UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Evolutie Fizica prin Algoritmi Genetici

propusă de

Mihaila Andrei

Sesiunea: Iulie, 2019

Coordonator științific

Asist. dr. Eugen Croitoru

UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Evolutie Fizica prin Algoritmi Genetici

Mihaila Andrei

Sesiunea: Iulie, 2019

Coordonator științific

Asist. dr. Eugen Croitoru

Avizat,
Îndrumător Lucrare de Licenţă
Titlul, Numele şi prenumele _____
Data _____ Semnătura _____

DECLARAŢIE privind originalitatea conţinutului lucrării de licenţă

Subsemntatul(a) ……

domiciliul în ……..

născut(ă) la data de ……., identificat prin CNP ……., absolvent(a) al(a) Universităţii „Alexandru Ioan Cuza" din Iaşi, Facultatea de …….. specializarea ……., promoţia ……., declar pe propria răspundere, cunoscând consecinţele falsului în declaraţii în sensul art. 326 din Noul Cod Penal şi dispoziţiile Legii Educaţiei Naţionale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licenţă cu titlul: _____

_____

_____elaborată sub îndrumarea dl. / d-na _____, pe care urmează să o susţină în faţa comisiei este originală, îmi aparţine şi îmi asum conţinutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licenţă să fie verificată prin orice modalitate legală pentru confirmarea originalităţii, consimţind inclusiv la introducerea conţinutului său într-o bază de date în acest scop.

Am luat la cunoştinţă despre faptul că este interzisă comercializarea de lucrări ştiinţifice in vederea facilitării fasificării de către cumpărător a calităţii de autor al unei lucrări de licenţă, de diploma sau de disertaţie şi în acest sens, declar pe proprie răspundere că lucrarea de faţă nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, …… Semnătură student ……

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licenţă cu titlul „*Evolutie Fizica prin Algoritmi Genetici*”, codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iaşi, să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iaşi, *27.06.2019*

Absolvent

_____

Cuprins

Introducere **2**

Motivație **4**

1 Related Work **5**

1.1 Locomotie prin algoritmi evolutivi 5

1.1.1 Cell Evolution 5

1.1.2 Muskoscheletal Evolution 6

1.1.3 Joint Evolution 6

1.1.4 Concluzii 7

1.2 Algoritmi evolutivi in cercetare 8

1.2.1 Evolved Antenna 8

2 Tehnologii Folosite **9**

2.1 C++ 9

2.2 OpenGL 9

2.3 OpenGL Mathematics (GLM) 9

2.4 GLFW 10

2.5 Concluzii 10

3 Mediul Fizic **12**

3.1 Integrarea ecuatiilor miscarii dupa dt 12

3.1.1 Moduri de integrare 13

3.2 Rigid Bodies 14

3.3 Coliziuni 14

3.3.1 Intersectie intre doua sfere 15

3.3.2 Intersectie intre doua triunghiuri 15

3.4 Efectul unei coliziuni 15

4 Algoritmul Genetic **17**

4.1 Cromozom 17

4.2 Fitness 18

4.3 Selectie 18

4.4 Crossover 18

4.5 Mutatie 19

5 Rezultate **20**

5.1 Optim fix 20

5.2 Optim dinamic 21

Concluzii **23**

5.3 Future Work 23

Bibliografie **25**

Introducere

Algoritmii genetici sunt un tip al algoritmilor de optimizare. Acestia cauta solutii optime intr-un spatiu larg,unde algoritmii generici ar fi prea greoi, sau unde nu exista un algoritm determinist. Acestia cauta sa minimizeze sau sa maximezeze o anumita functie data. Algoritmii genetici folosesc principiile selectiei si evolutiei, observate in natura pentru a produce mai multe solutii unei singure probleme. Datele de intrare ale unui algoritm genetic sunt un set de posibile solutii, iar apoi algoritmul genetic incearca sa imbunateasca solutiile pe parcursul a sute sau mii de generatii. Prin evolutie se presupune ca fiecare generatie este mai buna decat precedenta, astfel dupa multe iteratii algoritmii genetic pot produce solutii noi, eficiente care in mod normal nu ar putea fi gasite de mintea umana. Un astfel de exemplu sunt antele create de un algoritm genetic, (detaliat in Capitolul 1.2.1 Evolved Antenna).

Acest tip de algoritmi au aparut prima data in 1950 cand Alan Turing a propus o ”Masina care invata” ce ar imita principiile evolutiei. Mai tarziu in anii 1989 a aparut primul algoritm genetic comercializat. Acesta rezolva probleme de optimizare date de utilizator.

Intelegerea si replicarea evenimentelor fizice este un subiect in continua cercetare in ziua de azi. Pentru a intelege cat mai bine natura, algoritmii evolutivi au inceput sa creasca nivelul de abstractizare cat mai mult pentru a obtine rezultate cat mai fidele cu ceea ce vedem in natura. De exemplu pentru a simula locomotia, optimizarea modului in care este controlata o incheietura (1.1.3 Joint Evolution) aduce rezultate modeste, rigide care nu par naturale. Asa ca dupa incheieturi a fost abstractizat sistemul muscular (1.1.2 Muskoscheletal Evolution) iar apoi pana si celulele ce creeaza un corp (1.1.1 Cell Evolution) pentru a obtine cele mai fidele rezultate prin evolutie. In 2017 Google Deep Mind AI [1] au reusit sa creeze un individ care alearga pe cont propriu, avand un corp similar corpului uman.

[1] https://www.youtube.com/watch?v=gn4nRCC9TwQ

Aceasta lucrare isi propune sa rezolve o problema de optimizare folosind algoritmi genetici pentru a nimeri o tinta intr-un mediu fizic similar cu cel terestru. Adaugand obstacole atat statice cat si dinamice, problema de optimizare poate deveni o problema cu optim dinamic, in care pot exista mai multe solutii eficiente. Se cauta control total atat asupra evolutiei indivizilor cat si a mediuliu fizic in care acestia cauta solutia, astfel nu se va folosi nici un fel de librarie deja existenta pentru suport fizic sau algoritmi genetici.

