Tema 1

Invatare prin recompensa: Caramizi

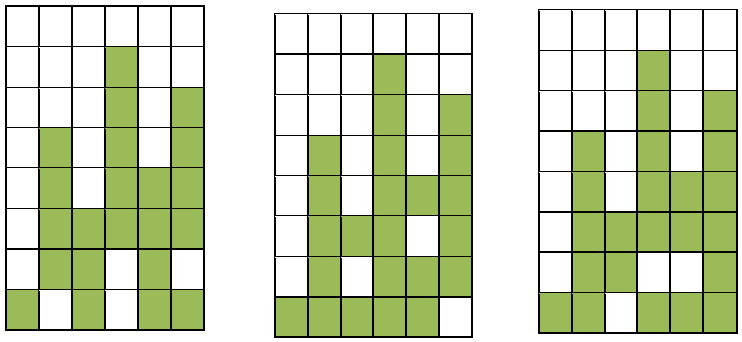
Limbaj de programare: Python

Detalii de implementare:

Datorita faptului ca positionarea unei piese se realizeaza prin setarea **rotatiei** si a **offsetului** fata de marginea din stanga fara a mai putea modifica acesti parametri pe parcurul caderii piesei se poate deduce simplu ca un mod eficient de a retine starea pieselor este de a mentine **inaltimea fiecarei coloane.**

Astfel starea **153746** ar reprezenta inaltimile de pe fiecare coloana in ordine de la stanga la dreapta.

Inaltimea nu reprezinta **numarul de blocuri ocupate** pe aceea coloana ci **inaltimea maxima ocupata de un bloc**.



Toate starile de sunt reprezintate de acceasi stare **153746** intrucat pentru a putea ajunge intr-o stare mai buna **este necesar** rezolvarea **starii actuale** doar **prin asezarea blocurilor de sus in jos conform cerintei,** deci **spatiile libere** interioare nu conteaza intr-o astfel de stare de joc

Starea de joc este codificata pe biti, cu cate 4 biti pentru fiecare coloana astfel:

Stare : 153746

Stare: 0110 0100 0111 0011 0101 0001
Stare: 6 4 7 3 5 1
Se retin valorile pe biti in oglinda. Codificare functioneaza pentru o inaltime maxima a jocului de 15 coloane. Testele au fost realizate doar pe specificatiile hartii de **(4, 4) (8, 5) si (8, 6)**
O actiune e reprezentata de grupul **(piesa, rotatie, offset)**

Codificarea este urmatoarea:

7 piese => 3 biti
4 rotatii posibile => 2 biti
Offset maxim 5 (in teste) => 3 biti

Intrucat pentru teste sunt maxim **6 coloane, deci 24 biti** am utilizat un dictionar pentru a retine fiecare (stare, actiune) prin intermediul unui **INTEGER pe 32 biti** codificat astfel:

Stare: 153746 – B – 270 grade – 4 pozitii
Stare: (0110 0100 0111 0011 0101 0001) (010) (11) (100)
= 0110 0100 0111 0011 0101 0001 0101 1100
= INTEGER 32

