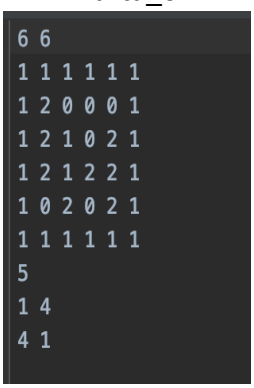
Cojocariu Sebastian
343C1

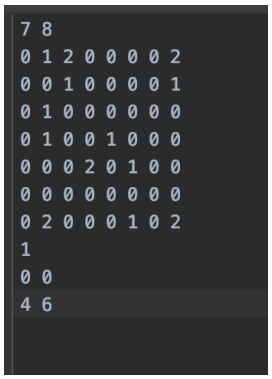
Tema 2
ML

Grafice

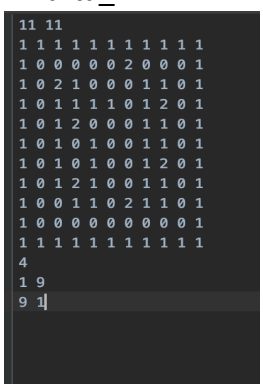
Harta_0



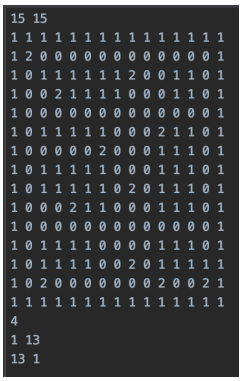
Harta_1



Harta_2



Harta_3



Detalii de implementare:
Avem **2 clase** pentru Qlearning (Maze si Trainer), respectiv 2 clase pentru Sarsa (Maze si Trainer_Sarsa). Datele de intrare sunt citite dintr-un fisier .json (configs_qlearning.json, respectiv configs_sarsa.json). Aici se poate seta:

- **numele hartii**
- **tipul de algoritm** pentru calcularea actiunii viitoare
- **detaliile referitoare la grafica jocului**(daca se afiseaza harta in timpul antrenarii, daca la sfarsitul antrenarii se arata un demo demonstrativ cu agentul antrenat, timpul de sleep intre afisarea unui pas)
- **rewardurile** pentru posibile situatii
- **training_config**: numarul episoadelor de antrenare (daca se alege modul final_show se pot da si de la tastatura alte episoade de antrenare)
- **eval_config:** frecventa (raportata la numarul de episoade de antrenare) la care se face validarea (practic la fiecare NR_TRAIN_EPISOADE/ frecventa se verifica politica). “eval_every” e util cand nu este specificata frecventa, “number_of_simulated_games” reprezinta numarul de jocuri cu care se valideaza politica, iar “eval_final_wins” este validarea finala (se pot da mai multe episoade de testare pentru a obtine o precizie mai buna)
- **game_hyperparameters:** valorile initiale ale lr, epsilon, discount, din cate in cate episoade se modifica fiecare, cu ce theta se modifica si ce tip de functie foloseste in modificare acestora.

Cojocariu Sebastian
343C1

```
{
    "map_file_name":"harta_3",

    "policy_type":"explorare_exploatare",

    "game_graphics": {
                    "final_show":"True",
                    "verbose":"False",
                    "sleep":0.1,
                    "plot":"True"
                    },

    "game_rewards":{
                    "move_reward":-0.1,
                    "win_reward":80,
                    "lose_reward":-100,
                    "cheese_reward":10,
                    "min_threshold":-20
                    },

    "training_config":{
                    "training_episodes":10000
                    "learning_rate":{
                                    "initial_value":0.8,
                                    "modify_every_no_episodes":100000,
                                    "type": "exponential",
                                    "theta": 1.0
                                    },
                    "epsilon":{
                            "initial_value":0.05,
                            "modify_every_no_episodes":100000,
                            "type": "exponential",
                            "theta": 1.0
                            },

                    "discount":{
                            "initial_value":0.99,
                            "modify_every_no_episodes":100000,
                            "type": "exponential",
                            "theta": 1.0
                            }
                    }
```

```
"eval_configs":{
        "frequency": 100,
        "eval_every":"None",
        "number_of_simulated_games":50,
        "eval_final_wins": 100
        },


"game_hyperparameters":{
                    "learning_rate":{
                                    "initial_value":0.8,
                                    "modify_every_no_episodes":100000,
                                    "type": "exponential",
                                    "theta": 1.0
                                    },
                    "epsilon":{
                            "initial_value":0.05,
                            "modify_every_no_episodes":100000,
                            "type": "exponential",
                            "theta": 1.0
                            },
```

Am ales sa ofer **4 rewarduri**: pentru miscarea simpla(-0.1), pentru o bucata de cascaval(+10), pentru intalnirea lui Jerry cu Tom(-100), respectiv pentru adunarea tuturor bucatilor de cascaval(+80)
Obs: am incercat mai multe posibilitati, insa acestea au fost cele mai bune

```
MOVE_REWARD = float(game_rewards["move_reward"])
WIN_REWARD = float(game_rewards["win_reward"])
LOSE_REWARD = float(game_rewards["lose_reward"])
CHEESE_REWARD = float(game_rewards["cheese_reward"])
MIN_SCORE_TRESHOLD= float(game_rewards["min_threshold"])
```

Jocul poate fi vizualizat astfel: daca s-a setat final_show la True, cand se termina de antrenat si se stabileste de la tastatura daca se mai antreaneaza cu episoade aditionale, avem 2 moduri de vizualizare: Daca se apasa enter se executa pas cu pas, iar daca se apasa z se executa pana la finalul jocului. La sfarsitul demoului putem fie sa mai vizualizam alt joc (acest lucru se poate face de cate ori se doreste), sau prin apasarea ‘x’ se incheie demo-ul.De mentionat ca la apasarea lui x in timpul demoului acesta se incheie.

Encodarea hartii (a starii) se face sub forma unui tuplu alcatuit din :
(coordonatele lui Jerry in matrice) + (coordonatele lui Tom in matrice) + (coordonatele bucatilor de cascaval in matrice – sortate crescator dupa prima,respectiv a 2-a componenta)

Pentru a determina **distanta** dintre TOM si JERRY folosim algoritmul lui Lee (BFS), calculat la inceputul Trainerului (in init pentru a optimiza timpul de antrenare). Calculam pentru fiecare pozitie (i,j) valabila din matrice distanta minima de la el la fiecare pozitie din matrice si le

stocam intr-un dictionar. (practic pentru fiecare pozitie valabila se aplica algoritmul lui Lee centrat in aceea pozitie). Daca distanta dintre pozitia lui Tom curenta si pozitia aleasa de Jerry este mai mica ca A, pentru fiecare actiune posibila pentru Tom calculam distanta de la celula care ar rezulta in urma actiunii la noua pozitie a lui Jerry. Se aleg actiunile cu distanta calculata minima si se aplica random peste ele(daca sunt mai multe).

Pentru **smoothing** am folosit un lfilter din python

Implementare politicilor:

best_action – alege actiunea cu utilitatea maxima (daca sunt mai multe egale alege random)

```
def best_action(self, Q):
    state = self.get_serialized_current_state()
    possible_actions = [action for action in self.legal_actions[state[:2]]]
    q_possible_actions = [Q.get((state, action), 0) for action in possible_actions]
    max_reward = max(q_possible_actions)
    max_choices = []
    for i in range(len(q_possible_actions)):
        if q_possible_actions[i] == max_reward:
            max_choices.append(possible_actions[i])

    return choice(max_choices)
```

1)maxFirst: Se alege actiunea returnata in urma apelului best_action

```
def maxFirst_policy(self, Q):
    next_action = self.best_action(Q)

    return next_action
```

2)random: Se alege o actiune random din cele valabile pentru starea curenta

```
def random_policy(self, Q):
    state = self.get_serialized_current_state()
    possible_actions = [action for action in self.legal_actions[state[:2]]]
    return choice(possible_actions)
```

Cojocariu Sebastian
343C1

3) exploatare: daca sunt actiuni neutilizate se ia random dintre acelea, altfel cu o probabilitate de epsilon se alege o actiune random si de 1-epsilon se alege actiunea cu utilitate maxima.

```
def exploatare_policy(self, Q):
    state = self.get_serialized_current_state()
    not_explored = [action for action in self.legal_actions[state[:2]] if not Q.get((state, action), None)]

    if len(not_explored) > 0:
        return choice(not_explored)

    elif random() < self.epsilon:
        all_possible_actions = [action for action in self.legal_actions[state[:2]]]
        return choice(all_possible_actions)

    return self.best_action(Q)
```

4) exploare exploatare: aplicam un softmax pentru valorile utilitatilor actiunilor posibile din starea curenta, nu inainte totusi de a scadea maximul dintre ele (pentru a evita overflow-ul). Codul comentat este o incercare de a folosi direct valorile utilitatilor pe post de weight-uri, prin scaderea minimului dintre ele (pentru a le face pozitive). Rezultatele sunt bune pentru un numar mare de episoade.

