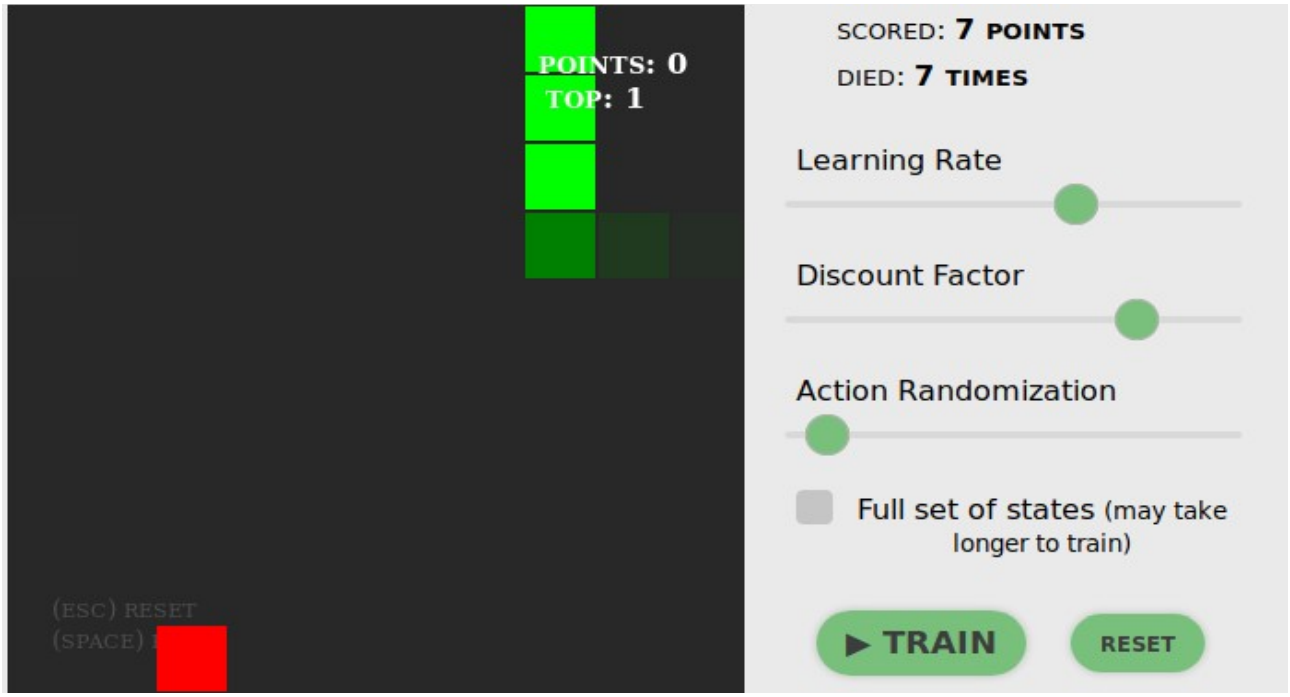
Brabie Angela Daiana

Snake Game

Am ales sa prezint implementarea pentru jocul Snake, scrisa in JavaScript in care am aplicat Q Learning pentru ca sarpele sa invete sa prinda obiectivul.

Descriere joc:

Snake este un joc in care sarpele trebuie sa se plimbe in spatiul acordat, fara a se lovi de vreun obstacol sau de sine, caz in care jocul se termina. Daca reuseste sa atinga obiectivul, dimensiunea lui creste.



Functionalitati:

- Learning Rate: cât de bine va învăța AI să joace (aproape de 0 va fi prea lent, în timp ce aproape de 1 va înlocui pur și simplu valoarea veche învățată cu cea nouă). Mai mare nu este neapărat mai bun.
- Discount Factor**:** importanța dintre recompensele imediate și recompensele viitoare
- Action Randomization: Procentul de timp în care se va executa o acțiune aleatorie în locul acțiunii dorite

Implementarea:

Implementarea a inceput de la jocul de baza (fisierul game.js) pe care am aplicat Q-learning (q-learning.js). Aceasta consta in:

s = game.state()	get state **s**
act = best-action(Q(s))	execute best action **act** given state **s**
rew = game.reward( )	receive reward **rew**
s' = game.state()	get next state **s'**
Q(s, act) = update-qTable( )	update Q-Table value **q** of state **s** and action **act**

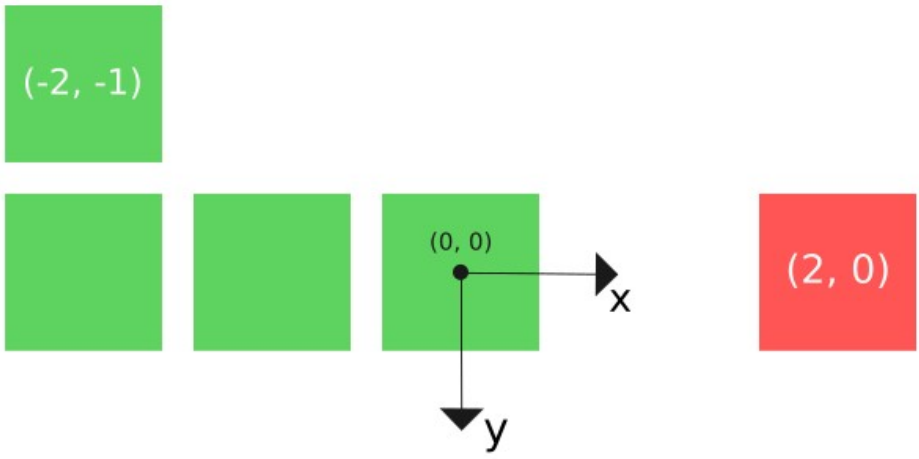
- **Matricea Q** este o matrice cu un set de stari și probabilitatea de succes a acțiunii respective. Când agentul explorează mediul, tabelul este actualizat. Acțiunea cu cea mai mare valoare este considerată cea mai bună acțiune de făcut. Exemplu:

STATES	ACTIONS			
	UP	DOWN	LEFT	RIGHT
...	...	...	...	...
.... .	0.43	0.25	0.12	**0.8**
. ....	-0.5	-0.3	**0.38**	-0.15
:..:	**0.6**	0.18	0	0.53
:'':'	-0.32	**0.75**	0.23	0.49
...	...	...	...	...

Actualizarea matricei Q se face aplicand urmatoarea formula care se execută după ce se intampla acțiunea și se cunoaște recompensa.:

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

Astfel, avand in vedere urmatoarea stare a sarpelui, matricea Q va arata ca mai jos:




```
{ '2,0,-3,0':   { up:0.43,  down:0.25, left:0.12, right:0.8   } },
{ '-2,0,3,0':   { up:-0.5,  down:-0.3, left:0.38, right:-0.15 } },
{ '0,-1,-3,-1': { up:0.6,   down:0.18, left:0,    right:0.53  } },
{ '0,1,3,1':    { up:-0.32, down:0.75, left:0.23, right:0.49  } }
```

Demo pentru joc se poate vizualiza accesand fisierul index.html. Se pot modifica **Learning Rate, Discount Factor, Action Randomization** in functie de care comportamnetul sarpelui difera dar si numar de puncte asimilate in comparatie cu numarul de esecuri.

Cel mai bun raport intre scored si died l-am obtinut avand urmatoarele setari: