quad f(x,y) = \begin{cases} x+y & \text{if } x+y <= 255 \\ 255 & \text{else} \end{cases} \tag{9}$$

Efekty týchto funkcií môžeme znázorniť na 2D plátne a vidieť na obrázku (obrázok 7), kde môžete spozorovať značne veľké rozdiely aj keď ide o rovnakú operáciu. U ob-

rázkoch `a)` a `c)` ide prevažne o pomalú zmenu, ale pri obrázku `b)` môžeme spozorovať veľmi agresívnu zmenu, čo môže mať úplne odlišný efekt na vygenerované obrázky.

Pôvodná reprezentácia, kde sa genotyp mapoval do stromovej štruktúry bola navrhnutá pre umožnenie použitia genetického operátoru kríženia pre genotyp variabilnej dĺžky. Stromová reprezentácia programu sa využíva hlavne v Genetickom a Evolučnom programovaní. Jednou z výhod grafovej reprezentácie je implicitné opakované využitie uzlov orientovaného grafu. [3, 1]

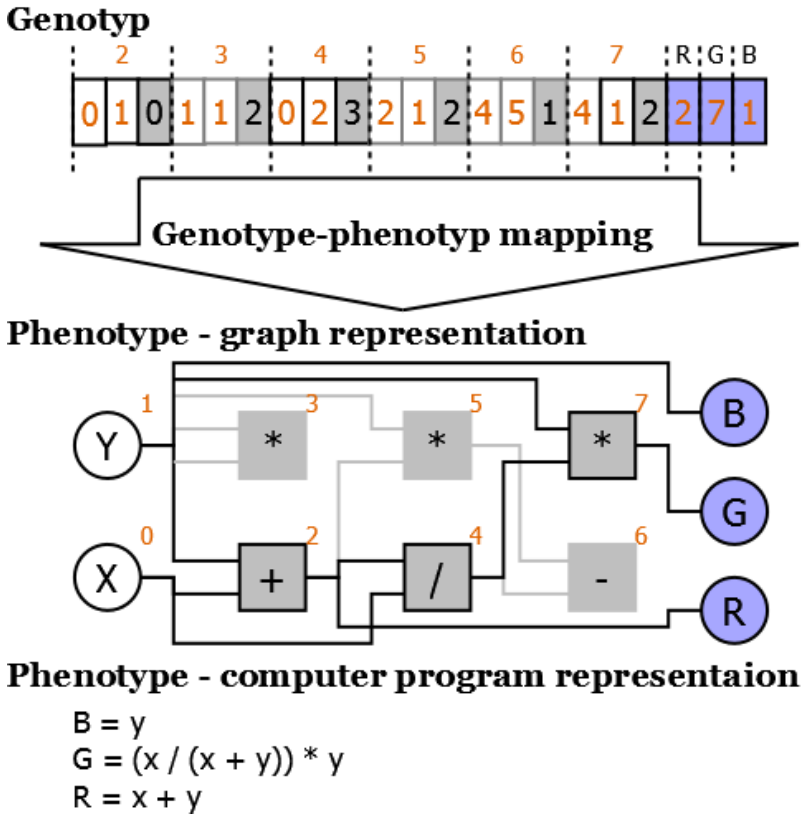
INPUT: r is number of rows, c is number of columns, i is number of inputs, o is number of outputs and F is array of functions
OUTPUT: Randomly generated genotype.

```
 1: FUNCTION GenerateGenotypeInCGP(r, c, i, o, F)
 2: BEGIN
 3:   genotype_length = o + r * c * 3
 4:   genotype = GENOTYPE(genotype_length)
 5:   number_of_available_nodes = i
 6:   gene_index = 0
 7:   column_index = 0
 8:   WHILE column_index < c DO
 9:   BEGIN
10:     row_index = 0
11:     WHILE row_index < r DO
12:     BEGIN
13:       genotype[gene_index] = random(0, number_of_available_nodes − 1)
14:       genotype[gene_index + 1] = random(0, number_of_available_nodes − 1)
15:       genotype[gene_index + 2] = random(0, length(F) − 1)
16:       gene_index = gene_index + 3
17:       row_index = row_index + 1
18:     END
19:     number_of_available_nodes = number_of_available_nodes + r
20:     column_index = column_index + 1
21:   END
22:   output_index = 0
23:   WHILE output_index < o DO
24:   BEGIN
25:     genotype[gene_index] = random(0, number_of_available_nodes − 1)
26:     gene_index = gene_index + 1
27:   END
28:   RETURN genotype
29: END
```

Výpis 3: Vytvorenie genotypu pre CGP

Hlavné motivácie pre využívanie grafovej reprezentácie bol fakt, že grafy sú univerzálnejšie ako stromy a mali lepší výkon pri evolúcii boolových programov. Grafová reprezentácia má síce oproti stromovej reprezentácii fixnú dĺžku genotypu, ale fenotyp má variabilnú dĺžku vďaka mapovaniu, ktoré nevyžaduje využitie všetkých génov. Nevyužité gény sa nazývaju `neutrálne gény` a majú veľmi pozitívny vplyv na efektivitu evolučného algoritmu. Dôležitosť a potenciálne výhody genotyp-fenotyp mapovania boli diskutované Banzhalfom v Binárnom genetickom programovaní (BGP). V BGP sa naviac oproti CGP musel genotyp reprezentovaný ako binárny reťazec vždy opraviť na validný

program, kde v CGP tieto opravy nie sú nutné, pretože už samotný graf vždy reprezentuje validný program.



Obrázek 8: Karteziánske genetické programovanie.

V CGP môžeme použiť ako genetický operátor kríženia `single`, `two`, resp. `multi-point crossover`. Vďaka grafovej reprezentácii majú každé dva genotypy rovnaké obmedzenia pre gény na tej istej pozícii v genotype. Preto ako môžete vidieť v pseudo algoritme (algoritmus 4), môžeme medzi jedincami zamieňať všetky gény.

INPUT: $parent\_a$ and $parent\_b$ are genotypes of parents a and b, $number\_of\_points$ defines number of crossover points .
OUTPUT: Offspring genotype.

```
1: FUNCTION MultipointCrossover(parent_a, parent_b, number_of_points)
2: BEGIN
3:    genotype_length = length(parent_a)
4:    offspring  = GENOTYPE(genotype_length)
5:    crossover_points = []
6:    FOR i = 0;  i  < number_of_points; i++ DO
7:    BEGIN
8:        crossover_points.push(random(0, genotype_length − 1))
9:    END
10:   crossover_points.push(genotype_length−1)
```

```
11:   crossover_points.sort_asc()
12:   gene_index = 0
13:   point = 0
14:   active_parent = parent_a
15:   WHILE point < length(crossover_points) DO
16:   BEGIN
17:       IF gene_index >= crossover_points[point] THEN
18:       BEGIN
19:           point = point + 1
20:           IF active_parent == parent_a THEN
21:               active_parent = active_b
22:           ELSE
23:               active_parent = active_a
24:           CONTINUE
25:       END
26:       offspring[gene_index] = active_parent[gene_index]
27:       gene_index = gene_index + 1
28:   END
29:   RETURN offspring
30: END
```

Výpis 4: Multipoint crossover pre CGP

2.7 Real-Valued CGP

Pri karteziánskom genetickom programovaní bolo preukázané, že má lepší výkon ako tradičné genetické programovanie. Implementácie CGP nepoužívajú genetický operátor kríženia, ale iba mutáciu, pretože neskoršie práce ukázali, že zhoršovala rýchlosť konvergencie. V tejto sekcii popíšem upravenú verziu CGP, ktorá implementuje nový genetický operátor kríženia, pri ktorom je konvergencia rýchlejšia ako pri použití iba mutácie v tradičnom karteziánskom programovaní. Táto nová technika sa nazýva `real-valued` karteziánske genetické programovanie. [2]

Inšpirácia pre vytvorenie tejto novej techniky bol `real-valued crossover` operátor použitý v `real-valued` genetických algoritmoch. Pri tradičnom CGP genotyp pozostáva zo zoznamu celých čísel, ktoré kódujú orientovaný graf. Začlenenie tohto nového typu kríženia do CGP vyžaduje úpravu samotnej reprezentácie CGP. RVCGP modifikuje reprezentáciu genotypu, kde je orientovaný graf reprezentovaný ako zoznam reálnych čísel. Každé reálne číslo odpovedá jednému génu v genotype CGP a jeho hodnota je v rozmedzí [0,1]. Celá zmena spočíva iba v reprezentácii genotypu, genotyp-fenotyp mapovanie už používa štandardnú celočíselnú formu genotypu. Dekódovanie RVCGP genotypu na CGP genotyp spočíva v kombinácii dvoch vzorcov[2]:

$$floor(gene_i * length(function\_set)) \tag{10}$$

$$floor(gene_i * number\_of\_available\_nodes\_for\_connection) \tag{11}$$

Vzorec (vzorec 10) je použitý na gény kódujúce funkciu `funkčného` vrcholu. Vzorec (vzorec 11) je použitý na gény kódujúce prepojenie vrcholov.[2]

Nová metóda kríženia je aplikovaná na genotype RVCGP s reálnymi číslami podobne ako u `floating point` genetických algoritmoch. Vybraní sú dvaja rodičia p a q a kríženie je prevedené použitím vzorca (vzorec 12) na jednotlivé gény p_i a q_i rodičov pre vytvorenie nových génov o_i potomka o. Hodnota r je jednotne vygenerované číslo pre všetky gény jedinca o, kde $0 < r < 1$. [2]

$$o_i = (1 - r) * p_i + r * q_i \tag{12}$$

Genotyp pri tejto novej reprezentácii stráca skoro všetky obmedzenia definované v tradičnom CGP a jedinými obmedzeniami sa stávajú: dĺžka genotypu a hodnoty všetkých génov v rozmedzí [0,1]. [1, 2] Porovnanie vytvorenia genotypu v CGP (algoritmus 3) a RVCGP (algoritmus 5).

