UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

# EVOLÚCIA MORFOLÓGIE

Bakalárska práca

2017 Tamás Bilek

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

# EVOLÚCIA MORFOLÓGIE

Bakalárska práca

| | |
|---|---|
| Študijný program: | Aplikovaná informatika |
| Študijný odbor: | 2511 Aplikovaná informatika |
| Školiace pracovisko: | Katedra aplikovanej informatiky |
| Školiteľ: | Mgr. Pavel Petrovič, PhD. |

Bratislava 2017 Tamás Bilek


Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

# ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Tamás Bilek
**Študijný program:** aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
**Študijný odbor:** aplikovaná informatika
**Typ záverečnej práce:** bakalárska
**Jazyk záverečnej práce:** slovenský
**Sekundárny jazyk:** anglický

**Názov:** Evolúcia morfológie
*Evolving morphologies*

**Cieľ:** Evolučné algoritmy inšpirované procesmi prirodzenej evolúcie v prírode sú bežne zaužívaná optimalizačná stochastická metóda. Okrem iného sa dajú použiť na evolúciu tvaru tela (morfológiu) nejakého umelého tvora poháňaného motormi. V tejto práci budú vytvorené umelé tvory, ktoré budú plniť stanovenú úlohu v simulácii a napokon budú na 3D tlačiarni reálne vytvorené a ich správanie odmerané v praxi a porovnané s výsledkami v simulácii.

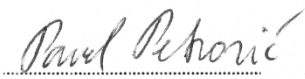
**Literatúra:** Karl Sims: Evolving Virtual Creatures, SIGGRAPH 94, dostupné online: archive.org/details/sims_evolved_virtual_creatures_1994
OpenSCAD User Manual, dostupné online: en.wikibooks.org/wiki/OpenSCAD_User_Manual
Open Dynamics Engine Manual, dostupné online: ode-wiki.org/wiki/index.php?title=Manual

**Vedúci:** Mgr. Pavel Petrovič, PhD.
**Katedra:** FMFI.KAI - Katedra aplikovanej informatiky
**Vedúci katedry:** prof. Ing. Igor Farkaš, Dr.

**Dátum zadania:** 03.10.2016

**Dátum schválenia:** 17.10.2016 doc. RNDr. Damas Gruska, PhD.
garant študijného programu

študent



vedúci práce

# Čestné prehlásenie

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Tamás Bilek

# Poďakovanie

Chcel by som sa poďakovať svojmu školiteľovi Mgr. Pavlovi Petrovičovi, PhD. za cennú pomoc, trpezlivosť, pozitívny prístup a dôležité rady, ktoré mi umožnili vypracovať bakalársku prácu.

# Abstrakt

BILEK, Tamás: *Evolúcia morfológie (Bakalárska práca)* – Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky; Katedra aplikovanej informatiky. – Školiteľ: Mgr. Pavel Petrovič, PhD.: FMFI UK, 2017

Evolučné algoritmy majú viacero využití. V tejto bakalárskej práci boli aplikované na virtuálne tvory. Po vyvinutí úspešného jedinca pomocou naprogramovaného softvéru bol jedinec reálne vytvorený na 3D tlačiarni. Tento model bol osadený servomotormi, ktoré boli pripojené na vývojovú dosku Arduino Nano. Nakoniec bolo porovnané správanie vytlačeného jedinca so simuláciou.

**Kľúčové slová:** evolučné algoritmy, virtuálne tvory, 3D tlač

# Abstract

BILEK, Tamás: *Evolving morphologies (Bachelor thesis)* – Comenius University in Bratislava, Faculty of Mathematics, Physics and Informatics; Department of Applied Informatics – Supervisor: Mgr. Pavel Petrovič, PhD.: FMFI UK, 2017

Evolutionary algorithms can be used for several purposes. In this bachelor thesis, they were applied on virtual creatures. After evolving a successful individual using my software, its model was created on a 3D printer. Afterwards, it was assembled with servomotors, which were connected to a development board called Arduino Nano. Finally, the behavior of the printed individual was compared to the simulation.

**Keywords:** evolutionary algorithms, virtual creatures, 3D printing

# Obsah

# 1 Úvod

Evolučné algoritmy sú dôkazom toho, že sa informatika môže inšpirovať prírodou. Pomocou evolúcie vzniklo celé dnešné ľudstvo. Javy, ktoré sa odohrávajú počas tohto procesu, môžu však byť využité softvérom, ako algoritmus. Týmto spôsobom sme schopní prinútiť náš počítač, aby napríklad vyvíjal jednoduché binárne reťazce, ale aj virtuálne tvory, ktoré plnia rozličné úlohy. To znamená, že nie je nutné, aby sme týchto jedincov priamo vytvorili, stačí napísať algoritmus, ktorý to urobí za nás v neobmedzenom počte.

Evolučné algoritmy majú veľa rôznych využití, ktoré môžu byť pre vývoj technológie užitočné. Preto je aj možné, že výrazne zmenia budúcnosť ľudstva, zjednodušia nám životy a počítače budú schopné vykonávať ešte viac práce namiesto ľudí, ako v dnešnej dobe. Pomerne jednoduché softvéry nám umožnia vytvoriť oveľa zložitejšie kódy rýchlejšie, ako by ich priamo vytvárali ľudia. Počítače sa budú vyvíjať podobným spôsobom, ako sa vyvíja život na Zemi, ale ideálne výsledky budú dosiahnuté za výrazne kratší čas. Tým, že sa evolučné algoritmy budú taktiež vyvíjať, aj samotný proces virtuálnej evolúcie bude čoraz efektívnejší a bude prinášať lepšie a lepšie výsledky.

Mojou bakalárskou prácou otestujem, ako sa správajú virtuálni jedinci reálne vytvorení na 3D tlačiarni, ktorí dosiahli priaznivé výsledky v simulácii. Týmto zistím, nakoľko je môj evolučný algoritmus ideálny na navrhovanie tvorov, ktoré sú schopné plniť rozličné úlohy aj v reálnom svete. Tieto experimenty nás môžu potenciálne posunúť ďalej napríklad vo vytváraní rozličných strojov pomocou softvéru, ktoré pre nás môžu byť užitočné, bez toho, aby ich manuálne navrhovali ľudia.

# 2 Východiská

## 2.1 Cieľ

Cieľom práce je vytvoriť virtuálneho tvora schopného splniť nejakú jednoduchú úlohu pomocou evolučných algoritmov a po reálnom vytvorení na 3D tlačiarni porovnať výsledky vo virtuálnom a fyzickom svete.

## 2.2 Prehľad problematiky

### 2.2.1 Evolučné algoritmy

Evolučné algoritmy sú inšpirované evolúciou živých organizmov v prírode. Používajú javy ako napríklad kríženie a mutácia morfológie a kontroleru genotypov alebo výber vhodných jedincov do ďalšej populácie. [1]

Takéto algoritmy budú použité na vyvinutie virtuálneho tvora počas viacerých generácií opakovaným aplikovaním javov evolúcie.

### 2.2.2 Reprezentácia morfológie virtuálnych tvorov

Na reprezentáciu morfológie je možné využiť viacero spôsobov, ako napríklad orientovaný graf [2], všeobecný strom alebo jednoduchý textový reťazec [3], ktorý je možné explicitne preformátovať na strom.

### 2.2.3 Reprezentácia kontroleru virtuálnych tvorov

Na reprezentáciu kontroleru je možné použiť napríklad jednoduchú postupnosť pohybov pre každú končatinu alebo ak má tvor dynamicky vnímať prostredie, použije sa funkcia, ktorá vypočíta vstupné hodnoty pre efektor z výstupnej hodnoty senzorov, čo znamená, že po spracovaní informácií získaných z prostredia odhadne vhodný pohyb, ktorý mu pomôže vo vykonaní svojej zadanej úlohy.

### 2.2.4 Evolúcia virtuálnych tvorov

Na začiatku sa vytvorí počiatočná populácia pozostávajúca z náhodne vytvorených tvorov daného počtu. Následne je každý jedinec ohodnotený pomocou účelovej funkcie, ktorá určuje vhodnosť jedinca na vykonanie požadovanej úlohy. Kombináciou týchto výsledkov a náhodného vygenerovania podľa šance sa vyberú tvory pripravené na ďalšie fázy evolúcie, ktorými sú kríženie a mutácia.

Počas kríženia sa vyberú náhodné časti dvoch jedincov, ktoré sa spoja a tak vznikne nový, potenciálne lepší jedinec. Tento tvor potom prejde cez mutáciu morfológie, napríklad pridanie, odobranie alebo zmenu veľkosti končatín a kontroleru, čo spôsobí zmenu v správaní tvora.

Po dokončení týchto fáz sa na takto vzniknutej novej populácii tieto kroky zopakujú niekoľkokrát. Nakoniec sa z poslednej generácie vyberie najvhodnejší jedinec, ktorý je schopný svoju úlohu vykonať najlepšie zo všetkých. [1]

### 2.2.5 Mutácia morfológie virtuálnych tvorov

Pri mutácií morfológie sa môžu pridať nové končatiny náhodnej veľkosti z daného intervalu na náhodné miesto tvora alebo odobrať existujúce končatiny, ktoré sa vytvorili počas predošlých generácií. Existujúce končatiny sa môžu taktiež zmeniť o náhodné hodnoty, napríklad ich veľkosť alebo umiestnenie.

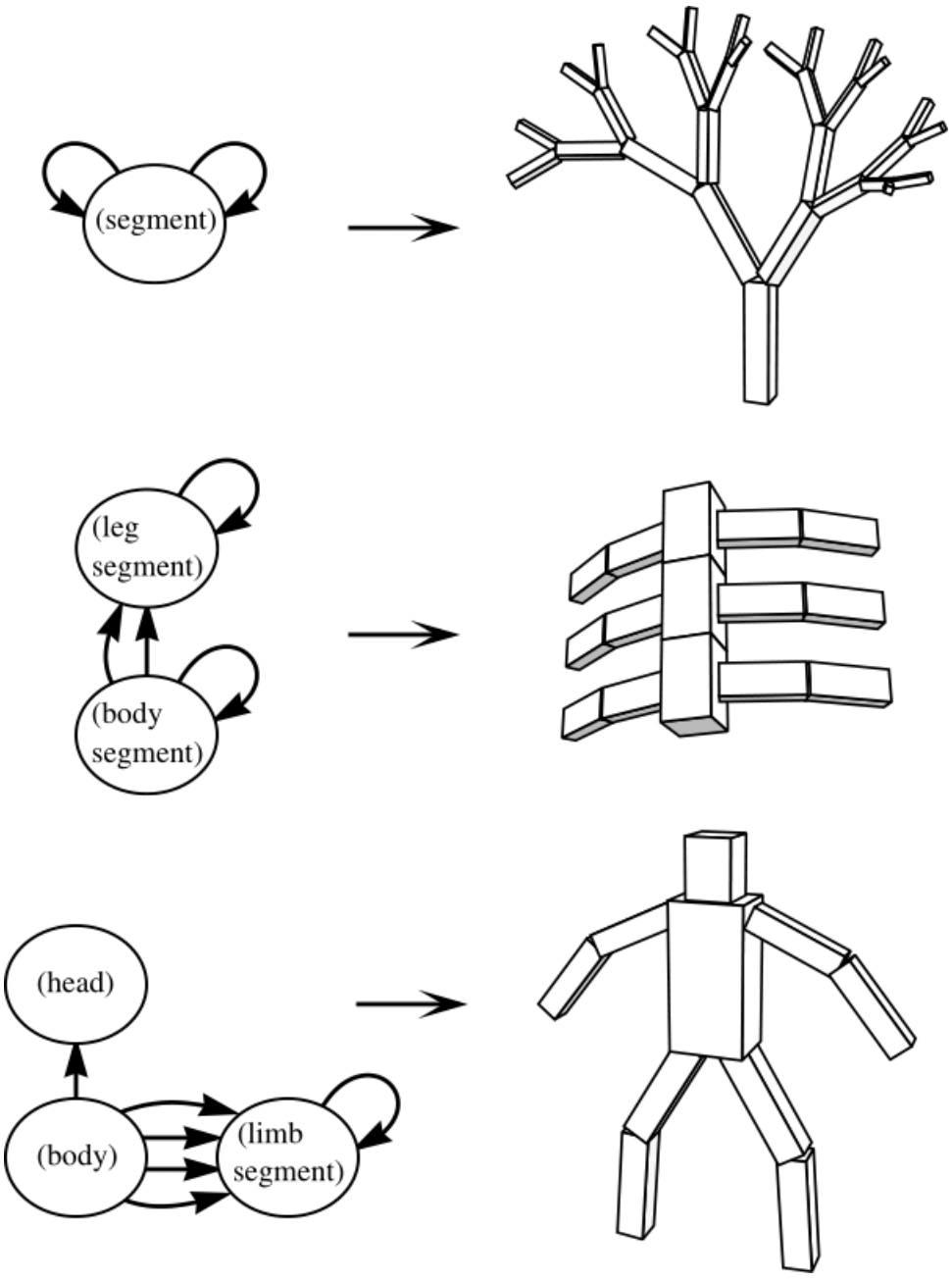
### 2.2.6 Mutácia kontroleru virtuálnych tvorov

Pri použití jednoduchej postupnosti pohybov sa niektorý z nich môže náhodne zmeniť na ľubovoľnú novú hodnotu. Pri použití funkcie, ktorá počíta pohyby podľa informácií získaných zo senzorov a posiela ich efektorom na vykonanie sa táto funkcia môže tiež náhodne zmeniť, aby vypočítala mierne rozličné hodnoty.

## 2.3 Prehľad existujúcich riešení

### 2.3.1 Karl Sims

Karl Sims sa evolúciou virtuálnych tvorov zaoberal v roku 1994. Ako reprezentáciu morfológie použil orientovaný graf, ktorý obsahuje vývojové inštrukcie použiteľné na vytvorenie podobných alebo rekurzívnych častí tvora. Hierarchia častí, ktoré sa syntetizujú na základe informácií vo vrcholoch je vytvorená podľa grafu od koreňa.

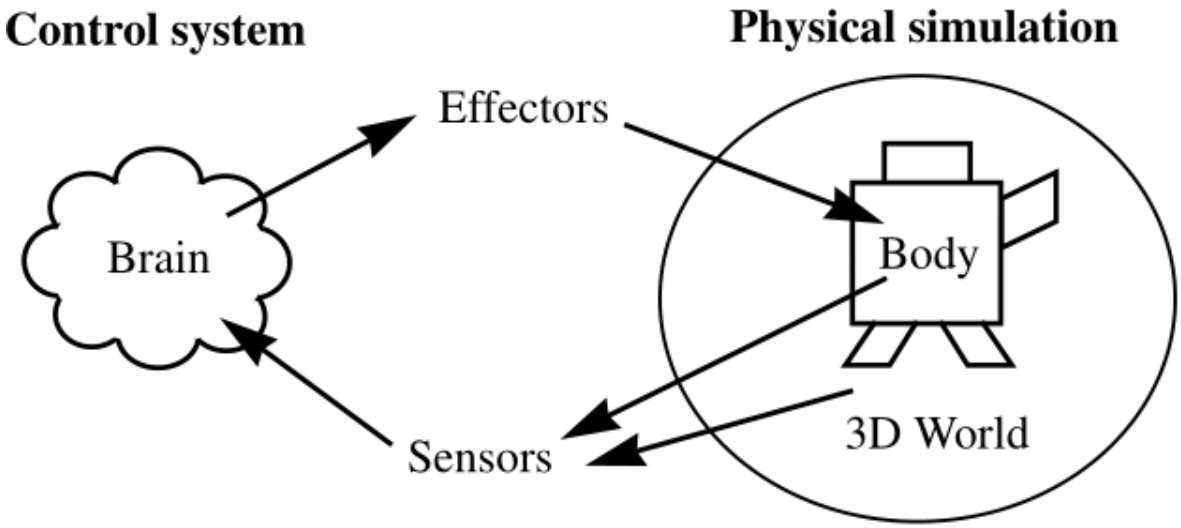



*Obr. 1 – Reprezentácia morfológie virtuálneho tvora pomocou orientovaného grafu*

Každý vrchol obsahuje aj informácie o pevnej časti tvora, ako fyzický tvar, typ kĺbu, ktorý ju spája s rodičom a určuje smer a zároveň maximálne množstvo relatívneho pohybu končatiny, limit rekurzie, ktorý určuje koľkokrát sa má končatina vytvoriť v cykle a spojenia s inými časťami tvora.

