Debreceni Egyetem
Informatika Kar

Egy kétszemélyes logikai játék számítógépes megvalósítása

Témavezető:
Mecsei Zoltán
Egyetemi tanársegéd

Készítette:
Kovács Katalin
Programtervező matematikus

Debrecen
2010

Tartalomjegyzék

Bevezetés

Amikor választanom kellett, hogy miből készítsem el a dolgozatomat, hamar a Mesterséges intelligencia 1. című tárgy tudománykörében kezdtem el keresni a megfelelő témacímet. A tantárgy hallgatása elegendően széles és nem utolsó sorban érdekes ismeretanyagot tárt fel számomra ahhoz, hogy ne tartsam sem túl egyszerűnek, sem túl bonyolultnak a diplomamunkám összeállítását ebben a témakörben.

A játék már ősidők óta az emberiség kedvelt időtöltése, amely képes összekovácsolni a közösségeket. Szórakoztató és sok közülük fejleszti az intellektuális készségeket. Ezen diplomadolgozatban az általam bemutatásra kerülő kétszemélyes logikai játék talán nem örvend olyan népszerűségnek, mint a Sakk, vagy nem olyan közismert térségünkben, mint a Dáma, vagy a Malom. Ellenben remélem, hogy idővel elfoglalja méltó helyét a táblajátékok rangsorában Európa-szerte mindenütt. Véletlenül akadtam rá az Interneten a Kalah nevű játékra és a szabályok gyors átfutása után nekikezdtem játszani. Hamar rájöttem, hogy nem olyan egyszerű, mint ahogy azt először hittem és elkezdtem jobban utána olvasni a játékszabálynak. Sikerült részletes angol nyelvű leírást találnom róla és felfedeztem, hogy bár „csak” számolni kell tudni, hogy játszhassak, a nyeréshez nem elegendő: előre gondolkodásra, stratégiára is szükség van. Magyar nyelvű irodalom alig található a Mancala (ezt az elnevezést általában összefoglalóan használják) típusú játékokról, de szerencsére külföldi elektronikus források akadtak bőven. Ugyan több száz variációja létezik ennek az ősi időkbe visszatekintő táblajátéknak (elterjedését, történetét a következő fejezetben az érdekesség és ismeretterjesztés kedvéért bővebben közlöm), hosszas mérlegelés után egyet kiválasztottam és ez az Abapa. Azért ezt, mert nem szerettem volna a többek által már leprogramozott és több honlapon online játszható Kalah-t, így helyette a nemzetközi versenyeken, többé-kevésbé egységes játékszabályokkal bíró verzió mellett döntöttem. Számomra teljesen ismeretlen volt ezelőtt ez a játékcsalád, habár hosszú múltra tekint vissza az emberiség történelmében. Mindezeket szem előtt tartva érdekes kihívásnak éreztem, hogy leprogramozzam és az is vezérelt, hogy én előttem valószínűleg nem sokan foglalkoztak még Magyarországon az

Abapa, vagy egyáltalán bármely mancala-típusú játéknak a számítógépes megvalósításával, de a szabályainak magyar nyelvű közlése sem jellemző. Egyedül a játékboltok kínálatában találhatjuk meg a Mancala asztali társasjátékot.

Dolgozatomban bemutatásra kerül az Abapa szabályrendszere, a számítógépes megvalósításához szükséges mesterséges intelligenciai alapismeretek, az állapottér-reprezentációja, a győzelemhez vezető stratégia, a lépésajánló algoritmusok és a hatékonyságfüggvények, melyek szükségesek voltak ahhoz, hogy megfelelően működő játékprogramot készíthessek. A megvalósított program egy Java nyelven írott asztali alkalmazás, amiből az utolsó fejezetben kódrészleteket ismertetek. Az irodalom jegyzék végén található az (1.1) fejezethet tartozó rövid szómagyarázatot. A függelék pedig a Játékvilág c. elektronikus könyvből tartalmaz egy részletet a Vari (awari, mankola, mancala) játékszabályáról.

[1. kép]

1. A „gödör és kavics” játékokról

Az Oware – a közismertebb, közel-keleti elnevezéssel Mancala - a legrégibb, a mai napig a világon széles körben elterjedt táblajáték. Az úgynevezett „gödör és kavics” – más forrásokban „számol és elfog” – játékok osztályába tartozik, melyek létezése hétezer évre visszamenőleg kimutatható. A pontos eredete elveszett a történelem során, de őshazájának Afrika tekinthető.

1.1 Történetük

A Time magazin 1963. június 14-ei számában közli ezen játékok ősi származását, és hogy milyen széles körben terjedtek el: „Véseteket találtak egy óriási sziklatömbön az ősi szír városban, Aleppo-ban. Két sorban hat sekély gödröt és két mélyebbet a sorok végén. Ugyanez a forma van az egyiptomi Karnak és Luxor templomának falába vésve és megtalálható a Nílus-völgyében egy korai sír falfestményén is. De látható még az athéni Theseumban és az ókori világ karavánútjainak mentén sziklák szélébe karcolva. Ma ugyanilyen lyukak, mélyedések találhatók Ázsiában és Afrikában mindenfelé, belevájva a puszta földbe, kimetszve fából, vagy aranyberakású elefántcsontból.”[1] A magazin cikkében közli, hogy William Champion, a játék Amerikában elterjedt modern verziójának, a Kalah-nak a megalkotója, egyben a játék kutatója felfedezett egy festményt egy urnán, amin éppen Aiasz és Akhilleusz Trója ostroma alatt játszik egy ilyen játékkal. Ezek és még sok más régészeti lelet tanúsítja a játék ókori eredetét. Champion szerint a játék mintegy hétezer éves múltra tekint vissza.

A „gödör és kavics” játékok elterjedtek Afrikából a földkerekség különböző tájaira. A rabszolga kereskedelem szolgálhat magyarázatul, hogy miért ugyanazok a szabályok Nyugat-Afrikában és a karibi térségben. Az egész világon játsszák a Távol- és Közel-Kelettől, Ázsián és Afrikán keresztül a Karib-tengerig.

Összességében több mint 300 különböző szabályrendszerű verziója van a „gödör és kavics” játékoknak. Négy fő típus különböztethető meg aszerint, hogy hány sorból áll a tábla. A kétsorosak a legkedveltebbek, ilyen például az Oware. Ezek a játékok különböző veremszámúak lehetnek 5-14-ig, de a hatlyukas táblák a legáltalánosabbak, ilyennel játszanak túlnyomórészt Nyugat-Afrikában és a Karibikon. A Bao a négysoros fajtára példa [2. kép] és Kelet-, illetve Dél-Afrikában használják. A háromsorosak - úgy mint a Gebeta - a legkevésbé népszerűek és tudomásunk szerint csak Etiópia és Eritrea területére korlátozódik, de létezik egysoros játék is, az ázsiai eredetű tchuka ruma. Elhelyezkedhet a sorokon kívül még a táblán két nagyobb mélyedés is, amelybe az adott szabályoktól függően kerülhetnek kavicsok, a játékosok ezekbe gyűjtik az elfogott köveket, vagy csak egyszerűen maguk mellé.

Az Oware vagy ezen név variációi (Warri, Awalé, Awele, Awari, stb.) a legkedveltebb elnevezései a 2x6-os formációnak. Az Oware nevét az akan nyelven beszélő ghánaiaktól kapta. Bár ez egy általános megnevezés és legalább három játékverziót takar. A hagyományosan és jelenleg is a versenyeken használt típust Abapának nevezik, melynek szó szerinti fordítása: „a jó változat”.

[2. kép]

Ezt mindig is a felnőttek játékának tekintették és azok játsszák, akik oklevelet szereztek a Nam-nam nevű gyermekverzióból.

A ghánai társadalomban mindig is kiugró szerepet töltött be az Oware: fő vonatkozásban úgy is, mint a korábbi Denkyira[2], majd az Ashanti[3] királyok játéka. Időnként néhány afrikai közösségben a koronázási szertartás részét képezte. A vezetők főnöki képességeik élesítésére használták és arra, hogy jobban megismerjék saját alattvalóikat.

