FACULTATEA DE INGINERIE ŞI INFORMATICĂ

Programul de studii universitare de masterat
TEHNOLOGII MODERNE IN INGINERIA SISTEMELOR INFORMATICE (TMISI)

PROIECT LA DISCIPLINA "INTELIGENTA COMPUTATIONALA IN INGINERIE SOFTWARE (ICIS)" (Semestrul 1, 2023-2024)

Îmbunătățirea Configurărilor Mașinilor de Curse Folosind Programarea Evolutivă

Titularul disciplinei:

Prof. univ. dr. Grigore ALBEANU

Masterand:

BOCȘE Iulian Ionel

BUCUREŞTI
2024

Cuprins

Introducere

Pe măsură ce mașinile tind să devină din ce în ce mai complexe, la fel și sporturile cu motor, un domeniu în care este foarte important să profiți de fiecare aspect al vehiculelor, precum mecanica, termodinamica și aerodinamica. Configurarea mașinilor de curse a fost de multă vreme o sarcină a inginerilor mecanici, care studiau și teoretizau cele mai bune ajust;ari posibile aduse vehiculului pentru a-i optimiza comportamentul, direcția, viteza și manevrabilitatea. Dar în vremurile curente, astfel de sarcini au devenit din ce în ce mai dependente de lumea informaticii, deoarece calculatoarele moderne au avantajul nu doar de a simula scenarii și condiții de curse precum cele din lumea reală, ci pot și simula, calcula sau pur și simplu ghici cei mai buni parametri necesari pentru a configura în cel mai bun mod o mașină de curse. Această lucrare introduce idea de a utiliza un algoritm evolutiv pentru a determina dacă aceștia pot fi utilizați pentru a obține una sau mai multe soluții de configurare a unei mașini de curse.

1 Istoria și Evoluția Mașiniilor de Curse

Natura competitivă a sporturilor face ca numai cei care înțeleg mai bine regulile și opțiunile la îndemână, obțin rezultate mult mai bune decât restul. Din acest motiv, fiecare sportiv încearca să fie cel mai bun, chiar dacă acest rezultat este uneori obținut prin maniere ilegale, cum ar fi cu "dopajul". Motivele din spatele acestui lucru, în afară de faptul că prin dopaj se obțin rezultate mult mai bune și mult mai repede decât fară este că, corpul uman este foarte limitat, chiar și atunci când este împins la extreme. Mai simplu spus, corpurile noastre au evoluat pentru ca noi să fim vânători-culegători, nu înotători olimipici sau baschetbaliși.

Spre deosebire de orice alt sport clasic în care fizicalitatea este printre prioritățile cele mai mari, sporturile cu motor au avut întodeauna avantajul de a evolua și de a se adapta alături de diversele domenii pe care se bazează, cum ar fi aerodinamica și termodinamica. La începutul sporturilor cu motor, în jurul anilor 1920, aceste concepte nu erau luate în considerare. Singurele două aspecte importante când venea vorba de a produce o mașina de curse erau *fiabilitatea* și *puterea* motorului. Motoarele mai mari produceau mai mulți cai putere, cea ce însemna o viteză mai mare, deci șansa de a câștiga creștea (atâta timp cât motorul era fiabil din punct de vedere mecanic). Aceasta era strategia de-facto a echipelor și producătorilor de mașini de curse din prima epocă a acestui sport [Figura 1.a] . Dar pe măsură ce cererea de performanță creștea, inginerii mecanici au trebuit să împrumute concepte din alte domenii ale științei, în afara mecanicii. În 1968, la 18 ani după primul "Grand Prix[1]" de Formula 1, Colin Chapman[2] a fost prima persoană care a produs o mașina de curse concepută având în vedere principii aerodinamice. Acea mașina, numită Lotus 49B [Figura 1.b], avea profiluri aerodinamice precum aripi pe partea anterioară, și un eleron în partea posterioară pentru a crește forta aerodinamică și stabilitatea mașinii la vitezele mai mari.

Figura 1.a: Ferrari 312, 1967. [wikipedia.org]

Figura 1.b: Lotus 49B, 1968. [Will Broadhead]

Lotus 49B devenise cea mai revoluționară mașină de curse produsă până atunci[1], schimbând pentru totdeauna peisajul curselor de Formula 1 cât și industria automobilistică. Multe

[1]Titlu acordat curselor internaționale de mare prestigiu, de obicei asociate cu Formula, GT sau MotoGP
[2]Colin B. Chapman 1928-1982. Inventator, inginer, pionier în industria automobilistică și proprietarul Lotus

invenții din lumea automobilistică provin de fapt din lumea motorsporturilor, precum motoarele DOHC, șasiurile din fibră de carbon, suspensia adaptivă și motoarele hibride.

Pe măsură ce domeniul automibilistic avansa, din punct de vedere tehnologic, la fel urma și domeniul motorsporturilor. În ultimii 20 de ani, mașinile de curse au devenit din ce în ce mai complexe și costisitoare. O mașina de Formula 1 de exemplu, este compusă din aproximativ 14,500 de componente individuale[2], fabricate la comandă, din materiale uneori deosebit de costisitoare, aducând prețul final al mașinii în jur de 12-15 milioane de dolari. Costul acestor vehicule nu este numai datorită faptului că aceste mașini sunt rapide, ci și prin faptul că unele dintre aceste părți sunt concepute pentru a avea un nivel de modularitate foarte deosebit. Ele pot fi schimbate, adaptate, ajustate și reglate conform nevoilor șoferului, condițiilor atmosferice sau specificațiile traseului.

2 Necesitatea Configurațiilor

În sporturile cu motor, construirea cele mai rapide și scumpe mașini de curse nu echivalează niciodată cu construirea cele mai bune mașini. O mașină de curse poate rezulta a fi foarte performantă pe un anumit traseu, dar nu pe altul. Lăsând deoparte performanța și capabilitățile șoferului la volanul vehicului, echipele de curse se confruntă cu diverse probleme când vine vorba de a configura mașina lor, deoarece există o multitudine de variabile care trebuie luate în considerare. Acestea pot fi grupate în 2 mari categori:

- Variabile **Interne**: acestea includ variabilele legate de mașină, precum configurarea aerodinamică, cantitatea de combustibil, configurația suspensiilor, puterea și polarizarea frânelor, echilibrul de greutate, tipologia și starea pneurilor, etc.

