A mesterséges intelligencia alapjai

kétszemélyes játékok

Áttekintés

- játékok
- tökéletes játszás
  - minimax döntés
  - alfa-béta nyesés
- korlátozott erőforrások és közelítő értékelés
- véletlent tartalmazó játékok
- hiányos információjú játékok

Játékok kontra keresési problémák

- „kiismerhetetlen” ellenfél ⇒ a megoldás egy **stratégia**
  - meg kell határozni a válaszlépést az ellenfél minden lehetséges lépésére
- időkorlát ⇒ a célkereséstől eltérően közelítő megoldásra van szükség
- a támadás terve
  - a gép figyelembe veszi a lehetséges játékmeneteket (Babbage, 1846)
  - algoritmus a tökéletes játszásra (Zermelo 1912, Neumann J. 1944)
  - véges horizont, közelítő kiértékelés (Zuse 1945, Wiener 1948, Shannon 1950)
  - első sakkprogram (Turing 1951)
  - gépi tanulás a kiértékelés pontosságának növelésére (Samuel 1952-57)
  - nyesés a mélyebb keresés érdekében (McCarthy 1956)

Játékok típusai

	determinisztikus	nemdeterminisztikus
teljes információjú	sakk, dáma, go, othello	ostábla, monopoly
nem teljes információjú	torpedó, vak amőba	bridzs, póker, scrabble

Grundy játék

- A Grundy játékban kezdetben van egy pénzoszlopunk.
- A soron következő játékos kiválaszt egy oszlopot és két (különböző méretű) részre osztja.
- Ha ezt nem teheti meg – mert minden oszlop 1 vagy 2 magas – a játékos vesztett.

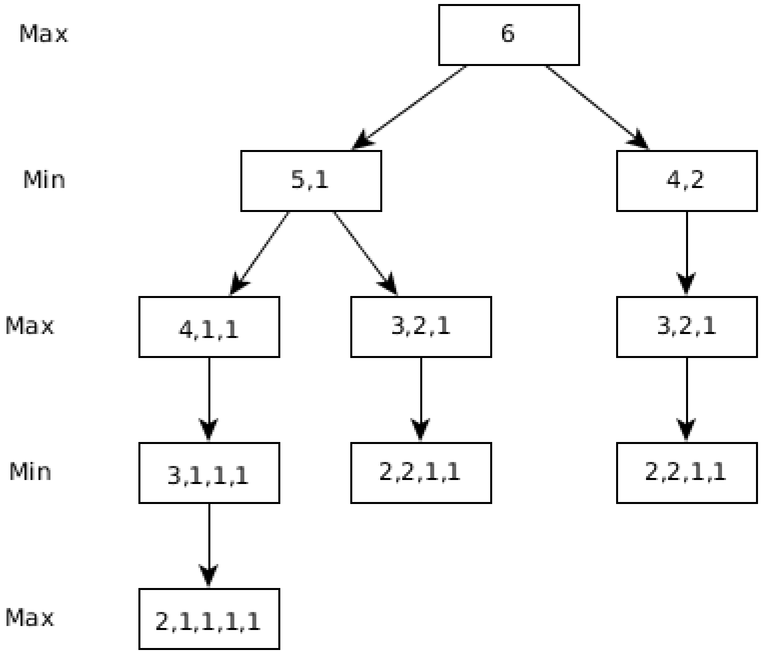
Kezdjünk egy 6 méretű oszloppal! Hogyan kell lépni, ha győzni akar?

- A 6 méretű oszlop a következőképpen osztható fel: 5-1, 4-2, ~~3-3~~, 2-4, 1-5
- Természetesen a sorrend nem számít, így pl. az 1-5 és az 5-1 egyesítve kezelhető.

Grundy játék játékfája

A kezdő – illetve soron következő – játékosra gyakran utalunk A vagy Max névvel, míg ellenfele B vagy Min nevet kap.

A fa levelei jelentik, hogy vége a játéknak. A bal levélnél Max nem tud lépni, így ő vesztett.

