Învățarea de către un agent a jocului Snake folosind algoritmul Q-Learning

Universitatea Tehnică „Gheorghe Asachi” din Iași

Facultatea de Automatică și Calculatoare din Iași

Autori
Daniel Rotariu
Eduard Pușcașu

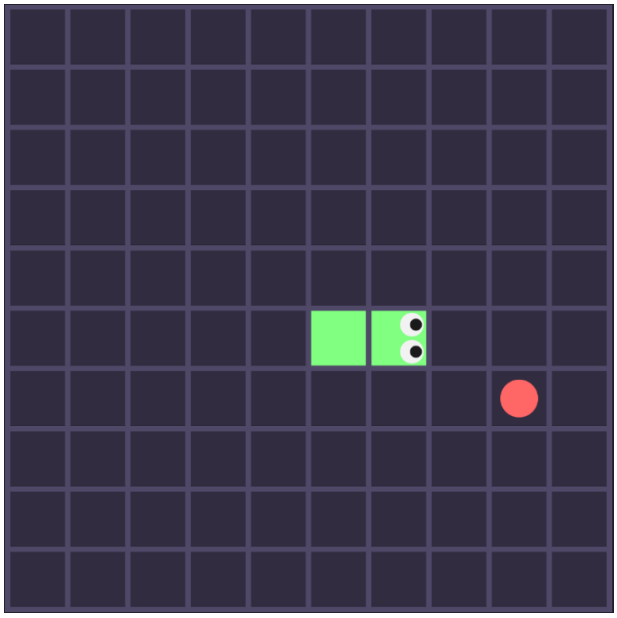
1. Introducere

Snake este un joc în care jucătorul manevreaza capătul unei linii în creștere, adesea cu tematica unui șarpe. Jucătorul trebuie să împiedice șarpele să se ciocnească atât de alte obstacole, cât și de el însuși. Scorul unui jucător este dat de numărul de bucăți de mancare consumate. Fiecare mancare consumată face ca șarpele sa creasca în lungime, facand astfel mai dificila manevrarea acestuia.

În aceasta lucrare ne propunem sa folosim algortimul Q-Learning pentru a antrena un agent la jocul Snake, scopul acestui agent fiind de a obține un scor cat mai mare.

2. Descrierea problemei

Avem un mediu 2D format din 3 componente principale: sarpele, mancarea si marginile spatiului in care sarpele se poate misca.

