

Üzleti Intelligencia

3. Előadás: Markov döntési folyamatok megoldása

Kuknyó Dániel
Budapesti Gazdasági Egyetem

2023/24
1.félév

- 1 Bevezetés
- 2 A rabló probléma
- 3 Dinamikus programozás
- 4 Politika iteráció

1 Bevezetés

2 A rabló probléma

3 Dinamikus programozás

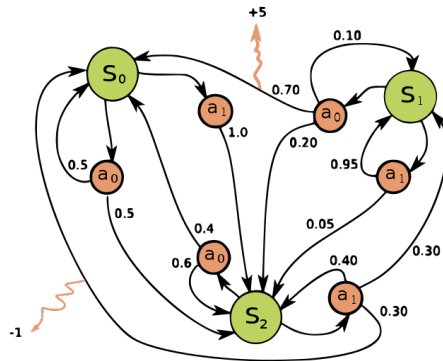
4 Politika iteráció

Az RL modellje

Markov döntési folyamat

$$MDP(S, A, P, R, s_0, \gamma)$$

- S : állapotok halmaza
- A : cselekvések halmaza
- $P : S \times A \times S \rightarrow [0, 1]$:
állapotátmeneti valószínűségek
- $R : S \times A \rightarrow \mathbb{R}$: azonnali jutalmak
- s_0 : kezdőállapot
- γ : diszkont faktor



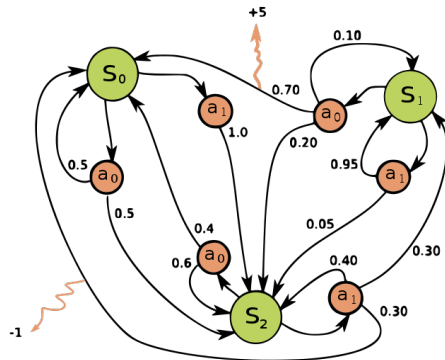
Az RL modellje

Az MDP folyamata:

- 1 Az ügynök s_0 állapotból indul
- 2 Az ügynök π politika szerint cselekszik:
 $a \sim \pi(s)$
- 3 A környezet reagál a cselekvésre, és visszaadja az ügynöknek r jutalmat és s' következő állapotot
- 4 Ez ismétlődik amíg a kilépési kritérium be nem teljesül

Cél: Az optimális politika megtalálása. A politika optimális, ha a hozamának várható értéke maximális:

$$E_{\pi} (r_1 + \gamma r_2 + \gamma^2 r_3 + \dots) \rightarrow \max$$



A mohó ügynök

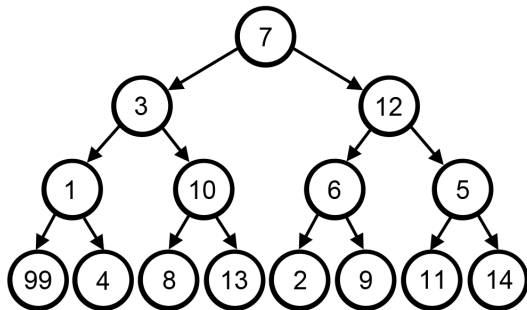
A legegyszerűbb cselekvés kiválasztási szabály, ha az ügynök mindig azt a cselekvést választja, ami számára a lehető legnagyobb várható hozammal rendelkezik.

Mohó cselekvés választás

Mohó politika mindig azt a cselekvést fogja választani, amelyik - egy lépéses távlatban - a lehető legnagyobb várható jutalommal fog járni az ügynök számára v_π szerint.

$$a_t = \underset{a}{\operatorname{argmax}} Q_t(a)$$

- Mi lenne a mohó politika ebben az esetben?
- Mindig ez a legjobb megoldás?
- A legjobb megoldás mindig mohó?