INPUT: r is number of rows, c is number of columns, i is number of inputs, o is number of outputs and F is array of functions
OUTPUT: Randomly generated genotype.

```
1: FUNCTION GenerateGenotypeInRVCGP(r, c, i, o, F)
2: BEGIN
3:    genotype_length = o + r * c * 3
4:    genotype = GENOTYPE(genotype_length)
5:    gene_index = 0
6:    WHILE gene_index < genotype_length DO
7:    BEGIN
8:       genotype[gene_index] = random(0.0, 1.0)
9:       gene_index = gene_index + 1
10:   END
11:   RETURN genotype
12: END
```

Výpis 5: Vytvorenie genotypu pre RVCGP

2.8 Formy evolučných algoritmov

V polovici 1960 boli zavedené tri základné formy evolučných algoritmov. Základy evolučného programovania boli predstavené Lawrencom Fogelom v San Diegu, základy genetických algoritmov boli vyvinuté na `University of Michigan` Hollandom a evolučné stratégie boli vyvinuté skupinou troch študentov, Bienerta, Rechenberga a Schwefela na `Technical University of Berlin`. V priebehu nasledovných 25 rokov boli všetky tieto formy vyvíjané nezávisle, čo viedlo k unikátnym paralelným históriám. Až v roku 1990 bolo zaznamenané organizované úsilie pre vytvorenie fóra na interakciu medzi jedno- tlivými výskumnými komunitami EA. Toto nabralo formu medzinárodného workshopu s názvom `Parallel Problem Solving from Nature`. [3]

2.8.1 Genetické algoritmy

Prvé poznatky základných myšlienok využitia evolúcie systémov sa nachádzajú v Hollandových dokumentoch začiatkom roku 1960. V nich Holland predstavuje ambiciózne prin-

cípy adaptívnych systémov, ktoré sú schopné modifikovať samy seba na základe interakcie s prostredím, v ktorom fungujú. [3]

Genetické algoritmy boli poprvé predstavené v roku 1975 Johnom Hollandom. Vtedy mali GA tri odlišné vlastnosti oproti ostatným evolučným algoritmom[3]:

- Použitá reprezentácia: bitové reťazce
- Metóda výberu: proporciálny výber
- Primárna metóda variácií: `crossover`

Mnoho odvodených implementácií GA používajú alternatívne metódy selekcií a variácií. Taktiež ich veľa zmenilo reprezentáciu bitového reťazcu na inú reprezentáciu, podľa vhodnosti pre riešený problém. Použité reprezentácie boli napríklad: `real-valued` parametre, permutácie alebo stromové hierarchie.

Koza vyvinul metódu založenú na GA, ktorú nazval genetické programovanie2.8.3, ktorá vyvíja programy pri ktorých sú dátové štruktúry `LISP S`-výrazy. [3]

S GA sa v evolučnom umení môžeme stretnúť hlavne pri dizajnovaní alebo pri modelovaní objektov. Pri dizajne to môže byť v podobe rozloženia komponentov alebo farebnej schémy užívateľského rozhrania. Pri vytváraní obrázkov môže GA reprezentovať použité farby (obrázok 1) a pod. [3, 6]

2.8.2 Evolučné programovanie a evolučné stratégie

Aj keď boli obe formy, evolučné programovanie a evolučné stratégie vyvinuté nezávisle, sú si veľmi podobné.

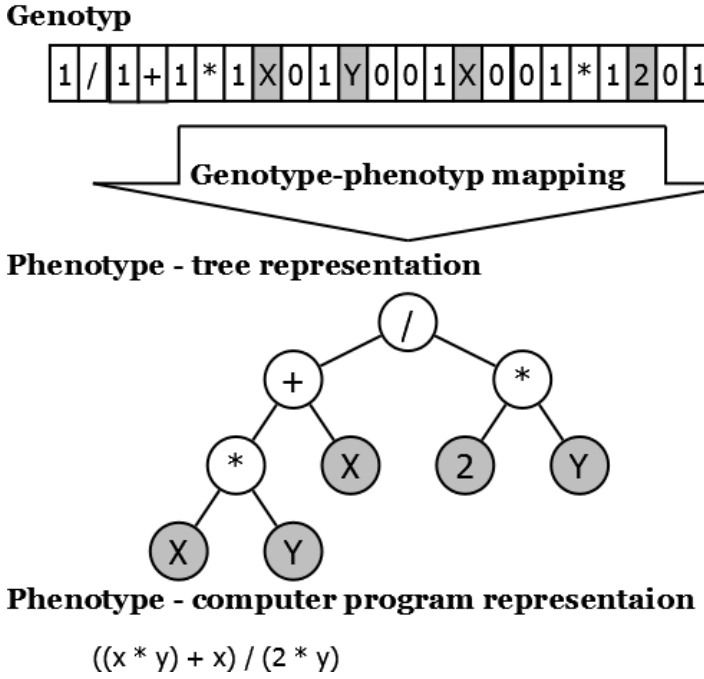
Evolučné programovanie bolo navrhnuté Lawrence J Fogel-om v roku 1960 počas služby na `National Science Foundation`. V tom čase bola UI hlavne zameraná na heuristiku a simuláciu primitívnych neurónových sietí. Fogelovi bolo jasné, že oba tieto prístupy boli obmedzené, pretože modelovali človeka skôr ako základný proces evolúcie. [3]

Evolučné stratégie vyvinuli traja študenti univerzity `Technical University of Berlin`, ktorí chceli zostrojiť výskumného robota, ktorý mal vykonávať rôzne experimenty v aerodynamickom tuneli pre minimalizovanie odporu. Pri hľadaní tejto minimlalizačnej metódy bola v roku 1965 vyvinutá prvá verzia evolučných stratégií, neskôr známych aj pod názvom `(1+1) ES`.

Dva hlavné rozdiely medzi EP a ES sú popísané v nasledujúcich vetách. Prvým rozdielom je použitie selekčnej metódy. EP zvyčajne používa stachostickú metódu selekcie pomocou turnamentu, v ktorom sú riešenia s najmenším počtom výhier vyradené a ES používa deterministickú metódu selekcie, v ktorej sú najhoršie riešenia vyradené z populácie priamo na základe ich ohodnotenia. Druhým rozdielom je použitie rekombinácie. EP rekombináciu poväčšine nepoužíva, pretože evolúcia je zameraná na vývoj druhov a k reprodukcii medzi rôznymi druhmi nedochádza. Pri ES je zasa evolúcia na úrovni individuálneho správania a preto prevažne používa rekombináciu.

2.8.3 Genetické programovanie

Genetické programovanie sa považuje za špeciálny prípad GA, kde je každé riešenie reprezentované počítačovým programom a dáta genotypu tvoria štruktúru tohto programu. Vyvíjané programy sú reprezentované v `LISP` stromoch (obrázok 9).



Obrázek 9: Genetické programovanie.

Počítačový program je v pamäti počítača reprezentovaný ako stromová štruktúra. Prvky tohto stromu sú rozdelené na takzvané `funkčné` vrcholy a `terminálne` vrcholy. `Funkčný` vrchol reprezentuje matematický operátor a môže byť buď koreňom alebo uzlom stromu. Každý takýto `funkčný` vrchol musí mať práve dvoch potomkov. `Terminálny` vrchol je listom stromu, reprezentuje operanda a je konečným prvkom vetvy.

V GP je hlavným genetickým operátorom kríženie. Avšak kvôli stromovej štruktúre sa tento operátor nemôže aplikovať priamo na genotyp, ale musí sa aplikovať na fenotyp a to formou zámeny celých podstromov. V GP je genotyp variabilnej dĺžky a medzi genotypom a fenotypom nie je rozdiel. A to v tom zmysle, že v GP nie sú neaktívne gény. [6, 8] V CGP sa do genotypu kódujú aj neaktívne gény, ktoré sa už na fenotype neprejavia.

Využitie GP v evolučnom umení je podobné ako pri CGP. GP je trochu viac zložitejšie na implementáciu a pre použitie viacero výstupov vo vyvíjaných programoch je nutné stromovú reprezentáciu trochu upraviť.

3 Implementácia aplikácie

Tvorbu aplikácie sme rozdelili do piatich častí: príprava, analýza, návrh, implementácia a testovanie.

3.1 Príprava

3.1.1 Účel aplikácie

Aplikáciu tvoríme pre overovanie teoretickej časti tejto práce.

3.1.2 Cieľ aplikácie

Cieľom tvorby aplikácie je poskytnutie prostredia, na ktorom budeme môcť overiť a otestovať jednotlivé časti práce. V evolučnom umení plánujeme otestovať rôzne reprezentácie riešení formou obrázkov a otestovať použitie ICE. CAE implementovať do aplikácie neplánujeme. Pokúsime sa implementovať a následne porovnať dva evolučné algoritmy a to CGP a RVCGP. V oboch evolučných algoritmoch implementujem genetické operátory mutáciu a kríženie.

3.2 Analýza

V tejto sekcii zhrnieme požadované vlastnosti a funkcionalitu aplikácie.