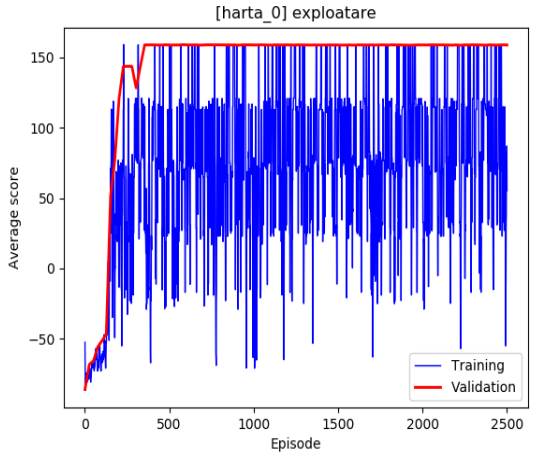
```
def explorare_exploatare_policy(self, Q):
    state = self.get_serialized_current_state()

    all_possible_actions = [key for key in self.legal_actions[state[:2]]]

    '''
    eps = 0.1
    n = len(all_possible_actions)
    q_possible_actions = [Q.get((state, action), 0) for action in all_possible_actions]
    minimum = min(q_possible_actions)
    for i in range(len(q_possible_actions)):
        q_possible_actions[i] = q_possible_actions[i] - minimum + eps

    next_action = choices(all_possible_actions, weights=q_possible_actions)

    return next_action[0]
    '''
    q_possible_actions = [Q.get((state, action), 0) for action in all_possible_actions]
    maximum = max(q_possible_actions)
    for i in range(len(q_possible_actions)):
        q_possible_actions[i] = q_possible_actions[i] - maximum

    weights = softmax(q_possible_actions)

    next_action = choices(all_possible_actions, weights=weights)

    return next_action[0]
```

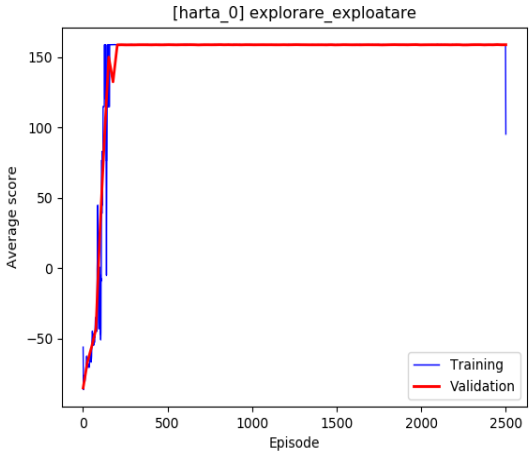
Implementarea Sarsa e aproape identica cu cea de la **Qlearning**, insa nu se mai alege max{a} Q(a,next_stare) ca la Qlearning cand se calculeaza Q(stare curenta, actiune curenta), ci se trece mai intai in starea next_stare, se aplica politica => next_actiune si se actualizeaza cu formula de la Qlearning doar ca in loc de max{a} Q(a,next_stare) se pune Q(next_stare, next_actiune) in formula pentru Q(stare curenta, actiune curenta).

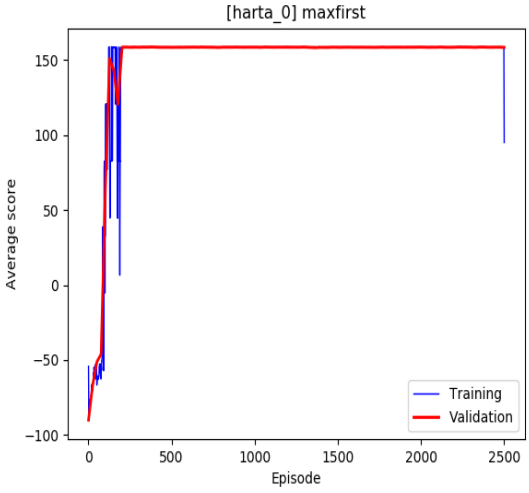
Cojocariu Sebastian
343C1

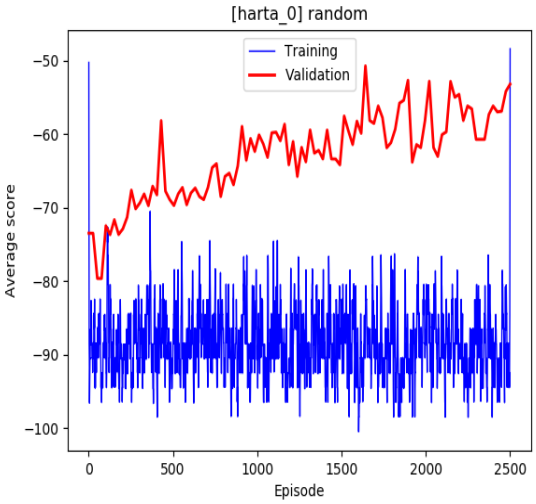
1)Evolutia scorului in functie de numarul episodului de antrenament

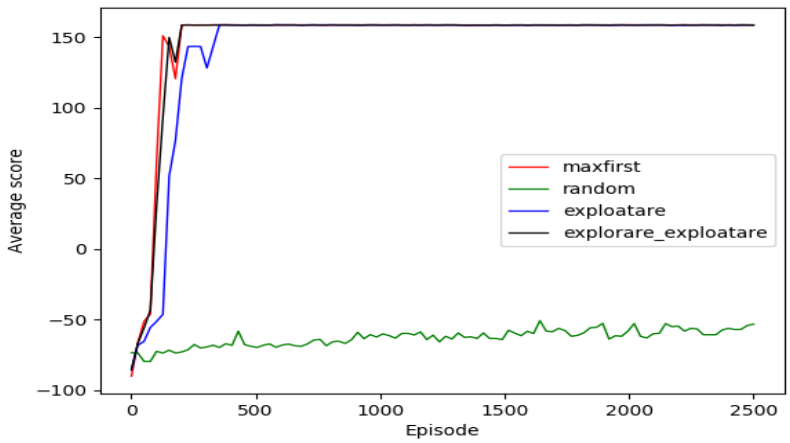
Harta 0







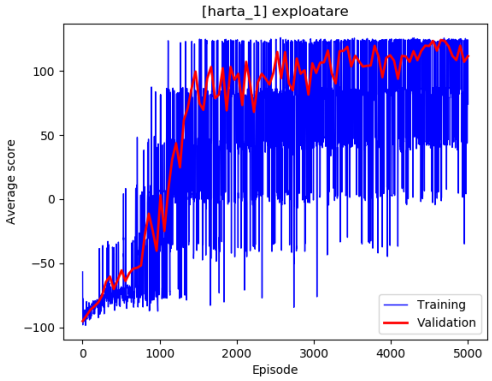


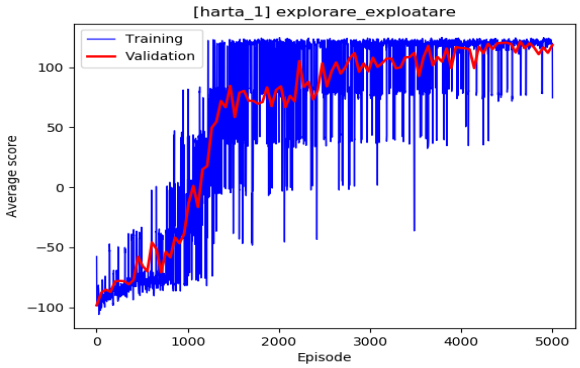


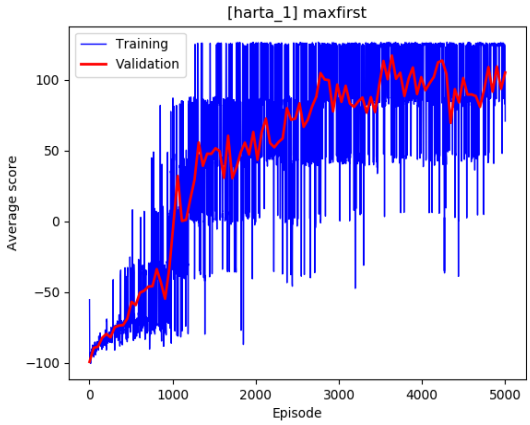
Intrucat harta este una simpla, dupa doar 200 de episoade toti algoritmii (cu exceptia random-ului) reusesc sa castige. Random nu reuseste sa ajunga la o solutie optima nici dupa 2500 de episoade☹

