

11. Előadás

Megerősített tanulás

Irányelvi hálózatok

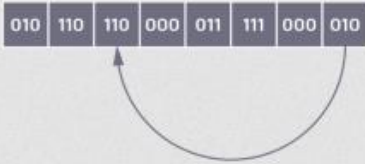
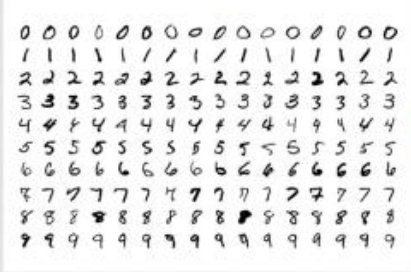
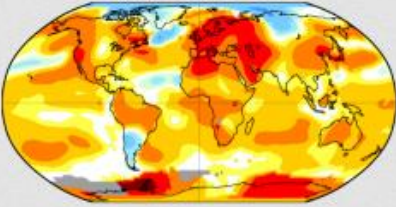

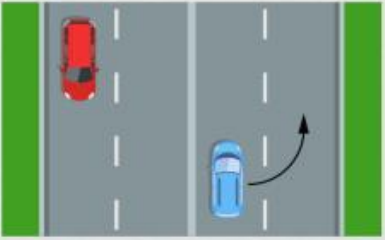

Q-Learning

Mire jó a megerősített tanulás?

🐍 RL-t olyan problémák esetén használunk, amikor az algoritmikus hozzáállás már kevés.

🐍 Ugyanakkor nincs elég adat/nem lehet elég adatot generálni ahhoz, hogy egy felügyelt tanulási modellt tanítsunk.

🐍 Pl. Starcraft játék, önvezető autó stb...



Example problems which require different kinds of training signal			
Training Signal	Algorithmic	Human	Beyond Human
Supervised Learning	Learning Data Structures 	Image Classification 	Long-term Prediction 
Reinforcement Learning	Playing Games 	Driving "Well" 	Designing Transit System 

Felügyelt vagy megerősített?

- 🐍 Vegyünk egy autóversenyző programot: a példában a TM-Nations.
- 🐍 Ha felügyelt tanítással szeretnénk a programot tanítani, kell egy adatbázis, amiben van rekord minden eshetőségről: balra, jobbra kanyar, gáz, fék stb...
- 🐍 Megerősített tanulás esetén viszont **nem adunk az algoritmusnak priori tudást**, hanem véletlenszerűen inicializáljuk a paramétereket. Modellek **generációi** alatt azokat a paramétereket tartjuk meg, amelyek jobban teljesítik a feladatot.



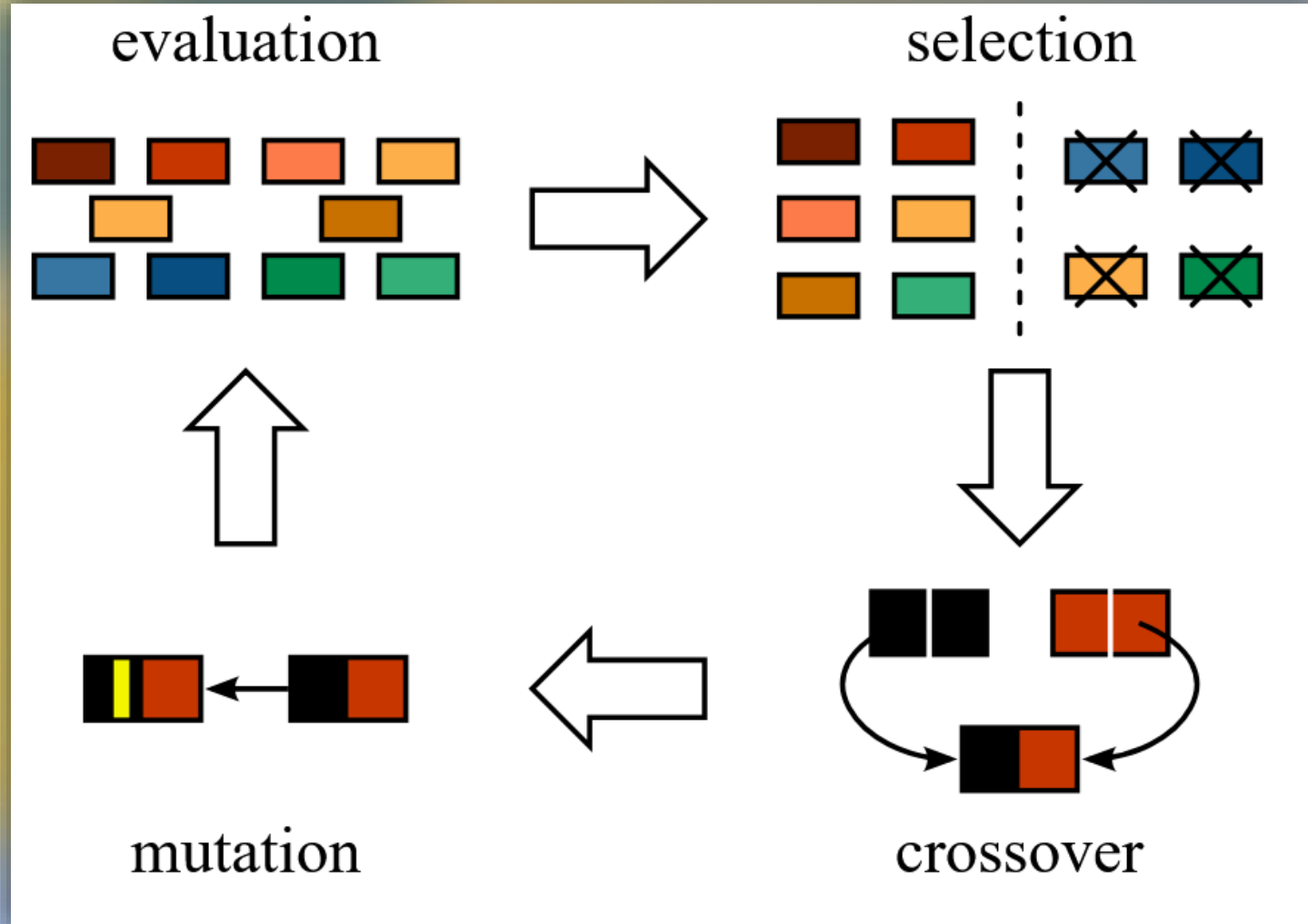
Visszajelzések

-  A két szemléletmód ott különbözik, hogy a felügyelő milyen visszajelzéseket ad a tanulóknak. A felügyelt tanulásban **teljes visszajelzésekről** beszélünk, mert a válasz önmagában a teljes jó megoldás.
-  A megerősített tanulásban viszont csak **részleges visszajelzések** vannak: a felügyelő válasza mindig csak a megoldás irányába vezet, nem pedig a megoldás.



Megerősített tanulás alap elgondolása

- A genetikus algoritmus a **populáció** genetikus struktúráját és viselkedését modellezi. A populációra lehet úgy gondolni, mint modellek összessége.
- Minden egyed egy lehetséges megoldási stratégiát modellez.
- A populáció összes egyede közül a legjobbakból készülnek az utódok (másolatok).
- Az utódok **mutáció** segítségével a szülők legjobb tulajdonságait öröklik.
- A mutáció egy apró változás a gének struktúrájában.



