Mecanism de menținere a unei formații – Metoda gradientului

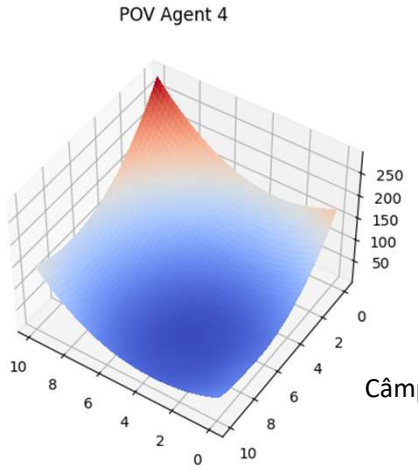
Stancu Tudor-Ștefan

Grupa 312

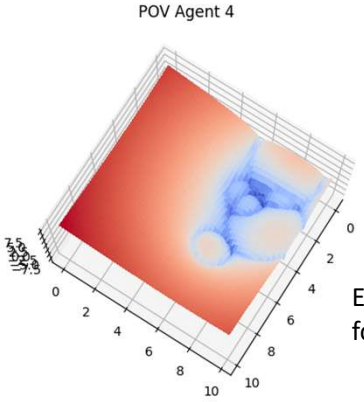
Algoritm 1:

- Asemănător temei 2 din laboratorul 7, am construit un câmp potențial pentru fiecare agent adăugând termenul de evitare a coliziunii: $\sum_{i \neq j} \left(|| p_i - p_j || - d_{ij} \right)^2$
- Astfel, pentru agentul A1, în jurul fiecărui agent vecin se va contura un cerc față de care agentul este penalizat, iar agentul va fii încurajat să se deplaseze către intersecțiile cercurilor.
- Nu am folosit viteza agenților.

Câmpul potențial și Comanda



Câmpul potențial



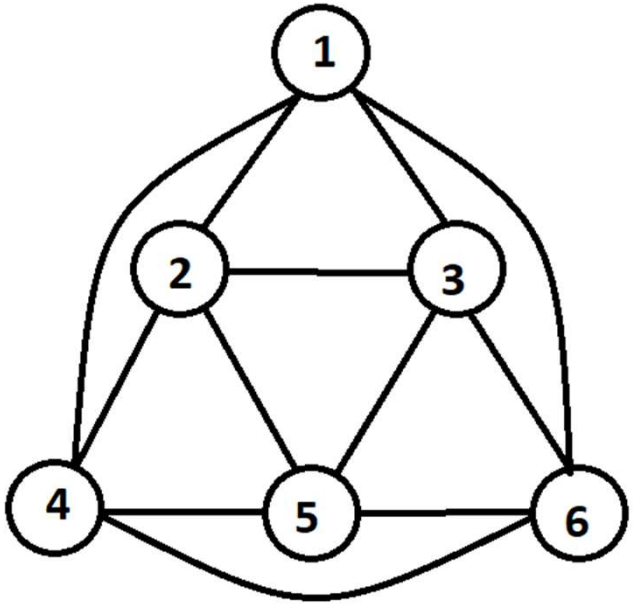
Exemplu exagerat folosind logaritm

Gradientul termenului de evitare a coliziunii:

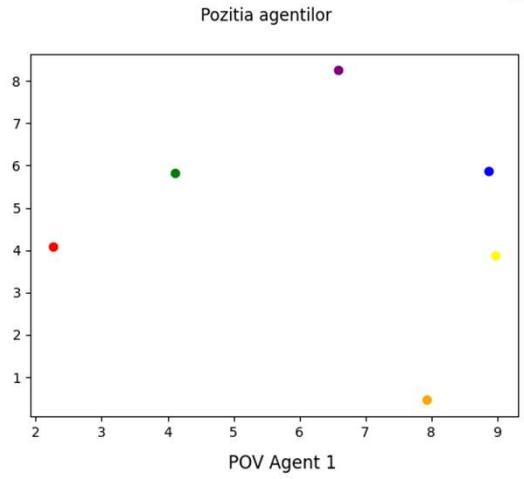
$$u = -\nabla P(p) = -\sum_i 2(\big|||p - a_i|| - d_{a_i}\big)\frac{p - a_i}{||p - a_i||}$$