- Variabile **Externe**: acestea nu țin de mașină sau de echipă, ci de factori externi, precum cei meteorologici și atmosferici: temperatură atmosferică, vânt, umiditate;
  și a pistei pe care se concurează: numărul de curbe, distanța, inclinația, lățimea, temperatura și calitatea asfaltului, etc.

Complexitatea alegerii configurației pentru mașina de curse, împreună cu faptul că în majoritatea cazurilor, cursele sunt la o saptămână distanță una față de cealaltă, a fost o problemă de multă vreme în istoria motorsporturilor.

3 Big data în Sporturile cu Motor

Pentru a aborda această problemă, în ultimii ani echipele de motorsport au încheiat diverse parteneriate cu companiile mari de IT precum Oracle, Microsoft, Amazon și Google. Aceste parteneriate permit echipelor de să se folosească de infrastructuri de calcul foarte puternice pentru a calcula toate aceste date în mod rapid și eficient. Conform unui studiu din 2019, o echipă de Formula 1 colectează în jur de 500 Gigabiți de date de la fiecare cursă, folosind aproximativ 300 de senzori plasați pe mașinile lor[3]. Aceste date, o dată prelucrate și analizate, pot fi folosite pentru a înțelege mai bine diferite aspecte atât ale mașinii cât și a le șoferului. De exemplu, aceste date oferă posibilitatea de a înțelege dacă șoferul ar putea să-și îmbunătățească turele, unde ar putea fi mai rapid sau unde greșește, dacă mașina suferă de o imbalanța la frănare, greutate sau forță aerodinamică, care sunt pneurile cele mai potrivite, etc.

Figura 2: O ehcipă de F1 își consultă datele colectate după o cursă [autosport.com]

Analizarea datelor colectate nu este singura metodă aplicată de către aceste echipe deoarece datele, oricât de multe ar fi, nu sunt întodeauna suficiente pentru a determina configurarea cea mai bună. Seriile de curse mai competitive precum F1, GT1 și LMP1 își permit să se folosească de aceste date pentru a rula simulări de curse. În trecut, simulările au fost folosite în principal pentru a permite șoferilor să se familiarizeze cu diferele trasee, dar în prezent simulatoarele folosite de către șoferi au devenit atât de sofisticate încât le permite lor să testeze configurația mașinii, sau orice altă configurație în mod sigur și rapid. Simulatoarele Pro-Sim Evolution, și AOTech [Figura 3] de exemplu, sunt construite la comandă, special pentru o anumită echipă sau o anumită mașină, ceea ce permite echipei să-și testeze și să-și perfecționeze configurațiile cu mult timp înainte de o cursă. Cu toate acestea, este clar că nu orice echipă de motorsport își permite să investească milioane de euro pe simulatoare și cloud computing.

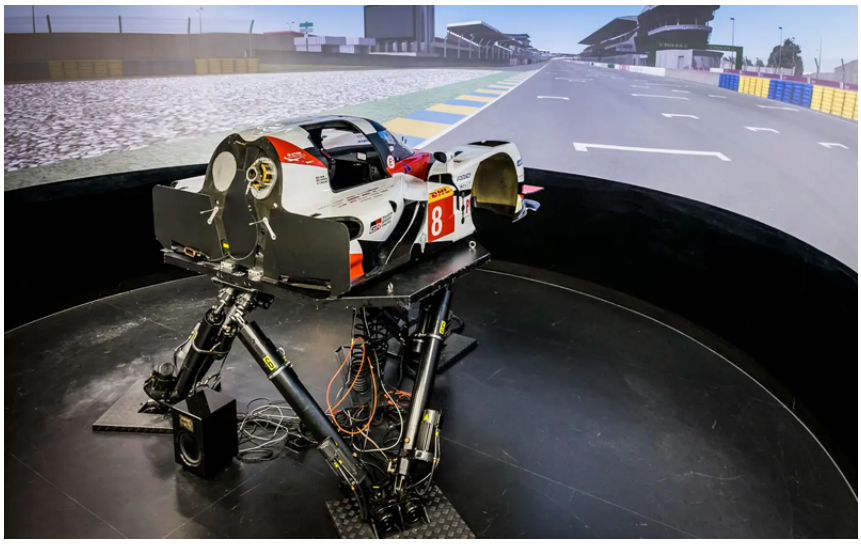

Figura 3: Simlatorul AOTech folosit de Toyota GZ pentru cursele Le Mans [popsci.com]

4 Abordarea Problemei Folosind Inteligența Computațională

Soluțiile deja existente (simulatoare și data crunching) sunt o formă de problem solving numită **hard computing**. Aceste soluții se bazează pe date concrete sau modele matematice, urmând o logica binară. Problema acestei abordări este că uneori problemele noastre nu pot fi traduse în totalitate folosind termeni absoluți de 0 și 1. Abordarea opusă a hard computing-ului este bineînțeles **soft computing**, un termen folosit când se face referire la algoritmele de *inteligență computațională* și *inteligență artificială*. Acestea se inspiră de la natură, și oferă posibilitatea de a aborda problemele complexe la care matematica tradițională nu reușește sau este incertă.

Există diferite algoritme de soft computing, precum cele de *fuzzy*, sau *rețelele neuronale*, dar în această lucrare problema a fost abordată prin utilizarea unui *algoritm evolutiv* (abreviat ca EA), o tipologie de algoritm de optimizare care face parte din domeniul calculului evolutiv. Algoritmele evolutive sunt similare cu algoritmele genetice (GA), dar reprezentarea lor este portivită pentru optimizarea numerică, adică pentru un vector de numere cu virgulă mobilă[4].

Schema generală a acestor alogritmi este următoarea[5]:

```
initializare POP(0)
evaluare(POP(0))
GEN = 0
while(!conditie_de_oprire) {
  selectarea a 2 parinti p1 si p2 din POP(GEN)
  crossover(p1,p2) -> o1
  mutatie(o1, param) -> o1*, param
  evaluare(o1*)
  aduaugare o1* in POP(GEN+1)
  // repetare pana cand POP(GEN+1) este plina
  GEN++
}
```

4.1 Parametrii Algoritmilor Evolutiv

Pentru a înțelege mai bine algoritmii EA, putem disucta despre pseudocodul menționat mai sus și definii parametrii și funcțiile din care este compus.