Algoritmul genetic incearca sa rezolve problema prin a aplica un vector forta unui corp ce are o anumita elasticitate, iar apoi acesta imbunatateste solutia evoluand vectorul forta si elasticitatea obiectului. Incepem intai prin a defini mediul fizic (3. Mediul Fizic) apoi algoritmul genetic (4.Algoritmul Genetic) si rezultate acestuia in urma experimentelor (5. Rezultate).

Contributii

Pentru a atinge scopul propus, a fost intai nevoie de dezvoltarea unui motor fizic ce poate aproxima toate actiunile fizice care pot aparea in cadrul unei simulari.

Incat acesta este responsabil de functia fitness, motorul fizic trebuie sa poate rula atat intr-un mod in care putem vizualiza o intreaga generatie, sau un singur individ (head), cat si intr-un mod in care acesta returneaza fitnessul cat mai rapid posibil algoritmului genetic, vizualizarea generatiilor nefiind necesara in pasul de evolutie.

In cadrul algoritmului genetic a fost cautat cel mai eficient mod de a crea un cromozom si diferite moduri in care acesta sa evolueze astfel incat sa rezolve problema indifierent de mediul in care acesta se gaseste. A fost urmarita schimbarea in generatii in functie de selectia folosita, recombinarea cat si mutatia.

Capitolul 1

Related Work

In prezent, exista mai multi algoritmi genetici ce sunt devoltati pentru a perfectiona o actiune fizica sau pentru a optimiza probleme prin selectie naturala.

1.1 Locomotie prin algoritmi evolutivi

1.1.1 Cell Evolution

O incercare de a simula evolutia locomotiei de la un nivel celular apare in articolul **Fine grained artificial development for body-controller coevolution of soft-bodiedanimats** [1] [2].

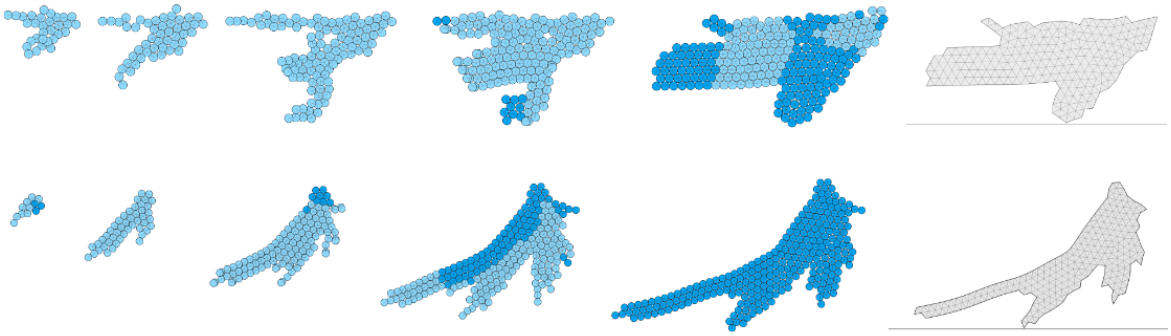
Procesul evolutionar incepe de la o singura celula ce sufera divizie celulara. O celula este reprezentata de un disc ce sufera coliziuni elastice simulate prin arcuri cu alte celule. O celula are o pozitie, viteza, si o orientare ce reprezinta directia diviziunii celulare. O celula se poate "contracta" iar compartamenul acesteia este controlat de o retea neuronala si poate fi afectata de celulele din jur.

Pentru procesul de evolutie a fost folosit **NEAT**[3], *neuroevolution of augmenting topologies Stanley and Miikkulainen, 2002*. Populatia aleasa a fost $\lambda = 300$ cu genomi de lungimi arbitrare si 2000 de generatii.

[1] https://www.mitpressjournals.org/doi/pdf/10.1162/978-0-262-32621-6-ch040

[2] https://www.youtube.com/watch?v=CXTZHHQ7ZiQ

[3] http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf



source: `https://www.mitpressjournals.org/doi/pdf/10.1162/978-0-262-32621-6-ch040`

video example:

`https://www.youtube.com/watch?v=Q6XXohgPP4A&feature=youtu.be`

Evolutia unui embrion rezultat. Fiecare celula luminoasa reprezinta o celula ce este pregatita de divzie celulara.

1.1.2 Muskoscheletal Evolution

Articolul **Flexible Muscle-Based Locomotion for Bipedal Creatures**[4] ce a simulat locomotia prezinta un algoritm evolutiv ce foloseste CMA-ES [5] (*covariance matrix adaptation evolution strategy*) cu o populatie $\lambda = 20$ si step-size $\sigma = 1$. Modul in care au optimizat problema a fost de a minimiza eroarea $E(K)$:

$$E(K) = E_{speed} + E_{headori} + E_{headvel} + E_{slide} + E_{effort}$$

Acestia reusesc sa optimizeze un set de muschi si incheieturi 3D pentru a obtine indivizi finali, care pot ajunge la o viteza dorita, fac fata unui teren inegal si a evenimentelor externe.

1.1.3 Joint Evolution

Un alt algoritm genetic, numit **Genetic Walkers** [6] incearca sa simuleze o miscare in 2D. Spre deosebire de abordarea anterioara acest algoritm optimizeaza distanta dintre *head* fata de *feet*, distanta pe care fiecare individ reuseste sa o parcurga. De asemenea

[4] `https://www.cs.ubc.ca/~van/papers/2013-TOG-MuscleBasedBipeds/2013-TOG-MuscleBasedBipeds.pdf`

[5] `https://en.wikipedia.org/wiki/CMA-ES`

[6] `https://rednuht.org/genetic_walkers/`

fiecare individ primeste 100 de puncte in fitness pentru fiecare pas corect realizat. Spre deosebire de primul articol, selectia se realizeaza prin o functie f, $f(k) = \frac{1}{k+2}$, unde k este indexul unui cromozom, ordonati in ordinea fitnessului. Primii doi cromozomi alesi vor realiza un copil in generatia urmatoare. Optimizarea se face asupra unghiului unui joint si a duratei de timp.