Každé spojenie obsahuje informácie o relatívnom umiestnení detí, ako pozícia, orientácia, veľkosť a reflexia, ktorá umožňuje opačné umiestnenie.

Ako reprezentáciu kontroleru použil virtuálny „mozog“, ktorý rozhoduje o správaní sa tvora. Funguje ako dynamický systém, ktorý spracuje hodnoty senzorov a vypočíta hodnoty efektorov, ktoré sú aplikované ako fyzické sily. Tieto hodnoty majú podobu skalárnych premenných nadobúdajúcich kladné hodnoty, ktoré znamenajú jeden smer pohybu a záporné, ktoré znamenajú opačný smer. [2]

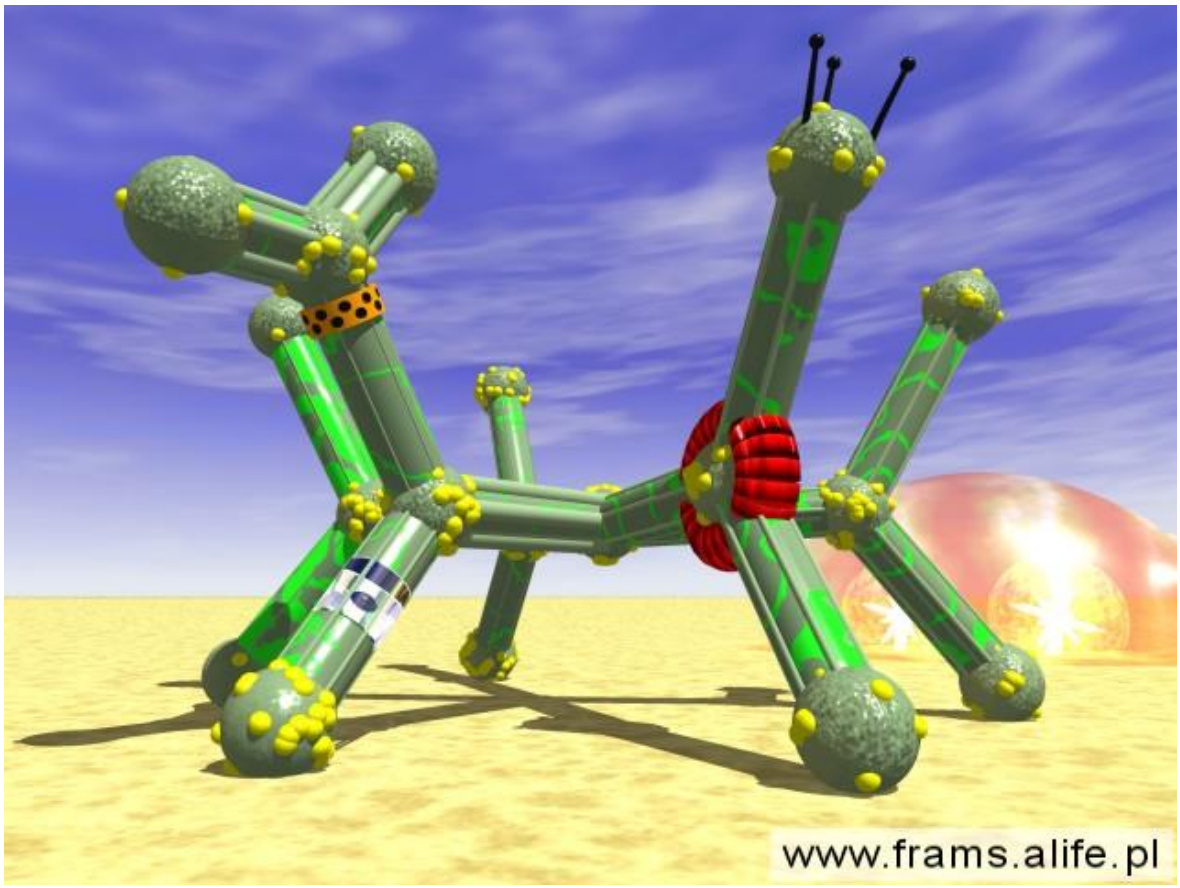



*Obr. 2 – Reprezentácia kontroleru virtuálneho tvora*

### 2.3.2 Framsticks

Framsticks je projekt určený na simuláciu života v prírode. Jeho súčasťou je reprezentácia morfológie a zároveň kontroleru virtuálnych tvorov, ktoré sa snažia napodobniť správanie reálnych živých organizmov a sú vyvinuté pomocou evolučných algoritmov s rôznymi nastaviteľnými parametrami.

Ako telo tvorov sú použité jednoduché paličky spojené kĺbmi, ktoré majú vlastné parametre, ako napríklad dĺžka a hmotnosť končatiny alebo rotácia kĺbu. Pohyb zabezpečujú neuróny s parametrami ako sila a výdrž svalu.




*Obr. 3 – Virtuálny tvor vyvinutý pomocou Framsticks*

Framsticks používa viacero rozličných formátov reprezentácie genotypu, ktorý určuje morfológiu a správanie tvora zároveň. Tieto formáty majú tvar textového reťazca, ale niektoré sa môžu považovať aj za strom.

#### *2.3.2.1 Formát f0*

Vo formáte f0 znamená každý riadok jeden objekt. Syntax vyzerá nasledovne:

```
CLASSID:PROPERTY1,PROPERTY2,...
```

CLASSID znamená typ objektu a môže reprezentovať končatinu, kĺb, neurón alebo definíciu vstupu pre neurón. Existujú dva druhy neurónu, jeden je senzor, ktorý získava informácie z prostredia a druhý je efektor, ktorý pohybuje kĺby. Definícia vstupu spája existujúce

neuróny pomocou ohodnotenej hrany. PROPERTY znamená vlastnosť objektu a vždy má nejakú predvolenú hodnotu, ktorú je možné zmeniť nasledovne:

```
NAME=VALUE
```

NAME znamená názov vlastnosti a VALUE znamená hodnotu. Ak však tieto vlastnosti zadáme podľa daného poradia, stačí vymenovať iba hodnoty a názvy vlastností vynechať. Napríklad majme definíciu objektu:

```
p:x=1,y=2,z=3
```

Táto definícia opisuje končatinu umiestnenú na pozíciu v [1, 2, 3] v 3D priestore, ale keďže súradnice x, y, z sú prvé tri parametre končatiny, môžeme ich zadať aj v zjednodušenom tvare a potom dostaneme kratšiu verziu:

```
p:1,2,3
```

Druhá užitočná možnosť je vynechanie parametrov. Ak pre niektoré parametre chceme použiť predvolenú hodnotu, ale pre nejaké iné, ktoré sa nachádzajú neskôr v postupnosti, chceme nastaviť vlastné hodnoty, napríklad túto definíciu môžeme taktiež zjednodušiť bez zadávania predvolených hodnôt:

```
p:0,0,3
```

Definícia opisuje končatinu umiestnenú na súradnice [0, 0, 3]. Avšak pre x, y sú predvolené hodnoty 0, preto ich môžeme jednoducho vynechať bez zmeny výsledku. Kratšia podoba definície bude tak vyzerať nasledovne:

```
p:,,3
```

Ak chceme použiť predvolené hodnoty pre všetky vlastnosti objektu, nemusíme ich zadávať vôbec. Predvolenú končatinu je tak možné definovať takto:

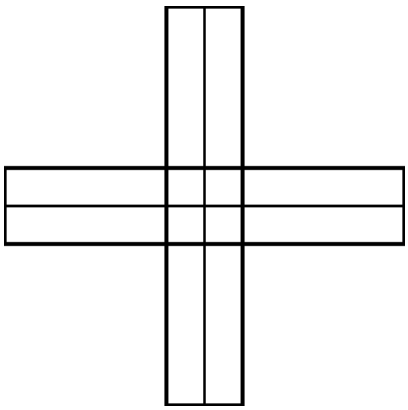
```
p:
```

Ďalšie vlastnosti končatiny sú napríklad hmotnosť, veľkosť alebo orientácia, ktorá určuje množstvo rotácie okolo každej súradnice. Nevýhodou tohto formátu reprezentácie sú dlhé definície, ale na druhej strane je veľmi prehľadný a jednoducho čitateľný.

#### *2.3.2.2 Formát f1*

Formát f1 je zostavený ako strom, v ktorom sú deti pripojené k rodičom cez jeden kĺb. Symbol X znamená končatinu a zátvorky vytvárajú kĺb cez ktorý sú spojené. Napríklad tri končatiny pripojené k jednej vyzerajú takto:

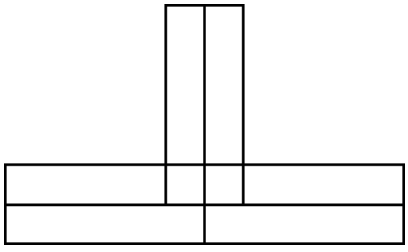
```
X(X,X,X)
```



*Obr. 4 – Tvar tvora „X(X,X,X)“ vo formáte f1*

V tomto prípade je kĺb rozdelený tak, aby medzi všetkými susediacimi končatinami bol uhol rovnakej veľkosti. Ak nechceme aby končatiny boli rozmiestnené rovnomerne, môžeme pridať čiarky, aby bol kĺb rozdelený na viacero častí. Potom medzi končatinami vzniknú uhly rozličnej veľkosti. Majme napríklad nasledujúcu definíciu:

```
X(X,,X)
```



*Obr. 5 – Tvar tvora „X(X,,X)“ vo formáte f1*

V tomto prípade budú končatiny rozmiestnené rovnako, ako pri predošlej definícii, keďže kĺb je rozdelený takisto na štyri časti. Jediný rozdiel je, že jedna končatina chýba, čo znamená že medzi pôvodnými susedmi teraz chýbajúcej končatiny je dvojnásobná veľkosť uhla, ako medzi ostatnými.

Výhodou tohto formátu reprezentácie voči f0 sú kratšie definície, ale na druhej strane je menej prehľadný a ťažko čitateľný.

Hranaté zátvorky vytvárajú neurón a obsahujú o ňom informácie, ako typ, parametre a vstupy neurónu. Syntax vyzerá nasledovne:

```
[NeuronType, PropertyAndInputList]
```

PropertyAndInputList je zoznam párov oddelených čiarkou. Prvý prvok páru určuje parametre neurónu a má nasledujúci tvar:

```
PropertyName:Value
```

Druhý prvok páru určuje vstupy neurónu a zadáva sa takto:

```
NeuronInput:Weight
```

#### *2.3.2.3 Formát f4*

Formát f4 používa nepriame genetické kódovanie, čo znamená že určuje pravidlá, na základe ktorých sa daný jedinec postaví namiesto toho, aby priamo opisoval hotovú štruktúru tvora. Kódy v tomto formáte sú interpretované ako príkazy, ktoré sa aplikujú na bunky genotypu. Tento proces vždy začína jednou bunkou, ktorá sa môže rozdeliť na dve, premeniť na končatinu alebo premeniť na neurón. Preto je aj možné tento formát reprezentácie tiež považovať za strom, podobne ako vo formáte f1. Na zápis definície sa ale takisto používa jednoduchý textový reťazec.

Jednoduchý príklad takejto definície je jeden príkaz, ktorý zmení počiatočnú bunku na končatinu. Zapisuje sa nasledovne:

```
X
```

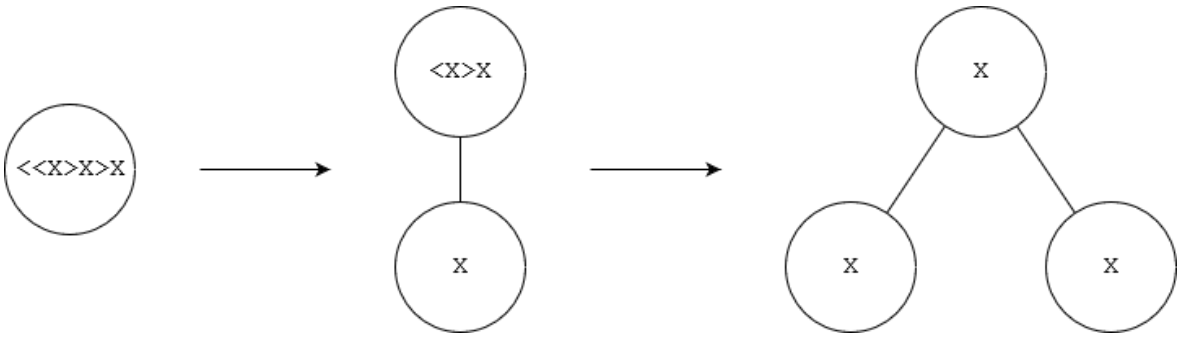
Po zbehnutí tohto kódu vnikne jedinec pozostávajúci z jedinej končatiny s predvolenými vlastnosťami. Táto definícia sa zapisuje rovnako aj vo formáte f1. Na druhej strane, ak chceme dostať napríklad ďalšiu končatinu pripojenú k tej pôvodnej, definícia vo formáte f4 bude vyzerať takto:

```
<X>X
```

Znak „<“ rozdelí bunku na dve a znak „>“ zastaví proces. Po rozdelení sa preto na jednu bunku aplikuje časť medzi „<>“ a na druhú sa použije časť začínajúca po „>“. To znamená že sa obidve bunky zmenia na končatinu a dostaneme očakávaný výsledok. Vo formáte f1 by bolo túto definíciu možné zapísať ako „XX“.

Priebeh procesu na vytvorenie dvoch končatín pripojených k jednej, znázornený na strome vyzerá nasledovne:

```
<<X>X>X
```




*Obr. 6 – Proces vytvárania tvora „<<X>X>X“ vo formáte f4, ktorý má k pôvodnej končatine pripojené ďalšie dve*

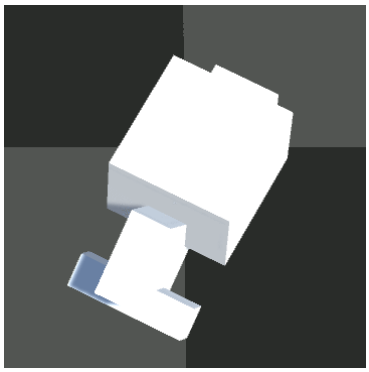
Výsledok je rovnaký, ako pri definícii „X(X,X)“ zapísanej vo formáte f1.