A mohó ügynök

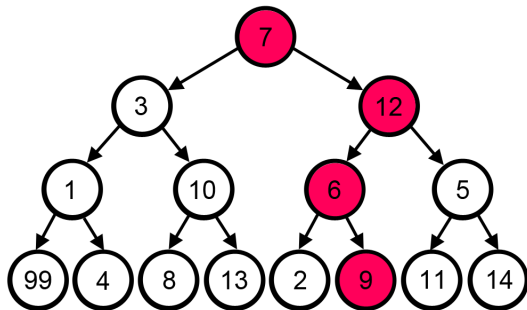
A legegyszerűbb cselekvés kiválasztási szabály, ha az ügynök mindig azt a cselekvést választja, ami számára a lehető legnagyobb várható hozammal rendelkezik.

Mohó cselekvés választás

Mohó politika mindig azt a cselekvést fogja választani, amelyik - egy lépéses távlatban - a lehető legnagyobb várható jutalommal fog járni az ügynök számára v_π szerint.

$$a_t = \underset{a}{\operatorname{argmax}} Q_t(a)$$

- Mi lenne a mohó politika ebben az esetben?
- Mindig ez a legjobb megoldás?
- A legjobb megoldás mindig mohó?



Az ε -mohó stratégia

Egy másik lehetőség, ha adott valószínűséggel az ügynök véletlen cselekvést hajt végre remélve, hogy ezzel elér egy olyan állapotba amelyhez nagy jutalom tartozik. A véletlen cselekvés a **felfedezés**, és végrehajtásának valószínűsége ε .

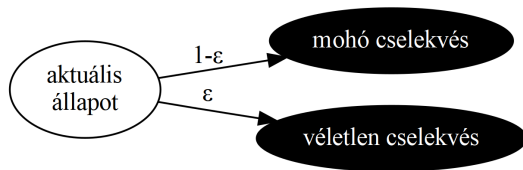
ε -mohó cselekvés választás

$$a_t \leftarrow \begin{cases} \underset{a}{\operatorname{argmax}} Q_t(a) & P=1-\varepsilon \\ a \sim A & P=\varepsilon \end{cases}$$

Ahol A az összes cselekvés halmaza.

Az ügynök tehát ε valószínűséggel véletlen cselekvést választ az ismeretlen, de nagyobb jutalom reményében. Ez a **felfedezés** művelete.

ε valószínűséggel pedig a már ismert és a legnagyobb várható jutalommal járó cselekvést hajtja végre. Ez a **kizsákmányolás** művelete.



Példák

A következő valós példák alkalmasak a felfedezés/kizsákmányolás dilemma bemutatására:

- Étterem választás:
 - **Kizsákmányolás:** elmész a kedvenc éttermedbe.
 - **Felfedezés:** elmész egy új étterembe, hátha találsz egy jobbat mint a kedvenced.
- Online hirdetés:
 - **Kizsákmányolás:** a legjobb reklám megmutatása a felhasználónak.
 - **Felfedezés:** egy új reklám megmutatása a felhasználónak, hátha tetszik neki.
- Olajfúrás:
 - **Kizsákmányolás:** Egy meglévő helyen fúrás az olajért.
 - **Felfedezés:** Egy új helyen fúrás.
- Klinikai kezelés:
 - **Kizsákmányolás:** A bevált kezelés alkalmazása.
 - **Felfedezés:** Új kezelés kipróbálása.

1 Bevezetés

2 A rabló probléma

3 Dinamikus programozás

4 Politika iteráció

A rabló probléma

A k -karú rabló problémája egy elméleti megerősítéses tanulás probléma. A játékos egy rablógépen játszik, amelynek k karja van.

Minden karhúzás után egy állandó eloszlásból választott jutalmat kap az ügynök. Az ügynök célja, hogy olyan politikát válasszon, ami az elvárt hozamot maximalizálja 1000 cselekvés vagy időlépés után.