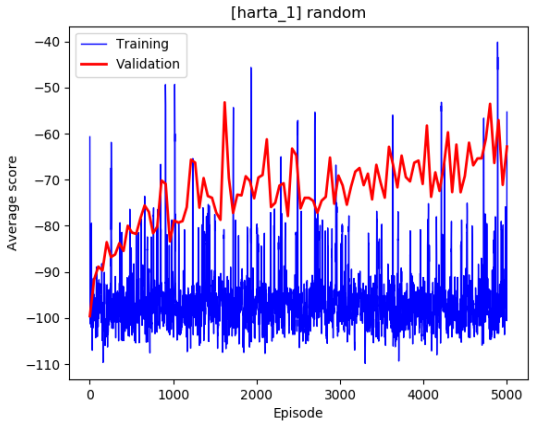
Cojocariu Sebastian
343C1

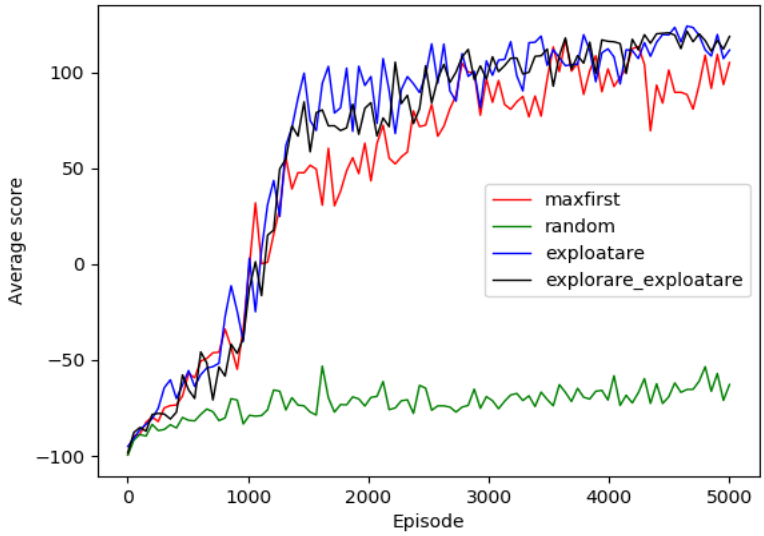
Harta 1







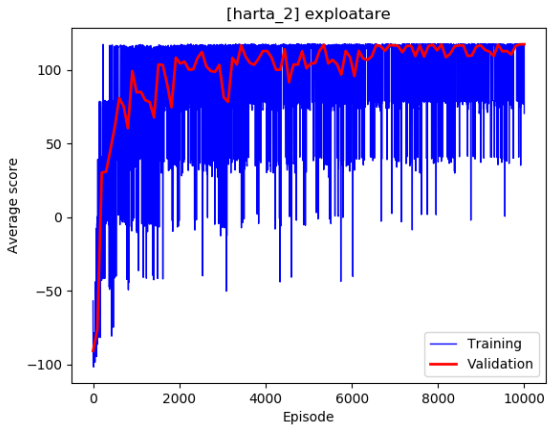


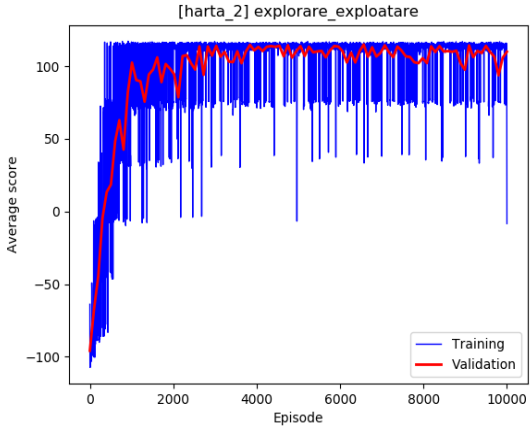


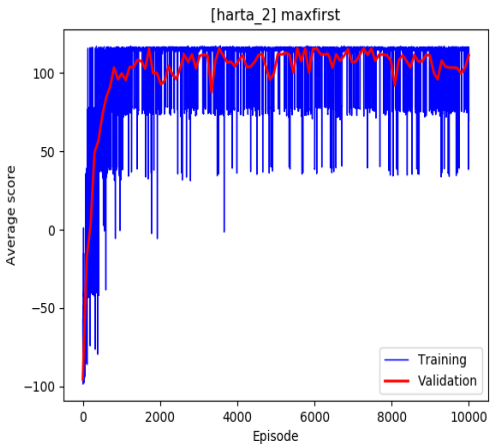
Din nou, randomul nu reuseste sa obtina rezultate satisfacatoare (pare ca reuseste in anumite cazuri sa se fereasca de Tom, insa nu suficient cat sa castige jocul). Pentru acest task, exploatarea si explorare_exploatare sunt cei mai buni, insa marginali mai rapizi decat maxFirst.

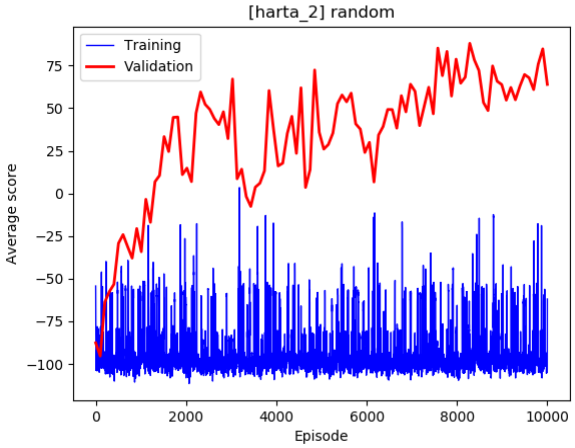
Cojocariu Sebastian
343C1

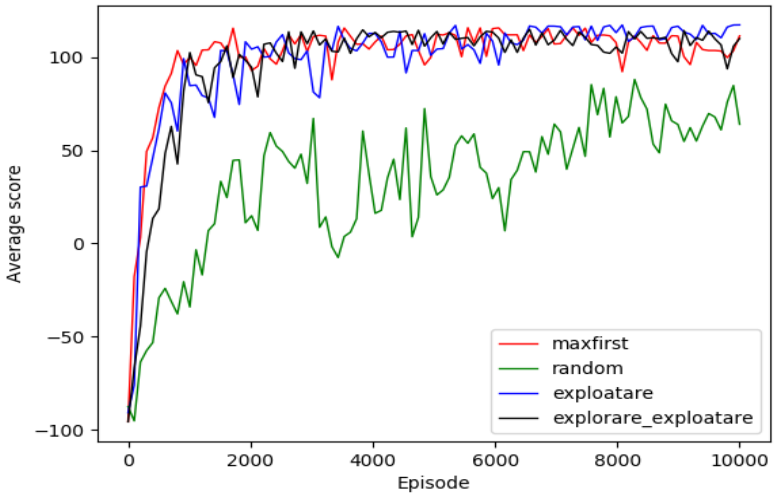
Harta 2







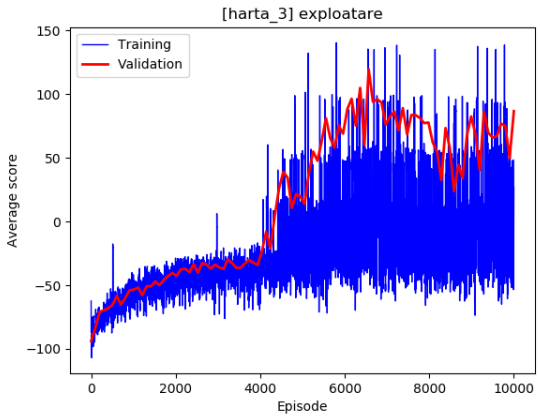


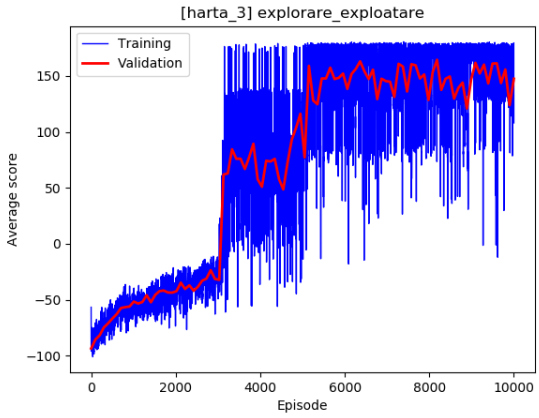


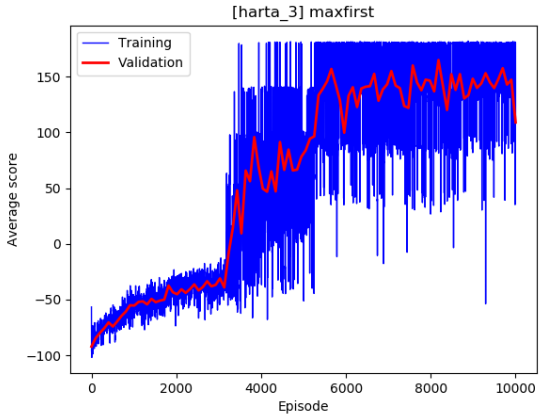
Surprinzator, random-ul obtine rezultate neasteptat de bune. Toti ceilalti algoritmi sunt extrem de apropiati ca viteza de convergenta.