Požadované vlastnosti aplikácie:

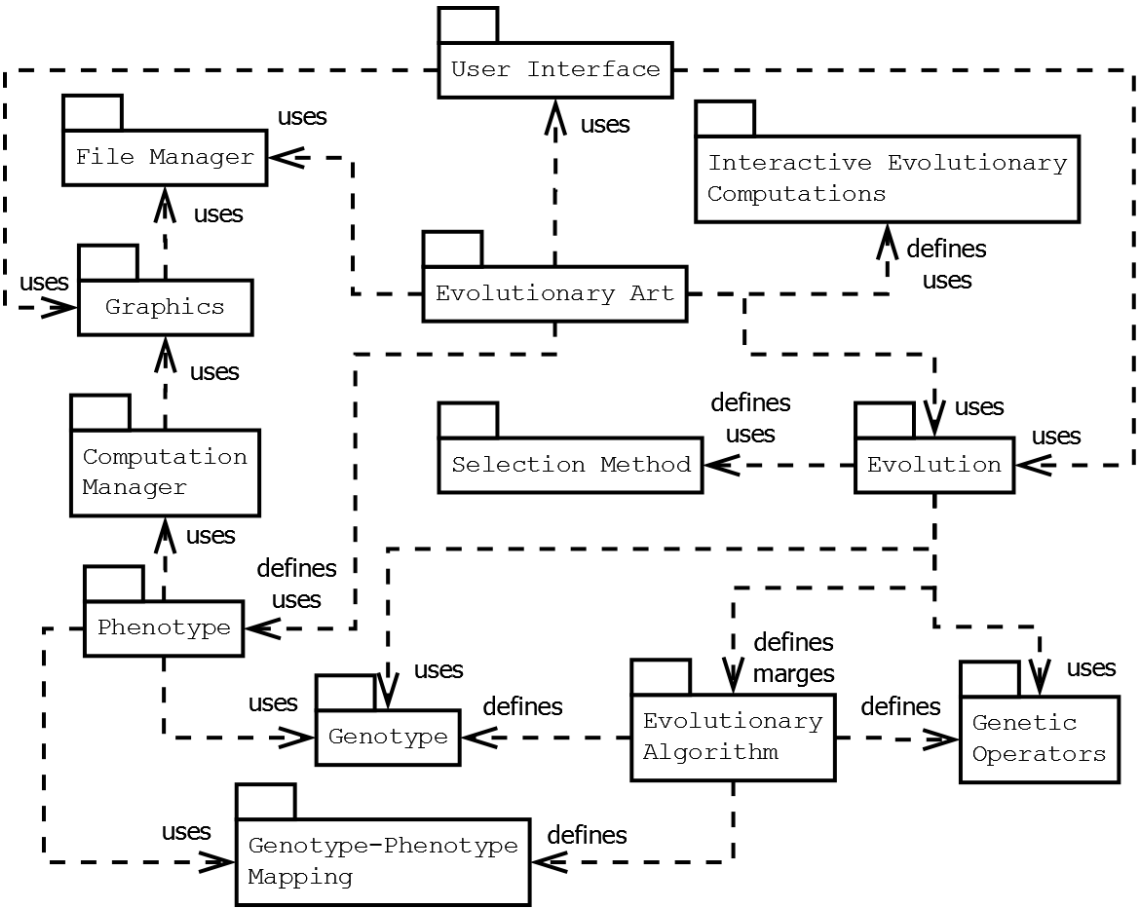
- Podpora **zobrazenia obrázkov** reprezentujúcich jedincov populácie.
- **Užívateľská interakcia**, zadávanie rôznych parametrov, výber a ohodnocovanie obrázkov.
- Podpora **práce s obrázkami**: vytvorenie obrázku, práca s pixlami obrázku.
- **Práca so súbormi**: uloženie vygenerovaných obrázkov, uloženie a načítanie projektu aplikácie.
- Pokročilé **matematické operácie**: odmocnina, sínus, kosínus, logaritmus a pod.
- Dobrý **výpočtový výkon**: podpora využívania viacero jadier CPU alebo technológií ako CUDA a pod.

3.3 Návrh

Pri návrhu štruktúry programu sa zameriame na oddelenie funkcionality komponentov. Pri návrhu užívateľského prostredia sa zameriame na prehľadnosť jedincov populácie.

3.3.1 Návrh štruktúry aplikácie

Aplikáciu sa pokúsime navrhnúť tak, aby jej jednotlivé komponenty boli nahraditeľné, bez nutnosti zásahu do iných komponentov. Zameriame sa na kľúčové komponenty evolučného umenia ako evolučného algoritmu, ICE, selekčnej metódy a reprezentácie jedincov.

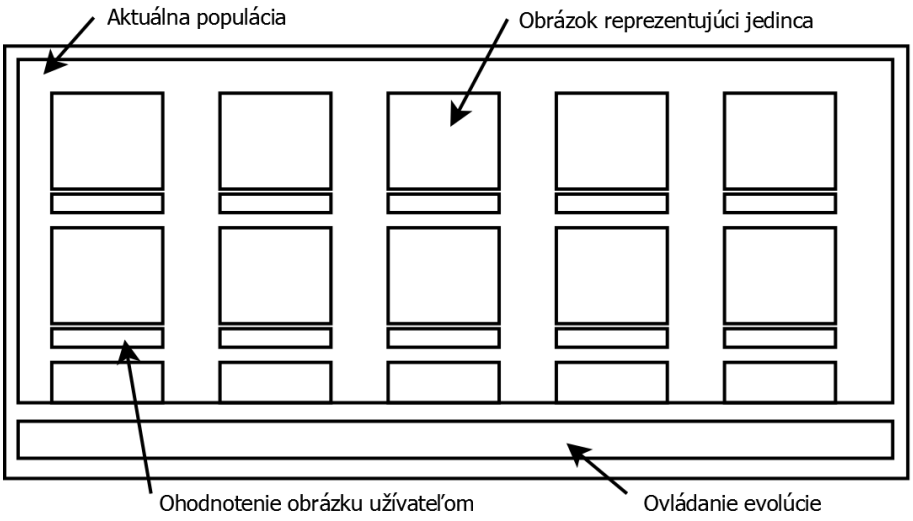


Obrázek 10: Návrh štruktúry aplikácie.

Štruktúru aplikácie môžete vidieť na `package` diagrame (obrázok 10). Diagram znázorňuje hlavné komponenty a ich vzájomné vzťahy. Ako môžete vidieť, balíček `Phenotype` sme oddelili od balíčku `Evolutionary Art` a podobne sme oddelili `Evolutionary Algorithm` od `Evolution`. Takto budeme môcť tieto časti zameniť a tak využiť viacero reprezentácií obrázkov a implementovať CGP aj RVCGP. Bližšie detaily a vzťahy môžete vidieť v diagrame tried (obrázok 12).

3.3.2 Návrh užívateľského rozhrania

Užívateľské rozhranie aplikácie by malo byť prehľadné a navrhnuté tak, aby väčšina rozhrania tvorila prezentáciu obrázkov aktuálnej populácie. Ovládacím prvkom aplikácie bude venovaná iba malá časť rozhrania a väčšina nastavení bude úplne oddelená do samostatných stránok.



Obrázek 11: Návrh užívateľského rozhrania.

Farebná schéma užívateľského rozhrania bude tmavšia, aby v aplikácii vygenerované obrázky vynikli.

3.4 Implementácia

Z predošlých častí máme dobrý základ a predstavu o tom, ako by mala aplikácia vyzerať a čo všetko by mala byť schopná vykonávať. Zostáva teda vybrať vhodnú platformu a vhodný programovací jazyk.

3.4.1 Voľba platformy a programovacieho jazyka

Aplikáciu sme sa rozhodli implementovať ako webovú aplikáciu za pomoci `HTML5`, `CSS3` a `JavaScript`-u. `HTML5` má veľmi dobrú podporu tvorby aj zobrazovania obrázkov vďaka elementu `Canvas` a veľmi rozmanité a ľahko vytvoriteľné uživateľské prostredie vďaka elementom HTML5 a podpore `CSS3` pre ich štýlovanie. Funkcionalitu a dynamickosť aplikácie zvládne skriptovací jazyk `JavaScript`.

Zhrnutie pozitív:

- Podpora na platformách podporujúcich novší webový prehliadač.
- Bez nutnosti inštalácie podporných programov alebo knižníc.

- Dostupnosť a distribúcia aplikácie pomocou internetu.
- Dôveryhodnosť aplikácie - implicitné obmedzenia.
- Dobrá podpora práce s obrázkami.
- Vysoký výpočtový výkon.
- Dynamické užívateľské prostredie.
- Dynamické, objektovo-orientované a prototypovo založené programovanie.

Nevýhody a obmedzenia implementácie webovej aplikácie:

- Nutnosť spúšťania aplikácie cez webový prehliadač.
- Čiastočné obmedzenie práce so súbormi.

3.4.2 Vývojové nástroje

Ako `HTML`, `JavaScript` a `CSS` editor sme zvolili `Notepad++`, ktorý je veľmi jednoduchý a vhodný pre všetky tri typy jazykov. `JavaScript` je automaticky kompilovaný a optimalizovaný pri jeho spúšťaní. A ako jeho kompiler a zároveň debugger sme si vybrali webový prehliadač `Google Chrome`, ktorý má zabudované výborné nástroje na debuggovanie a analýzu webových stránok.

3.4.3 Analýza implementácie

`HTML5` elementy budú tvoriť štruktúru užívateľského rozhrania a obsah webových stránok. Zoznam použitých `HTML` elementov:

- Div, P a Span elemet - kontajner dcérskych elementov, výstupy aplikácie.
- Input, Select a Option elemet - vstupy aplikácie.
- Canvas a Img - zobrazenie obrázkov.
- Script - funkcionalita aplikácie v javascript-e.

