

Învățare Automată - Tema 1

Învățare prin recompensă: Cărămizi

Tudor Berariu

Laboratorul AIMAS

Facultatea de Automatică și Calculatoare

17 martie 2014

1 Pe scurt...

Se cere să se implementeze algoritmul **SARSA** pentru învățarea unei strategii pentru o versiune a jocului Tetris. Se va testa eficiența algoritmului pentru diferite niveluri de complexitate a jocului.

2 Motivația temei

Scopul acestei teme îl reprezintă familiarizarea cu un algoritm de învățare automată, SARSA, precum și implementarea acestuia. Se urmărește înțelegerea diferențelor dintre acesta și algoritmul Q-Learning, precum și aplicarea SARSA pe o problemă în care numărul stărilor este foarte mare.

3 Descrierea jocului

Jocul *Cărămizi* este o variantă a jocului Tetris concepută pentru studenții cursului de Învățare Automată cu o listă de modificări descrisă în cele ce urmează.

Se consideră o fântână reprezentată ca un spațiu de lățime L și înălțime H și 7 tipuri de cărămizi, enumerate în Tabela 1.




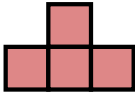
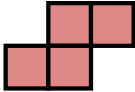
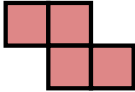

A	
B	
C	
D	
E	
F	
G	

Tabela 1: Tipurile de cărămizi

3.1 Jucătorii și obiectivele lor

Cărămizi opune doi jucători:

Meșterul este jucătorul din Tetris clasic, cel care așează piesele primite în fântână. Scopul lui este să completeze linii și să obțină un punctaj cât mai mare.

Cărămidarul este jucătorul ce stabilește distribuția de probabilitate cu care va fi generată secvența de cărămizi.

Se vor așeza 18 cărămizi după cum urmează. Cărămidarul stabilește distribuția de probabilitate, o nouă cărămidă este generată cu ajutorul aces-

teia, iar Meșterul o așează în fântână.

Meșterul va așeza cărămidzile primite în fântână unele peste altele. O cărămidă va putea fi așezată doar deasupra celor fixate deja. Spre deosebire de Tetris clasic, aici cărămida va fi întâi rotită și așezată deasupra poziției dorite și abia apoi lăsată să cadă fără a mai fi deplasată sau rotită până ajunge jos (vezi Figura 1).

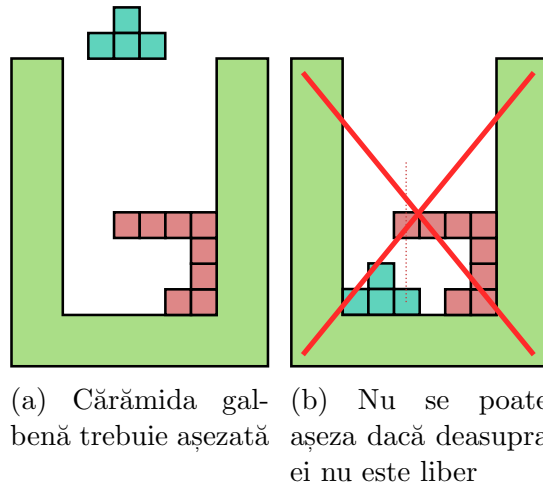


Figura 1: Exemplu de așezare incorectă a unei cărămizi

Jocul se încheie atunci când au fost așezate 18 cărămizi sau construcția depășește înălțimea H .

3.2 Punctarea

Atunci când în urma așezării unei cărămizi se completează una sau mai multe linii, acestea vor dispărea (*magie!*) și Meșterul va primi:

- 1 punct pentru o linie completată,
- 3 puncte pentru 2 linii completate,
- 9 puncte pentru 3 linii completate,
- 27 puncte pentru 4 linii completate.

Liniiile completate nu trebuie să fie consecutive (vezi exemplul din Figura 2).