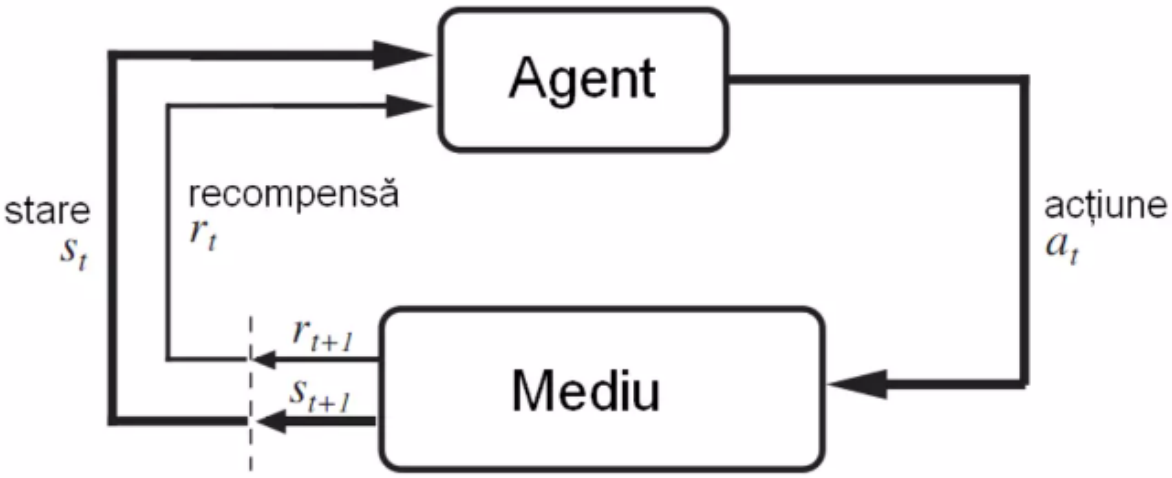


In aceasta problema, scopul nostru este de a antrena agentul astfel incat acesta sa faca deciziile corecte în ceea ce privește mișcarea sarpelui pentru a se îndrepta spre bucata de mancare existentă la un moment dat în timp ce evita sa iasă în afara spațiului permis sau sa se ciocnească de el însuși.

3. Aspecte teoretice privind algoritmul

- **Învățare cu întărire**

  Este o arie a învățării automate ce are ca obiectiv maximizarea recompensei primite de catre un agent care ia o serie de decizii într-un mediu dinamic.



Pașii făcuți de un agent sunt următorii:

a. Observa mediul
b. Folosește o strategie pentru a decide ce acțiune va alege în funcție de starea mediului
c. În urma acțiunii alese anterior ajunge într-o nouă stare
d. Primeste o recompensa sau o penalizare
e. Învață din experiența sa și își modifica strategia
f. Itereaza pana cand găsește o strategie optimă

Exista doua tipuri de algoritmi de învățare prin întărire:

a. Bazati pe model: învață modelul de tranziții și recompense și îl folosește pentru a descoperi politica optimă
b. Fara model: descoperă politica optimă fără a învăța sau estima modelul

- **Algoritmul Q-Learning**

Algoritmul Q-Learning este un algoritm de învățare prin întărire fără model. Acest algoritm va găsi cea mai buna secvență de acțiuni pornind de la starea în care se afla agentul. In funcție de starea mediului în care se afla agentul, acesta va alege următoarea acțiune fără a se baza pe o politică existenta, ci bazandu-se doar pe experiența sa.

După un număr de n pași în viitor, agentul va fi nevoit sa aleaga următoarea sa acțiune, ponderea recompensei acestei acțiuni va fi γ^n, unde γ(discount factor) reprezinta un numar in intervalul [0,1] și are ca efect alegerea unei acțiuni care va aduce o recompensa mai mare într-un număr cat mai mic de pași.

Așadar, algoritmul are o funcție care calculeaza recompensa primită de agent în funcție de starea în care se afla și de acțiunea pe care o poate lua.

$$Q : S \times A \rightarrow R$$

Valorile acestei funcții sunt stocate într-o structură de date numita Q-Table.

Inainte ca procesul de invatare sa inceapa, aceasta tabela este initializata cu valori arbitrare fixe. Apoi, la orice moment de timp *t*, agentul care se afla în starea s_t alege o actiune a_t, observa recompensa r_t primită si ajunge într-o nouă stare s_{t+1} care poate depinde de starea anterioara și de acțiunea aleasă. In final tabela Q este actualizata. Ecuatia care sta la baza acestui algoritm este *Ecuatia lui Bellman*.

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{current value}} \right)}^{\text{temporal difference}}$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

unde r_t reprezinta recompensa primita de catre agent trecand din starea s_t in starea s_{t+1} prin actiunea a_t.

Un episod al algoritmului se termina cand starea s_{t+1} este o stare terminala. Pentru toate starile finale, s_f tabela Q nu se actualizează.

- **Influenta parametrilor**
  - Learning rate - α**:** determina în ce masura informațiile noi invatate suprascriu informațiile vechi. Valoarea 0 pentru acest parametru va determina ca agentul sa nu învețe nimic, bazandu-se doar pe cunoștințele proprii, iar valoarea 1 va determina agentul să folosească doar informațiile cele mai recente.
  - Discount factor - γ: determina importanta viitoarelor recompense. Valoarea 0 va determina agentul ca acesta sa aleaga mereu recompensele curente și să le ignore pe cele din stările viitoare, iar valoarea 1 va determina agentul să ia în considerare cea mai mare recompensa cumulata din stările viitoare.