Výhodou tohto formátu reprezentácie je nepriame kódovanie genotypu. Definície sú podobne krátke ako pri formáte f1. [3]

## 2.4 Prehľad technológií a nástrojov

### 2.4.1 Unity

Unity [4] je game engine, v ktorom je možné vytvoriť 3D virtuálne tvory. Obsahuje aj rôzne druhy kĺbov, pomocou ktorých je možné spájať objekty.



*Obr. 7 – Virtuálny tvor vytvorený v Unity*

Virtuálny tvor vytvorený v Unity môže vyzerať napríklad ako ukazuje obrázok napravo. Tento tvor bol vyvinutý na prekonanie čo najväčšej vzdialenosti za daný čas.

### 2.4.2 Visual Studio

Visual Studio [5] je vývojové prostredie ktoré je možné použiť na editovanie a kompilovanie kódu napísaného v rozličných programovacích jazykoch. Medzi tieto jazyky patrí aj C#, ktorý je vhodný na tvorbu skriptov pre Unity. Inštalačný balík Unity obsahuje aj dodatok Visual Studio Tools [6], ktorý umožní tvorbu skriptov v prostredí Visual Studio a ich debugovanie v editore Unity.

### 2.4.3 C#

Pre prácu s Unity je nutné ku kódom napísaných v programovacom jazyku C# [7] importovať knižnicu UnityEngine, ktorá je súčasťou skriptovacej API poskytovanej aplikáciou Unity. Táto knižnica umožní napríklad vytváranie 3D objektov a ich spájanie pomocou rozličných kĺbov. O simuláciu vo virtuálnom svete sa postará samotný Unity. Týmto spôsobom je možné aplikovať evolučné algoritmy na virtuálnych tvoroch.

Kód pre vytvorenie jednoduchej kocky použitím programovacieho jazyka C# a knižnice UnityEngine vyzerá napríklad nasledovne:

```
GameObject cube = GameObject.CreatePrimitive(PrimitiveType.Cube);
cube.transform.position = new Vector3(0, 0.5F, 0);
```

Prvý riadok tohto kódu vytvorí GameObject s parametrom PrimitiveType.Cube, čo znamená že bude mať tvar kocky s predvolenými vlastnosťami, ako napríklad veľkosť [1; 1; 1]. Druhý

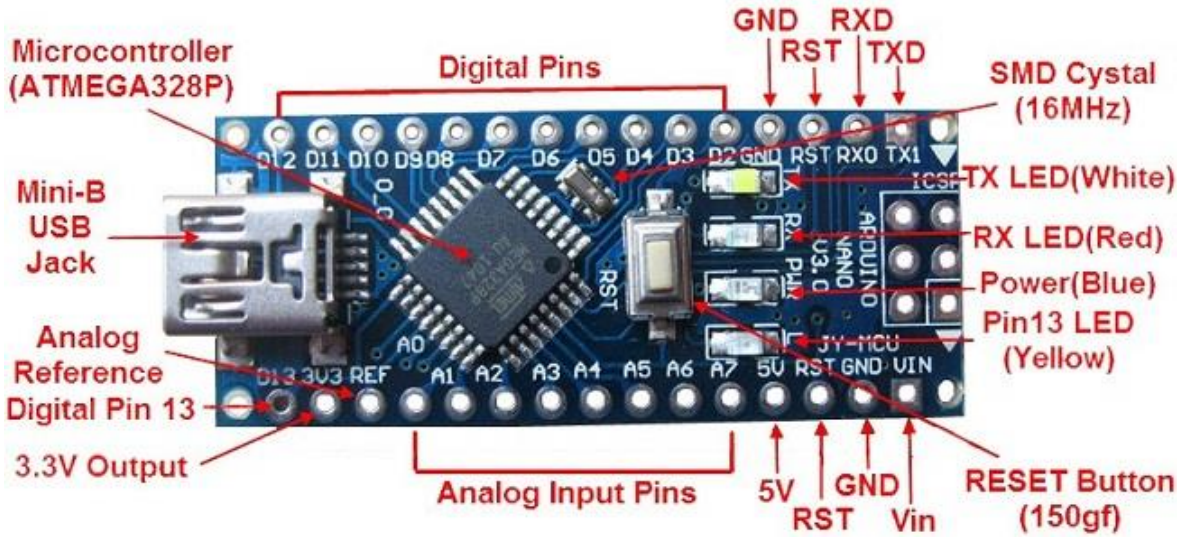
riadok presunie stredový bod vytvorenej kocky na pozíciu [0; 0,5; 0], čo znamená že jej dolná časť sa dostane na vertikálnu súradnicu y = 0. Na ľavom obrázku vidíme polohu kocky po vytvorení z boku, kde sivá čiara znamená x-ovú os. Na pravom obrázku vidíme kocku po presunutí o pol jednotky smerom hore.

*Obr. 8 – Premiestnenie kocky v Unity z pozície [0; 0; 0] na pozíciu [0; 0,5; 0] z bočného pohľadu*

### 2.4.4 Virtuálne tvory v realite

Po reálnom vytvorení virtuálneho tvora na 3D tlačiarni budú kĺby osadené servomotormi, ktoré budú pripojené na Arduino Nano [8].

### 2.4.5 Arduino Nano




*Obr. 9 – Arduino Nano*

Arduino Nano je vývojová doska, ktorú je možné pripojiť k PC pomocou USB kábla a naprogramovať v prostredí Arduino IDE. [9]

# 3 Návrh

## 3.1 Reťazcová reprezentácia jedincov

Morfológia a kontroler jedincov sú reprezentované, ako jednoduchý znakový reťazec, podobne ako vo Framsticks [3] vo formáte f4. Jednotlivé znaky tohto reťazca sú interpretované ako príkazy, na základe ktorých sa telo daného jedinca vytvorí a neskôr pohybuje.

Tvorba jedinca začína počiatočnou bunkou, ktorá zatiaľ nemá žiadne dané vlastnosti. Táto bunka sa môže rozdeliť na dve nové alebo sa zmeniť na končatinu. Ak sa ľubovoľná bunka už raz premenila na končatinu, nemôže sa ďalej rozdeľovať, ani sa premeniť naspäť. Takisto ani jej vlastnosti už nie je možné ďalej meniť. Ak neostala žiadna bunka, ktorá sa ešte nepremenila na končatinu, jedinec je úplne vytvorený a nie je možné v jeho tvorbe nijakým spôsobom pokračovať.

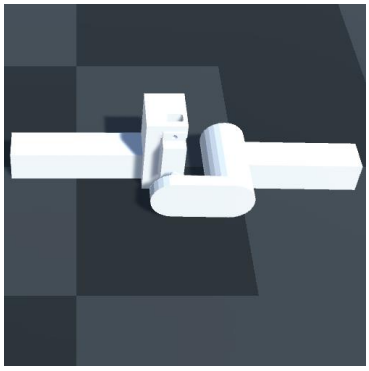
Rozdelenie bunky na dve sa značí znakom „<“ a koniec vývoja znakom „>“. To znamená, že po rozdelení bunky sa na jednu z tých dvoch nových aplikujú príkazy, ktoré sa nachádzajú medzi znakom „<“ a prvým znakom „>“.

Premenenie bunky na končatinu sa značí znakom „L“. Končatiny sú k sebe pripojené pomocou kĺbov. Každý kĺb môže mať nejaký kontroler, ktorý určuje jeho pohyby. Príkazy tohto kontroleru sú umiestnené medzi „[“ a „]“ priamo za znakom „L“.

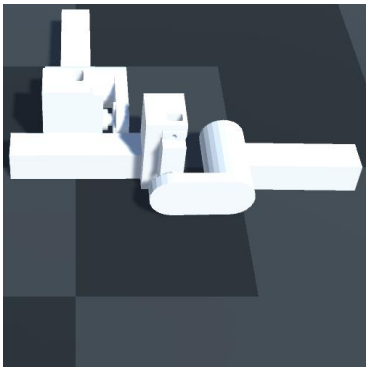
### 3.1.1 Štruktúra končatín

Po rozdelení počiatočnej bunky na dve a premenenie takto vzniknutých nových buniek na končatinu bude koniec prvej výslednej končatiny pripojený na začiatok druhej končatiny pomocou pántového kĺbu. Ak po rozdelení počiatočnej bunky prvú z dvoch nových buniek rozdelíme znova, jedná sa o dvojité rozdelenie bunky. Pri takomto rozdelení, na rozdiel od predošlého prípadu, bude na prvú končatinu pripojená aj ďalšia, ale už na ľavý bok tej pôvodnej. V prípade trojitého rozdelenia počiatočnej bunky bude ďalšia končatina pripojená na pravý bok tej pôvodnej.

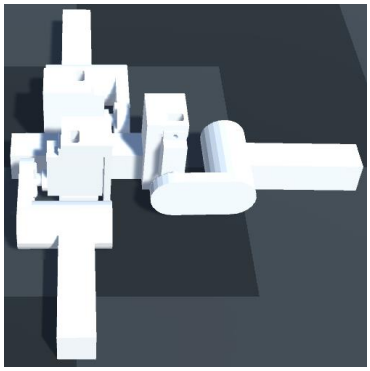
<L>L <<L>L>L <<<L>L>L>L



*Obr. 10 – <L>L*



*Obr. 11 – <<L>L>L*



*Obr. 12 – <<<L>L>L>L*

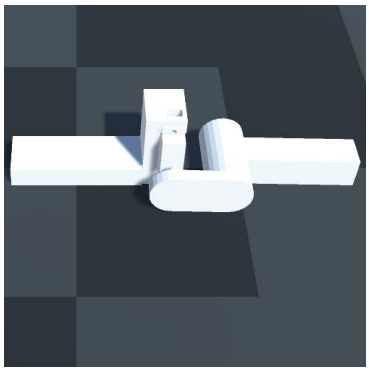
Na obrázku vyššie vidíme jednoduché, dvojité a trojité rozdelenie počiatočnej bunky.

### 3.1.2 Modifikátory

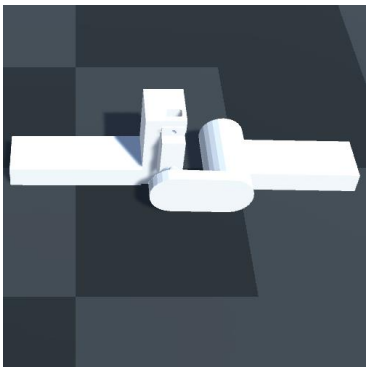
Modifikátory menia rozmery končatiny. Ak sa bunka premení na končatinu bez toho, že by bola predtým modifikovaná, budú pre takto vzniknutú končatinu použité predvolené vlastnosti. Ak sa modifikovaná bunka rozdelí, obidve vzniknuté bunky budú mať tie isté modifikované vlastnosti.

Používajú sa znaky „x“, „X“, „y“, „Y“, „z“ a „Z“. Veľké písmeno daný rozmer zväčší a malé ho naopak zmenší. Modifikátorov môže byť ľubovoľný počet. Čím viac je rovnakých modifikátorov, tým viac sa daný rozdiel zmení.

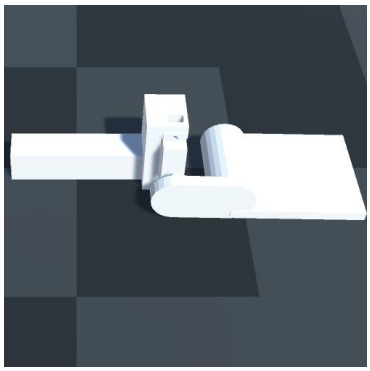
<yZL>L yZ<L>L <L>yyyZZZL



*Obr. 13 – <yZL>L*



*Obr. 14 – yZ<L>L*



*Obr. 15 – <L>yyyZZZL*

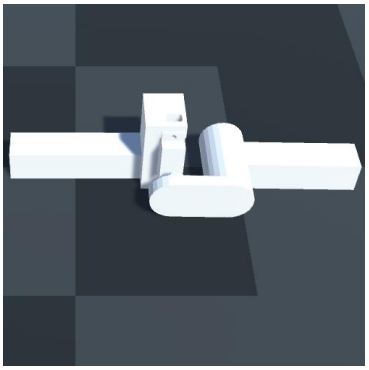
Na prvom obrázku vyššie sú modifikátory „y“ a „Z“ umiestnené po znaku „<“, čo znamená, že sú aplikované iba po rozdelení počiatočnej bunky a iba na prvú z dvoch novo vzniknutých buniek. Modifikátor „y“ zmenšuje rozmer y a modifikátor „Z“ zväčšuje rozmer z. Na druhom obrázku sú tie isté modifikátory umiestnené ešte pred rozdelením počiatočnej bunky, čo znamená, že vplýva na obidve novo vzniknuté bunky. Na treťom obrázku sú tieto modifikátory použité iba na druhú z buniek vzniknutých po rozdelení počiatočnej bunky, ale na rozdiel od prvých dvoch obrázkov sa vyskytujú trikrát za sebou, čo znamená, že dané rozmery menia podstatnejšie. Výsledkom takto je tenká a široká končatina.

### 3.1.3 Rotácia kĺbov

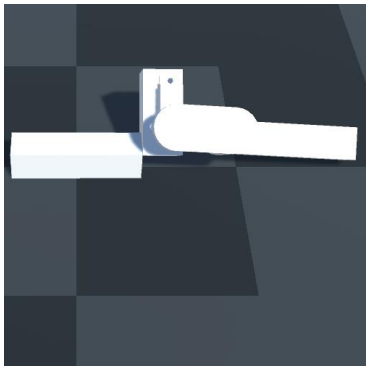
Príkazy rotácie kĺbov určujú mieru otočenia medzi kĺbom a končatinou na ktorú je pripojený. Rovnako ako pri modifikátoroch, aj rotačné príkazy platia pre všetky nové bunky, ktoré vznikli pomocou rozdeľovania z pôvodnej bunky, pred ktorou bol príkaz otočenia umiestnený. Na rozdiel od modifikátorov však tieto príkazy nemenia predchádzajúce hodnoty, ale absolútne určujú novú. Ak sa pred nejakou bunkou takýto príkaz nenachádza, použitá bude predvolená hodnota, ktorá sa rovná otočeniu o 0 stupňov.

Používa sa reprezentácia „R(x,y)“. Má dva parametre, ktoré určujú mieru otočenia kĺbu v stupňoch podľa x-ovej a y-ovej osi. Ak sa obidva tieto parametre rovnajú nule, tento príkaz nemá žiaden vplyv na morfológiu jedinca.