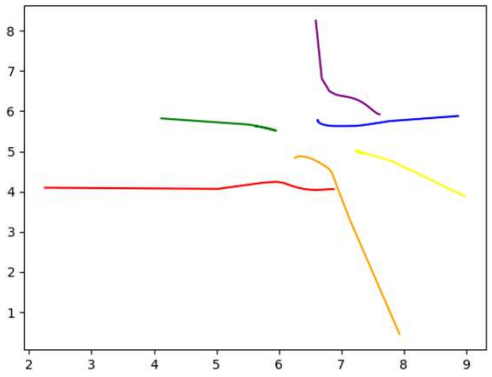
```
# Aplicare comanda
for i in range(len(agents)):
    u = 0
    a = np.array(agents[i][0:2])
    positions[i].append(a)
    for j in range(len(agents)):
        if goal[i][j] > 0:
            b = np.array(agents[j][0:2])
            u = u + 2 * (d(a, b) - goal[i][j]) / d(a, b) * (a - b)
    a = a - u * 0.1
    agents[i] = list(a) + [agents[i][2]]
```

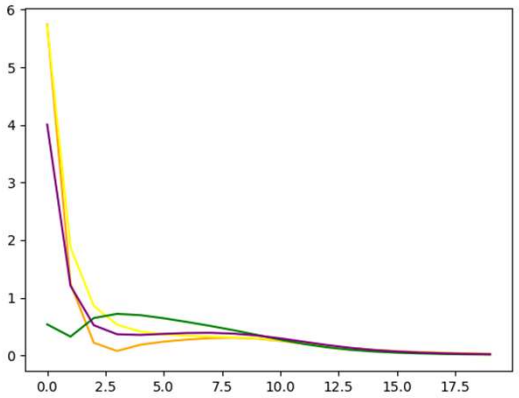
Am introdus datele pentru ca agenții să se poziționeze în felul următor:

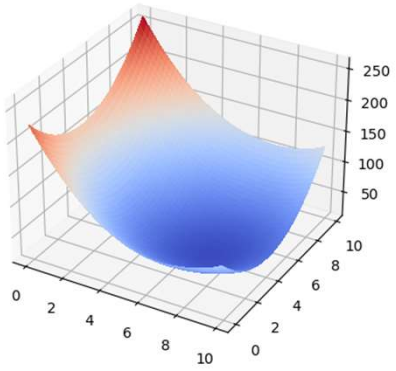


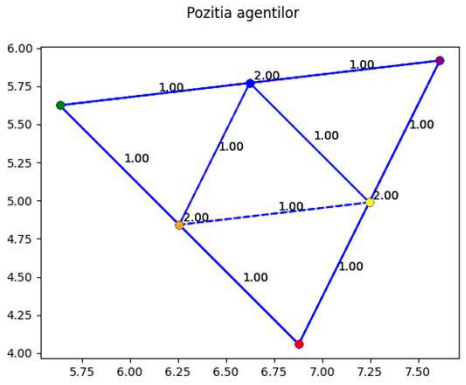
Simulare reușită:



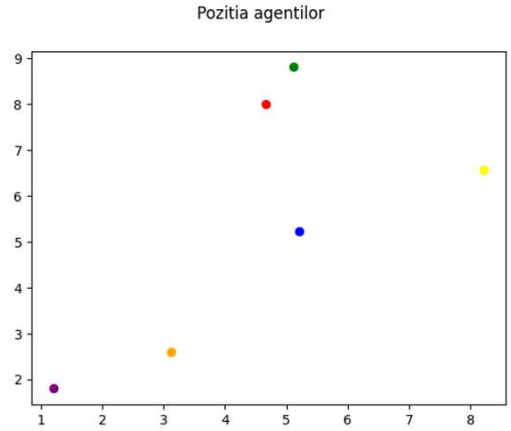


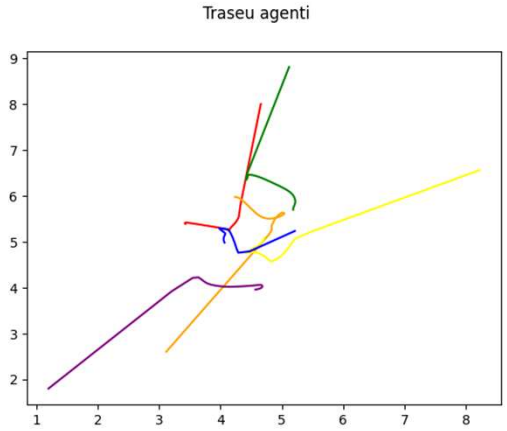


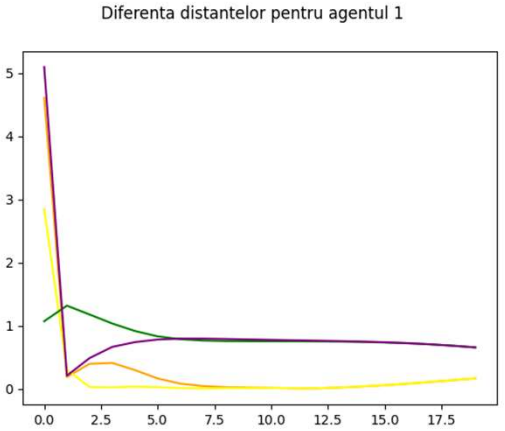


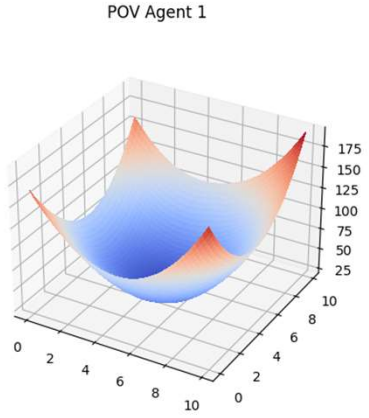


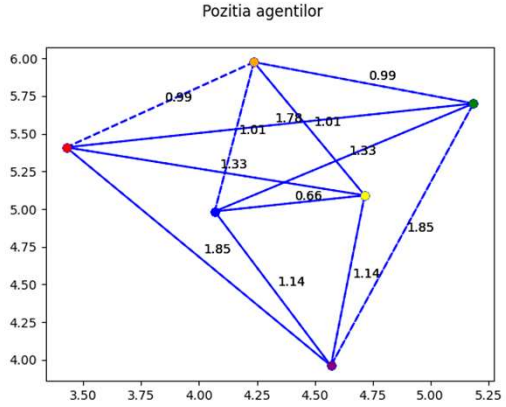
Simulare nereușită:

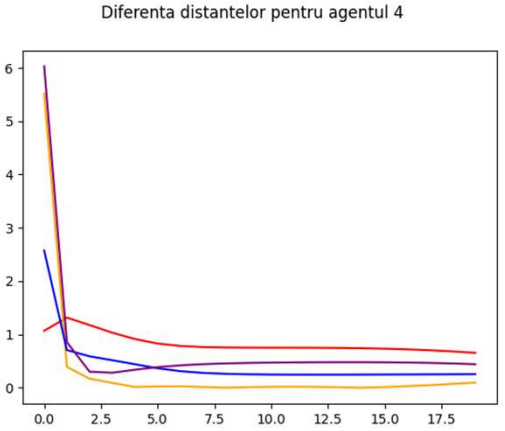




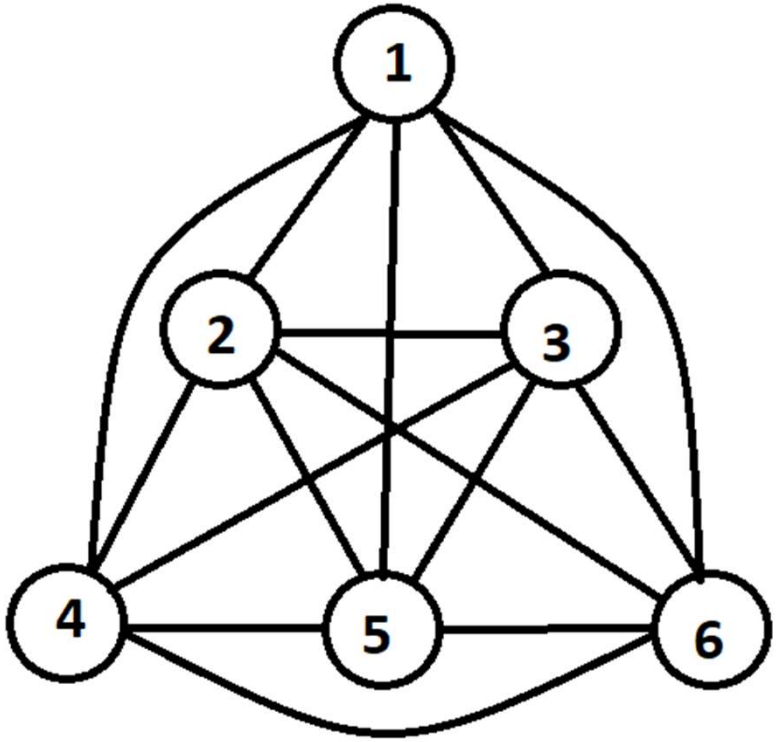




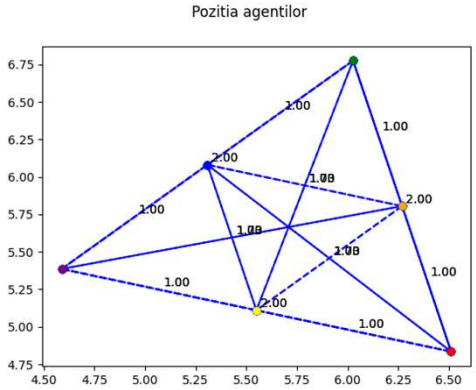




Am completat datele pentru toți vecinii pentru ca fiecare agent să aibe destinația mai precisă (să nu se mai formeze bazine locale):



Am avut simulări reușite în 100% din cazuri



Algoritm 2:

- Am folosit algoritmul din laboratorul 7 din codul formation-control.
- În loc să folosesc puncte și viteze relative, am folosit gradientul din Algoritmul 1. Agenții își urmăresc direcția și viteza, dar îi îndemn să se întoarcă către pantă, asemenea unei bile (cu inerție) lăsate libere. În plus definesc un nou câmp potențial pentru viteze.