1.1.4 Concluzii

Ambii algoritmi lucreaza cu un corp 3D fixat de la inceput, dar evolueaza miscarea in moduri diferite, unul lucureazaza low-level, cu un schelet muscular iar altul strict cu rotatia unui joint. Algoritmul low level produce rezultate [7] mult mai fidele in timp ce al doilea algoritm reuseste sa faca 3 pasi dupa 500 de generatii.

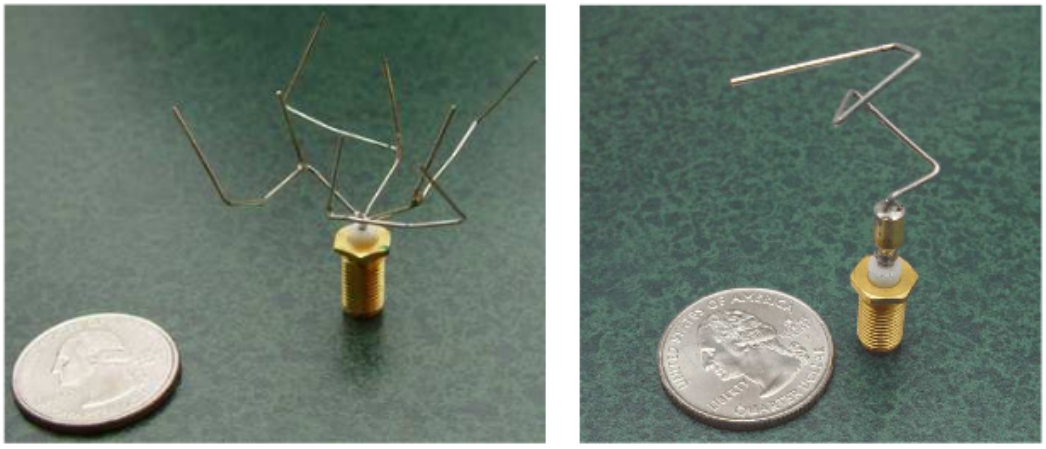
[7] https://www.youtube.com/watch?v=pgaEE27nsQw

1.2 Algoritmi evolutivi in cercetare

1.2.1 Evolved Antenna

In radio comunicatii, o **Antena Evoluata** [8] repreza o antena proiectata complet sau partial de un algoritm genetic.

Pentru a gasi solutii pentru probleme noi, cum ar fi detectarea anumitor radiatii, au fost incercati algoritmii genetici, iar in 2006, in cadrul misiunii *Space Technology 5* [9], a fost dezvoltata prima antena folosita in spatiu. Urmatoarele doua antene au fost proiectate de catre un algoritm genetic:



(a) (b)

source: *https://ti.arc.nasa.gov/m/pub-archive/1244h/1244%20(Hornby).pdf*

[8] https://en.wikipedia.org/wiki/Evolved_antenna

[9] https://ti.arc.nasa.gov/m/pub-archive/1244h/1244%20(Hornby).pdf

Capitolul 2

Tehnologii Folosite

Pentru implementarea propriu-zisa a licentei, a fost utilizat un anumit set de tehnologii, care sa ajute la implementarea motorului fizic si a algoritmului genetic de la un nivel cat mai primitiv, de unde sa poate fie modelat orice tip de comportament dorit.

2.1 C++

Mediul de lucru este C++17 [1]sub Visual Studio 2019. C++ a fost ales in principal datorita performantei si a experientei anterioare. Prin intermediul OOP au fost create structuri pentru o implementare cat mai naturala a motorului fizic si a algoritmului genetic.

2.2 OpenGL

Reprezentarea grafica se realizeaza cu ajutorul librariei OpenGL 3.3[2]. Motivatia librariei OpenGL a fost atat introducerea ei in Anul 3 cat si permitarea implementarii cat mai primitiva a motorului fizic.

2.3 OpenGL Mathematics (GLM)

Fiind un motor fizic, este necesar ca toate calculele matematice sa fie realizate cat mai corect si rapid, astfel libraria GLM [3] a fost aleasa pentru viteza si compatibilitatea

[1] http://www.cplusplus.com/

[2] https://www.opengl.org/

[3] https://glm.g-truc.net/0.9.9/index.html/

ei cu OpenGL.

Principalele structuri folosite din GLM au fost:

- glm::vec2/glm::vec3/glm::vec4
- glm::quat, (glm quaternions). Motivatia quaternionilor in favoarea matricelor de rotatie a fost performanta si evitarea problemelor unghiurilor euler, cum ar fi Gimbal Lock [4]
- glm::mat3/glm::mat4

Librarii alternative au fost: Eigen [5] si CML [6]. Fiecare librarie aduce un anumit punct forte, GLM a fost ales pentru compatibilitatea cu OpenGL. Rezultatele exacte sunt urmatoarele:

	laptop	laptop (SSE)	armeabi	armeabi-v7a	armeabi-v7a with neon
Eigen additions	8065	30	9944	2181	2145
Eigen multiplications	22404	86	59460	5143	5113
GLM additions	2375	76	10256	1506	1407
GLM multiplications	7337	400	59008	2189	3108
CML additions	12336	96	9587	2885	2996
CML multiplications	21603	551	58399	5306	5280

source: https://github.com/mfoo/Math-Library-Test [7]

2.4 GLFW

Pentru initializarea ferestrei OpenGL a fost folosit GLFW [8]. Aceasta librarie ofera in mod simplistic un context OpenGL si mai multe optiuni cum ar fi VSync sau DoubleBuffering. De asemenea, keyboard si mouse events sunt capturate prin GLFW.