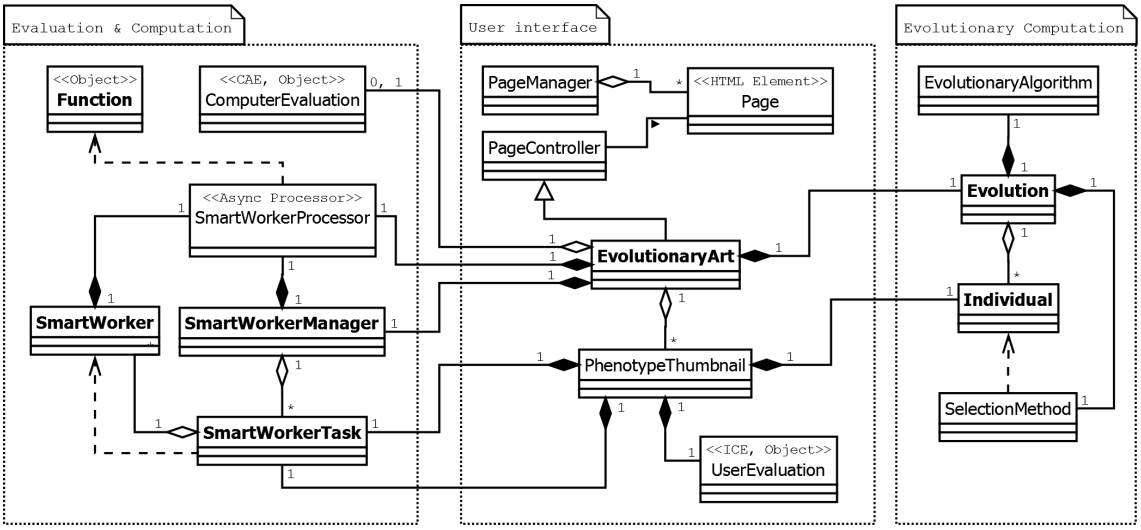
Funkcionalita a generovanie dynamického obsahu bude naskriptovaná v `JavaScript`-e. Použité `Web API`:

- CanvasRenderingContext2D - práca s obrázkami, získava sa metódou `Canvas` elementu `getContext('2d')`, pomocou neho môžeme kresliť na obrázok `Canvas`-u.
- ImageData - priama manipulácia s pixlami obrázku, získava sa metódou `getImageData` objektu `CanvasRenderingContext2D`.
- Worker - využitie viacero jadier CPU spustením samostatného `JavaScript` súboru.

- File, FileList, FileReader - načítanie obsahu súboru.
- TypedArray, Blob - práca s binárnymi dátami.
- Document - vytváranie dynamického `HTML` obsahu.
- URL - vytváranie `URL` objektov.
- Event - zachytávanie udalostí aplikácie.

3.4.4 Štruktúra aplikácie

Hlavným súborom aplikácie je `index.html`, ktorý obsahuje hlavnú štruktúru aplikácie a načítava externé obrázky, súbory `JavaScript`-u a súbory štýlov `CSS`.



Obrázek 12: Čiastočný diagram tried aplikácie.

Čiastočný diagram tried je na obrázku (obrázok 12) a kompletný diagram tried bude priložený na CD. Diagram je rozdelený do troch častí:

- **Vyhodnotenie a výpočty**: Tu prebieha vytváranie obrázkov a ich prípadné ohodnocovanie počítačom. Triedy `SmartWorkerManager`, `SmartWorkerTask`, `SmartWorker` a `SmartWorkerProcessor` majú za úlohu rozložiť výpočtový výkon do viacero jadier CPU pri vytváraní obrázkov. Úlohou triedy `Function` je vytvorenie spustiteľného programu jedinca reprezentovaného zdrojovým kódom. Vytvorenie zdrojového kódu (algoritnus 6). `SmartWorkerProcessor` reprezentuje `JavaScript` kód pre vytvorenie obrázku z programu jedinca a je spúšťaný v samostatnom vlákne. Program jedinca je inicializovaný pomocou triedy `SmartWorkerTask` a spúšťaný pomocou triedy `SmartWorker`. `SmartWorkerManager` spravuje tieto úlohy a priraďuje vlákna pre ich výpočet. Trieda `ComputerEvaluation` je abstraktná trieda umožňujúca rozšírenie EA o fitness funkciu.

- **Užívateľské rozhranie**: Trieda `PageManager` slúži na ovládanie a prepínanie stránok. Celé užívateľské rozhranie je postavené na architektonickom vzore `MVC`. Pohľad tvorí `HTML` element `Page`, model je dynamicky generovaný `Javascript`-om a radič tvorí `PageController`. Trieda `EvolutionaryArt` je hlavnou triedou evolúcie a zároveň jej pohľadom. Jedinci populácie sú v užívateľskom prostredí reprezentovaní abstraktnou triedou `PhenotypeThumbnail`.
- **Evolučné výpočty**: Hlavnou triedou je `Evolution`, ktorá implementuje evolučný cyklus a používa selekčnú metódu reprezentovanú abstraktnou triedou `SelectionMethod` a evlučný algoritmus reprezentovaný abstraktnou triedou `EvolutionaryAlgorithm`. Jedinci populácie sú reprezentovaní triedou `Individual`.

Abstraktná trieda	**Podtrieda**
EvolutionaryAlgorithm	CartesianGeneticProgramming RealValuedCartesianGeneticProgramming
SelectionMethod	TournamentSelection
PhenotypeThumbnail	PictureThumbnailLogic AnimationThumbnailLogic LinePainterThumbnailLogic SnakeThumbnailLogic
SmartWorkerProcessor	RGB256SmartWorkerProcessor RGBSmartWorkerProcessor RGBAnimationSmartWorkerProcessor RGBASmartWorkerProcessor LinePainterSmartWorkerProcessor SnakeAgentSmartWorkerProcessor
ComputerEvaluation	ImageComparison SnakeFitnessEvaluator
PageController	EvolutionPageControll ElitePageControll LaboratoryPageControll (a dalšie)

(13)

Tabulka 2: Tabuľka implementovaných podtried rozširujúcich abstraktné triedy.

Konkrétne implementácie podtried abstraktných tried sú vypísané v tabuľke (tabuľka 2).

Výhodou `Javascript`-u je kompilovanie zdrojového kódu za behu programu. Genotyp jedinca môžeme preto previesť pomocou genotyp-fenotyp mapovania do zdrojového kódu namiesto vytvárania grafovej štruktúry programu pomocou polí. S vytvoreným zdrojovým kódom programu v podobe textu môžeme ľahko manipulovať a kompiler `Javascript`-u môže naviac takýto zdrojový kód lepšie optimalizovať, čo by malo viesť k lepšiemu výkonu.

INPUT: $genotype$ of an individual.
OUTPUT: Source code of mapped program.

```
 1: function ConstructComputerProgram(genotype)
 2: {
 3:   var activeNodeMask = new Uint8Array(rows * columns + inputs);
 4:   for(var i = 0; i < activeNodeMask.length; i++)
 5:     activeNodeMask[i] = 0;
 6:   var nodeLinkIndex;
 7:   var j = genotype.length;
 8:   var body = ' ]; ' ;
 9:   for(var i = outputs - 1; i > 0; i--)
10:   {
11:     nodeLinkIndex = genotype[--j];
12:     activeNodeMask[nodeLinkIndex] = 1;
13:     body = ' , ' + VarNameAtIndex(nodeLinkIndex) + body;
14:   }
15:   nodeLinkIndex = genotype[--j];
16:   activeNodeMask[nodeLinkIndex] = 1;
17:   body = 'return [' + VarNameAtIndex(nodeLinkIndex) + body;
18:   for(var i = rows * columns - 1; i >= 0; i--)
19:   {
20:     if (activeNodeMask[i + inputs])
21:     {
22:       var functionCode = functionSet[genotype[--j]];
23:       for(var fi = 0; fi < numberOfFunctionNodeInputs; fi++)
24:       {
25:         nodeLinkIndex = genotype[--j];
26:         activeNodeMask[nodeLinkIndex] = 1;
27:         functionCode = functionCode.replace(new RegExp('\\\\textdollar' + ( fi + 1), 'g'),
                                                VarNameAtIndex(nodeLinkIndex));
28:       }
29:       body = 'var ' + VarNameAtIndex(inputs + i) + '=' + functionCode + body;
30:     }
31:     else
32:     {
33:       j = j - (1 + numberOfFunctionNodeInputs);
34:     }
35:   }
36:   var params = [];
37:   for(var i = 0; i < inputs; i++)
38:     params.push(VarNameAtIndex(i));
39:   return {params: params, body: body};
40: }
```

Výpis 6: Mapovanie genotypu do zdrojového kódu v CGP

Algoritmus pre vytvorenie zdrojového kódu programu jedinca pri použití CGP (algoritnus 6). Vytvorený zdrojový kód programu jedinca, ktorý má 7 aktívnych vrcholov, dva vstupy a tri výstupy sú vidieť vo výpise (výpis 7). Premenné `_A` a `_B` sú vstupy programu, premenné `_D`, `_E`, `_F`, `_J`, `_K`, `_N` a `_Q` nadobúdajú výstupné hodnoty aktívnych vrcholov a premenné `_Q`, `_J` , `_N` sú zároveň aj výstupmi programu.

```
function(_A,_B)
{
    var _D=(Math.sin(_B*Math.PI)+Math.sin(_B*Math.PI))/2;
    var _E=Math.abs((Math.sin(2*Math.PI*_A)+Math.cos(2*Math.PI*_B)))/2;
    var _F=Math.abs((Math.cos(2*Math.PI*_B)-Math.cos(2*Math.PI*_A)))/2;
    var _J=Math.sqrt((_F*_F+_F*_F)/2);
    var _K=(_D==0||_E==0)?0:((_D/_E)>1)?(_E/_D):(_D/_E);
    var _N=1-_F;
    var _Q=Math.min(_K,_K);
    return [_Q,_J,_N];
}
```

Výpis 7: Príklad vytvoreného zdrojového kódu

3.5 Popis aplikácie

V tejto časti práce si vysvetlíme hlavné časti a funkcie implementovanej aplikácie. Grafické rozhranie aplikácie je rozdelené do niekoľkých stránok. Hlavnú stránku webovej aplikácie, v ktorej prebieha evolúcia obrázkov, môžete vidieť na obrázku (obrázok 14).

3.5.1 Spustenie, vytvorenie a načítanie projektu aplikácie

Webová aplikácia je spustiteľná súborom `index.html` v novších verziách webových prehliadačov `Google Chrome`, `Mozilla Firefox`, `Opera` a `Internet Explorer 11` (poznámka 3.1).

Poznámka 3.1 Pri použití webového prehliadača `Internet Explorer 11` je nutné k webovej aplikácii pristupovať pomocou webového servera.

Pre nastavenie veľkosti grafického užívateľského rozhrania môžete použiť `zoom` zabudovaný v prehliadačoch. Minimálne rozlíšenie stránok pri štandardnom `zoom`-e je 980x300.