A rabló probléma

Az ügynöknek számon kell tartania, mennyi a jutalom várható értéke, ha adott egy a cselekvés. Ez a $Q(s, a)$ állapot-cselekvés minőség függvény. A rabló problémában csak egy állapot van, ezért elég csak a cselekvésekhez tartozóan számon tartani:

$$Q_*(a) = E[r_t | a_t = a]$$

A jutalom várható értéke:

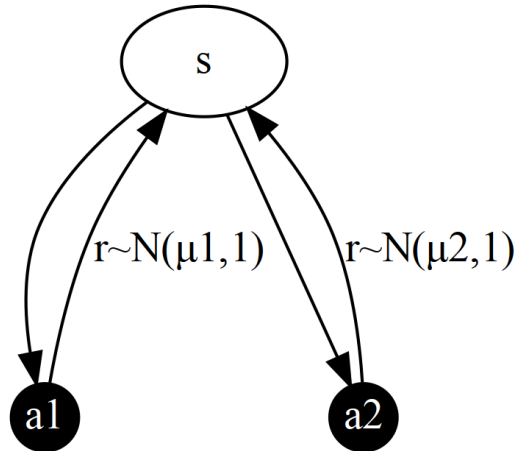
$$Q_n = \frac{r_1 + r_2 + \dots + r_{n-1}}{n - 1}$$



A rabló folyamat modellje

A példában egy kétkarú rabló folyamat modellje látható. A modell egyetlen állapotot tartalmaz. az ügynök minden lépésben innen választhat, hogy melyik kart húzza meg. Ez a két cselekvéssel egyezik meg: a_1, a_2 .

A jutalmak minden cselekvés után egy normál eloszlásból származnak, valamilyen μ_1 és μ_2 várható értékkel és 1 szórással: $r \sim N(\mu, 1)$.



Algoritmus 1: Rabló játék

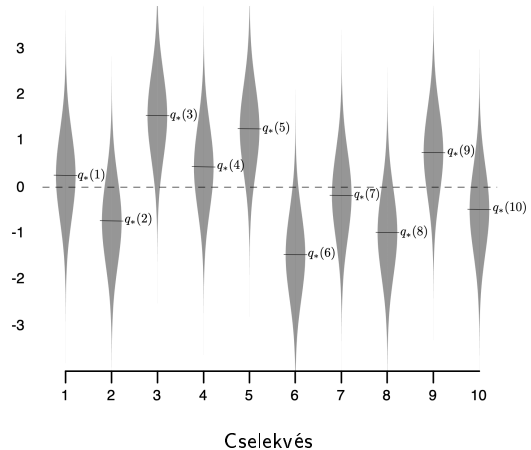
```
 $Q(a) \leftarrow 0$  for  $a = 1 \rightarrow k$ ;           /* Cselekvés minőségének függvénye */  
 $N(a) \leftarrow 0$  for  $a = 1 \rightarrow k$ ;       /* Kar meghúzásainak a száma */  
for  $t = 1 \rightarrow \max_t$  do  
     $p = \text{random}(0, 1)$ ;                /* Véletlen szám 0 és 1 között */  
    if  $p > \varepsilon$  then  
         $a \leftarrow \underset{a}{\operatorname{argmax}} Q(a)$ ; /* Legnagyobb ismert jutalom cselekvése */  
    else  
         $a \leftarrow a \sim A$ ;              /* Véletlen cselekvés */  
    end  
     $r \leftarrow \text{env}(a)$ ;                /* Cselekvés végrehajtása a környezetben */  
     $N(a) \leftarrow N(a) + 1$ ;             /* Cselekvés számlálójának növelése */  
     $Q(a) \leftarrow Q(a) + \frac{1}{N(a)} [r - Q(a)]$ ; /*  $Q$ -érték frissítése */  
end
```

Egy példa rabló

Hogy meg lehessen mérni a mohó és ε -mohó stratégiák teljesítményét, szükség van egy tesztrablóra. A példában szereplő egy 10-karú rabló. Minden karhoz tartozóan a jutalmak eloszlása Gauss-i eloszlást követ 1 varianciával, viszont nem 0 átlaggal.

Valamelyik karok nagyobb valószínűséggel járnak magas jutalommal mint a többi. Az ügynök feladata megtalálni melyik kartól remélhet nagyobb jutalmat. Ehhez szükség van arra, hogy végig próbálja őket.