- **Populație**: în acești algoritmi, toate soluțiile care concurează fac parte din ceea ce se numește *populație*. Aceasta poate fi reprezentată cu un număr întreg, de exemplu 20, 50 sau 100.
- **Generație**: simplu pus, este numărul de iterații în care populația de soluții a evoluat. Generația 50 de exemplu, este compusă din populația care a evoluat de la începutul rulări algoritmului, după 50 de modificări subite.
- **Genotip**: fiecare soluție este definită de un număr de parametri sau caracteristici. În algoritmii genetici (GA) de exemplu, aceasta se este reprezentată printr-o matrice de biți, dar în programarea genotipul se reprezintă ca un vector de numere reale (float). În contextul acestui proiect, genotipul poate fi descris ca parametrii vehiculelor de evoluat.
- **Fitness**: este valoarea de "potrivire" a fiecarui soluții. Această valoare determină cât de potrivită/bună este o anumită soluție în comparație cu o altă soluție. Idea acestor algoritmi este că pe parcurs ce populația evoluează, această valoare de fitness medie ar trebui să se marească păna la un anumit punct.
- **Condiția de Oprire**: deoarece evoluția acestor algoritmi se petrece într-o buclă, este nevoie de o condiție pentru a termina generarea de soluții ulterioare. Această condiție diferă de la caz la caz, și poate fi de exemplu când fitness-ul mediu ajunge la o anumită valoare, sau când numărul de genrații depașeste o anumită valoare.

4.2 Funcțiile Algoritmiilor Evolutivi

- **Evaluare**: funcția de evaluare evaluează dacă o anumită soluție din populația curentă satisface gradul de adaptare la mediu pe baza valorii ei de fitness.
- **Selecție**: la sfârșitul fiecârei generații, funcția de selecție are scopul de a alege soluțiile ce vor supraviețui în generația următoare (cele care satisfac funcția de evaluare). Din toate soluțiile prezente, funcția de selecție alege cele 2 soluții cu valoare de fitness mai înalta, "părinți" p_1 și p_2. Aceștia vor fi apoi folosiți pentru a regenera noua populație în următoara generație.
- **Recombinare**: numită și *crossover*, este operația genetică prin care genotipurile părinților p1 și p2 sunt combinate pentru a produce un nou genotip o1, membru în populația generației viitoare.

- **Mutație**: cel de al doilea oprator genetic, utilizat pentru a menține diversitatea genetică a genotipelor unei populații. Este analog cu mutația biologică[4].

5 Soluția Propusă

Pentru a determina dacă algorimii evolutivi au șansa de a rezolva problema despre care s-a discutat, soluția propusă folosește acești algoritmi pentru a conduce o populație de mașini în jurul unei piste. Această populație va fi inițializată cu parametrii și caracteristici aleatoarii, dar pe parcurs ce algoritmul va evolua, acești parametri se vor normaliza către valorile cele mai optime posibile. Rezultatul așteptat este cel de a începe cu o populație (GEN 1) care este incapabilă de a completa o singură tură pe traseu, datorită faptului că parametrii soluților care concurează sunt improprii pentru a putea conduce în mod optim. Pe măsură ce populația evoluează și valorile de fitness cresc, se va atinge obiectivul final în care ultima populație (GEN X) este ceea ce deține o valoare medie de fitness destul de înaltă încât majoritatea soluțiilor să fie capabile de a conduce mașina în mod optim pe lungimea întregului traseu.

6 Tehnologiile Utilizate

Pentru a realiza o implementare care să satisfacă cerințele propuse, și în același timp să permită vizualizarea rezultatelor și a evoluției algoritmului, a fost folosit Unity 2021.3.33f1, un motor de joc 3D. Datorită acestuia a fost posibilă implementarea unor parametri conformi cu parametrii și caracteristicele mașinilor de curse din viata reală. Bineînțeles asta nu înseamnă că implementarea propusă este 1:1 cu realitatea, ci este doar un proof-of-concept că algoritmii de inteligența computațională, în particular cel cele evoluționistice, reușesc să satisfacă problema propusă. Numărul și implementarea acestor caracteristici poate fi extins și îmbunătățit în diferite moduri, păstrând aceleași funcții evoluționistice la bază.

6.1 Meniul Principal

La executarea programului ni se prezintă meniul principal [Figura 4]. Acest meniu a fost creat cu scopul de a implementa o metodă, de a modifica anumite setări ale programului, precum populația, timpul acordat fiecarei generații, dar și opțiunea de debug și ceea de "dump", pentru exportarea datelor și a rezultatelor obținute în format text, pentru a putea fi folosite de exemplu, pentru a trasa grafice.

Figura 4: Meniul principal al programului (deschis în editorul de joc)

Apăsând pe butonul central de *START* partea principală a programului va fi încărcată, cu parametrii și opțiunile setate din meniu.

6.2 Implementarea Circuitului

Una dintre componentele cheie ale programului propus este circuitul [Figura 5]. Implementarea acestuia a fost realizată printr-o listă circulară de *waypoints* (puncte de referință). Această listă este compusă în total de 75 waypoint-uri, 25 pentru fiecare din cele 3 sectoare[3] ale circuitului.

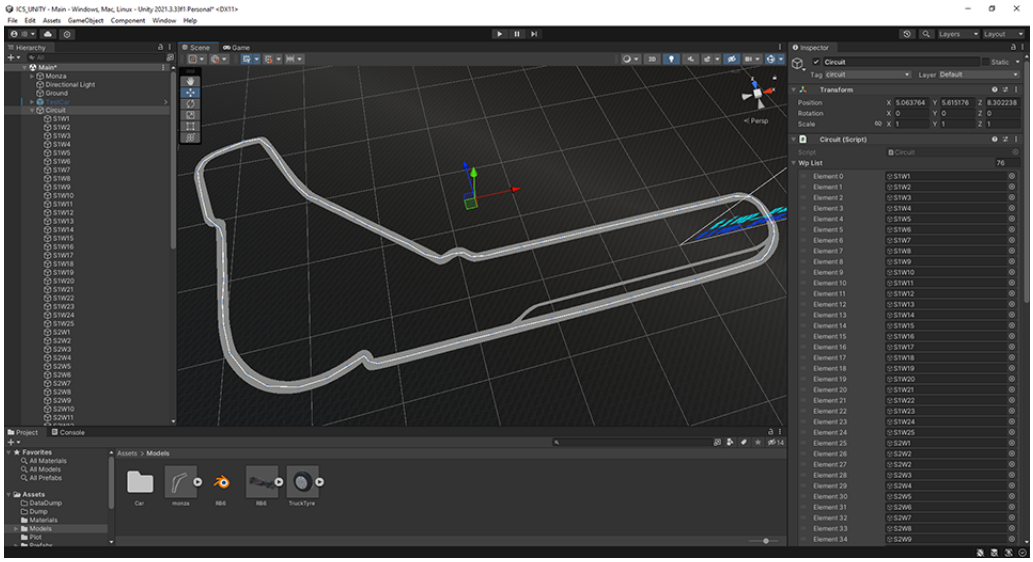

Figura 5: Circuitul (centru) și lista de waypoint-uri (dreapta)

[3]Modul în care un circuit este împărțit

Lista de waypointuri nu numai că are scopul de a instrui mașinilor traseul potrivit de urmărit, dar este și folosită pentru a determina valoarea de **fitness** a fiecărui vehicul: o implementare în care fitness-ul este determinat de către lungimea sau distanța parcursă, deși poate părea corectă, este de fapt greșită. Putem lua ca exemplu o mașină cărei caracteristici sunt încă sub-optime: această nu va frâna corect înainte de o curbă, și va ieși de pe circuit, încercând să reintre pe parcursul cel bun urmând un traseu mult mai larg, și astfel acoperind o distanță mult mai mare. Această implementare ar duce la o evaluare a valorii de fitness incorectă, deoarece nu vor fi aleși candidații ale căror caracteristici le permite un condus mai corect, și cei care din contră, greșesc mai mult în a urmării traseul propus.

6.3 Vehiculelor

Populația programului este compusă bineînțeles din mașini, fiecare cu o listă de valori ce determină caracteristicile lor [Figura 6]. Aceste caracteristici sunt determinate de diferite scripturi, fiecare acoperind o parte diferită a logicii de conducere a unei mașini.

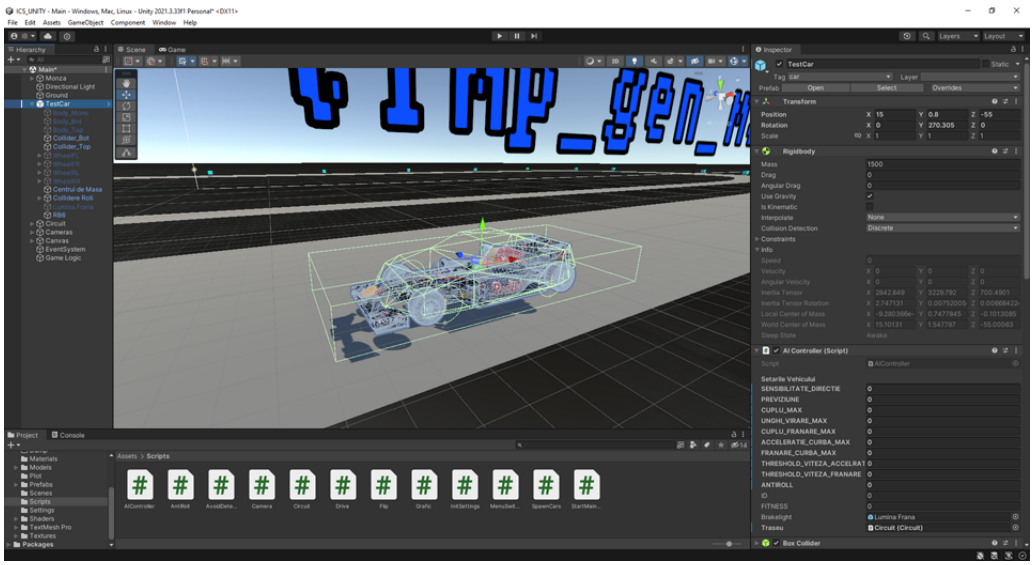

Figura 6: Mașina folosită și parametrii ei (dreapta)

- Avoid Detector: acest script conține logica de evitare a obstacolelor. Circuitul în sine nu conține obstacole, dar bineînțeles, scriptul a fost implementat pentru a permite vehiculelor posibilitatea de a se ferii de alte vehicule prezente pe circuit, cu scopul de a-și mări sănsa de supraviețuire, deoarece o dată ce două mașini se izbesc, ambele au o șansă înaltă de a pierde din timpul alocat generației de a-și mări valoarea de fitness. În cazul în care parametrii aferenți nu permit mașinii să evite un obstacol, sau impactul a fost cauzat de o reacție în lanț, scriptul permite posibilitatea de a-și inversa direcția de mers (analog cu marșarier-ul) până în momentul în care reintrarea pe pistă este posibilă.

- Flip: există cazuri rare în care mașiniile se pot răsturnă cu susul în jos. Acest script

verifică la un interval stabilit din meniul principal dacă mașina este răsurnată, și o întoarce în modul corect dacă este cazul. Valoarea acestui interval nu intră în lista de parametrii manipulați de algoritmul evolutiv.

- Anti Roll: când o mașină cu tracțiune pe spate (cazul mașiniilor folosite în programul propus, dar și celor din realitate) trimite o forță mult prea mare către roțiile posterioare, centrul de masă a vehiculului s-a dezichilibrat, cauzând o rotire a vehiculului pe axa orizontală. În lumea sporturilor cu motor acest fenomen se numește *spin* sau *roll*. Mașiniile comerciale vin echipate de foarte mulți ani cu un sistem mecanic numit *Traction Control*, sau TC pe scurt, care ajută la combaterea acestui fenomen și la îmbunătățirea stabilității. Dar aceste sisteme de "ajutor" care reglează imperfecțiunile cauzate în condusul unei mașini de către șofer (TC, ABS, ERS), nu sunt permise în toate cateogiile sau ligile sporturilor cu motor deoarece sunt văzute ca limitative, deoarece reduc pragul de dexteritate, talent și experiența a șoferilor. Din păcate, nu putem replica "experiența" și "dexteritatea" unui pilot în acest tip de algoritm, deoarece acestea ies din discuție în ceea ce privește scopul soluției propuse. Din acest motiv, a fost implementat acest script care simulează sistemul TC din viată reală. Scriptul aplică o forță de *Anti Roll* asupra vehiculului în cazul în care acesta se dechilibrează din motivele explicate mai devreme.