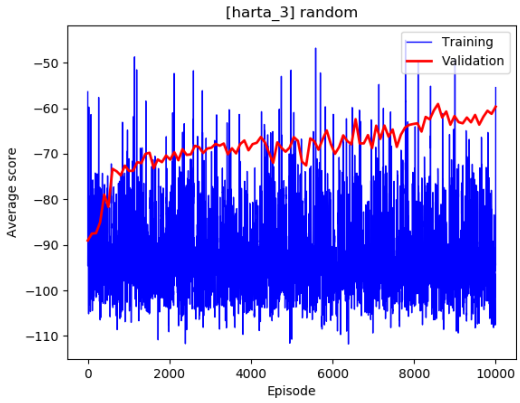
Cojocariu Sebastian
343C1

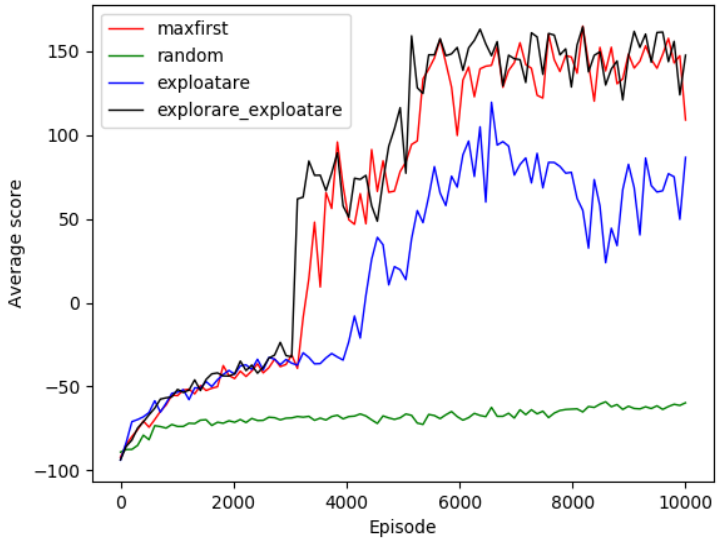
Harta 3









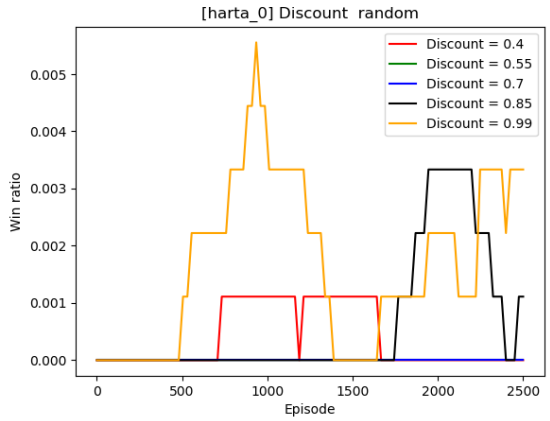


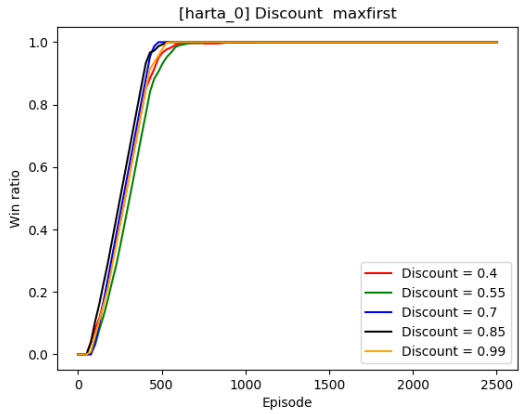
Exploatarea simpla pare sa nu fi atins inca platoul de convergenta (presupun ca inca 5000 de episoade ar fi fost suficient). MaxFirst si explorare_exploatare se comporta destul de similar, fiind si cei mai buni algoritmi dintre cei prezentati.

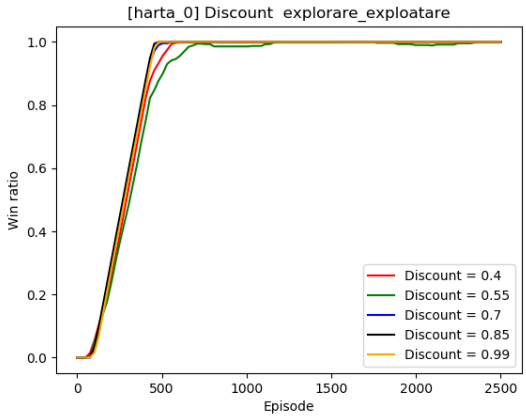
Cojocariu Sebastian
343C1

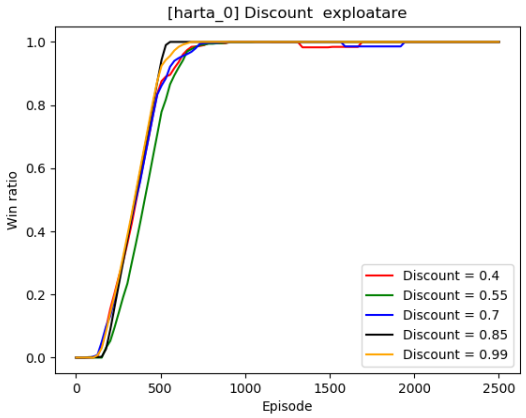
2) Procentul de jocuri castigate în funcție de valoarea discountului

Harta 0







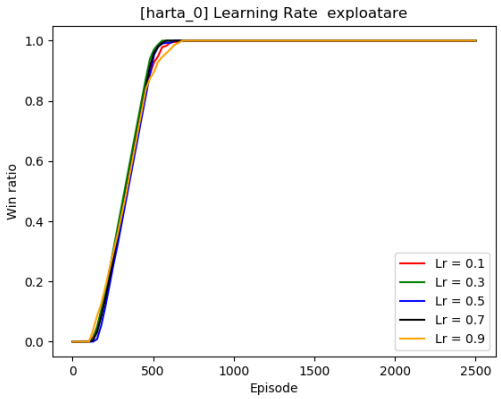


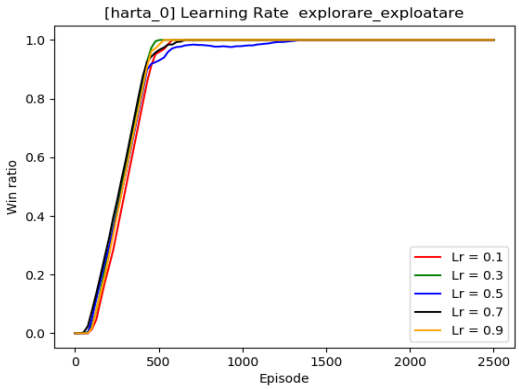
Rezultatele obtinute pentru random sunt extrem de irelevante (nu reuseste sa convearga pentru niciun discount dat). Ceilalti algoritmi par a nu fi influentati de modificarea discountului in ceea ce priveste timpul de convergenta.