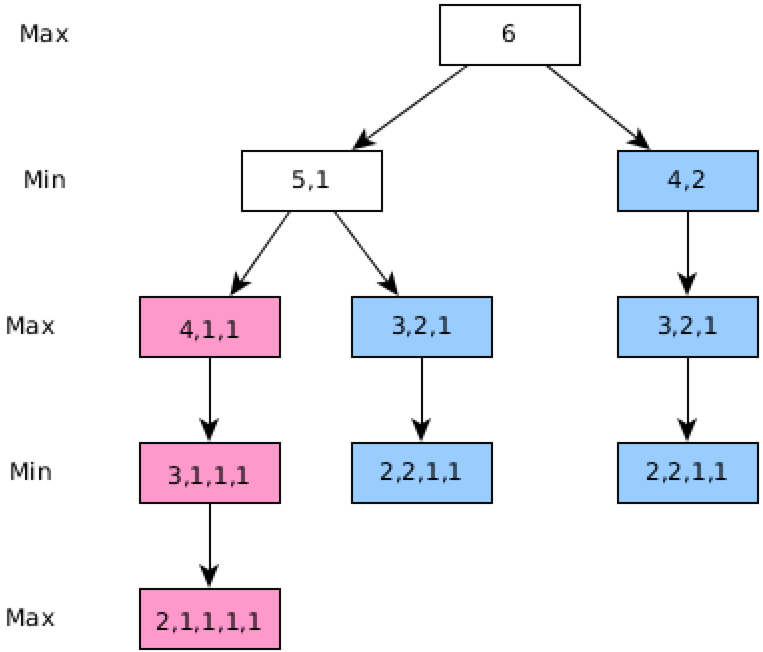


Grundy játékfa címkézése

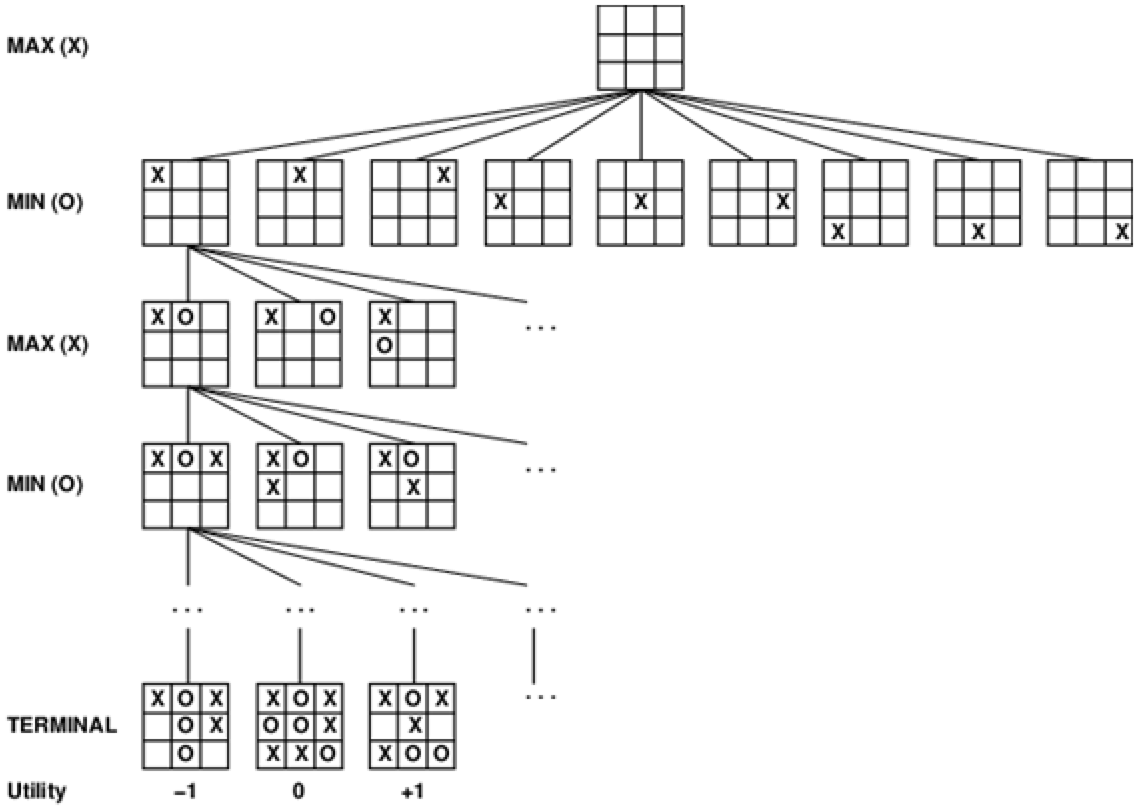
A levelet aszerint címkézzük, hogy

- a kezdő játékos nyer (kék szín, +1, N)
- vagy veszít (piros, -1, V)

Ez a címkézés terjeszthető a gyökér irányába.

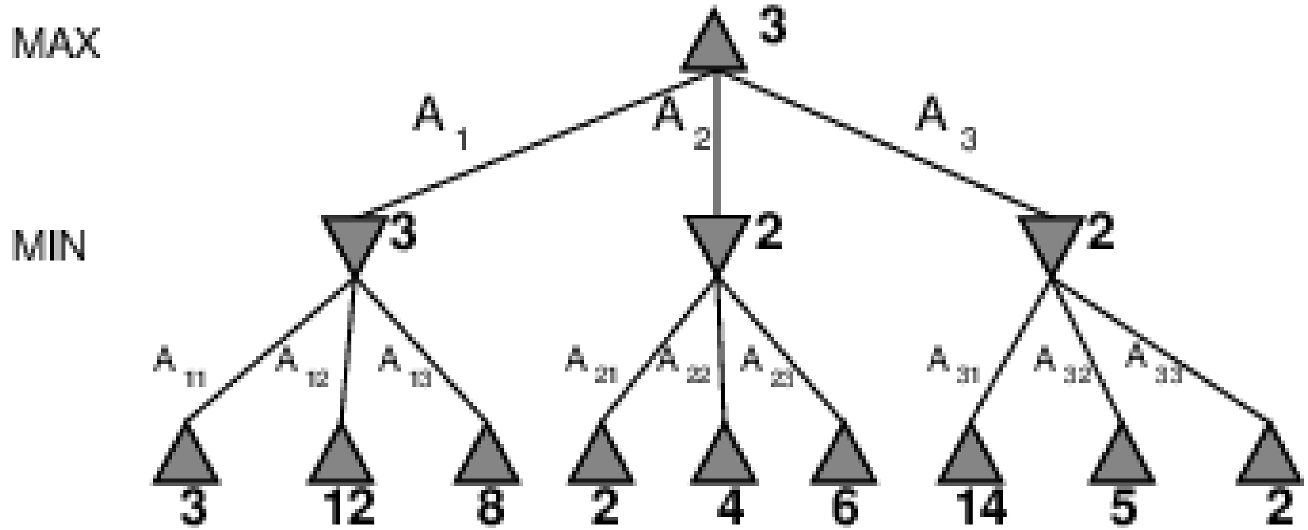


Játékfa (kétszemélyes, determinisztikus, lépésenkénti szintek)



Minimax

- teljes információjú, determinisztikus játék tökéletes játszása
- ötlet: lépjen oda, ahol a legmagasabb a **mimimax érték**
    - legjobb elérhető végeredmény a legjobb ellenfél ellen
- Például kétszemélyes játék esetén:



Minimax algoritmus

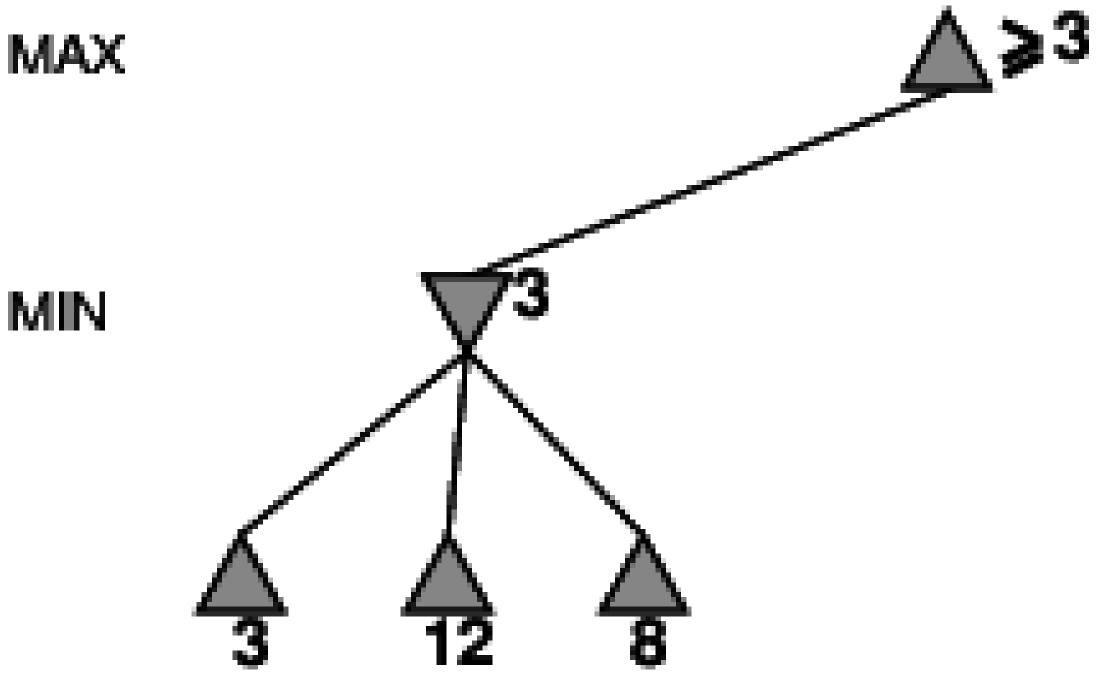
```
function Minimax-Decision(state): művelet  (state: aktuális állapot a játékban)
        return Actions(state) azon „a” elemét, mely maximálja Min-Value(Result(a, state))

function Max-Value(state): hasznosságérték
        if Terminal-Test(state) then return Utility(state)
        v := -∞
        for (a, s) in Successors(state) do
                v := Max(v, Min-Value(s))
        return v

function Min-Value(state): hasznosságérték
        if Terminal-Test(state) then return Utility(state)
        v := ∞
        for (a, s) in Successors(state) do
                v := Min(v, Max-Value(s))
        return v
```

A minimax tulajdonságai

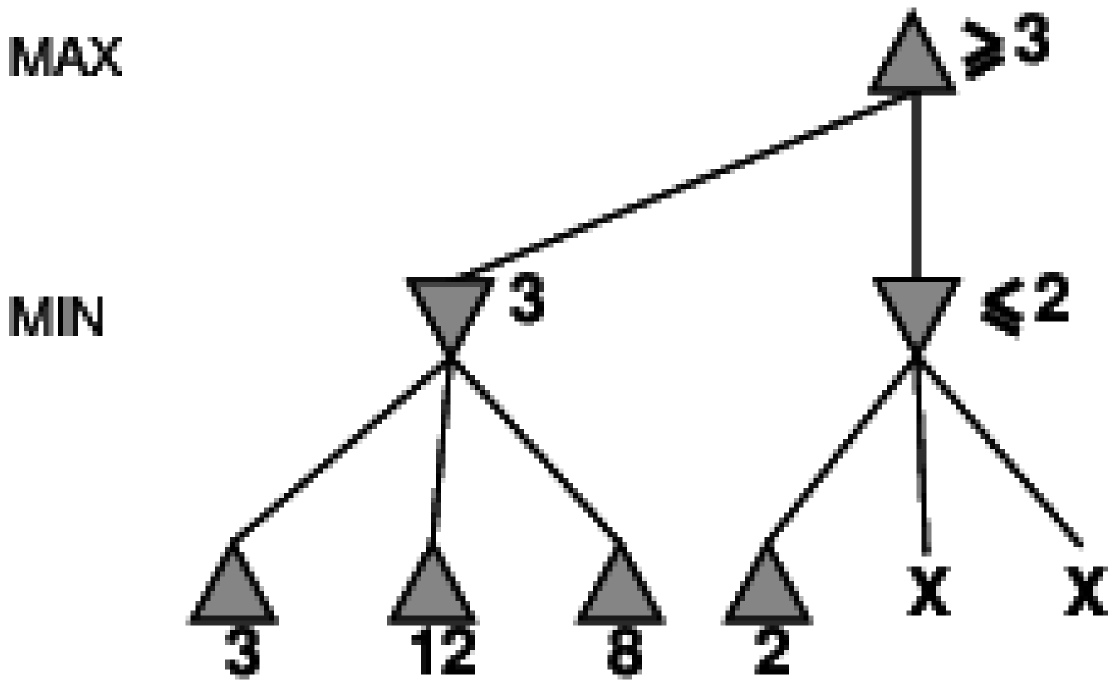
- teljesség
  - igen, ha a fa véges
- optimalitás
  - igen, optimális ellenfél ellen (egyébként?)
- időbonyolultság
  - $O(b^m)$
- tárbonyolultság
  - O(bm) – mélységi keresés esetén

- sakk esetén b ≈ 35, m ≈ 100 (ésszerű játék esetén)
  - a pontos megoldás elérhetetlen
- fel kell deríteni minden útvonalat?