Jutalmak optimalizálása

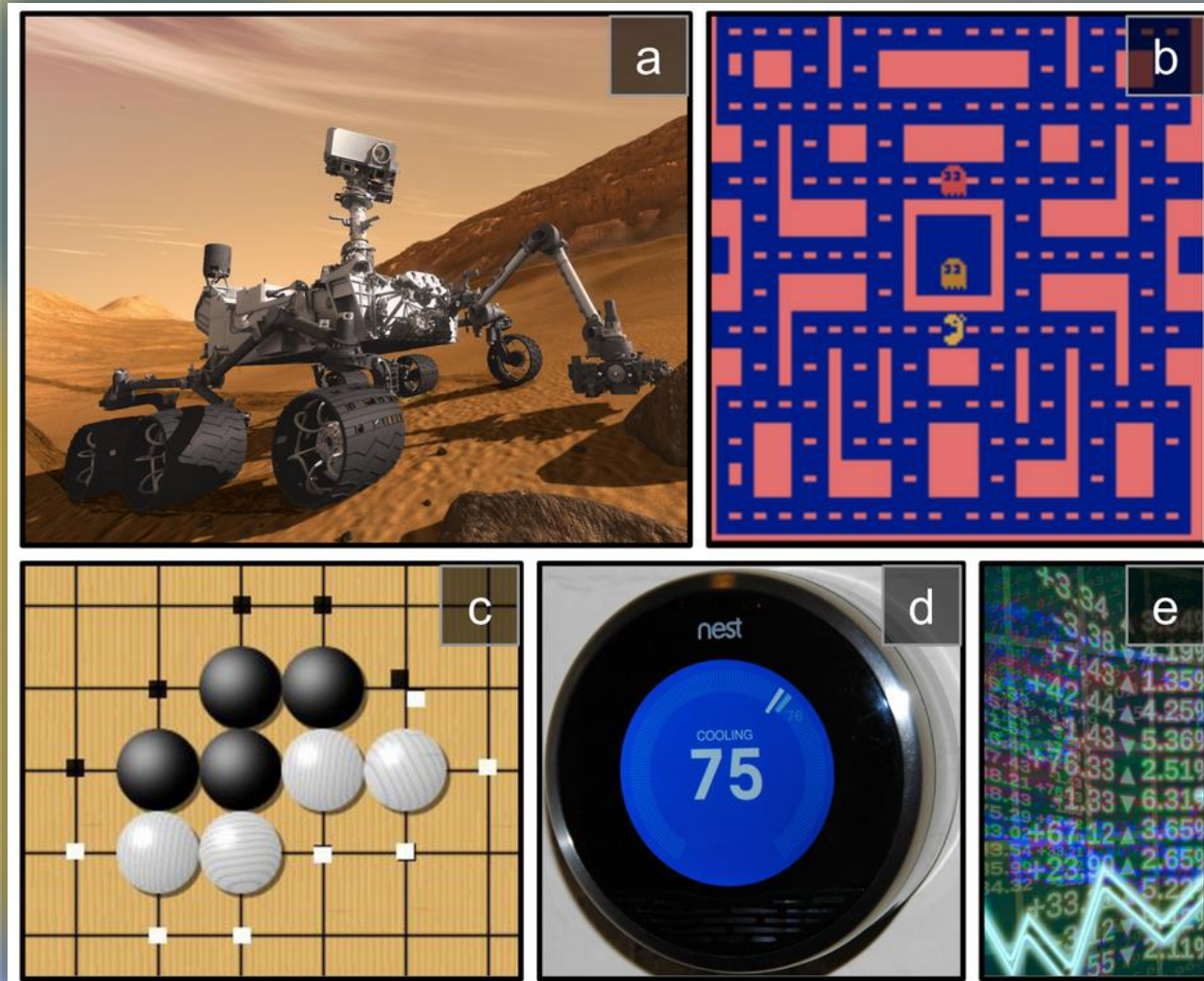
🐍 A megerősített tanulásban egy **ágens (cselekvő)** megfigyeléseket tesz valamilyen környezetben, és a cselekvéseiért cserébe **jutalmakat** kap.

🐍 Az ágens célja, hogy maximalizálja a várható jutalmait adott időn belül.

🐍 A jutalmak lehetnek pozitívak, illetve negatívak is.

🐍 Az ágens lehet:

- a) Az a program, ami egy robotot irányít.
- b) A program, ami Ms. Pac-Man-t irányítja.
- c) Egy go-t játszó program.
- d) Nem muszáj fizikailag irányítania valamit: lehet akár egy termosztát is.
- e) Kereskedő a tőzsdén.

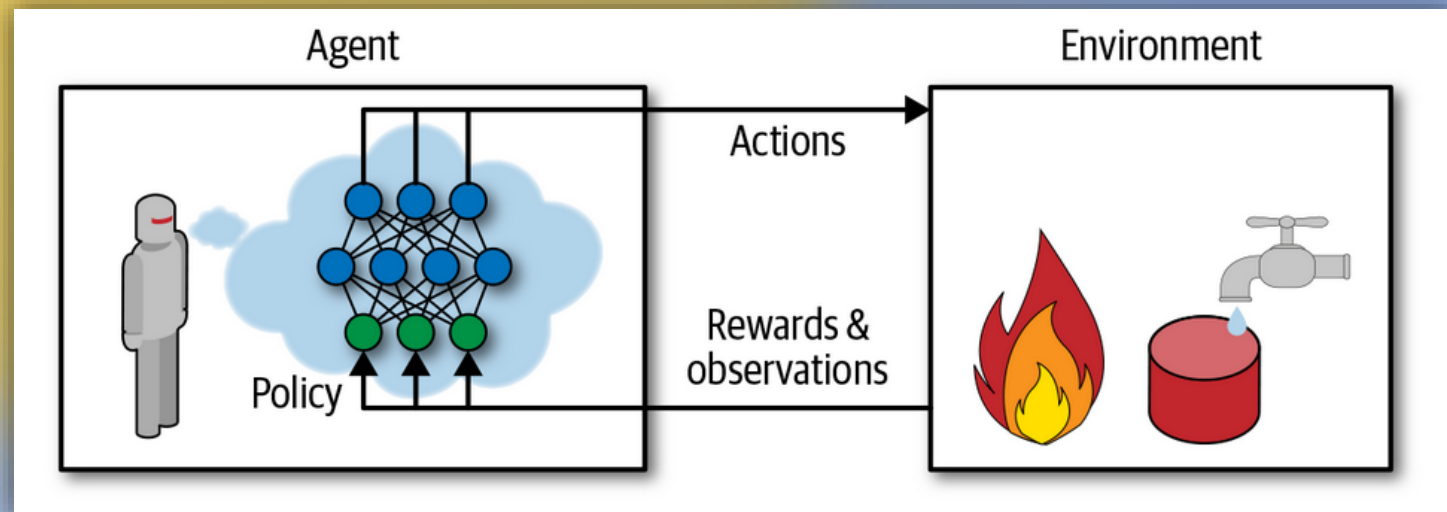
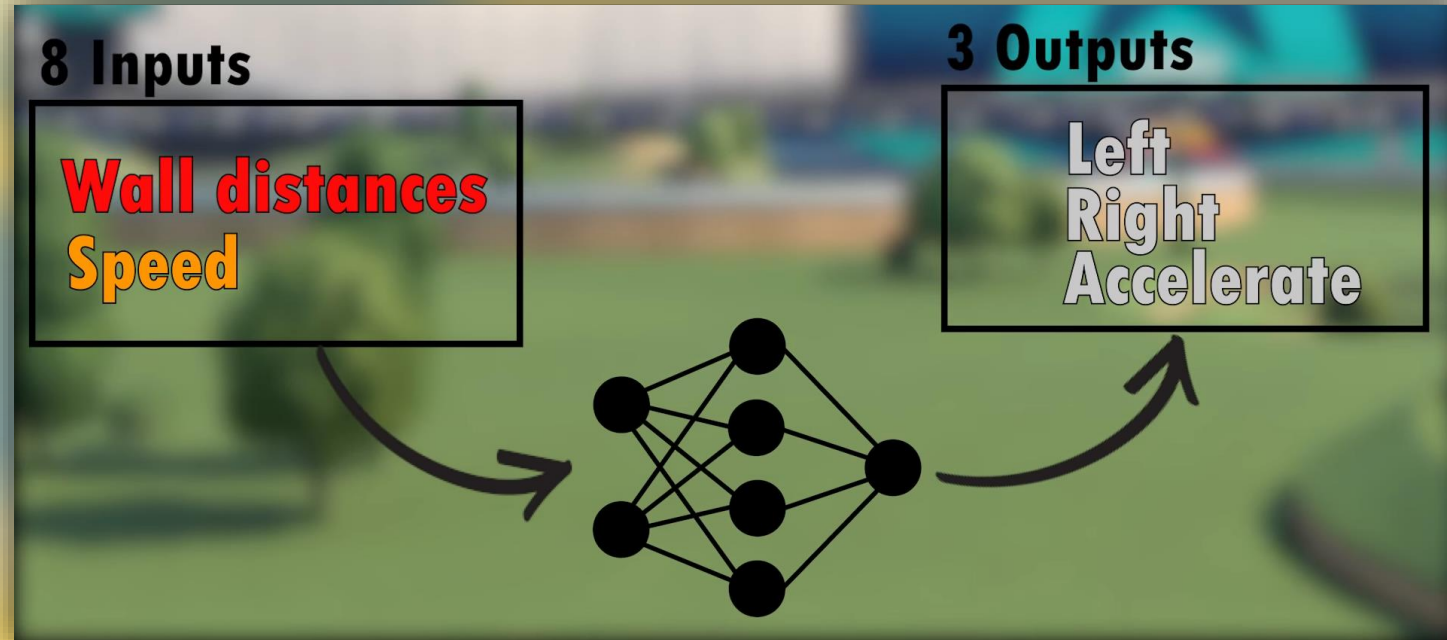


Irányelvek (policy)

🐍 Az az algoritmus, amit az ágens alkalmaz a cselekvéseinek meghatározására, az **irányelve**.

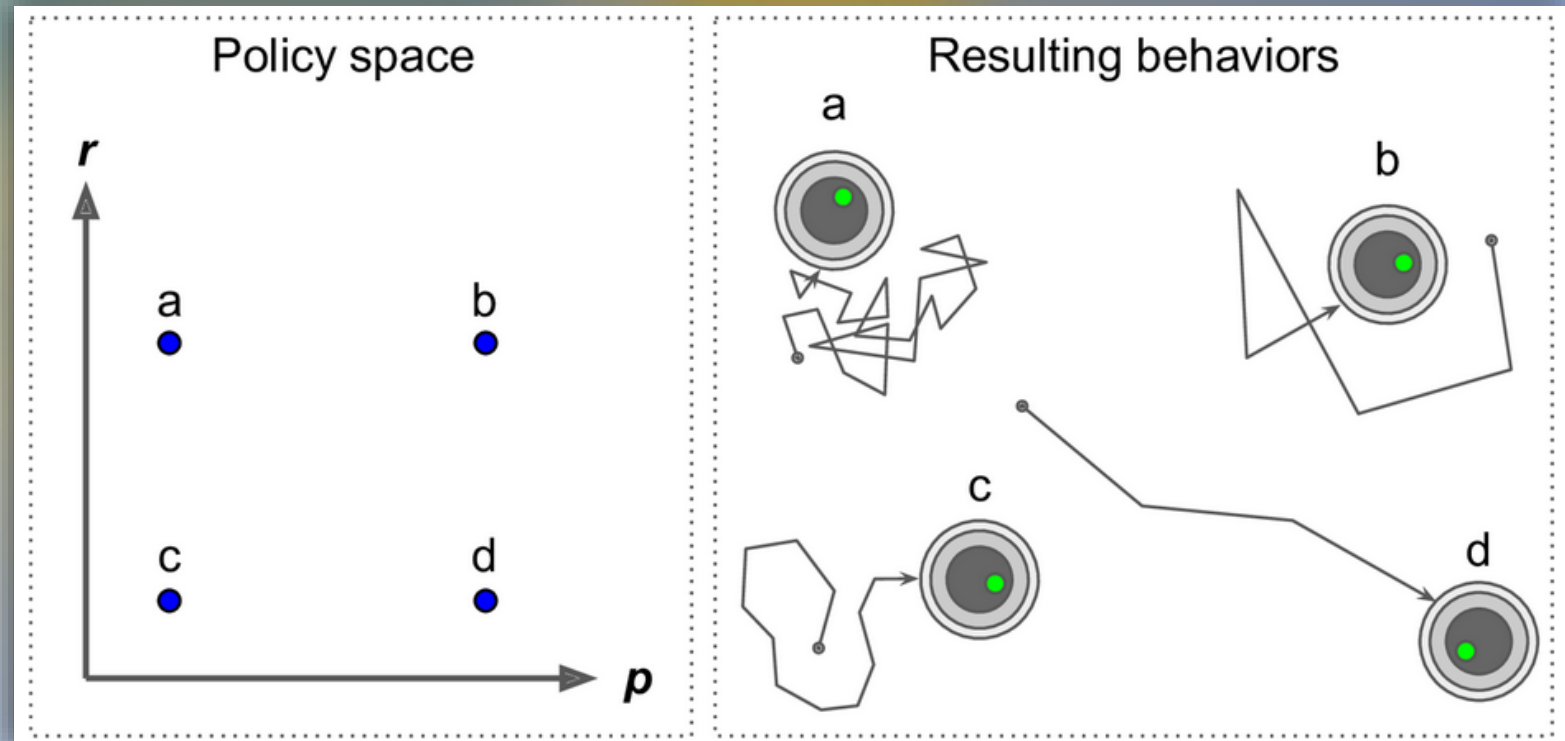
🐍 Gondolhatunk rá úgy, mint egy modell, amely a környezetet leíró változókat fogadja bemeneti adatként, és ezek alapján valamilyen cselekvés lesz az outputja, amit el is végez.

🐍 Az irányelv lehet bármilyen algoritmus amit el tudunk képzelni. Sőt, nem is kell determinisztikusnak lennie!



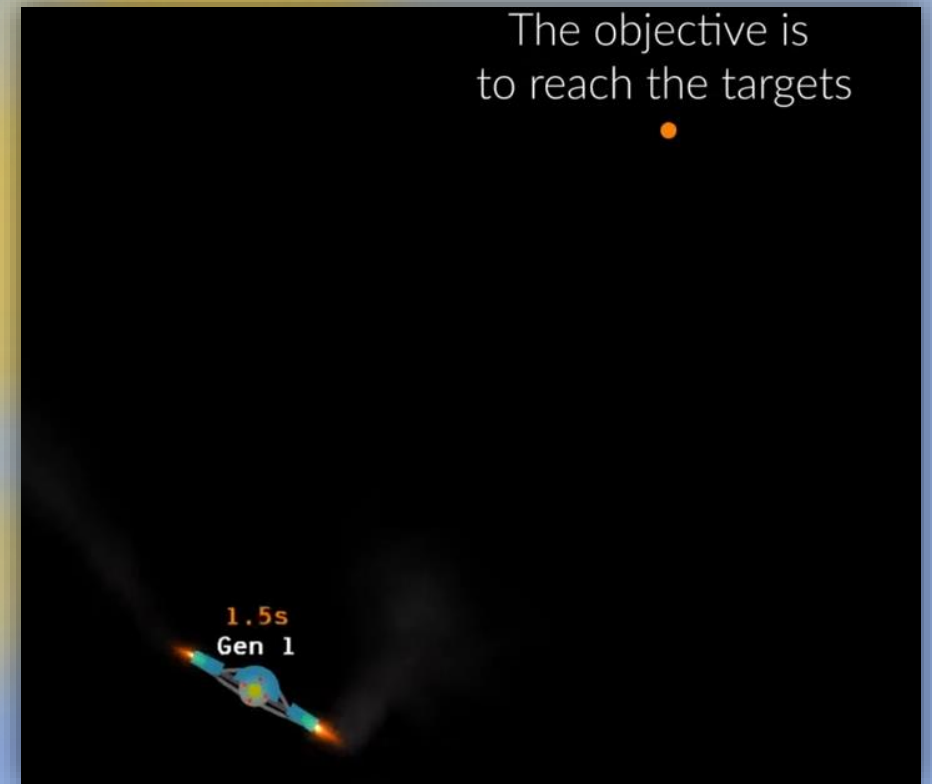
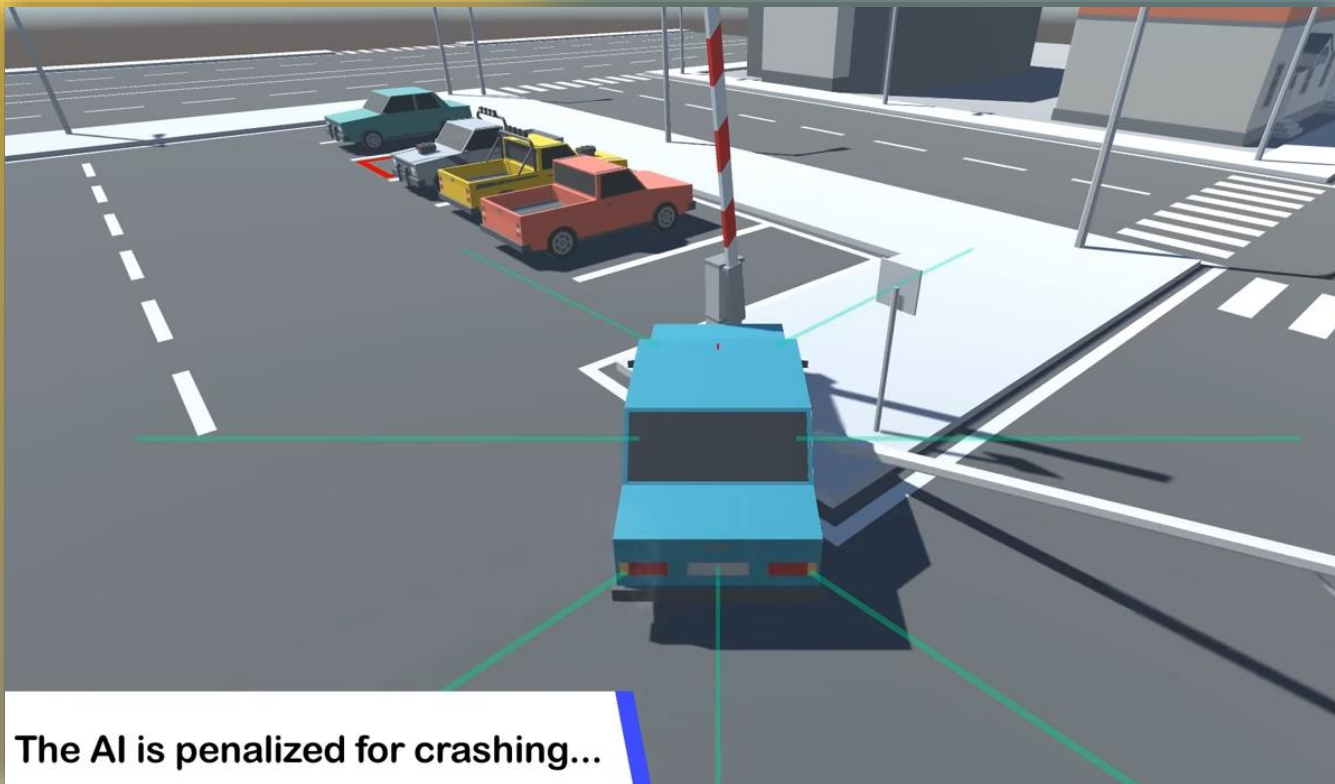
Írányelvek javítása