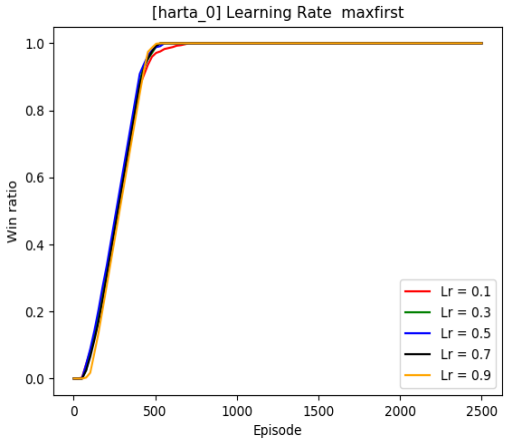
Cojocariu Sebastian
343C1

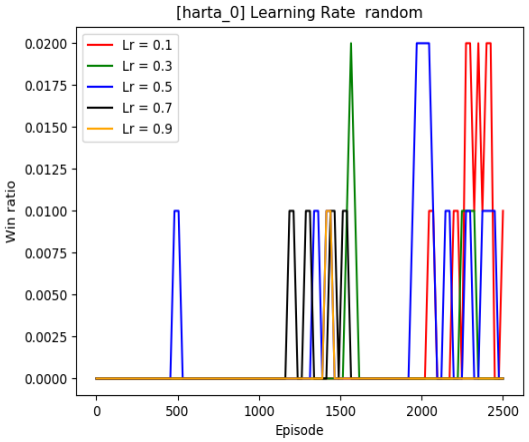
2) Procentul de jocuri castigate în funcție de valoarea learning-rateului

Harta 0







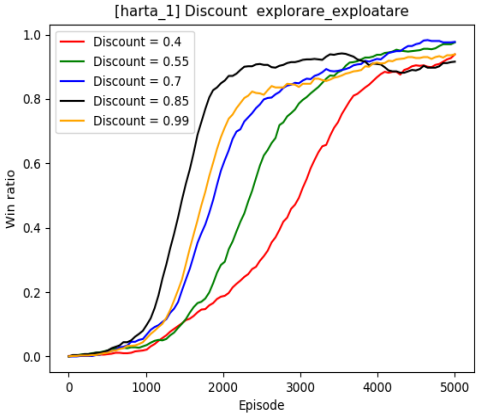


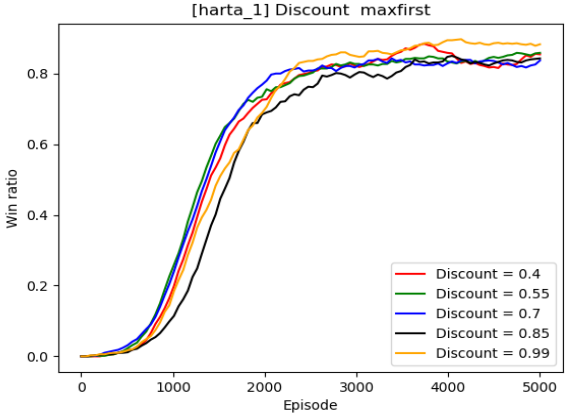
Rezultatele obtinute pentru random sunt din nou irelevante (nu reuseste sa convearga pentru niciun learning rate dat). Ceilalti algoritmi sunt robusti la modificarea lr.

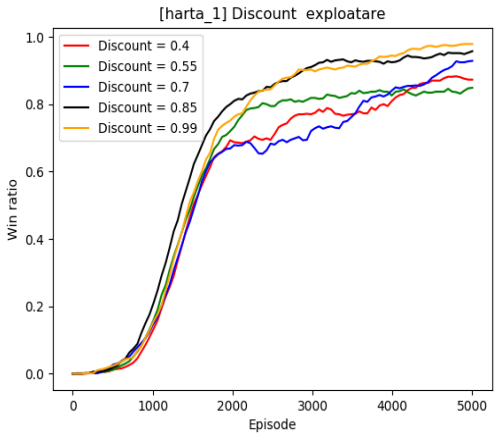
Cojocariu Sebastian
343C1

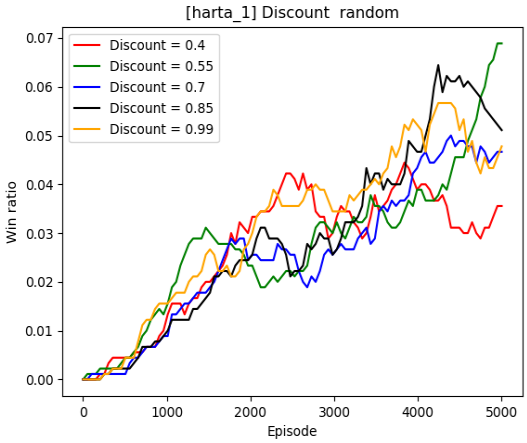
2) Procentul de jocuri castigate în funcție de valoarea discountului

Harta 1







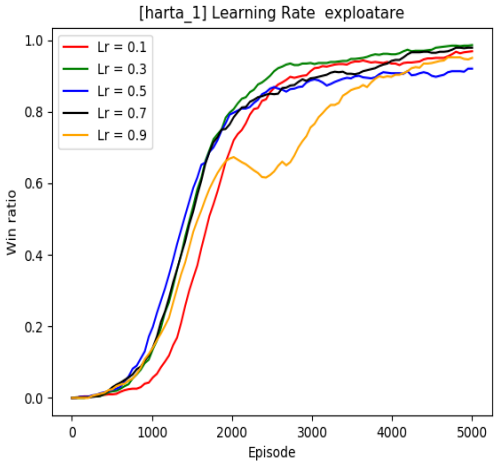


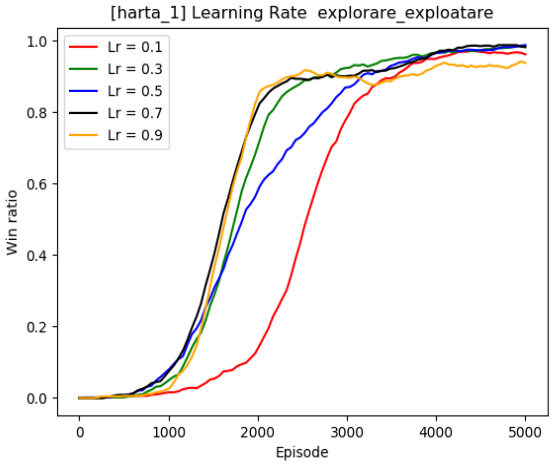
Se observa preferinta pentru discounturi mari (0.99, 0.85) in ceea ce priveste timpul de convergenta. In cazul algoritmului random se observa o performanta minimala (dar totusi mai buna decat la harta anterioara)

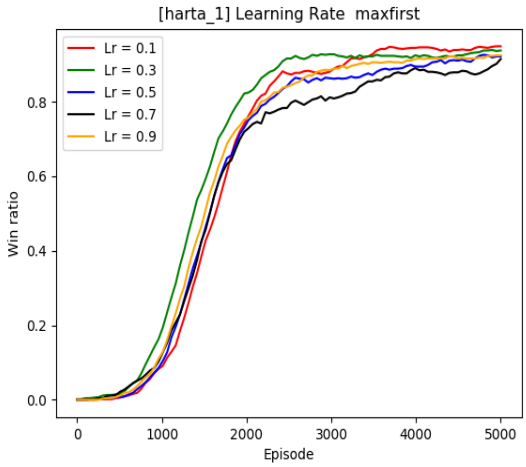
Cojocariu Sebastian
343C1

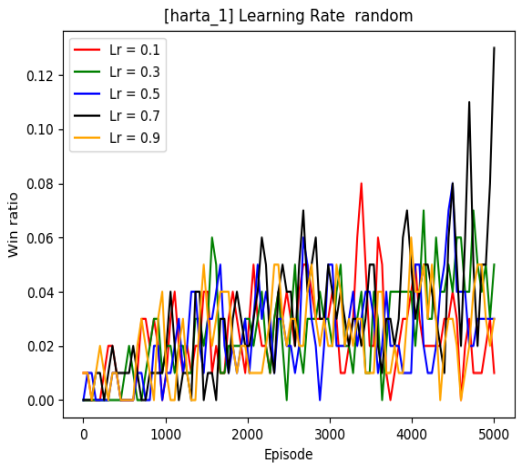
2) Procentul de jocuri castigate în funcție de valoarea learning-rateului

Harta 1







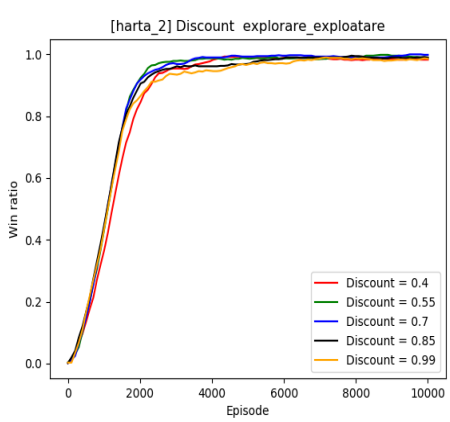


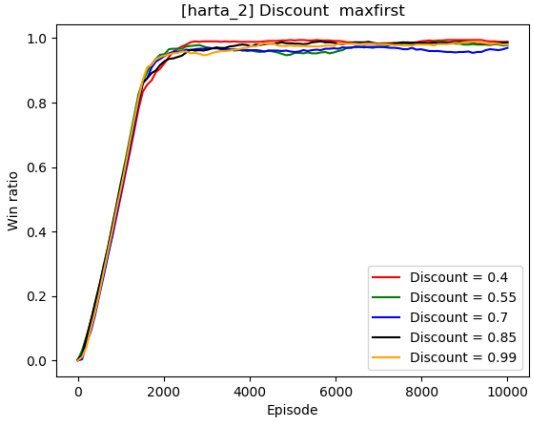
Din nou pare ca lr nu influenteaza extrem de mult timpul de invatare pentru principalii 3 algoritmi (lr=0.1 pare totusi destul de slab in exploare_exploatare). Randomul obtine performanta cea mai buna cu lr=0.7 (mai buna ca data trecuta)