Évezredekig az Oware-t a gyermekek oktatására használták Afrikában és más világrészeken is. Az elmúlt négyszáz évben ezen történelmi jellegéből kiindulva keltette fel az európaiak figyelmét. Sok akadémikus és pedagógus vitte hírét a játék nevelő hatású jelentőségének,

felszólalt az iskolákban való hasznosítása mellett. Manapság elismert jelentőségű oktatási eszköz, amelyet egyre több iskolában alkalmaznak.

Nemzetközi viadalokat rendeznek minden évben a Karib-térségben (Antigua-n) és Európában (Franciaországban) - illetve tervek szerint ehhez Ghána is csatlakozni fog. Helyi versenyeket rendszeresen tartanak sok afrikai, karibi és európai államban és az országok listája évről évre gyarapszik. A „National Schools Tournament" minden évben megrendezésre kerül az Egyesült Királyságban.

Végül a számítógépek és az Internet korának beköszöntével, ma bőségesen találhatunk számítógépen játszható verziókat. [Erre példa az Irodalomjegyzék Online játék linkek címszó alatt található.]

1.2. Egy játék általános leírása és játékszabályai: Az Abapa

A „gödör és kavics" játékok történeti áttekintése után az általam implenetálásra kiválasztott variáns, az Abapa szabályait ismertetem ebben a fejezetben. A játék megkezdéséhez csupán két játékos kell. Még egyedi táblára sincs szükség és bábukra sem. Ahhoz, hogy otthon játszhassunk megteszi néhány kivájt mélyedés vagy tálka is, illetve adott számú kavicsot, vagy növényi magvat kell gyűjtenünk, amik a bábukat helyettesítik és már kezdhetjük is. (A munkámban a mag terminológiát vettem át a forrásokból.)

A tábla:

A táblán két sorban hat-hat mélyedés található, mindegyikben négy-négy mag van induláskor, összesen tehát 48 mag vesz részt a játékban. (lsd.: [1.kép]) A magok között nem teszünk különbséget. A mélyedéseket klasszikusan házaknak nevezik. Azt a sor házat, amely a játékoshoz közelebb áll a játékos saját oldalának hívjuk, míg a másik sor az ellenfél oldala. Továbbá mindkét játékos oldalán van egy-egy gyűjtőrekesz – egy számláló - a learatott magok nyilvántartásához, ami kezdetben 0-0. A könnyebbség kedvéért a táblát mindig a

kezdő játékos nézőpontjából szemléljük, és nem fordítjuk meg, ha az ellenfele következik. Így a kezdő házainak sorszáma balról jobbra 1, …, 6, az ellenfélé pedig jobbról balra 7, …, 12.

A játék célja:

A két játékos felváltva lép. A feladat, hogy annyi magot gyűjtsenek össze, amennyit csak tudnak. A játék addig folyik, amíg egyikőjük 25 darab, vagy ennél több magot nem szerez. Mivel páros számú mag vesz részt a játékban, ezért a döntetlennek (24-24 elkapott mag) is van esélye.

A lépések:

a) Amikor valaki sorra kerül, választ a saját oldalán lévő hat ház közül és kiveszi a benne található összes magot. Ez a mozzanat a *felmarkolás*.

b) Órajárásnak ellentétes irányban a játékos a soron következő házakba sorra ledob egyet, amíg el nem fogynak a magok a kezéből. Ezt a műveletsort *vetés*nek nevezzük. A 12-es ház után újra az 1-es következik. Mindig csak egyszerre egyet dobhat egy házba és nem hagyhat ki házat kivéve a c) eset..

c) Amikor a játékos egy házból több mint 11 magot markol fel, akkor mindegyik házba letesz egyet, mindaddig, amíg ahhoz nem ér, amelyből kiindult. Ebbe a házba nem szabad dobnia – ez a *kihagyás* művelet. A maradék mag vagy magok elvetését az üresen hagyott ház rákövetkezőjétől kell folytatnia. (lsd: ábrák)

[1.a. ábra]

[1.b. ábra]

d) Az *aratás* a játék lényegi mozzanata. Ezzel fogják el a játékosok a magokat és teszik be a gyűjtőjükbe. Ez akkor következik be, amikor egy játékos az utolsónak elvetett maggal az ellenfele térfelén kettővé, vagy hárommá egészít ki egy házat. [2.ábra] Kiemelném újra, hogy aratni csak az ellenfél oldalán lehet. Ha viszont valaki kettővé vagy hárommá egészít ki egy házat, de van még mag a kezében, akkor nem arathatja le.

[2. ábra]: B az utolsónak vetett maggal arat 2-t.

e) *Többszörös aratás*nak nevezzük, ha egy játékos úgy arat, hogy az ellenfele oldalán az elfogott házat közvetlenül megelőzően van vagy vannak még kettőt vagy hármat tartalmazó házak, akkor azokat is learatja. [3.a. és 3.b. ábra] Így legfeljebb öt ház tartalma vihető. Az aratási sor megszakadása esetén nem lehet a szakadást megelőző 2-3-at tartalmazó házakat learatni. Ezt a helyzetet szemlélteti a [4.a. és 4.b. ábra].

[3.a. ábra]

[3.b. ábra]: „A” aratott három házat.

[4.a. ábra]

[4.b. ábra]: „A" nem arathatta a 8-as házat.

f) Egyes helyeken úgy játsszák a játékot, hogy az ellenfél minden háza aratható – ezt nevezik „*Grand Slam*"-nek. Viszont nemzetközi versenyeken általában az a bevett szokás, hogy a Grand Slam műveletsor szabályos, ámde nem jár együtt aratással, tehát a táblán kell ilyenkor hagyni az elvetett magokat. [5.a. és 5.b. ábra] Így ugyanis az ellenfél folytatni tudja a játékot, mivel nem ürülnek ki a házai. Más szabályrendszerek engedik az aratást, de vagy az első, vagy az utolsó ház tartalmát a táblán kell hagyni, hogy lépni tudjon a kifosztott játékos. Máshol viszont tiltott Grand Slam lépést tenni. Én a versenyeken használt módosított Grand Slam változatot választottam.

[5.a. ábra]

[5.b. ábra]: „B" Grand Slam-et lépett.

g) Abban az esetben, ha egy játékosnak minden háza kiürül a lépése következtében, az ellenfele köteles olyan lépést tenni, amivel magokat juttat át a térfelére. Ezt a műveletet *etetés*nek nevezzük.

h) Előfordulhat olyan helyzet is, amikor az egyik játékosnak elfogy az összes magja, mint az előbb, de az ellenfele nem képes őt etetni. Ilyenkor az ellenfél egyszerűen begyűjti a térfelén fennmaradt magokat, azaz *üríti* a házait és a játéknak vége.

i) Kialakulhat az úgy nevezett *végtelen kör* nevű végjáték, amikor a játékosok nem tudnak úgy lépni, hogy bármelyikük learathassa a fennmaradó magokat, lépéseik csak körbejárást eredményeznének. Ez esetben a felek megegyeznek a játék felfüggesztéséről és megosztoznak a táblán maradt magokon. Mindenki a saját házai tartalmát viszi és a játék véget ér.

2. Út a számítógépes megvalósítás felé

2.1. Az Abapa besorolása a mesterséges intelligencia területére

A Mesterséges intelligenciában a játékokat különböző osztályokba sorolják. Egyfajta osztályozást jelent például az, hogy hány fő vesz részt a játékban: egy, kettő, vagy több. Az Abapa a *kétszemélyes* játékok családjába tartozik, ahol a játékosok egymás után felváltva következnek lépni. Vannak továbbá az úgynevezett *diszkrét* játékok, ahol egy játszma egyik állásból másik állásba vivő lépések sorozatából áll. Ezen kívül fontos csoportot alkotnak a *véges* játékok, ami azt jelenti, hogy a játékosoknak az egyes állásokban véges sok lehetséges lépése van, és a játszmák véges sok lépés után véget érnek. A Mesterséges intelligencia a kétszemélyes játékok közül az úgynevezett stratégiai játékokkal foglalkozik. A nem stratégiai sztochasztikus, más néven szerencsejátékok esetén (mint amilyenek általában a kártyajátékok) nagy szerep jut a véletlennek. Az olyan játékokat, ahol a véletlen nincs jelen: *determinisztikus* játékoknak hívjuk. A szerencsejátékokban a játékosok elől rejtve maradnak bizonyos információk (ilyenek például az ellenfél lapjai), amelyek ismerete nélkül nem lehetünk képesek stratégiát kialakítani, kiszámítani előre a lehetséges lépéseinket. Újabb osztályt alkotnak tehát a *teljes információjú* játékok, ahol a játékosok a játék folyamán a játékkal

kapcsolatos összes információ birtokában vannak. Ha pedig a játék szabályai olyanok, hogy a játékosok nyereségeinek és veszteségeinek az összege nulla, akkor beszélünk a *zéró-összegű* játékokról. Ez annyit tesz, hogy amely lépés az egyik játékos számára valamennyi pozitív hasznossággal bír, az a lépés a másik számára éppen ugyanannyi negatív hasznosságot jelent.