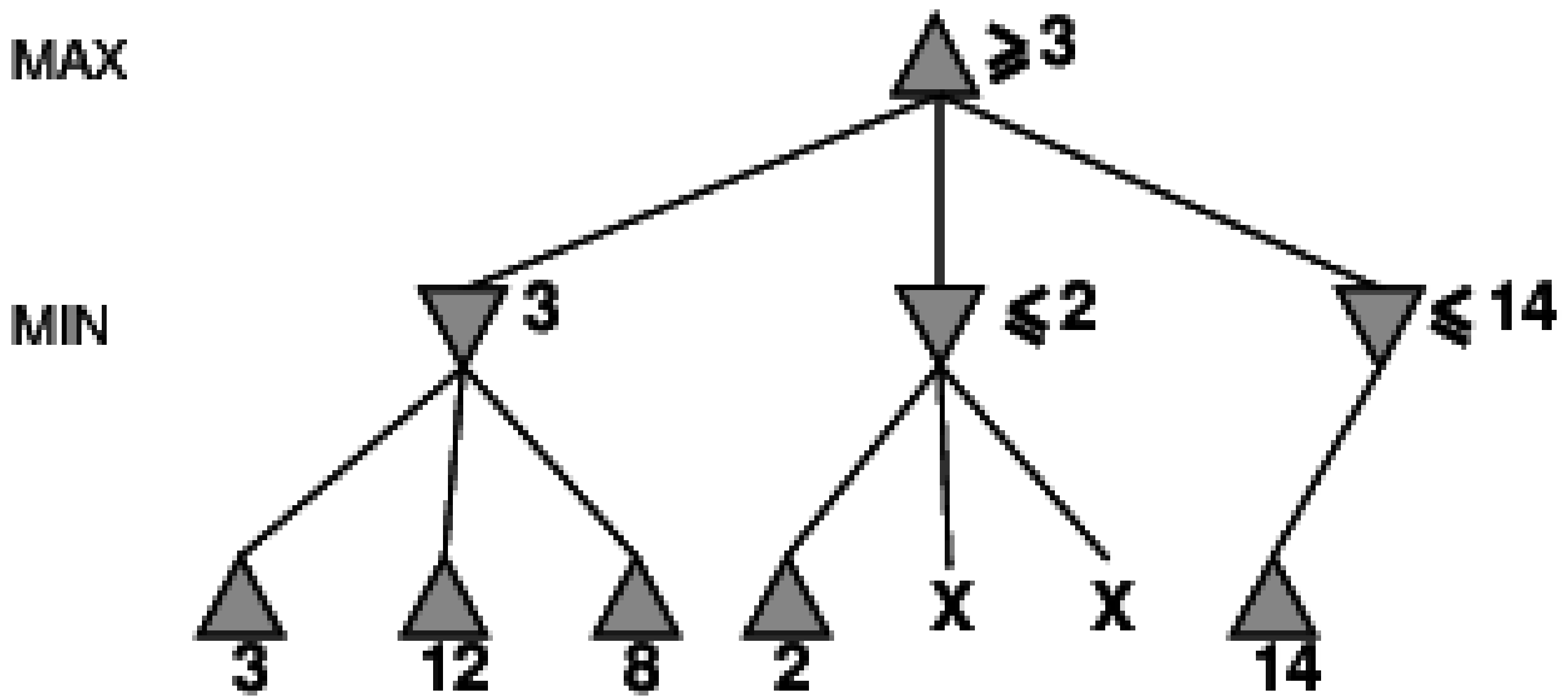
α-β nyesés



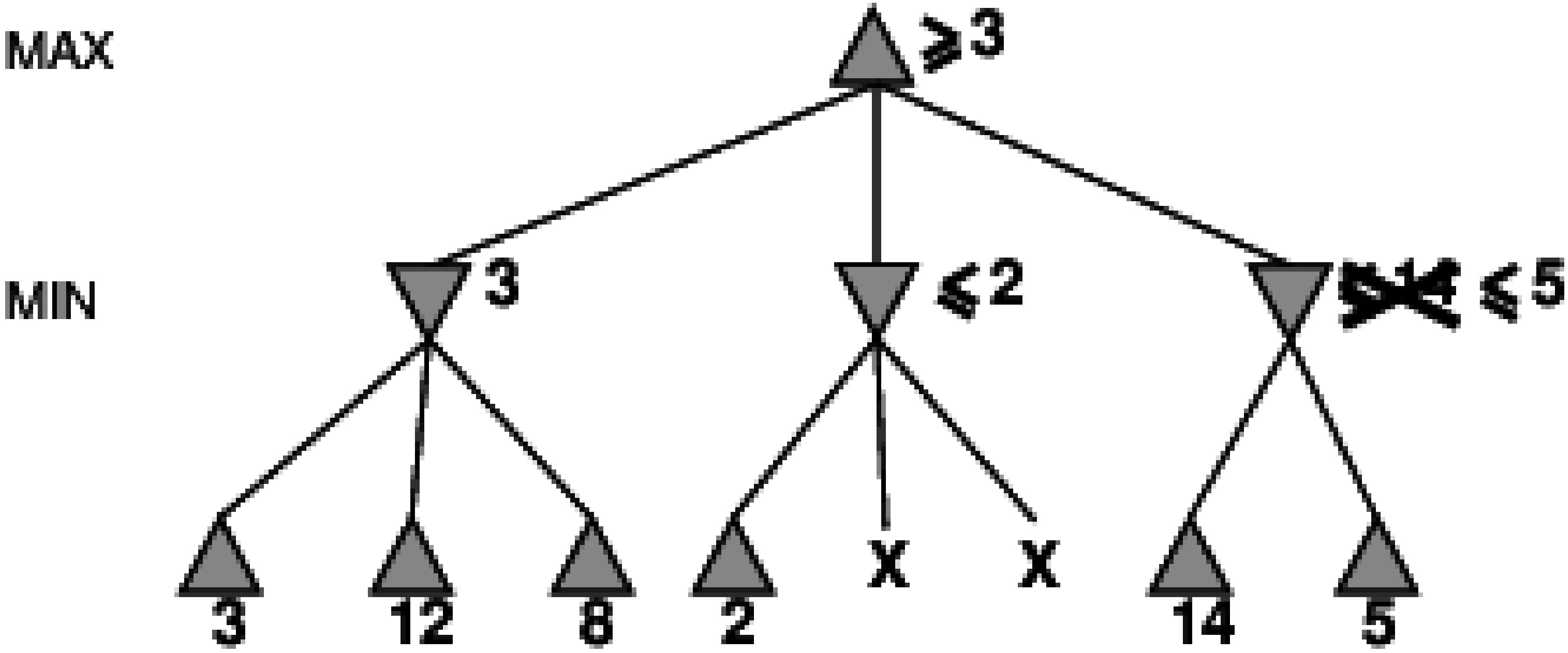
α-β nyesés



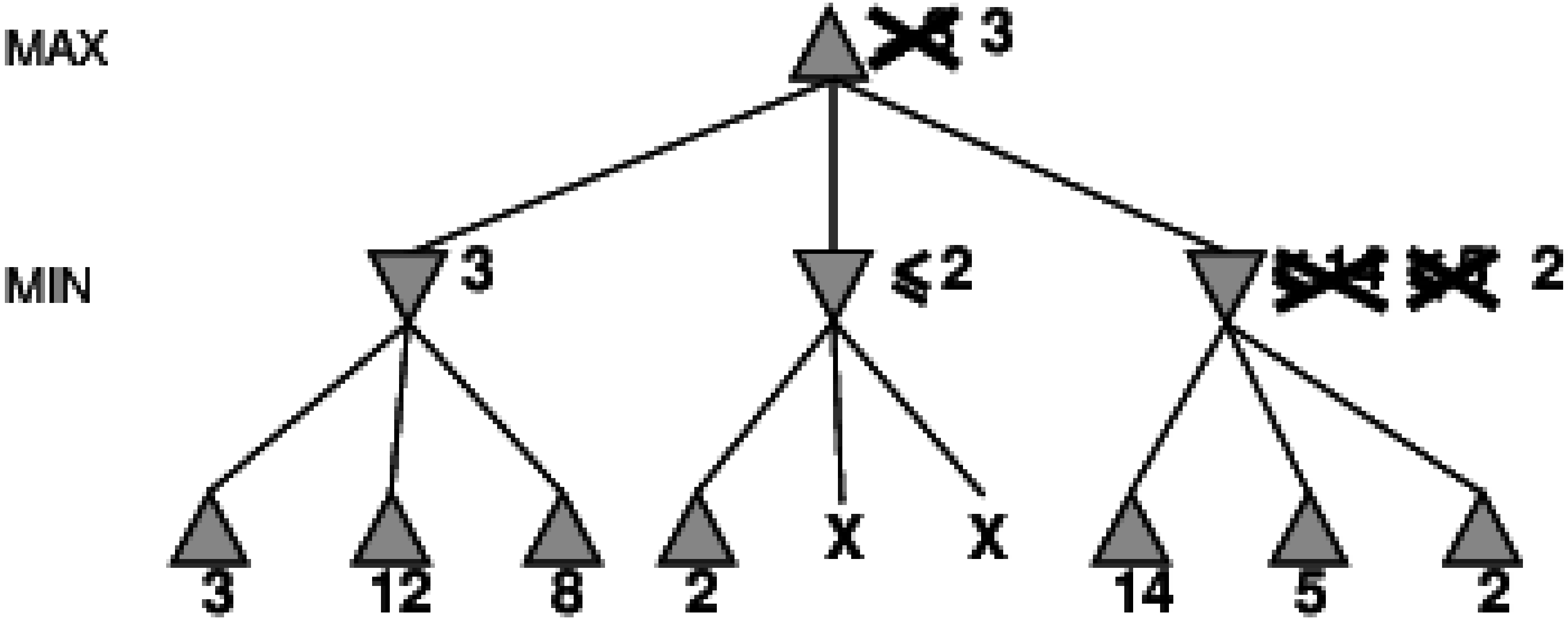
α-β nyesés



α-β nyesés

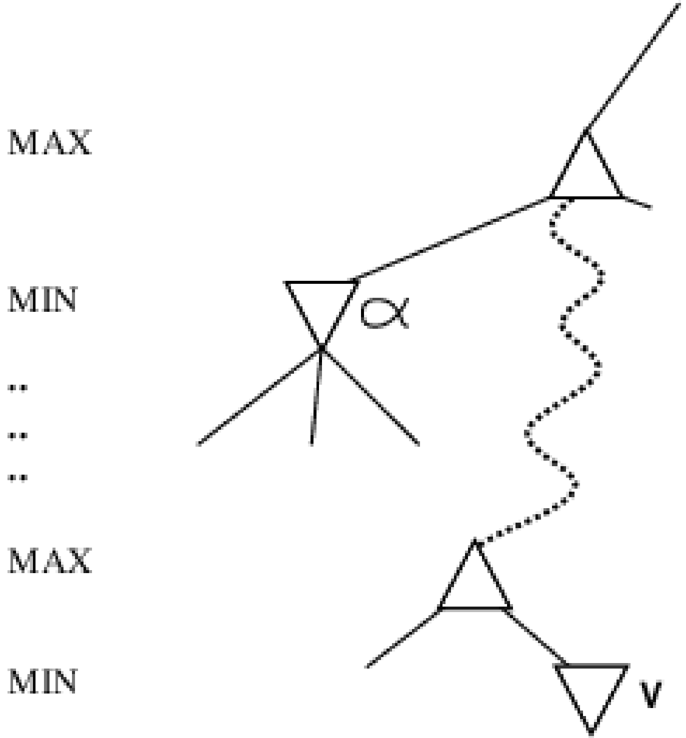


α-β nyesés



Miért α-β?

- α a legjobb érték (MAX szerint), amit eddig az aktuális úton találtunk
- Ha V rosszabb, mint α, akkor elkerüljük
    - adott ág nyesése
- β hasonlóképpen definiált MIN-re



α-β algoritmus

function Alpha-Beta-Decision(state): művelet

  return Actions(state) azon „ a” elemét, amely maximalizálja Min-Value(Result(a, state), -∞,∞)

function Max-Value(state, alpha, beta): hasznosságérték

  state: játék aktuális állapota

  alpha: a state-ig vezető úton, a MAX szerinti legjobb alternatíva értéke

  beta: a state-ig vezető úton, a MIN szerinti legjobb alternatíva értéke

  if Terminal-Test(state) then return Utility(state)

  v:= -∞

  for (a, s) in Successors(state) do

    v := Max(v, Min-Value(s, alpha, beta))

    if v >= beta then return v

    alpha := Max(alpha, v)