- AI Controller: acesta este unul din cele două scripturi principale ale programului. El se ocupă de toată logica de condus a mașiniilor, aplicând caracteristicile mașinii in practică. Caracteristicile (genotipul) introduse sunt următoarele: *Sensibilitatea virajului*, *Previziunea*, *Cuplul maxim*, *Unghiul maxim de virare*, *Cuplul maxim în frânare*, *Accelerația maximă în curbă*, *Frânarea maximă în curbă*, *Pragul superior al vitezei de accelerație*, *Pragul superior al vitezei de frânare*, și *Fortă de anti-roll* (doar declarată cu scopul de a fi accesată de către algoritmul EA; este pusă în practică de către scriptul de evitare a obstacolelor).

- Game Logic: al doilea script princial se ocupă de logica programului și de algoritmul evoluționist. Acest script execută sarciniile de: generare a populației și poziționare lor în "grila de cursă"; executarea algoritmului evolutiv (evaluare, selecție, recombinare, mutație); interfața grafică [Figura 7]; funcționalitatea de exportare a datelor obținute.

7 Interfața Programului

Pentru a înțelege mai bine ce se întâmplă pe parcursul execuției, a fost implementată o interfață grafică [Figura 7] ce detaliază câteva aspecte ale algoritmului evolutiv.

Figura 7: Interfața grafică a programului în acțiune

Din figura de mai sus se pot observa următoarele elemente grafice:

- Stânga-sus: rezultatele de fitness maxim și fitness median din generația trecută;
- Textul central-sus: generația curentă;
- Dreapta-sus: dimensiunea populației (valoare stabilită din setările meniului principal);
- Stânga-jos: timpul parcurs de la ultima generație și timpul maxim alocat fiecărei generații (valoare stabilită din setările meniului principal);
- Centru-jos și drapta-jos: aceste elemente corespund cu porțiunea traseului din fereastra de joc. Pentru a permite vizionarea întregului traseu, a fost creat un script care permite utilizatorului să schimbe camera de joc[4] activă cu oricare dintre cele 4 camere disponibile.

[4]Obiecte în motoarele grafice ce captează o anumită porțiune a mediului 2D/3D și o afișează pe ecran

8 Implementarea Algoritmului

Soluția proprusă, la fel ca oricare alt program în Unity Engine, este compusă din două funcții principale: funcția **Start()**, executată o singură dată la lansarea programului, și funcția **Update()**, executată continuu în fiecare frame (cadru)[5]. Organigrama de mai jos [Figura 8] evidențiază secvența de instrucțiuni exectutate de către program:

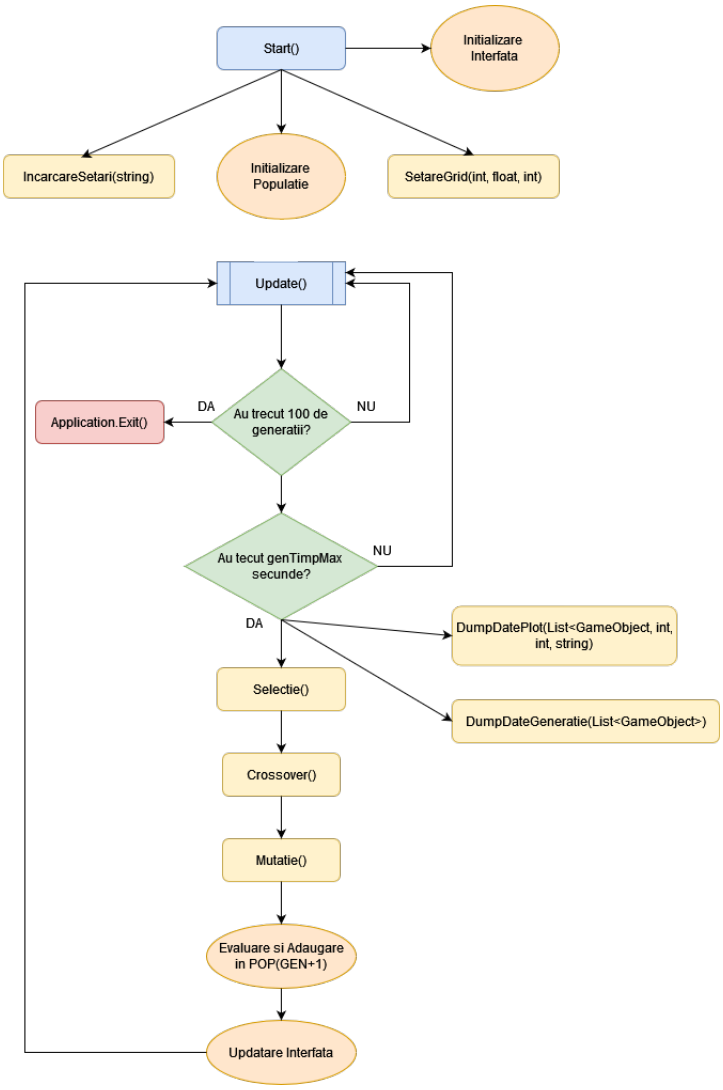

Figura 8: Flowchart-ul programului

START

Începând cu funcția *Start()*, aceasta inițializează toate variabilele, listele și obiectele, folosind parametrii impostați din meniul principal, legăturile directe din editorul Unity, sau în cazul pozițiilor de start ale grilei, generându-le cu o altă funcție numită *SetareGrid(int, float, int)*. Populația inițială este apoi generată în mot iterativ, atribuind fiecărei mașini un set de caracteristici (genotip) aleatoriu. Valorile minime și maxime folosite pentru a genera fiecare dintre

[5]De fiecare dată când imaginea ferestrei este reîmprospătată. In cazul acestui program, frecvența de generare a cadrelor a fost fixată la 60 de ori pe secundă.