Az Abapa játék tehát nem más, mint egy kétszemélyes, véges, teljes információjú, diszkrét, determinisztikus, zéró-összegű stratégiai játék.

2.2 Kétszemélyes stratégiai játékok - alapismeretek

2.2.1 Mi az állapottér-reprezentáció?

Ha a mesterséges intelligencia témakörében foglalkozni szeretnénk egy problémával, például egy játék „leprogramozásával", akkor először is meg kell valahogy próbálnunk leírni ezt a problémát. Ehhez szükségünk van arra, hogy elképzeljük magunk előtt a problémát meghatározó legfőbb elemeket, tehát hogy mik azok a jellemzők, leíró tényezők, amelyekkel modellezni tudjuk a probléma világát. Ezek a jellemzők rendre egy-egy értéket vesznek fel, amellyel jellemzik a probléma adott pillanatát. Ha n darab ilyen leíró elemet találtunk, amelyek rendre h_1, h_2, …, h_n értékekkel rendelkeznek, akkor azt mondhatjuk, hogy a problémánk épp a (h_1, h_2, …, h_n) érték n-essel leírt *állapot*ban van. Az összes ilyen állapotoknak a halmazát nevezzük *állapottér*nek. Vegyük az n jellemző közül az i-ediket és az általa felvehető összes értéket jelölje a H_i halmaz. Ekkor a probléma világának állapotai elemei a $H = H_1 \times H_2 \times \ldots \times H_n$ halmaznak. A H-t a játékállások halmazának nevezzük, a játékosok halmazát J-vel, az állapotteret $\mathcal{A}$ –val jelöljük. A modell tervezésekor oda kell figyelni, hogy vajon minden érték n-es valódi, létező állapot e. Ezt a szűrést a kényszerfeltételek leírásával tehetjük meg. Kiemelt szerepe van a reprezentációban a *kezdőállapot*nak és fontos meghatározni azt is, hogy milyen állapotot nevezünk a játék végének. Ezt *végállapot*nak hívjuk, amit az állapotváltoztatásokkal kívánunk elérni. Végállapot több is lehet a játékszabályoktól függően, ezért halmazzal jelöljük és ekkor meg

kell nevezni azt a játékost is, aki nyert. A játékosok egymás után felváltva lépnek és ezen állapotváltozásokat leképezések, úgynevezett *operátorok* segítségével érhetjük el. De nem minden operátor alkalmazható mindegyik állapotra. Így meg kell adnunk a különböző operátoroknál azok értelmezési tartományát, tehát az operátor-alkalmazási előfeltételeket. Ha úgy találjuk, hogy alkalmazható egy operátor egy állapotra, akkor azt is meg kell állapítani, hogy milyen állapotváltozást okoz az operátor alkalmazása, azaz milyen új állapot keletkezik. Ezt a problémamodellezési-technikát *állapottér-reprezentáció*nak nevezzük és az
$<A, kezdő, V, O>$ négyessel jelöljük.

2.2.1.1 Az Abapa állapottér-reprezentációja

Az általános bevezető után most az általam választott játék konkrét reprezentációja következik:

Halmazok:

$H_1 = \{0, \ldots, 19\}$, $H_2 = \{0, \ldots, 19\}$, $H_3 = \{0, \ldots, 19\}$, $H_4 = \{0, \ldots, 19\}$, $H_5 = \{0, \ldots, 19\}$, $H_6 = \{0, \ldots, 19\}$, $H_7 = \{0, \ldots, 19\}$, $H_8 = \{0, \ldots, 19\}$, $H_9 = \{0, \ldots, 19\}$, $H_{10} = \{0, \ldots, 19\}$, $H_{11} = \{0, \ldots, 19\}$, $H_{12} = \{0, \ldots, 19\}$, $H_{A\_gyűjtött} = \{0, \ldots, 39\}$, $H_{B\_gyűjtött} = \{0, \ldots, 39\}$, ahol a számokkal indexelt halmazok a 2×6 darab házat jelölik, a további kettő pedig a játékosok által összegyűjtött magokat tartalmazzák.

A játékállások halmaza:
$H = H_1 \times H_2 \times H_3 \times H_4 \times H_5 \times H_6 \times H_7 \times H_8 \times H_9 \times H_{10} \times H_{11} \times H_{12} \times H_{A\_gyűjtött} \times H_{B\_gyűjtött}$

H = { (4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0), (0, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 4, 0, 0), (4, 0, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 0, 0), (4, 4, 0, 5, 5, 5, 5, 4, 4, 4, 4, 4, 0, 0), (4, 4, 4, 0, 5, 5, 5, 5, 4, 4, 4, 4, 0,0), (4, 4, 4, 4, 0, 5, 5, 5, 5, 4, 4, 4, 0, 0), (4, 4, 4, 4, 4, 0, 5, 5, 5, 5, 4, 4, 0, 0), (0, 5, 5, 5, 5, 4, 0, 5, 5, 5, 5, 4, 0, 0), (0, 5, 5, 5, 5, 4, 4, 0, 5, 5, 5, 5, 0, 0), (1, 5, 5, 5, 5, 4, 4, 4, 0, 5, 5, 5, 0, 0),

(1, 6, 5, 5, 5, 4, 4, 4, 4, 0, 5, 5, 0, 0), (1, 6, 6, 5, 5, 4, 4, 4, 4, 4, 0, 5, 0, 0), (1, 6, 6, 6, 5, 4, 4, 4, 4, 4, 4, 0, 0, 0), …, (0, 1, 2, 0, 1, 2, 0, 0, 1, 0, 4, 3, 12, 12), (1, 0, 0, 0, 1, 2, 0, 0, 1, 0, 4, 0, 12, 27),…, (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 25, 22), (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 25, 23), (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 23, 25), (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 24, 24) }

A J ∈ {A, B}, így $\mathcal{A} = H \times J$,

azaz $\mathcal{A} = \{ (h, j) \mid (h, j) \in H \times J$ és kényszerfeltételek(h, j) }

Kényszerfeltételek:

A 12 db ház és a két gyűjtő összege pontosan 48. A házakban összesen maximum 48 mag lehet, míg a gyűjtőkbe összesen szintén maximum 48 kerülhet.

Kezdőállapot:

$kezdő = \{ ((4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0), A) \} \in \mathcal{A}$

Végállapotok halmaza:

$\mathcal{V} = \{ ((x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, n, v), j),$

$((x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, v, n), j),$

$((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, n, v), j),$

$((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, v, n), j),$

$((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 24, 24), j) \} \in \mathcal{A}$

A $\Sigma x_i \in \{0,\ldots,23\}$, az $n \in \{25,\ldots,39\}$, az $v \in \{0,\ldots,23\}$, j = {A, B}.
Ekkor az első esetben A, a másodikban B nyert. A harmadik-negyedikben pedig kiürültek a házak és vagy A vagy B nyert. Az utolsóban döntetlen az eredmény.

