# Car boy

-Rezolvarea problemei de optimizare a unei mașini cu vectori suport cu ajutorul unui algoritm evolutiv-

## Introducere

Dezvoltarea sistemelor autonome reprezintă una dintre direcțiile principale ale cercetării moderne în inteligența artificială. Probleme precum navigația autonomă, evitarea obstacolelor și luarea deciziilor în timp real necesită integrarea mai multor tehnici de învățare automată și optimizare.

În acest proiect se propune o soluție bazată pe mașini cu vectori suport (Support Vector Machines – SVM), ale căror parametri sunt optimizați folosind un algoritm evolutiv standard. Soluția este implementată într-un mediu de simulare, utilizând motorul Unity și un model de mașină.

Scopul principal este evaluarea eficienței algoritmilor evolutivi în optimizarea unui sistem decizional bazat pe SVM, aplicat unei probleme concrete de control.

## Descrierea problemei considerate

Problema abordată poate fi formulată ca una de luare a deciziilor de control pentru un vehicul autonom. Mașina trebuie să se deplaseze pe o pistă delimitată de pereți laterali, evitând coliziunile, maximizând distanța parcursă și minimizând timpul parcurs până la final.

La fiecare pas de timp, agentul:

- primește informații despre mediul înconjurător sub forma unor distanțe măsurate de senzori;
- trebuie să decidă una dintre acțiunile posibile: mers înainte, viraj la stânga sau viraj la dreapta.

Această problemă este transformată într-o problemă de clasificare, unde intrările sunt vectorii de distanțe, iar ieșirile sunt deciziile de control.

## Aspecte teoretice privind Mașinile cu Vectori Suport

Mașinile cu vectori suport sunt modele de învățare supervizată utilizate în principal pentru clasificare. Un SVM caută să determine un hiperplan care separă datele în spațiul caracteristicilor, maximizând marginea dintre clase.

În forma sa liniară, funcția de decizie a unui SVM este:

$$f(x) = w \cdot x + b$$

unde:

- $w$este vectorul de ponderi,
- $b$este termenul de bias,

- $x$este vectorul de intrare.

Semnul funcției $f(x)$determină clasa atribuită.

În cadrul proiectului, SVM-ul este utilizat ca element decizional, iar parametrii $w$și $b$sunt optimizați.

## Algoritmi evolutivi – noțiuni teoretice

Algoritmii evolutivi sunt metode de optimizare inspirate din selecția naturală. Aceștia lucrează cu o populație de soluții și aplică operatori genetici pentru a obține soluții din ce în ce mai bune.

Un algoritm evolutiv standard presupune următoarele etape:

1. inițializarea populației;
2. evaluarea fitness-ului;
3. selecția părinților;
4. crossover (recombinare);
5. mutație;
6. elitism

În acest proiect, algoritmul evolutiv este utilizat pentru optimizarea parametrilor SVM-urilor, fiecare individ reprezentând un set complet de clasificatoare.

## Mediul de simulare

Simularea este realizată în Unity, folosind asset-ul Prometeo Car Controller, care oferă un model fizic realist pentru deplasarea mașinii. Acest lucru permite evaluarea performanței agentului într-un mediu apropiat de unul real.

Pista este delimitată de pereți generați automat, iar coliziunile sunt detectate folosind sistemul de fizică Unity.