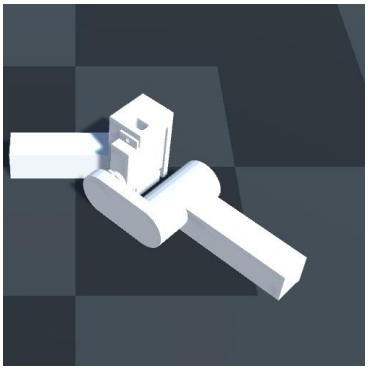
`<L>R(0,0)L` `<L>R(45,0)L` `<L>R(0,45)L`



*Obr. 16 – <L>R(0,0)L*



*Obr. 17 – <L>R(45,0)L*



*Obr. 18 – <L>R(0,45)L*

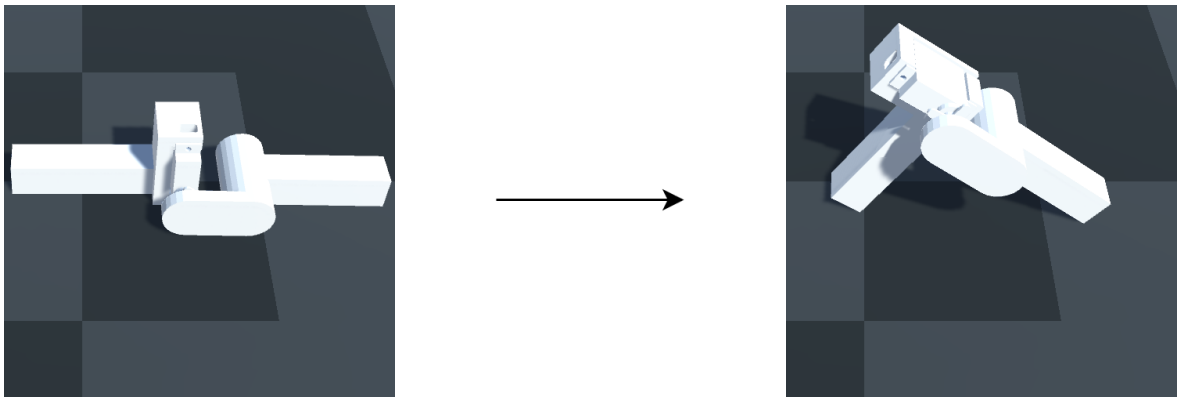
Na obrázkoch vyššie vidíme kĺb s predvoleným otočením, kĺb otočený o 45 stupňov podľa x-ovej osi a kĺb otočený o 45 stupňov podľa y-ovej osi.

### 3.1.4 Príkazy pohybu

Začiatok príkazov pohybu sa značí znakom „[“ a ich koniec sa značí znakom „]“. Znak „[“ je umiestnený vždy priamo za znakom „L“, ktorý značí premenenie bunky na končatinu. To znamená, že každé pole príkazov patrí presne k jednej končatine. Toto pole určuje pohyby kĺbu, pomocou ktorého je daná končatina jedinca pripevnená k predošlej.

Medzi hranatými zátvorkami sa nachádzajú čísla 0, 1 a 2. Číslo 0 znamená pohyb do jedného smeru, číslo 1 pohyb do druhého smeru a číslo 2 žiaden pohyb. Tieto príkazy sa vykonávajú v danom časovom intervale, každý príkaz rovnako dlho. Ak sa nachádzajú tie isté príkazy dvakrát priamo za sebou, daný pohyb sa bude vykonávať dvakrát tak dlho. Ak kĺb dokončí vykonanie posledného príkazu, začne znova od prvého. Počet príkazov pohybu je ľubovoľný pre každú končatinu.

```
<L>L[0]
```



*Obr. 19 – Pohyb jedinca reprezentovaného ako „<L>L[0]“*

### 3.1.5 Alternatívy príkazov pohybu

Aby jedinec bol schopný vnímať svoje okolie, jeho kĺby potrebujú viacero alternatív polí pohybu. Tieto alternatívy môžeme považovať za prvky dvojrozmerného poľa príkazov kĺbu. Po vyhodnotení okolia sa jedinec rozhodne, ktorú alternatívu pohybov použije. Toto mu umožní, aby sa správal rôzne za rôznych okolností. Napríklad ak chceme, aby sa jedinec otáčal za svetlom, môžu sa mu vyvinúť dve rozličné polia príkazov pre všetky kĺby. Použitím prvého poľa sa bude otáčať jedným smerom a použitím druhého poľa druhým smerom. Pole si vyberie podľa toho, na ktorej strane sa svetlo nachádza. Týmto spôsobom bude schopný otáčať sa vždy do správneho smeru za svetlom. Polia pohybov sú rozdelené znakom „|“.

## 3.2 Vytváranie jedincov podľa reťazcovej reprezentácie

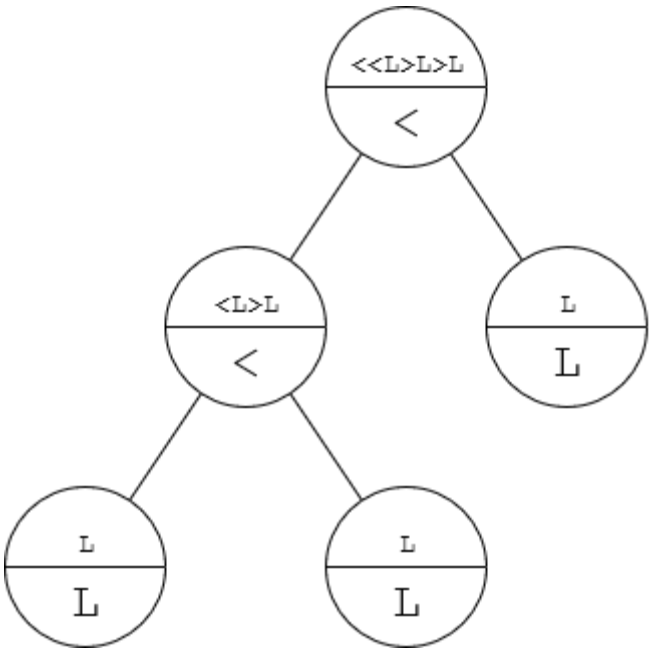
Pri vytváraní jedincov je reťazcová reprezentácia interpretovaná ako binárny strom. Končatiny vznikajú z buniek postupne počas prehľadávania tohto stromu, pričom sú súčasne aplikované aj modifikátory, ktoré menia ich rozmery. Taktiež sa použijú aj príkazy rotácie kĺbov, ktoré určujú mieru otočenia.

### 3.2.1 Stavanie končatín

Počas prehľadávania binárneho stromu vzniknutého interpretáciou reťazcovej reprezentácie vzniká nový podstrom pri aplikovaní znaku „<“, ktorý znázorňuje rozdelenie bunky na dve. Pri aplikovaní znaku „L“ vznikne končatina a proces pre daný podstrom sa zastaví.

Na obrázku nižšie vidíme proces prehľadávania stromu, počas ktorého vzniká jedinec reprezentovaný ako „<<L>L>L“. Horné časti vrcholov znázorňujú časť reťazca, ktorá bude počas prehľadávania ešte spracovaná. Dolná časť znamená príkaz, ktorý bude použitý ako ďalší krok procesu. Tento znak sa zhoduje s prvým znakom reťazca v hornej časti vrcholu, keďže je reťazec spracovaný zľava doprava.

<<L>L>L



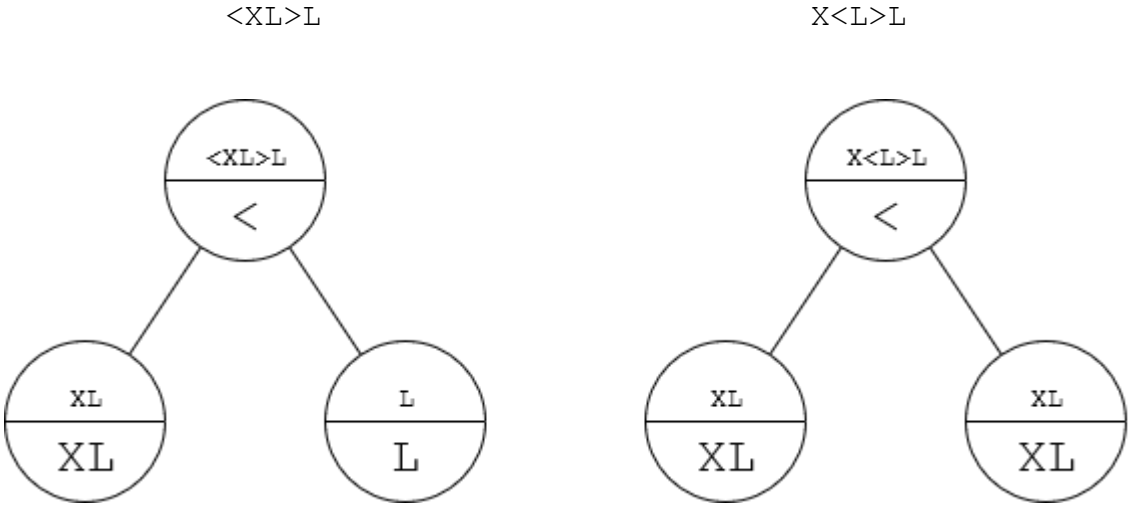

*Obr. 20 – Prehľadávanie stromu reprezentovaného ako „<<L>L>L“*

Listy tohto stromu znamenajú končatiny jedinca. Preto aj vieme, že tento jedinec bude mať tri končatiny. Keďže kĺby jedinca vznikajú pri rozdelení buniek, na tomto strome sa ich počet zhoduje s počtom vrcholov, ktoré nie sú listami. Preto má tento jedinec dva kĺby. Uvedomme si, že počet kĺbov jedinca je vždy o jedno menší, ako počet jeho končatín, keďže jeden kĺb spája dve končatiny.

### 3.2.2 Aplikácia modifikátorov

Modifikátory sú aplikované na celý podstrom, pred ktorým sa nachádzajú. Toto umožní, aby mali vplyv na všetky končatiny, ktoré vzniknú z bunky umiestnenej za nimi. To znamená, že modifikované budú aj končatiny, ktoré boli vytvorené z nových buniek vzniknutých po rozdelení spomínanej bunky. Samozrejme to rekurzívne platí pre všetky novo vzniknuté bunky po ľubovoľnom počte rozdelení.

Na obrázku nižšie vidíme porovnanie aplikácie modifikátora značeného znakom „X“, ktorý zväčšuje daný rozmer končatiny. Na ľavom strome je tento modifikátor aplikovaný iba na prvú končatinu. Na pravom strome je ale ten istý modifikátor aplikovaný na obidve končatiny, keďže je umiestnený ešte pred rozdelením počiatočnej bunky. Toto rozdelenie je znázornené znakom „<“.




*Obr. 21 – Porovnanie modifikátora aplikovaného na jednu a na obidve končatiny*

Samotný modifikátor na tomto obrázku nie je považovaný za príkaz, iba znázorňuje zmenený rozmer končatín jedinca.

### 3.2.3 Aplikácia rotácie kĺbov

Rovnako ako pri modifikátoroch, rotačné príkazy sú tiež aplikované na celý podstrom, ktorý sa nachádza pod nimi. Jediný rozdiel je, že ak sa po takomto príkaze nachádza ďalší príkaz rotácie, ten predchádzajúci už nemá žiaden vplyv na ďalšie nové kĺby vytvorené po aplikovaní posledného rotačného príkazu, ktorý sa nachádza pred nimi. To znamená, že tieto príkazy určujú vždy absolútne mieru otočenia kĺbov, ktoré sa za nimi nachádzajú, namiesto toho, aby menili nejaké predchádzajúce hodnoty, ako modifikátory. Ak sa nachádzajú dva rotačné príkazy priamo vedľa seba, ten prvý nebude mať žiaden vplyv na morfológiu výsledného jedinca. Napríklad reťazec „<L>R(90,90)R(45,45)L“ reprezentuje presne rovnakého jedinca, ako reťazec „<L>R(45,45)L“, keďže v prvom prípade bol príkaz „R(90,90)“ prepísaný príkazom „R(45,45)“, ktorý je použitý takisto aj v druhom reťazci.

## 3.3 Mutačné funkcie morfológie

Pri mutácií morfológie jedinca sa použije viacero rôznych funkcií. Počas vzniku novej generácie môže každá z týchto funkcií zbehnúť raz alebo vôbec. Pri vysokej pravdepodobnosti mutácie je ľahko možné, že zbehnú všetky mutačné funkcie a naopak, pri nízkej pravdepodobnosti sa môže stať, že nezbehne ani jedna. Tieto funkcie pridávajú alebo odoberajú presne jeden príkaz v reťazcovej reprezentácii jedinca.

### 3.3.1 Pridávanie a odoberanie končatín

Pri odoberaní končatiny sa náhodne vyberie jedna končatina jedinca a odstráni sa. Pri pridávaní sa nájde voľné miesto pre končatinu, na ktoré sa následne vytvorí.

### 3.3.2 Mutácia modifikátorov

Podobne ako pri končatinách, aj pri odoberaní modifikátorov sa náhodne vyberie niektorý modifikátor, ktorý sa následne odstráni. Pri pridávaní sa vloží náhodný modifikátor na náhodnú pozíciu reťazcovej reprezentácie daného jedinca.

### 3.3.3 Mutácia rotácie kĺbov

Rotačné funkcie kĺbov sa odoberajú rovnako, ako modifikátory a pridávajú sa s náhodnými parametrami na náhodnú pozíciu reťazcovej reprezentácie daného jedinca.

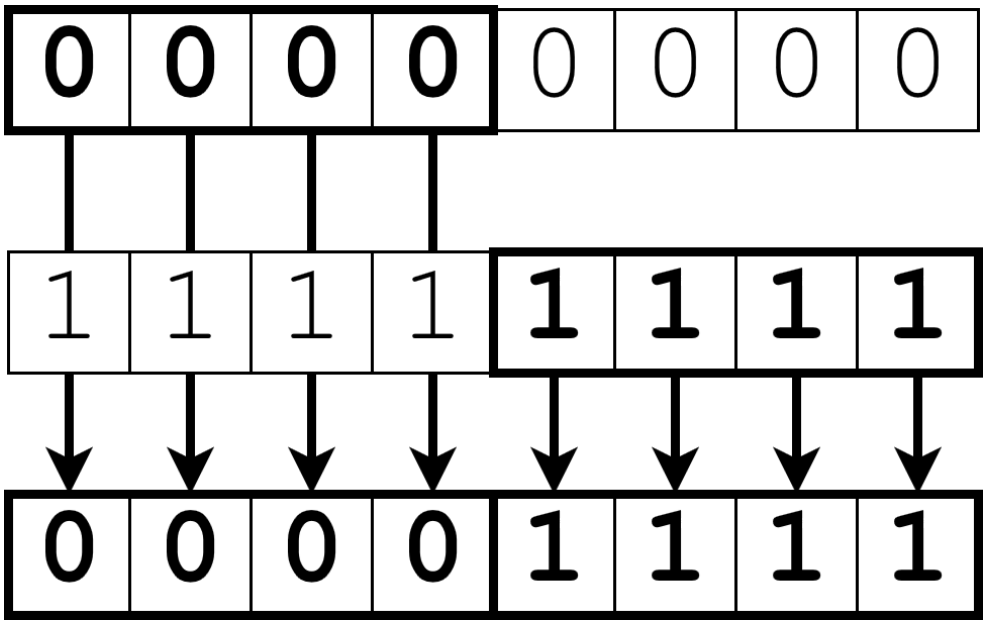
## 3.4 Mutačné funkcie kontroleru

Kontrolery kĺbov jedinca majú iba dve základné mutačné funkcie. Jedna z nich odoberá náhodný príkaz z náhodného kontroleru. Odoberať môže aj znak „|“, ktorý určuje pozíciu hranice dvoch susedných kontrolerov pri použití viacerých alternatív. Druhá mutačná funkcia kontroleru pridáva náhodný príkaz na náhodnú pozíciu náhodného kontroleru daného jedinca. Rovnako ako pri odoberaní, pridávať môže tiež aj rozdeľovač kontrolerov kĺbu, ktorý je značený znakom „|“.