Operátorok:

O = { Felmarkol(honnan, mennyit) } $\in \mathcal{A}$

Szükség van néhány segédfüggvényre és konstansra a könnyebbség kedvéért:

- A_térfeléről_markolás = { igaz, ha honnan $\in$ { 1,…6 }
  { hamis, ha honnan $\in$ { 7,…12 }
- B_térfeléről_markolás = $\neg$ A_térfeléről_markolás
- A_térfelére_vetés(i) = { igaz, ha i $\in$ { 1,…6 }
  { hamis, ha i $\in$ { 7,…12 }
- B_térfelére_vetés(i) = $\neg$ A_térfelére_vetés (i)
- utolsó_mag = (honnan + mennyit)%12 + (mennyit /12)
- maradék_magok_A = { Σh_i | i = 1, …, 6 }
- maradék_magok_B = { Σh_i | i = 7, …, 12 }
- Vethet(k) = { 12, ha k == 12
  k%12, különben }
- Arat(i,A) = { h_i, ha (j = A) $\wedge$ B_térfelére_vetés(i) $\wedge$ (h_i = 2 $\vee$ 3)
  0, egyébként}
- Arat(i,B) = { h_i, ha (j = B) $\wedge$ A_térfelére_vetés(i) $\wedge$ (h_i = 2 $\vee$ 3)
  0, egyébként }
- Arat_többször(i,A) = { Arat(utolsó_mag,A) + Arat(utolsó_mag-i,A) | i = 1…4}
- Arat_többször(i,B) = { Arat(utolsó_mag,B) + Arat(utolsó_mag-i,A) | i = 1…4}
- Grand_Slam(i,A) = { igaz, ha Arat_többször(i,A) $\wedge$ (maradék_magok_B=0)
  hamis, egyébként }
- Grand_Slam(i,B) = { igaz, ha Arat_többször(i,B) $\wedge$ (maradék_magok_A=0)
  hamis, egyébként }
- Etet(A) = { igaz, ha B_térfelére_vetés(utolsó_mag) $\vee$ (mennyit > 11)
  hamis, egyébként}
- Etet(B) = { igaz, ha A_térfelére_vetés(utolsó_mag) $\vee$ (mennyit > 11))
  hamis, egyébként}

- Végtelen_kör = { ha maradék_magok_A < 4 ∧ ha maradék_magok_B < 4 }
- Ürít(A) = { igaz, ha A_térfelére_vetés(i+h_i) | i = 1,…6
  hamis, egyébként}
- Ürít(B) = { igaz, ha B_térfelére_vetés(i+h_i) | i =7,…12
  hamis, egyébként}

Operátor-alkalmazási előfeltételek:

A Felmarkol(honnan, mennyit) operátor alkalmazható egy ((h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9, h_{10}, h_{11}, h_{12}, $h_{A\_gyűjtött}$, $h_{B\_gyűjtött}$) j) ∈ $\mathcal{A}$ állapotra, ha teljesülnek a következő alkalmazási előfeltételek:

1) (ha (j=A) ∧ A_térfeléről_markolás ∧ (0 < mennyit ∧ mennyit = h_{honnan}) ∧
2) (ha (maradék_magok_B=0) ∧ (Etet(A) ∨ Ürít(A)))) ∨
3) (ha (j=B) ∧ B_térfeléről_markolás ∧ (0 < mennyit ∧ mennyit = h_{honnan}) ∧
4) (ha (maradék_magok_A=0) ∧ (Etet(B) ∨ Ürít(B))))

Operátor-alkalmazás hatása:

A Felmarkol(honnan, mennyit) operátort egy ((h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9, h_{10}, h_{11}, h_{12}, $h_{A\_gyűjtött}$, $h_{B\_gyűjtött}$) j) ∈ $\mathcal{A}$ állapotra alkalmazva, a következőképpen definiált ((h_1', h_2', h_3', h_4', h_5', h_6', h_7', h_8', h_9', h_{10}', h_{11}', h_{12}', $h_{A\_gyűjtött}$', $h_{B\_gyűjtött}$') j') ∈ $\mathcal{A}$ állapotot kapjuk:

j' = { B, ha j = A
{ A, ha j = B
h_i' = { { 0, - ha i = honnan ∨
- ha Arat_többször(i,A) ∨
- ha Arat_többször(i,B) ∨
- ha Végtelen_kör | i = 1,..., 12 ∨
- ha Ürít(A) | i = 1,...,6 ∨
- ha Ürít(B) | i = 7,...,12 }

$\{ h_i + 1,$ - ha $i = Vethet(honnan+m) \wedge i \neq honnan \mid m = 1,..., mennyit \vee$

- ha Grand_Slam(i,A) $\vee$
- ha Grand_Slam(i,B) $\}$

$h_{A\_gyűjtött}' = \{ h_{A\_gyűjtött} + x,$ - ha x = Arat_többször(i,A) $\vee$

- ha Végtelen_kör, akkor x = maradék_magok_A $\vee$
- ha Ürít(A), akkor x = maradék_magok_A

$h_{B\_gyűjtött}' = \{ h_{B\_gyűjtött} + x,$ - ha x = Arat_többször(i,B) $\vee$

- ha Végtelen_kör, akkor x = maradék_magok_B $\vee$
- ha Ürít(B), akkor x = maradék_magok_B

2.2.2 Gráf-reprezentáció

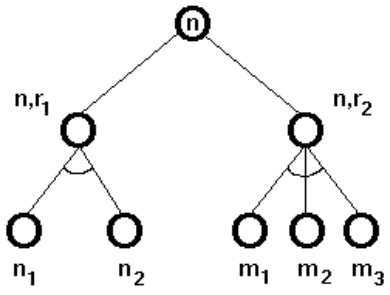
2.2.2.1 Az állapottér-reprezentáció ábrázolása állapottér-gráffal

Az állapottér-reprezentáción kívül van egy másik eszköz is, amit szívesen használunk a játékok modellezése során, mert szemléletes. A problémákat minden esetben átfogalmazhatjuk egy irányított gráfban történő útkeresési feladattá. Legyen a probléma állapottér-reprezentáció az $<\mathcal{A}, kezdő, V, O>$. Szemléltessük a játék lehetséges állásait gráfbeli csúcsokkal, tehát minden csúcs egy-egy állapotot jelöljön. A kezdőállapotot szemléltető csúcsot *startcsúcs*nak, a végállapototokat jelölőeket pedig *terminális csúcs*oknak nevezzük. Ezek a játék végső fázisait jelentik, ahol valamelyik játékos nyert vagy döntetlen alakult ki. Jelölje az a, a' $\in \mathcal{A}$ állapotokat szemléltető csúcsokat n_a és $n_{a'}$. Minden n_a-ból $n_{a'}$ csúcsba irányított él vezet a gráfban, ha a-ból közvetlenül elérhető az a' állapot. Ez azt jeleneti, hogy ha az a állapotot tartalmazó n_a-ra alkalmazható egy o operátor és azt meg is lépjük, akkor az újonnan keletkező a' állapotot tartalmazó $n_{a'}$ csúcs közvetlenül elérhető az n_a-ból méghozzá az o él mentén. A startcsúcs után következő csúcsok azok az állások, amelyek a kezdőállapotból az összes alkalmazható operátor alkalmazását követően egy lépésben előálltak. A jobb átláthatóság miatt a gráfot alakítsuk át fává, aminek gyökéreleme a

startcsúcs legyen, levélelemei pedig a terminális csúcsok. A fa páros és páratlan szintjein az egyik illetve a másik játékos lehetséges lépései szerepelnek, tehát a fában az egyes csúcsokból kilépő élek a soron lévő játékos megengedett lépéseinek felelnek meg. A startcsúcs a 0. szinten helyezkedik el, ahol a kezdő játékos következik lépni. A startból valamely terminálisba vezető úton végiglépkedve egy lehetséges *játszmá*t kapunk. Ha gráf tartalmazza az adott játék összes lehetséges játszmáját, akkor azt *játékgráf*nak, játékfának hívjuk. Minden játékot ábrázolni tudunk egy ilyen irányított gráffal. Ezen gráf csúcsai reprezentálják a játék során előálló összes lehetséges állást, élei pedig az összes, a játék során megtehető lehetséges lépést.