## Implementarea soluției

### 1. Senzorii de distanță

Mașina este echipată cu un set de senzori de tip raycasting, care măsoară distanța până la obstacole în mai multe direcții. Valorile sunt normalizate și formează vectorul de intrare pentru SVM-uri.

```
void CastRays()
{

    for (int i = 0; i < directions.Length; i++)
    {
        Vector3 worldDir = transform.TransformDirection(directions[i]);
        Ray ray = new Ray(transform.position, worldDir);
        if(Physics.Raycast(ray, out RaycastHit hit, sensorRange, detectionLayer))
        {
            distances[i] = hit.distance / sensorRange;


            if (debugMode) {
                rays[i][0].SetPosition(0, transform.position);
                rays[i][0].SetPosition(1, hit.point);

                rays[i][1].SetPosition(0, hit.point);
                rays[i][1].SetPosition(1, transform.position + worldDir * sensorRange);
            }
            else
            {
                rays[i][0].SetPosition(0, transform.position);
                rays[i][0].SetPosition(1, transform.position);
                rays[i][1].SetPosition(0, transform.position);
                rays[i][1].SetPosition(1, transform.position);
            }
        }
        else
        {
            distances[i] = 1f;

            if (debugMode)
            {
                rays[i][0].SetPosition(0, transform.position);
                rays[i][0].SetPosition(1, transform.position + worldDir * sensorRange);

                rays[i][1].SetPosition(0, transform.position + worldDir * sensorRange);
                rays[i][1].SetPosition(1, transform.position + worldDir * sensorRange);
            }
            else
            {
                rays[i][0].SetPosition(0, transform.position);
                rays[i][0].SetPosition(1, transform.position);
                rays[i][1].SetPosition(0, transform.position);
                rays[i][1].SetPosition(1, transform.position);
            }
```

Direcțiile sunt generate simetric în fața mașinii:

```
for(float angle = -90f; angle <= 90f; angle += angleStep)
{
    directions[i] = Quaternion.Euler(0, angle, 0) * transform.forward;
}
```

Acești senzori permit agentului să perceapă mediul și să anticipeze coliziunile.

## 2.Clasificatorul SVM

Fiecare SVM este definit prin: un vector de ponderi reale și un termen de bias.

```
public float Evaluate(float[] x)
{
    float sum = bias;
    for (int i = 0; i < x.Length; i++)
```

```
        sum += weights[i] * x[i];
    return sum;
}
```

Ponderile și bias-ul sunt inițializate aleator:

```
public void Randomize()
{
    for (int i = 0; i < weights.Length; i++)
        weights[i] = Random.Range(-1f, 1f);
    bias = Random.Range(-1f, 1f);
}
```

Acești parametri sunt ulterior optimizați de algoritmul evolutiv.

### 3. Driverul inteligent

Agentul autonom utilizează trei SVM-uri:

- unul pentru viraj stânga;
- unul pentru viraj dreapta;
- unul pentru mers înainte.

```
float leftScore = svmLeft.Evaluate(distances);
float rightScore = svmRight.Evaluate(distances);
float forwardScore = svmForward.Evaluate(distances);
```

Pentru fiecare pas de timp, sunt evaluate cele trei scoruri, iar acțiunea asociată scorului maxim este selectată. Astfel, decizia este rezultatul unui proces de clasificare.

```
if (leftScore > rightScore && leftScore > forwardScore)
{
    selectedDirection[LEFT] = true;
}
```

Această abordare transformă controlul într-o problemă de clasificare multi-decizie.

### 4. Reprezentarea individului – Brain

Un individ din populația evolutivă este reprezentat de clasa Brain, care conține lista de SVM-uri ale agentului. Această structură permite:

- copierea indivizilor;
- salvarea celui mai bun individ;

*string json = JsonUtility.ToJson(this, true);*
*File.WriteAllText(path, json);*

- încărcarea unei soluții din fișier.

**5. Algoritmul evolutiv**

Algoritmul evolutiv gestionează o populație de mașini, fiecare având propriul Brain. Fitness-ul fiecărui individ este definit ca distanța parcursă până la coliziune.

*rankedPopulation.Add((driver.GetSVM(), driver.GetDistanceTravelled()));*

Selecția este realizată prin turneu.