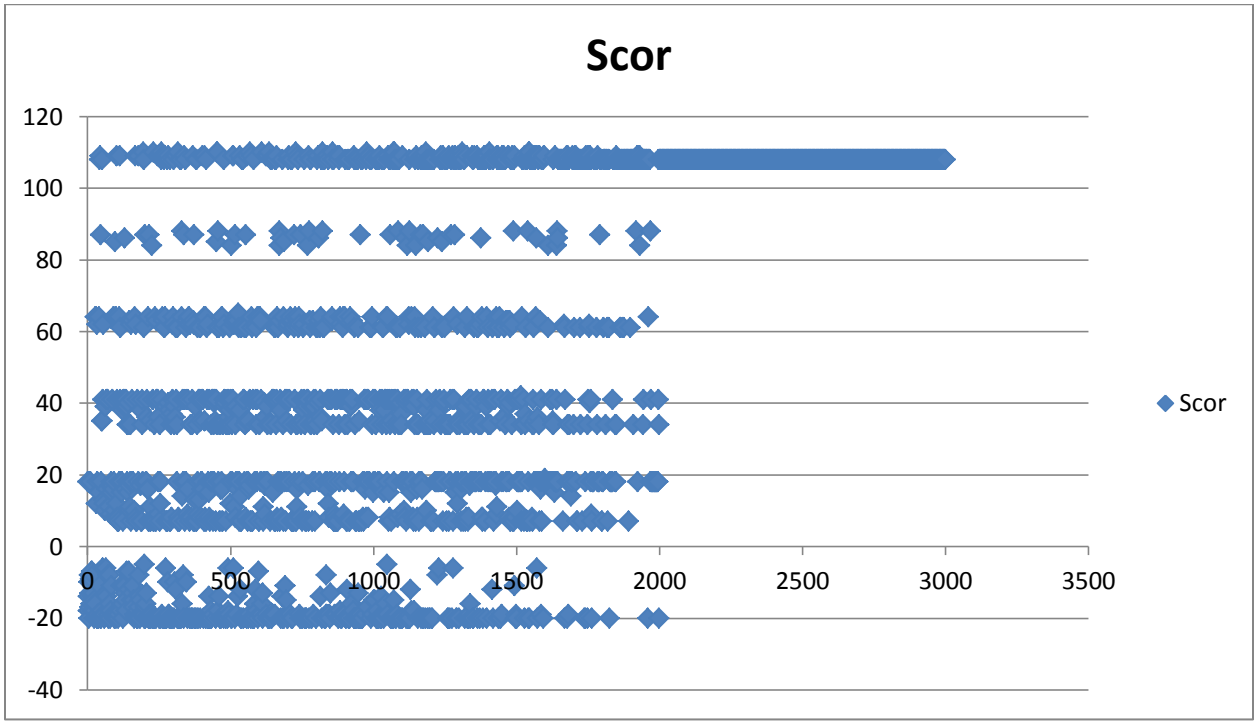
Astfel in dictionar starea **(153746 – B – 270 grade – 4 pozitii)** este retinuta printr-un INTEGER 32

Teste

I. Fantana 4 – 4 (H - L)

Fisier distributie: **dist1**
ALPHA = 0.5
GAMMA = 0.5
EPSILON = 0.2

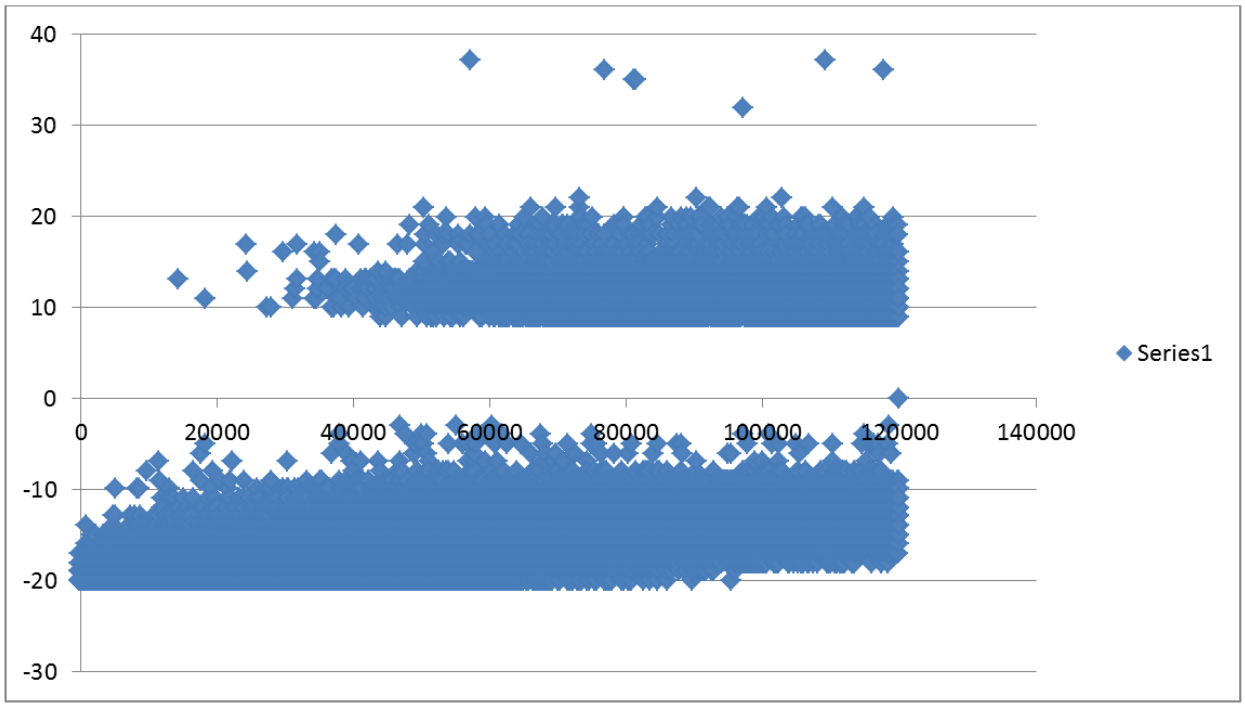
EPSILON scade la **fiecare 100 jocuri cu 0.01 pana la 0.001 minim**
Dupa aproape 2000 de jocuri se obtine un rezultat aproape constant de **108 puncte,** cateva de **109** si foarte rar **110** maxim posibil pentru acest joc.