## 3.5 Kríženie kontrolerov

Jedna z operácií evolučných algoritmov je kríženie jedincov. Na kríženie znakových reťazcov existuje veľmi jednoduchý spôsob. Začiatok prvého reťazca spojíme s koncom druhého reťazca a tak dostaneme nový výsledný reťazec vytvorený z podreťazcov reprezentácie pôvodných kontrolerov.

Keďže v našej reťazcovej reprezentácii kontroleru sa môžu vyskytovať znaky „0“, „1“, „2“ a „|“ v ľubovoľnom poradí na ľubovoľných pozíciách, nemusíme nič ani kontrolovať, iba jednoducho spojiť znakové reťazce. Ako príklad, na obrázku nižšie vidíme znázornenie tejto operácie na reťazcových reprezentáciách dvoch náhodných kontrolerov.




*Obr. 22 – Jednoduchý spôsob kríženia kontrolerov*

Rovnako ako pri mutácií, aj pri krížení jedincov je daná hodnota pravdepodobnosti, aby sa táto operácia vykonala. Ak sa nevykoná, jednoducho sa vyberie počet jedincov, ktorý určuje hodnota parametra veľkosti populácie. Ak sa však vykoná, namiesto každého jedinca sa vyberú dvaja, ktorí sa následne skrížia. Z obidvoch jedincov sa vyberie náhodný kontroler. Z reťazovej reprezentácie prvého sa vyberie podreťazec náhodnej dĺžky začínajúci prvým znakom pôvodného reťazca. Z reprezentácie druhého kontroleru sa tiež vyberie podreťazec náhodnej dĺžky, ale tento už končí posledným znakom pôvodného reťazca. Tieto podreťazce sa spoja a vznikne reťazcová reprezentácia nového, potenciálne lepšieho kontroleru.

## 3.6 Výber jedincov do ďalšej generácie

Jedinci populácie sa do ďalšej generácie evolúcie vyberajú náhodne. Jedinci s vyššou vhodnosťou však majú väčšiu šancu na to, aby boli vybratí. Preto sa aj ľahko môže stať, že jedinec s relatívne vysokou vhodnosťou bude vybratý viackrát a jedinec s nízkou vhodnosťou nebude vybratý ani raz, čo spôsobí jeho zaniknutie. Ak je niektorý jedinec vybratý do novej generácie viac ako raz, znamená to, že naňho môže byť aplikovaná mutácia viackrát s rozličným výsledkom a z pôvodného jedinca vznikne niekoľko nových. Keďže každý z týchto jedincov je výsledkom mutácie jedinca s relatívne vysokou vhodnosťou, je vysoká pravdepodobnosť, že viacero z nich bude mať tiež vysokú vhodnosť a aspoň jeden ju bude mať ešte vyššiu, ako ten pôvodný z predošlej generácie. Týmto spôsobom sa evolúcia posúva postupne dopredu a vnikajú stále lepší a lepší jedinci.

## 3.7 Parametre evolúcie

Evolúcia má tri základné parametre. Pravdepodobnosť mutácie, veľkosť populácie a trvanie jednej generácie počas evolúcie. Tieto parametre sa počas evolúcie nemenia.

### 3.7.1 Pravdepodobnosť mutácie

Pravdepodobnosť mutácie určuje šancu na zbehnutie jednotlivých mutačných funkcií. Pri vyššej hodnote je šanca, že vhodnosť jedincov bude stúpať rýchlejšie a evolúcia tak prebieha rýchlejšie. Na druhej strane však príliš vysoká pravdepodobnosť mutácie spôsobuje, že evolúcia bude menej stabilná a ľahko sa môže stať, že sa vývoj prestane posúvať dopredu alebo sa dokonca obráti a do novej generácie vzniknú jedinci s nižšou vhodnosťou.

### 3.7.2 Veľkosť populácie

Veľkosť populácie určuje počet jedincov v jednej generácii, ktorí sa zúčastňujú evolúcie. Vyššia hodnota umožňuje viacero mutácií naraz, a preto zvyšuje aj pravdepodobnosť, že počas mutácie niektorého jedinca vznikne nový jedinec s vyššou vhodnosťou, ako pri nižšej veľkosti populácie. Väčšia populácia samozrejme tiež znižuje výkon algoritmu.

### 3.7.3 Trvanie generácie

Trvanie generácie určuje ako dlho jedna generácia evolúcie existuje. Jednotlivý jedinci majú možnosť tento čas využiť na to, aby ukázali svoju vhodnosť. Výhoda vysokej hodnoty je vyššia presnosť určenia vhodnosti a nevýhoda je pomalší beh evolúcie.

### 3.7.4 Vývojárske parametre

Vývojárske parametre evolúcie sú parametre, ktorých hodnotu bežný užívateľ nemá možnosť určiť, keďže nie sú súčasťou užívateľského rozhrania. Jeden z dôvodov je, že týchto parametrov je zbytočne veľa a užívateľom by ich trvalo príliš dlho nastaviť tak, aby dosiahli ideálne výsledky. Užívateľ preto môže meniť iba základné parametre, ktoré však tiež majú predvolené hodnoty pre každú úlohu evolúcie. Tieto hodnoty boli nastavené podľa výsledkov experimentov pre čo najideálnejšiu evolúciu.

## 3.8 Úlohy evolúcie

### 3.8.1 Prekonanie vzdialenosti

Pri tejto úlohe ide o prekonanie čo najväčšej vzdialenosti za daný čas. Tento čas sa rovná hodnote parametra trvania generácie.

### 3.8.2 Dosiahnutie výšky

Pri tejto úlohe ide o dosiahnutie čo najvyššej vertikálnej polohy presne po uplynutí času trvania generácie.

### 3.8.3 Demolácia objektov

Pri tejto úlohe ide o búranie objektu postaveného z viacerých menších kociek.

### 3.8.4 Otáčanie za svetlom

Pri tejto úlohe ide o vyvinutie jedinca, ktorý je schopný vnímať svetlo a otáčať sa za ním.

## 3.9 Priebeh evolúcie

Na začiatku evolúcie sa vytvorí počiatočná generácia. Po uplynutí času, ktorý určuje parameter trvania generácie, sa vypočíta vhodnosť jednotlivých jedincov. Na základe týchto hodnôt sa náhodne vyberie daný počet jedincov, ktorý určuje parameter veľkosti populácie, do ďalšej generácie. Keďže sa hodnoty parametrov počas behu evolúcie nemenia, aj hodnota tohto parametra bude rovnaká pre všetky generácie, čo znamená, že všetky populácie budú rovnako veľké. Jedinci s vyššou vhodnosťou sa pravdepodobne vyberú viackrát, ako jedinci s nižšou vhodnosťou. Vo výslednej populácii potom bude viacero jedincov, ktorí vznikli mutáciou toho istého jedinca z predošlej generácie. Iní jedinci jednoducho zaniknú a ďalej sa už nezúčastnia evolúcie.

Po výbere jedincov do ďalšej generácie sa s pravdepodobnosťou určenou hodnotou parametra pravdepodobnosti mutácie vybraní jedinci náhodne zmutujú. Pri nízkej hodnote tohto parametra je pravdepodobné, že sa do novej generácie niektorí jedinci dostanú bez akejkoľvek mutácie. To znamená, že nová generácia môže obsahovať rovnakých jedincov, ako predošlá generácia. Tento jav sa vyskytuje hlavne vtedy, ak je niektorý jedinec populácie už blízko dokonalosti alebo sa z iného dôvodu nie je schopný vyvíjať ďalej. V takomto prípade bude veľa jedincov rovnakých, čo spôsobuje vyššiu pravdepodobnosť, že sa niektorý z nich nezmutuje a zostane taký istý, ako niektorí jedinci z predošlej generácie.

## 3.10 Určenie vhodnosti jedincov

Po uplynutí trvania generácie sa určí vhodnosť každého jedinca populácie. Vhodnosť môže byť napríklad prekonaná vzdialenosť, dosiahnutá výška alebo stav demolácie objektu, podľa úlohy evolúcie. V niektorých prípadoch je nutné tieto hodnoty merať aj počas trvania generácie, nie len na konci. Jedná sa o viacfázové určenie vhodnosti. Pre takýto typ určenia vhodnosti jedincov je príkladom úloha otáčania za svetlom. Pre dosiahnutie dostatočnej presnosti sa počas trvania generácie svetlo niekoľkokrát premiestni a odmeria uhol medzi jedincom a polohou svetla. Na konci generácie sa vyráta geometrický priemer týchto uhlov.

## 3.11 Reálne vytváranie jedincov pomocou 3D tlačiarne

### 3.11.1 Exportovanie STL

Na to, aby sme mohli jedinca vytvoriť pomocou 3D tlačiarne, musíme najprv exportovať 3D modely jeho končatín napríklad vo formáte STL. Pre softvér Unity [4] existuje bezplatný plugin s názvom pb_Stl [10], ktorý to umožní. Vytvoril ho jeden z užívateľov Unity a nahral ho na GitHub.

### 3.11.2 Osadenie kĺbov servomotormi

Na to, aby sa reálne vytvorený jedinec bol schopný pohybovať, je nutné jeho kĺby osadiť servomotormi. V našom evolučnom softvéri sa preto jedinci vyvíjajú tak, aby bolo možné na miesto ich kĺbov pripevniť servomotor pomocou skrutiek. Konkrétne sa jedná o model HD-1440A [11].


*Obr. 23 – Servomotor HD-1440A*

### 3.11.3 Pripojenie servomotorov na Arduino

Po osadení kĺbov jedinca servomotormi je nutné tieto motory pripojiť napríklad na vývojovú dosku Arduino Nano [8]. Hnedý drôt pripojíme na pin GND, červený drôt na pin 5V a žltý drôt na ľubovoľný digitálny pin od D2 až po D12. Každý servomotor musíme ale pripojiť na rozličné digitálne piny, aby sme mali možnosť ovládať každý motor zvlášť.

### 3.11.4 Pripojenie fotosenzorov na Arduino

Na to, aby náš reálne vytvorený jedinec bol schopný vnímať svetlo, potrebujeme k vývojovej doske Arduino Nano pripojiť aj fotosenzory. Použijeme model L-NP-3C1 [12]. Kratší drôt pripojíme na pin GND a dlhší drôt na ľubovoľný analógový pin od A0 až po A7. Samozrejme musíme všetky senzory pripojiť na rozličné analógové vstupy, aby sme po získaní hodnoty vedeli určiť, o ktorý senzor sa jedná. Dlhší drôt senzoru ešte pripojíme na odpor, ktorý pripojíme ďalej na pin 5V dosky Arduino Nano.

### 3.11.5 Programovanie jedinca v Arduino IDE

Vývojové prostredie Arduino IDE nám umožní napísať program pre vývojovú dosku Arduino Nano a zároveň ho na ňu nahrať cez USB port. Program napísaný v tomto prostredí sa nazýva sketch. Takýto sketch má dve základné metódy. Jedna z nich je metóda „setup“, ktorá zbehne iba raz, priamo po spustení programu. Druhá je metóda „loop“, ktorá beží stále dookola po zbehnutí metódy „setup“, ako cyklus. Príklad pre jednoduchý sketch, ktorý ovláda servomotor, vyzerá nasledovne:

```
#include <Servo.h>

Servo myservo;          // vytvorí objekt pre servomotor

void setup() {
  myservo.attach(2);    // pripojí objekt servomotoru na digitálny pin D2
}

void loop() {
    myservo.write(0);   // pohne servomotor do pozície 0
    delay(1000);        // zastaví sa na jednu sekundu
    myservo.write(90);  // pohne servomotor do pozície 90
    delay(1000);        // zastaví sa na jednu sekundu
}
```

Tento sketch pohybuje servomotor pripojený na digitálny pin D2 dookola na pozíciu 0, potom na pozíciu 90 v sekundovom intervale. To znamená, že sa motor pohybuje 90 stupňov tam a naspäť každú sekundu. Na začiatku však musíme vytvoriť premennú typu „Servo“ a zavolať jej metódu „attach“ s parametrom čísla pinu, na ktorý máme motor pripojený.

Príklad pre jednoduchý sketch, ktorý číta hodnoty z analógového pinu a môže byť použitý napríklad na určenie sily svetla, ktoré dopadá na fotosenzor, vyzerá nasledovne:

```
void setup() {
  Serial.begin(9600);               // inicializuje sériovú
                                    // komunikáciu s rýchlosťou
                                    // 9600 bitov za sekundu
}

void loop() {
  int sensorValue = analogRead(A0); // prečíta hodnotu
                                    // z analógového pinu A0
  Serial.println(sensorValue);      // vypíše hodnotu na sériový monitor
  delay(1);                         // zastaví sa na 1 ms pre stabilitu
}
```

Kombináciou týchto jednoduchých programov je možné vytvoriť sketch, ktorý ovláda jedinca s fotosenzormi. Takýto jedinec je schopný vnímať svetlo, ktoré dopadá na jeho senzory, čo znamená, že toto svetlo môže ovplyvniť jeho správanie. Tento spôsob nám umožní reálne vytvorenie jedinca, ktorý je napríklad schopný otáčať sa za svetlom.

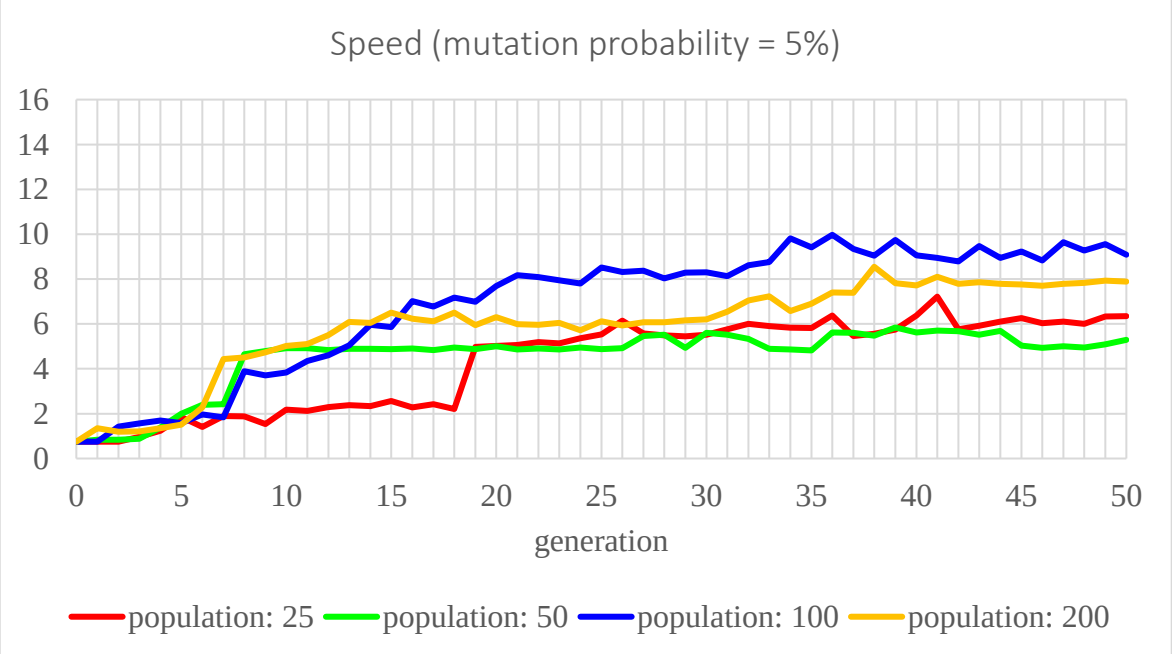
# 4 Popis experimentov

## 4.1 Porovnanie parametrov evolúcie

Pre jednotlivé úlohy evolúcie bol vykonaný experiment za účelom nájsť čo najideálnejšie parametre. Porovnané boli rôzne kombinácie parametra veľkosti populácie a parametra pravdepodobnosti mutácie.

