Agenti de Reinforcement Learning pentru Geometry Dash

Mădălin Ioana
Rareș Stancu

Ianuarie 2026

Contents

1 Introducere

1.1 Motivatie

Geometry Dash este un joc de platforming cu fizica simpla dar care necesita timing foarte precis. Jocul este interesant pentru reinforcement learning deoarece:

- Spatiul de actiuni este minimal (doar saritura/nu saritura)
- Environment-ul este determinist
- Feedback-ul este clar (ai trecut de obstacol sau nu)
- Dar timing-ul trebuie sa fie foarte precis

Scopul proiectului este sa implementam si sa comparam 4 algoritmi diferiti de RL pe acest joc: Q-learning, SARSA, DQN si PPO.

1.2 Obiective

1. Crearea unui environment custom bazat pe Gymnasium pentru Geometry Dash
2. Implementarea a 4 algoritmi de RL: 2 tabulari (Q-learning, SARSA) si 2 deep (DQN, PPO)
3. Antrenarea si evaluarea agentilor pe acelasi environment
4. Analiza comparativa a performantelor si limitarilor fiecarui algoritm
5. Identificarea si explicarea motivelor pentru care 3 din 4 agenti nu au invatat bine

2 Environment design

2.1 Spatiul de observatie

Am implementat un sistem de tip LIDAR pentru ca agentul sa perceapa environment-ul:

- **30 de scanuri** distribute radial in fata jucatorului
- **Range maxim**: 600 pixeli
- Pentru fiecare scan, agentul primeste:
  - Distanta pana la cel mai apropiat obstacol (normalizata 0-1)
  - Tipul obstacolului (spike/platform)
  - Inaltimea obstacolului (normalizata)
- Plus starea interna: `on_ground` (0/1)

Total: **91 de valori continue** in spatiul de observatie.

2.2 Spatiul de actiuni

Spatiul de actiuni este discret si minimal:

- **Actiunea 0**: Nu face nimic
- **Actiunea 1**: Sari (doar daca e pe pamant)

2.3 Reward shaping

Sistemul de reward a fost extins iterativ pentru a rezolva probleme de comportament:

- **+0.05**: Pentru fiecare frame de supravietuire
- **+10.0**: Pentru trecerea cu succes de un obstacol
- **-10.0**: Pentru coliziune (terminare episod)
- **-1.0**: Penalizare pentru sarituri inutile (in aer sau fara obstacol apropiat)
- **+0.1**: Bonus pentru a sta pe pamant (incentiveaza sa nu sara constant)

Ultimele doua componente au fost adaugate dupa ce am observat ca agentii tabulari invatau sa sara constant.

3 Algoritmi implementati

3.1 Q-Learning

3.1.1 Descriere

Q-learning este un algoritm tabular off-policy care invata functia Q optimal:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{1}$$

3.1.2 Implementare

- **Discretizarea spatiului de stari**: Am folosit quantile-based discretization pentru a converti cele 91 de observatii continue in 10 bins fiecare
- **Hiperparametri**:
  - Learning rate: $\alpha = 0.1$
  - Discount factor: $\gamma = 0.99$
  - Epsilon decay: de la 1.0 la 0.01 in 10000 episoade
- **Antrenare**: 15000 episoade

3.1.3 Probleme intampinate

Principala problema este **explozia dimensionala**. Cu 91 features discretizate in 10 bins fiecare, spatiul teoretic de stari este 10^{91}, care este imposibil de explorat. In practica, agentul exploreaza doar o fractiune minuscula din spatiul de stari, ceea ce duce la:

- Generalizare foarte slaba
- Comportament inconsistent in stari nevazute
- Convergenta lenta si instabila

3.2 SARSA

3.2.1 Descriere

SARSA este un algoritm tabular on-policy similar cu Q-learning, dar foloseste actiunea efectiv luata:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)] \tag{2}$$

3.2.2 Implementare

Implementarea este identica cu Q-learning in privinta discretizarii si hiperparametrilor, doar regula de update difera.

3.2.3 Probleme intampinate

SARSA sufere de aceleasi probleme ca Q-learning: explozie dimensionala si lipsa de generalizare. In plus, fiind on-policy, este mai conservator si mai lent la invatare.