II. Fantana 8 – 5 (H - L)

Fisier distributie: **dist2**
ALPHA = 0.5
GAMMA = 0.5
EPSILON = 0.7

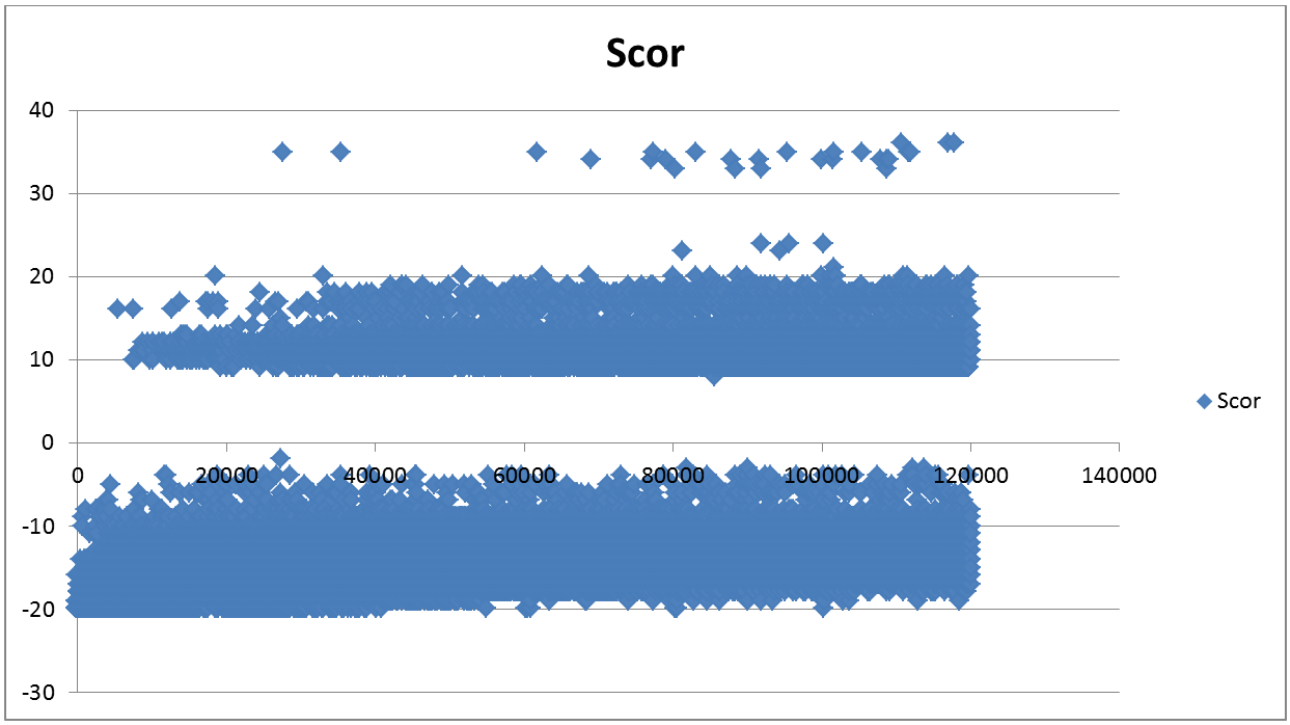
Politica: la fiecare **1000 episoade** EPSILON scade cu 0.01
EPSILON_MINIM = 0.0001