A jutalmak eloszlása

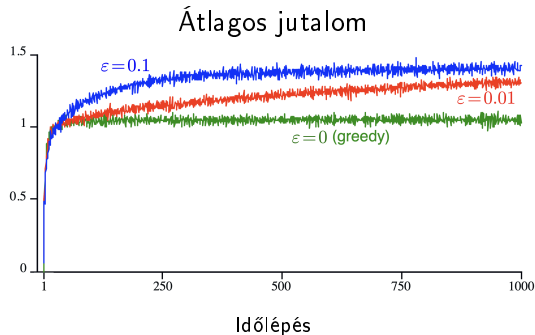


A futás teljesítménye

Az algoritmus 1000 időlépésen keresztül futott $\varepsilon = 0, \varepsilon = 0.01, \varepsilon = 0.001$ hiperparaméterekkel. Minél nagyobb a ε érték, annál nagyobb a felfedezés valószínűsége.

Mindegyik módszer megbecsülte az állapot-cselekvés minőség függvényt a rabló minden karára a mozgóátlagolás technikájával. A diagramon a várható jutalom mértékét mutatja az időlépések függvényében.

A mohó stratégia kezdetben gyorsabban javult mint a többi, de kisebb értékre konvergált a futásidő végére.

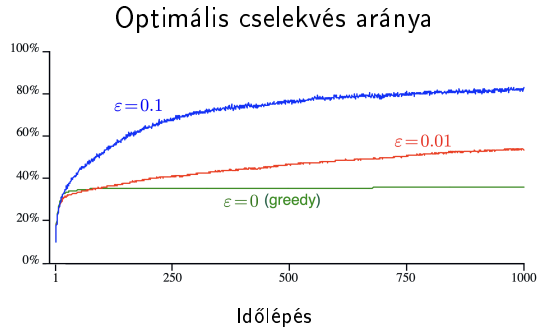


A futás teljesítménye

Az ábra azt mutatja, hogy a mohó módszer csak a feladatok mintegy 30%-ában találta meg az optimális műveletet. Az ε -mohó módszerek végül jobban teljesítettek, mert folytatták a felfedezést és javították az esélyüket az optimális művelet felismerésére.

Az $\varepsilon = 0.1$ módszer többet fedezett fel, és általában korábban megtalálta az optimális műveletet, de soha nem választotta ki azt több mint 91%-ban.

Az $\varepsilon = 0.01$ módszer lassabban javult, de végül mindkét teljesítménymérőn jobban teljesített.

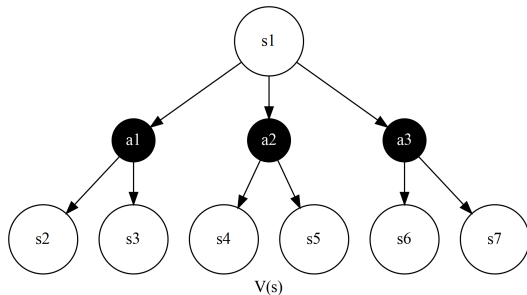


- 1 Bevezetés
- 2 A rabló probléma
- 3 Dinamikus programozás
- 4 Politika iteráció

Dinamikus programozás alapjai

A dinamikus programozás egy gyűjtőfogalom olyan algoritmusokra amelyekkel kiszámolható az optimális politika *ha adott egy tökéletes környezeti modell egy Markov döntési folyamatként.*

A klasszikus dinamikus programozási algoritmusok ritkák a megerősítéses tanulásban **mert egy tökéletes környezeti modellt feltételeznek** és mert rendkívül erőforrás igényesek.

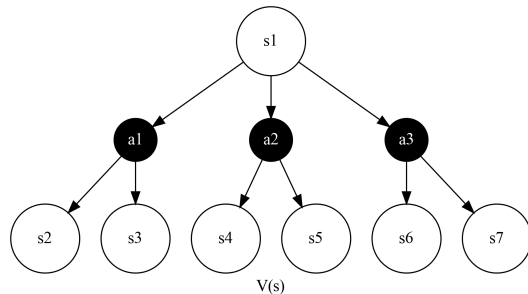


Dinamikus programozás alapjai

Dinamikus programozás

A DP algoritmusok a komplex problémákat alproblémákra bontják, majd a végső megoldást az alproblémák megoldásaiból állítják elő. Ehhez két feltételnek kell érvényesnek lennie:

- **Optimális alstruktúra:** Az almegoldásoknak felhasználhatóknak kell lenniük a probléma megoldására.
- **Átfedésben lévő alproblémák:** Bizonyos alproblémák megoldásait többször is fel lehet használni hasonló feladatok elvégzéséhez.