- 🐍 Egy hozzáállás a kereséshez az lenne, hogy kipróbálunk sok irányelv kombinációt, majd megtartjuk a legjobbat. Ez az **írányelv keresés**: egy nagyobb térben (ami általában elmondható) olyan, mint tűt keresni a szénakazalban.
- 🐍 Egy másik hozzáállás a **genetikus algoritmus**: véletlenszerűen csinál pl. 100 irányelvet, kipróbálja, majd a 80 legrosszabbat „megöli”.
- 🐍 A legjobb 20 irányelv mindegyike 4 utódhoz létre, és az iteráció folytatódik.
- 🐍 **Utód**: a szülő másolata, plusz egy kevés random variáció.



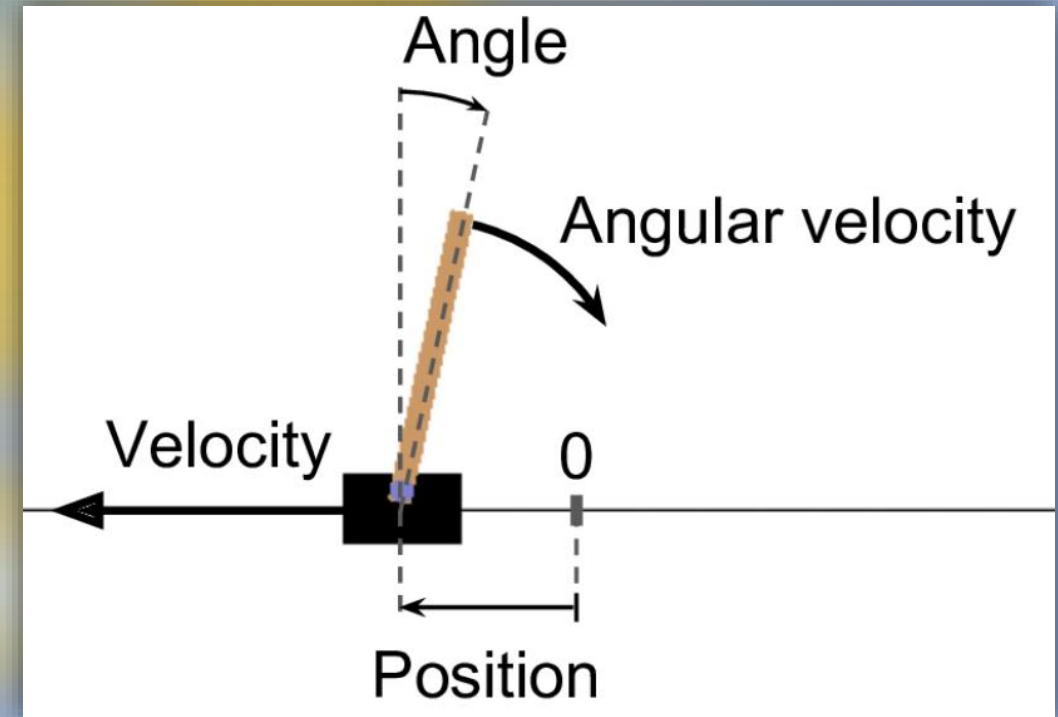
Epizódok

- 🐍 A megerősített tanulásban egy tanítási iterációt **epizód**nak nevezünk:
- 🐍 Az a folyamat, ami alatt az ágens végleges állapotba nem ér el. Ez lehet a cél teljesítése, a teljes bukás, időkeret lejárása és részfeladat teljesítése is.
- 🐍 Egy generáció jellemzően több epizódot játszik.



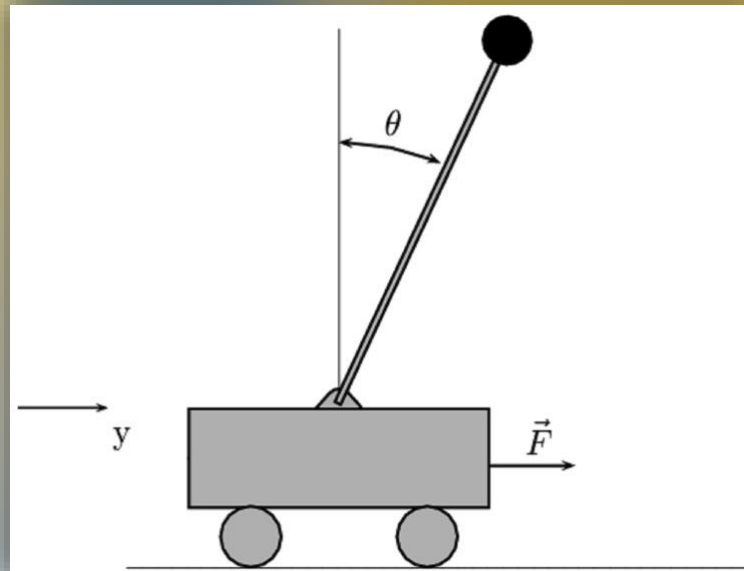
OpenAI gym

- 🐍 Ahhoz, hogy ágenseket lehessen tanítani, szükségünk van egy környezetre. Az OpenAI gym egy olyan eszköztár, ami különböző szimulált környezeteket biztosít.
- 🐍 Például, a CartPole környezet: egy 2D szimuláció, amiben egy kocsit lehet balra és jobbra tologatni annak érdekében, hogy a rajta álló rudat kiegyensúlyozzuk.
- 🐍 Az eszköztár biztosítja a környezet leíró változóit. Ebben az esetben:
 - 🐍 A rúd dőlésszöge (Angle)
 - 🐍 A kocs sebessége (Velocity)
 - 🐍 A rúd eldőlésének sebessége (Angular velocity)
 - 🐍 A kocs helyzete (Position)
- 🐍 Ezek minden időpillanatban teljes képet adnak a környezetről.
- 🐍 Az eszköztár le is tudja renderelni képként.