ALPHA = 0.5
GAMMA = 0.5
EPSILON = 0.1
MIN_EPSILON = 0.0001

Politica: la fiecare **1000 episoade** EPSILON scade cu 0.01
La episoadele 20000 si 40000 EPSILON a fost crescut din nou la 0.1 (cu aceasi scadere de pas de 0.01 / 1000 Episoade)

Dimensiune Q: 350.000 – grupuri (s, a)

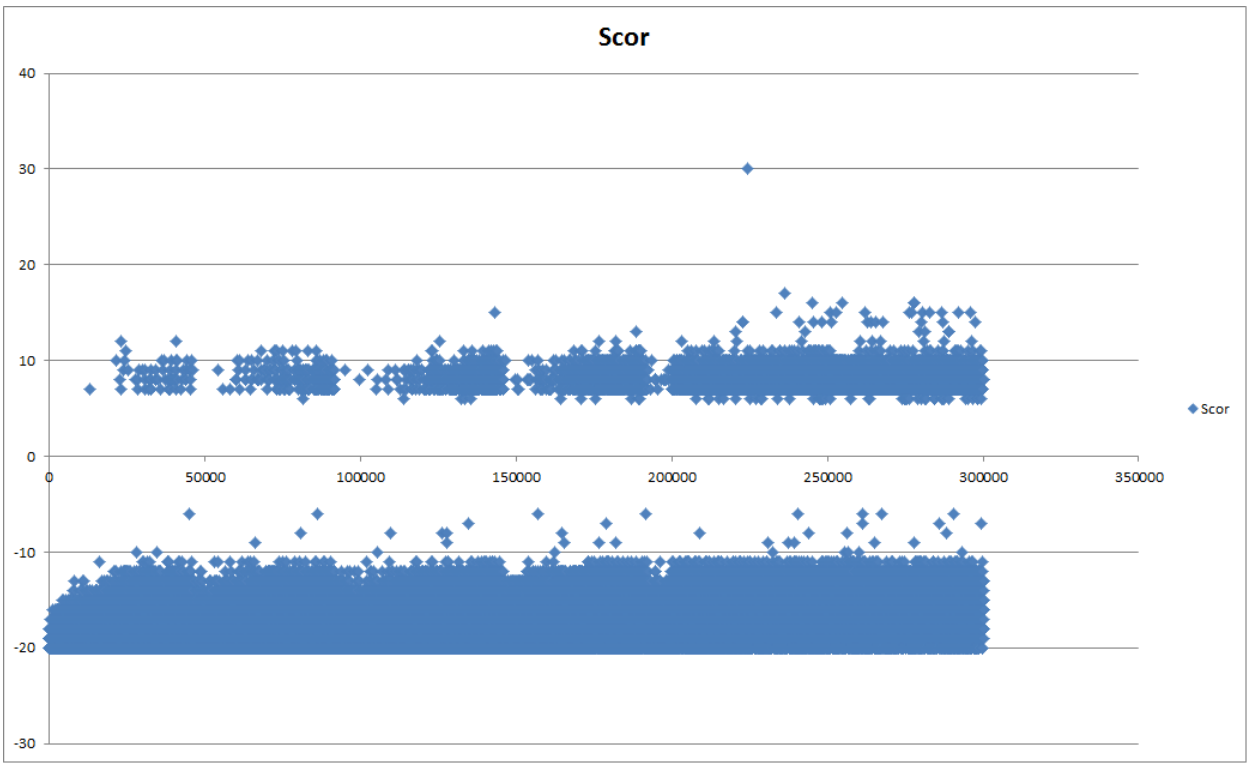


I. Fantana 8 – 6 (H - L)

Fisier distributie: **dist3**

ALPHA = 0.7
GAMMA = 0.4
EPSILON = 0.3
MIN_EPSILON = 0.0001

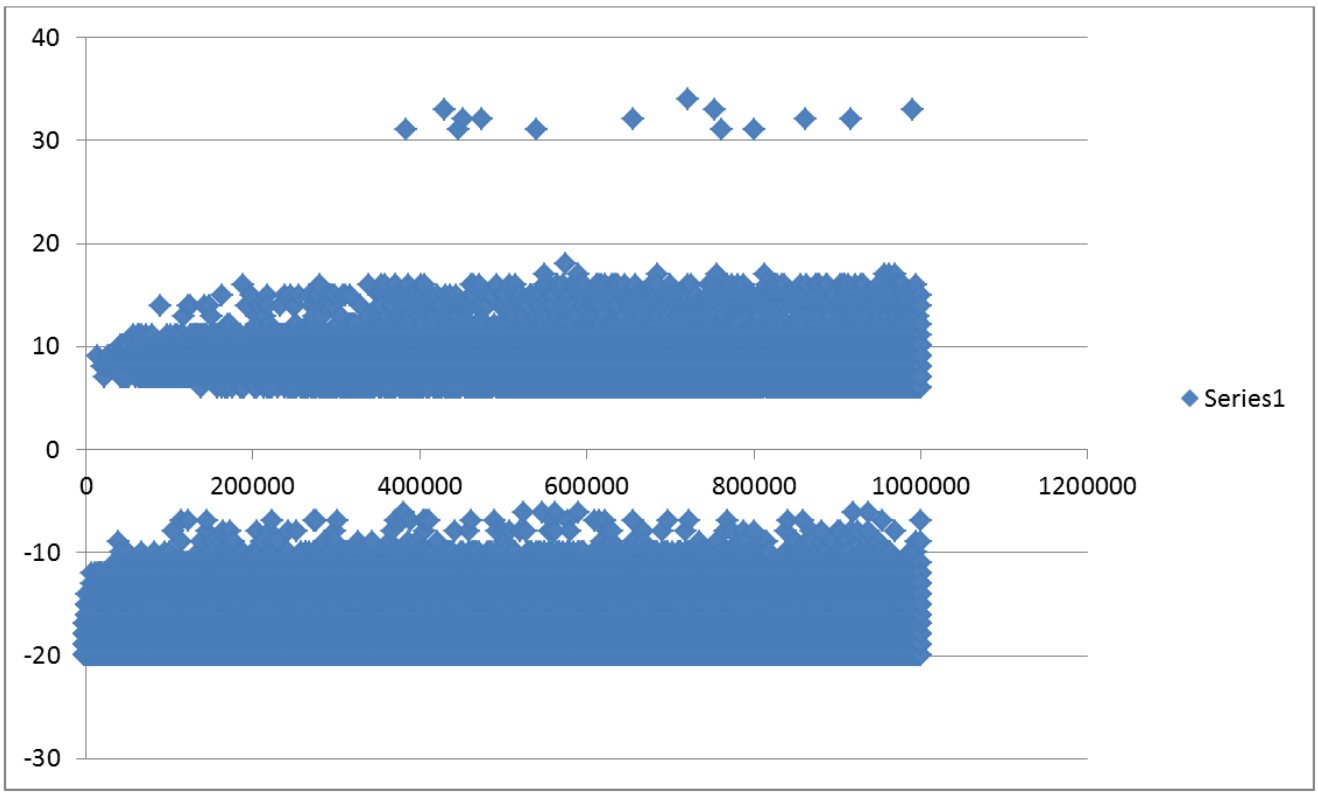
Politica: la fiecare **1000 episoade** EPSILON scade cu 0.01.
De la 0.02 EPSILON scade cu 0.001 pana.