V hlavnej ponuke máte možnosti vytvorenia nového projektu a načítania predošle uloženého projektu aplikácie.

Pri načítaní uloženého projektu aplikácia prejde priamo na hlavnú stránku 3.5.3 evolúcie. Pri vytváraní nového projektu nasleduje pár stránok s rôznymi nastaveniami. Pri prvej stránke vyberáte problém a fenotyp. Druhá stránka obsahuje výber a nastavenia evolučného algoritmu 3.5.2. Tretia stránka obsahuje nastavenia genetických operátorov. Posledná stránka s nastaveniami obsahuje výber selekčnej metódy, jej nastavenia a distribúciu populácie.

3.5.2 Nastavenia evolučného algoritmu

Nastavenia evolučného algoritmu majú vlastnú stránku. Pri evolučnom algoritme CGP a RVCGP nastavenia obsahujú počet riadkov grafu, počet stĺpcov grafu a výber preddefinovaných funkcií. Sadu preddefinovaných funkcií, ako aj jednotlivé funkcie môžete

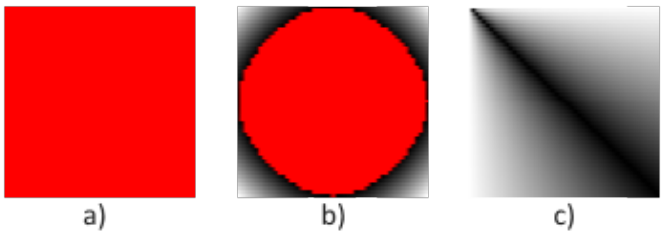
ľahko zmeniť. Každá funkcia má dostupné dva parametre `$1` a `$2` reprezentujúce vstupy funkcie a priradené 2D plátno, na ktorom sa znázorňuje efekt funkcie, ako môžeme vidieť na obrázku (obrázok 7). Na obrázku (obrázok 13) môžete vidieť efekty nasledovných funkcií pre rozmedzie hodnôt [0,1]:

$$\text{a)} \quad \$1 * /\$2 \tag{14}$$

$$\text{b)} \quad Math.sin(\$1 * Math.PI) + Math.sin(\$2 * Math.PI) \tag{15}$$

$$\text{c)} \quad (\$1 == 0||\$2 == 0)?0 : ((\$1/\$2) > 1)?(\$2/\$1) : (\$1/\$2) \tag{16}$$

Pri funkcii `a)` je plátno zafarbené na červeno, pretože funkcia obsahuje syntaktickú chybu. Pri funkcii `b)` je na plátne znázornený región, opäť červenou farbou, v ktorom funkcia vracia neplatné hodnoty. Funkcia `c)` je syntakticky správna a vracia validné hodnoty v rozmedzí [0,1].



Obrázek 13: Znázornenie efektu funkcie na 2D plátne.

3.5.3 Hlavná stránka evolúcie obrázkov

Stavový a ovládací panel evolúcie sa nachádza v dolnej časti stránky. Štítok `Generation` ukazuje aktuálnu generáciu jedincov populácie.

Tlačítkom `Workers` môžete zmeniť počet výpočtových vlákien. Prvá hodnota ukazuje počet aktívnych vlákien a druhá hodnota ukazuje počet dostupných vlákien. Ak zvolíme počet vlákien väčší ako počet jadier CPU, grafické užívateľské prostredie môže horšie reagovať.

Nastavenia evolúcie sú na samostatnej stránke dostupnej pomocou tlačítka `Settings`. Stránka `Settings` obsahuje nastavenia selekcie, distribúcie populácie a poskytuje výber automatickej vyhodnocovacej funkcie. Pre automatickú evolúciu obrázkov je dostupná jedna vyhodnocovacia funkcia `Image Comparison`, ktorá porovnáva vygenerované obrázky so vzorom na základe porovnávania farieb alebo jasu pixelov.

Najväčšiu časť hlavnej stránky tvorí aktuálna populácia. Jedinci populácie sú reprezentovaní ako orámované obrázky štvorcového tvaru. Pod každým jedincom je osobitný ovládací panel3.5.6 pre ohodnotenie, pridanie jedinca do stránky `Elite` 3.5.4, pridanie jedinca do stránky `Lab` 3.5.5 a uloženie genotypu, programu alebo obrázku jedinca.

Obrázek 14: Hlavná stránka pre evolúcie obrázkov.

3.5.4 Stránka Elite

Táto stránka slúži pre odkladanie jedincov, ktorých chce užívateľ zachovať. Táto populácia jedincov je statická a iba užívateľ môže pridávať či odoberať týchto jedincov. Pridávať jedincov môže kliknutím na hviezdičku v ovládacom paneli jedinca. Ak je jedinec v tejto populácii, hviezdička je vysvietená.

Zaškrkávacie políčko `Include to selection` umožní jedincom tejto populácie dostať sa do selekčného procesu evolúcie. Takto užívateľ získava kontrolu nad `Elitizmom` 2.5.4.

3.5.5 Stránka Lab

Táto stránka slúži na používanie genetických operátorov mimo evolučných cyklov. Jej cieľom je umožnenie užívateľovi priamu manipuláciu s jedincami. Novovytvorených jedincov tejto populácie síce nemôže priamo začleniť do evolúcie, ale môže ich pridať do stránky `Elite`. Pridávať a odoberať jedincov môžeme kliknutím na ikonku `DNA` v ovládacom paneli jedinca. Stránka populácie je rozdelená na dve časti, prvá časť reprezentuje rodičov pre genetické operátory a druhá časť reprezentuje potomkov. Genetické operátory mutácia a kríženie sú prístupné podľa počtu rodičov. Pre jedného rodiča je prístupná mutácia a pre dvoch kríženie.

3.5.6 Ovládací panel jedinca

Ovládací panel jedinca pozostáva zo štyroch tlačítok.

Prvé tlačítko reprezentuje ohodnotenie obrázku. Ohodnotenie je veľmi jednoduché a môže nadobudnúť tri stavy, `+1`, `+2` a `+3`.

Druhé tlačítko v tvare hviezdičky reprezentuje prítomnosť jedinca na stránke `Elite` 3.5.4.

Obrázek 15: Ovládací panel jedinca.

Tretie tlačítko s obrázkom `DNA` reprezentuje prítomnosť jedinca na stránke `Lab` 3.5.5.

Posledné tlačítko slúži na uloženie jedinca vo forme genotypu, počítačového programu alebo obrázku.

3.6 Test implementovanej aplikácie

Všetky testy sme uskutočnili na tom istom zariadení a implementovanú webovú aplikáciu sme spúšťali pomocou desktopovej verzie `Google Chromu` 41.0.2272.101 m.

Špecifikácia testovacieho zariadenia:

- Typ zariadenia: Stolný počítač
- CPU: AMD Athlon(tm) ll X4 630 Processor, 2.80 GHz, x64
- RAM: 4.00 GB
- OS: Windows 8.1 Pro, 64-bit
- Rozlíšenie obrazu: 1680x1050

Základné nastavenia webovej aplikácie pri testoch:

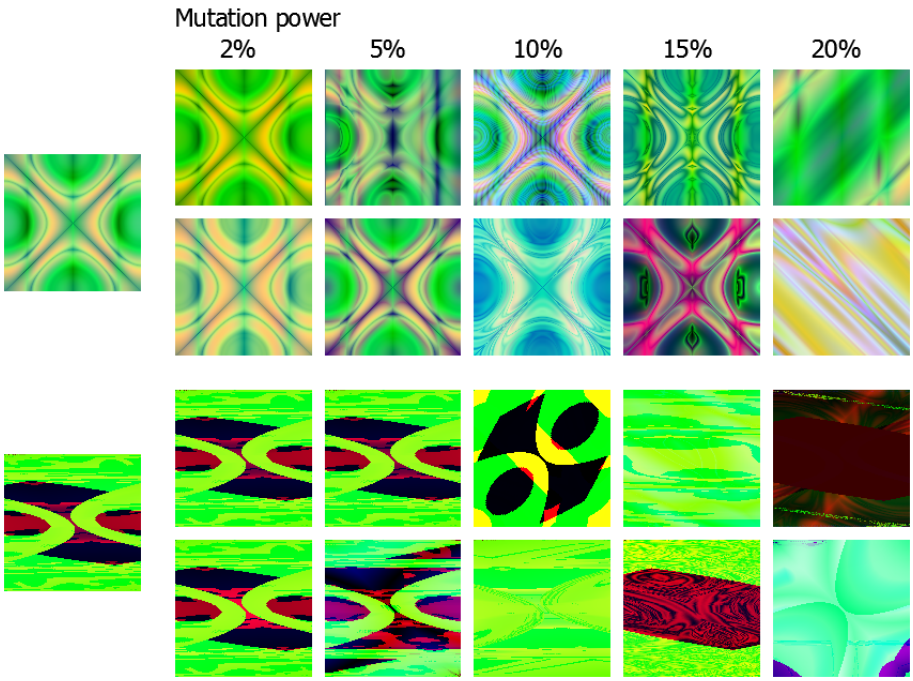
- Evolučný algoritmus CGP. Počet riadkov mriežky grafu 1, počet stĺpcov mriežky grafu je 300 a použili sme sadu funkcií definovanú v tabuľke (tabuľka 1).
- Počet `Workerov` je nastavený na hodnotu 3.
- Typ obrázku `Picture RGB (normalized)`.
- Veľkosť populácie je nastavená na 10 jedincov.

- Sila mutácie je obmedzená na jeden až 15% génov genotypu.
- V testoch používam `Tournament selection` o veľkosti turnamentu 3.
- Distribúcia populácie je nastavená pre prežitie 0 jedincov, variáciu 4 jedincov, rekombináciu 3 a náhodné vytvorenie 3 jedincov.