Hardcode irányelvek

- 🐍 Implementáljunk egy egyszerű irányelvet, ami balra megy, amikor az oszlop balra dől, és jobbra, amikor az oszlop jobbra dől.
- 🐍 Futtassuk 500 epizódon keresztül, hogy megtudjuk, mennyi az átlagos jutalom.
- 🐍 500 próbálkozás alatt sosem sikerült az ágensnek 66 lépésnél tovább megtartania az oszlopot
- 🐍 Ez a fajta explicit szabályrendszer a **hardcode** irányelv.



```
import gym
env = gym.make('CartPole-v1')

def basic_policy(obs): # jutalmak
    angle = obs[2]
    return 0 if angle < 0 else 1 # az oszlop balra vagy jobbra dől?

totals = []
for episode in range(500): # 500 epizód
    episode_rewards = 0 # kezdeti jutalom
    obs = env.reset() # környezet init

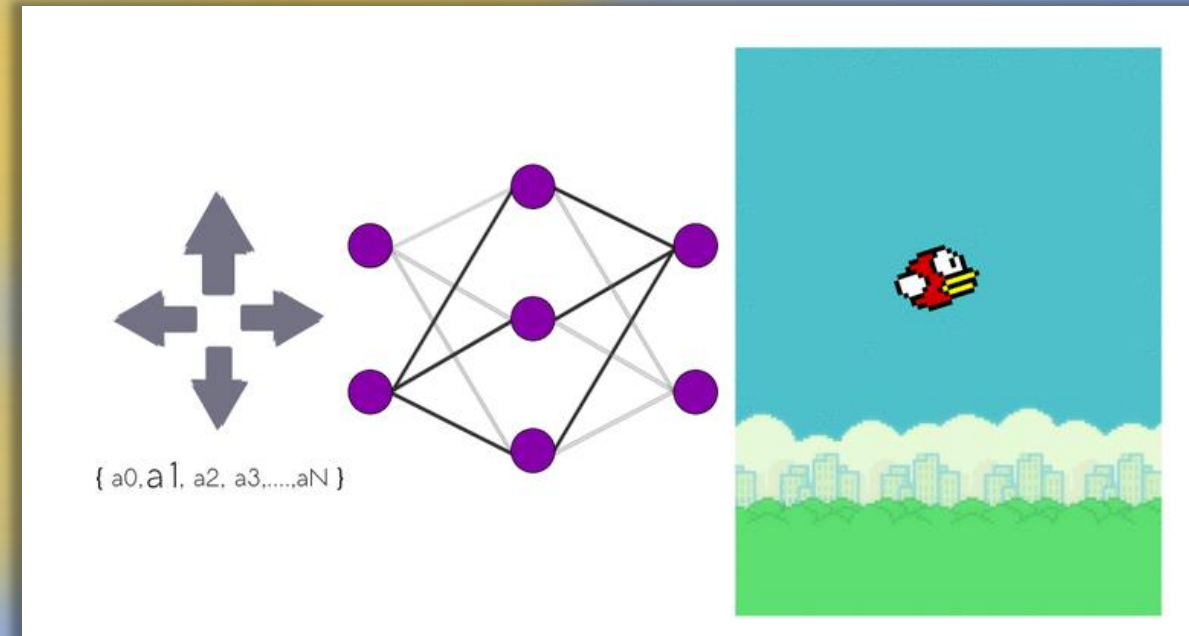
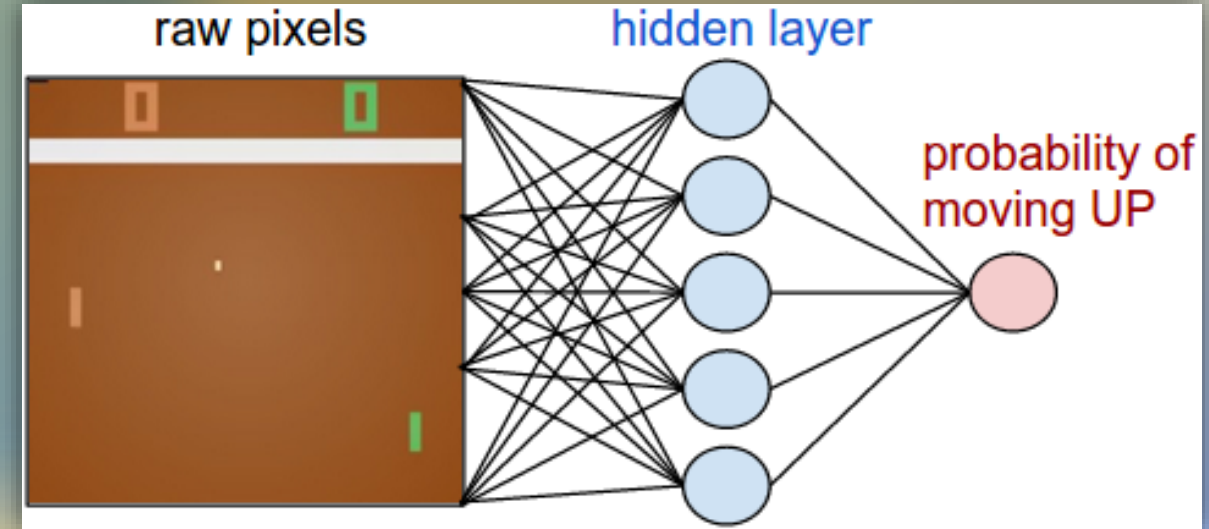
    for step in range(200): # 200 válasz iterációként
        action = basic_policy(obs) # jelenlegi szög
        obs, reward, done, info = env.step(action) # válasz
        episode_rewards += reward # jutalmak mentése
        if done: # kilépés
            break
    totals.append(episode_rewards)

print('átlag', np.mean(totals))
print('szórás', np.std(totals))
print('min', np.min(totals))
print('max', np.max(totals))
env.close()
```

```
átlag 41.6
szórás 8.769264507357502
min 24.0
max 66.0
```

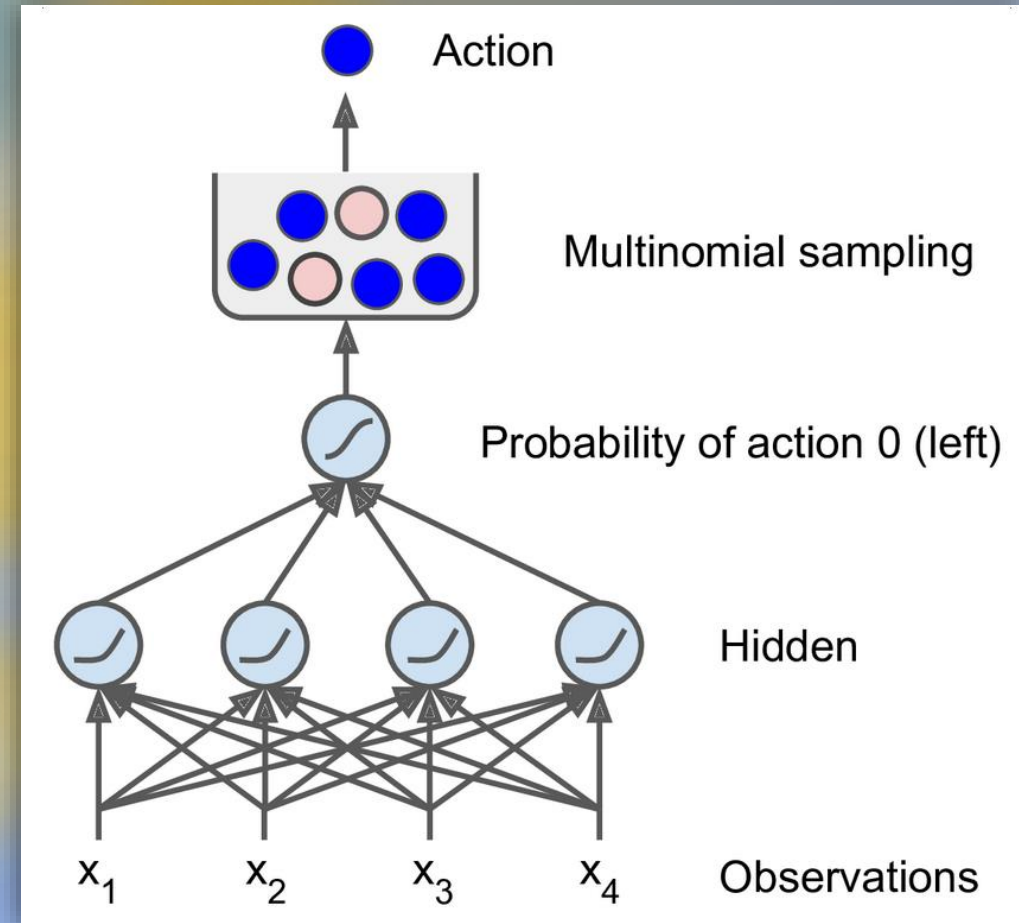

Irányelvi hálózat modell

- 🐍 A cselekvések kézzel való implementálása hosszas, és nem vezet hamar jó eredményre, a változók sokasága miatt.
- 🐍 Az irányelvet definiálhatjuk egy neurális hálózatként (vagy egyéb ML modellként).
- 🐍 A modell bemenete ebben az esetben a környezeti változók vektora, a kimenete pedig a cselekvés, amit az ágens végre fog hajtani.
- 🐍 A tanítás eljárása a neurális hálózat javításaként értelmezhető: a jó háló fejlődjön, a rossz menjen.
- 🐍 Probléma: a háló csak a pillanatnyi változókat ismeri, a múltbelieket nem!



Írányelvi hálózat működése

- A hálózat kimeneti rétegében minden neuron egy cselekvést reprezentál, amihez tartozóan kiszámol egy valószínűséget a környezet állapotától függően.
- A CartPole környezethez tartozóan két osztály lehetséges (balra és jobbra), ezért egy output neuronja lesz a hálózatnak.
- Az output neuron értéke a 0-ás cselekvés (bal) valószínűsége p , így az 1-es cselekvésé $(1 - p)$.
- Ezután a hálózat a kapott valószínűségeket eloszlásként felhasználva mintát vesz egy halmazból, amiben a cselekvések vannak.
- A választott cselekvést fogja az ágens végrehajtani. Ez a hozzáállás tanítja meg az ágenst arra, hogy megtalálja az egyensúlyt a **már működő**, és az **új cselekvések** között.



A kredit hozzárendelési probléma

- 🐍 Ha tudnánk minden lépésnél, hogy melyik a legjobb cselekvés, akkor a neurális hálót úgy lehetne tanítani, mint általában: a kereszt-entrópia minimalizálásával.
- 🐍 Viszont a megerősített tanulásban az egyetlen visszajelzés, amit az ágens kap, a jutalom. Ezek pedig ritkák és késleltetettek. Pl. ha az algoritmus 100 lépésen keresztül egyensúlyozta a rudat, honnan tudjuk melyik cselekvés volt jó, illetve rossz?
- 🐍 Ha a rúd leesik, az utolsó cselekvés a felelős érte?
- 🐍 Ez a kredit hozzárendelési probléma: mikor az ágens jutalmat kap, nehéz tudnia, miért kapott pozitív vagy negatív jutalmat.