4. Modalitatea de rezolvare

Performanța acestui algoritm este puternic influențată de modul în care ne definim stările mediului. Un numar prea mic de stari nu ar cuprinde toate situațiile în care s-ar putea regasi agentul, asta ar duce la alegerea unei acțiuni nepotrivite și la obținerea în medie a unui scor mic. Un număr prea mare de stări poate duce la un consum inutil de resurse.

Abordarea aleasă de noi defineste o stare ca o combinație a următoarelor caracteristici:

- Direcția în care se deplasează șarpele: aceasta poate lua 4 valori(stanga, dreapta, sus, jos)
- Direcția mancarii raportata la poziția capului sarpelui: initial, valorile posibile au fost stanga, dreapta, sus, jos. Acest lucru a dus la un comportament gresit al agentului atunci cand mancarea nu se afla în linie dreapta cu capul sarpelui. Spre

exemplu, cand mancarea se afla in stanga sus fata de acesta, agentul trebuie sa aleagă dacă direcția mancarii este in stanga sau in sus și uneori va alege acțiuni ce îl vor indeparta de aceasta. Prin urmare, am ales ca valorile posibile pentru aceasta caracteristica sa fie stanga, dreapta, sus, jos, stanga-sus, stanga-jos, dreapta-sus, dreapta-jos.

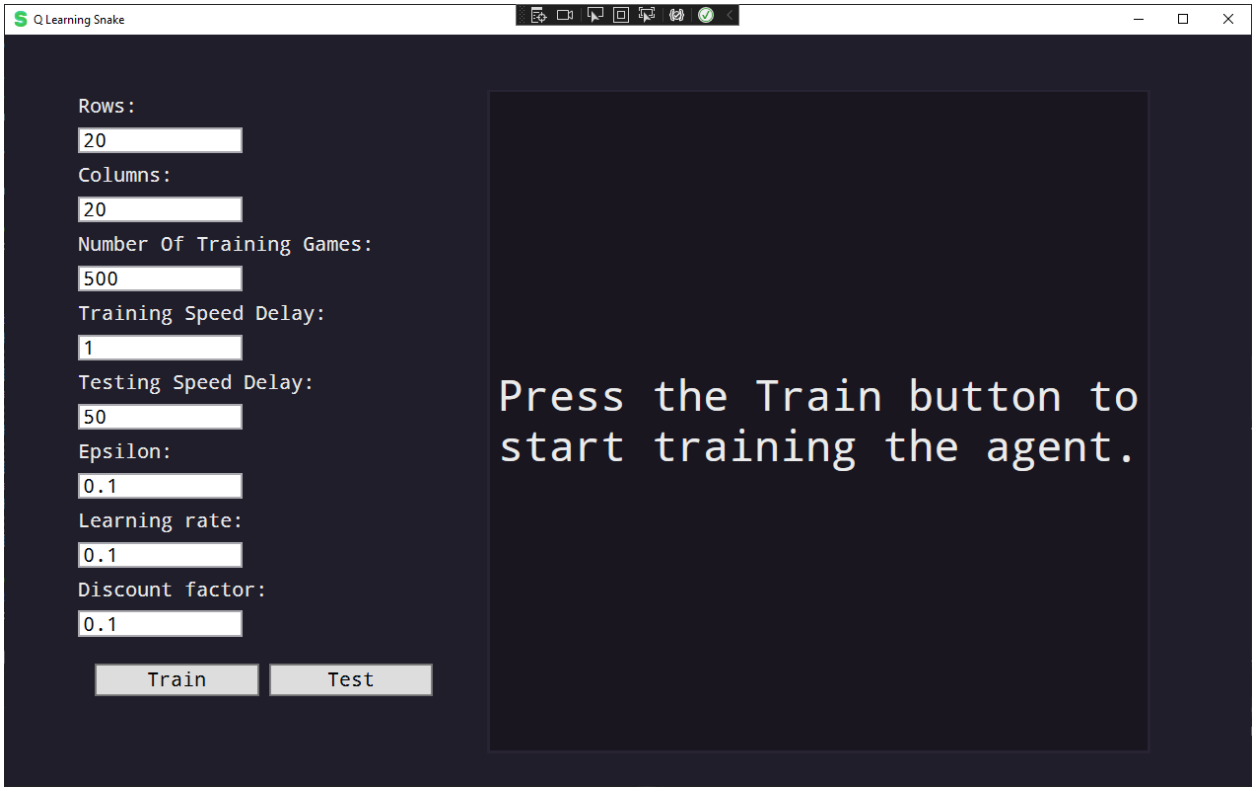
- Pericol in directiile stanga, dreapta, sus, jos: deoarece putem avea mai multe pericole simultan, vom avea 4 variabile ce pot lua valorile adevărat sau fals.