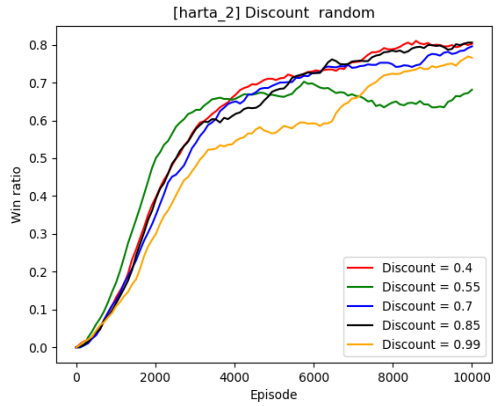
Cojocariu Sebastian
343C1

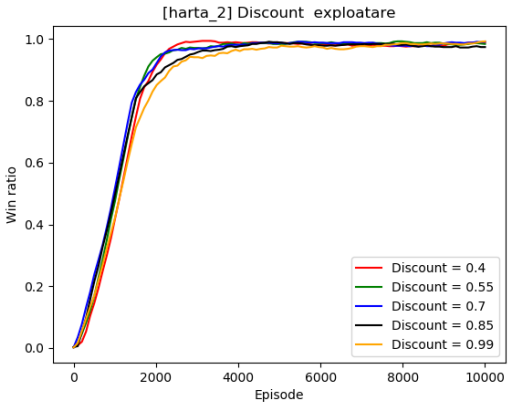
2) Procentul de jocuri castigate în funcție de valoarea discountului.

Harta 2







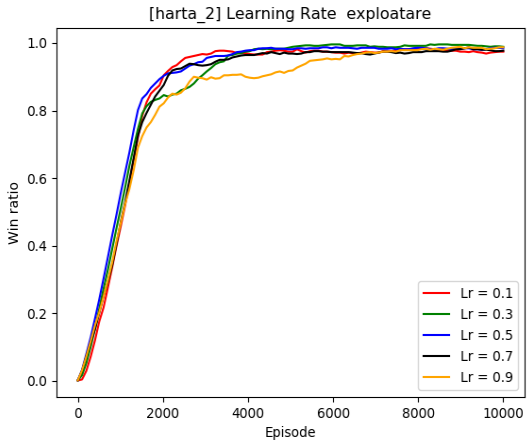


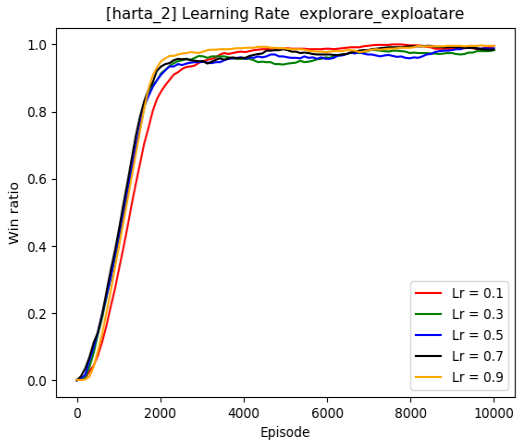
Din nou, inafara de politica random (unde exista mici fluctuatii in functie de discountul ales), celelalte politici nu sunt afectate de modificarea discountului.

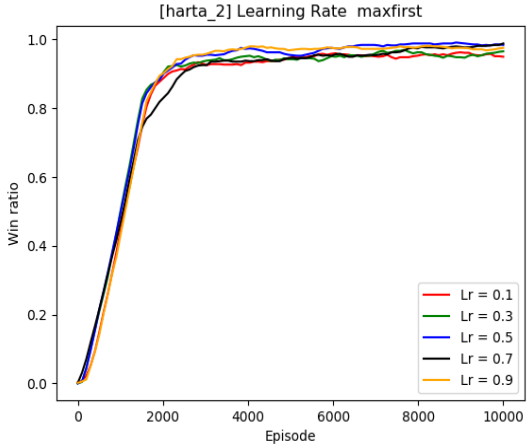
Cojocariu Sebastian
343C1

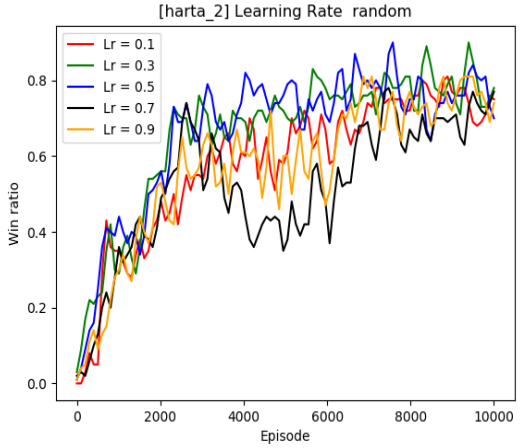
2) Procentul de jocuri castigate în funcție de valoarea learning-rateului

Harta 2







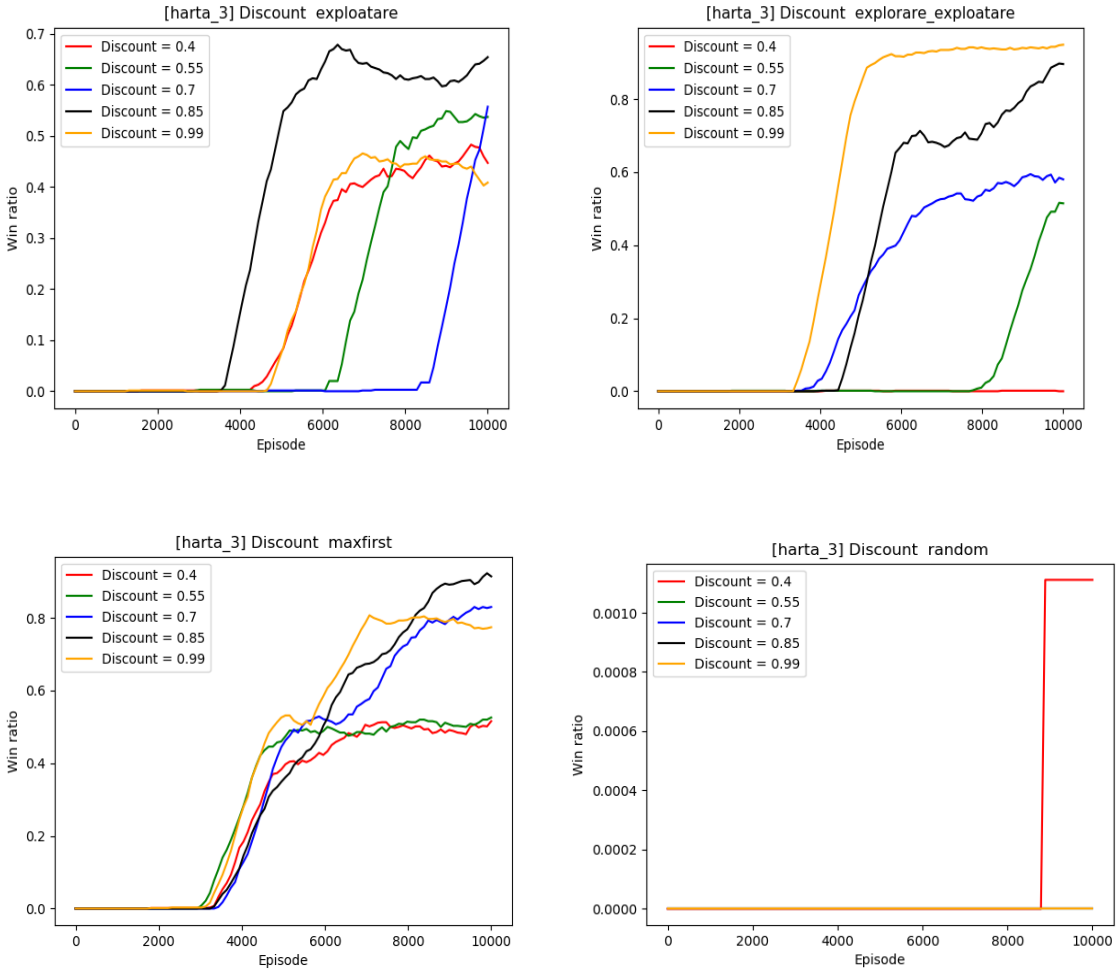


Asemanator cazului anterior. De remarcat totusi ca politica random castiga in 80% din cazuri!:)

Cojocariu Sebastian
343C1

2) Procentul de jocuri castigate în funcție de valoarea discountului.