Alternative au fost : GLUT [9]. GLFW a fost ales in favoarea GLUT deoarece pe masina unde a fost dezvoltata aplicatia, GLUT nu reusea sa intanstieze OpenGL 3.3+.

2.5 Concluzii

C++ impreuna cu OpenGL si GLM ofera un mediu de lucru cat mai aproape de procesor si in acelasi timp un set de structuri si unelte pentru dezvoltarea naturala a

[4] https://en.wikipedia.org/wiki/Gimbal_lock/
[5] http://eigen.tuxfamily.org/index.php?title=Main_Page
[6] https://github.com/demianmnave/CML/wiki/The-Configurable-Math-Library
[7] https://github.com/mfoo/Math-Library-Test
[8] https://www.glfw.org/
[9] http://freeglut.sourceforge.net/

Motorului Fizic. Pentru algoritmului genetic, C++ contine deja tot de ce este nevoie si permite implementarea unui algoritm genetic rapid, ”from scratch”.

Capitolul 3

Mediul Fizic

Algoritmul genetic isi va optimiza problema in functie de mediul fizic oferit. Asta inseamna ca putem simula orice conditii fizice atat timp cat le putem reproduce. In cazul curent a fost simulat un mediu terestru cu $g = 9.8m/s$. Mediul fizic incearca sa aproximeze fenomene fizice naturale folosind formule matematice.

3.1 Integrarea ecuatiilor miscarii dupa dt

Stiind ca $F = ma$ atunci putem scoate acceleratia unui obiect in functie de masa acestuia: $a = F/m$. Daca integram dupa timp atunci $dv/dt = a$ dar nu uitam ca $a = F/m$ deci putem spune ca viteza unui obiect va fi rata de schimbare a pozitiei acestuia in functie de timp $dx/dt = v$.

Punand tot cap la cap obtinem:

$$a = F/m$$
$$dv = a * dt$$
$$dx = dv * dt$$

Deci daca vrem sa obtinem pozitia unui obiect dupa un anumit dt vom calcula in urmatorul mod:

$$a = F/m$$
$$v = v + a * dt$$
$$pos = pos + v * dt$$

Acesta este modul de integrare Semi-Implicit Euler.

3.1.1 Moduri de integrare

Exista mai multe moduri te integrare, cel prezentat mai sus este Semi-Implicit Euler.

Explicit euler integreaza pozitia inaintea vitezei si anume:

$$dx = dv * dt$$
$$a = F/m$$
$$dv = a * dt$$

Astfel apare o eroare in integrarea vitezei, iar aceasta depinde de timp. De exemplu aplicam o forta $F = 10N$ asupra unui corp de masa m, $m = 1kg$ timp de 10 secunde ($t = 10s$). Folosind formula miscarii uniforme $x = x_0 + v_0t + \frac{at^2}{2}$ atunci $x = 500$, deci conform formulei, corpul se va misca 500 de unitati. Rezultatele integratorilor difera si anume:

Eroarea integratorului Explicit Euler creste in functie de dt ales. Pentru $dt = 1.0$:

```
t = 1,  position = 0
t = 2,  position = 10
t = 3,  position = 30
t = 4,  position = 60
t = 5,  position = 100
t = 6,  position = 150
t = 7,  position = 210
t = 8,  position = 280
t = 9,  position = 360
t = 10,  position = 450
```

La fel si pentru Semi-Implicit Euler

```
t = 1,  position = 10
t = 2,  position = 30
t = 3,  position = 60
t = 4,  position = 100
t = 5,  position = 150
t = 6,  position = 210
t = 7,  position = 280
t = 8,  position = 360
t = 9,  position = 450
t = 10,  position = 550
```

Se observa ca ambele integratoarea au acelasi grad de eroare, dar Semi-Implicit euler are o eroare in fata, comparat cu rezultatul cautat $x = 500$ in timp ce Explicit-Euler este in urma. Eroare ambelor integratoare scade in functie de dt, si anume pentru $dt = 0.5$ si $dt = 0.016$ (timpul unui frame pentru on monitor cu o rata de reimprospatare de 60Hz):

Explicit Euler, dt=0.5 & dt = 0.016

```
t = 5.5,  position = 137.5
t = 6,  position = 165
t = 6.5,  position = 195
t = 7,  position = 227.5
t = 7.5,  position = 262.5
t = 8,  position = 300
t = 8.5,  position = 340
t = 9,  position = 382.5
t = 9.5,  position = 427.5
t = 10,  position = 475
```

```
t = 9.856000000000007,  position = 484.9151999999955
t = 9.872000000000007,  position = 486.49215999999547
t = 9.888000000000007,  position = 488.07167999999547
t = 9.904000000000007,  position = 489.65375999999543
t = 9.920000000000007,  position = 491.2383999999954
t = 9.936000000000007,  position = 492.8255999999954
t = 9.952000000000007,  position = 494.41535999999536
t = 9.968000000000007,  position = 496.00767999999533
t = 9.984000000000007,  position = 497.6025599999953
t = 10.000000000000007,  position = 499.19999999999527
```

Semi-Implicit Euler, dt=0.5 & dt = 0.016

```
t = 5.5,  position = 165
t = 6,  position = 195
t = 6.5,  position = 227.5
t = 7,  position = 262.5
t = 7.5,  position = 300
t = 8,  position = 340
t = 8.5,  position = 382.5
t = 9,  position = 427.5
t = 9.5,  position = 475
t = 10,  position = 525
```

```
t = 9.856000000000007,  position = 486.49215999999547
t = 9.872000000000007,  position = 488.07167999999547
t = 9.888000000000007,  position = 489.65375999999543
t = 9.904000000000007,  position = 491.2383999999954
t = 9.920000000000007,  position = 492.8255999999954
t = 9.936000000000007,  position = 494.41535999999536
t = 9.952000000000007,  position = 496.00767999999533
t = 9.968000000000007,  position = 497.6025599999953
t = 9.984000000000007,  position = 499.19999999999527
t = 10.000000000000007,  position = 500.79999999999524
```

Pentru $dt = 0.016$ integratorul Semi-Implicit Euler obtine o eroare $\epsilon = 0.79$ iar integratorul Explicit Euler $\epsilon = 0.81$, deci Semi-Implicit Euler este mai bun cu $\epsilon = 0.02$ fara nici un cost suplimentar din punct de vedere al timpului de calcul sau a memoriei folosite.