3.3 Deep Q-Network (DQN)

3.3.1 Descriere

DQN foloseste o retea neurala pentru a aproxima functia Q, rezolvand problema exploziei dimensionale:

$$Q(s,a;\theta) \approx Q^*(s,a) \tag{3}$$

3.3.2 Arhitectura retelei

Listing 1: Arhitectura DQN

```
Input: 91 features
Layer 1: Dense(256) + ReLU
Layer 2: Dense(128) + ReLU
Layer 3: Dense(64) + ReLU
Output: Dense(2) - Q values pentru cele 2 actiuni
```

3.3.3 Hiperparametri

- **Learning rate**: 0.0001 (Adam optimizer)
- **Discount factor**: $\gamma = 0.99$
- **Replay buffer size**: 100,000 tranzitii
- **Batch size**: 64
- **Target network update**: La fiecare 1000 de pasi
- **Epsilon decay**: De la 1.0 la 0.01 in 50,000 de pasi
- **Antrenare**: 30,000 episoade

3.3.4 Performanta

DQN este singurul agent care a invatat sa joace bine:

- **Mean score**: 37.26 de puncte
- **Max score**: 157 de puncte
- **Std deviation**: 23.12

Experience replay si target network-ul permit DQN sa generalizeze mult mai bine decat algoritmii tabulari. Agentul a demonstrat ca poate invata sa treaca constant de 30-40 de puncte, cu performante maxime de peste 150.

3.4 Proximal Policy Optimization (PPO)

3.4.1 Descriere

PPO este un algoritm policy-based care optimizeaza direct politica:

$$L^{CLIP}(\theta) = E_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{4}$$

3.4.2 Implementare

Am folosit implementarea din Stable-Baselines3 cu hiperparametrii default:

- **Learning rate**: 0.0003
- **N steps**: 2048
- **Batch size**: 64
- **Antrenare**: 1,000,000 timesteps

3.4.3 Probleme intampinate

PPO a avut performante slabe pe acest environment din mai multe motive:

- **Sparse rewards**: Recompensele sunt concentrate (la trecerea de obstacol), ceea ce face grea estimarea advantage-ului
- **Timing precis**: PPO invata o politica stochastica, dar jocul necesita actiuni deterministe la momentele exacte
- **Sample efficiency**: PPO necesita multe mai multe samples decat DQN pentru a converge, iar in environment-uri cu timing critic nu este ideal
- **Lipsa de explorare**: On-policy learning limiteaza explorarea comparativ cu epsilon-greedy

4 Rezultate si analiza

4.1 Metrici de performanta

Am evaluat fiecare agent pe 500 de episoade de test pentru a obtine statistici robuste. Rezultatele sunt in tabelul de mai jos:

Table 1: Performanta finala a agentilor (500 episoade)

Agent	Mean Score	Std Dev	Max Score	Min Score
DQN	37.26	23.12	157	11
Q-Learning	11.99	5.24	37	7
SARSA	11.07	4.97	46	7
PPO	12.30	5.60	51	7

4.2 Curbe de invatare

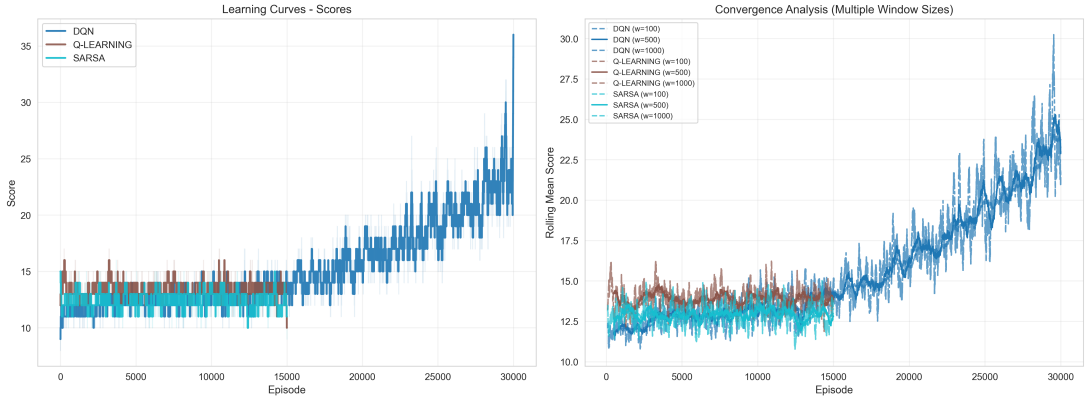

Figure 1: Evolutia rewardului pe parcursul antrenamentului. Se observa ca doar DQN converge catre o politica buna, ajungand la scoruri medii de 37. Q-learning si SARSA au convergenta instabila si raman la 11-12.