*List<SVM> parent = TournamentSelection(rankedPopulation);*

```
List<SVM> TournamentSelection(List<(List<SVM> genome, float fitness, float lapTime)> rankedPop)
{
    var best = rankedPop[Random.Range(0, rankedPop.Count)];

    for (int i = 1; i < tournamentSize; i++)
    {
        var candidate = rankedPop[Random.Range(0, rankedPop.Count)];
        if (candidate.fitness > best.fitness)
        {
            best = candidate;
        }
    }
    return CloneGenome(best.genome);
}
```

Crossover-ul este de tip aritmetic.

*svmChild.weights[w] = alpha * p1[i].weights[w] + (1f - alpha) * p2[i].weights[w];*

Mutația modifică ușor ponderile

*if (Random.value < mutationRate)*
*{*
*svm.weights[w] += Random.Range(-0.2f, 0.2f);*
*}*

Aceste operații asigură explorarea și exploatarea spațiului soluțiilor.

### 6. Interfața grafică

Interfața grafică permite:

- pornirea și oprirea simulării;
- modificarea vitezei de execuție;
- setarea dimensiunii populației;
- afișarea generației curente și a celui mai bun rezultat.

Aceasta facilitează analiza comportamentului algoritmului evolutiv.

## Rezultate experimentale

În urma rulării simulării, se observă o creștere progresivă a distanței maxime parcurse de agenți de la o generație la alta. Algoritmul evolutiv reușește să optimizeze parametrii SVM-urilor astfel încât agentul să evite mai eficient coliziunile.

Rezultatele demonstrează că soluțiile evoluate sunt net superioare celor inițiale, generate aleator.