2.2.2.2 Az ÉS/VAGY gráfok

A közönséges gráfokon nem tudunk olyan vizsgálatokat végezni, amelyekkel különbséget tehetnénk a szorosabban, illetve a kevésbé összetartozó élek között. Az *ÉS/VAGY gráfok* olyan irányított hipergráfok, amelyekben egy hiperél egy csúcsból egy csúcshalmazba vezet. Tehát a G = (N,HE) egy ÉS/VAGY gráf, ahol N jelöli a *csúcsok* halmazát, a $HE \subseteq \{ (n,M) \in N \times 2^N \mid 0 \neq |M| < \infty \}$ pedig az irányított *hiperélek* halmazát. Irányított *hiperút*nak nevezzük az n csúcsból az M csúcshalmazba vezető olyan részgráfot, amelyben mindegyik csúcsból legfeljebb egy hiperél indul ki. Az M csúcsaiból nem indul hiperél, hanem úgymond „egyszerű” élek indulnak az őket követő csúcsokba.



[3. kép]

Az ábrán látható, hogy az n egy csúcs, az $\{m_1, m_2, m_3\}$ pedig egy csúcshalmaz és az ezeket összekötő éleket együttesen nevezzük hiperélnek.

Azért nevezik az ilyen típusú fákat ÉS/VAGY gráfoknak, mert a játékosok által tehető lépéseket VAGY-, illetve ÉS-élekként tüntethetjük fel a keresési fában. Amikor egy játékosra sor kerül, dönthet, hogy milyen lépést tesz: egyik VAGY másik irányba lép. Viszont nem tudhatja, hogy az ellenfele erre mivel fog reagálni, ezért annak lépéseit egységesen kell kezelnie. Az ellenfélnek tehát egyszerre több lehetséges lépése van, és nem láthatjuk előre, hogy melyiket fogja választani. A fában ezeket a hiperéleket nevezünk ÉS éleknek. Az ábrán ezeket az ívvel összekötött élkötegek jelölik,még a szimplák pedig a VAGY élek.

2.2.2.3 A nyerő stratégia

Az ÉS/VAGY gráfokra azért van szükségünk, hogy egy adott játékos szemszögéből vizsgálódhassunk és számára nyerő stratégiát kereshessünk. *Stratégiá*nak nevezünk egy olyan döntési tervet, amely a játékos számára előírja, hogy a játék során azon állásokban, amelyekben ő következik lépni, a megtehető lépései közül melyiket válassza. Ezen játékos lépéseit szemléltető élek mindegyike egy szimpla él (VAGY élek), míg az ellenfelének egy-egy állásban megtehető lépéseit szemléltető élköteg egy-egy hiperél lesz (ÉS élek). A játékos stratégiáit az ÉS/VAGY gráfban a startcsúcsból valamely levélcsúcsba vezető hiperutak jelentik. Egy játékos számára akkor létezik *nyerő stratégia*, ha minden eléje kerülő szituációban mindig van legalább egy olyan lépés (olyan irány a fában), hogy számára nyerő végállapotba tud kerülni, azaz ellenfele bármilyen stratégiája esetén is győzni tud. Elmondhatjuk, hogy a teljes információjú, kétszemélyes, véges játékok esetén mindig létezik nyerő stratégia az egyik játékos számára, ha a játék nem végződhet döntetlennel. Ha viszont igen, akkor pedig úgy fogalmazunk, hogy valamelyik játékos számára biztosan létezik *nem vesztő stratégia*. A jó stratégiát úgy lehet szemléletesen megmutatni, hogy ha lerajzoljuk a teljes játékfát, a fa szintjeit pedig megjelöljük a soron következő játékos nevével. Így tehát a 0. szinten, ahol a startcsúcs helyezkedik el a kezdő játékos (jelöljük A-val) következik lépni, az 1. szinten az ellenfele (jelöljük B-vel) és így tovább lefelé a fában. Ezt követően megvizsgáljuk a terminális csúcsokat, hogy az adott végállásban ki nyert, majd megcímkézzük a levélelemeket ezen játékos nevével. Szintenként visszafelé lépkedve a fán a közbenső csúcsokat is megcímkézzük: ha a szinten A következik lépni és a csúcs

leszármazottjai között van A jelű gyermeke, akkor a csúcs A címkét kap, egyébként B-t. Ezt az eljárást követve felfelé haladva a véges fán utoljára kap egy címkét a startcsúcs is, amin leolvasható az a játékos, akinek van nyerő stratégiája. Több nyerő stratégiája is lehet a játékosnak és mindegyik lekövethető a fán.

2.2.3 Hasznosságfüggvény: a heurisztikus kiértékelés

A teljes játékfa felépítése nem minden esetben triviális elvárás, ugyanis a legtöbb játék sok állapotból áll és a fa mérete igen nagy lehet. Ezért annak a elvárásnak nehéz eleget tenni a legtöbb játék esetén, hogy a startcsúcstól az összes terminális csúcsig belátható időn belül megalkossuk a fát. Ezért született meg az az ötlet, hogy egyszerre csak a fa egy részfája legyen kiértékelve. Ezzel nagymértékben meggyorsítható a számítási eljárás. A részfa levélelemeire heurisztikus függvényt alkalmazva egy becslést kapunk arról, hogy mennyire lenne hasznos az adott irányba lépni. A *heurisztika* ökölszabályt, következtetésekben alkalmazott általános elvet jelent, amely inkább tanácsot ad mint tévedhetetlen utasítást. Megfigyelés, tapasztalat, gyakorlás útján kialakított szabályrendszert takar. Nagy mértékben javíthatók vele a fában történő keresések, viszont rosszul felállított heurisztikával könnyen tévútra is kerülhetünk. A heurisztikák, amelyek bizonyos felismerhető mintákra lehetséges válaszlépéseket „sugallnak”, segítségünkre lehetnek abban, hogy egy bonyolult problémában ki tudjunk igazodni. A játékot elemezve összegyűjthetők olyan jellemzők, amelyekből olyan *heurisztikus vagy* másként nevezve *statikus kiértékelő függvény* írható fel, amelyet alkalmazva a levelekre számszerűsítve kaphatjuk meg az állások hasznosságát. Ezt úgy képezzük, hogy a támogatott játékos szempontjából előnyös állapotú leveleknek +1, az ellenfélnek kedvező állásokban -1, míg a semleges helyzetekben 0 értéket adunk. Ez a legegyszerűbb kiértékelés, viszont a legtöbb játék esetében ennél összetettebben szokás súlyozni a leveleket. A heurisztika megállapításánál szélesebb skálát is használhatunk, hiszen könnyen elképzelhető, hogy egy játékos számára van több jó állás is, viszont egyáltalán nem biztos, hogy azok egyformán előnyösek. Ahhoz, hogy kedvező és kedvező állás között is különbséget tudjunk tenni elvárjuk, hogy minél jobb egy állás a tekintett játékos számára, a függvény egy annál

nagyobb pozitív számmal becsülje azt. Az Abapa esetén én egy -6 - +6-ig terjedő értékelést használtam, ahol a +6 a győztes, míg a -6 a vesztes állását jelöli.

3. A megvalósításhoz használt algoritmikus módszerek

A gráfokról szóló fejezetben ismernünk kellett a teljes játékfát ahhoz, hogy elemezhessük az esélyeket. Viszont egy egyszerűnek tűnő játék is túl bonyolult lehet ahhoz, hogy megadhassuk a teljes játékfát. Az Abapa fájának is igen nagy a mérete, hiszen ha feltesszük, hogy egy állásból átlag 5 különböző lépés tehető és egy átlagos játszma 15 lépésváltásból áll, akkor a játékfa $5^{(2*15)} = 5^{30}$ csúcsból tevődik össze. Tehát nem feltétlenül tudunk biztos nyerő stratégiát meghatározni egy játékhoz, helyette csak egy „elég jó” lépést tudunk keresni egy adott állásban a soron következő játékosnak. Az adott állásból kiindulva csak a következő lépést határozzuk meg, így a játékfának csupán egy részét építjük fel egy bizonyos mélységig.