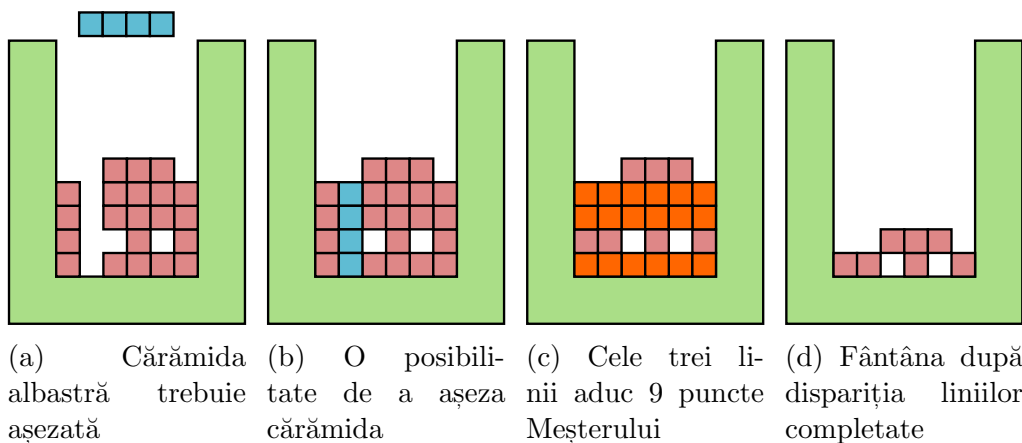


Figura 2: Exemplu de completare a liniilor

Dacă jocul se încheie prin depășirea înălțimii fântânii, atunci Meșterului i se vor retrage 20 de puncte.

De fiecare dată când Meșterul primește un număr de puncte în urma completării unor linii, Căramidarului îi sunt retrase la fel de multe puncte. În mod similar, dacă Meșterul va construi peste înălțimea H , Căramidarul va primi 20 de puncte.

Toate regulile de punctare sunt trecute în Tabela 2.

Eveniment	Puncte Meșter	Puncte Căramidar
1 linie completată	1	-1
2 linii completate	3	-3
3 linii completate	9	-9
4 linii completate	27	27
Depășirea înălțimii fântânii	-20	20

Tabela 2: Punctarea

4 Algoritmul SARSA

Algoritmul SARSA (State-Action-Reward-State-Action) [RN94, SS96] este similar algoritmului Q -learning (făcut la curs și la laborator).

Algoritmul 1 Algoritmul SARSA

```
1: pentru toate stările  $s$  execută
2:   pentru toate acțiunile  $a$  execută
3:      $Q(s,a) \leftarrow 0$ 
4:   termină ciclu
5: termină ciclu
6: pentru toate episoadele execută
7:    $s \leftarrow$  stare inițială
8:    $a \leftarrow \epsilon$ -greedy( $Q, s$ )
9:   repetă
10:    execută  $a$ 
11:    observă  $s', r$ 
12:     $a' \leftarrow \epsilon$ -greedy( $Q, s'$ )
13:     $Q(s, a) \leftarrow Q(s, a) + \alpha \left( r + \gamma Q(s', a') - Q(s, a) \right)$ 
14:     $s \leftarrow s'$ 
15:     $a \leftarrow a'$ 
16:   până când  $s$  este stare finală
17: termină ciclu
```

Regula de actualizare a funcției Q este diferită față de cea a algoritmului Q -Learning (vezi formula 1) prin faptul că pentru *recompensele viitoare* nu se alege întotdeauna valoarea maximă pentru $Q(s', *)$, ci se alege o acțiune a' conform politicii (care include și componenta de explorare) și se utilizează valoarea $Q(s', a')$. Din puncte de vedere conceptual, această diferență se traduce prin faptul că SARSA este un algoritm *on-policy* spre deosebire de Q -Learning care este un algoritm *off-policy*.

Formula de actualizare a funcției Q în algoritmul SARSA:

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha_t \left(R(s_t, a_t) + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \right) \quad (1)$$

În cazul unui proces de învățare continuu este de așteptat ca algoritmi on-policy să se comporte mai bine decât algoritmi off-policy întrucât politica învățată ține cont și de componenta de explorare. Acest lucru este util mai ales atunci când mediul este unul dinamic și nu se poate renunța la explorare.

Avantajul algoritmului SARSA față de Q -Learning este foarte bine surprinsă în următorul paragraf din [BI99]: [...] *perhaps it would be better to learn the policy that is optimal, given that you will explore ϵ of the time.*

It would be like a person who, when walking, always takes a random step every 100 paces or so. Such a person would avoid walking along the top of a cliff, even when that is the optimal policy for a person who doesn't explore randomly.

Pseudocodul SARSA este prezentat în Algoritmul 1. Pentru alegerea acțiunii dintr-o stare s folosiți tehnica ϵ -greedy, descrisă de Algoritmul 2.

Algoritmul 2 Algoritmul ϵ -greedy

Intrări: Q , starea s, ϵ, A

Ieșire: acțiunea a

- 1: $A' \leftarrow \{a \in A \mid a \text{ acțiune validă în } s\}$
 - 2: $p \leftarrow \text{random}(0, 1)$
 - 3: **dacă** $p < \epsilon$ **atunci**
 - 4: $a \leftarrow \text{random}(A')$
 - 5: **altfel**
 - 6: $a \leftarrow \underset{act \in A'}{\operatorname{argmax}} Q(s, act)$
 - 7: **termină dacă**
-

5 Cerințe

[7 puncte] Implementarea algoritmului SARSA

Să se implementeze algoritmul SARSA pentru a determina o politică (o strategie de joc) cât mai bună pentru Meșter în jocul *Cărămizi*. Se cere o optimizare a reprezentării stărilor. Implementarea se poate face de la zero sau se pot folosi ca punct de plecare serverul de joc și scripturile din arhivă (vezi Anexa A).