Harta 3

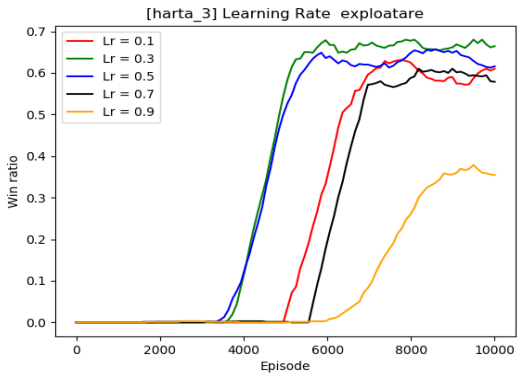


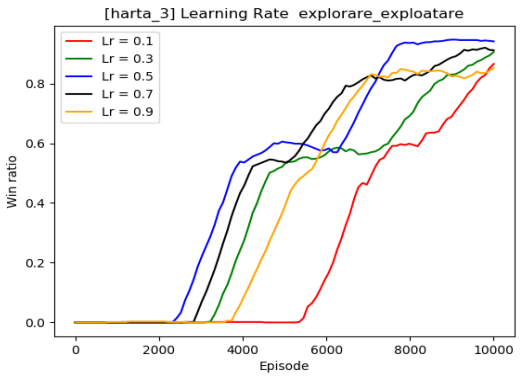
Cand vine vorba de cea mai complexa harta (15 x 15), observa, preferinta clara pentru discounturile mari (0.99 si 0.85), celelalte obtinand performante clar sub acestea. Random irrelevant.

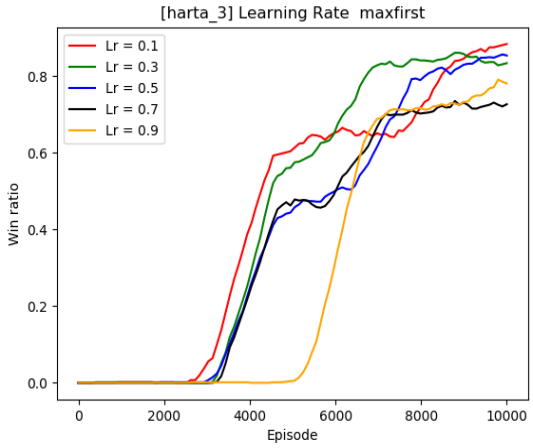
Cojocariu Sebastian
343C1

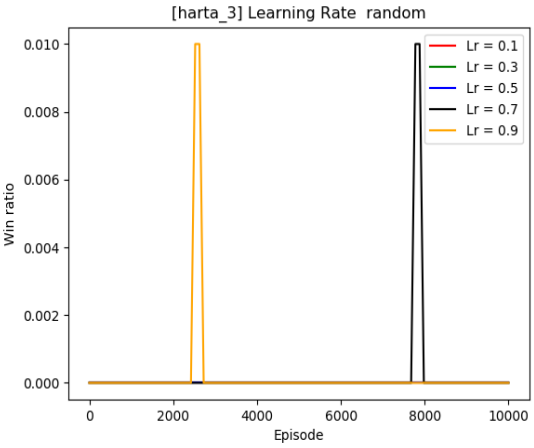
2) Procentul de jocuri castigate în funcție de valoarea learning-rateului

Harta 3







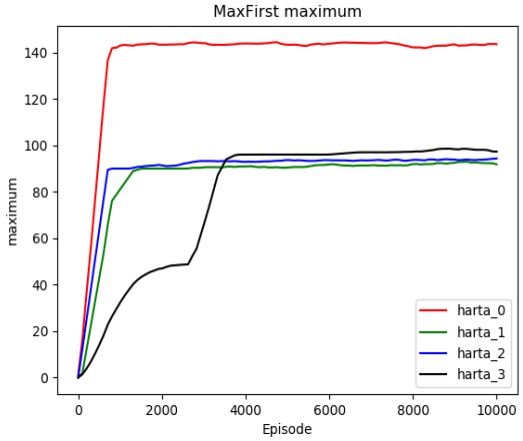


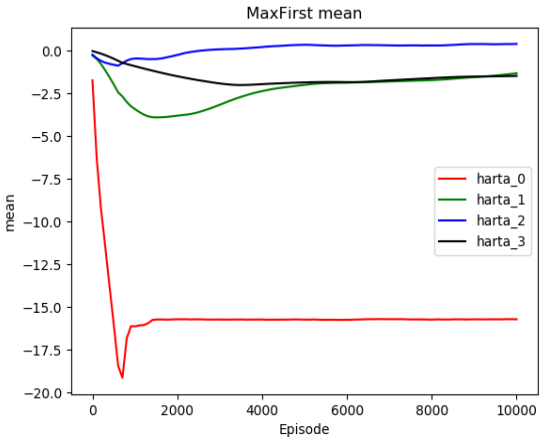
Si aici exista fluctuatii destul de vizibile, insa dupa 10000 episoade agentul pare sa convearga indifferent de lr ales (mai putin in cazul exploatarii pentru lr=0.9, lucru usor de inteles dat fiind faptul ca harta este foarte mare, iar randomnessul dat de epsilon alaturi de o valoare mare a learning rateului pare sa ingreuneze updateul utilitatilor spre convergenta).

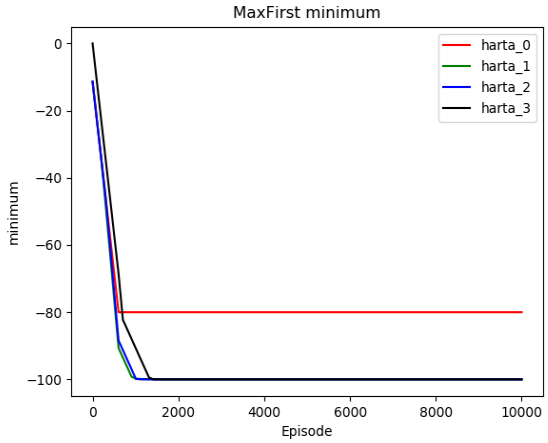
Cojocariu Sebastian
343C1

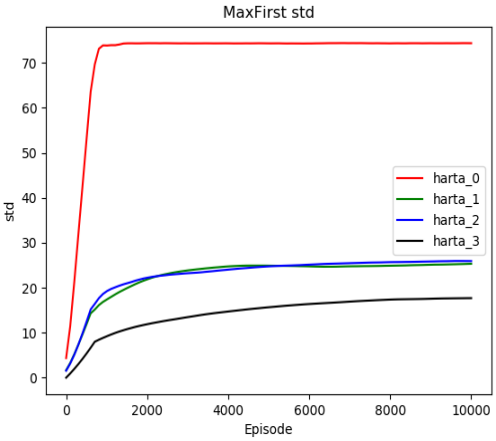
3)Grafice comparative

a) Efectul numarului de episoade de antrenament asupra valorilor din tabela de utilitati în cazul strategiei maxfirst.