3.1 Lépésajánló algoritmusok

Általánosságban annyit mondhatunk el a fejezetcímben említett módszerekről, hogy a céljuk az, hogy az aktuális játékállásban kiszámítsanak egy megfelelő következő lépést. Arról, hogy mit tartunk megfelelőnek az egyes módszereknél külön-külön lesz szó. Ellenben mindnél közös az, hogy működésük közben mindegyikük felépít a memóriában egy fa adatszerkezetet, mint amiről már a korábbi, a gráfokról szóló fejezetekben beszéltünk. Ezen fának ágait bizonyos mélységig felépítve kiszámítható a választható utak „jóság” értéke. Ez annyit tesz, hogy a program kiszámítja, hogy merre érdemes továbbmenni, tehát melyik az az útvonal a fában, amelyet követve nagyobb valószínűséggel juthatunk el győztes állásba. Persze minden újabb kör, amikor valaki lépni következik más és más helyzetet eredményez, ezért mindig újabb részfát épít fel a program dinamikusan attól függően, hogy mi volt az ellenfél lépése. Most bemutatásra kerülnek a leggyakrabban használt hatékony számítási módszerek, ahol az

A és B játékosokat MAX-nak és MIN-nek jelöljük. A MAX lép először, majd pedig felváltva lépnek, amíg a játék véget nem ér.

3.1.1 A Minimax algoritmus

Ennek a módszernek a lényege, hogy az algoritmus egy „elég jó” lépést ajánljon a támogatott játékosnak. Nevezzük MAX-nak azt a játékost, aki számára meg szeretnénk határozni az optimális stratégiát és a legkedvezőbb kezdőlépést és MIN-nek az ellenfelét. A (2.2.3)-as fejezetben kifejtett hasznosságfüggvényt fogjunk alkalmazni a keresőfa leveleire, amely számszerűen jelzi az aktuális csúcsban levő állás „jóságát”. Ha találtunk egy alkalmas heurisztikát, akkor azt alkalmazhatjuk a megadott mélységig felépített részfa levélelemeire. A MAX számára egy olyan lépés a kedvező, amellyel eljuthat egy maximális értékű állásba. De mivel MIN célja ezzel ellentétes, ezért az ő lehetséges lépéseit is figyelembe kell vennünk a választáskor. MAX-nak ezért olyan kezdőlépést érdemes tennie, amely őt a lehető legkedvezőbb állásba juttatja még akkor is, ha MIN-től válaszként a legerősebb ellenlépését várjuk. A Minimax algoritmus az úgynevezett *legnagyobb biztos előnyszerzés elvé*t valósítja meg a lehető legjobb első lépés kiválasztásakor. Ha a csúcs szintje páros (a startcsúcs a 0. szinten áll), akkor MAX a lépő játékos, ellenkező esetben MIN. Innen az eljárás elnevezése. MAX-csúcs esetén a csomópont gyermekeiben lévő hasznosságok közül a maximálist választja ki az algoritmus, amit jóságértékül kap a csúcs. Viszont ha MIN a soron következő játékos, akkor pedig a legkisebb hasznosságút választja, mert MIN-nek ez a legerősebb ellenlépése. Az eljárással mindössze a MAX következő lépését tudjuk meghatározni. Nem tudhatjuk, hogy a MN erre a számára legkedvezőbb lépést teszi-e meg.

Az Minimax algoritmus lépései a következők:

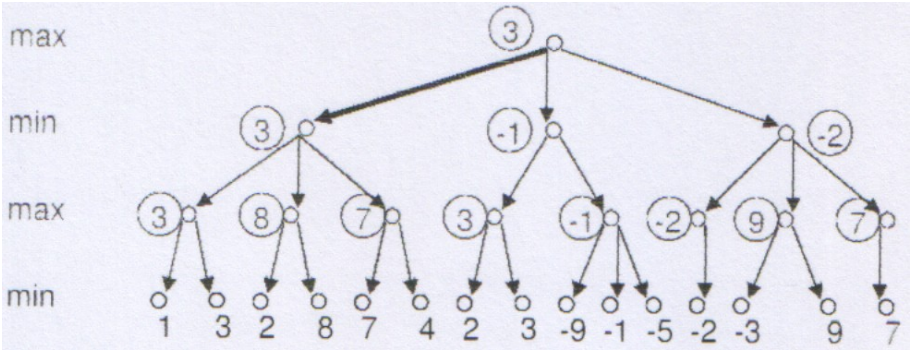
1) A részfa megadott mélységig való felépítése.
2) A részfa levélelemeiben található állásokra a támogatott játékos szempontjából heurisztikus becslést adni.
3) A kapott jóságértékekből ($h(n_1)$, $h(n_2)$, …., $h(n_k)$) meghatározni a fában az egy

szinttel feljebb levő csomópont hasznosságát ($h(n_{szülő})$):

- amennyiben a szülő szintje páros, akkor $h(n_{szülő}) = \max(h(n_1), h(n_2), \ldots, h(n_k))$,
- ellenben, ha páratlan, akkor $h(n_{szülő}) = \min(h(n_1), h(n_2), \ldots, h(n_k))$ legyen.

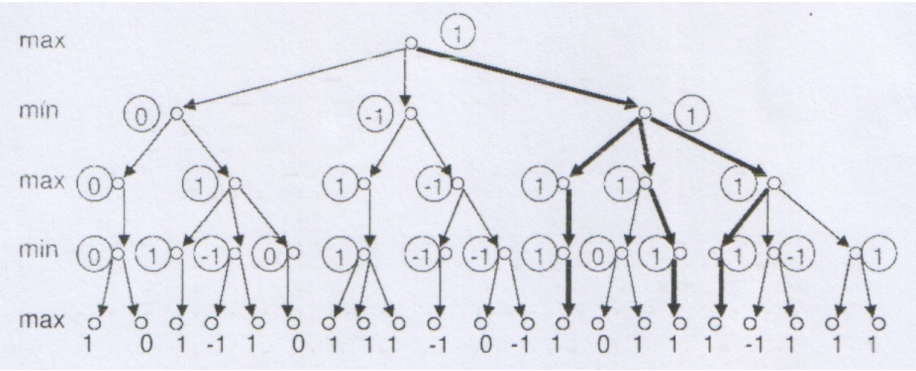
4) A fában szintenként felfelé lépkedve folytatódjon a csomópontok kiértékelése egészen a részfa gyökeréig.

Ha a nulladik szintre jutunk az eljárással, akkor a játékfa startcsúcsa is értéket kap, ami nem más, mint a gyermekei jóságértékei közül a legnagyobb. A [6. kép] egy fa minimax kiértékelésére példa, ahol a vastaggal rajzolt nyíl a lehető legjobb első lépést mutatja.



[6. kép]: Egy játékfa minimax kiértékelése

Ha meg van adva a teljes játékgráf, akkor a Minimax algoritmus meghatározza a nyerő stratégiát, hiszen a terminális csúcsokban szereplő állásokról eldönthető, hogy ott ki nyert vagy vesztett és ha MAX a nyerő, akkor +1-et, ha MIN, akkor -1-et, ha pedig döntetlen az állás, akkor 0-t rendelünk hozzájuk. Így a startcsúcs értéke is meghatározható ami megadja, hogy kinek van nyerő stratégiája és azok hol vannak a fában. Erre példát a [7. kép] ad.



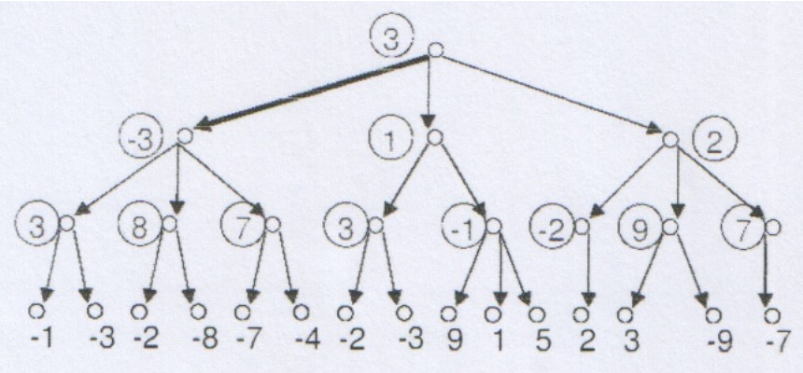
[7. kép]: A nyerő stratégia meghatározása minimax eljárással

3.1.2 A Negamax algoritmus

Ez az eljárás az előző algoritmus változata, de egyszerűbb számítógépes megvalósítási módszert ad a jó lépések meghatározására. Különbsége abban rejlik a Minimaxhoz képest, hogy a részfa levélelemeiben lévő állások jóságát a lépésben következő játékos szempontjából becsli, nem pedig MAX szempontjából. Ezzel egyszerűbb kiértékelést kapunk, mivel MAX és MIN ellentétes szempontját kifejezhetjük eltérő előjellel. A csomópontok értékei ez esetben szintenként előjelet váltanak, ezért minden szinten elég csak maximumot képeznünk. A korlát mélységig felépített részfa leveleire szintén adunk egy becslést, mint a Minimaxban. Viszont a különbség itt az, hogy ha a levélelemek páratlan (MIN) szinten vannak, akkor a becslés -1-szeresét adjuk értékül a csúcsoknak. A részfa közbenső elemeinek kiértékelése úgy történik, hogy a gyermekcsúcsok értékeinek a negáltját képezzük, majd ezek közül a legnagyobbat kapja a csomópont.