3.2 Rigid Bodies

Un corp rigid este un corp ce nu sufera deformatii in niciul fel. In cadrul algoritmului genetic si a mediului fizic, orice obiect este considerat un corp rigid cu un anumit factor de elasticitate $\sigma \in [0, 0.7]$. Avand in considerare scopul curent al algoritmului genetic, au fost alese cuburi cu $l = 1$.

3.3 Coliziuni

Din moment ce calcularea fitnessului are loc in mediul fizic, este nevoie ca simularea sa se efectueze cat mai repede, astfel se va recurge la verificarea coliziunii in 2 faze.

3.3.1 Intersectie intre doua sfere

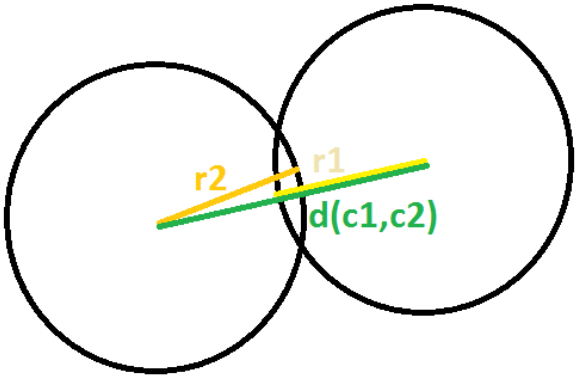
Primul pas in detectarea unei coliziuni va fi compararea intre 2 sfere $S1, S2$ ce se realizeaza in mod rapid:

$$\textit{if } d(C_{S1}, C_{S2}) < (r_{S1} + r_{S2}) \textit{ then further checks... else false}$$

Fiecare obiect va fi invelit intr-o sfera de raza r:

$$r = d_{max}((0,0), v) \text{ unde } v \in Vertexes.$$

Astfel la fiecare frame, este necesar in prima faza doar o verificare de coliziune intre sfere, fapt care salveaza mult timp.



exemplu coliziune intre 2 sfere

3.3.2 Intersectie intre doua triunghiuri

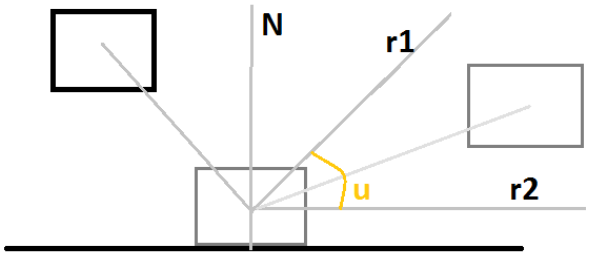
Odata trecut cazul intersectiei intre doua sfere, pasul urmatorul este cautarea unui punct de intersectie intre triunghiurile obiectelor. Acest lucru se realizarea prin verificarea intersectiei dintre doua triunghiuri pentru toate perechile a cate doua triunghiuri dintre cele doua obiecte.

Articolul **A Fast Triangle-Triangle Intersection Test** [1] prezinta un mod optimizat de a realiza testul de intersectie si a fost folosit in cadrul implementarii.

3.4 Efectul unei coliziuni

In cazul nostru, coliziunile au loc doar intre indivizi si corpuri imobile. Astfel s-a folosit un mod simplu de impuls linear.

[1] `http://web.stanford.edu/class/cs277/resources/papers/Moller1997b.pdf`



Normala la contact N se calculeaza din coordinatele varfurilor. Pentru un triunghi $T(p_1, p_2, p_3)$ unde p_0, p_1, p_2 sunt varfurile de forma $p(x, y, z)$, atunci

$$N = (p_1 - p_0) \times (p_2 - p_0)$$

In functie de coeficientul de elasticitate μ al corpului, vectorul directie rezultat in urma coliziunii va fi intre $[r1, r2]$.

- $\mu = 1$. Are loc o coliziune perfect elastica, fara pierdere de energie. Vectorul rezultat este r1. Acest caz a fost eliminat din algoritmul genetic.
- $\mu \in [0.9, 0)$. Are loc o coliziune inelastica, o parte din energia cinetica este pierduta. Vectorul rezultat va fi intre $[r1, r2]$.
- $\mu = 0$. Are loc o coliziune perfect inelastica, obiectul va urma directia $r2$.

Capitolul 4

Algoritmul Genetic

Problema propusa algoritmului genetic este gasirea unei traiectorii catre o tinta intr-un spatiu 3D cu posibile obstacole dat un proiectil fix si o forta de aplicare $F = 50N$ cu $dt = 50ms$.

S-a inceput cu o populatie $\lambda = 100$ si 50 de generatii pentru teste relativ simple.

4.1 Cromozom

Vectorul forta F necesar pentru a reprezenta o traiectorie este $F(F_x, F_y, F_z)$, fiecare reprezentand forta aplicata pe fiecare axa exprimata in newtoni. De asemenea in cazul reflectarii de pe suprafete/alte obstacole fiecare obiect are nevoie de un anumit factor de elasticitate $\mu \in [0, 0, .7]$.