  return v

A MinValue(state, alpha, beta) hasonló fv, csak alpha és beta szerepe felcserélődik

α-β tulajdonságai

- a nyesés nincs hatással a végeredményre
- a lépések átrendezése növelheti a nyesés hatékonyságát
- „tökéletes” sorrend esetén az időbonyolultság $O(b^{m/2})$
  - a keresési mélység megduplázható
- 35^{50} még mindig hatalmas szám
- egyszerű példa annak bemutatására, hogy mennyire hasznos a következtetés annak belátására, hogy mely számítások relevánsak

Korlátozott erőforrások

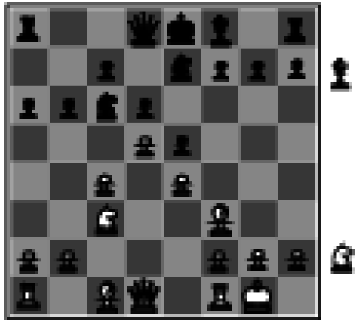
- Szokásos megoldás
  - **`Cutoff-Test`** a **`Terminal-Test`** helyett
  - mélységi korlát (egyensúly állapotok keresése, ahol nincs nagy változás)
- heurisztikus **`Eval`** használata a **`Utility`** helyett
  - kiértékelő függvény, mely az állapot/pozíció kívánatosságát adja meg
- Tegyük fel, hogy 100 másodpercünk van és 10^4 csúcs/s sebességgel haladunk
  - $\Rightarrow 10^6$ csúcs lépésenként, $\approx 35^4$
  - alfa-béta eléri a 8 mélységet $\Rightarrow$ jó képességű sakkprogram