Az Negamax algoritmus lépései a következők:

1) A részfa megadott mélységig való felépítése.
2) A részfa levélelemeiben található állásokra a soron következő játékos szempontjából heurisztikus becslést adni:
    - ha páratlan (MIN) szinten állunk, akkor képezzük a becslés negáltját,
    - különben (MAX szint) marad a kapott hasznosságérték.
3) A kapott jóságértékekből ($h(n_1)$, $h(n_2)$, …., $h(n_k)$) meghatározni a fában az egy szinttel feljebb levő csomópont hasznosságát ($h(n_{szülő})$):
    - $h(n_{szülő}) = \max(-h(n_1), -h(n_2), …., -h(n_k))$.
4) A fában szintenként felfelé lépkedve folytatódjon a csomópontok kiértékelése egészen a részfa gyökeréig.



[8.kép]: Egy játékfa negamax kiértékelése

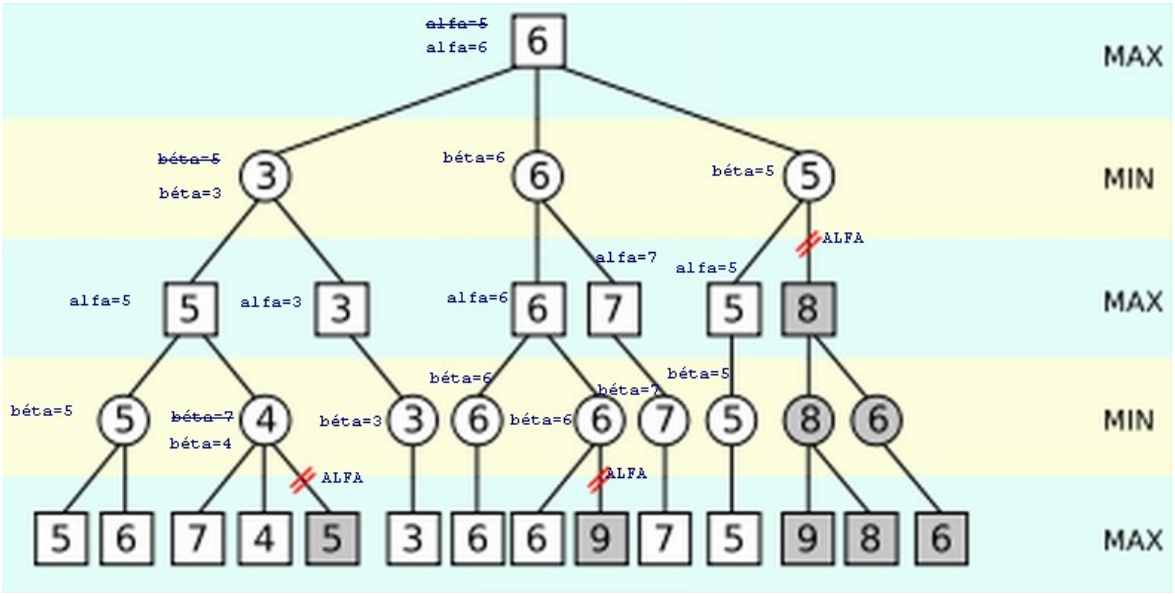
3.2 Hatékonyságnövelés Alfa-Béta vágással

Az alábbi módszer gyakran alkalmazott technika olyan számítógépes játékprogramok esetén, amelyek reprezentációjában a játékgráf nagyszámú csúcsot tartalmaz. Ez az eljárás is a Minimaxon alapul, végeredményül ugyanazt a lépést ajánlja, de jóval optimalizáltabb. A felesleges ágak lenyírása lehetővé teszi, hogy a fa-reprezentáció igen nagy részeit egyszerűen elhagyjuk. Erre azért lehet okunk, mivel az ezeken a „felesleges” ágakon keresztül vezető utak kevés esélyt adnak a játékosnak a győzelemre, így egyszerűen nem vesszük figyelembe őket. A vágás után lecsökken a bejárandó utak száma, így gyorsabbá válik a keresés. A módszer hatékonysága abban keresendő, hogy párhuzamosan végezzük a részfa generálást, a levélelemek és a szüleik becsült értékének visszaadását, így nincs szükség minden esetben az összes csomópont kiértékelésére. Egy csúcsot akkor hagyhatunk figyelmen kívül, ha a kiterjesztése előtt már eldől, hogy tőle nem kaphat értéket a szülő csúcs. Egy ilyen felismerés esetén érthető, hogy az adott oldalággal nem kell tovább foglalkoznunk. A vágás megvalósításához a Minimax 3)-4)-es lépéseit kell módosítanunk. A részfa levélelemeire ugyanúgy elvégezzük a heurisztikus becslést, majd a közbenső csúcsok hasznosságát alfa, illetve béta értékekkel címkézzük, attól függően, hogy MAX vagy MIN szinten vannak-e. Egy csúcs akkor kapja meg valamelyik jelzőt, ha már van legalább egy kiterjesztett utóda.

Egy csomópont kiértékelése kétféleképpen történhet:

a) Ha az aktuális csomópontban minimumot (béta) számolunk, ekkor a szülője MAX szinten (alfa) áll. Az aktuális csúcs kiterjesztését nem kell tovább folytatnunk, ha a rá kapott béta érték (minimum) kevesebb, mint a szülőben kapott alfa (maximum). Azt mondhatjuk tehát, hogy *Alfa-vágás*t végzünk egy részfán, ha a gyökércsúcsa béta értéket kap és ez kisebb, mint az egy szinttel feljebb lévő alfa érték.

b) Az előbbivel analóg módon történik, de ekkor az aktuálisban maximumot (alfa) számolunk, a szülőjében pedig minimumot (béta). Az aktuális csúcs kiterjesztését nem kell tovább folytatnunk, ha a rá kapott alfa érték (maximum) nagyobb, mint a szülőben kapott béta (minimum). Azt mondhatjuk tehát, hogy *Béta-vágás*t végzünk egy részfán, ha a gyökércsúcsa alfa értéket kap és ez nagyobb, mint az egy szinttel feljebb lévő béta érték.



[8. kép]: Alfa-Béta vágás

Megállapítható, hogy akkor végezhetünk vágást, ha alfa ≥ béta teljesül. Az ábrán a négyzetes csúcsok alfák, míg a karikázottak a béták. Három Alfa-nyesést láthatunk. Elsőnek a legbaloldalibb béta csúcsba minimumot kerestünk. Először megkapta bétánk az 5-öt, amely érték felfelé a fán végigöröklődött. Így lett a startcsúcs alfa = 5. A legbaloldalibb bétacsúcs a 6-ot nem vehette fel, mivel itt a minimumot keressük. Mellette a testvére a béta = 7-et kapta meg először, de tovább keresünk az ágon, hátha van kisebb. Mivel a 4 < 7, ezért béta = 4 lesz.

Észreveszzük, hogy ez kisebb, mint az eggyel magasabb szinten lévő alfa érték, ezért nem szükséges továbbkeresni a részgráfban. Az alfa > béta miatt a maradék ágat vizsgálatlanul levághatjuk, ami egy Alfa-metszés lesz, mivel béta csúcs alatt vágunk. A fa többi ágán is hasonlóképp zajlik az elemzés.