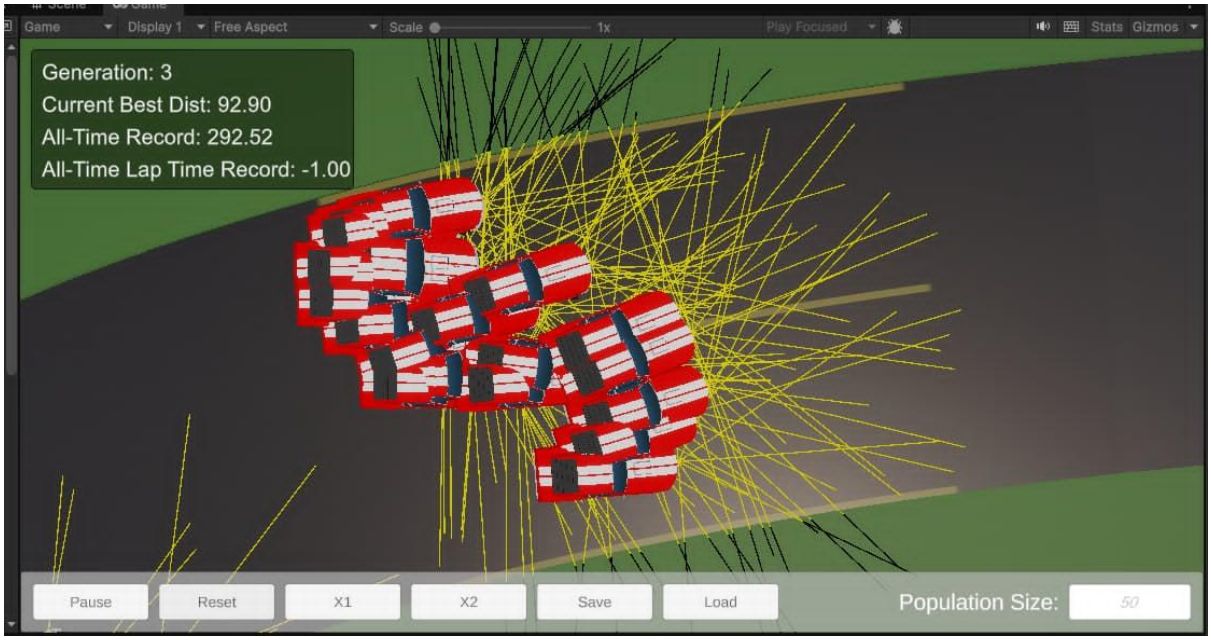


Fig. 1. Rezultatele după generația 2

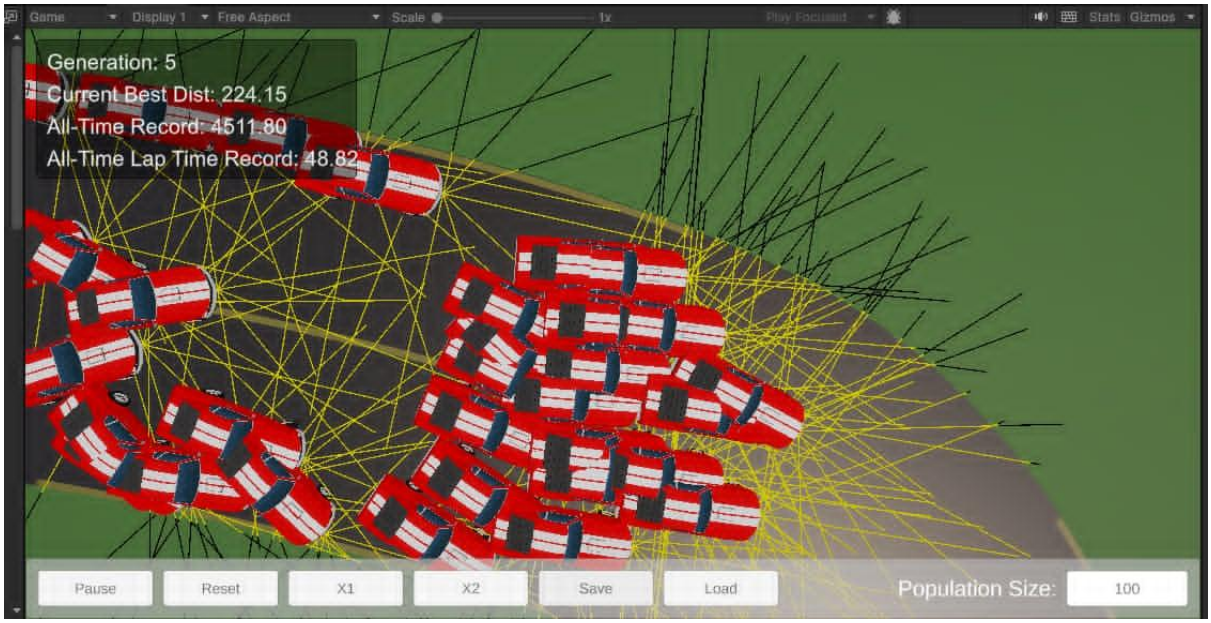


Fig. 2. Rezultatele după generația a 4 a

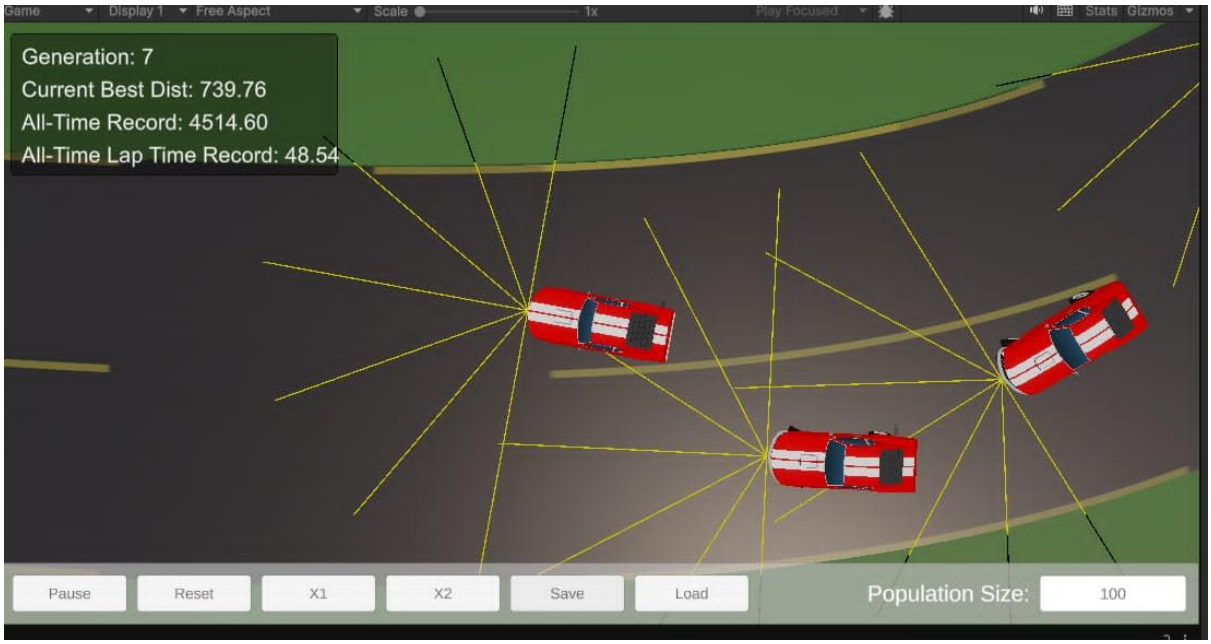


Fig. 3. Rezultate după generația a 6 a

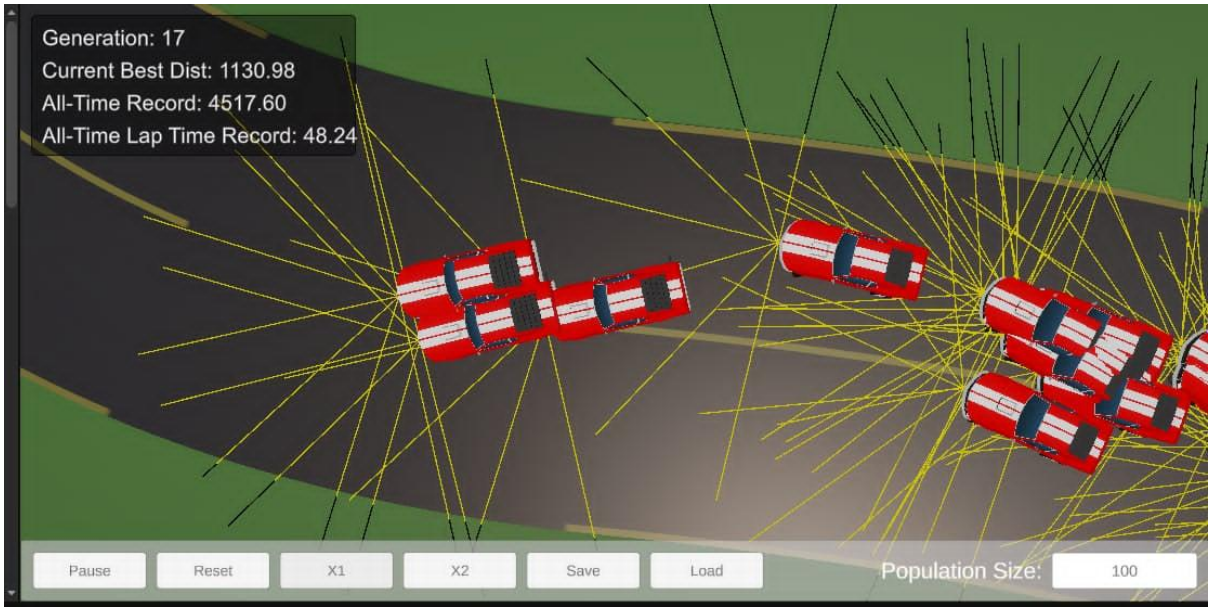


Fig. 4. Rezultat după generația a 16 a


Fig. 5. Rezultatele după generația a 80 a

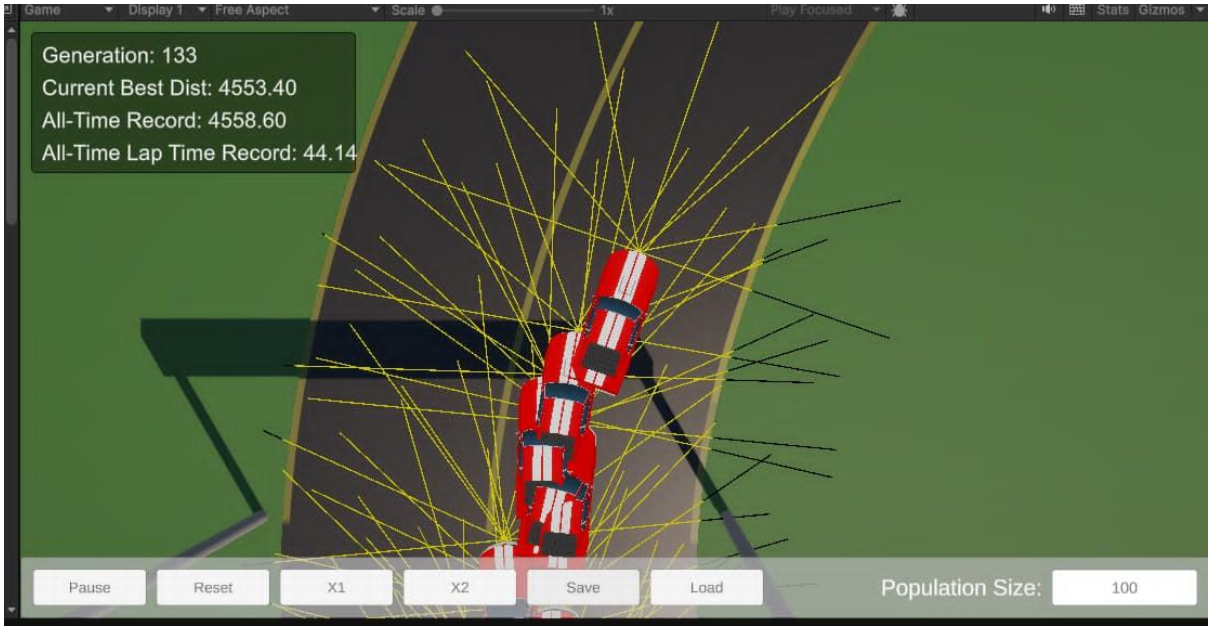


Fig. 6. Rezultatele dupa generația 133

Din figurile 1, 2, 3, 4, 5 și 6 se observă că, pe măsură ce algoritmul evoluează, performanța acestuia se îmbunătățește progresiv, indicând un proces clar de convergență către soluții din ce în ce mai bune.

## Concluzii

Proiectul demonstrează că algoritmii evolutivi pot fi utilizați cu succes pentru optimizarea mașinilor cu vectori suport într-o problemă practică de control. Integrarea dintre SVM, algoritmi evolutivi și simularea fizică oferă o platformă flexibilă pentru studierea comportamentului agenților autonomi.

Rezultatele obținute confirmă eficiența metodei și deschid posibilitatea extinderii proiectului către probleme mai complexe.

**Link Github:** https://github.com/mihaela-gabrielaghiata/car-boy

## Contribuția membrilor echipei

- Ghiata Mihaela – scena în Unity, integrarea mașinii, controlerul mașinii, pista, documentația
- Posmangiu Silviu – simularea populației și suport pentru SVM
- Alupului Diana – implementarea algoritmului genetic de antrenare

## Bibliografie

1. Prometeo Car Controller – Unity Asset Store https://assetstore.unity.com/packages/tools/physics/prometeo-car-controller-209444