4.3 Distributia scorurilor

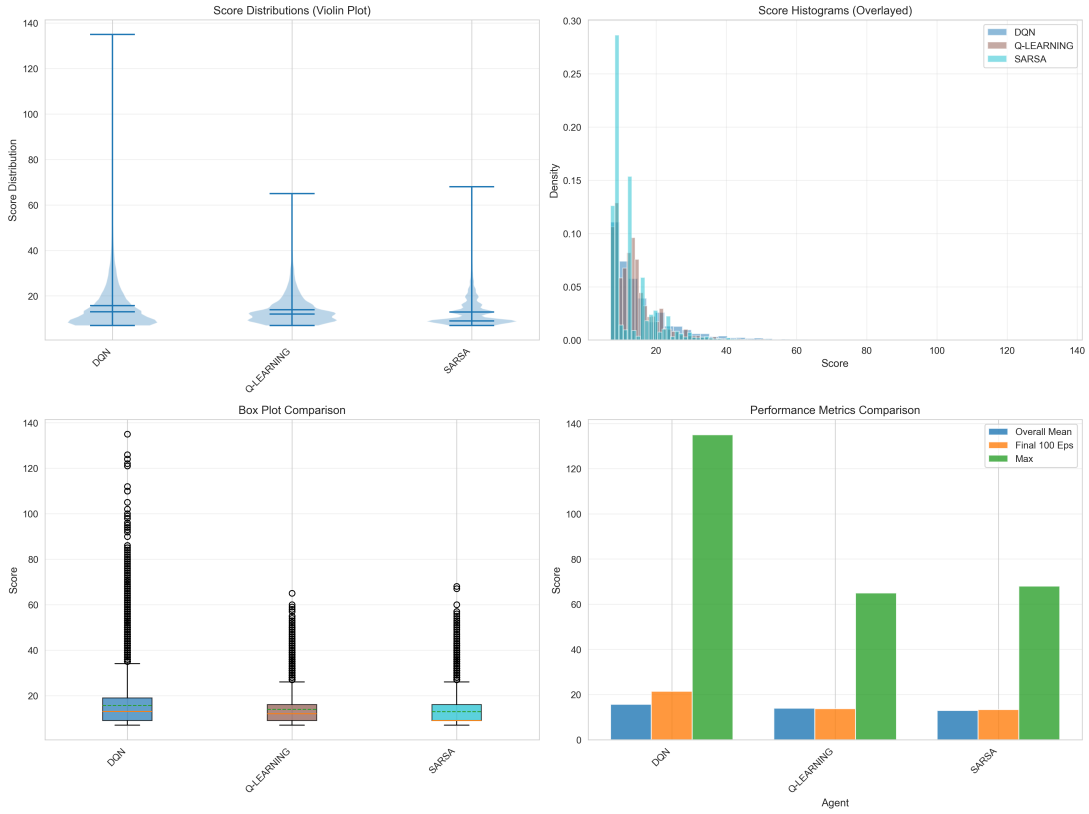

Figure 2: Distributia scorurilor obtinute in evaluarea finala (500 episoade). DQN are o distributie larga centrata in jurul valorii 30-40, cu varful la peste 150 obstacole. Algoritmii tabulari si PPO au scoruri concentrate in jurul valorii 10-12, cu maxime de 40-50.