Se vor acorda 5 puncte pentru implementarea algoritmului și 2 puncte pentru optimizarea reprezentării stărilor.

[3 puncte] Evaluarea eficienței algoritmului

Se va evalua metoda de învățare propusă prin observarea scorului obținut de Meșter de-a lungul unui număr mare de jocuri (Meșterul își păstrează utilitățile de la un joc la altul).

Se va realiza un grafic cu evoluția scorului obținut de meșter, eventual netezit.

Se va testa algoritmul pentru cele trei cazuri descrise în Tabela 3.



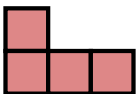
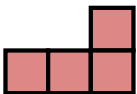

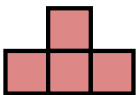
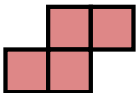
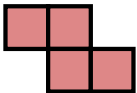
Nr	Fântâna		Căramidarul		
	H	L	Distribuția fixă?	Distribuția de probabilitate	
				Căramida	Probabilitatea de apariție
1	4	4	Da		1
2	8	5	Da		0.2
					0.4
					0.4
3	8	6	Da		0.16667
					0.66665
					0.08334
					0.08334

Tabela 3: Scenariile de test

[2 puncte] BONUS

Aplicați un algoritm de învățare automată (eventual tot învățare prin recompensă, chiar și SARSA) pentru strategia Căramidarului. Acesta va putea modifica înaintea fiecărei mutări distribuția de probabilitate cu care se generează cărămizile.

6 Trimiterea temei

În arhiva temei includeți:

- toate fișierele sursă (eventual cu Makefile / script de compilare și rulare)
- fișier pdf cu descrierea detaliilor de implementare: cum sunt reprezentate stările, care sunt acțiunile, care este sistemul de recompensare, care este valoarea lui ϵ (politica), numărul de jocuri în care Meșterul a învățat, grafic (sau grafice dacă ați încercat mai multe variante) cu evoluția scorului, etc.

A Arhiva

În arhiva ce însoțește acest document se găsește implementarea unui server de joc pentru *Căramizi*. Pentru a folosi acest server de joc la testarea temei trebuie să comunicați prin socketi TCP respectând protocolul descris în Figura 3 și *povestit* în Secțiunea A.1. Dacă binarele nu pot fi folosite, sursele se găsesc la adresa <https://github.com/tudor-berariu/bricks-ml-assignment>.

A.1 Protocolul de comunicație

Toate mesajele schimbate de cele trei entități (serverul de joc, căramidarul și meșterul) se încheie cu `\n`.

Serverul va controla evoluția jocului și va cere pe rând fiecărui jucător să mute (Căramidarul să ofere o piesă, iar Meșterul să o poziționeze).

Întâi cei doi jucători trebuie să se conecteze la server și să trimită un mesaj cu numele lor (`BRICKLAYER\n`, respectiv `BRICKMAKER\n`). Drept răspuns vor primi dimensiunea fântânii (de exemplu un mesaj `10,6\n`).

Începând din acest moment, pentru fiecare joc:

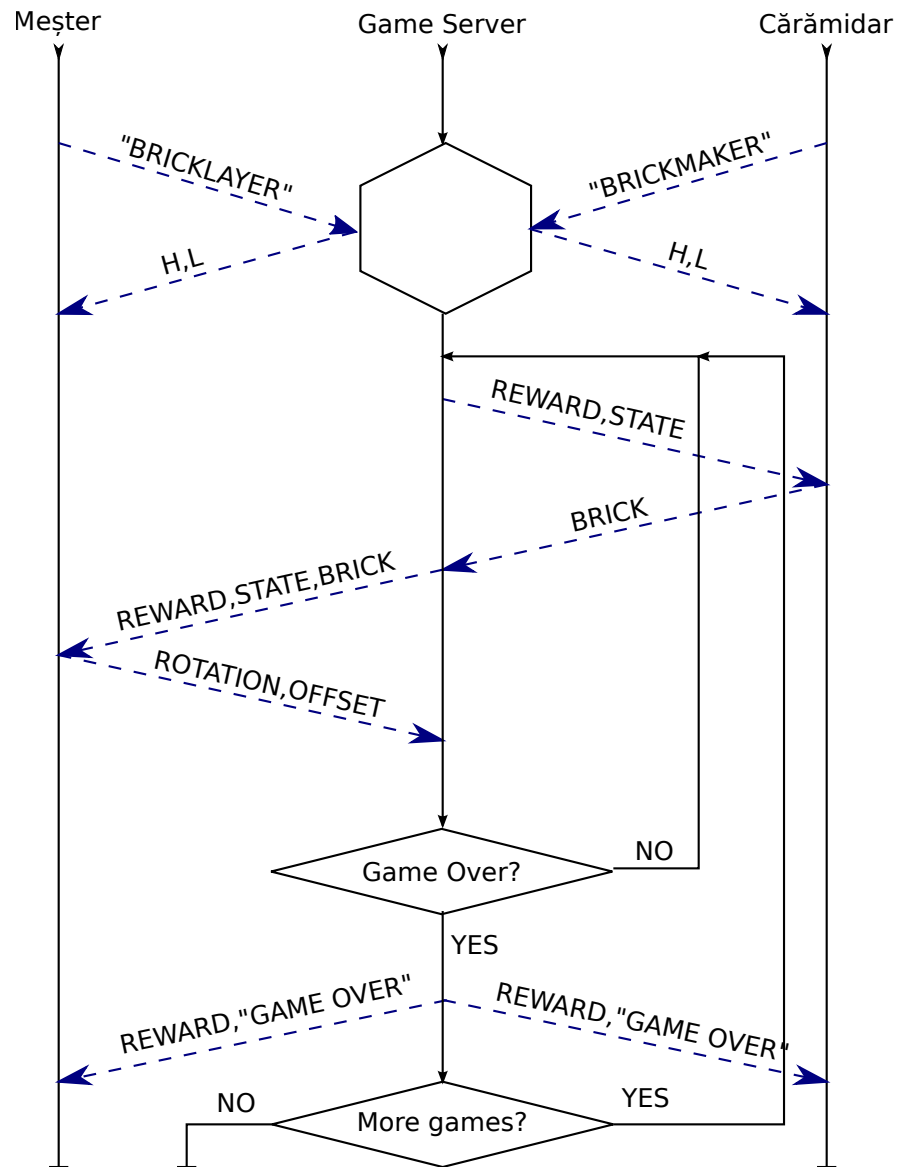


Figura 3: Protocolul de comunicație

1. serverul îi trimite căramidarului un mesaj:

`reward,state\n`

unde `reward` reprezintă numărul de puncte obținute în urma mutării

anterioare, iar **state** noua stare a jocului. De exemplu:

```
-9,|    |    |    #|  ##|## #|\n
```

2. cărămidarul răspunde cu un mesaj corespunzător literei următoarei cărămzi (de exemplu: **C\n**)
3. serverul îi trimite Meșterului un mesaj ce conține numărul de puncte obținute pentru mutarea anterioară, starea jocului, dar și piesa ce trebuie poziționată. Un exemplu de mesaj este:

```
9,|    |    |    #|  ##|## #|,C\n
```

4. Cărămidarul îi trimite serverului două numere: numărul de rotații în sensul acelor de ceasornic ce trebuie executate și numărul de celule distanță de la peretele din stânga. Un exemplu de mesaj este:

```
3,2\n
```

pentru o piesă rotită cu 270 de grade și care va fi plasată la distanță de 2 căsuțe de peretele din stânga (începând cu a treia coloană).

5. Dacă jocul s-a încheiat (s-au pus 18 piese sau s-a depășit înălțimea fântânii), serverul trimite fiecărui jucător un mesaj cu ultimele puncte câștigate și textul **GAME OVER**. De exemplu:

```
27,GAME OVER\n
```

A.2 Utilizarea fișierelor din arhivă

Se lansează serverul de joc astfel:

```
./bricks_server
```

sau, pentru o configurare mai precisă a parametrilor:

```
./bricks_server --verbose --height=4 --length=4 --games-no=200
```

Pentru a vedea toți parametrii ce pot fi configurați se execută:

```
./bricks_server --help
```

Un Cărmidar care citește distribuția de probabilitate dintr-un fișier dat este deja implementat. Se poate lansa astfel (alegând desigur o valoare corespunzătoare pentru port):

```
./brickmaker --port=9999 --in-file distributions/dist1
```

Pentru alte detalii, rulați:

```
./brickmaker --help
```

Un Meșter care așează piesele aleator este implementat în `bricklayer.py`. Pentru rezolvarea temei se poate pleca de la codul din acest fișier.

Pentru a evalua acest jucător, se rulează `run.sh`. Se vor crea trei fișiere: `scores1`, `scores2`, `scores3`, corespunzătoare celor 3 scenarii cerute în enunț.

Dacă binarele nu funcționează, se pot descărca sursele de la <https://github.com/tudor-berariu/bricks-ml-assignment>.

Bibliografie

- [BI99] L.C. Baird III. *Reinforcement learning through gradient descent*. PhD thesis, Citeseer, 1999.
- [RN94] G.A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering, 1994.
- [SS96] S.P. Singh and R.S. Sutton. Reinforcement learning with replacing eligibility traces. *Recent Advances in Reinforcement Learning*, pages 123–158, 1996.