3.6.1 Test č. 1 - Test genetického operátoru mutácie

Cieľom testu je zachytiť rozdielne efekty mutácie so zvyšujúcou sa silou mutácie. Pri tomto teste sme použili genetický operátor mutácie so silou mutácie 1% až 30%.

Mutácie so silou do 2% poväčšine nemali žiadny alebo iba veľmi malý efekt na obrázok jedinca. Pri sile mutácie 5% sa efekt prejavoval skoro u všetkých jedincov a občas sa už začala prejavovať malá deformácia od pôvodného obrázku. Od 5% až po 20% bol efekt mutácie pomerne lineárny a mutácie so silou nad 20% mali už poväčšine iba deštrukčný efekt a väčšinu pôvodných vlastností nový obrázok stratil.



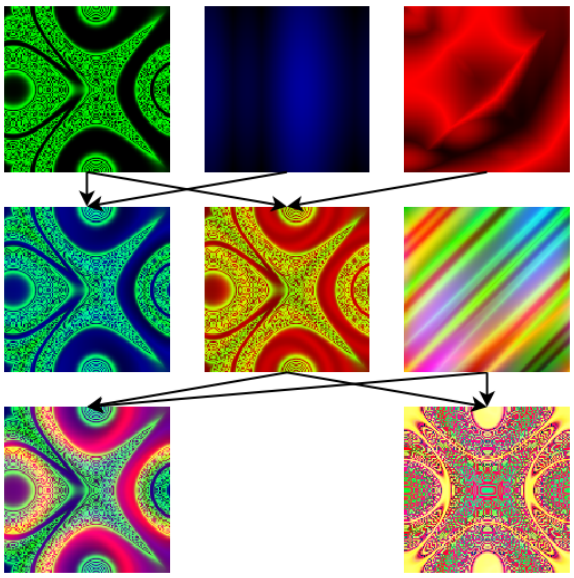
Obrázek 16: Výber obrázkov z testu č. 1.

Na obrázku (obrázok 16) môžeme vidieť efekt genetického operátora mutácie pri sile 2%, 5%, 10%, 15% a 20% u dvoch rôznych jedincov. So zvyšujúcou sa silou mutácie môžete spozorovať väčšie rozdiely až úplné deformácie od pôvodných obrázkov.

3.6.2 Test č. 2 - Test genetického operátoru kríženia

Tento test mal zachytiť efekt genetického operátoru kríženia. Pri tomto teste sme použili metódu kríženia `Two Point Crossover`.

Pri krížení by sme chceli spozorovať kombináciu vlastností obrázkov. V teste sme zistili, že väčšina vygenerovaných obrázkov pomocou kríženia nemala očakávané vlastnosti. Poväčšine mal potomok vlastnosti zdedené iba od jedného rodiča alebo mal potomok úplne odlišné vlastnosti od oboch rodičov.



Obrázek 17: Výber obrázkov z testu č. 2.

Aj keď potomkovia nemali očakávané vlastnosti, často prekvapili novým tvarom a farbami.

Pri niektorých obrázkoch a pri väčšom množstve pokusov sa občas podarilo skombinovať niektoré vlastnosti. Hlavne sme mohli spozorovať dedenie farieb alebo tvaru obrázku. Výber niektorých obrázkov z tohto testu môžete vidieť na obrázku (obrázok 17).

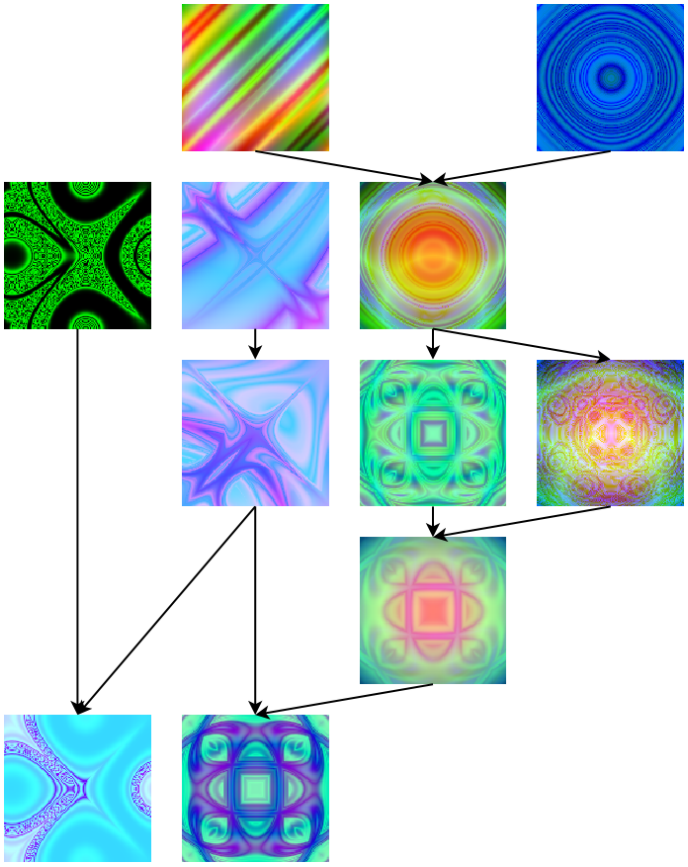
3.6.3 Test č. 3 - Porovnanie CGP a RVCGP

V tomto teste sa pokúsime porovnať generované obrázky pomocou CGP a RVCGP. U RVCGP taktiež vyskúšame novú metódu kríženia `floating-point crossover` a porovnáme ju s výsledkami v teste č. 2 (test 3.6.2).

Pri tomto teste sme nepostrehli žiadne rozdiely medzi vygenerovanými obrázkami pomocou CGP a RVCGP. Efekt genetického operátora kríženia `floating-point crossover` na obrázkoch bol podobný, ako pri teste č. 2 (test 3.6.2).

3.6.4 Test č. 4 - Evolúcia obrázkov

Cieľom tohto testu je otestovať vývoj obrázkov pri evolúcii. V tomto teste budeme hodnotiť obrázky jednotlivých generácií o veľkosti 15-tich jedincov. Každú generáciu bude tvoriť rovnomerne 5 náhodne vygenerovaných jedincov, 5 jedincov vytvorených pomocou mutácie a 5 jedincov vytvorených pomocou kríženia. Pri testovaní bude použitá selekčná metóda `Tournament Selection` s veľkosťou turnamentu 3.



Obrázek 18: Výber obrázkov z testu č. 4.

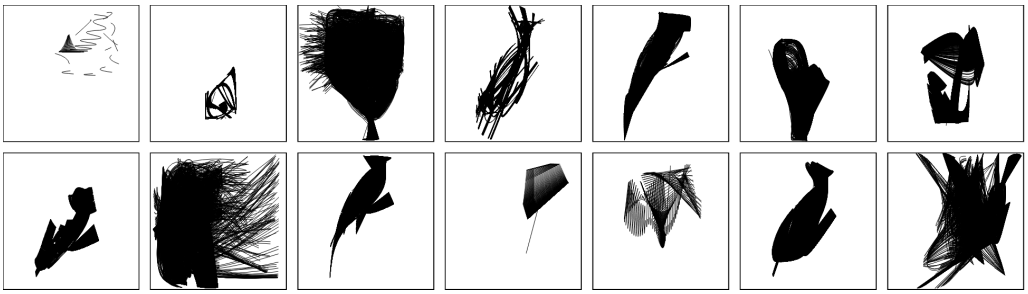
Evolúciu sme vyvíjali po 200 generácií a pri teste sme postrehli, že niektoré vygenerované obrázky mali dobré estetické vlastnosti už v prvých generáciách. Pri neskorších generáciách sa objavovali aj variácie predošlých obrázkov, ktoré mali poväčšine podobnú estetickú hodnotu. Estetická hodnota obrázkov bola veľmi podobná po celú dobu evolúcie a nejaký veľký vývoj estetickej hodnoty v neskorších generáciách sme nepostrehli.

Výber obrázkov z testu môžete vidieť na obrázku (obrázok 18).

3.6.5 Test č. 5 - Evolúcia obrázkov pomocou iterujúcej funkcie

Cieľom tohto testu je otestovať vývoj obrázkov pri evolúcii podobne ako u predošlého testu (test 3.6.4). Použijeme rovnaké nastavenia a vytváranie obrázku bude pomocou iterujúcej funkcie. V aplikácii je táto reprezentácia pomenovaná `Line Painter`.

Pri tomto teste použijeme počiatočnú vstupnú hodnotu iterujúcej funkcie 0 a na vytvorenie obrázku použijeme výsledky z 1000 iterácií funkcie v podobe karteziánskych súradníc zakrivených čiar a informácie či má byť čiara zakreslená.



Obrázek 19: Výber obrázkov z testu č. 5.

Pri tomto teste, narozdiel od testu (test 3.6.4), boli prvé generácie jedincov veľmi neuspokojivé. Väčšina obrázkov bolo prázdnych a iba raz za pár generácií sa objavila machuľa, ktorej sme priradil dobré hodnotenie. Po pár generáciách sa začali generovať lepšie obrázky a už neskoršie generácie prebiehali podobne ako pri teste č. 4.

Niekoľko vygenerovaných obrázkov z testu môžete vidieť na obrázku (obrázok 19). Niektoré vygenerované funkcie mali tendenciu skonvergovať do jedného bodu, niektoré zasa obiehali v cykloch a niektoré sa chovali chaoticky a mali tendenciu generovať unikátne súradnice vo všetkých iteráciách.