4.4 Analiza convergentei

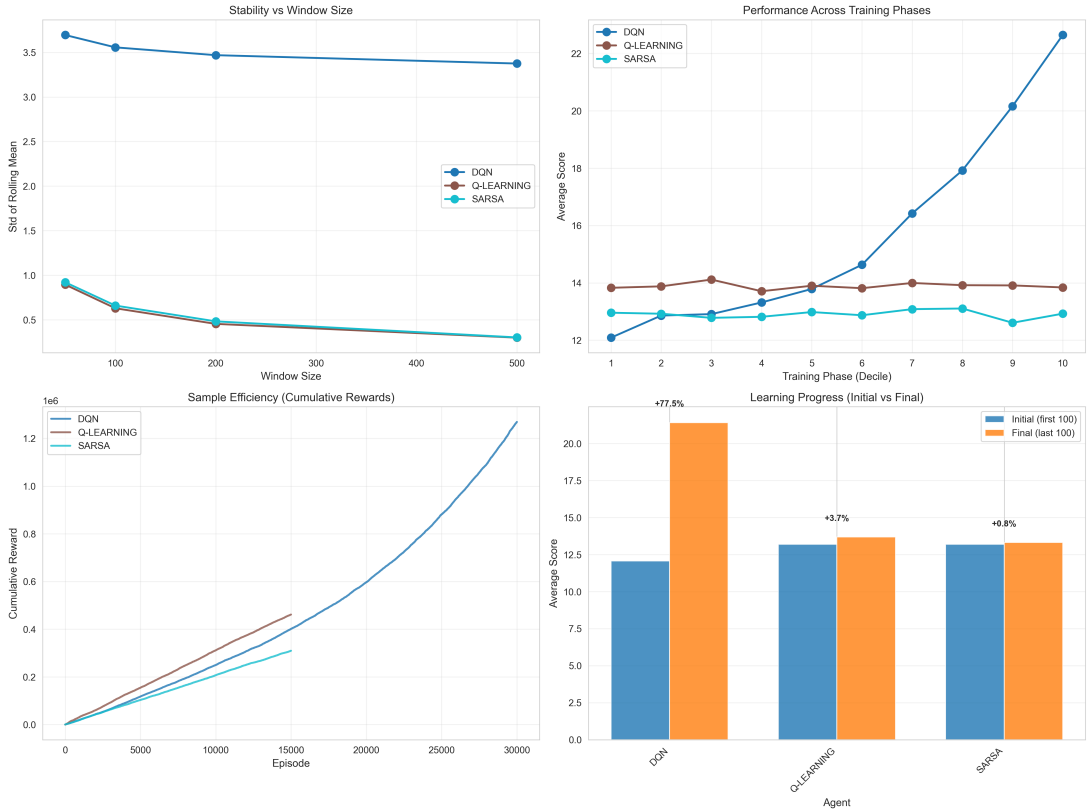

Figure 3: Media rulanta a rewardurilor (window=100 episoade). DQN arata o convergenta clara dupa aproximativ 15000 episoade, ajungand consistent la scoruri de 30-40 obstacole. Algoritmii tabulari oscileaza constant in jurul valorii 10-12.