Példa dinamikus programozásra

A példa a Fibonacci számok kiszámításának dinamikus programozása. A Fibonacci számokat a következőképpen lehet definiálni:

Fibonacci sorozat

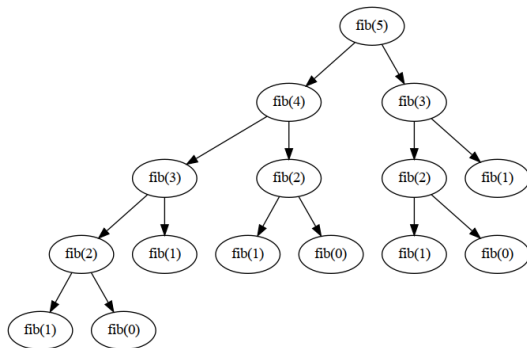
$$F_0 = 0 ; F_1 = 1$$

és

$$F_n = F_{n-1} + F_{n-2}$$

Tehát a sorozat első pár tagja:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144



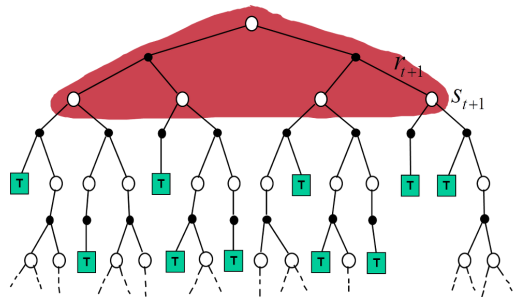
Dinamikus programozás az RL-ben

DP állapot-érték frissítési szabálya

$$V(s) \leftarrow E_{\pi} [r + \gamma V(s')]$$

- E_{π} : Várható érték π politika alatt
- $V(s)$: Cselekvés-érték függvény az aktuális s állapotban
- r : Jutalom a cselekvésért
- γ : Diszkont faktor
- $V(s')$: állapot-érték függvény s_{t+1} következő állapotban.

A megerősítéses tanulásban a dinamikus programozás egy szélességi bejárásnak felel meg. Mivel az állapotok tere túlságosan nagy, ez gyakran nem vezet megoldáshoz.



Markov döntési folyamatok dinamikus programozása

A Markov döntési folyamatok kielégítik a dinamikus programozás feltételeit.

Az értékfüggvény eltárolja és újra felhasználja a kiszámított megoldásokat: ez egy gyorsítótárként szolgál azoknak az információknak az MDP-ről, ami megadja, hogy mennyi a jutalom várható értéke egy s állapotból indulva:

$$V_{\pi}(s) = E_{\pi} [G_t | S_t = s] = E_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid S_t = s \right]$$

A Bellman egyenlet megadja, hogyan kell lebontani az optimális állapot-érték függvényt két részre: a következő időlépés optimális cselekvése és az összes többi lépés optimális cselekvése:

$$V_{\pi}(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) [r + \gamma V_{\pi}(s')] \quad minden s \in S - re$$

Dinamikus programozás felhasználásai

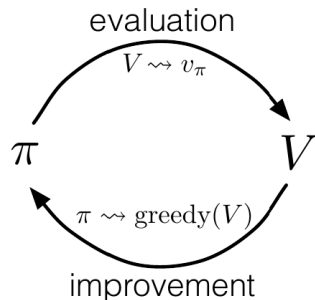
A dinamikus programozási eljárások akkor tudnak megoldani megerősítéses tanulási problémákat, ha adott a környezet dinamikája (az állapotok, az állapotátmeneti valószínűségek, jutalmak). Ezért két fő felhasználása van:

1. Politika kiértékelés

Ha adott egy $MDP(S, A, P, R, \gamma, s_0)$ és egy π politika, a feladat megtalálni π -hez tartozó v_π állapot-érték függvényt ahhoz, hogy meg lehessen mondani mennyire jövedelmező a politika.