In total avem 4*8*2*2*2*2*2 = 512 stari posibile. Pentru fiecare stare în care se afla, agentul poatea alege intre 3 actiuni: sa isi pastreze directia, sa schimbe directia și să meargă în sus sau în jos(dacă direcția curentă de deplasare este stanga sau dreapta) sau sa schimbe directia și sa mearga in stanga sau in dreapta(dacă direcția curentă de deplasare este în sus sau în jos).

Recompensele stabilite pentru fiecare rezultat al acțiunilor alese de agent sunt următoarele:

- Dacă se apropie de mancare: +1
- Dacă se indeparteaza de mancare: -1
- Dacă ajunge la mancare: +10
- Dacă se lovește de un obstacol și pierde jocul: -1000

Din interfața putem selecta dimensiunea spațiului în care se poate mișca șarpele, numărul de jocuri pentru care dorim sa antrenam agentul, viteza de antrenare/testare, si parametrii algoritmului.



La apăsarea butonului Train, se inițializează tabela Q. Am ales un dicționar pentru a stoca valorile in care cheia o reprezinta combinatia caracteristicilor descrise anterior(adica starea), iar valoare este o lista în care indicii reprezinta actiunile și valorile reprezinta recompensele asociate actiunilor respective.

```
foreach (Direction headDirection in headDirections)
{
    foreach (Direction foodDirection in foodDirections)
    {
        foreach (bool dangerLeft in dangerLeftOptions)
        {
            foreach (bool dangerRight in dangerRightOptions)
            {
                foreach (bool dangerUp in dangerUpOptions)
                {
                    foreach (bool dangerDown in dangerDownOptions)
                    {
                        String key = $"{headDirection.GetHashCode()},{foodDirection.GetHashCode()},{dangerLeft},{dangerRight},{dangerUp},{dangerDown}";
                        qTable.Add(key, new double[3] { 0, 0, 0 });
                    }
                }
            }
        }
    }
}
```

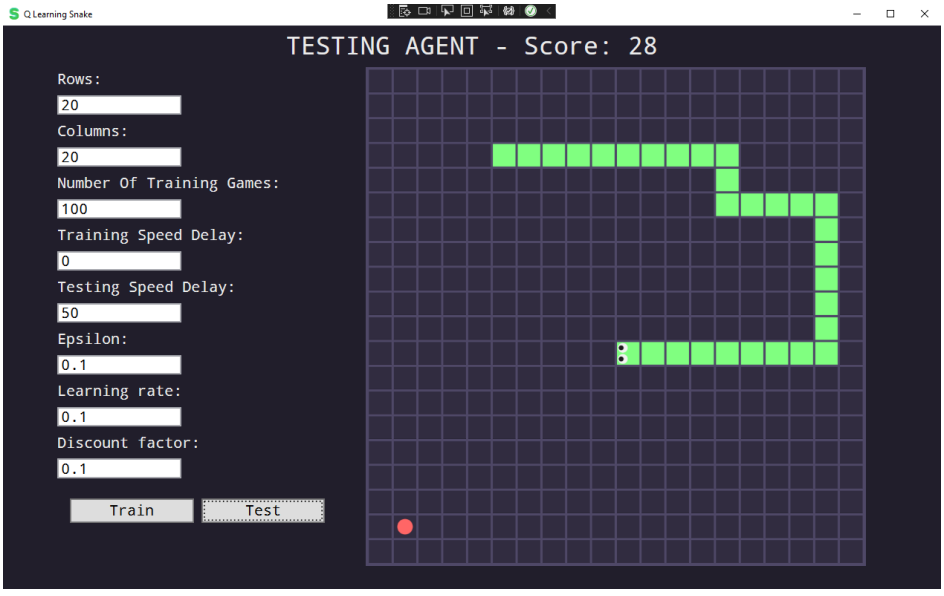
La fiecare pas, calculăm starea în care se afla agentul, iar la alegerea unei acțiuni din acea stare, dacă agentul se antrenează vom folosi tehnica *Epsilon-Greedy*. În majoritatea cazurilor, agentul va alege cele mai bune acțiuni în funcție de experiența sa, acest lucru poate duce la ignorarea unei acțiuni neexplorate care poate aduce o recompensa mai mare. Deci folosim un parametru ε ca agentul sa aleagă în ε% cazuri o acțiune aleatoare.

```
1 reference
public Direction GetNextMove(GameState gameState, bool isTraining)
{
    int action;
    double prob = random.NextDouble();
    String currentState = GetState(gameState);

    if (isTraining && prob <= epsilon)
    {
        action = GetRandomAction();
    }
    else
    {
        action = GetBestAction(currentState);
    }

    Direction direction = GetActionDirection(gameState, action);
    GameState nextGameState = gameState.DeepClone();

    nextGameState.ChangeDirection(direction);
    nextGameState.Move();

    UpdateQTable(gameState, action, nextGameState);

    return direction;
}
```

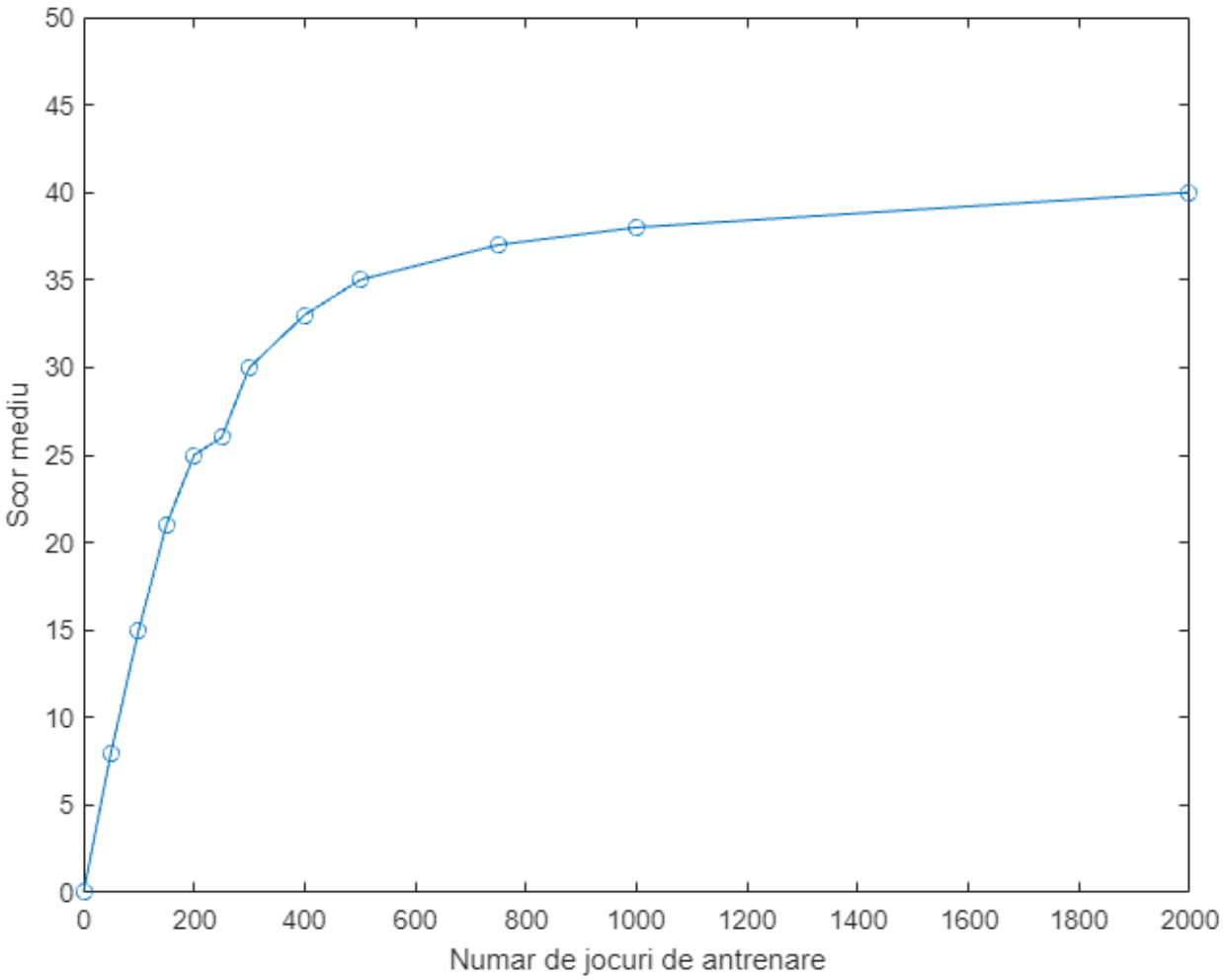
După ce agentul alege acțiunea, vom calcula noua stare în care acesta ajunge și vom actualiza tabela Q folosind *Ecuatia lui Bellman* și recompensele corespunzătoare.

```
private void UpdateQTable(GameState gameState, int action, GameState nextGameState)
{
    String oldState = GetState(gameState);
    String newState = GetState(nextGameState);
    int reward;

    if (nextGameState.GameOver)
    {
        reward = GAME_OVER_REWARD;

        // Bellman Equation - No Future State
        qTable[oldState][action] = qTable[oldState][action] + learningRate * (reward - qTable[oldState][action]);
    }
    else
    {
        if (nextGameState.HeadPosition().Equals(gameState.foodPosition))
        {
            reward = EAT_FOOD_REWARD;
        }
        else if (nextGameState.RowDistanceFromFood() < gameState.RowDistanceFromFood() ||
            nextGameState.ColDistanceFromFood() < gameState.ColDistanceFromFood())
        {
            reward = MOVE_CLOSER_TO_FOOD_REWARD;
        }
        else
        {
            reward = MOVE_AWAY_FROM_FOOD_REWARD;
        }

        // Bellman Equation
        int nextStateBestAction = GetBestAction(newState);
        qTable[oldState][action] = qTable[oldState][action] + learningRate *
            (reward + discountFactor * qTable[newState][nextStateBestAction] - qTable[oldState][action]);
    }
}
```

La finalul perioadei de antrenare, folosind butonul Test putem testa agentul, în aceasta faza acesta se va baza numai pe cunoștințele dobandite în faza de antrenare.