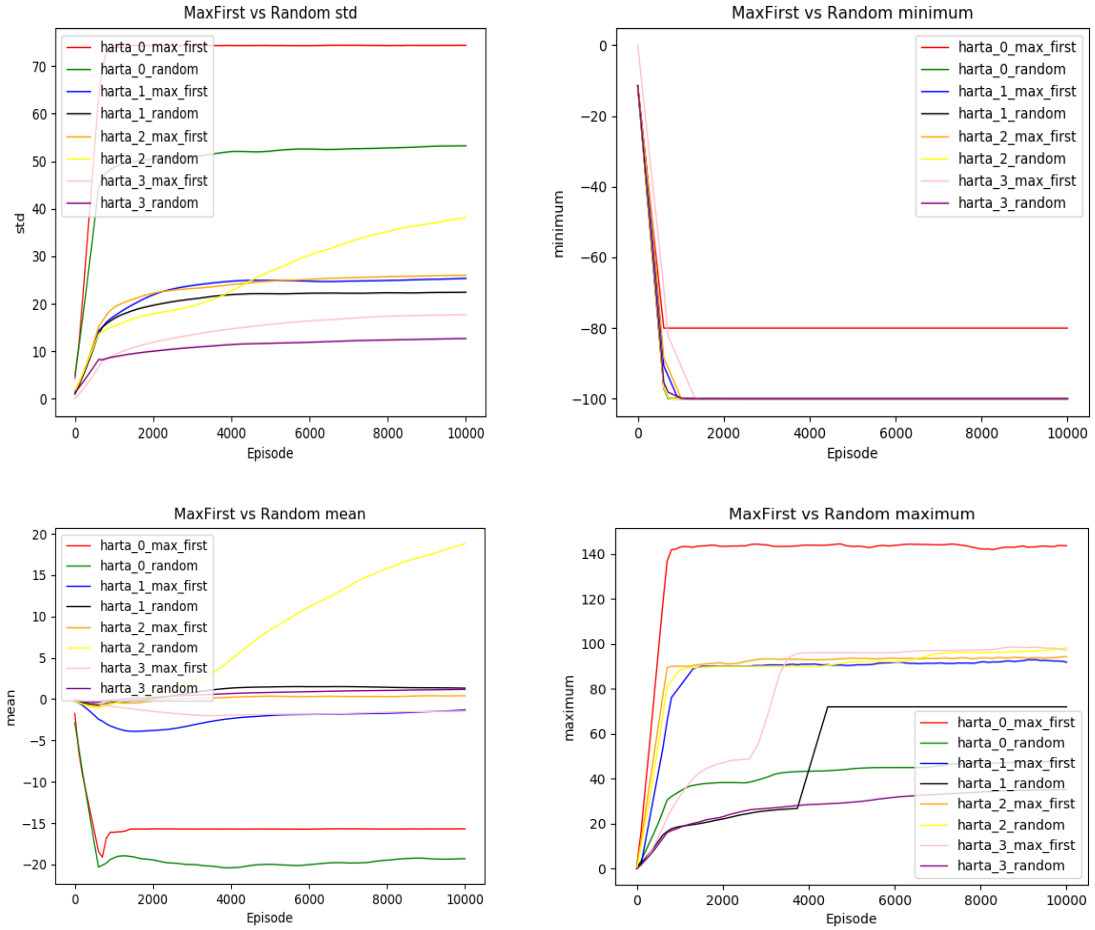




Pentru harta 0 se obtine valoarea maxima (nu trebuie sa se plimbe foarte mult, caz in care nu este penalizat prea mult pentru fiecare celula pe care ajunge), avand totusi o medie a utilitatilor minima (fiind putin celule, multe stari vor fi afectate de situatiile in care soarecele e prins si penalizat cu -100). Celelalte harti obtin o medie cuprinsa intre -10-0 , avand un maxim de aproximativ 85. Valorile par ca tind sa scada destul de mult la inceput, dupa care se stabilizeaza spre o valoare negativa.

Cojocariu Sebastian
343C1

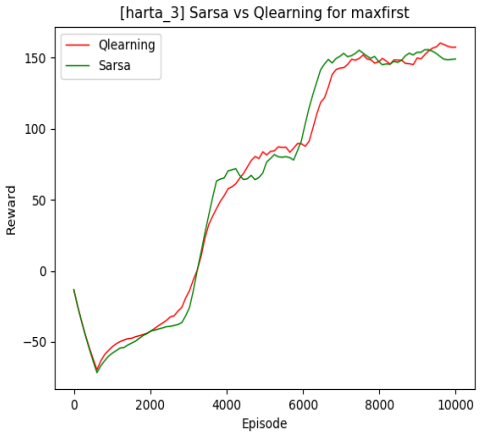
2) Diferențele intre tabela de utilitati din cazul strategiei maxfirst și random

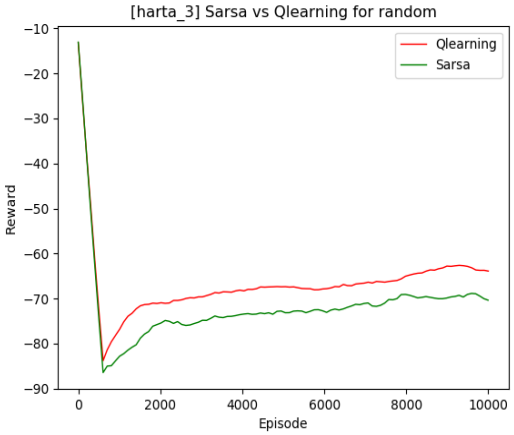


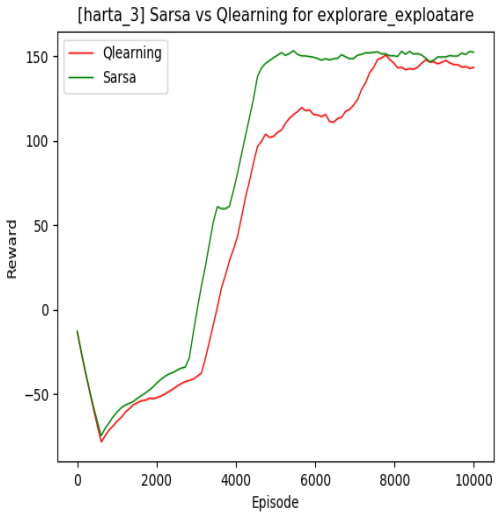
Media valorilor utilitatilor in cazul antrenarii pe aceeasi mapa cu random, respectiv cu maxFirst este mai mare in cazul antrenarii random. Acest lucru inseamna ca majoritatea utilitatilor tind sa cresca (chiar marginal) in cazul random, in timp ce in cazul maxFirst anumite utilitati (un numar mic) vor avea valoare foarte mare, insa multe vor fi foarte mici (a se vedea std).

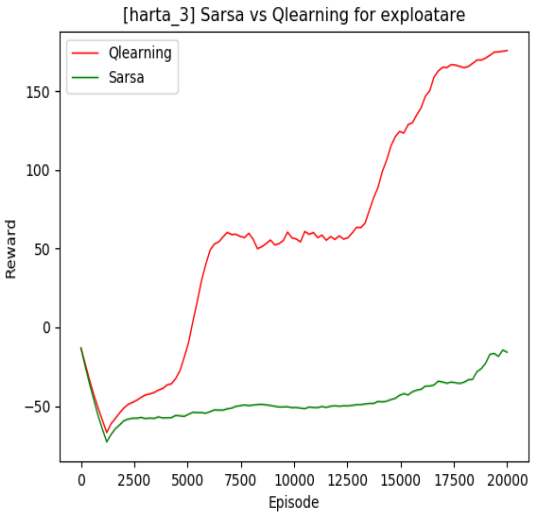
Cojocariu Sebastian
343C1

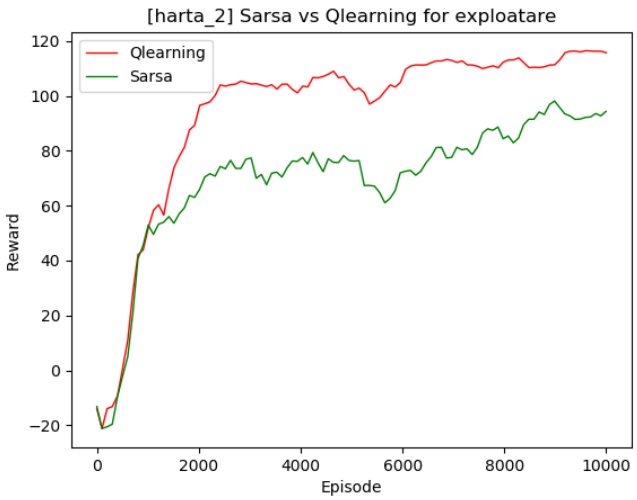
Bonus











Pentru cazul **random** Q learning face o treaba mai buna, insa nu ajunge sa obtina rezultate satisfacatoare.
In cazul **maxFirst** rezultatele sunt extrem de similar intre cei 2 algoritmi.

Cojocariu Sebastian
343C1

In cazul de **explorare_exploatare** Sarsa obtine rezultate putin mai bune decat Qlearning, cu mentiunea ca ambele se stabilizeaza cam dupa acelasi numar de episoade.
In cazul **exploatarii**, Qlearning castiga detasat. Acest lucru se datoreaza dificultatii hartii (harta_3) pe care se antreneaza, Sarsa nefiind capabila sa invete foarte bine caracteristicile hartii. Pentru o alta harta, Sarsa resuseste sa convearga, insa din nou average reward este sub Q learning.