Start episode	Interval Episod	Jocuri castigate (scor > 0)	Procentaj castig
0	100.000	386	0%
100.000	200.000	3446	3.44%
200.000	300.000	11846	11.84%
300.000	400.000	18033	18.03%
400.000	500.000	23164	23.16%
500.000	600.000	27445	27.45%
600.000	800.000	65234	32.16%
800.000	1.000.000	75108	37.50%

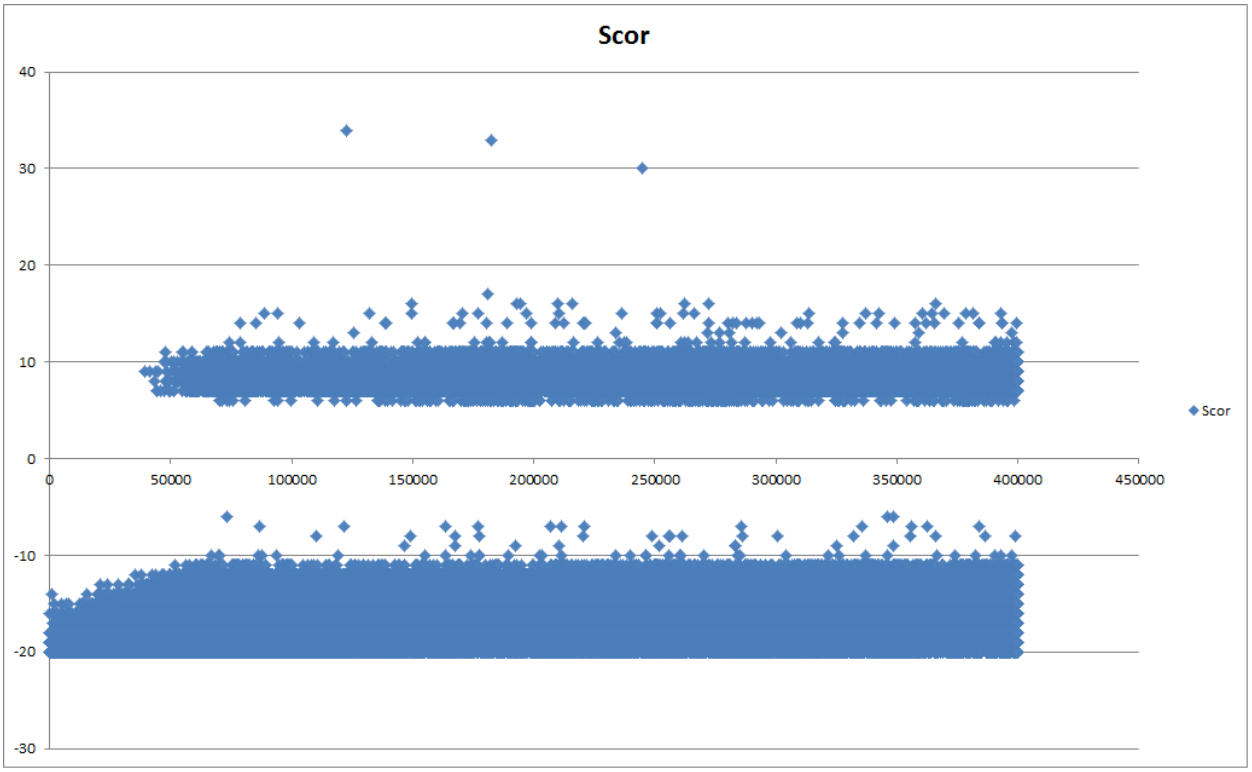
ALPHA = 0.5
GAMMA = 0.5
EPSILON = 0.1
MIN_EPSILON = 0.0001
QSIZE: 1731618

Politica: la fiecare **1000 episoade** EPSILON scade cu 0.01
De la 0.02 EPSILON scade cu 0.001 pana. La 100K si 200K EPSILON se reseteaza la 0.1 urmand aceasi politica. Incepand cu 200K EPSILON e constant 0.0001
Se poate observa faptul convergenta foarte rapida pentru aceasta politica si procentajul de castig foarte mare comparativ cu alte rezultate unde se poate obtine o medie de 2-3%.