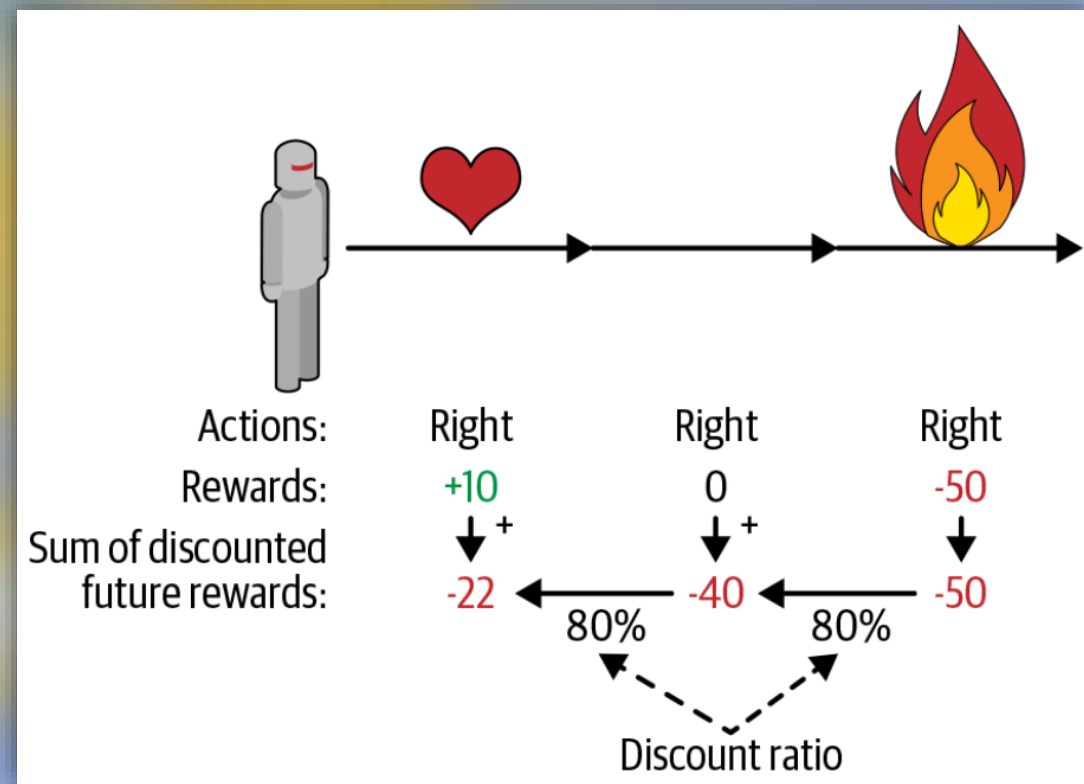
A probléma megoldása

🐍 Egy gyakori stratégia, hogy az utána következő jutalmak összege alapján értékelünk ki egy cselekvést. Minden lépésben valamilyen **diszkont faktort** (γ) alkalmazunk. A diszkontált jutalmak összege a **visszatérés**.

🐍 Példa: ha az ágens háromszor jobbra megy, ezután a három jutalma $[+10, 0, -50]$, és $\gamma = 0.8$ diszkont faktort alkalmazunk, az első cselekvés visszatérése: $10 + 0\gamma + \gamma^2(-50) = -22$.

🐍 Ha a $\gamma = 0$, a modell a jelenbeli jutalmat részesíti előnyben. Ha viszont 1 közeli az értéke, a jövőbeni jutalmak szinte annyira fognak számítani, mint a jelenbeliek.

🐍 A CartPole esetében a döntéseknek rövidtávú következményei vannak, ezért indokolt egy 1-hez közeli γ .



Írányelvi gradiensek (Policy Gradient)

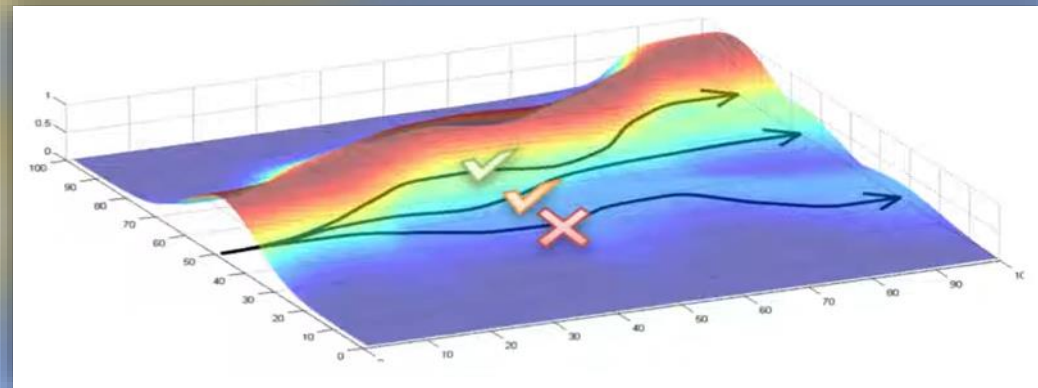
🐍 A PG algoritmusok úgy optimalizálják az irányelvek paramétereit, hogy mindig a magasabb jutalom felé mutató gradienseket követik. Egy gyakori eljárás:

1. Hagyjuk, hogy az irányelv többször játsszon, és minden lépésnél számoljuk ki a gradienseket, viszont ne alkalmazzuk őket.
2. Mikor lefutott több epizód, számoljuk ki mindegyiknek az előnyét (az előzőben leírtak alapján).
3. Ha a cselekvés előnye pozitív, valószínűleg jó cselekvés volt, és alkalmazzuk rá a gradienseket, hogy a jövőben is gyakrabban választódjon ki.

Viszont, ha az előny negatív előjelű, a cselekvés valószínűleg rossz volt, és az ellentétes gradienseket szeretnénk rá alkalmazni, hogy a jövőben kevésbé legyen valószínű.

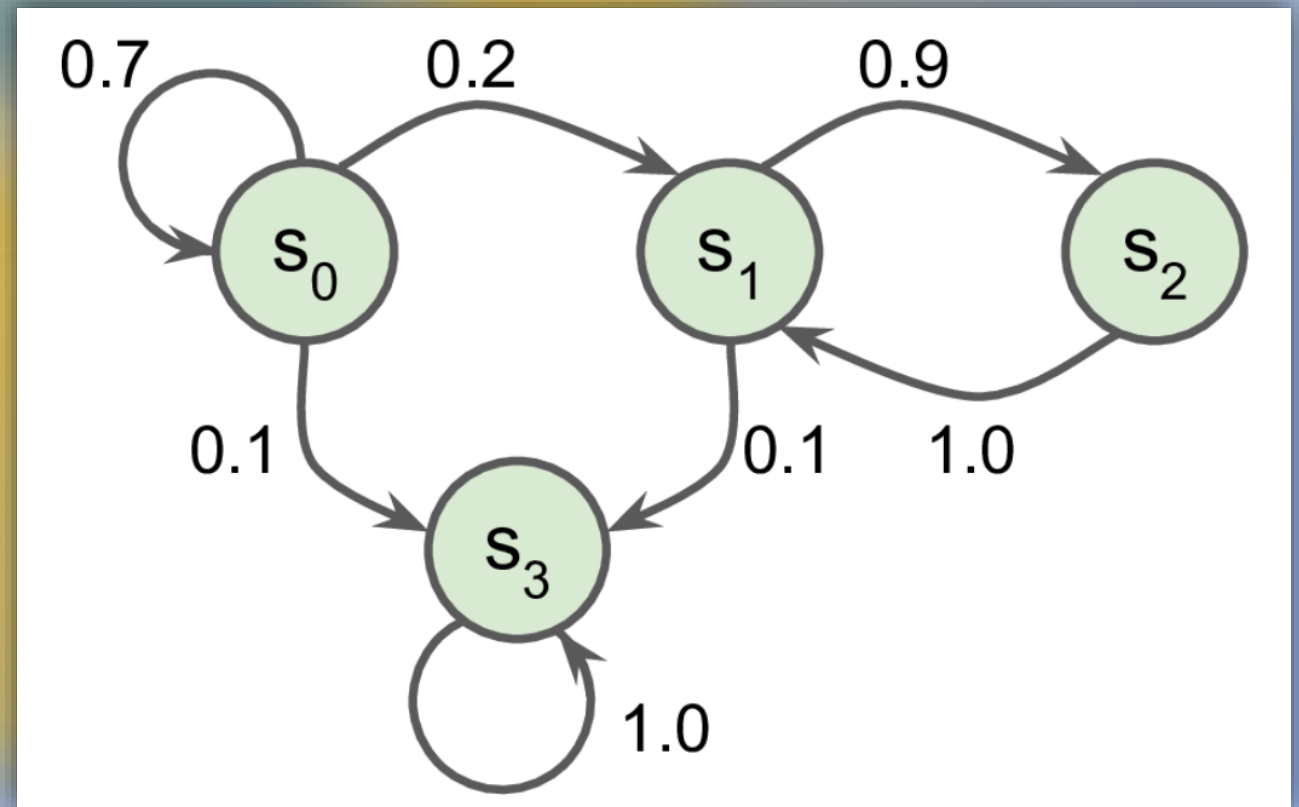
Tehát: minden gradiens vektort a neki megfelelő cselekvés előnyével összeszorozunk.