aceste caracteristici au fost alese pentru a permite mașinii generate de a fi mai mult sau mai puțin eligibilă pentru a se putea măcar mișca, și de a evita excepții, erori sau probleme grave cu toate scripturile folosite pentru logica de conducere, discutate în capitolul trecut. Aceste intervale au fost determinate după diferite testări efectuate în faza de dezvoltare a programului, și permit generarea unei populații cu o diversitate destul de înaltă, în timp ce omit cazurile în care diverși parametri generați pot duce la impostări privite de sens (de exemplu un cuplu negativ, o previziune de 0, sau o sensibilitate la viraj excesiv de înaltă)

```
...
public List<GameObject> vehList;
public List<Vector3> vehPositiiList;
public int POPULATIE;
public int GENERATIE;
public int genTimpStart = 0;
public int genTimpMax;
public int IDcounter = 0;
...
void Start()
{
  // Incarcarea setariilor
  LoadSettings(Application.dataPath + @"/" + "Settings/GameLogic.ini");
  System.IO.Directory.CreateDirectory("DataDump");

  // Generarea positiilor de start
  vehPositiiList = SetareGrid(15, 0.8f, -55);

  // Generarea initiala a populatiei
  for (int i = 0; i < POPULATIE; i++)
  {
    GameObject newVehClona = Instantiate(carprefab, vehPositiiList[i],
 this.transform.rotation);
    AIController newVehClona = newVehClona.GetComponent<AIController>();

    // Adaugarea parametrilor aleatori
    newVehClona.ID = i;
    newVehClona.SENSIBILITATE_DIRECTIE = UnityEngine.Random.Range(0.01f,
 0.1f);
    newVehClona.PREVIZIUNE = UnityEngine.Random.Range(1.0f, 50.0f);
    newVehClona.CUPLU_MAX = UnityEngine.Random.Range(100.0f, 500.0f);
    newVehClona.UNGHI_VIRARE_MAX = UnityEngine.Random.Range(10.0f, 100.0f
);
    newVehClona.CUPLU_FRANARE_MAX = UnityEngine.Random.Range(1000.0f,
 9000.0f);
    newVehClona.ACCELERATIE_CURBA_MAX = UnityEngine.Random.Range(1.0f,
 50.0f);
```

```
33      newVehClona.FRANARE_CURBA_MAX = UnityEngine.Random.Range(1.0f, 10.0f)
    ;
34      newVehClona.THRESHOLD_VITEZA_ACCELERATIE = UnityEngine.Random.Range
    (1.0f, 50.0f);
35      newVehClona.THRESHOLD_VITEZA_FRANARE = UnityEngine.Random.Range(1.0f,
     25.0f);
36      newVehClona.ANTIROLL = UnityEngine.Random.Range(1000.0f, 9000.0f);
37      ...
38      vehList.Add(newVehClona);
39    }
40    // Initializare interfata grafica
41    ...
42  }
```

UPDATE

Funcția de *Update()* este destul de simplă, făcând doar două lucruri: actualizează șirul cu timpul parcurs din interfața grafică, și verifică dacă acest timp a depășit limita maximă alocată fiecărei generații. Dacă acesta este cazul, atunci pornește lanțul algortmului evolutiv prin execuția funcției de *Selecite()*, si exportă datele generației curente (dacă această funcționalitate a fost activată din setăriile meniului principal) prin funcția *DumpDateGeneratie(List<GameObject>)*.

```
1   void Update()
2   {
3     // Actualizeaza interfata grafica
4     txtTimpParcurs.text = "timp: " + (Time.realTimeSinceStartup - (
    genTimpMax * (GENERATIE - 1)));
5     // Daca timpul alocat generatiei sa terminat
6     if (Time.realtimeSinceStartup > genTimpStart + genTimpMax)
7     {
8       if(DUMPDATA) DumpDateGeneratie(vehList);
9       Selectie()
10    }
11  }
```

8.1 Selecția și Evaluarea

Strategia implementată pentru a determina genotipurile supraviețuitoare în fiecare generație este de a păstra doar jumătate din populația curentă. Lista ce conține toate soluțile prezente este sortată în mod descrescător pe baza valorii de fitness. Asta permite iterarea funcții de *Crossover(AIController, AIController)* doar pe primele $\frac{POP}{2}$ elemente din listă, ignorând a doua jumătate ce conține soluții cu fitness din ce în ce mai scăzut.

```
1   void Selectie()
2   {
3     // Resetare ceas
```

```
    genTimpStart = Time.realtimeSinceStartup;

    // Creare unei liste sortate descendent pe baza valorii de fitness
    List<GameObject> masiniSortate = vehList.OrderByDescending(o => o.
   GetComponent<AIController>().FITNESS).ToList();
    vehList.Clear();

    // Evaluare: este aleasa doar jumatatea populatiei,
    for (int i = 0; i < (int)(masiniSortate.Count / 2.0f); i++)
    {
      // Crossover
      vehList.Add(Crossover(masiniSortate[i].GetComponent<AIController>(),
   masiniSortate[i+1].GetComponent<AIController>()));
      vehList.Add(Crossover(masiniSortate[i+1].GetComponent<AIController>()
   , masiniSortate[i].GetComponent<AIController>()));
    }
    ...
    // Dump date pentru pyplot
    if(DUMPDATA) dumpDatePlot(vehList, avgFitness, GENERATIE, "DataDump/
   plot_avg.txt");
    if(DUMPDATA) dumpDatePlot(vehList, bestFitness, GENERATIE, "DataDump/
   plot_bst.txt");

    // Curatare populatie curenta
    for(int i = 0; i < masiniSortate.Count; i++) Destroy(masiniSortate[i]);

    // Conditia de oprire
    if (avgFitness > 75) Application.Quit();

    // Avansare generatie
    GENERATIE++;
    IDcounter = 0;

    // Actualizare interfata grafica
    ...
  }
```

După ce o nouă populație a fost generată din operația de recombinare și mutație, aceasta este adăugată în lista de soluții *vehList* și contorul generației avansează cu 1, dar nu înainte să fie verificată condiția de oprire a alogrimului: în fragmentul de cod de mai jos a fost folosită condiția de *if (Valoare de Fitness Medie > 75) then STOP*, deoarece 75 este și numărul total de waypoint-uri din care este compus circuitul utilizat în acest caz. Acesata poate fi interpretată astfel: soluția "medie" este suficientă pentru a permite unei mașini să parcurgă o tură completă în limita de timp propusă. Altfel de condiții pot fi bineînțeles implementate în locul acesteia, de exemplu se poate impune o limită de generații, sau se poate verfica dacă fitness-ul mediu minim ajunge peste un anumit prag.

8.2 Recombinarea

Funcția de recombinare generează noi soluții descendente cu caracteristici determinate de cei doi părinți selectați mai devreme. Strategia folosită pentru acest proces a fost de a alege, pentru fiecare parametru în parte, media dintre valoriile părinților: $o_i = \frac{p_i + p_{i+1}}{2}$

```
GameObject Crossover(AIController parinteX, AIController parinteY)
{
  ...
  // Generare descendent o1
  GameObject newVehClona = Instantiate(carprefab, this.transform.position
 , this.transform.rotation);
  AIController newVehClonaAI = newVehClona.GetComponent<AIController>();

  // Atribuirea valorilor parintiilor p1 si p2 lui o1
  newVehClonaAI.ID = (IDcounter++);
  newVehClona.SENSIBILITATE_DIRECTIE = (parinteX.SENSIBILITATE_DIRECTIE +
   parinteY.SENSIBILITATE_DIRECTIE) / 2.0f;
  newVehClona.PREVIZIUNE = (parinteX.PREVIZIUNE + parinteY.PREVIZIUNE) /
 2.0f;
  newVehClona.CUPLU_MAX = (parinteX.CUPLU_MAX + parinteY.CUPLU_MAX) / 2.0
 f;
  newVehClona.UNGHI_VIRARE_MAX = (parinteX.UNGHI_VIRARE_MAX + parinteY.
 UNGHI_VIRARE_MAX) / 2.0f;
  newVehClona.CUPLU_FRANARE = (parinteX.CUPLU_FRANARE + parinteY.
 CUPLU_FRANARE) / 2.0f;
  newVehClona.ACCELERATIE_CURBA_MAX = (parinteX.ACCELERATIE_CURBA_MAX +
 parinteY.ACCELERATIE_CURBA_MAX) / 2.0f;
  newVehClona.FRANARE_CURBA_MAX = (parinteX.FRANARE_CURBA_MAX + parinteY.
 FRANARE_CURBA_MAX) / 2.0f;
  newVehClona.THRESHOLD_VITEZA_ACCELERATIE = (parinteX.
 THRESHOLD_VITEZA_ACCELERATIE + parinteY.THRESHOLD_VITEZA_ACCELERATIE) /
 2.0f;
  newVehClona.THRESHOLD_VITEZA_FRANARE = (parinteX.
 THRESHOLD_VITEZA_FRANARE + parinteY.THRESHOLD_VITEZA_FRANARE) / 2.0f;
  newVehClona.ANTIROLL = (parinteX.ANTIROLL + parinteY.ANTIROLL) / 2.0f;
  ...

  // Generare mutatie
  newVehClonaAI = Mutatie(newVehClonaAI, UnityEngine.Random.Range(0,9));

  return newVehClona;
}
```

8.3 Mutația

La finalul recombinări, dar înainte de a returna noua soluție generată, este aplicată funcția de *Mutație(AIController, int)* pe un parametru aleatoriu *r* din genotipul descendentului: această stratiegie de mutație poartă numele de **mutație cu resetare aleatorie**. Parametrul extras pentru mutație este "resetat" cu o valoare aleatorie (conform minimelor și maximelor din procesul de generație inițială).

```
AIController Mutatie(AIController V, int r)
{
  switch(r)
  {
    case 0: V.SENSIBILITATE_DIRECTIE = UnityEngine.Random.Range(0.01f,
 0.1f); break;
    case 1: V.PREVIZIUNE = UnityEngine.Random.Range(1.0f, 50.0f); break;
    case 2: V.CUPLU_MAX = UnityEngine.Random.Range(100.0f, 500.0f); break
 ;
    case 3: V.UNGHI_VIRARE_MAX = UnityEngine.Random.Range(10.0f, 100.0f);
  break;
    case 4: V.CUPLU_FRANARE_MAX = UnityEngine.Random.Range(1000.0f,
 9000.0f); break;
    case 5: V.ACCELERATIE_CURBA_MAX = UnityEngine.Random.Range(1.0f, 50.0
 f); break;
    case 6: V.FRANARE_CURBA_MAX = UnityEngine.Random.Range(1.0f, 10.0f);
 break;
    case 7: V.THRESHOLD_VITEZA_ACCELERATIE = UnityEngine.Random.Range(1.0
 f, 50.0f); break;
    case 8: V.THRESHOLD_VITEZA_FRANARE = UnityEngine.Random.Range(1.0f,
 25.0f); break;
    case 9: V.ANTIROLL = UnityEngine.Random.Range(1000.0f, 9000.0f);
 break;
  }
  return V;
}
```

Deși există diferite strategii pentru implementarea acestei funcții, ținând cont de limitele impuse de către logica de condus a vehiculelor și neuniformitatea ponderilor între acestea, mutația cu resetare aleatorie rămane ca ceea mai bună strategie. Idea operatorului de mutație în algoritmi EA este de a introduce un nivel de diversitate de la o generație la alta[6], pentru a evita convergera în aceași gamă restrânsă de soluții de fiecare dată când alogritmul este executat. Folosind o altă strategie precum mutația *inversă*, *amestecată* sau *swap*, ar fi existat posibilitatea de a genera soluții cu parametri în afara scopului propus, impactând în mod negativ fitnessul viitorilor generații.

9 Rezultate

Următoarele rezultate au fost obținute după diverse rulări ale programului folosind diferite configurații: prima tranșă de teste a fost executată folosind o populație α de 40 de mașini, și o a doua populație β având doar 20 de mașini (numărul mediu de mașini care participă la o oare-care cursă). Timpul maxim alocat configurării a fost setat la 35 de secunde, deoarece această valoare este destul de înaltă cât să permită soluțiilor să parcurgă întregul circuit, admitând doar o margine de eroare de maxim 2 secunde.

