Debreceni Egyetem

Informatika Kar

KÉTSZEMÉLYES JÁTÉKOK

Témavezető:

Mecsei Zoltán

Egyetemi tanársegéd

Készítette:

Briz Ádám

Programtervező informatikus Bsc

Debrecen

2009

TARTALOMJEGYZÉK

1. Bevezetés

A mesterséges intelligencia olyan funkciók megvalósítására alkalmas gépek megalkotásának tudománya, mely funkciókhoz intelligenciára van szükség, amennyiben azokat emberek valósítják meg. (Kurzweil, 1990)

Azok a problémák tartoznak ide, amelyek nem rendelkeznek fix megoldó mechanizmussal, előre rögzített megoldási sorrenddel, hanem szerepet kap a próbálkozás és az azt irányító emberi szakértelem. A kétszemélyes játékok témaköre az elsők között keltette fel a mesterséges intelligenciával foglalkozók érdeklődését. Vonzó tulajdonsága volt a játékok kutatásának az, hogy a teljesítmény közvetlenül mérhető: egy program és egy játékos, vagy két program mérkőzései alapján eldönthető, hogy melyikük képvisel nagyobb játékerőt. A kihívás tehát adott volt: Létrehozni olyan számítógépes programokat, amelyekkel túlszárnyalható az emberi teljesítmény. Ez mára valósággá vált: Például 2006-ban a Deep Fritz nevű sakkprogram legyőzte a világbajnok Vlagyimir Kramnyikot.

A továbbiakban szó lesz a játékok reprezentációs eszközeiről, stratégiáiról, lépésajánló algoritmusairól; valamint bemutatásra kerül egy konkrét játék implementációja is.

2. Kétszemélyes játékok

Játékelmélet

Egy játék alapvetően három komponensből áll: játékosokból, a játékszabályokból és az eredmények értékeléséből: Mindegyik játékos felállít egy rangsort a játék lehetséges kimenetelei között. A cél természetesen ezek közül a legjobbat elérni. A játékos ezt szem előtt tartva választja lépéseit a játékszabályok betartásával.

Független attól, hogy hányszor vagy mikor kerül döntéshelyzetbe, az a döntéssorozat terv, amely a játék minden lehetséges döntéshelyzetére és az ebben tapasztalható minden lehetséges állapotára tud egy konkrét döntést, a játékos egy stratégiája. A játékban előálló helyzetek természetesen függnek a játékostársak lépéseitől is, de a játékos stratégiája nem, csak legfeljebb más válaszlépést ír elő.

A stratégiának tehát egy olyan teljes tervnek kell lennie, amelyet az ellenfél nem tud keresztülhúzni. Akármit választ ugyanis, az a mi lehetséges döntéssorozataink halmazával együtt éppen a stratégia leírásának részét képezi. A cél: a lehetséges stratégiák közül az optimális megtalálása.

Játékok típusai

Egy játék kategorizálásához a következőket kell ismernünk:

- *Játékban résztvevők számát.*
- *Kiinduló állást és a játék közben lehetséges állásokat.*
- *Mi számít szabályos lépésnek.*
- *Mikor ér véget a játék és ki a győztes.*
- *Játékosok a játék közben milyen információval rendelkeznek.*
- *Véletlennek van-e szerepe.*

Ezek alapján különböző csoportba sorolhatjuk az egyes játékokat:

(1) Játékosok száma szerint:

- *egyszemélyes játékok*
- *kétszemélyes játékok*
- *három vagy többszemélyes játékok*

(2) Véletlen szerepe szerint:

- *Determinisztikus játékok*: Véletlen nem játszik szerepet.
- *Sztochasztikus/valószínűségi játékok*: Véletlennek is szerepe van a játék folyamán.

(3) Játék végessége szerint:

- *Véges játékok*: Minden állásban a játékosoknak véges sok lépési lehetősége van és a játék véges sok lépés után befejeződik.

(4) Információ mennyisége szerint:

- *Teljes információjú játékok*: Minden játékos rendelkezik a játék során felmerülő összes információval.

(5) Nyereségek és veszteségek szerint:

- *Zérusösszegű játékok*: Játékosok nyereségeinek és veszteségeinek az összege 0.

Néhány példa ezekre a csoportokra:

	Determinisztikus	Valószínűségi
Teljes információjú	sakk, amőba	dáma, monopoly
Részleges információjú	torpedó	bridge, póker

Továbbiakban a kétszemélyes, determinisztikus, véges, teljes információjú, zérusösszegű játékokról lesz szó.

Összefoglalva, ezt a játékosztályt a következőképpen definiálhatjuk:

- Két játékos lép felváltva egymás után, a megadott szabályok szerint.
- Minkét játékos birtokában van a játékokkal kapcsolatos összes információnak. Ismerik a játék során eddig megtett lépéseket és az aktuális állás minden részletét látják. Szerencsének semmilyen szerepe sincs a játék alakulásában.
- Játék során minden egyes állásban véges számú szabályos lépés közül választhatnak.
- Játék szabályai olyanok, hogy végtelen játszmák nem fordulhatnak elő.
- Játék végén az egyik játékos nyer, a másik veszít. Néha előfordulhatnak olyan esetek, amikor a végeredmény döntetlen.

(Ehhez először egy megfelelő reprezentációs eszközt kell választani.)

Állapottér-reprezentáció

Egy feladat megoldása mindig azzal kezdődik, hogy formális módon reprezentáljuk azt. Az egyik ilyen, meglehetősen általánosan használható technika az állapottér-reprezentáció.
Egy probléma reprezentálásához meg kell keresni az adott probléma világának véges sok, a probléma megoldása során fontosnak vélt tulajdonságát, jellemzőjét (pl.: szín, méret, pozíció, stb.). Például, ha ezeket a jellemzőket a $t_1,...,t_n$ értékek jellemzik, akkor a $(t_1,...,t_n)$ vektorral azonosított állapotban van a probléma világa. Ha az i. jellemző által felvehető értékek halmaza T_i, akkor a probléma világának állapotai elemei a $T_1 \times T_2 \times ... \times T_n$ halmaznak. Egyes

állapotokban tehát a feladat minden adata képviseli magát egy-egy értékkel, az *állapottér* pedig az összes lehetséges adat-kombinációt tartalmazza.
Meg kell adni azt a speciális állapotot, amely a probléma világához tartozó jellemzők kezdőértékeit határozza meg: ez a *kezdőállapot*.
A probléma megoldása során a kezdőállapotból indulva a probléma világának előálló állapotai meg fognak változni és végül egy számunkra megfelelő *célállapot*ba fogunk jutni. Ebből akár több is megadható.
Specifikálni kell, hogy mely állapotokat lehet megváltoztatni, és hogy ezek megváltoztatása milyen állapotokat eredményez. Állapotváltozásokat leíró függvények az *operátor*ok.
Nem minden operátor alkalmazható minden állapotra, ezért az operátorok (mint függvények) értelmezési tartományát az *operátoralkalmazási előfeltétel*ek segítségével rögzíthetjük.

Tehát az *állapottér-reprezentáció* egy $\langle A, k, C, O \rangle$ formális négyes, ahol:

- $A = \{(a, p) \mid a \in H,\ p \in \{1,2\}\}$ az állapotok halmaza.

  H: a játékállások halmaza; 1,2(vagy A,B??): a játékosok.
- $k \in A$ kezdőállapot.
- $C \subseteq A$ célállapotok halmaza.

  Minden $(a, p) \in C$ esetén meg kell mondani: ki nyer vagy veszít, esetleg döntetlen-e.
- $O = \{o \mid o : Dom(o) \to A\}$ operátorok halmaza.

  Minden $o \in O$ operátor egy $o : Dom(o) \to A$ függvény, ahol

  $Dom(o) = \{a \mid a \notin C \wedge előlőfeltél_o(a)\} \subseteq A$.

  Ha $o(a, p) = o'(a', p')$, akkor:
  - a-ból szabályos lépés vezet a'-be
  - p játékos után p' lép

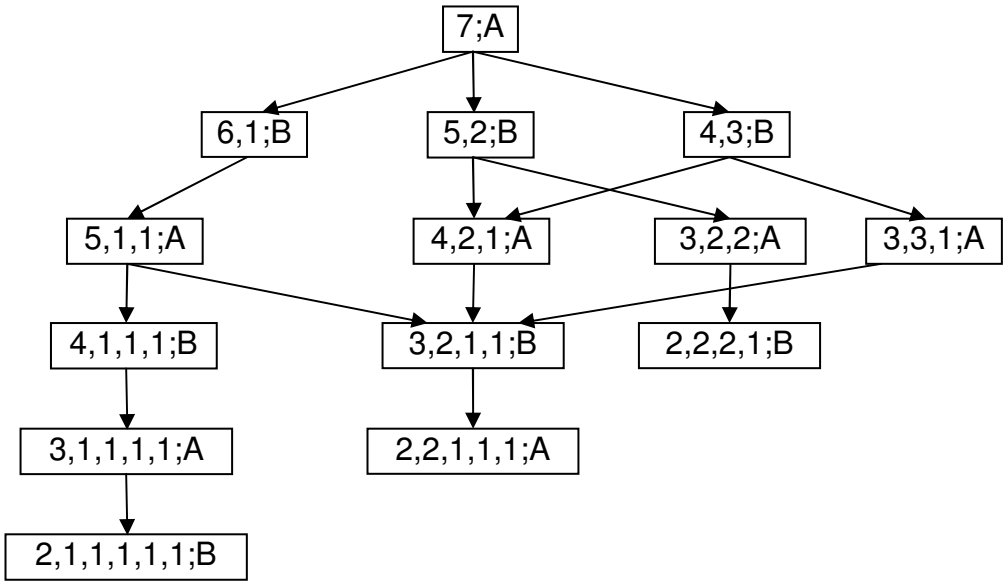
Reprezentálás gráffal

Az állapottér reprezentációt szemléletessé tehetjük irányított gráf alkalmazásával. A gráf csúcsai maguk az állapotok és két csúcs között akkor és csak akkor húzunk élt, ha az egyik

csúcsból (állapotból) a másik csúcs (állapot) közvetlenül elérhető. Az éleket a közvetlen elérhetőséget lehetővé tevő operátorral címkézhetjük meg.
A gráfnak azt a csúcspontját, amelyből csak kivezető élek indulnak (vagyis a gráf gyökerét), a kiinduló állással feleltethetjük meg. Azok a csúcspontok, amelyekből nem vezet ki él, a játszma végállapotait tartalmazzák. Egyes szinteken lévő csúcspontokban a játék állásai mellett szükségünk van még arra a plusz információra is, hogy melyik játékos az éppen soron lévő. Ezekből a csúcspontokból kivezető élek a játékos szabályos lépéseinek felelnek meg.
Ezt a gráfot játékgráfnak nevezzük, és tartalmazza az összes lehetséges játszmát. Egy konkrét játszmának pedig egy olyan út felel meg, amely a gráf kezdőcsúcsától a végcsúcsába vezet.

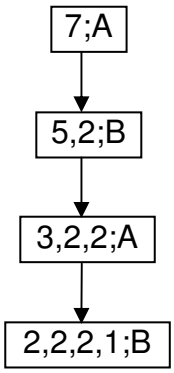
Példáként tekintsük a Grundy-féle játékot. Ennek a játéknak a kezdetén adott egy oszlop, például pénzérmékből, amely legalább három érmét tartalmaz. Úgy kell kettéválasztani ezt az oszlopot, hogy a kialakuló két oszlop különböző számú érmét tartalmazzon. A második játékos a kialakult két oszlop közül választ egyet és az előzőek szerint ismét kettéválasztja. Ezt felváltva teszik mindaddig, míg lehetséges; vagyis amikor az összes oszlop már csak egy vagy két érmét tartalmaz.

1. ábrán látható a teljes játékgráf. (Kezdetben 7 darab pénzérménk van egy oszlopban. A-val jelöltük az első, B-vel a második játékost.)



1. ábra: A Grundy-féle játék teljes játékgráfja

2. ábrán egy lehetséges játszma látható, ahol az A nyer, ugyanis a végállapotban a B már nem tud megtenni további lépést. Ennek megfelelő út az 1. ábrán látható teljes játékgráfnak egy része.



2. ábra: A Grundy-féle játék egy lehetséges játszmája.

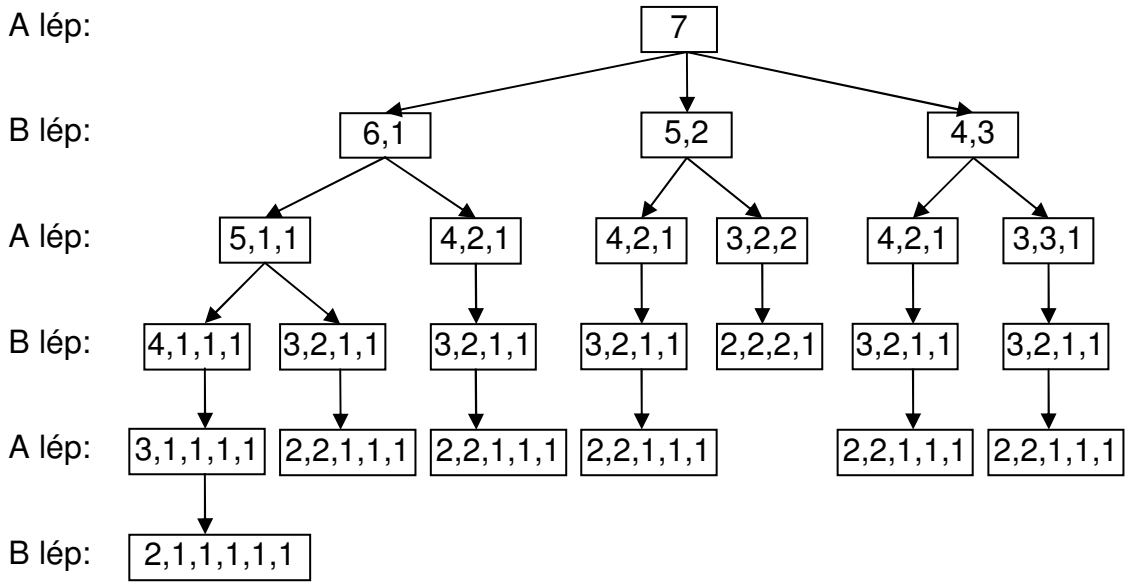
Reprezentálás fával

Gráf és a fa között az alapvető különbség, hogy a gráfban egy csúcspontba több úton is eljuthatunk. Ez felel meg a tényleges helyzetnek is, hiszen egy állásba több különböző lépéssorozattal is eljuthatunk, a gyakorlatban mégis inkább a játékfát használjuk a játékok ábrázolására.

Egy csúcspontba ezáltal egy úton jutunk el, így annyiszor fog szerepelni egy állás a fában, ahányféleképpen eljuthatunk hozzá a kezdőállásból. Ezzel viszont jelentősen megnőhet a csúcspontok száma, de a fa szerkezet egyszerűbb, könnyebben kezelhető adatszerkezet.

Fa esetében beszélhetünk annak szintjeiről. A kezdőállás a nulladik szinten van. Szintek megkülönbözetéséből adódik, hogy a felváltva lépő játékosoknak megfeleltethetők a fa páros, illetve páratlan szintjei és ezáltal nem szükséges az egyes állások mellett feltüntetni, hogy melyik az éppen soron következő játékos.

3. ábrán látható a Grundy-féle játék fával történő ábrázolása, amely az 1. ábrán látható gráf fává alakított változata.

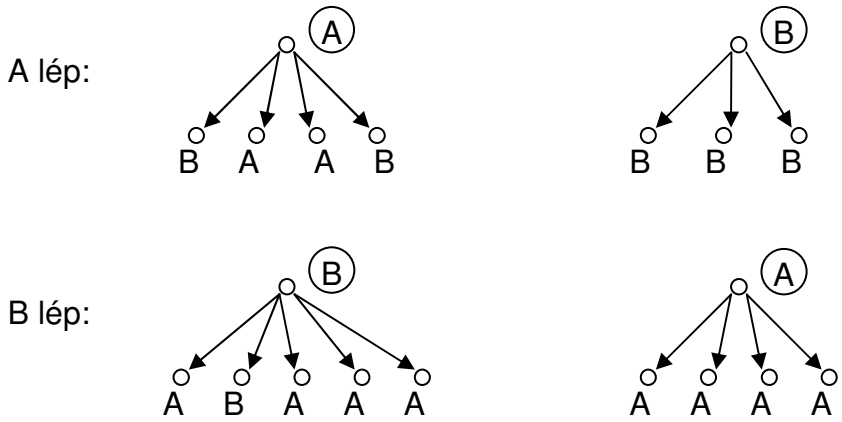


3. ábra: A Grundy-féle játék teljes játékfája

Nyerő stratégia

A játékfa alkalmas egy játék összes játszmáinak ábrázolására. A játékkal kapcsolatos alapvető kérdés azonban a nyerő stratégia meghatározása. Ez mindig az egyik fél szempontjából vetődik fel.

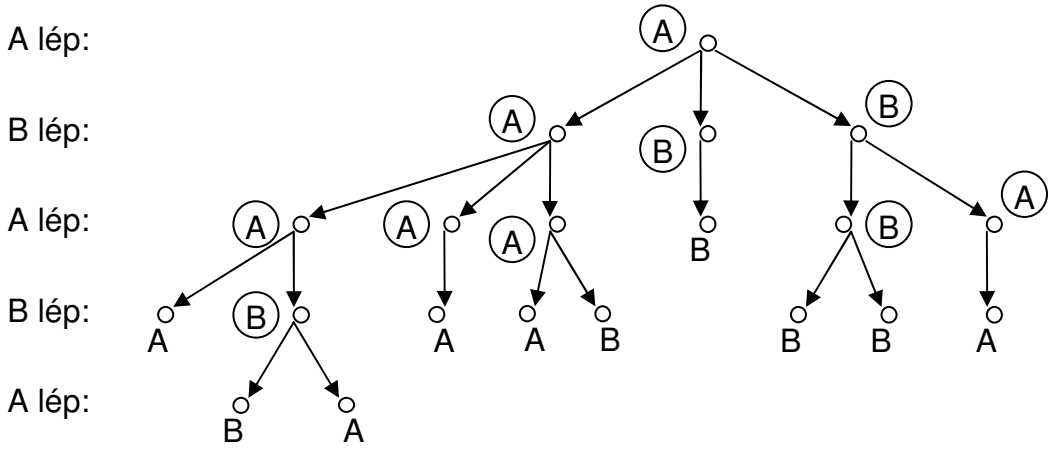
Feltesszük, hogy adott egy játék teljes játékfája. Ebben az esetben a fa levélelemeiről el tudjuk dönteni, hogy azok mely játékos számára nyerőek és ezek alapján eljelölhetjük például A-val, illetve B-vel. Majd alulról felfelé haladva, egészen a gyökérig, eljelölhetjük a közbülső csúcsokat. Ha az adott csúcspontban az A lép és van is olyan kivezető ág, ami A jelű, akkor az a csúcspont A lesz, ellenkező esetben B. Hasonlóképpen járunk el abban az esetben, amikor a B lép; azonban ha van B kivezető ág, a csúcsot B-vel jelöljük, ellenkező esetben A-val. Néhány lehetséges címkézési eset látható a 4. ábrán, ahol a megállapított címkék be vannak karikázva a vizsgált csúcspont mellett.



4. ábra: Egy csúcspont megcímkézésének néhány esete

A csúcspontok címkéi így meghatározzák, hogy egy állás melyik játékos számára kedvező. Mivel a játékfa véges, így végül a gyökérelem is kap valamilyen jelzést és az itt jelzett játékos számára meg is adtuk a nyerő stratégiát: Mindig tud olyan csúcspontra lépni, amely az ő jelzésével van ellátva, vagyis az ellenfél tetszőleges lépése esetén, mindig van legalább egy olyan lépés, mellyel számára kedvező végállapotba tud kerülni.

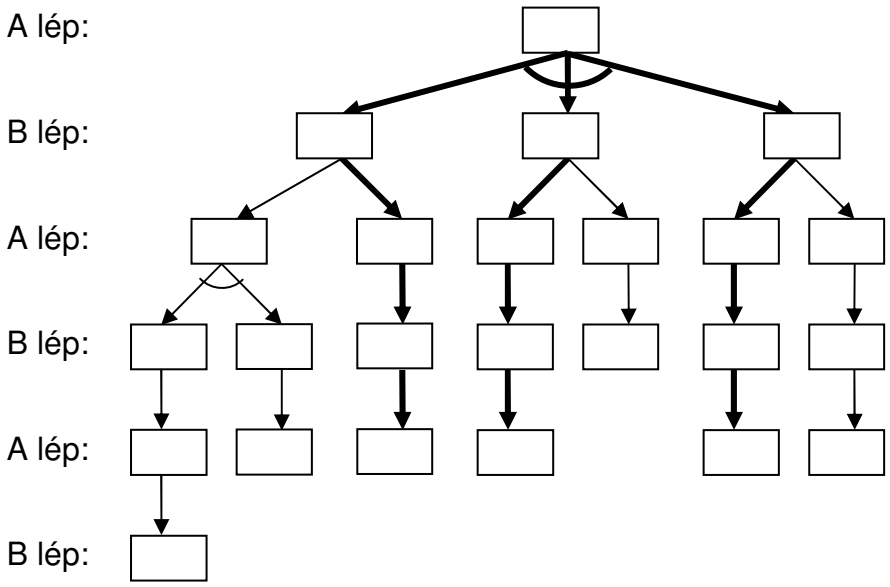
5. ábrán egy ilyen megcímkézett játékfa látható, amiben az A játékosnak van nyerő stratégiája.



5. ábra: Egy játékfa megcímkézése

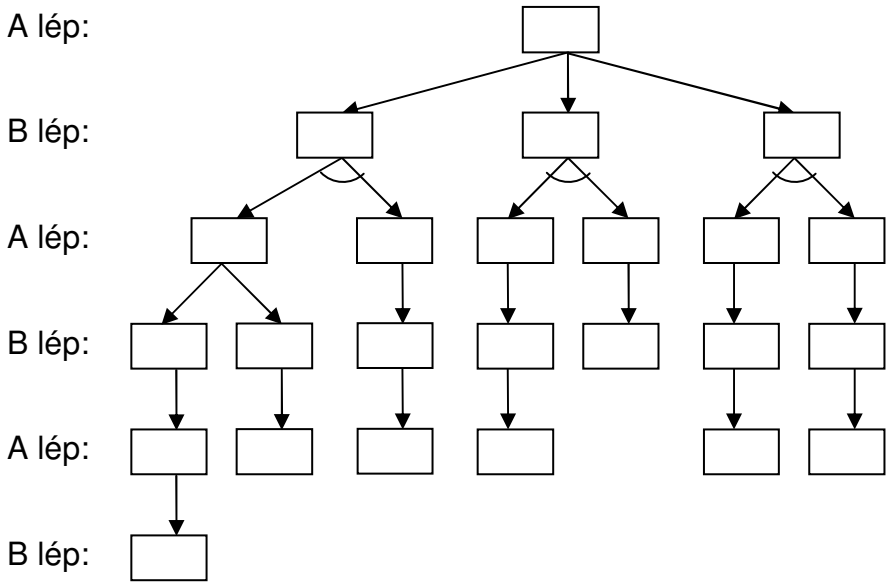
A két játékos szemszögéből tehát más és más az elérendő cél.

Tekintsük most azt az esetet, amikor a B szempontjából készítjük el a fát. Ekkor ahol az A-nak egyszerre több lehetséges lépése van, ott az A lépéseit körívvel kötjük össze (ÉS kapcsolatúak). Ezeket ugyanis mind figyelembe kell venni a stratégia kialakítása során, hiszen nem tudhatjuk melyik lépése lesz a választott. Majd A minden lehetséges kezdőlépésre keresnünk kell (B-nek) egy jó válaszlépést (VAGY kapcsolat) és az így elérhető állapotokból ismét meg kell vizsgálni az A játékos összes lehetséges lépését, és így tovább. A játékfát ilyen módon az egyik játékos szempontjából ÉS/VAGY fává alakították át. (6. ábra)



6. ábra: A Grundy-féle játék ÉS/VAGY fája a B játékos szempontjából

Egy adott játékos számára akkor létezik a nyerő stratégia, ha a teljes ÉS/VAGY fában található olyan részfa, amelynek levélelemei mind az ő számára nyerő állások és a részfa tartalmazza a fa gyökerét; valamint ha egy csúcspontban az adott játékos következik, akkor szerepel benne legalább egy kimenő él, ha viszont az ellenfél következik, akkor pedig az összes él. Ilyenből általában több is lehetséges, a 6. ábrán vastag vonallal vannak jelölve a lehetséges részfák. Azonban van olyan helyzet, amikor nem tudunk kijelölni ilyen részfát, például abban az esetben, ha az előbbi példát az A játékos szempontjából tekintjük. (7. ábra)



7. ábra: A Grundy-féle játék ÉS/VAGY fája az A játékos szempontjából

Játékfa részleges kiértékelése

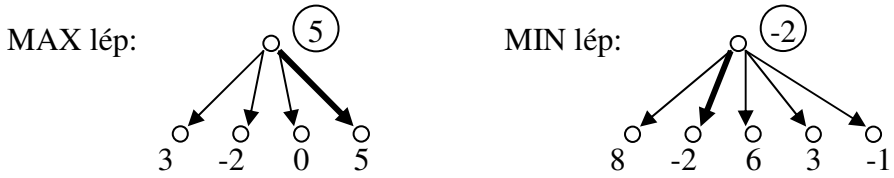
Eddig feltételeztük a teljes játékfa ismeretét, azonban ez csak egyszerűbb játékok esetén lehetséges. Minél bonyolultabb egy játék, a teljes játékfa annál nagyobb.
Például a sakk esetében egy állásból kb. 35 szabályos lépés tehető. Egy átlagos játszmában, ha csak 30 lépésváltás történik, akkor is a fának már $35^{(2*30)} = 35^{60}$ kiértékelendő eleme van. Ennek reális időn belül történő feldolgozása jelenleg nem lehetséges; így egy biztos nyerő stratégiát nem is tudunk meghatározni, csak néhány lépés erejéig tudunk „előre gondolkodni”. Ahhoz, hogy ezt a feldolgozási időt lerövidítsük, valamilyen mélységi korlátot kell szabnunk, mivel nem akarjuk a fát teljes egészében legenerálni. Ez általában egy rögzített szintszám, de lehet például idő vagy memória korlát is.

Minimax eljárás

A minimax eljárás a soron következő játékos legjobb első lépésének a kiválasztására ad módszert. Arra ez elvre épül, hogy például a mi legjobb lépésünk az ellenfél számára a legrosszabb és fordítva, az ellenfél legjobb lépése számunkra a legrosszabb.
Egy kiértékelő függvényt alkalmazunk a fa levélelemeire, amellyel megpróbáljuk minősíteni az állást, hogy az mennyire lesz „pozitív” a győzelem szempontjából. Minél jobb egy állás

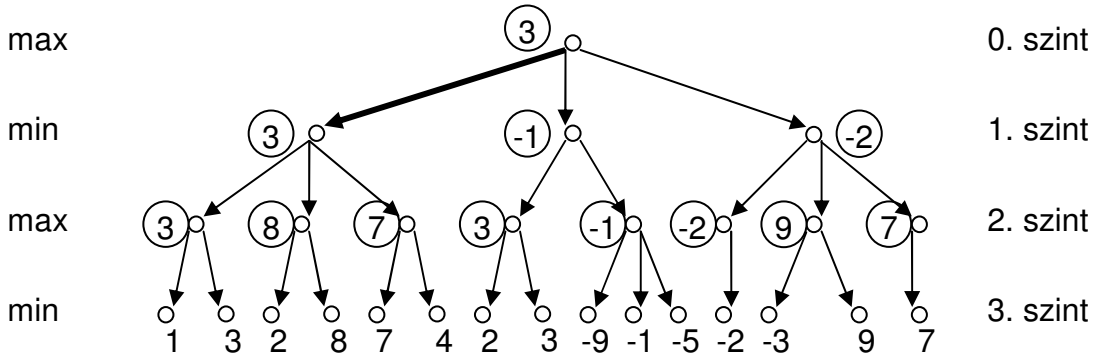
számunkra, annál nagyobb pozitív számmal, ha viszont az ellenfél számára kedvezőbb a lépés, negatív számmal jellemzi a függvény. Döntetlen esetén ez a szám nulla vagy egy ehhez közeli érték.

A nem levélelemek esetében, amikor MAX lép, a maximális értékű állásba vezető lépést választjuk és az adott csúcspont értéke ez lesz. Hasonlóképpen járunk el abban az esetben is, amikor MIN lép, de ilyenkor a minimális értékű állásba vezető lépést választjuk és ez lesz az adott csúcspont értéke. (8. ábra)



8. ábra: A következő lépés meghatározásának elve

Ezt –egy bizonyos mélységig felépített– fában alulról felfelé végezzük, egészen a gyökércsúcsig. MAX számára a legkedvezőbb lépés így az az állás, amelyhez ez a legutóbbi érték van hozzárendelve. Ezzel a módszerrel kiépített, 3 mélységű játékfa látható a 9. ábrán.



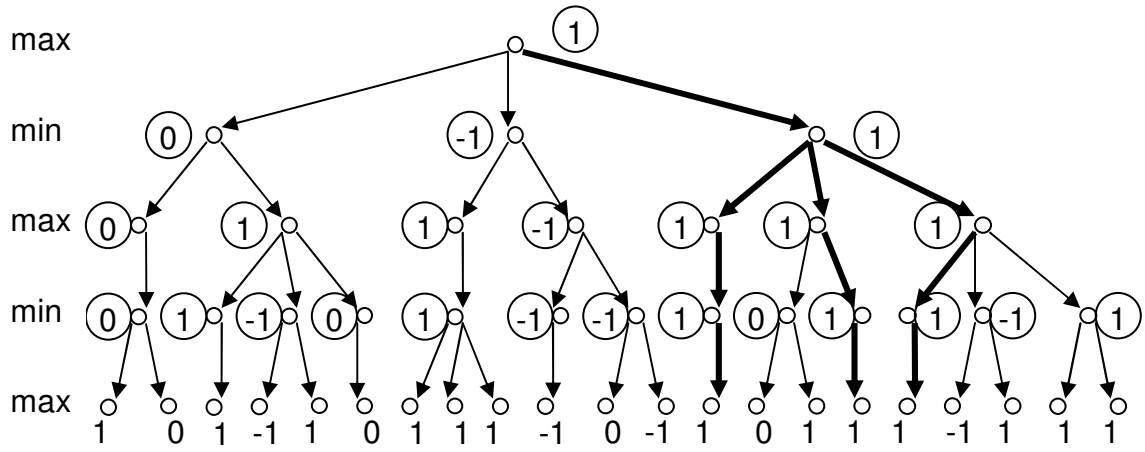
9. ábra: Egy játékfa minimax kiértékelése

Példában a második szinten MAX lép, így a harmadik szinten lévő levélelemek értékei közül a maximálisak kerülnek ide. Ezután az első szintű MIN csúcsokba kerülnek a második szint minimumai és végül a gyökérelem kap értéket, ismét a lehetségesek közül a maximálisat.

Fontos megjegyezni, hogy a minimax módszer csak MAX következő lépését határozza meg, mivel nem lehetünk abban biztosak, hogy MIN ténylegesen azt a lépést teszi-e meg, mint amit

mi a legkedvezőbbnek ítéltünk számára. Lehetséges, hogy egy számára végrehajtott minimax eljárás (akár más famélységig) más lépést adott kedvezőbbnek, vagy más módszert, értékeket használt, vagy egyszerűen csak hibázott.

Másik fontos észrevétel, hogy a minimax módszerrel nyerő stratégiát lehet meghatározni, ha rendelkezésre áll a teljes játékfa. Ekkor ugyanis a fa levélelemeiről egyértelműen el tudjuk dönteni, melyik játékos a győztes, majd a korábban említettek alapján minden csúcsponthoz tudunk például 1, -1 vagy 0 értéket rendelni, aszerint hogy MAX vagy MIN nyer, esetleg döntetlen az állás. Így végül a gyökérelem is kap értéket, és ha ez például 1, akkor az első játékosnak van nyerő stratégiája. Ha a továbbiakban úgy játszik tovább, hogy mindig az 1 értékű csúcsokra lép, akkor ő kerül ki győztesként. 10. ábrán látható egy minimax kiértékelésű játékfa, amelyben vastag vonallal van jelezve a nyerő stratégia.



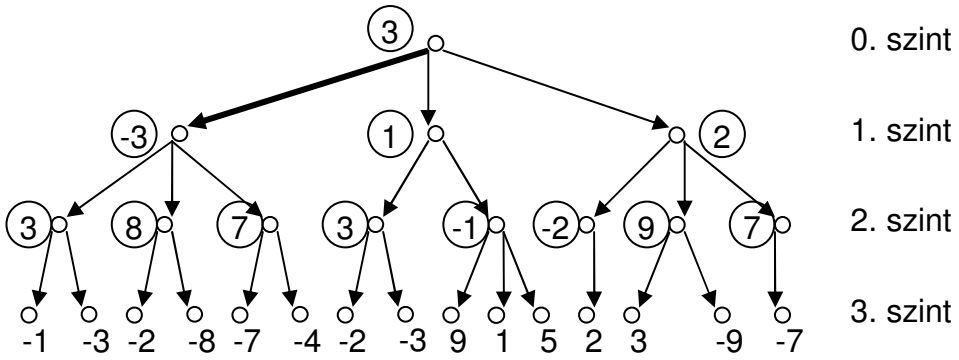
10. ábra: A nyerő stratégia meghatározása minimax eljárással

A nyerő stratégiáról szóló fejezetben alkalmazott módszer ugyanez. 5. ábrán látható fánál is hasonlóképpen jártunk el, csak ott 1 helyett A-t, -1 helyett B-t használtunk.

Negamax eljárás

A minimax eljárás során, a két játékos ellentétes stratégiája úgy jelenik meg a csúcspontok kiválasztásánál, hogy a fa páros (MAX) szintjein maximum, páratan (MIN) szinteken minimum értékűt választjuk. Ezt egyszerűsíthetjük úgy, hogy az eltérő szempont a

csúcspontok értékeinek az előjeleiben jelenjen meg és ezáltal minden szinten elég csak maximumképzést használnunk.
A fa levélelemeire ugyanúgy alkalmazzuk a kiértékelő függvényt, mint a minimax eljárásnál; azzal a különbséggel, hogy ha a levélelemek páratlan (MIN) szinten helyezkednek el, akkor a függvény értékeinek a negáltját kapják meg a csúcspontok. Nem levélelemek esetében, a csúcspontokból kivezető összes állás értékének vesszük az ellentettjét és ezek közül a legnagyobb lesz az adott csúcspont értéke. Ezt szintén a fában alulról felfelé, egészen a gyökérelemig tesszük meg. Majd ennek az értéknek a negáltja megegyezik azzal a csúcsponttal, ahová javasolt lépnünk. (11. ábra)



11. ábra: Egy játékfa negamax kiértékelése

Ezzel a módszerrel ugyanazt az eredményt kapjuk, mint a minimax esetében, előnye hogy számítógépen történő megvalósítása egyszerűbb.

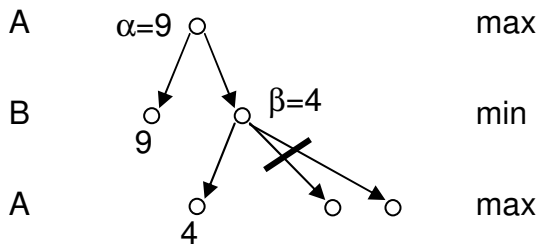
Alfa-béta eljárás

Minimax eljárás során a fa egyes részei gyakran feleslegesen vannak előállítva, ezért célszerű javítani ezen a módszeren.
Alfa-béta eljárással egy csúcs kiterjesztését szakíthatjuk félbe, úgymond levágjuk a további részfát. Ezt abban az esetben tehetjük meg, amikor már a kiterjesztés befejezése előtt kiderül, hogy az éppen kiterjesztés alatt álló csúcsnak nem lesz „beleszólása” a szülő értékének alakulásába.

A két lehetséges vágás:

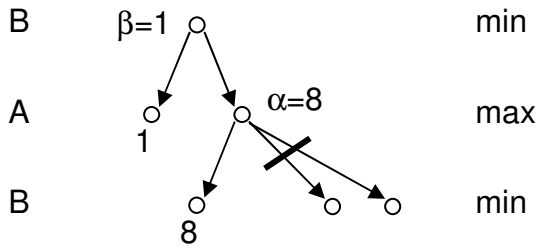
• Alfa-vágás: Aktuális csúcsban minimumot számolunk, a szülőjében maximumot. Az aktuális csúcs kiterjesztését félbeszakíthatjuk, ha a csúcsban eddig kiszámolt minimum (béta) kisebb, mint a szülőben kiszámolt maximum (alfa).

12. ábrán látható egy ilyen helyzet. Az aktuális csúcs értéke 4, hiszen már egy 4 értékű rákövetkezőjét kiterjesztettük. Hasonló okokból a szülő értéke 9. Ettől az aktuális csúcs kisebb, így a kiterjesztést felesleges tovább folytatnunk.



12. ábra: Alfa-vágás

• Béta-vágás: Ugyanaz, mint az alfa-vágás, csak ebben az esteben az aktuális kiterjesztett csúcsban maximumot (alfa), annak szülőjében minimumot (béta) számolunk. 13. ábrán láthatjuk ezt.

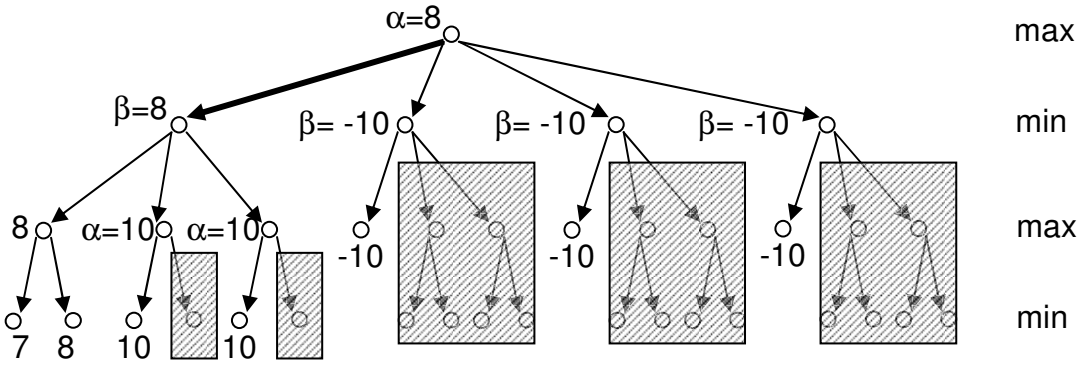


13. ábra: Béta-vágás

Vagyis ha $\alpha \geq \beta$ teljesül, akkor lehet vágni.

Ez a módszer is ugyanazt az eredményt szolgáltatja, mint a minimax módszer, de előnye hogy csökken a játékfa mérete. Eljárás elméletben tovább javítható, ha a csúcsok rendezve vannak és mindig a legjobb lépést vizsgáljuk először. Ezt a valóságban azonban csak megközelíteni tudjuk, hiszen ha ismernénk a legjobb lépést, akkor nem is lenne szükség a játékfa kiértékelésére.

14. ábrán egy konkrét példa látható. Azok a csúcspontok, amelyeket elő sem kellett állítani, a ferde vonalas keretben vannak. Így csak 15db csúcs előállítására volt szükség, vagyis sikerült kivágnunk a fa 57%-át.



14. ábra: Egy játékfa alfa-béta vágás esetén

3. Amőba játék

Egy konkrét kétszemélyes játék bemutatásához az amőbát választottam. Megvalósításához a Java programozási nyelvet használtam. Egy egyszerű grafikus felhasználói felület került kialakításra, amely csupán a játéktábla megjelenítésére szorítkozik, melynek mezői az egér segítségével választhatóak ki.

A játék leírása

Az amőba az egyik legismertebb kétszemélyes játék. Általában egy négyzetrácsos táblán játsszák. A játékosok felváltva helyeznek el egy-egy jelet a tábla üres négyzetébe. Mindkét játékosnak van egy saját jele, ami leggyakrabban X és O. Az a játékos nyer, akinek hamarabb sikerül a saját jeleiből ötöt egymás mellé helyeznie egyenes vonalban. Ez lehet vízszintesen, függőlegesen vagy átlósan.

Nevét onnan kapta, hogy a játszmák végére általában egy szabálytalan ábra alakul ki, amely leginkább az amőba nevű egysejtűre hasonlít. A jelek „terjeszkedésének” csak a tábla mérete szab határt. Döntetlenhez vezet a játék, ha a tábla betelt és egyik játékosnak sem sikerült még az öt jelet egyvonalban elhelyeznie.

Szokták még sok más névvel is illetni ezt a játékot. Sokaknak ismerős lehet még a Gomoku, vagy az 5-in-a-row („5 egy sorban”) elnevezés is. Az eredeti Gomoku Kínából származik, neve viszont a japán nyelvből ered. Amikor versenyszerűen játsszák, a küzdőtér egy 19x19-es go tábla, amelynek nem a mezőire, hanem a pontjaira helyeznek fehér illetve fekete köveket. Az általános szabályok mellé az évszázadok során több különböző plusz megszorító szabály is párosult, ezek alkalmazásáról a játékosok közösen döntenek a játszma előtt.

Állapottér reprezentáció

Állapotok halmaza: Az állapotokban le kell tárolni a lépésen lévő játékost és a teljes táblát. Egy mező értéke abban az esetben nulla, ha oda még egyik játékos sem tette a jelét, különben a játékoshoz rendelt számot tartalmazza.

$$A = \left\{ \left(T_{(12\times12)}, p \right) \middle| \, \forall i, j \; T_{ij} \in \{0,1,2\}, \; p \in \{1,2\} \right\}$$

Kezdőállapot: A kezdőállapotban a tábla teljesen üres és az 1-es (vagyis az ember) játékos kezd.

$$k = \left(\begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 \end{pmatrix}, 1 \right)$$

Célállapotok halmaza: Két okból érhet véget a játék:

- Valamelyik játékosnak kigyűlik egymás mellett az 5 jele.
- Betelik a tábla.

Vagyis $C = C_1 \cup C_2$, ahol

$$C_1 = \left\{ (T,p) \middle| \begin{array}{c} \exists i \exists j \;\; T_{i,j} = T_{i,j+1} = T_{i,j+2} = T_{i,j+3} = T_{i,j+4} = p' \\ \vee \\ \exists i \exists j \;\; T_{i,j} = T_{i+1,j} = T_{i+2,j} = T_{i+3,j} = T_{i+4,j} = p' \\ \vee \\ \exists i \exists j \;\; T_{i,j} = T_{i+1,j+1} = T_{i+2,j+2} = T_{i+3,j+3} = T_{i+4,j+4} = p' \\ \vee \\ \exists i \exists j \;\; T_{i,j} = T_{i+1,j-1} = T_{i+2,j-2} = T_{i+3,j-3} = T_{i+4,j-4} = p' \end{array} \right\}, p' nyer$$

és

$C_2 = \{(T,p) \notin C_1 \mid \neg\exists i,j \ T_{i,j} = 0\}$,*döntetlen*

Operátorok halmaza: Operátorok a tábla egy mezőjére rakják le a játékos jelét.

$O = \{lerak_{x,y} \mid 0 \le x, y \le 11\}$

Alkalmazási előfeltétel: A $lerak_{x,y}$ operátor akkor alkalmazható egy (T,p) állapotra, ha a T (x,y) mezője üres. Vagyis az előfeltétel:

$T_{x,y} = 0$

Alkalmazási függvény: A (T,p) állapotból előálló (T',p') állapot, a $lerak_{x,y}$ operátor alkalmazásával:

$lerak_{x,y}(T,p) = (T',p')$, ahol

$$T'_{i,j} = \begin{cases} p & ,ha\ i = x \wedge j = y \\ T_{i,j} & ,egyébként \end{cases} \quad \text{ahol } 0 \le i,j \le 11$$

$$p' = \begin{cases} 1 & ,ha\ p = 2 \\ 2 & ,ha\ p = 1 \end{cases}$$

A játék implementálása

Az állapottér-reprezentációban is látható, hogy szükségünk van a teljes tábla tárolására és tudnunk kell, hol van még üres mező, hogy hová helyezhetünk még el jelet. Minden lépés, minden egyes lerakott jel egy új helyzetet, vagyis egy új állapotot eredményez. Ezt az *Allapot* osztály valósítja meg, amely a következő változókat tartalmazza:

```
package amoba;

public class Allapot {

    private int jatekos;
    private int gyoztes;
    private int[][] tabla = new int[Amoba.MERET][Amoba.MERET];
    private int[][] ertek_tabla = new int[Amoba.MERET][Amoba.MERET];
    private int ertek;
```

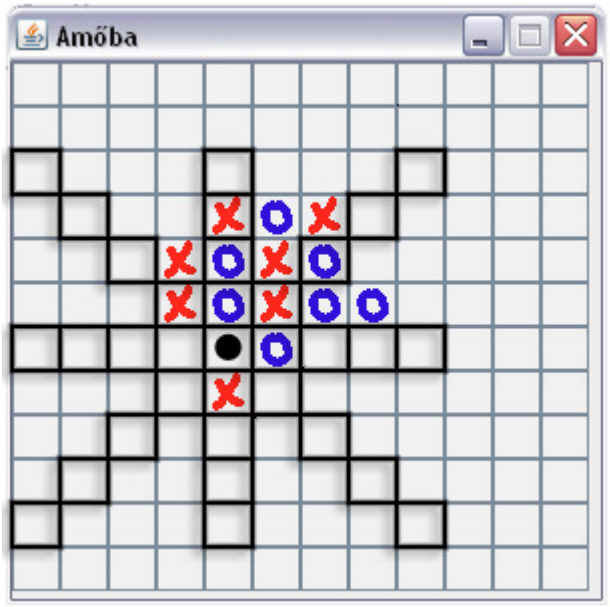
```
public int getJatekos() {
    return jatekos;
}

public void setJatekos(int jatekos) {
    this.jatekos = jatekos;
}

public int valtJatekos() {
    return (jatekos == 1 ? 2 : 1);
}

public int getGyoztes() {
    return gyoztes;
}

public void setGyoztes(int gyoztes) {
    this.gyoztes = gyoztes;
}

public int getErtek() {
    return ertek;
}

public void setErtek(int ertek) {
    this.ertek = ertek;
}

public int getTablaElem(int x, int y) {
    return tabla[x][y];
}

public void setTablaElem(int x, int y, int ertek) {
    tabla[x][y] = ertek;
}

public Allapot() {
    jatekos = 1;
```

```
        gyoztes = 0;
        ertek = 0;
    }

...

}
```

A tábla 12*12 mezőből áll, ennek reprezentálása egy egészeket tartalmazó kétdimenziós tömbbel (*tabla*) történik. Kezdetben minden értéke 0, mivel az egész tábla üres.
Ember játékosnak az X, a gép játékosnak az O jel lett választva. *Jatekos* változó tárolja az éppen lépő játékos jelét. A változó 2-es értéke esetén a gép játékos, 1-es értéke esetén (amely a kezdőállapot) az ember játékos lép. A lépés után pedig ezt az értéket megkapja a tábla –játékos által– választott mezője.
Gyoztes vátozóba abban az esetben kerül az előbbi értékek közül valamelyik, ha a hozzá tartozó játékos nyert, vagyis az adott állásban kialakult az öt jele egymás mellett. Ellenkező esetben az értéke mindvégig 0.
A táblával megegyező méretű *ertek_tabla* tömbben tároljuk minden egyes mező jóságát. Ez annak megfelelően tartalmaz egy számértéket, hogy a tábla azonos mezőjére helyezett jel, mennyire jó választás.
Az egész táblára jellemző értéket az *ertek* változó tartalmazza, amely az előbb említett *ertek_tabla* mezőinek összértéke.

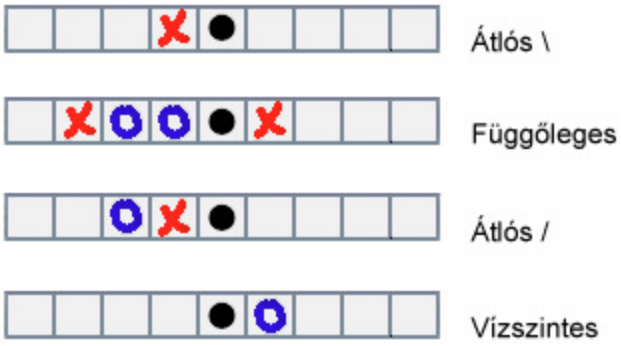
A mezők kiértékelése

Természetesen csak azzal a mezővel foglalkozunk, ahová még egyik játékos sem tett jelet. Ennek környezetében négy irányt kell megvizsgálnunk: függőleges, vízszintes és két átlós. Így egy csillag alakzatot kapunk, melynek a négy ága az előbb említett négy irány.
15. ábrán látható a táblán kiemelve egy ilyen alakzat, melynek a középső, ponttal jelölt mezője az értékelendő mező.



15. ábra: Egy csillag alakzat a táblán

Csillag alakzat ágaira bontva:



16. ábra: Csillag alakzat ágai

Öt mező nagyságú „ablakot” (17. ábra) léptetünk végig mindegyik ágon és mintaillesztést végzünk előre definiált mintákkal.

17. ábra

Természetesen csak azok jöhetnek számításba, ahol ezekben az öt mezőnyi ablaknégyzetekben csak és kizárólag az egyik játékos jele található. Ellenkező esetben –a másik játékos egy korábban elhelyezett jele miatt– már biztosan nem alakulhat ki a győzelemhez szükséges öt azonos jel.
Elméletben összesen 32 variáció alakulhat ki, de azzal az esettel nem foglalkozunk, amikor az öt mezőben egy jel sincs.
A játékosonként 31db minta egy HashMap-ben van eltárolva. A kulcs indexek a mintákat tárolják, a hozzájuk tartozó értékkel pedig egyezés esetén módosul az értékelendő mező.

Az *Elemez* osztály tartalmazza a mintákat:

```
package amoba;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class Elemez {
    private static HashMap mintak1 = new HashMap<String, Integer>();
    private static HashMap mintak2 = new HashMap<String, Integer>();

    static {
        //1es játékos mintái
        mintak1.put("10000", 1); mintak1.put("01000", 1);
        mintak1.put("00100", 1); mintak1.put("00010", 1);
        mintak1.put("00001", 1); //5db 1X

        mintak1.put("11000", 5); mintak1.put("10100", 5);
        mintak1.put("10010", 5); mintak1.put("10001", 5);
        mintak1.put("01100", 5); mintak1.put("01010", 5);
        mintak1.put("01001", 5); mintak1.put("00110", 5);
        mintak1.put("00101", 5); mintak1.put("00011", 5); //10db 2X

        mintak1.put("11100", 25); mintak1.put("11010", 25);
        mintak1.put("11001", 25); mintak1.put("10110", 25);
        mintak1.put("10101", 25); mintak1.put("10011", 25);
        mintak1.put("01110", 25); mintak1.put("01101", 25);
        mintak1.put("01011", 25); mintak1.put("00111", 25); //10db 3X
```

```
        mintak1.put("11110", 100); mintak1.put("11101", 100);
        mintak1.put("11011", 100); mintak1.put("10111", 100);
        mintak1.put("01111", 100); //5db 4X

        mintak1.put("11111", 10000); //1db 5X

        //2es játékos mintái
        ...
    }

    public List<String> csillag(Allapot a, int vizsgaltX, int vizsgaltY,
int vizsgaltJatekos) {
      ...
    }

    public int mezoErteke(List<String> csillag_agai, int vizsgaltJatekos) {
      ...
    }
}
```

Előállítjuk a csillag alakzat ágait úgy, hogy a tábla vizsgált mezőjére ideiglenesen a lépésen lévő játékos jelét helyezzük el. Ezzel elméletben megteremtjük azt a helyzetet, hogy ez a lépés mintha már a végleges döntésünk lett volna. Kiértékelés után ezt a jelet természetesen töröljük.

Minden ág két részből áll elő: Például a balról jobbra átlós irány esetén a középső mezőtől balra fel és jobbra le irányokból. Hasonlóképpen történik a többi ág előállítása is:

```
    public List<String> csillag(Allapot a, int vizsgaltX, int vizsgaltY,
int vizsgaltJatekos) {
        StringBuffer cs = new StringBuffer();
        List<String> csillag_agai = new ArrayList<String>();

        int e = a.getTablaElem(vizsgaltX, vizsgaltY);
        if (vizsgaltJatekos != 0) {
            a.setTablaElem(vizsgaltX, vizsgaltY, vizsgaltJatekos);
        }
```

```
        //átlós \
        int x = vizsgaltX - 1;
        int y = vizsgaltY - 1;
        while (x >= 0 && y >= 0 && vizsgaltX - x <= 4 && vizsgaltY - y <=
4) {
            cs.append(a.getTablaElem(x, y));
            x--;
            y--;
        }
        cs.reverse();
        x = vizsgaltX;
        y = vizsgaltY;
        while (x < Amoba.MERET && y < Amoba.MERET && x - vizsgaltX <= 4 &&
y - vizsgaltY <= 4) {
            cs.append(a.getTablaElem(x, y));
            x++;
            y++;
        }
        csillag_agai.add(cs.toString());
        cs.setLength(0);


        //átlós /
            ...
        // függőleges |
            ...
        //vízszintes -
            ...

        a.setTablaElem(vizsgaltX, vizsgaltY, e);

        return csillag_agai;
    }
```

Az előállított ágak mindegyikén végigléptetjük az 5 négyzet hosszúságú „ablakot” és az vizsgált játékos mintáival keresünk egyezéseket. Amennyiben találtunk ilyeneket, azoknál a mintához tartozó értékeket összegezzük.

A mintaillesztéshez használt mintákat és –konvertálás után– a csillag alakzat ágait is sztringként tároljuk, mivel így az ezekkel történő műveletvégzés egyszerűbb.

```
public int mezoErteke(List<String> csillag_agai, int vizsgaltJatekos) {
    int ertek = 0;

    for (String csillag_ag : csillag_agai) //Mintaillesztés
    {
        int hossz = csillag_ag.length();
        while (hossz >= 5) {
            String key = csillag_ag.substring(hossz - 5, hossz);
            if (vizsgaltJatekos == 1 && mintak1.containsKey(key)) {
                ertek += ((Integer) mintak1.get(key)).intValue();
            }
            if (vizsgaltJatekos == 2 && mintak2.containsKey(key)) {
                ertek += ((Integer) mintak2.get(key)).intValue();

            }
            hossz--;
        }
    }

    return ertek;
}
```

A csillag alakzat egy ágának teljes vizsgálata után, annak fejében, hogy hányszor és hány jel szerepelt, a tábla vizsgált mezője a következő értékekkel módosul:

érték	0szor	1szer	2szer	3szor	4szer	5ször
1 db X illetve O jel	0	1	2	3	4	5
2 db X illetve O jel	0	5	10	15	20	25
3 db X illetve O jel	0	25	50	75	100	125
4 db X illetve O jel	0	100	200	300	400	500
5 db X illetve O jel	0	10000	20000	30000	40000	50000

A 16. ábrán bemutatott példát alapul véve (az X játékos szempontjából vizsgálva) a kiértékelés után eredményül kapott értékek a 18. ábrán láthatóak.

18. ábra: X játékos szempontjából kiértékelt ágak

Gyakran előfordul olyan eset, amikor a táblára helyezett jel egyben „támad” és „véd” is. Ezért mindkét játékos szempontjából kiértékeljük a mezőt és a végleges értéke a kettőnek a különbsége lesz. Ezáltal egy abszolút pontszámhoz jutunk. Minél előnyösebb egy pozíció egy játékosnak, annál nagyobb abszolút értékben ez a szám.
18. ábrán lévő példa vizsgált mezőjének értéke X játékos esetén 33, O játékos esetén 23. Az abszolút pontszám az O és az X játékos értékeinek a különbsége: -10.

Az *Allapot* osztály *modositErtekTablaElem* eljárása határozza meg egy adott mező értékét és módosítja az ezt tartalmazó tábla megfelelő elemét illetve az egész táblát jellemző összértéket:

```
public void modositErtekTablaElem(int xx, int yy) {
        Elemez e = new Elemez();

        int regiErtek = this.ertek_tabla[xx][yy];
        int ujErtek = (e.mezoErteke(e.csillag(this, xx, yy, 2), 2) -
                e.mezoErteke(e.csillag(this, xx, yy, 1), 1));
        this.ertek_tabla[xx][yy] = ujErtek;

        this.ertek -= (regiErtek - ujErtek); // összérték módosítás
    }
```

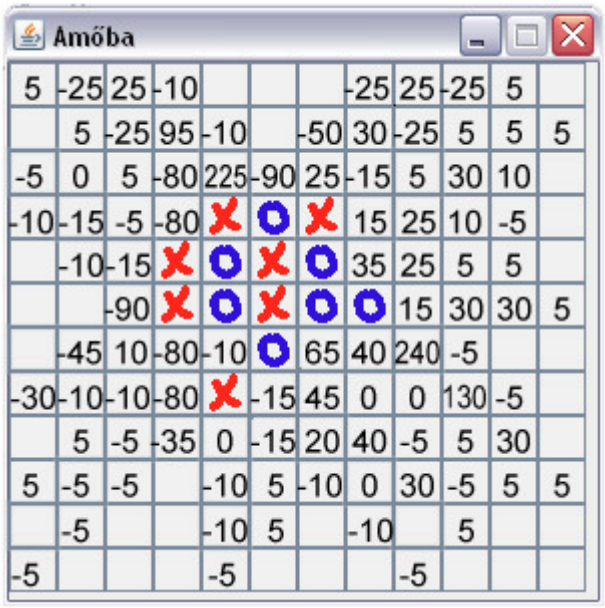
Amikor egy jel a táblára kerül, nem szükséges az összes mezőt újraszámolni, mivel csak a csillag alakú környezetében következik be változás. Ezáltal elegendő csak ezeknek a mezőknek a módosítása.

Az *Allapot* osztály *modositErtekTabla* eljárása csak ezeket a mezőket járja be és a korábban említett *modositErtekTablaElem* frissíti az értékeket. Az egyik átlós irány megvalósítása:

```
public void modositErtekTabla(int x, int y) {
        //Új jel helyének értéke 0 és módosul az összérték is.
        ertek -= ertek_tabla[x][y];
        ertek_tabla[x][y] = 0;

        //átlós \
        int xv = x - 1;
        int yv = y - 1;
        while (xv >= 0 && yv >= 0 && x - xv <= 4 && y - yv <= 4) {
            if (tabla[xv][yv] == 0) {
                modositErtekTablaElem(xv, yv);
            }
            xv--;
            yv--;
        }

        xv = x + 1;
        yv = y + 1;
        while (xv < Amoba.MERET && yv < Amoba.MERET && xv - x <= 4 && yv -
y <= 4) {
            if (tabla[xv][yv] == 0) {
                modositErtekTablaElem(xv, yv);
            }
            xv++;
            yv++;
        }

        //átlós /
        ...
        // függőleges |
        ...
```

```
        //vízszintes -
        ...
    }
```

19. ábrán látható a teljes tábla a kiértékelés után. A példán jól látszik, hogy az X játékosnak a minél negatívabb (pl.: -80, -90); míg az O játékosnak a minél pozitívabb számmal értékelt mezők (pl.: 225, 240) jelentik az ígéretes lépést.
A program normál működése esetén ez nem jelenik meg a képernyőn.



19. ábra: A mezők kiértékelése

A mezők kiválasztása

A játékosok által választható mezőknek az operátorok felelnek meg. Az O*perator* osztály *hovaX* és *hovaY* változója tárolja ezeknek a koordinátáit. A tábla összes lehetséges mezőjét egy statikus vektor tartalmazza:

```
public class Operator {

    private int hovaX;
    private int hovaY;
```

```
    public int getHovaX() {
        return hovaX;
    }

    public int getHovaY() {
        return hovaY;
    }

    public Operator(int x, int y) {
        hovaX = x;
        hovaY = y;
    }

    public static Vector<Operator> operatorok = new Vector<Operator>();


    static {
        for (int i = 0; i < Amoba.MERET; i++) {
            for (int j = 0; j < Amoba.MERET; j++) {
                operatorok.add(new Operator(i, j));

            }
        }
    }
```

Minden esetben feltétel, hogy egy választott mezőn ne legyen már valamelyik játékos jele. Ember játékos esetén –amennyiben ez teljesül– a táblára kerülhet a jele.

A gép esetében viszont van egy plusz előfeltétel is: Azokra az üres mezőkre fektetjük a hangsúlyt, amelyek közelében (maximum 2 mezőnyi távolságra) találhatóak korábban elhelyezett jelek. Itt ugyanis sokkal nagyobb esély van egy előnyösebb helyzet kialakítására (akár a mi pozíciónk erősítésével, akár az ellenfél akadályozásával). A tábla ezen kívül eső területét felesleges megvizsgálni, hiszen érdemi lépéseket ezeken úgysem tudunk alkalmazni és csak a feldolgozási időt tennék hosszabbá.

A gép számára a minimax algoritmus javasol egy operátort. Ezt a *Minimax* eljárás valósítja meg:

```
public abstract class Lepesajanlo {

    int melyseg;
    Operator operator;
    int hasznossag;
}
```

```
public class Minimax extends Lepesajanlo {

    public Minimax(Allapot allapot, int melyseg) {

        if (allapot.celAllapot() == true || melyseg == 0) {
            hasznossag = allapot.getErtek();
        } else {
            if (allapot.getJatekos() == 2) {
                hasznossag = Integer.MIN_VALUE;
            } else {
                hasznossag = Integer.MAX_VALUE;
            }
            for (Operator o : Operator.operatorok) {
                if (o.elofeltetel2(allapot)) {
                    Allapot uj = o.alkalmaz(allapot);
                    Minimax mm = new Minimax(uj, melyseg - 1);

                    if (allapot.getJatekos() == 2 && (mm.hasznossag >
hasznossag)) {
                        operator = o;
                        hasznossag = mm.hasznossag;
                    }

                    if (allapot.getJatekos() == 1 && (mm.hasznossag <
hasznossag)) {
                        operator = o;
                        hasznossag = mm.hasznossag;
                    }
```

```
                }
            }
        }

    }
}
```

Amint megtörtént akár az ember, akár a gép választása, előáll egy új állapot: A táblán immár szerepel a játékos jele és megtörténik a mezőket jellemző értékek módosítása. Ezt követően megvizsgáljuk, hogy a lépésünkkel nem kerültünk-e célállapotba. Ehhez ismételten mintaillesztést végzünk, megvizsgáljuk a négy irányt, hogy tartalmaz-e öt darab azonos jelet:

```
public int vanE_5db(List<String> csillag_agai) {

        for (String csillag_ag : csillag_agai) //Mintaillesztés
        {
            if (csillag_ag.length() >= 5 && (csillag_ag.contains("11111")))
{
                return 1;
            }
            if (csillag_ag.length() >= 5 && csillag_ag.contains("22222")) {

                return 2;
            }

        }
        return 0;
    }
```

Amennyiben egyezést találunk, a játék véget ér és kiírjuk, hogy ki lett a győztes játékos.

4. Összefoglalás

A mesterséges intelligencia feladatai általában nem egyszerűek. Az ember számára is okoz fejtörést, a számítógépek számára ez azonban hatványozottan igaz.
Már a tudományág kezdeti szakaszától kezdve a kutatók szemlélete egyre inkább a speciális feladatok megoldása felé fordul, mint például a kétszemélyes játékok.
A XX. század közepén jelent meg Neumann János és Oskar Morgenstern –a témában alapműnek számító– könyve, melyben a játékelmélet alapjait rakják le. Már ebben részletesen kifejtésre kerül a minimax algoritmus, mely a későbbiekben a számítógépes programok egyik alapvető algoritmusává vált.
A dolgozat célja részben ezen elméleti alapok, erre épülő technikák, algoritmusok és ezek hibáinak javításával kialakult újabb változatok bemutatása; valamint ezeket az ismeretek felhasználva egy egyszerű játékprogram létrehozása volt.

Irodalomjegyzék

Fekete István–Gregorics Tibor–Nagy Sára: Bevezetés a mesterséges intelligenciába. Budapest: LSI Oktatóközpont a Mikroelektronika Kultúrájáért Alapítvány, 1990.

Starkné Werner Ágnes: Mesterséges intelligencia. Veszprém: Veszprémi Egyetemi Kiadó, 2004.

Csákány Béla: Diszkrét matematikai játékok. Szeged: Polygon, 2005.

Kóczy Á. László: A Neumann-féle játékelmélet. Közgazdasági Szemle, LIII. évf. 2006. 1. sz., 31-34. p.

Dr. Várterész Magda: Mesterséges intelligencia 1
http://www.inf.unideb.hu/~varteres/mi/tartalom.htm (2009.10.01.)

Dr. Kovásznai Gergely-Dr. Kusper Gábor: Mesterséges intelligencia
http://files.szt.ektf.hu/dl.php?file=files%2FTan%C3%A1ri+Megoszt%C3%A1sok%2FKov%C3%A1sznai+Gergely%2FMesters%C3%A9ges+intelligencia%2Fmesterseges_intelligencia.v1_0.pdf (2009.10.10.)

Pongrácz Gábor: Amőba típusú játékok
http://www.stud.u-szeged.hu/Pongracz.Gabor/amoba.pdf (2009.11.01.)