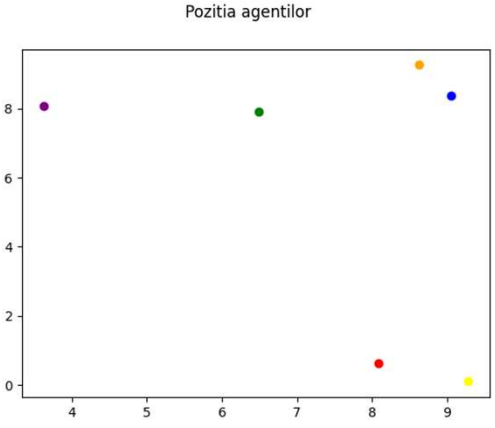
Comanda

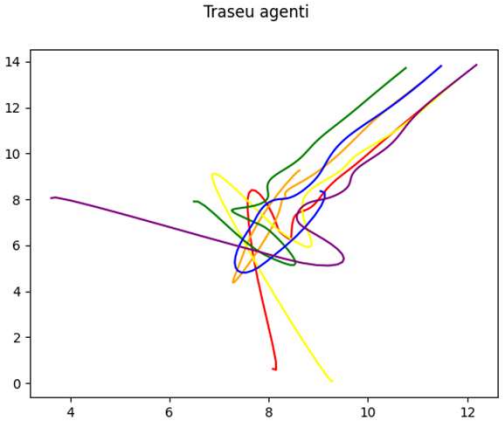
```
# Simulare step
for i in range(len(agents)):
    positions[i].append(agents[i])
    u = 0
    v = 0
    a = np.array(agents[i][:-1])
    va = np.array(vAgents[i])
    for j in range(len(agents)):
        b = np.array(agents[j][:-1])
        vb = np.array(vAgents[j])
        if goal[i][j] > 0:
            u = u + (a - b) * 2 * (d(a, b) - goal[i][j]) / d(a, b) * (a - b)) * goal[i][j]
            if d(va, vb) == 0:
                v = v
            else:
                v = v + 2 * (d(va, vb)) / d(va, vb) * (va - vb) * goal[i][j]
    comanda = - kp * u - kv * v
    # Aplicare comanda
    agents[i] = list((np.array(agents[i][:-1]) + T * np.array(vAgents[i]))) + [agents[i][2]]
    vAgents[i] = list(np.array(vAgents[i]) + T * comanda)
```

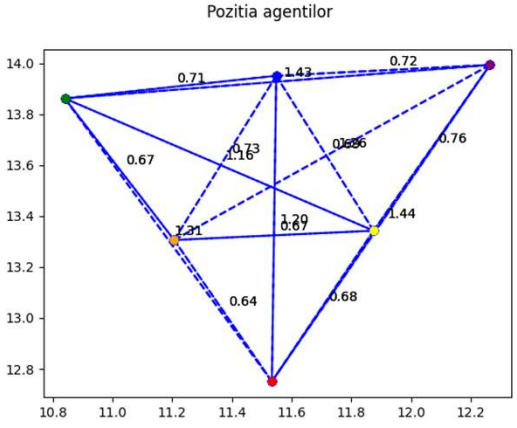
$$u_i = -k_p \sum_{j \in \mathcal{N}_i} \omega_{ij}(p_i - p_j - p_i^\star + p_j^\star)$$
$$- k_v \sum_{j \in \mathcal{N}_i} \omega_{ij}(v_i - v_j - v_i^\star + v_j^\star),$$

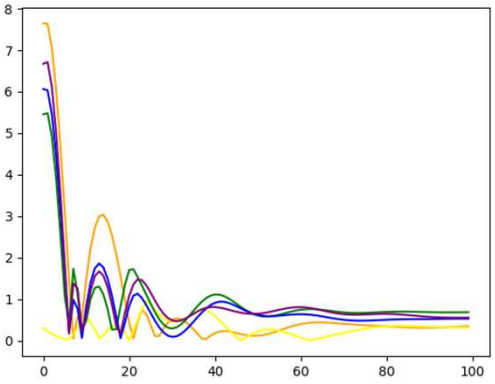
În loc să folosesc pozițiile și vitezele relative, am determinat vectorul conform gradientului

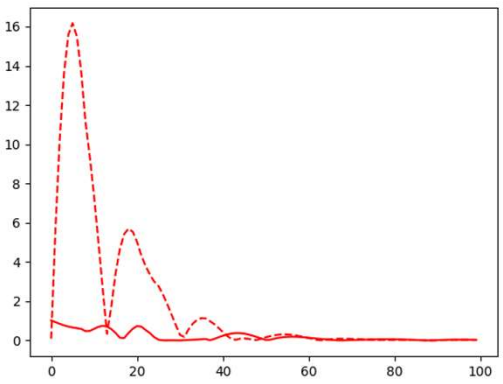
Rezultat: (steps = 100, kp = 0.3, kv = 0.1)











Concluzii

- Spre deosebire de algoritmul din laborator, metoda gradientului aduce agenții în formația dată fără să țină cont de direcție (vârful triunghiului poate fii îndreptat în orice direcție).
- Trebuie introduse informații pentru fiecare pereche de agenți, nu doar vecini, pentru a nu întâlni bazine locale.

Bibliografie

- F. Stoican, Planificarea mișcării în formație, Laborator 7-8 (SPER) (2023)
- F. Stoican, Python source code: formation-control.py (2023)
- F. Stoican, Python source code: potential-field.py (2023)
- Y. Singer, Advanced Optimization, Lecture 9 (2016)