3.6.6 Test č. 6 - Evolúcuia animácie

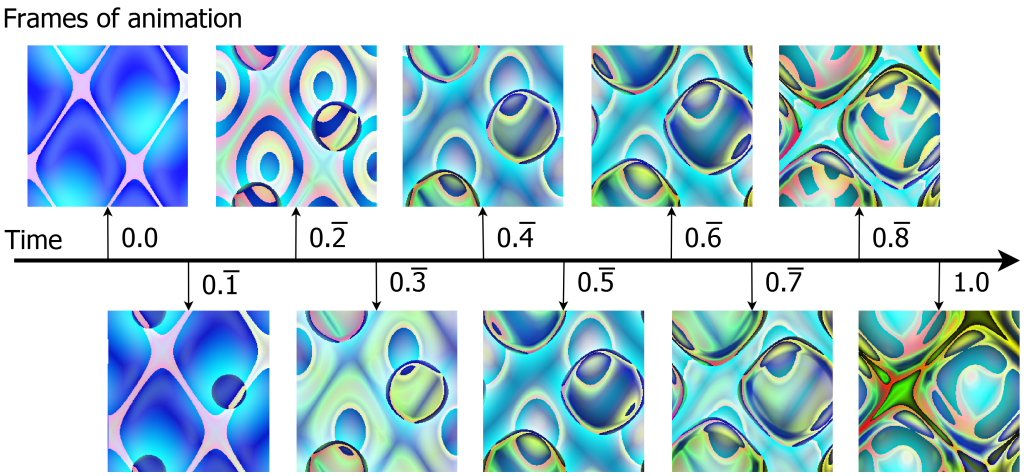
Pri tomto teste používame nastavenia a reprezentáciu obrázkov z testu č. 4 rozšírenú o tretiu dimenziu reprezentujúcu čas. Animáciu tvoríme sekvenciou obrázkov, ktoré sú postupne zamieňané v závislosti od času (obrázok 20).

V teste budeme vyvíjať animácie o desiatich obrázkoch a vstupná hodnota času pre jednotlivé obrázky číslovené od nuly bude definovaná rovninou:

$$time = frame\_number/(number\_of\_frames - 1) \tag{17}$$

Obrázky jednej animácie z tohto testu môžete vidieť na obrázku (obrázok 20). Zmeny obrázkov animácie v závislosti od času sú rôzne. Často sme spozorovali rôzne zmeny v tvare alebo farbách v rôznych úsekoch animácie. Veľkosť zmeny medzi susednými obrázkami animácie bola taktiež často nelineárna. To môžeme vidieť aj pri animácii na

obrázku (obrázok 20), kde môžeme najväčšiu zmenu spozorovať medzi druhým a tretím obrázkom animácie resp. hodnotami času $0.\overline{1}$ a $0.2\overline{1}$.



Obrázek 20: Obrázky animácie z testu č. 6.

Zvýšením počtu obrázkov alebo ohraničením časového úseku animácie môžeme docieliť plynulejšie prechody obrázkov a tak vytvárať dlhšie, či kvalitnejšie animácie.

3.6.7 Test č. 7 - Automatická evolúcia obrázkov

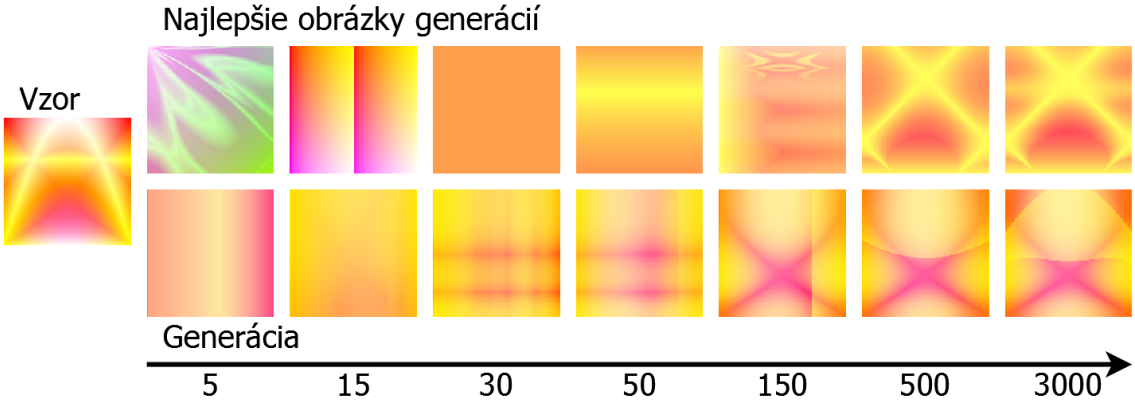
V tomto teste sa budeme snažiť o evolúciu obrázkov k definovanému cieľu. Obrázky budú automaticky ohodnocované funkciou `Image Comparison`, ktorá porovnáva farby pixelov so vzorovým obrázkom.

Cieľom tohto testu je otestovať nielen schopnosť evolučného algoritmu optimalizovať perfektnú repliku od vzorového obrázku, ale taktiež schopnosť generovať podobné obrázky s potenciálne lepšími estetickými vlastnosťami.

Obrázek 21: Porovnanie ohodnotenia obrázkov funkciou `Image Comparison`.

Ohodnocovacia funkcia na základe porovnávania farieb pixelov obrázkov použitá v tomto teste javí značné nedostatky a evolúcia väčšinou veľmi rýchlo skonverguje do

obrázkov s dominantnou farbou vo vzore. Tvar obrázku je pritom viacmenej ignorovaný. Na obrázku (obrázok 21) môžete vidieť porovnanie ohodnotenia tejto funkcie na obrázku s podobnými farbami a obrázku s podobným tvarom.



Obrázek 22: Priebeh dvoch evolúcií pri teste č. 7.

V teste sme ako vzor použili predošle vygenerovaný obrázok aplikáciou, aby sme mali istotu, že je aplikácia schopná takýto obrázok vygenerovať. Evolúciu sme spustili niekoľkokrát pri rovnakých nastaveniach a priebeh dvoch môžete vidieť na obrázku (obrázok 22). Pri všetkých pokusoch evolúcia veľmi rýchlo, poväčšine do 30-tej generácie, skonvergovala do jednoduchého obrázku s dominantnou oranžovo-žltou farbou. Medzi 150-tou a 500-tou generáciou obrázky nadobúdali finálnu podobu a v ďalších generáciách sa prejavovali už iba veľmi malé zmeny. Zaujímavosťou bolo, že pri každom pokuse boli výsledné obrázky podobne komplexné, ale celkovo značne rozdielne.

Pri evolúcii obrázkov s rôznymi vzormi sme postrehli, že so zvyšujúcou sa komplexitou vzorového obrázku sa výsledky evolúcie zhoršovali a pri veľmi komplexných obrázkoch sa evolúcia zasekla už v prvých generáciách.

Aj keď boli výsledky tohto testu z pohľadu optimizačnej metódy neuspokojivé, v evolučnom umení by mohla byť aj takáto metóda celkom užitočná. Bližšie preskúmanie a vylepšenie takejto metódy by mohol mať prínos ako do evolučného umenia, tak aj do iných oblastí. Ako optimizačná metóda by mohla nájsť uplatnenie v kryptografii alebo pri kompresii obrázkov.

Zlepšenie výsledkov optimizácie by sme mohli docieliť hlavne lepšou ohodnocovacou funkciou. Ďalší vývoj ohodnocovacej funkcie by mohol spočívať v implementácii rôznych analýz obrázkov ako napríklad detekcie hrán, segmentácie obrázkov a pod. Druhým nedostatkom, ktorému by sa oplatilo bližšie venovať je nedostatočné udržiavanie divergencie populácie, pretože najlepší obrázok poväčšine úplne potlačil ostatné druhy obrázkov jeho malými mutáciami.

3.6.8 Test č. 8 - Evolúcia agenta

V tomto teste aplikáciu použijeme na optimalizáciu netriviálneho problému inšpirovaného hrou `Snake`. V probléme `Snake Agent` je riešenie reprezentované ako `agent`, ktorého bude evolučný algoritmus optimalizovať.

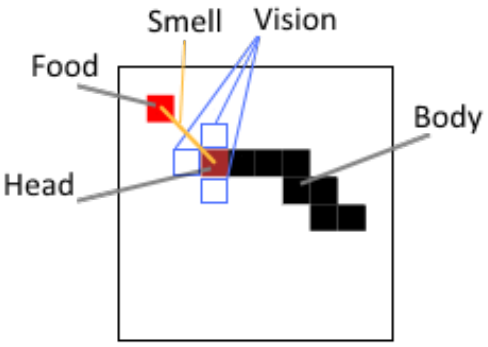
Úlohou hadíka je pozbierať čo najviac jedla, pričom nesmie naraziť do svojho tela. Naviac je počet ťahov hadíka obmedzený energiou. Energia sa pri zobraní jedla obnoví podľa vzorcu:

$$energy = 20 + snake\_length * 10 \tag{18}$$

Vyvíjaný program má päť vstupov, ktoré reprezentujú vnemy hadíka (vzorce 19). Hadík vidí do políčka pred, vľavo a vpravo od hlavy a cíti aktuálnu a predošlú silu vône v závislosti od jeho vzdialenosti od jedla.

$$\begin{aligned} vision &= \begin{cases} 0.0 & \text{if food} \\ 0.5 & \text{if empty} \\ 1.0 & \text{if snake body} \end{cases} \\ smell_{previous} &= smell_{current} \\ smell_{current} &= distance\_between(head, food)/max\_distance \end{aligned} \tag{19}$$

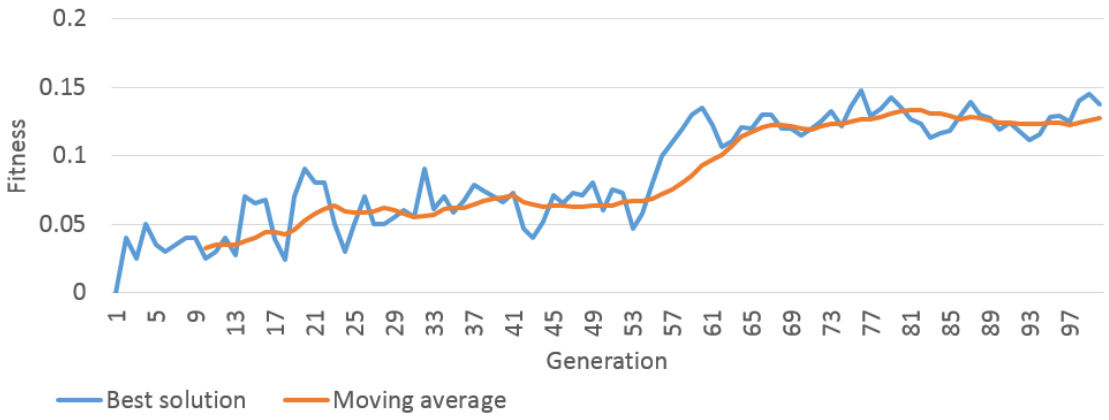
Tri výstupy programu reprezentujú vôlu hadíka ísť rovno, zabočiť doľava a zabočiť doprava. Najväšia hodnota z týchto výstupov rozhoduje o príslušnej akcii hadíka.