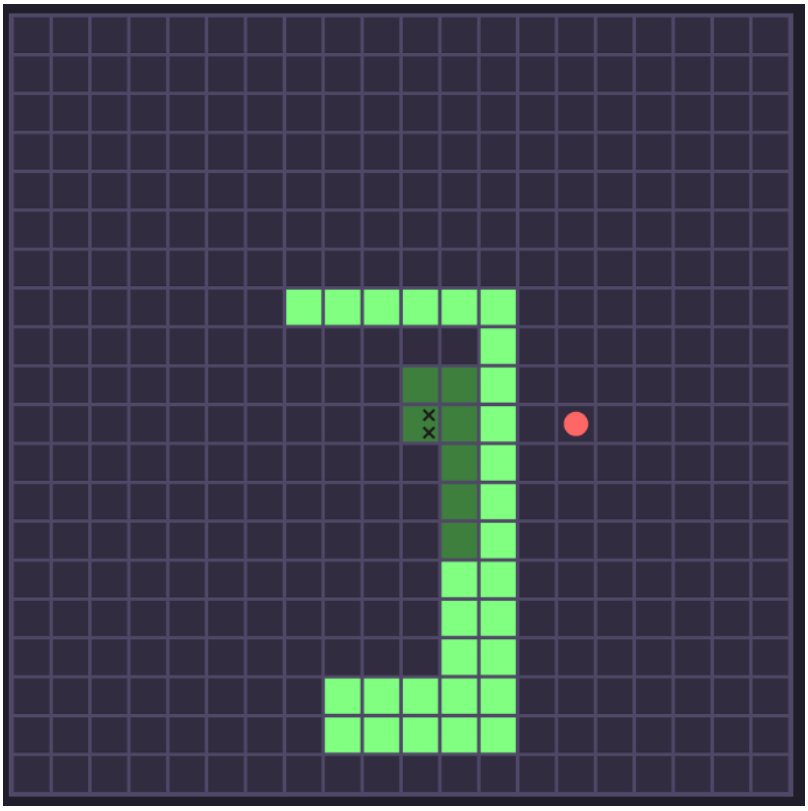
5. Rezultate obtinute



Observam ca scorul mediu pe care reușește să îl obțină agentul nostru crește proporțional cu numărul de jocuri de antrenare și atinge un platou după aproximativ 500-600 de jocuri. După acest prag e nevoie de un număr de jocuri mai mare de 10.000 (timpul necesar antrenarii fiind de peste 3 ore) pentru a sesiza o creștere considerabilă a scorului.

6. Concluzii

In implementarea propusa de noi, agentul nu va juca niciodată un joc perfect. Acesta reușește sa se apropie de mâncare și să evite pericolele imediate, însă nu este suficient de inteligent pentru a evita sa se blocheze în propriul corp.



În scenariul de mai sus, agentul ar trebui sa aleaga direcția stanga ca următoarea mișcare a sarpelui, pentru a-si evita corpul, însă alege sa nu se indeparteze de mancare, blocandu-se astfel. Ar trebui modificat spațiul stărilor astfel incat sa includem și acest caz pentru ca agentul sa poată învăța să îl evite.

7. Contributii

Implementare joc Snake	Eduard Pușcașu
Implementare interfata	Daniel Rotariu
Stabilire stari, recompense, parametri	Daniel Rotariu, Eduard Pușcașu
Implementare algoritm Q-Learning	Daniel Rotariu
Testare agent in diverse situatii	Eduard Pușcașu
Documentatie - Sectiunile 1, 2, 3	Daniel Rotariu
Documentatie - Sectiunile 4, 5, 6	Eduard Pușcașu

8. Bibliografie

- https://en.wikipedia.org/wiki/Q-learning
- https://www.slideshare.net/FlorinLeon/invatarea-cu-intarire
- https://towardsdatascience.com/q-learning-algorithm-from-explanation-to-implementation-cdbeda2ea187
- https://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html
- http://cs229.stanford.edu/proj2016spr/report/060.pdf
- https://openreview.net/pdf?id=iu2XOJ45cxo
- https://towardsdatascience.com/teaching-a-computer-how-to-play-snake-with-q-learning-93d0a316ddc0