In acest caz putem forma un cromozom din 4 numere naturale, F putand fi exprimat in urmatorul mod:

- F_x, 6 biti ce reprezinta o forta $F \in [0, 63]$
- F_y, 6 biti ce reprezinta o forta $F \in [0, 63]$
- F_z, 6 biti ce reprezinta o forta $F \in [0, 63]$
- μ, 3 biti ce reprezinta o elasticitate $\mu \in [0, 7]$

Astfel un cromozom este un sir de lungime fixa pentru orice individ de lungime $l = 21$.

$$\underbrace{001010}_{10}\ \underbrace{001111}_{15}\ \underbrace{000000}_{0}\ \underbrace{100}_{4}$$

Exemplu un cromozom cu $F(10, 15, 0), \mu = 0.4$

Lungimea de 6 biti pentru F_x, F_y, F_z a fost aleasa cu motivatia ca o forta de $63N$ este de ajuns pentru a muta un cub cu masa de $1kg$ intr-un spatiu relativ mic, ceea ce este de ajuns pentru a crea mai multe nivele de dificultate.

Pentru populatia initiala fiecare cromozom primeste 4 valori randomizate pentru cele 4 variabilare posibile. Alte posibilitati au fost randomizarea unei singure variabile sau a unei valori de 21 de biti. Dupa fiecare calcul al fitnessului, indivizii sunt sortati dupa fitness pentru a fi pregatiti pentru selectie.

4.2 Fitness

Fitnessul se calculeaza in doua moduri in functie de rezultatul fiecarui individ:

- In cazul in care un individ loveste tinta, atunci $f(x) = dt$, unde- dt este timpul in care individul ajunge din punctul de start la tinta.
- In caz contrar, $f(x) = d(x) * max_{dt}$, unde $d(x)$ este distanta individului fata de tinta in momentul max_{dt}, iar max_{dt} este durata de timp maxima alocata unui test.

Pentru rularea cat mai rapida a testului, coliziunea dintre indivizi este dezactivata pentru a putea fi testata o intreaga generatie in acelasi timp. Acest lucru permite si vizualizarea unei generatii.

4.3 Selectie

Modul de selectia utilizat a fost Tournament Selection.

In fiecare generatie pot exista un total de n_t tournamnets, $n_t \in [1, 3]$. Pentru fiecare tournamnet se alege in mod uniform din populatie un individ ce nu a castigat deja un tournament anterior. In fiecare tournament pot exista maxim $3 * n_t$ participanti, iar cei mai buni 2 indivizi din fiecare tournament sunt selectati pentru crossover,mutatie si sunt adaugati in generatia urmatoare. Pentru a crea noua populatie, sunt adaugati in ordinea fitnessului cromozomii din generatia precedenta pana cand $\lambda = 100$.

4.4 Crossover

Crossover se realizeaza in mod classic intre doi parinti, dupa un indice i_1 complet random. Prima parte a primului parinte va fi legata de a doua parte a partenerului.

Exemplu crossover intre un cromozom cu $\{F_1(10, 15, 0), \mu_1 = 4\}$ si $\{F_2(33, 11, 5), \mu_7 = 4\}$ la indicele $i = 11$:

$$\underbrace{001010}_{10}\ \underbrace{001111}_{15}\ \underbrace{000000}_{0}\ \underbrace{100}_{4}$$

$$+$$

$$\underbrace{100001}_{33}\ \underbrace{001011}_{11}\ \underbrace{000101}_{5}\ \underbrace{111}_{7}$$

$$\Downarrow$$

$$\underbrace{001010}_{10}\ \underbrace{001111}_{15}\ \underbrace{000101}_{5}\ \underbrace{111}_{7}$$

In TwoPointCrossOver, se genereaza un al doilea indice i_2 care indica unde va fi introdus restul genomului primului parinte, genomul partenerului fiind plasat intre i_1 si i_2.

Numarul de indivizi prezenti in populatia $i + 1$ rezultati prin crossover este maxim n_t unde n_t este numarul de tournamente ce au avut loc in populatia i.

4.5 Mutatie

Folosind tournament selection, mutatiile au loc doar asupra castigatorilor. Fiecare castigator sufera o mutatie de un bit asupra cromozomului, iar cromozomul mutat ajunge in generatia urmatoare. Alte modalitati sunt:

- Posibila mutatie a oricarui cromozom din populatie fata doar de campioni.
- Procentaj de mutatie fata de o mutatie fixa de 1 bit.

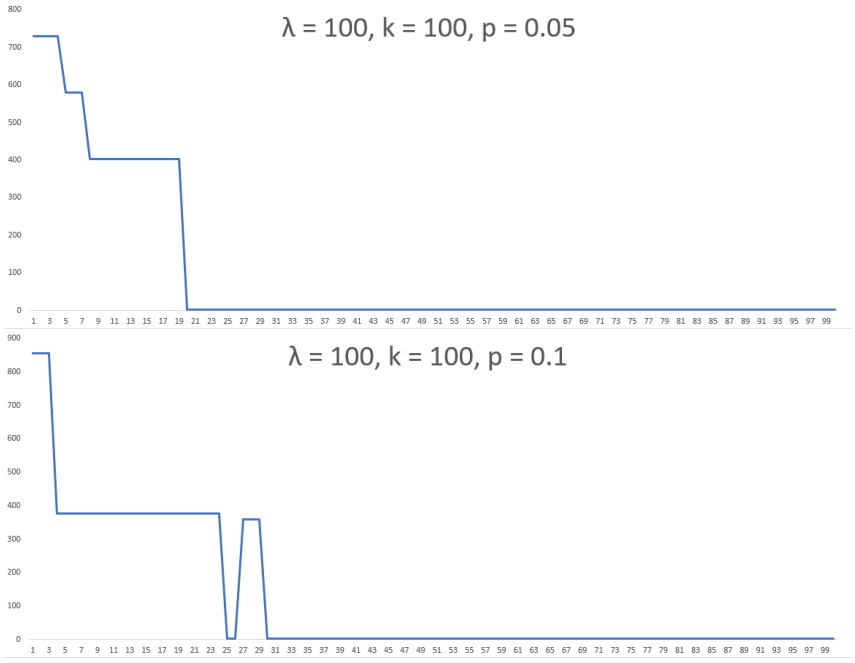
Ambele modalitati sunt viabile si au fost abordate in Rezultate.