Kiértékelő függvények

Sakk esetén rendszerint tulajdonságok lineárisan súlyozott összege

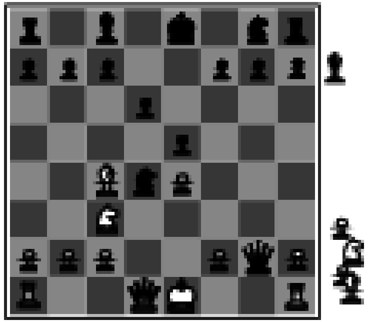
$$\text{Eval}(s) = w_1 f_1(s) + \ldots + w_n f_n(s)$$

ahol pl. $w_1 = 9$, $f_1(s)$ a fehér vezérek száma - fekete vezérek száma



Black to move

White slightly better

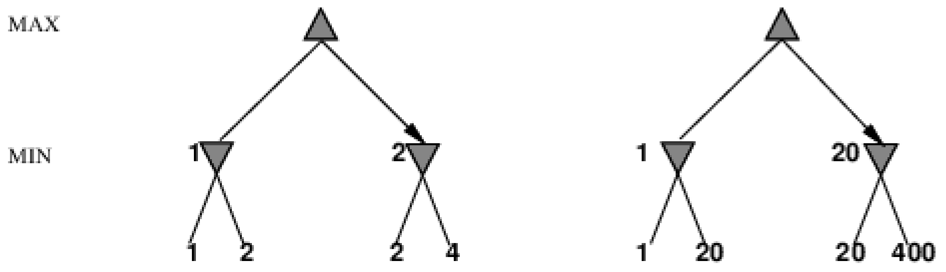


White to move

Black winning

Kitérő: a pontos érték nem számít

- a viselkedés megmarad az Eval bármely *monoton* transzformációja során
- csak a sorrend számít
  - determinisztikus játékok végeredménye tekinthető hasznosságfüggvénynek

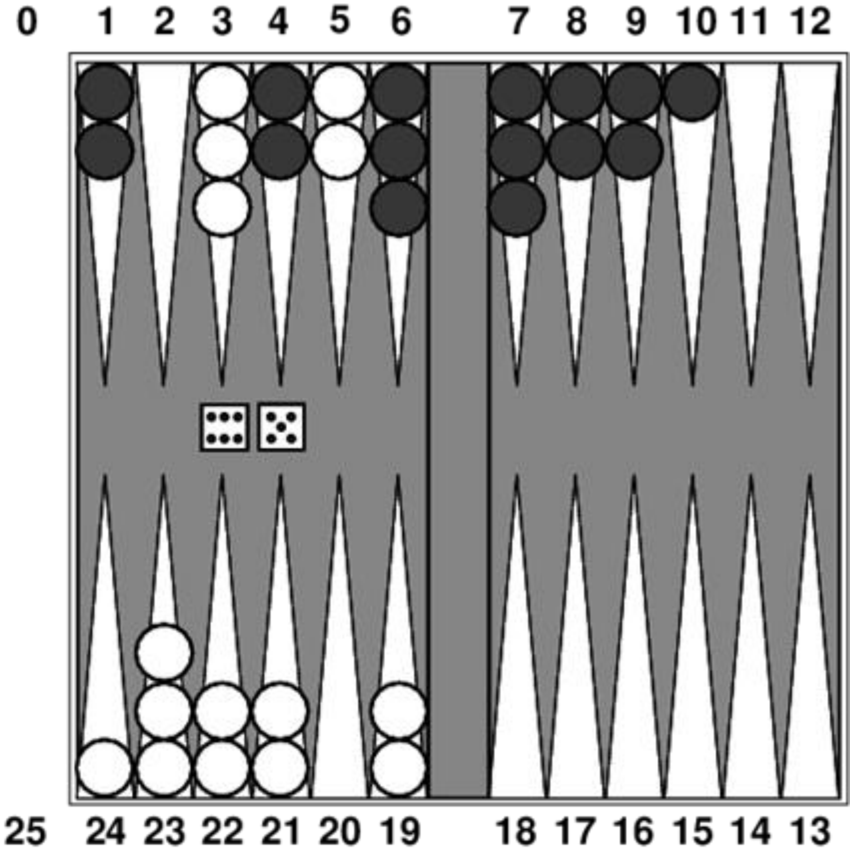


Determinisztikus játékok a gyakorlatban

- **dáma** (1994), egy PC-n futó program (Chinook) megverte a világbajnokot. Közel 444 milliárd állás alapján épült fel a végjáték-adatbázisa a 8 vagy kevesebb figurát tartalmazó állások tökéletes kezelésére.
- **sakk** (1997), Deep Blue (30 db. IBM RS/6000 + 480 célprocesszor, 2 x 10^8 állás/s, 40 lépésig keresés, 8000 tulajdonság) 6 játszmás összecsapás
- **othello**/reversi (2002) 6-0-ra megverte a világbajnokot
- **go**
  - b>300, hatalmas keresési tér, igen gyenge játékosok (2004)
  - 2016 március, AlphaGo (Google) 4-1-re megverte a 9 danos mestert

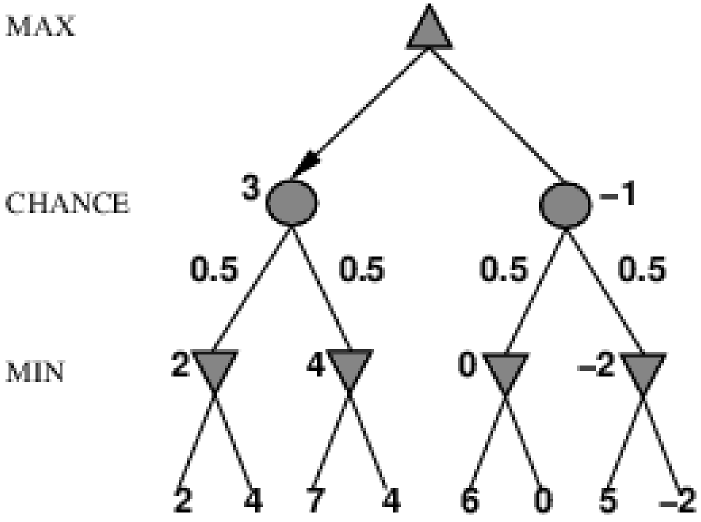
Nemdeterminisztikus játékok: ostábla/backgammon

kockadobás alapján dől el, hogy mennyit léphet a játékos.