4.5 Radar chart - comparatie multi-dimensionala

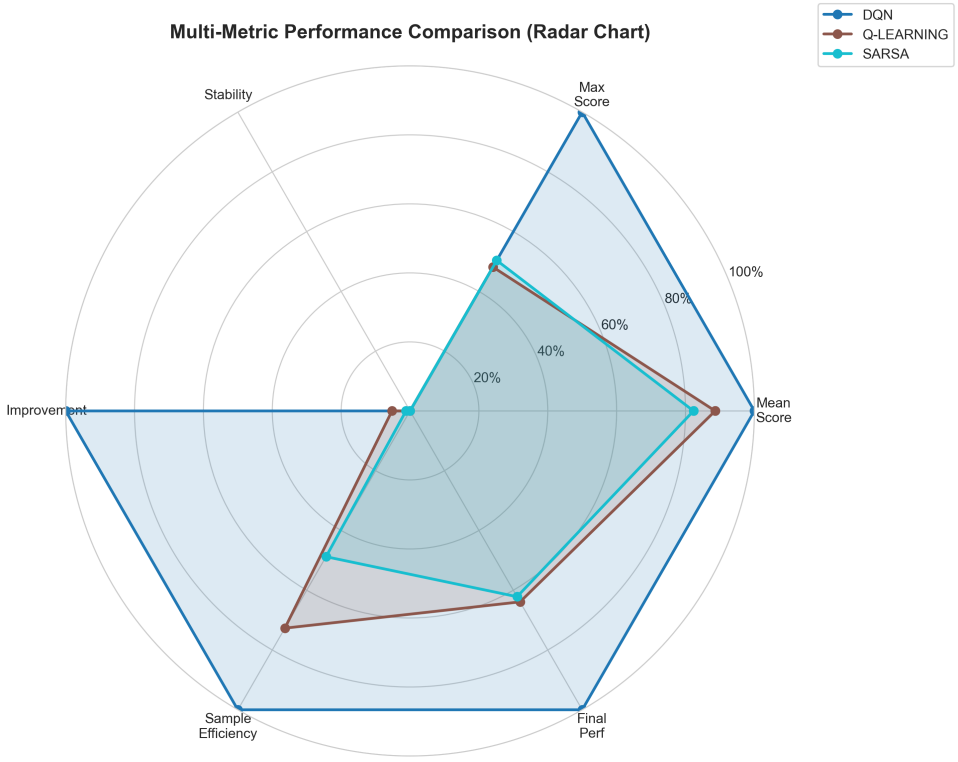

Figure 4: Comparatie multi-dimensionala a agentilor. DQN domina pe toate dimensiunile: performanta medie, stabilitate, viteza de invatare si scor maxim.

4.6 Heatmap - reward pe episoade

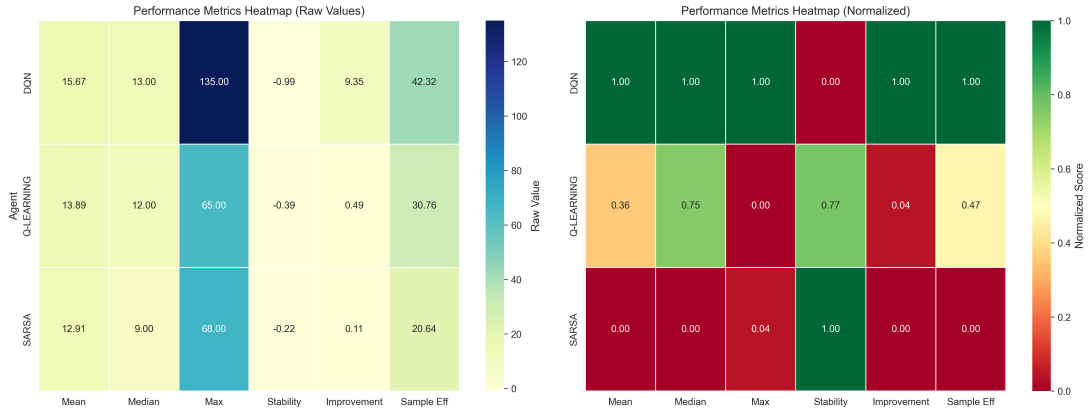

Figure 5: Heatmap al evolutiei rewardurilor. Culorile mai calde indica rewarduri mai mari. DQN arata o tranzitie clara catre zone de reward mai mare dupa 10000-15000 episoade.

5 Limitari si probleme identificate

5.1 De ce Q-Learning si SARSA au esuat

Algoritmii tabulari au esuat din urmatoarele motive:

1. **Explozie dimensionala**:
   - Spatiul de stari 10^{91} este imposibil de explorat
   - Chiar cu discretizare agresiva, majoritatea starilor ramand nevazute
   - Tabelul Q ramane foarte rar (sparse)
2. **Lipsa de generalizare**:
   - Q-learning nu poate generaliza intre stari similare
   - O stare noua (chiar si foarte asemanatoare cu una vazuta) incepe cu Q=0
   - Necesita sa invete fiecare configuratie de obstacole separat
3. **Discretizare grosolana**:
   - Pentru a reduce dimensiunea spatiului, am folosit doar 10 bins
   - Timing-ul precis se pierde in discretizare
   - Doua stari foarte diferite pot ajunge in acelasi bin
4. **Comportament constant jump**:
   - Initial, agentii invatau sa sara constant
   - Am adaugat penalizare -1.0 pentru sarituri inutile
   - Chiar si asa, generalizarea ramane foarte slaba

5.2 De ce PPO nu a functionat bine

PPO, desi este un algoritm modern si puternic, nu s-a descurcat bine pe acest task:

1. **Sample efficiency scazuta**:
   - PPO este on-policy si arunca datele vechi
   - Necesita mult mai multe samples decat DQN cu replay buffer
   - 1M timesteps nu au fost suficienti
2. **Politica stochastica vs timing determinist**:
   - PPO invata o distributie de probabilitate peste actiuni
   - Geometry Dash necesita actiuni deterministe la momente exacte
   - Sampling-ul stochastic introduce variabilitate nedorita
3. **Credit assignment dificil**:
   - Rewardurile sunt sparse (doar la trecerea de obstacol)

- Advantage estimation devine greu de estimat corect
- PPO se bazeaza pe estimari bune de advantage pentru update-uri stabile

4. **Explorare limitata**:
   - PPO exploreaza prin stochasticitatea politicii
   - Pe environment-uri cu timing critic, explorarea aleatoare nu ajuta
   - DQN cu epsilon-greedy are explorare mai structurata

5.3 De ce DQN a functionat

DQN a reusit sa invete din urmatoarele motive:

1. **Generalizare prin retea neurala**:
   - Retea neurala poate generaliza intre stari similare
   - Nu mai avem problema exploziei dimensionale
   - Poate invata features relevante automat
2. **Experience replay**:
   - Pastreaza si refoloseste experienta trecuta
   - Rupe corelatia dintre samples consecutive
   - Foarte sample efficient
3. **Politica determinista in evaluare**:
   - La testare foloseste argmax(Q) - fully deterministic
   - Perfect pentru timing precis
4. **Target network**:
   - Stabilizeaza antrenarea
   - Previne oscilatiile in estimarile Q

6 Concluzii

6.1 Rezumat

Proiectul a demonstrat ca alegerea algoritmului de RL este critica si depinde foarte mult de natura task-ului:

- **Algoritmi tabulari** (Q-learning, SARSA) ajung la performante modeste (12 obstacole) dar esueaza sa generalizeze mai departe
- **DQN** a invatat sa joace cu adevarat bine, ajungand la o medie de 37 scor si maxim 157, demonstrand capacitate clara de generalizare

- **PPO** are performante similare cu algoritmii tabulari (12 obstacole), dar nu reuseste sa rivalizeze DQN pe acest task

Diferenta substantiala de performanta (DQN: 37.26 vs restul: 11-12) arata ca function approximation prin retele neurale este esentiala pentru acest tip de environment cu spatiu de stari mare.

6.2 Interpretarea rezultatelor

Dupa evaluarea extinsa pe 500 episoade, observam urmatoarele:

1. **DQN domina clar**: Cu o medie de 37.26 si maxim 157, DQN a invatat strategii complexe de navigare. Scorurile ajung constant peste 30-40, indicand o politica stabila.

2. **Algoritmii tabulari plateaza**: Q-learning (11.99) si SARSA (11.07) ajung la un plafon din cauza exploziei dimensionale. Nu pot generaliza dincolo de configuratiile vazute direct in training.

3. **PPO nu converge eficient**: Cu 12.30 medie, PPO e usor mai bun decat algoritmii tabulari, dar foarte departe de DQN. Sample efficiency scazuta si politica stochastica il limiteaza pe acest task.

4. **Variabilitate**: DQN are std dev de 23.12 (comparat cu 5 pentru restul), indicand ca uneori face greseli, dar poate atinge performante exceptional de inalte.

6.3 Lectii invatate

1. **Reward shaping este esential**: Initial agentii invatau comportamente suboptimale (constant jumping). Penalizarile si bonusurile au ghidat invatarea.

2. **Dimensiunea spatiului de stari conteaza**: Pentru spatii mari continue, function approximation (retele neurale) este obligatoriu.

3. **Sample efficiency**: Pe environment-uri greu de explorat, off-policy cu replay (DQN) bate on-policy (PPO).

4. **Determinism vs stochastic**: Task-uri cu timing precis necesita politici mai deterministe.

6.4 Concluzie finala

Proiectul a reusit sa implementeze si sa compare 4 algoritmi diferiti de RL pe un joc de platforming. Rezultatele arata clar ca DQN este cel mai potrivit pentru acest tip de task, in timp ce algoritmii tabulari si PPO au limitari semnificative. Analiza detaliata a limitarilor ofera un insight valoros in alegerea algoritmului potrivit pentru diferite tipuri de probleme de RL.