### 4.1.1 Prekonanie vzdialenosti

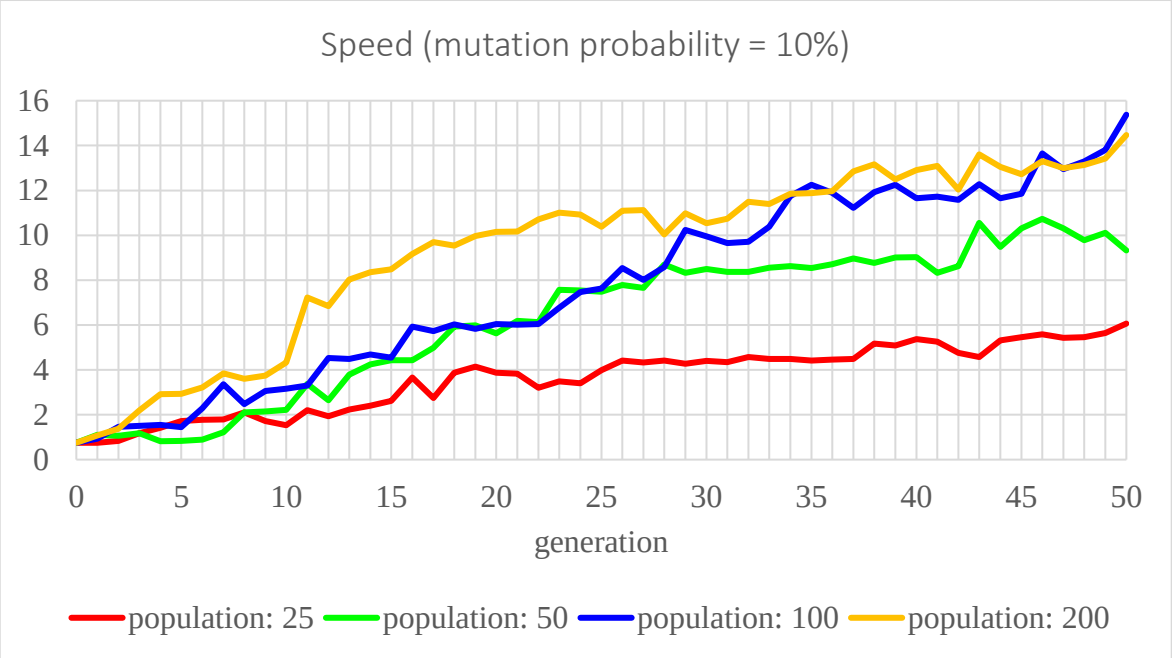
Pri experimente pre úlohu prekonania čo najväčšej vzdialenosti bol nastavený parameter pravdepodobnosti mutácie na hodnoty 5%, 10%, 20% a 30%. Ako parameter veľkosti populácie boli použité hodnoty 25, 50, 100 a 200. Výsledky experimentu boli zaznamenané počas 50-tich generácií.




*Graf 1 – Výsledky experimentu porovnania rozličných hodnôt parametra veľkosti populácie pre úlohu prekonania vzdialenosti s parametrom pravdepodobnosti mutácie nadobúdajúceho hodnotu 5%*

Ako si môžeme všimnúť aj na grafe vyššie, pri 5%-nej pravdepodobnosti mutácie dostávame najlepšie výsledky pri veľkosti populácie nastavenú na 100 jedincov. Avšak ani v tom prípade jedinci nenadobúdajú vhodnosť vyššiu ako 10. Taktiež môžeme usúdiť, že po

uplynutí približne 35-tich generácií sa výsledky ďalej nezlepšujú. Dôvodom tohto javu je, že príliš nízka pravdepodobnosť mutácie neumožní vyvíjaným jedincom prejsť cez dostatočne výrazné zmeny, aby sa boli schopní ďalej vylepšovať. Pre lepšie výsledky je preto nutné zvýšiť hodnotu parametra pravdepodobnosti mutácie.

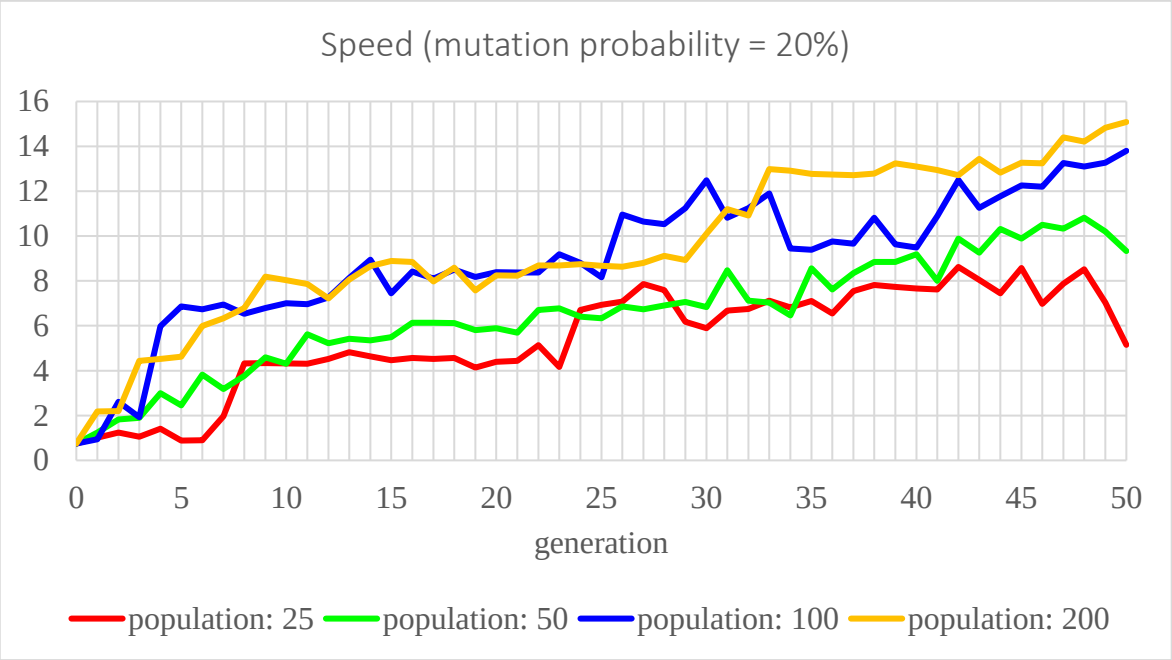



*Graf 2 – Výsledky experimentu porovnania rozličných hodnôt parametra veľkosti populácie pre úlohu prekonania vzdialenosti s parametrom pravdepodobnosti mutácie nadobúdajúceho hodnotu 10%*

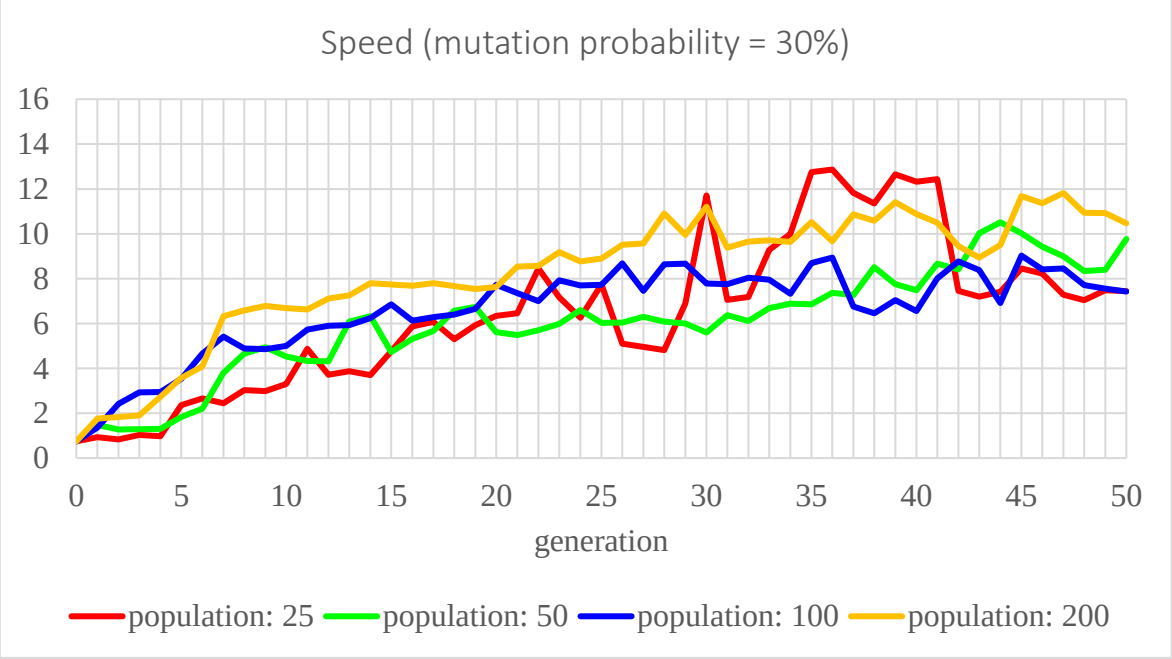
Z tohto grafu je zrejmé, že zvýšenie hodnoty parametra pravdepodobnosti mutácie výrazne zlepšilo výsledky evolúcie. Môžeme si všimnúť aj, že vyššia veľkosť populácie zväčša tiež spôsobuje úspešnejšiu evolúciu. Pri použití 100 a 200 jedincov je konečný výsledok veľmi podobný, ale na začiatku evolúcie sa ukázalo, že vyšší počet umožní rýchlejší vývoj. To znamená, že ak chceme dostať čo najrýchlejšie pomerne vhodného jedinca, oplatí sa použiť väčšiu populáciu.

Pri použití ešte vyššej hodnoty parametra pravdepodobnosti mutácie sa výsledky ďalej nezlepšili. Na nasledujúcich grafoch vidíme tento parameter nadobúdať hodnoty 20% a 30%. Pri 20%-nej pravdepodobnosti mutácie sa konečné výsledky výrazne nezmenili, ale stabilita evolúcie sa očividne znížila, čo znamená, že v niektorých prípadoch vhodnosť vyvíjaných jedincov klesá. Tento jav je spôsobený tým, že kvôli vyššej pravdepodobnosti mutácie jedinci prechádzajú cez výraznejšie zmeny, čo môže ich vhodnosti aj uškodiť. Preto máme menšiu istotu, že dosiahneme očakávaný výsledok evolúcie. Pri 30%-nej pravdepodobnosti

mutácie klesla stabilita ešte viac, čo spôsobilo, že konečné výsledky evolúcie sú už horšie, ako pri predošlých dvoch prípadoch. Na druhej strane, evolúcia s najmenším počtom jedincov, ktorá bola tentoraz najúspešnejšia, dosiahla doteraz najvyššiu vhodnosť.



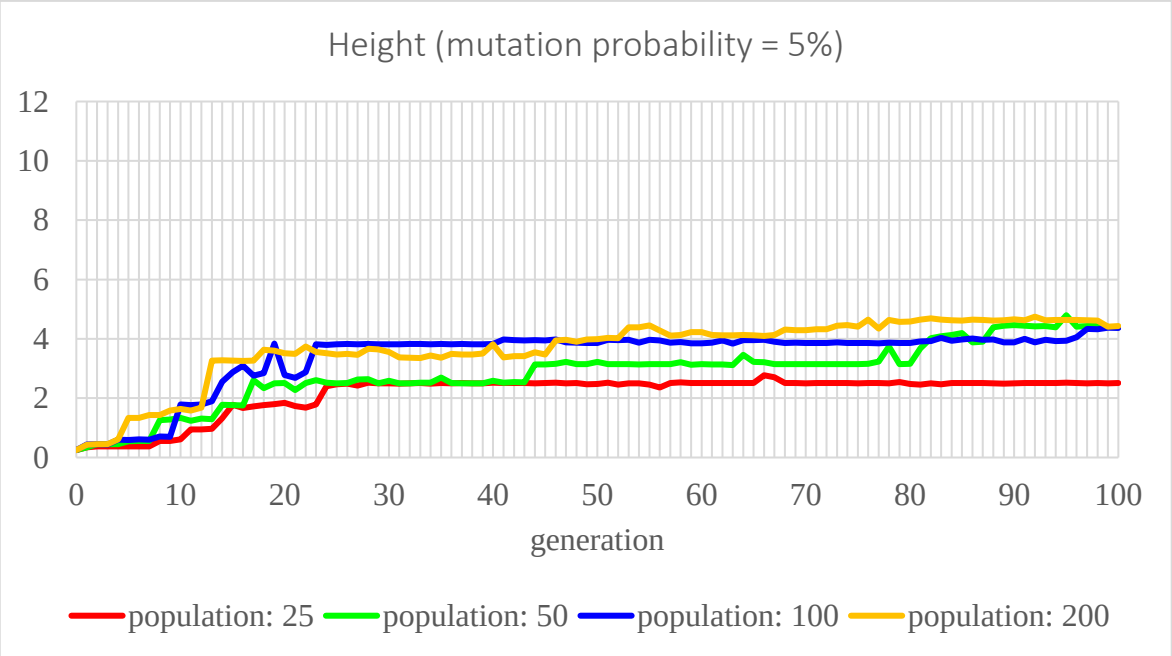

*Graf 3 – Výsledky experimentu porovnania rozličných hodnôt parametra veľkosti populácie pre úlohu prekonania vzdialenosti s parametrom pravdepodobnosti mutácie nadobúdajúceho hodnotu 20%*




*Graf 4 – Výsledky experimentu porovnania rozličných hodnôt parametra veľkosti populácie pre úlohu prekonania vzdialenosti s parametrom pravdepodobnosti mutácie nadobúdajúceho hodnotu 30%*

### 4.1.2 Dosiahnutie výšky

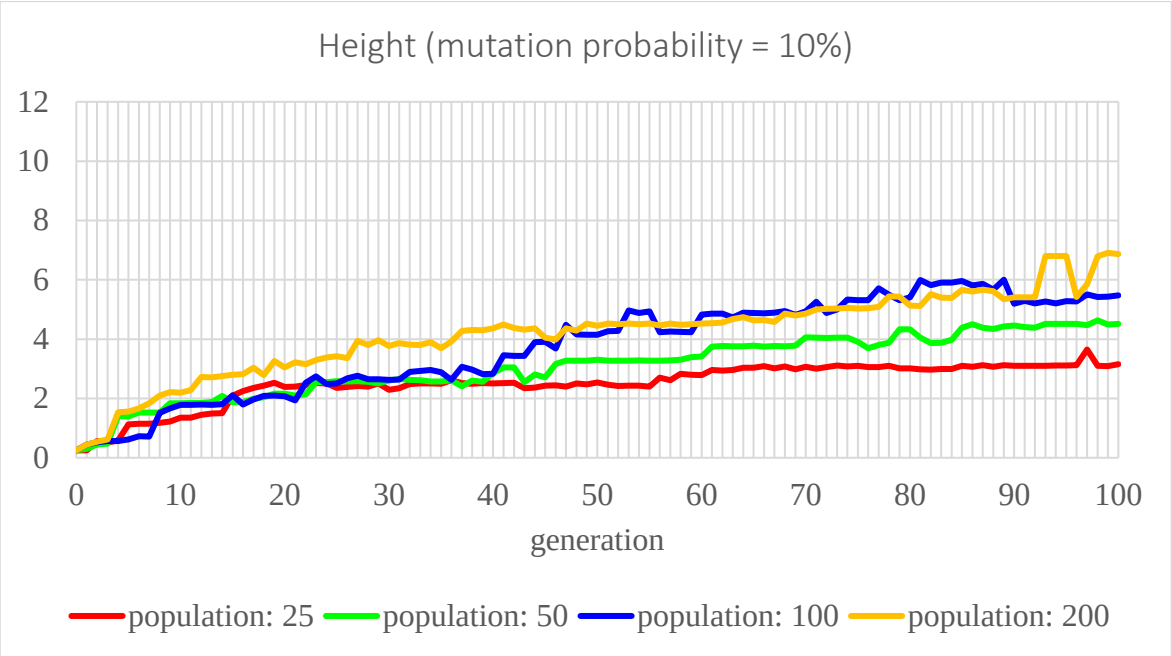
Pri tomto experimente boli porovnané rovnaké hodnoty parametrov, ako pri úlohe prekonania čo najväčšej vzdialenosti. Avšak v tomto prípade boli výsledky zaznamenané počas 100 generácií namiesto 50.