Nemdeterminisztikus játékok általában

- A nemdeterminisztikus játékokban a véletlen a kockadobással, kártyakeveréssel jelenik meg
- egyszerű játék pénzfeldobással:



Algoritmus nemdeterminisztikus játékokra

- az **`Expectiminimax`** tökéletes játszást eredményez
- hasonló a **`Minimax`**-hoz, csak kezeli a véletlent is:

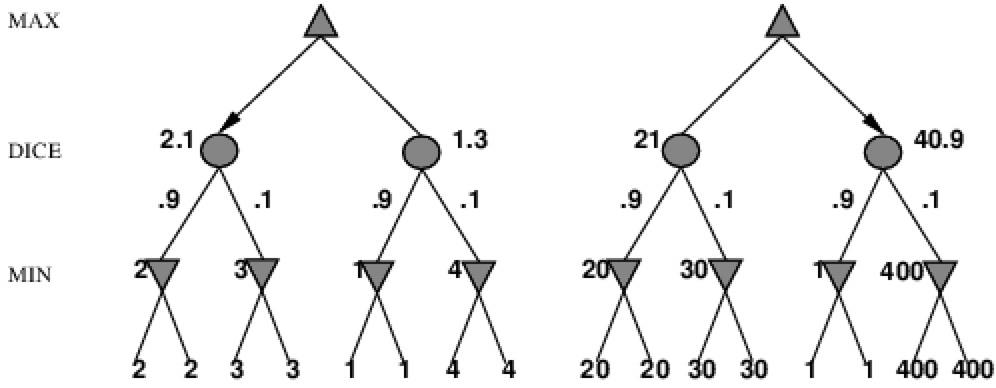
...
if state is a MAX node then
    return the highest ExpectiMinimax-Value of Successors(state)
if state is a MIN node then
    return the lowest ExpectiMinimax-Value of Successors(state)
if state is a chance node then
    return average of ExpectiMinimax-Value of Successors(state)
...

Nemdeterminisztikus játékok a gyakorlatban

- a kockadobás növeli *b* értékét: 2 egyforma kockával 21 különböző eredmény
  - ostábla ≈ 20 lehetséges lépés egy állapotban (dupla dobásnál akár több ezer is lehet)
  - depth(4) = $20x(21x20)^3 \approx 1.2x10^9$
- ahogy a mélység növekszik, egy adott állapot elérésének a valószínűsége csökken
  - nem igazán hatásos a előrenéző vizsgálat
- az alfa-béta vágás nem olyan hatékony mint determinisztikus esetben
  - TDGammon 2 mélységű keresést használ és egy jó kiértékelő függvényt
    - világbajnoki szintű

Kitérő: számítanak a pontos értékek

- a viselkedés csak az **`Eval`** pozitív lineáris transzformációi esetén marad ugyanaz
- az **`Eval`**-nak arányosnak kell lenni a játék végeredményével (kifizetéssel)

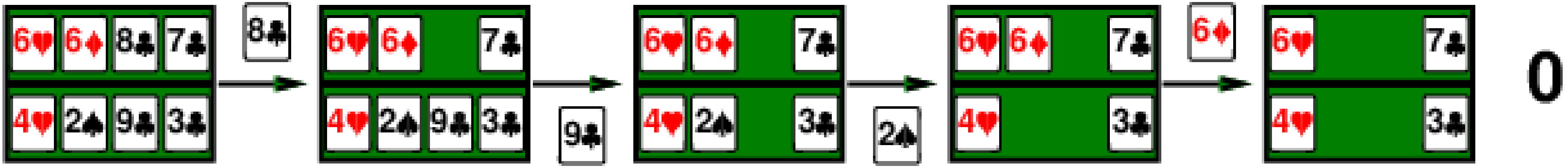


Nem teljes információjú játékok

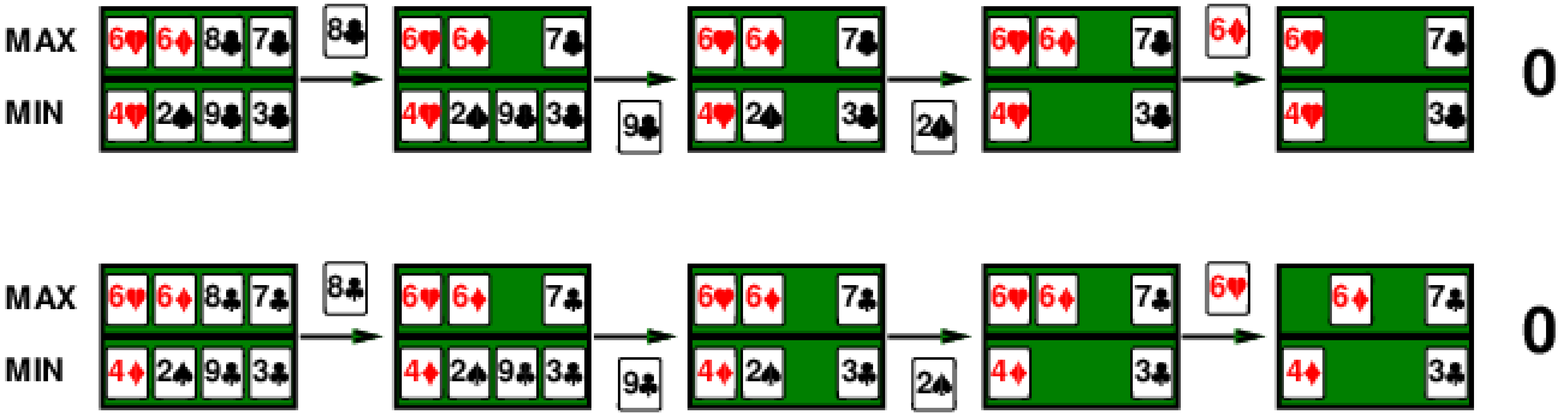
- pl. kártyajátékok, ahol az ellenfél lapjai nem ismertek
- rendszerint minden lehetséges leosztás valószínűségét kiszámoljuk
- mintha egy sok oldalú kockát dobtunk volna a játék elején
- ötlet: számoljuk ki a minimax értékét az összes lépésnek (műveletnek), és válasszuk közülük azt, melynek a legmagasabb várható értéke van az összes leosztást tekintve
- GIB (a legjobb bridzsprogram 1998-ban) a következőképpen működik
  - a bemondásokkal konzisztens 100 leosztás generálás
  - azt a lépést választja, mely átlagban a legjobb eredményt éri el

Példa

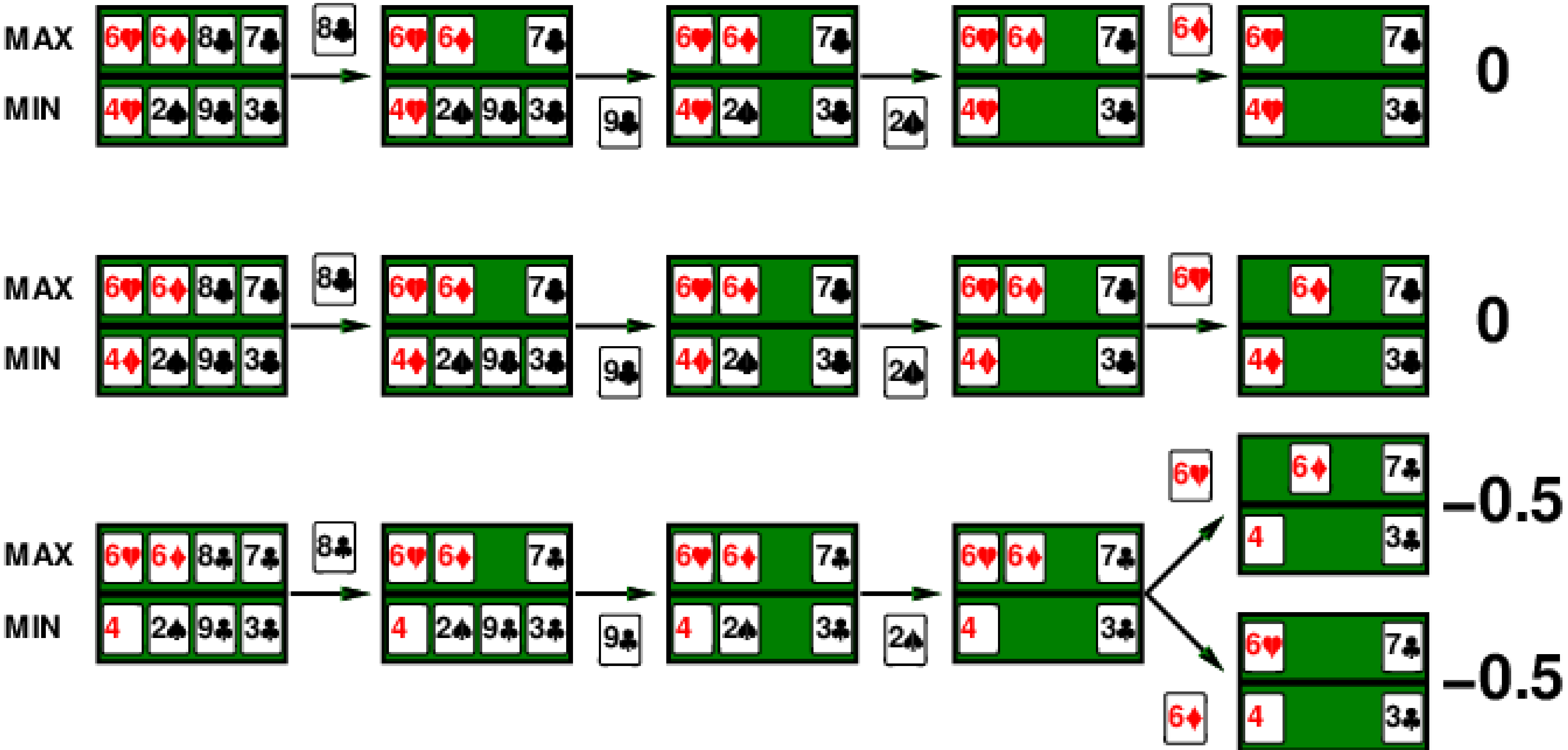
Négy kártyás bridzs, a MAX kezd



Példa



Példa



Józan ész kritikája

- 1. nap
  - Az *A* út egy halom aranyhoz vezet.
  - A *B* út egy elágazáshoz vezet.
    - Ha az elágazásnál balra fordul, egy ékszerhegyet talál,
    - ha az elágazásnál jobbra fordul, elüti egy busz.
- 2. nap
  - Az *A* út egy halom aranyhoz vezet.
  - A *B* út egy elágazáshoz vezet.
    - Ha az elágazásnál jobbra fordul, egy ékszerhegyet talál,
    - ha az elágazásnál balra fordul, elüti egy busz.
- 3. nap
  - Az *A* út egy halom aranyhoz vezet.
  - A *B* út egy elágazáshoz vezet.
    - Ha jól választ, egy ékszerhegyet talál,
    - ha rosszul, elüti egy busz.

Pontos analízis

- intuíció szerint a lépés értéke az értékek átlaga
  - minden aktuális állapotban HIBÁS
- részleges információ esetén a lépés értéke az ágens hiedelmi állapotától függ
- az hiedelmi állapotok fája generálható és kereshető
- racionális viselkedés
  - információt szerző lépés
  - jelzés a partnernek
  - véletlen módon cselekedni, hogy minél kevesebb információt tegyük közzé

Összegzés

- a játékok szórakoztatóak
- az MI fontos pontjait mutatják meg
  - a tökéletesség nem elérhető ⇒ közelítés szükséges
  - érdemes elgondolkozni azon, hogy a program min gondolkozzon (vegyen számításba)
  - a bizonytalanság korlátozza értékek rendelését az állapotokhoz
  - optimális döntés a hiedelmi állapoton múlik, nem a valós állapoton
- a játékok az MI számára az, ami a Formula1 az autógyártók számára