Obrázek 23: Ukážka hadíka a jeho zmyslov.

Značným problémom pri evolúcii je náhodné generovanie počiatočnej pozície hadíka ako aj náhodné generovanie pozície jedla. To znamená, že niektorí jedinci môžu byť zvýhodnení, čo vedie k náhodným situáciám, kedy dostane zlé riešenie lepšie hodnotenie ako dobré riešenie a naopak.

Hodnotu `fitness` preto počítame z priemernej hodnoty dvoch pokusov, čo aspoň z časti zredukuje veľmi velké rozdiely v hodnotení. V grafe (obrázok 24) môžete vidieť značné kolísanie hodnoty `fitness` najlepších riešení, spôsobené týmto problémom.



Obrázek 24: Priebeh evolúcie prvých generácii hadíka v teste č. 8.

V teste sa správanie hadíka vyvíjalo postupne v niekoľkých etapách:

- Prvá etapa - Prvé generácie hadík jedlo poväčšine ignoroval a často šiel stále jedným smerom, až kým sa mu neminie energia.
- Druhá etapa - Hadík má tendenciu pohybovať sa chaoticky, pričom dochádza ku náhodnému kontaktu s jedlom.
- Tretia etapa - Hadík už javí známky vyhľadávania jedla, ale väčšinou k jedlu pristupuje iba z jednej strany, čo vedie k častému narazeniu do vlastného tela.
- Štvrtá etapa - Hadík smeruje priamo najkratšou cestou k jedlu. Ak je v tejto fáze hadík dlhší, skoro vždy naráža do svojho chvostu.
- Piata etapa - Hadík sa správa podobne ako pri predošlej etape a naviac sa vyhýba svojmu telu. Často ale tvorí slepé uličky alebo si zablokuje cestu k jedlu a minie sa mu energia.

Po mnoho ďalších generácií hadík nejavil známky ďalšieho vývoja a výsledky boli viacmenej rovnaké. Až po približne 3000 generáciách sa objavilo nové a úplne odlišné správanie. Hadík sa k jedlu ponáhľal iba zozačiatku a pristupoval k nemu iba z jednej strany. Pri väčšej dĺžke začal obchádzať svoje telo, čo úplne eliminovalo zablokovanie cesty k jedlu a čiastočne zredukovalo tvorbu slepých uličiek.

4 Záver

Evolučné algoritmy boli pôvodne vyvinuté pre riešenie problémov, kde plnili úlohu optimizátora. V evolučnom umení si však našli aj inú úlohu a to úlohu generátora umenia. Náhodné generovanie jedincov poskytuje veľkú rozmanitosť riešení, genetický operátor mutácie poskytuje možnosť ich menšej, či väčšej variácie a genetický operátor kríženia kombináciu vlastností.

Aj keď je evolučné umenie aktívnou praktikou už niekoľko rokov, stále čelí problémom s estetickým ohodnocovaním. Doposiaľ použité automatické metódy identifikovania a ohodnotenia estetickej hodnoty v evolučnom umení sú nepostačujúce a najlepšou alternatívou je stále interaktívne ohodnocovanie užívateľom.

Vhodnosť jednotlivých foriem evolučných algoritmov v evolučnom umení je veľmi individuálna a z väčšej časti záleží od požadovaného artistického cieľu. Pre dizajn sú všeobecne výhodnejšie genetické algoritmy, kde je riešenie reprezentované formou parametrov a pre generovanie obrázkov sú výhodnejšie evolučné algoritmy, ktoré reprezentujú riešenia formou počítačového programu. Výhodami karteziánskeho genetického programovania pri použití v evolučnom umení oproti ostatným evolučným technikám je veľká neutrálnosť genotypu, podpora a použitie genetických operátorov mutácie a kríženia priamo na genotype a reprezentácia programu, ktorá poskytuje využitie ľubovoľného počtu vstupov a výstupov.

Implementovaná aplikácia nám poskytla dobré prostredie na prevedenie rôznych testov na viacero typoch vizuálneho umenia. V prvých dvoch testoch sme zachytávali rôzne efekty genetických operátorov mutácie a kríženia. Mutácia mala očakávaný efekt a výsledky boli nadmieru uspokojivé. Pri krížení sa požadovaná kombinácia vlastností prejavovala zriedka, avšak výsledné obrázky často prekvapili unikátnymi vlastnosťami a estetickou hodnotou. V teste č. 3 sme porovnávali obrázky generované pomocou dvoch evolučných algoritmov CGP a RVCGP, pri ktorom sme testovali aj novú metódu kríženia `floating point crossover`. Výsledky boli veľmi podobné, z čoho sme usúdili, že efektivita optimizácie na základe estetických hodnôt nehrá veľkú úlohu. To sme z časti zachytili aj v teste č. 4, 5 a 6, kde sme testovali priebeh samotnej evolúcie. Pri teste č. 7 a 8 sme sa pokúšali vyvíjať jedincov pomocou evolúcie s automatickým ohodnocovaním. V testoch išlo o vývoj obrázkov k zadanému vzoru a o vývoj umelej inteligencie, kde sme hľadali estetickú hodnotou v samotnom správaní sa jedincov. Aj keď boli implementované ohodnocovacie funkcie v oboch testoch značne obmedzené, čo sa podstatne odrazilo na efektivite a schopnosti optimalizácie problémov, dosiahnuté výsledky sú zaujímavé a určite si zaslúžia zvýšenú pozornosť.

Celkovo boli výsledky práce uspokojivé a poskytli rôzne pohľady a možnosti pre využitie evolučných algoritmov v evolučnom umení.

Ďalší vývoj aplikácie by mohol nabrať niekoľko smerov. Aplikácia by sa mohla rozšíriť o `WebGL`, ktoré by sprístupnilo lepšie grafické nástroje a umožnilo modelovanie 3D objektov a scén. Interaktívne ohodnocovanie by mohlo byť rozšírené o kolektívne ohodnocovanie prostredníctvom internetu. Pri automatickej evolúcii by bolo vhodné venovať väčšiu pozornosť ohodnocovacím funkciám a poprípade zhodnotiť implementáciu ohodnocovacej funkcie založenej na ohodnocovaní estetickej hodnoty.

5 Referencie

[1] Julian F. Miller (Eds.): *Cartesian Genetic Programming*, Springer Berlin Heidelberg 2011, ISBN 978-3-642-17310-3.

[2] Janet Clegg, J. A. Walker, J. F. Miller: *A New Crossover Technique for Cartesian Genetic Programming*, New York, NY, USA, 2007, 978-1-59593-697-4.

[3] Thomas Back, David B. Fogel, Z. Michalewicz (Eds.): *Handbook of Evolutionary Computation*, Oxford University Press; Lslf edition, Apríl 17, 1997, ISBN 978-0750303927, The Background, Kenneth De Joung, et al.: *A history of evolutionary computation*

[4] Cecilia Di Chio, Anthony Brabazon et al.: *Applications of Evolutionary Computation*, Springer Berlin Heidelberg, 2010, ISBN 978-3-642-12241-5, Philip Galanter: *The Problem with Evolutionary Art Is...*, pages 321-330.

[5] Antoni Hoffman: *Arguments on Evolution: A Paleontologist's Perspective*, New York: Oxford University Press 1989, ISBN 978-0195044430.

[6] Juan Romero, Penousal Machado (Eds.): *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*, Springer 2008, Natural Computing Series, ISBN 978-3-540-72876-4, Part I., Matthew Lewis: *Evolutionary Visual Art and Design*, pages 3-37.

[7] C. Di Chio, A. Brabazon, et al. (Eds.): *Applications of Evolutionary Computation*, Springer Berlin Heidelberg 2010, Lecture Notes in Computer Science, ISBN 978-3-642-12242-2.

[8] School of Computer Science, The Carnegie Mellon University, *AI-related FAQs.*, Genetic Algorithms FAQ. (online) 6. okt. 2014, <`http://www.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/html/faqs/ai/genetic/top.html`>.

[9] Marián Mach: *Evolučné algoritmy: Prvky a princípy*, Košice : Elfa s.r.o, 2009, ISBN 978-80-8086-123-0.

[10] GEATbx, www.geatbx.com, *GEATbx: Genetic and Evolutionary Algorithm Toolbox for use with MATLAB Documentation*, (online) 10. okt. 2014, <`http://www.geatbx.com/docu/algindex.html`>.

[11] Jyh-Rong Chou: *Applying Fuzzy Linguistic Preferences to Kansei Evaluation*, KEER2014. Proceedings of the 5th Kanesi Engineering & Emotion Research; International Conference; Linköping; Sweden; Jún 11-13, 2014, ISBN 978-91-7519-276-5.

[12] Vivek Sharma, Dr Parveen Kalra, Arvind Kumar: *Customer Perception Assessment Technique Kansei Engineering: A Review*, International Journal of Scientific Research; Júl, 2013, ISSN 2277-8179.

[13] Wolfram, mathworld.wolfram.com, *Wolfram Mathworld*, (online) 2. 2. 2015, <`http://mathworld.wolfram.com/`>.

A Príloha na CD

Obsah CD
Priložené CD obsahuje zdrojové súbory implementovanej aplikácie a diagram tried.