Capitolul 5

Rezultate

5.1 Optim fix

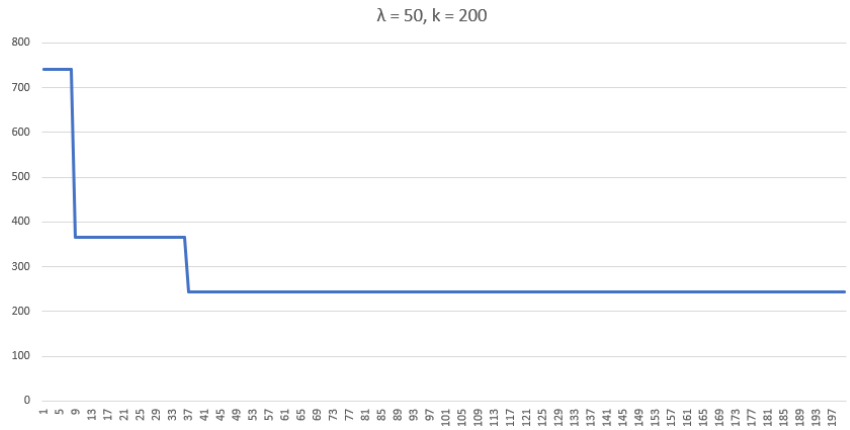
Pentru un nivel simplu, cu un singur obstacol, in care atat obstacolul cat si tinta sunt statice, algoritmul genetic reuseste sa convearga la tinta relativ rapid. Urmatoarele doua teste ruleaza cu o populatie $\lambda = 100$, un numar maxim de iteratii $k = 100$ unde o rata de mutatie de 10% a genomului poate avea loc uniform in populatie cu o probabilitate $p \in \{0.05, 0.1\}$, iar metoda de crossover este Single Point Crossover.



Pentru o populatie $\lambda = 100$ si un nivel relativ usor, algoritmul genetic converge in cateva zeci de generatii, tipul de crossover sau probabilitatea mutatiei avand impact minimal.

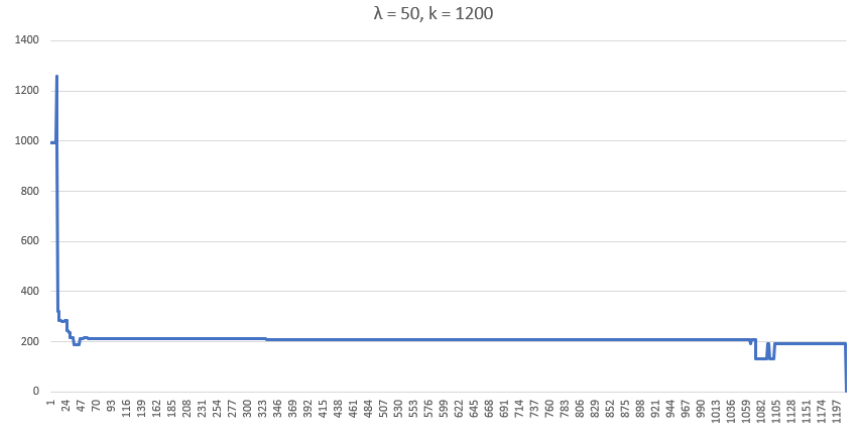
Odata cu scaderea populatiei maxime, algoritmul genetic are sanse tot mai mari

de a se bloca intr-un optim local:



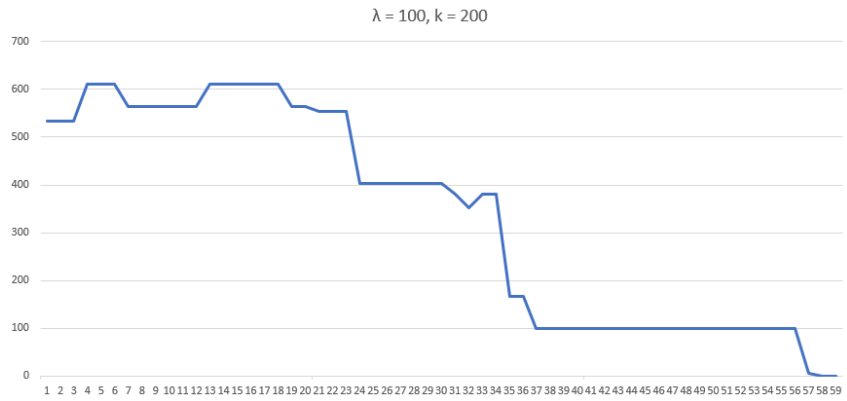
Avand 2^{21} combinatii posibile pentru un cromozom de lungime 21, fara o generatie initiala bogata, algoritmul poate ajunge intr-un optim local. Vizualizand populatia, invidivii ajung langa tinta dar nu o ating sau raman sub ea.