ALPHA = 0.4
GAMMA = 0.6
EPSILON = 0.6
MIN_EPSILON = 0.0001

Politica: la fiecare **1000 episoade** EPSILON scade cu 0.01.



Implementare Bonus

Pentru a implementa bonusul am folosit aceeasi reprezentare de stare de joc ca si la Mester la care am adaugat caramida reprezentata pe biti (7 tipuri de caramizi = 3 biti)

Stare: 153746 – caramida B
Stare: (0110 0100 0111 0011 0101 0001) (010)
= 0011 0010 0011 1001 1010 1000 1010
= INTEGER 32

Am aplicat fix acelasi algoritm SARSA folosit si in cazul Mesterului

Punctajul Mesterului si al Caramidarului sunt mereu opuse. Pentru a testa am considerat pragul de 0 puncte pragul intre Castig/Pierdere. Astfel daca Mesterul castiga, Caramidarul pierde, si invers.

Intrucat dupa un numar X destul de mare Mesterul o sa invete sa joace correct indifferent de actiunile Caramidarui, am facut comparatie intre Caramidarul ofserit in arhiva si cel scris de mine. In ambele cazuri Mesterul a fost rulat cu aceeasi parametric ALFA, GAMA, EPSILON

Fantana	Episoade	WINS Meșter (Caramidar Arhiva)	WINS Meșter (Caramidar SARSA)	Parametrii Caramidar (alpha gama, eps)
8 - 5	100.000	12581 (12.58%)	49 (0.05%)	(0.9, 0.5, 0.5)
8 - 5	100.000	13075 (13.07%)	9 (0.01%)	(0.2, 0.7, 0.1)
8 – 6	200.000	2533 (1.30%)	16 (0.01%)	(0.3, 0.6, 0.4)
8 – 6	200.000	2755 (1.38%)	4 (0.01%)	(0.5, 0.7, 0.1)

Fantana 8 – 5 : distributie/dist2
Fantana 8 – 6 : distributie/dist3

Se poate observa ca aplicarea algoritmului SARSA asupra caramidarului ingreuneaza extreme de mult munca Mesterului fapt ce determina o convergenta mult mult mai tarzie a algoritmului de invatare al Mesterului. Ploturile grafice nu au foarte mult sens pentru acest test intrucat doar se modifica timpul (deci si numarul de episoade) necesar convergentei algoritmului.

Informatii aditionale

Testele au fost rulate cu versiunile de testing modificate pentru a rula pe UNIX Sockets.
Am adaugat binarele noi compilate pentru testare in arhiva. Nu necesita specificarea unui port.
Binarele le am de la Gilca Mircea 342C5 (el a venit cu idea de a folosi Unix Sockets). Sporul de performanta adus de UNIX Sockets este de peste 10 ori, si mi-a permis astfel sa pot face mai multe teste.

Makefile:

Numar de jocuri: GAMES=10000

make run44 GAMES=3000	fantana 4 - 4
make run85 GAMES=100000	fantana 8 - 5
make run86 GAMES=200000	fantana 8 – 6
make layer	Python BrickLayer
make maker DIST=0	Python BrickMaker, caramizi [‘A’, ‘B’, ‘C’]
DIST = [[‘A’, ‘B’, ‘C’], [‘A’, ‘D’, ‘E’, ‘F’]]	Pentru fantana 8-5 DIST=0
	Pentru fantana 8-6 DIST=1
make server GAMES=100 H=8 L=6	ruleaza serverul pe 100 de episoade si fantana 8 - 6