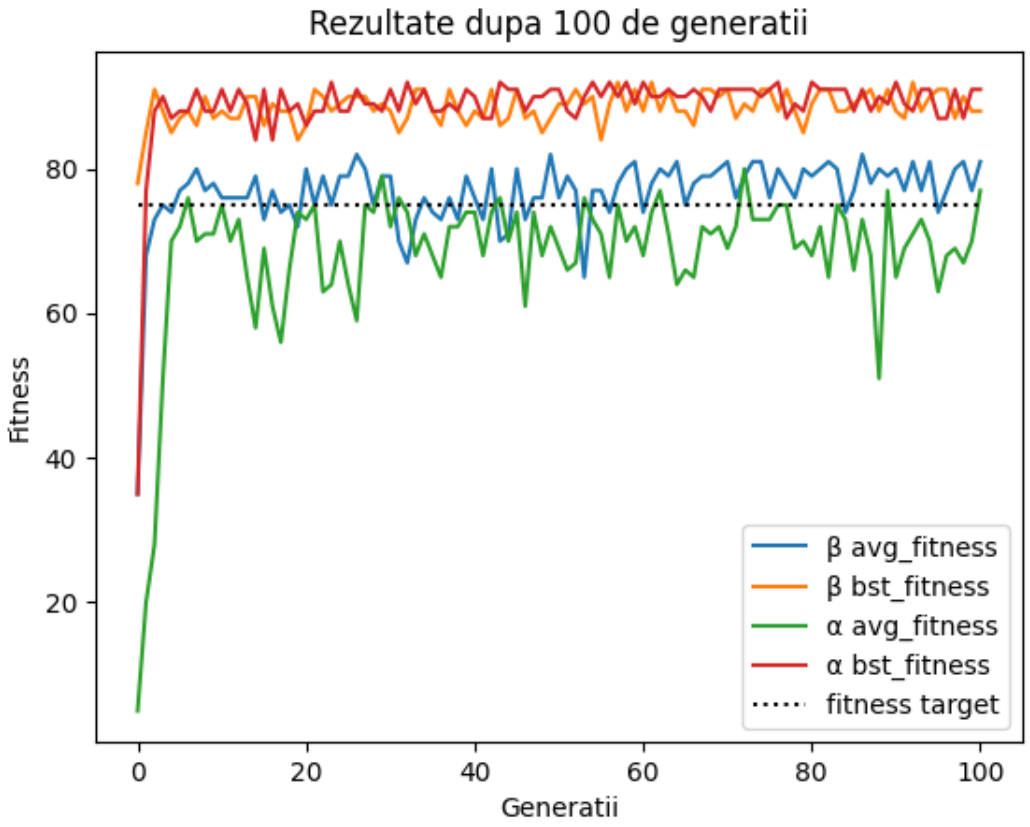

Figura 9: Valoriile de fitness maxim și mediu din 100 de generații cu o populație de 20 și 40

După 100 de generații, media totală a valorilor de fitness pentru populația α a rezultat fiind 68.3465, iar media populației β 76.6336. Valoarea maximă de fitness atinsă în medie este de 88.5247 și respectiv 88.8811. Rezultatele evidențiază clar că testarea făcută pe o populație mai mică (β) a produs rezultate mult mai bune în medie, depășind obiectivul de 75 fitness pentru majoritatea soluțiilor de pe parcursul generațiilor. Dar, concenctrându-ne doar pe soluțiile cele mai bune a acestor două seturi de rezultate, obținem seturi de caracteristici care nu diferă foarte mult între ele:

```
[BEST SOLUTION]
[POPULATIE: 40]
[GENERATIE: 100]
SENSIBILITATE_DIRECTIE: 0.0987701
PREVIZIUNE: 40.54642
CUPLU_MAX: 357.5551
UNGHI_VIRARE_MAX: 33.81911
CUPLU_FRANARE_MAX: 5421.413
ACCELERATIE_CURBA_MAX: 22.3498
FRANARE_CURBA_MAX: 7.320456
THRESHOLD_VITEZA_ACCELERATIE: 30.28899
THRESHOLD_VITEZA_FRANARE: 16.07583
ANTIROLL: 5605.962

[BEST SOLUTION]
[POPULATIE: 20]
[GENERATIE: 100]
SENSIBILITATE_DIRECTIE: 0.08247425
PREVIZIUNE: 39.61036
CUPLU_MAX: 372.549793
UNGHI_VIRARE_MAX: 54.9793
CUPLU_FRANARE_MAX: 5461.8435605.962
ACCELERATIE_CURBA_MAX: 25.28397
FRANARE_CURBA_MAX: 3.376127
THRESHOLD_VITEZA_ACCELERATIE: 33.34121
THRESHOLD_VITEZA_FRANARE: 14.88022
ANTIROLL: 4673.877
```

Concluzii

Scopul acestui proiect a fost de a vedea inițial dacă aplicarea algoritmilor evolutivi în contextul configurării mașinilor de cursă este posibilă. După diverse teste efectuate, rezultă că aceste metode sunt o modalitate validă de rezolva problema în cauză. Nu numai că implementarea discutată ar putea fi extinsă adăugând ulterior parametrii, circuite sau introducerea factorilor externi precum vântul, ploaia și temperatura, dar ideaa care stă la baza programării genetice ar putea fi extinsă către altre probleme sau concepte din lumea acestei tipologii sportive. Domeniul sporturilor cu motor reflectă necesitatea de soluții dinamice, care țin pasul cu evoluția continuă a condițiilor de curse și a avansamentului tehnologic.

Surse

Codul sursă: iulian-b/ICS-Project [github.com]
Mesh 3d circuit: Ryan Brands, monza-gp-model-1 [grabcad.com]
Mesh 3d mașină: edwinamadormeza, Red Bull EB6 2010 Free 3D model [cgtrader.com]

Referințe

[1] U. R. Bhatnagr, „Formula 1 Race Car Performance Improvement by Optimization of the Aerodynamic Relationship Between the Fron and Rear Wings", Teză de doct., 2014.

[2] F.I.A. (Federation Internationale de l'Automobile), „2023 Formula 1 Technical Regulations", în Doc. Technic, 2022.

[3] A. Mourad P. Delzell P. McCabe, „Automation of Data Analysis in Formula 1", în 2019.

[4] M. Komosinski, *Artificial Life and Nature-Inspired Algorithms*, Lectură, 2023.

[5] L. Dioșan, *Inteligența Artificială: Rezolvarea problemelor de căutare*, Lectură, 2020.

[6] J.E. Smith A.E. Eiben, *Introduction to Evolutionary Computing*, 2015.

[7] D. Zaharie, *Algoritmi metaeuristici - Curs 5 și 6*, Lectură, 2015.