2. Politika keresés

Ha adott egy $MDP(S, A, P, R, \gamma, s_0)$, a feladat megtalálni az optimális állapot-érték függvényt (v_π) és a hozzá tartozó π_* optimális politikát.



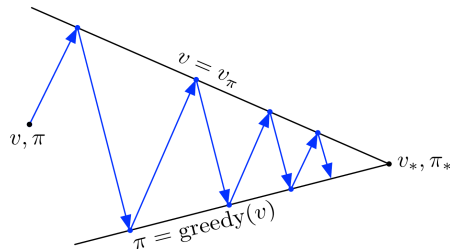
- 1 Bevezetés
- 2 A rabló probléma
- 3 Dinamikus programozás
- 4 Politika iteráció

Politika iteráció

Miután egy π politika javult v_π segítségével annak érdekében, hogy egy π' jobb politikát eredményezzen ki lehet számítani $v_{\pi'}$ javított állapot-érték függvényt és felhasználni egy újabb javított politika, π'' kiszámítására. Ezáltal egy monoton javuló politika - értékfüggvény sorozatot eredményezve:

$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} V_*$$

- E : Kiértékelés (evaluation)
- I : Javítás (improvement)
- V_* : Optimális állapot-érték függvény
- π_* : Optimális politika



Algoritmus 2: Politika kiértékelése

```
while  $\Delta > \theta$  do
   $\Delta \leftarrow 0$ ;                                /* Hiba nullára állítása */
  for  $s \in S$  do
     $v \leftarrow V(s)$ ;                            /* Jelenlegi állapot-érték */
     $V(s) \leftarrow \sum_{s',r} p(s',r,|s,\pi(s)) [r + \gamma V(s')]$ ; /* Új állapot-érték */
     $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ ;      /* Hiba kiszámolása */
  end
end
```

- Δ : $V(s)$ jelen állapot-érték és $V(s')$ következő állapot-érték különbsége.
- θ : hibahatár: egy alacsony szám ami a becslés pontosságát adja.
- $p(s',r,|s,\pi(s))$: s' következő állapot és r jutalom valószínűsége ha adott s állapot és $\pi(s)$ cselekvés π politika szerint (a környezet dinamikája).

Algoritmus 3: Politika javítása

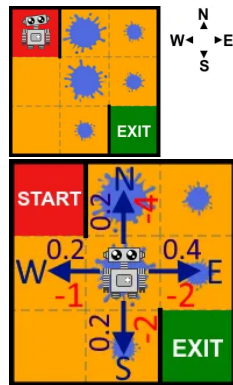
```
 $\pi_{instabil} \leftarrow false;$                                 /* Politika instabilon indul */  
while  $\pi_{instabil}$  do  
  for  $s \in S$  do  
     $a \leftarrow \pi(s);$                                 /* Jelenlegi cselekvés */  
     $\pi(s) \leftarrow \underset{a}{argmax} \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')];$  /* Új cselekvés */  
    if  $a \neq \pi(s)$  then  
       $\pi_{instabil} \leftarrow false;$                 /* politika instabillá állítása */  
    end  
  end  
end  
return  $V \approx V_*, \pi \approx \pi_*$ 
```

A politika ebben az esetben mohó, tehát úgy választja ki a cselekvést, hogy melyik következő állapothoz tartozik a lehető legnagyobb várható jutalom. Egy politika akkor számít stabilnak, amikor egyik lépésben sem változik a cselekvés.