4. Számoljuk ki az összes, így kapott gradiens vektort, és használjuk fel arra, hogy egy lépés gradiens ereszkedést hajtsunk végre.



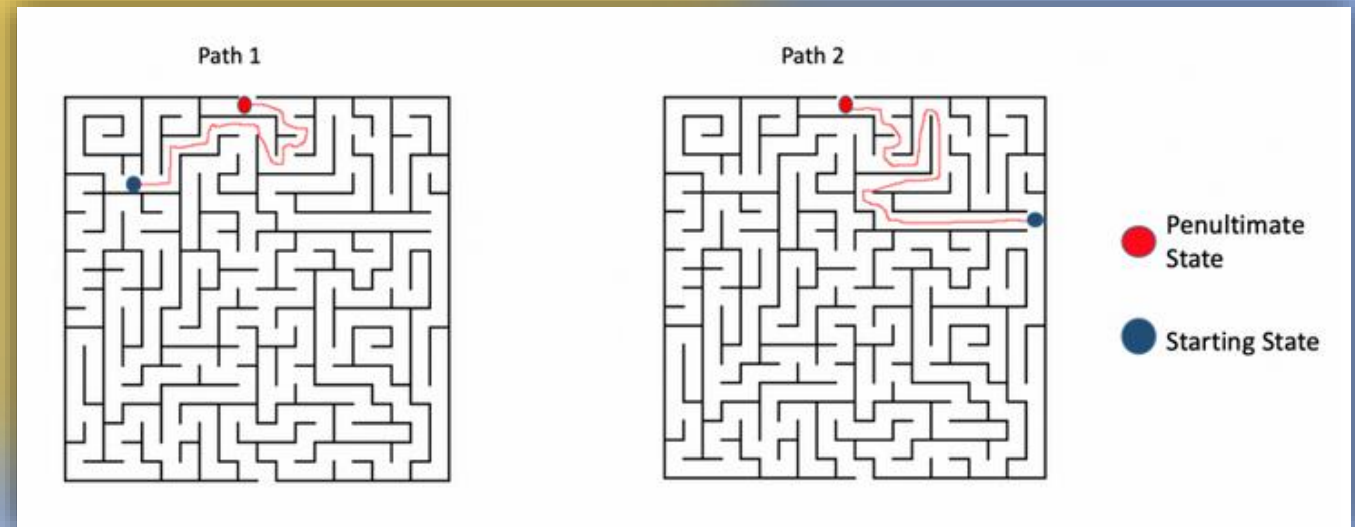
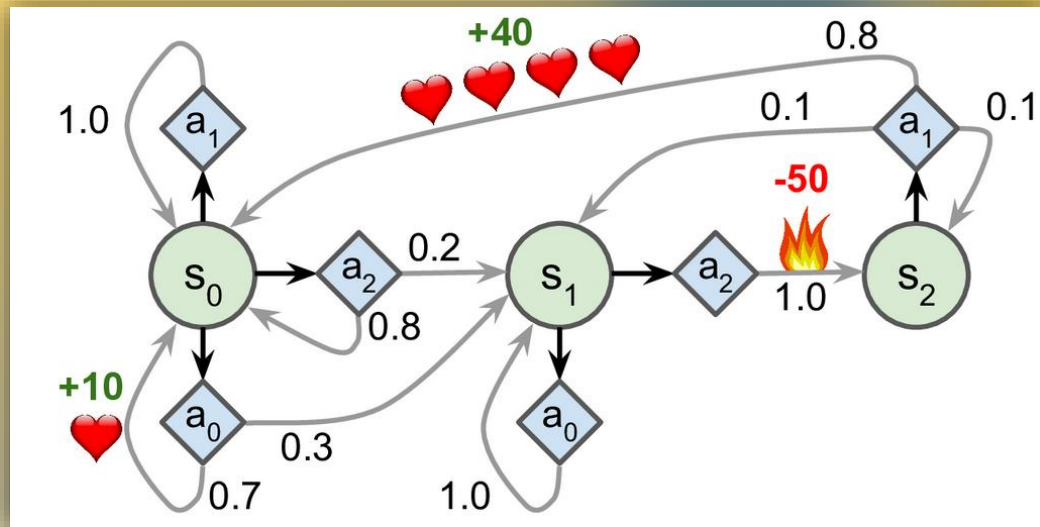
Markov láncok

- A korai 20. században Andrey Markov memória nélküli sztochasztikus folyamatokat tanulmányozott. Ezeket ma Markov láncoknak nevezzük.
- Egy ilyen folyamatnak fix számosságú állapota van, és minden lépésben véletlenszerűen fejlődik tovább a következő állapotba.
- Annak a valószínűsége, hogy s állapotból s' -be fejlődjön fix, és nem múlik korábbi állapot változtatásokon.
- A folyamat valamelyik állapotból elindul, illetve van terminális állapota (s_3 ebben az esetben).



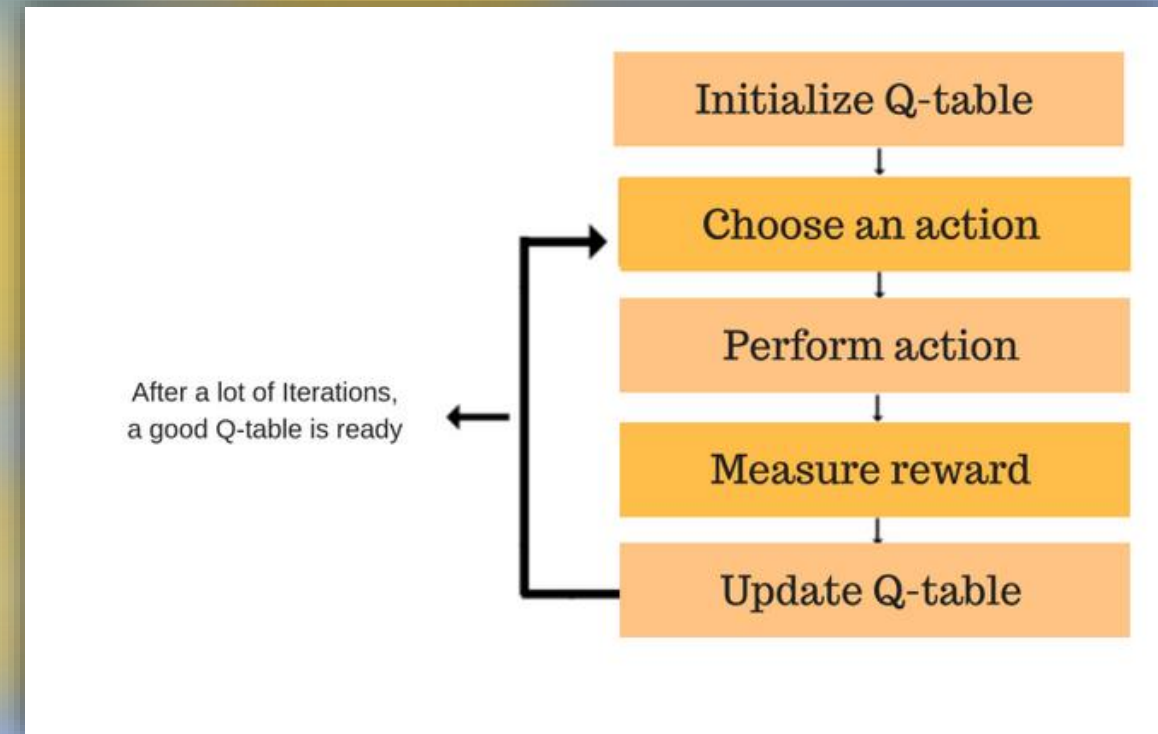
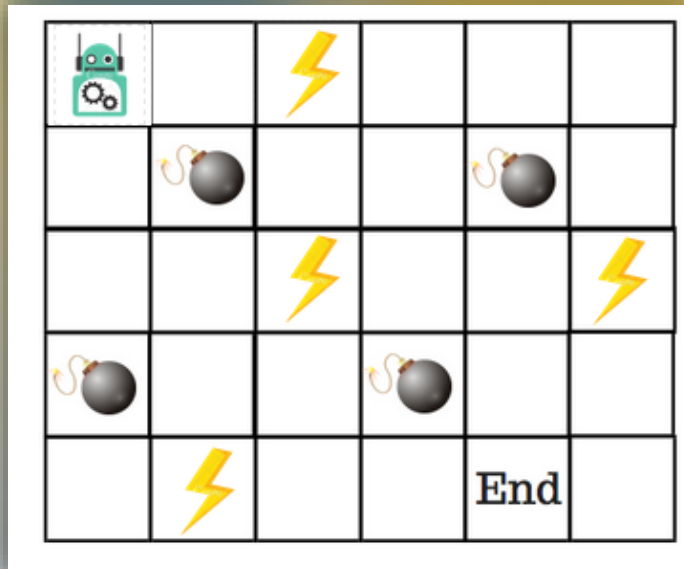
Markov döntési folyamatok