O solutie pentru blocarea intr-un optim local, este marirea ratei de mutatie, a iteratiilor sau a modalitatilor de selectie. Dupa marirea parametrilor de mutatie la $p = 0.15$, a ratei de mutatie a genomului la 15% si a numarului de iteratii pana la 1500, algoritmul genetic reuseste sa iasa ajunga in optimul global dupa 1200 de iteratii:

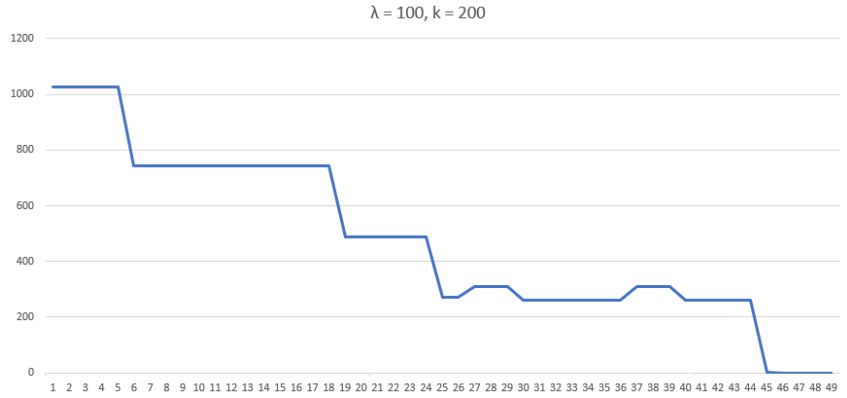


5.2 Optim dinamic

Pentru a obtine o problema de optim dinamic in mediul fizic, inducem o rotatie obstacolelor, deci acum au ”ferestre” in care se pot trece prin ele iar tinta se va roti si ea , nu mai putand fi atinsa in aceleasi locuri la orice moment de timp. Luand initial o populatie $\lambda = 100$ si $k = 200$ iteratii, algoritmul genetic reuseste sa conveargа in sub 100 de generatii:



Adaugand o translatie pe axa y a obstacolelor si folosind aceeasi configuratie pentru populatia initiala si iteratii, algoritmul genetic gaseste o traiectorie care trece pe sub obstacol imediat dupa ce acesta se ridica.



Se observa convergenta rapida pentru $\lambda = 100$ atat in optime dinamice cat si fixe.

Concluzii

Algoritmul genetic este dependent de cantitatea de material genetic pe care o are in prima generatie. In cazul lucrarii, un numar mic de indivizi in generatia initiala poate fi daunator, blocand algoritmul intr-un optim local. O probablititate de mutatie mai mare poate evita optimurile locale dar scade sansele algoritmului genetic de a atinge optimul global contaminand populatia mai mult decat ar trebuii.

Trecerea de la optim fix la optim dinamic nu s-a dovedit a fi dificila pentru algoritmul genetic. Acesta inca ajunge la optimul global la fel de repede.

Lucrarea "Evolutie Fizica prin Algoritmi Genetici" aduce posibile directii de cercetare in continuare atat in partea algoritmului genetic, cat si a motorului fizic. Lucrarea surprinde comportamentul algoritmilor genetici intr-un mediu static cat si dinamic, folosind unelte dezvoltate de la 0, pentru a avea control complet asupra tuturor factorilor dintr-o simulare.

5.3 Future Work

Lucrarea poate fi continuata in doua moduri, cu motorul fizic sau cu algoritmul genetic: Algoritmul genetic poate fi continuat prin:

- Implementarea unor cromozomi mai detaliati, mai puternici fara limitare la $63N$.
- Adaugarea unor moduri de selectie diferite care pot trata optimurile locale mai bine.
- Gasirea unei metode de a evita optime locale.
- Adaugarea unor rotatii in cromozom pentru a reactiona diferit la coliziuni.

Cat pentru motorul fizic, acesta poate fi imbunatatit din mai multe puncte de vedere:

- Implementarea unor sisteme de control in motorul fizic si a rotatiei angulare. Asta ar permite coliziuni mai realiste si dezvoltarea unor indivizi mai avansati. Implementarea unor sisteme de control in motorul fizic ar conduce algoritmul genetic spre evoluarea locomotiei.

- Schimbarea modului in care se face render in modul head pentru acelasi tip de obiect. Schimbarea pe instanced rendering ar permite vizualizarea de mii de indivizi per generatie fara pierdere de performanta.

Bibliografie

- Kenneth O. Stanley, Risto Miikkulainen. **Evolving Neural Networks through Augmenting Topologies**, http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf
- Michał Joachimczak, Reiji Suzuki, Takaya Arita. **Fine grained artificial development for body-controller coevolution of soft-bodiedanimats**, https://www.mitpressjournals.org/doi/pdf/10.1162/978-0-262-32621-6-ch040
- Thomas Geijtenbeek, Michiel van de Panne, A. Frank van der Stappen. **Flexible Muscle-Based Locomotion for Bipedal Creatures** https://www.cs.ubc.ca/~van/papers/2013-TOG-MuscleBasedBipeds/2013-TOG-MuscleBasedBipeds.pdf
- Rafael Matsunaga. **Genetic Walkers**, https://rednuht.org/genetic_walkers/
- **Evolved Antenna** https://en.wikipedia.org/wiki/Evolved_antenna
- Gregory S.Hornby and Al Globus, Derek S. Linden, Jason D. Lohn. **Automated Antenna Design with Evolutionary Algorithms** https://ti.arc.nasa.gov/m/pub-archive/1244h/1244%20(Hornby).pdf
- Martin Foot. **Math-Library-Test**, https://github.com/mfoo/Math-Library-Test
- Glenn Fiedler. **Integration Basics** https://web.archive.org/web/20190405191806/https://gafferongames.com/post/integration_basics/
- Glenn Fiedler. **Physics in 3D** https://web.archive.org/web/20181107181511/https://gafferongames.com/post/physics_in_3d/
- Glenn Fiedler. **Fix Your Timestep!** https://web.archive.org/web/20190403012130/https://gafferongames.com/post/fix_your_timestep/

- Glenn Fiedler. **Collision Response and Coulomb Friction** https://web.archive.org/web/20181107181448/https://gafferongames.com/post/collision_response_and_coulomb_friction/
- Tomas Moller. **A Fast Triangle-Triangle Intersection Test** http://web.stanford.edu/class/cs277/resources/papers/Moller1997b.pdf
- Huayang Xie, Mengjie Zhang **Tuning Selection Pressure in Tournament Selection** https://pdfs.semanticscholar.org/a243/b77fe7f4032dedbc7e0a17e26c pdf