*Graf 5 – Výsledky experimentu porovnania rozličných hodnôt parametra veľkosti populácie pre úlohu dosiahnutia výšky s parametrom pravdepodobnosti mutácie nadobúdajúceho hodnotu 5%*

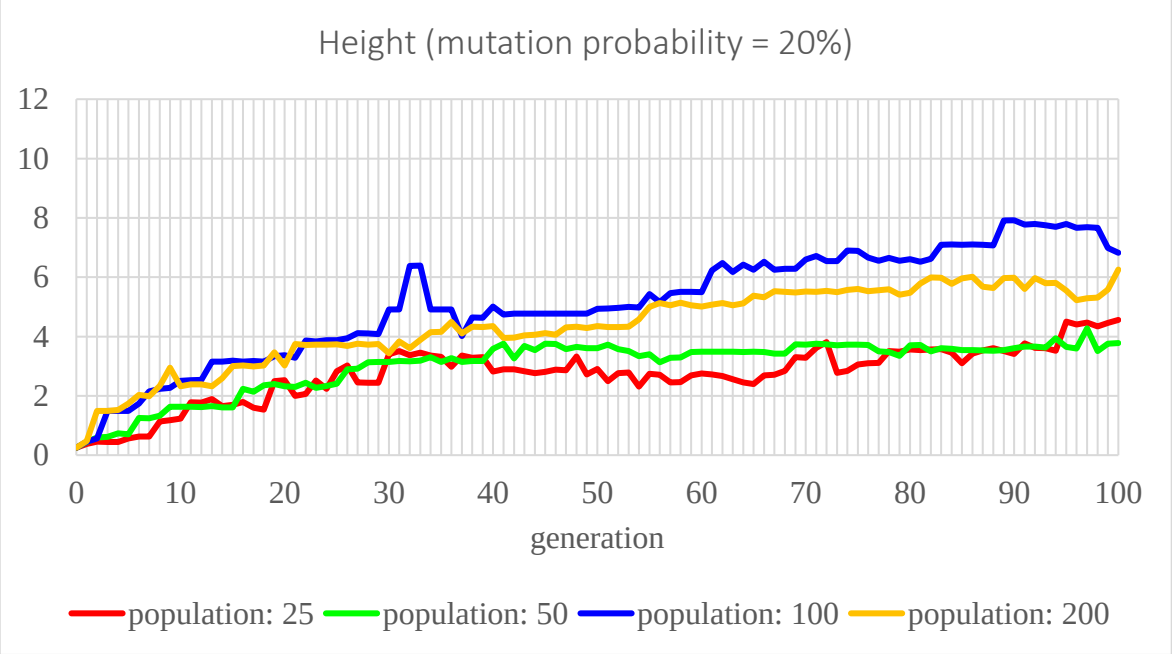
Pri použití 5%-nej pravdepodobnosti mutácie boli výsledky podobne slabé, ako pri predošlom experimente. Populácia pozostávajúca z 25-tich jedincov priniesla najhoršie výsledky. Ostatné veľkosti populácie skončili s lepšími jedincami, ktorí mali veľmi podobnú vhodnosť. Avšak vyššie hodnoty počtu jedincov začali dosahovať lepšie výsledky rýchlejšie, ako tie nižšie. Tento jav je spôsobený tým, že vo väčšej populácii sa prirodzene nájde ľahšie jedinec s vyššou vhodnosťou, ako v menšej populácii.

Pri použití 10%-nej pravdepodobnosti sa tiež neukázali oveľa lepšie výsledky, aj keď pri tejto fáze experimentu začala mať veľkosť populácie výraznejší vplyv na priebeh evolúcie. Ako si môžeme ľahko všimnúť na grafe nižšie, čím vyšší počet jedincov sa použil, tým vyššiu vhodnosť dosiahli po uplynutí 100 generácií. Počas prvých zhruba 40 generácií však jediný výrazný rozdiel spôsobilo použitie populácie obsahujúcej 200 jedincov. Dovtedy sa všetky menšie populácie vyvíjali podobnou, nižšou rýchlosťou.



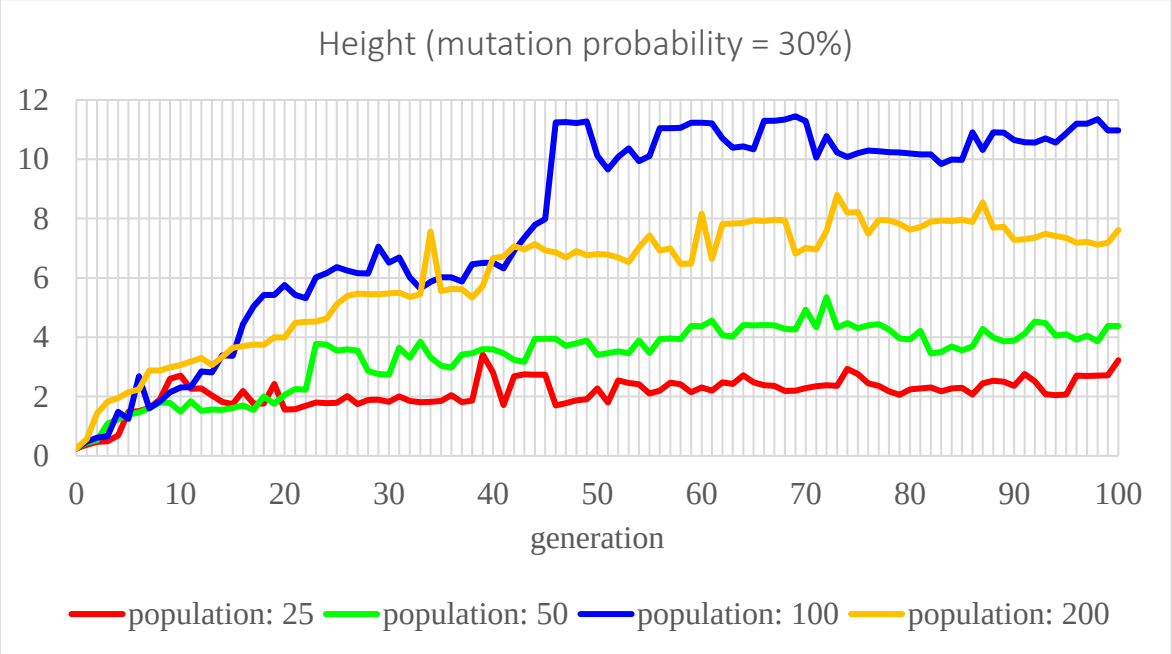

*Graf 6 – Výsledky experimentu porovnania rozličných hodnôt parametra veľkosti populácie pre úlohu dosiahnutia výšky s parametrom pravdepodobnosti mutácie nadobúdajúceho hodnotu 10%*

Na rozdiel od predošlého experimentu, pri použití 20%-nej pravdepodobnosti mutácie stále nevidíme výrazné zlepšenie. Najľahšie si všimneme, že priebeh evolúcie stratil stabilitu oproti fázam s nižšou pravdepodobnosťou mutácie.




*Graf 7 – Výsledky experimentu porovnania rozličných hodnôt parametra veľkosti populácie pre úlohu dosiahnutia výšky s parametrom pravdepodobnosti mutácie nadobúdajúceho hodnotu 20%*

Ako vidíme na grafe nižšie, ďaleko najideálnejšie výsledky sme dostali pri použití 30%-nej pravdepodobnosti mutácie a populácie pozostávajúcej zo 100 jedincov. Hoci na úplnom začiatku evolúcie sa ukázala byť väčšia populácia úspešnejšia, z celého priebehu môžeme jednoznačne usúdiť, že populácia, ktorá obsahovala iba 100 jedincov, dosiahla výrazne lepšie výsledky, ako tá väčšia.

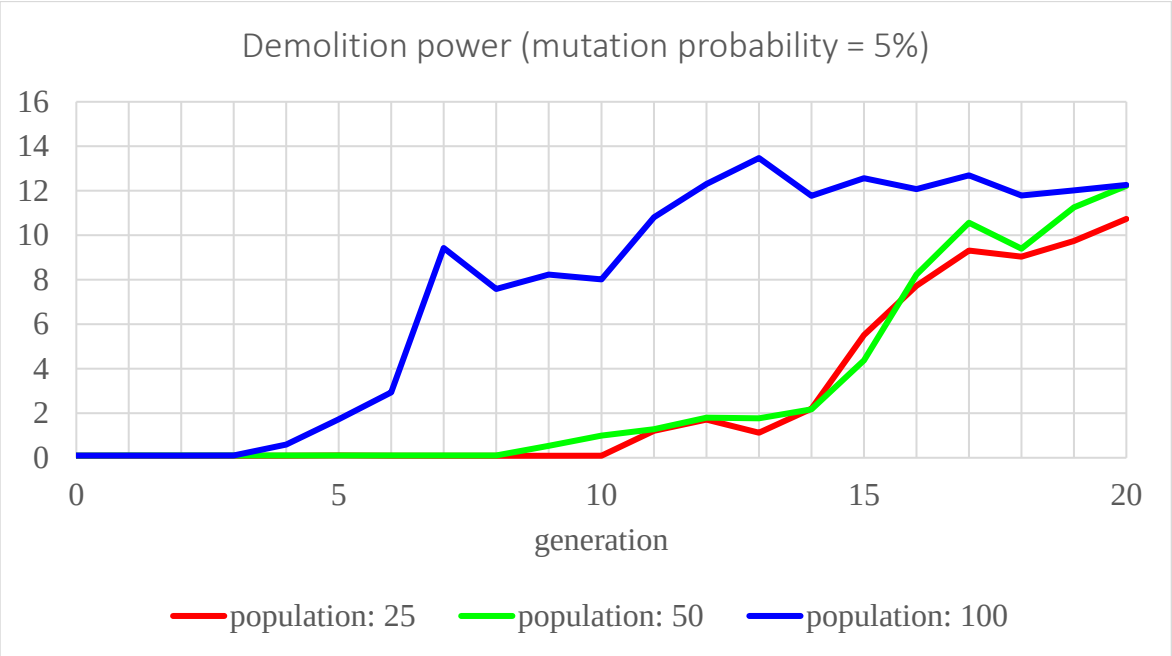



*Graf 8 – Výsledky experimentu porovnania rozličných hodnôt parametra veľkosti populácie pre úlohu dosiahnutia výšky s parametrom pravdepodobnosti mutácie nadobúdajúceho hodnotu 30%*

### 4.1.3 Demolácia objektov

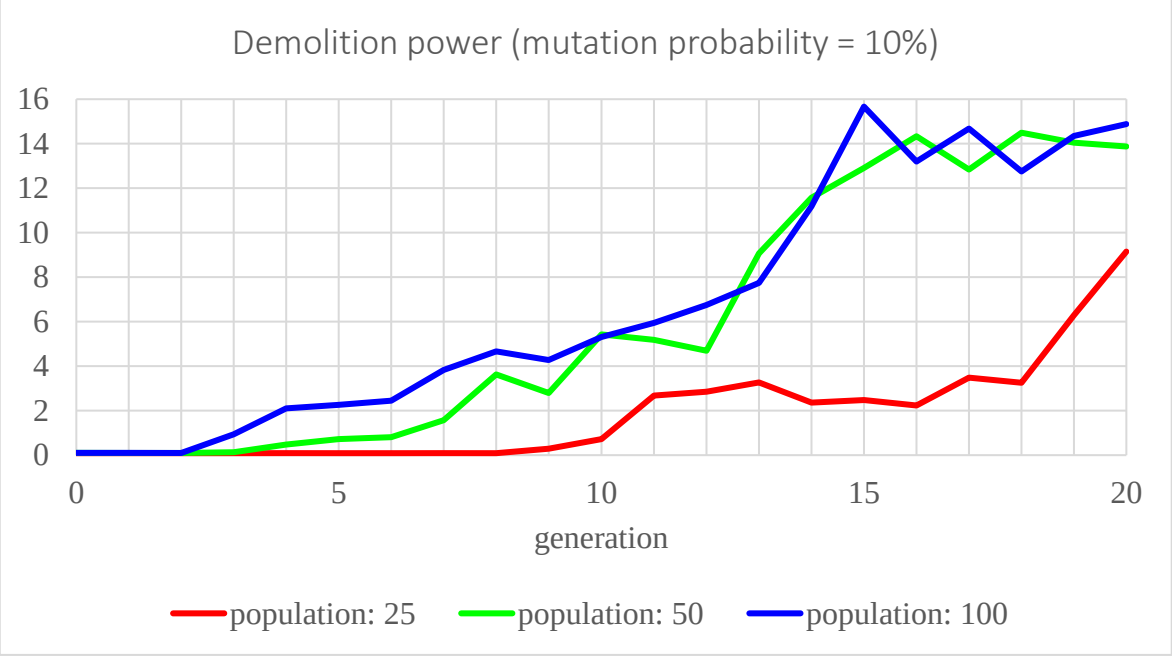
Pri tomto experimente na rozdiel od prvých dvoch nebola použitá populácia pozostávajúca z 200 jedincov. Použité boli iba veľkosti 25, 50 a 100. Evolúcia trvala iba 20 generácií, keďže pri tejto úlohe sa rýchle vyvinú skoro dokonalí jedinci a neskôr by už neprichádzalo k žiadnym relevantným zmenám. Hodnoty parametra pravdepodobnosti mutácie boli použité rovnaké, ako pri experimentoch pre ostatné úlohy.

Na grafe nižšie vidíme, že už aj pri použití 5%-nej pravdepodobnosti mutácie sme dosiahli pomerne ideálne výsledky. Veľkosť populácie neovplyvnila výrazne konečné výsledky. Avšak na začiatku priebehu evolúcie bola populácia veľkosti 100 oveľa úspešnejšia, ako tie menšie. To znamená, že ak chceme pri tejto hodnote parametra pravdepodobnosti mutácie dosiahnuť lepšie výsledky čo najskôr, oplatí sa použiť populácia s vyšším počtom jedincov.



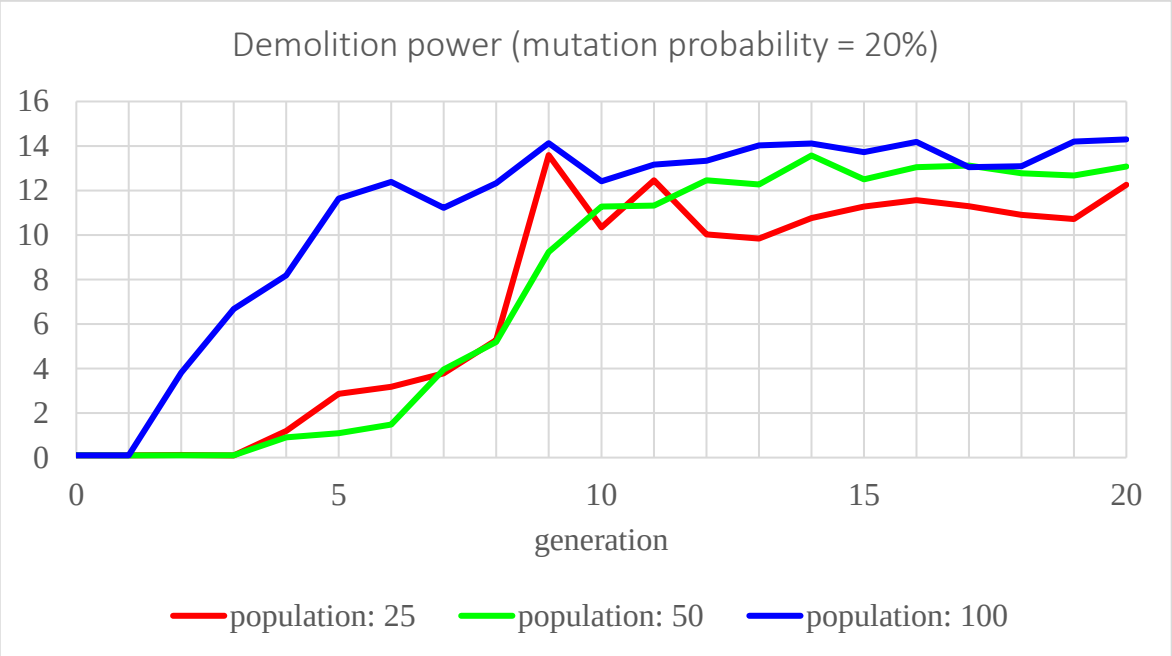

*Graf 9 – Výsledky experimentu porovnania rozličných hodnôt parametra veľkosti populácie pre úlohu demolácie objektov s parametrom pravdepodobnosti mutácie nadobúdajúceho hodnotu 5%*

Pri použití 10%-nej pravdepodobnosti mutácie bola populácia veľkosti 50 podobne úspešná, ako väčšia populácia pozostávajúca z dvojnásobného počtu jedincov. To znamená, že vďaka vyššej pravdepodobnosti mutácie nám stačí polovičná hodnota veľkosti populácie.

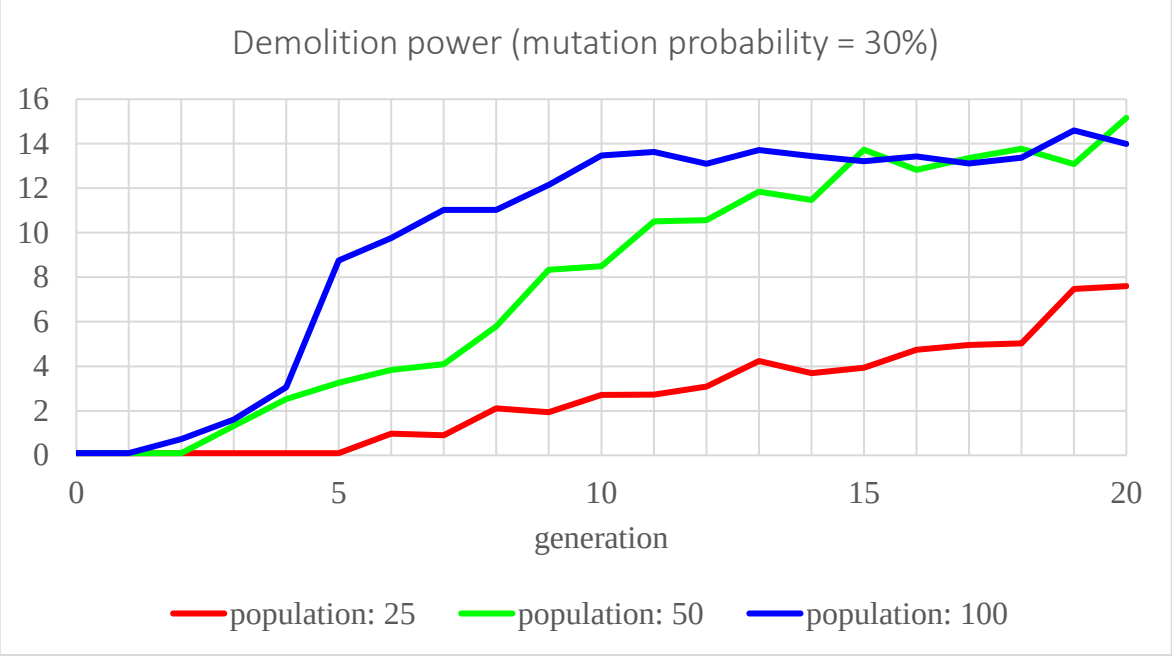



*Graf 10 – Výsledky experimentu porovnania rozličných hodnôt parametra veľkosti populácie pre úlohu demolácie objektov s parametrom pravdepodobnosti mutácie nadobúdajúceho hodnotu 10%*

Ako si aj môžeme všimnúť na grafe nižšie, po ďalšom zvýšení pravdepodobnosti mutácie až na hodnotu 20% sme dostali ešte ideálnejšie výsledky na začiatku evolúcie pri väčšej populácii. Po zvýšení pravdepodobnosti mutácie na hodnotu 30% sa ale výsledky zhoršili.




*Graf 11 – Výsledky experimentu porovnania rozličných hodnôt parametra veľkosti populácie pre úlohu demolácie objektov s parametrom pravdepodobnosti mutácie nadobúdajúceho hodnotu 20%*

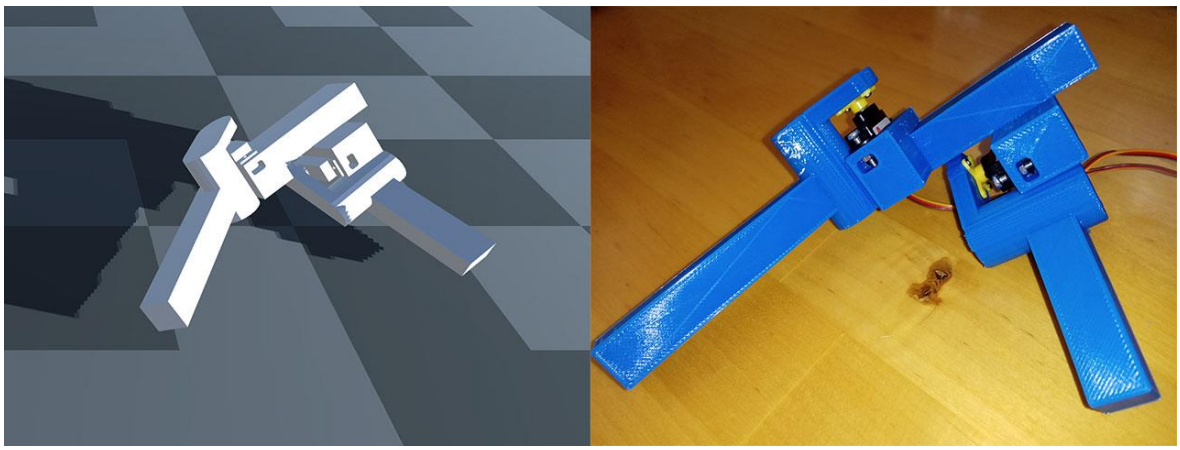



*Graf 12 – Výsledky experimentu porovnania rozličných hodnôt parametra veľkosti populácie pre úlohu demolácie objektov s parametrom pravdepodobnosti mutácie nadobúdajúceho hodnotu 30%*

## 4.2 Porovnanie simulácie s realitou

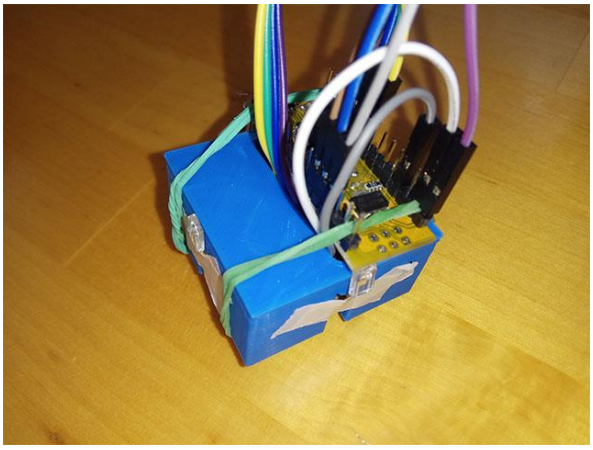
### 4.2.1 Popis testovania vytlačeného jedinca

Testovaný bol jedinec vyvinutý pomocou nášho evolučného softvéru. Mal nastavenú úlohu otáčania za svetlom. To znamená, že musí určiť polohu zdroja svetla, ktoré naňho svieti. Taktiež podľa toho musí byť schopný otáčať sa do obidvoch smerov. Na obrázku nižšie vidíme spomínaného jedinca vo virtuálnom svete a vo vytlačenej podobe.



*Obr. 24 – Testovaný jedinec vytvorený na 3D tlačiarni vedľa pôvodného jedinca vyvinutého v simulácii*

Na testovanie jedinca vytvoreného na 3D tlačiarni bola vytvorená „hlava“, ktorá obsahuje štyri fotosenzory, jeden na každej strane. Tieto senzory simulujú oči jedinca potrebné na to, aby bol schopný určiť polohu zdroja svetla, na základe ktorej sa bude pohybovať.



*Obr. 25 – „Hlava“ jedinca vytvoreného na 3D tlačiarni, ktorá určuje polohu svetla pomocou fotosenzorov*

### 4.2.2 Výsledky testovania vytlačeného jedinca

Po úspešnom vytvorení testovaného jedinca na 3D tlačiarni a jeho následnom osadení servomotormi sa jedinec nesprával vôbec tak, ako mohlo byť očakávané podľa výsledkov simulácie. Po jednoduchom upravení sa však podarilo dosiahnuť, aby sa jedinec bol schopný otáčať do jedného smeru podobne, ako vo virtuálnom svete. Rýchlosť otáčania bola stále výrazne nižšia a pohyb dosť nepravidelný, ale cieľ bol splnený.

Porovnanie správania spomínaného jedinca v simulácii a v reálnom svete znázorňuje video napálené na CD, ktoré je prílohou tejto práce.

# 5 Záver

Cieľom tejto bakalárskej práce bolo otestovať, či sa virtuálny tvor vyvinutý pomocou môjho evolučného algoritmu dokáže správať podobne aj v reálnom svete po vytvorení na 3D tlačiarni, ako v simulácii.

Po naprogramovaní evolučného softvéru som vykonával experimenty, aby som našiel čo najideálnejšie hodnoty parametrov pre dosiahnutie očakávaných výsledkov. Ukázalo sa, že na týchto hodnotách naozaj záleží a oplatí sa vyskúšať rôzne kombinácie, aby sme náš algoritmus využili čo najlepšie.

Pre simuláciu jedincov a určenie ich vhodnosti som použil softvér Unity, ktorý obsahuje pomerne realistické fyzikálne prostredie. Napriek tomu sa správanie vybraného jedinca vo vytlačenej podobe výrazne odlišovalo od jeho správania v simulácii. Tento fakt dokazuje, že simulovať reálne fyzikálne podmienky nie je ľahké a ani jeden z najlepšie vypracovaných softvérov nie je zďaleka dokonalý. V budúcnosti sa to ale pravdepodobne môže zmeniť a tvory vytvorené pomocou softvéru, ktoré môžu byť potenciálne užitočné pre ľudstvo, bude možné bez problémov vyvíjať bez ľudskej pomoci.

Tému tejto práce považujem za zaujímavú. Taktiež verím, že môže byť aj užitočná pre vývoj technológie a časom prinesie nové možnosti, ktoré boli predtým takmer nepredstaviteľné. Okrem toho môže byť aj zdrojom zábavy, čo je tiež dôležitá časť nášho života.

# Použitá literatúra


[1] O. Zuzana, O. Pavel, Š. Miloš, V. František a Z. Ivan, Evoluční výpočetní techniky - principy a aplikace, 2008.

[2] K. Sims, „Evolving Virtual Creatures," 1994. [Online]. Available: http://www.karlsims.com/papers/siggraph94.pdf.

[3] M. Komosinski a S. Ulatowski, „Framsticks Manual," 2008. [Online]. Available: http://www.framsticks.com/files/common/Framsticks_Manual.pdf.

[4] „Unity User Manual," Unity Technologies, [Online]. Available: https://docs.unity3d.com/Manual/index.html.

[5] „Visual Studio IDE," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/dn762121.aspx.

[6] „Visual Studio Tools for Unity," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/dn940019.aspx.

[7] „C#," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/kx37x362.aspx.

[8] „Arduino Nano," Arduino AG, [Online]. Available: https://www.arduino.cc/en/Main/arduinoBoardNano.

[9] „Arduino IDE," Arduino AG, [Online]. Available: https://www.arduino.cc/en/Main/Software.

[10] „pb_Stl," [Online]. Available: https://github.com/karl-/pb_Stl.

[11] „Datasheet for the Power HD sub-micro servo HD-1440A,“ HuiDa RC International Inc., [Online]. Available: https://www.pololu.com/file/0J319/HD-1440A.pdf.

[12] „Datasheet for the L-NP-3C1,“ Bright LED Electronics Corp., [Online]. Available: http://www.americanbrightled.com/pdffiles/infrared/BPT-NP03C1.pdf.

# Prílohy

Prílohou tejto práce je CD, ktoré obsahuje skompilovanú aplikáciu spolu so zdrojovým kódom napísaným v programovacom jazyku C#. Na disku sa taktiež nachádza krátke video o porovnaní správania testovaného jedinca v simulácii a v reálnom svete po vytvorení na 3D tlačiarni a osadení servomotormi.