- 🐍 Először az 1950-es években, Richard Bellmann írta le őket. Lényegében a Markov láncokat implementálja a megerősített tanulás kontextusában.
- 🐍 Minden lépésben az ágens több lehetséges cselekvés közül választhat, és az állapotok közötti átmeneti valószínűségek a cselekvéstől függenek.
- 🐍 Az állapot átmenetek (pozitív, negatív vagy nulla) jutalmakat térítenek vissza, és a cél megtalálni azt az irányelvet, ami maximalizálja a jutalmakat.
♥: pozitív jutalom, 🔥: negatív jutalom, ●: állapot, ◇: cselekvés



Q-learning

- 🐍 A Q-learning nem egy irányelv, hanem egy **érték alapú** megerősített tanulás megközelítés. Az érték alapú algoritmusok valamilyen függvény szerint frissítik a lehetséges lépések valószínűségét.
- 🐍 A Q-learning egy **irányelven kívüli** (off-policy) tanulási eljárás: megtalálja az optimális irányelv értékeit az ágens cselekvéseitől függetlenül.
- 🐍 Az algoritmus a hosszú távon jól teljesítő szabályokat részesíti előnyben.
- 🐍 Példa: jusson el a robot a célhoz anélkül, hogy bombára lépne.



A Q-Tábla

🐍 Egy olyan keresési táblázat, amiben minden cselekvéshez és állapothoz tartozóan meg van adva a jövőbeli elvárt jutalom maximuma ($Q(s, a)$).

🐍 A Q-táblát a tanítás folyamata alatt iteratívan javítjuk.
De hogyan számolódnak ki a tábla értékei?

$$Q(s_t, a_t) = \underline{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s_t, a_t]$$

Cselekvéshez tartozó
Q-érték, ha adott egy
állapot

Várható diszkontált kumulatív jutalom

Ha adott egy állapot
és cselekvés

🐍 A Q-tábla inicializálása 0 értékekkel:

🐍 A robot esetében 4 cselekvés és
5 állapot létezik, ezért 4×5 -ös
Q-táblánk lesz.

		⚡	UP ↑		
	💣	LEFT ←	🤖	RIGHT →	💣
		⚡	Down ↓		⚡
💣			💣		
	⚡			End	

	Actions :	↑	→	↓	←
Start		0	0	0	0
Nothing / Blank		0	0	0	0
Power		0	0	0	0
Mines		0	0	0	0
END		0	0	0	0

	Actions :	↑	→	↓	←
Start					
Nothing / Blank					
Power					
Mines					
END					

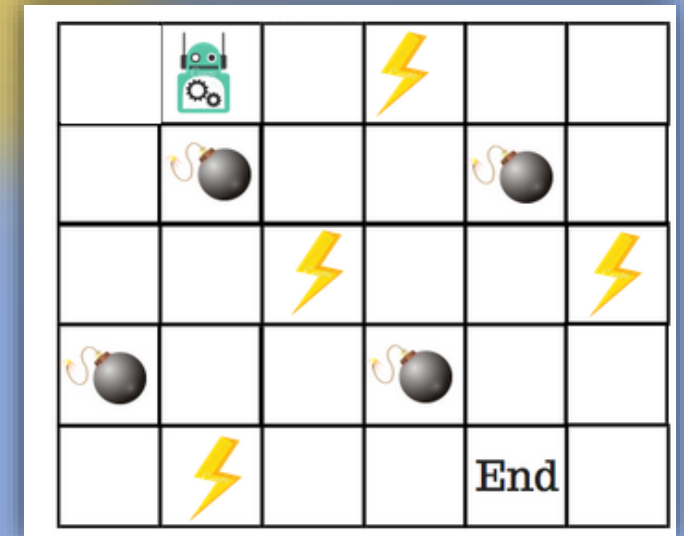
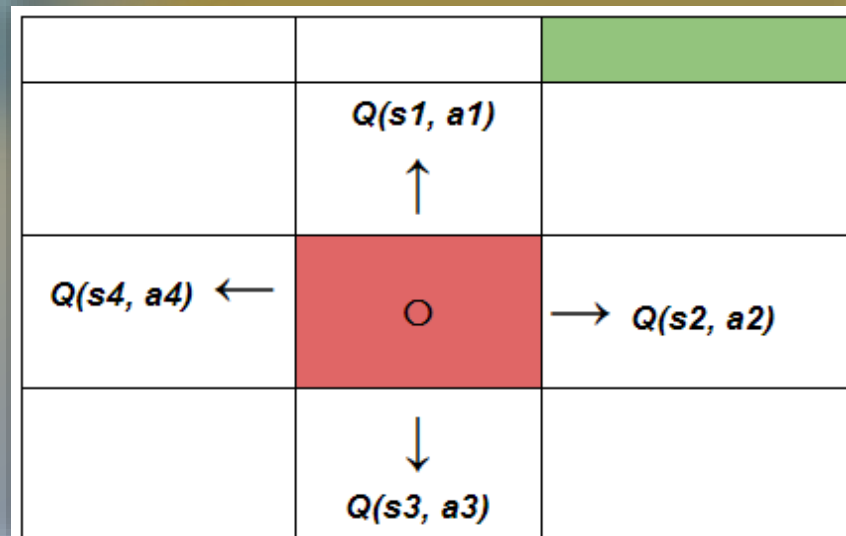
Cselekvés választás és végrehajtás

A lépéseknek ezen kombinációja akármeddig futhat: a programozónak kell gondoskodnia a ciklus lezárásról. Itt jön be az **exploration-exploitation tradeoff**.

Epszilon-mohó stratégia:

- Kezdetben az ε magasabb: az ágens felfedezi a környezetet, és random választ cselekvést, mert nem tud róla semmit.
- Ahogy az ágens megismeri a környezetet, az ε csökken, hogy a robot elkezdje kihasználni a környezetét.
- A megismerés folyamata közben az ágens egyre magabiztosabb lesz a Q-értékek megbecslésében.

Ezután a Q-értékek frissítésre kerülnek.



A Bellman-szabály

Valamely cselekvés után a $Q(s, a)$ függvényt frissíteni kell:

$$\text{New } Q(s, a) = Q(s, a) + \alpha [R(s, a) + \gamma \times \max Q'(s', a') - Q(s, a)]$$

új Q érték az állapotért és cselekvésért

jelenlegi Q érték

tan. seb.

jutalom a cselekvésért

diszkont ráta

a várható maximum jutalom az új állapot (s') és az új állapotban lévő összes lehetséges cselekvésre

jelenlegi Q érték

Valamely cselekvés után a $Q(s, a)$ függvényt frissíteni kell. Ha a jutalmak:

Villám = +1

Bomba = -100

Cél = +100

	$\gamma^4 T$	$\gamma^3 T$	$\gamma^2 T$	γT	T
$t=0$	0	0	0	0	1
$t=1$	0	0	0	0.8	1
$t=2$	0	0	0.64	0.8	1
$t=3$	0	0.512	0.64	0.8	1
$t=4$	0.4	0.512	0.64	0.8	1

	Actions : ↑ → ↓ ←			
Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0

$$\text{New } Q(\text{start}, \text{jobb}) = Q(\text{start}, \text{jobb}) + [\text{value}]$$

$$[\text{value}] = R(\text{start}, \text{jobb}) + \max(Q'(\text{semmi}, \text{le}), Q'(\text{semmi}, \text{bal}), Q'(\text{semmi}, \text{jobb})) - Q(\text{start}, \text{jobb})$$

$$[\text{value}] = 0 + 0.9 \times 0 - 0 = 0$$

$$\text{New } Q(\text{start}, \text{jobb}) = 0 + 0.1 \times 0 = 0$$

Példák

- [!\[\]\(6302aad5aed157b291fddf37b4870784_img.jpg\) Genetikus algoritmus drónok reptetésére](#)
- [!\[\]\(a9ca2c237943a6d0a9f22252f295b6f3_img.jpg\) PPO Neurális hálózat kocsit parkol](#)
- [!\[\]\(9a01a64e0b4ff865df7d32ee7991fe8b_img.jpg\) Bújócskázás több ágenssel](#)
- [!\[\]\(6aefe9a3d997eb8b55c40ecd5fa7053f_img.jpg\) Autóversenyző genetikus algoritmus](#)