Példa politika iterációra

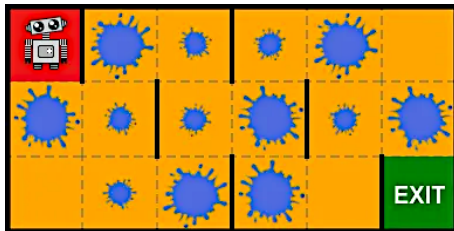
Cél: a robotnak el kell jutnia a célhoz, miközben minél kevesebb üzemanyagot használ. A környezetben a következő változók érvényesek:

- Száraz kockák:
 - 1 időegység alatt megy végig rajta a robot. -1 jutalmat kap ha egy ilyen kockára lép.
 - A célállapotot mindig eléri, mert ilyenkor nem csúszik el.
- Kis tócsák:
 - 2 időegység rajta átjutni, tehát -2 jutalmat kap érte a robot.
 - A csúszás valószínűsége 0.4 , ezért az idő 40% -ában nem éri el a célállapotot, hanem valamelyik másik lehetséges célállapotba csúszik át.
- Nagy tócsák:
 - 4 időegység alatt lehet rajta átjutni, ezért -4 jutalom jár érte.
 - A csúszás valószínűsége 0.6 .



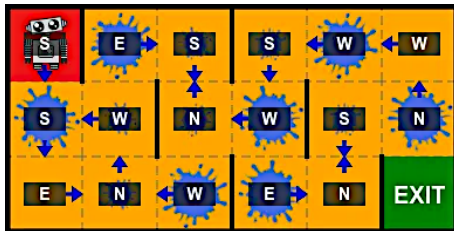
Példa politika iterációra

A robotnak a bal felső kockából kell a jobb alsóba eljutnia úgy, hogy a falakat megkerüli.



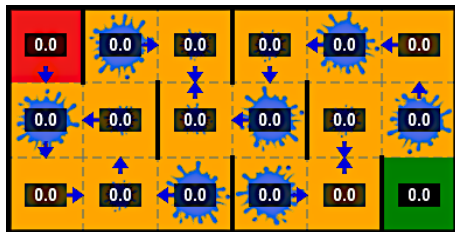
Példa politika iterációra

A kezdeti politika véletlenszerű és determinisztikus.



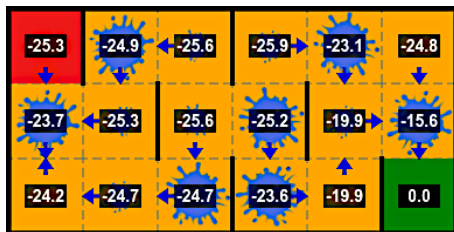
Példa politika iterációra

A $V(s)$ állapot értékek 0 értékkel indulnak.



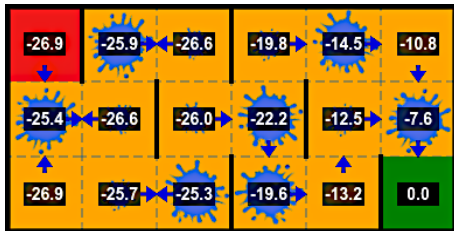
Példa politika iterációra

$\gamma = 0.9$ diszkont rátával a politika kiértékelés 75 iteráció alatt konvergál.



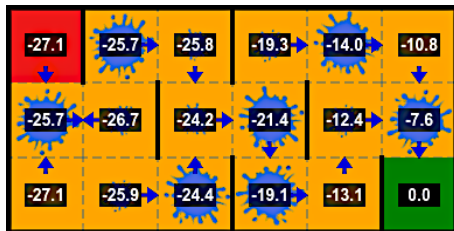
Példa politika iterációra

A következő iterációban a politika változik. A konvergálás 55 iteráció alatt megtörtént.



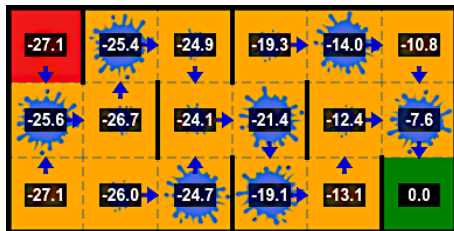
Példa politika iterációra

A következő futtatással 26 iteráció alatt konvergált.



Példa politika iterációra

A következő futtatással 21 iteráció alatt konvergált.



Példa politika iterációra

A következő futtatással 26 iteráció alatt konvergált.