4. A játék megvalósítása

Ebben a fejezetben először a programom felépítéséről kívánok egy átfogó vázlatot adni, majd pedig a játékprogramomban szereplő néhány könnyebb, bonyolultabb metódus kifejtésére szeretnék rátérni.

4.1 Osztályhierarchia

Az Abapa játékprogramja Java nyelven íródott, amely nyelv objektum orientált paradigmát vall. Az osztályok három csomagban – package-ben- helyezkednek el. Ezek elnevezése sorra: `abapa`, `algoritmus` és `allapotter`.

a) Az `allapotter` package tartalmazza az `Operator` és `Allapot` absztrakt osztályokat, illetve a `Jatek` osztályt. Ez utóbbiban van az egyik legfontosabb eljárás, a `jatszik()`, amelyet a `main()` metódus hív meg és tulajdonképpen a játék vezérléséért felelős. Ezen metódus vizsgálja meg, hogy
   - az adott játékállás végállapot-e,
   - ha igen, akkor ki nyert, illetve
   - itt kerül kiírásra a játék aktuális állapota,
   - a lépő játékos,
   - a gép lépése, továbbá
   - ha kértük, akkor a gép által nekünk megajánlott lépés.

   Ebben az osztályban kerül meghatározásra az is, hogy
   - ember vagy gép ellen kívánunk-e játszani ezen kívül, hogy
   - az ember vagy a gép kezdje-e a játékot,

- kérünk-e a géptől ajánlatot,
- melyik lépésajánló algoritmust kívánjuk használni, illetve
- milyen mélységben vizsgálja meg a fát.

b) Az `algoritmus` csomagban kaptak helyet a lépésajánló algoritmusok. A `Lepesajanlo` egy absztrakt osztály és az itt deklarált adatmezők a belőle származtatott osztályokban lettek implementálva. Ezek a `MinmaxAlgoritmus`, a `NegamaxAlgoritmus`, `AlfabetaMinimaxAlgoritmus` és `AlfabetaNegamaxAlgoritmus` osztályok, amelyek konstruktoraiban meghívásra kerülnek a `minimaxHasznossag()` illetve a `negamaxHasznossag()` metódusok, amelyek dolga a pillanatnyi állásban a leghasznosabb lépés kiválasztása.

c) Végül pedig az `abapa` packeg-et ismertetem. Ebben kapott helyet az `Operator`-ból származtatott `Felmarkol` osztály, az `Abapa`, aminek az ősosztálya az `Allapot`, illetve a `Main` class is, amiben meghívásra kerül a `jatszik()` eljárás.

   - A `Felmarkol.java`-ban kerül definiálásra a játék egyetlen operátora. Amikor ennek az operátornak a konstruktorát meghívjuk, akkor megadjuk, hogy melyik házból (`honnan`) kívánjuk felmarkolni a magokat (`mennyit`). Mint tudjuk a játék szabályai azt mondják, hogy a felmarkolással az összes magot kivesszük a kiválasztott házból. Ebben az osztályban egy fontos adatmezőt is deklaráltam, az `utolso`-t, ami nem más, mint a honnan és mennyit mezők összegének 12-vel vett modulója. Ez adja meg annak a háznak az indexét, amibe vetés során utolsónak kerül mag.
   - Az `Abapa` osztályban lettek definiálva a játékszabályokat megvalósító eszközök:
     - A tábla egy 13 elemű `int`-eket (valós egész számokat) tartalmazó egydimenziós tömb: `hazak = new int[13]`. A Java 0-tól indexeli a tömböket, de én mivel a házak 1-12-ig számozásúak, a 0-s indexűt nem használom. Kezdetben mindegyikben 4-4 mag van. Ezen kívül használok két `int` típusú mezőt, `gyujtottA`-t és `gyujtottB`-t, melyek szerepe, hogy számon tartsák az A és B játékosok learatott magvait.
     - Szerepelnek továbbá olyan metódusok, amelyeket az Allapot osztálytól örökölt az Abapa osztály. Ezek feladata a játék végének felismerése (`isVegAllapot()`) és, hogy ki nyert ekkor (`nyerA(), nyertB()`), ezentúl,

hogy alkalmazható-e egy operátor(`isAlkamlmazhato(Operator o)`) és ha igen, mi az alkalmazásának az eredménye (`alkalmaz(Operator o)`). A játék állapottér-reprezentációjában formálisan megadott követelmények megvalósításai ezek a függvények.

- Itt lettek definiálva a `minimaxHasznossag()` és a `negamaxHasznossag()` függvények is.
- A játék különböző lépéseinek megfogalmazása is az Abapa osztályban történik, melyek formalizált alakja szintén megtalálható az állapottér-reprezentációban. Ilyenek például a vető és arató függvények.

4.2 Az állapottér-reprezentációs követelmények implementálása

Ezen a helyen az Abapa állapottér-reprezentációjában szereplő néhány függvénynek és eljárásnak a Java-ban leprogramozott megvalósítása lesz olvasható magyarázatokkal kiegészítve.

A Felmarkol(honnan, mennyit) operátor-alkalmazási előfeltétele:

Az operátor Felmarkol-lá kasztolása után következik két nagyobb if, melyek ellenőrzik, hogy az A, illetve a B játékosok a saját oldalukról markolnak-e fel, ezen kívül a `mennyit` mező értékét figyelik. A belső if-else szerkezet az állapottér-reprezentáció operátor-alkalmazási előfeltételéből vizsgálja, hogy az ellenél házai kiürültek-e. Ha igen, akkor vagy etetés van vagy pedig - ha false-t ad vissza - ürítés.

```
public boolean isAlkalmazhato(Operator o) {
    if (o instanceof Felmarkol) {
        Felmarkol f = (Felmarkol) o;
        if(jatekos == 'A' && (1 <= f.getHonnan() && f.getHonnan() <=6)
   && (f.getMennyit() > 0 && f.getMennyit() == hazak[f.getHonnan()])) {
            if(osszeAd(7,12,this) == 0) {
                if(etet(this,f))
                    return true;
                return false;
            }
            else return true;
```

```
            }
            if(jatekos == 'B' && (7 <= f.getHonnan() && f.getHonnan() <=12)
        && (f.getMennyit() > 0 && f.getMennyit() == hazak[f.getHonnan()])) {
                if(osszeAd(1,6,this) == 0) {
                    if(etet(this,f) )
                        return true;
                    return false;
                }
                else return true;
            }
        }
        return false;
    }
```

A Felmarkol(honnan, mennyit) operátor alkalmazásának hatása:

Ha az aktuális állapotra alkalmazható egy operátor, akkor meghívjuk rá az `alkalmaz(Operator o)` függvényt, amely az operátor alkalmazásának hatásaként egy új állapotot hoz létre a régiből. Az `Abapa uj = new Abapa(this);` az aktuális állapotnak egy másolatát készíti el, mivel az alábbi konstruktor kerül meghívásra:

```
    public Abapa(Abapa a){
        hazak = Arrays.copyOf(a.hazak, a.hazak.length);
        gyujtottA = a.gyujtottA;
        gyujtottB = a.gyujtottB;
        jatekos = a.jatekos;
    }
```

A következő függvényt három fő egységre lehet szétbontani:

a. a vetés elvégzése,
b. az aratás felügyelete és
c. a házak kiüríthetőségének figyelése.

```
    public Allapot alkalmaz( Operator o ) {
        aratott_magok = 0;
        Abapa uj = new Abapa(this);
        if( o instanceof Felmarkol ) {
            Felmarkol f = (Felmarkol) o;
//VET:
            uj = vet(f,uj);

            if( jatekos == 'A' ) {
                aratott_magok = AratA(f,uj);
                if( aratott_magok > 0 ) {   //ha tortent aratas
//GRAND SLAM A:
                    if( ( aratott_magok - GrandSlamA(f,uj)) == 0 ) {
                        int ures = uresHazak();
```

```
                        for(int i = 12-ures; i >= 7; i--)
                            uj.hazak[i] = hazak[i] + 1;
                    }
//ARAT "A":
                    else {
                        aratott_hazak = szamolHazak(f,uj);
                        int j = 0;
                        while( j <= aratott_hazak-1 ) {
                            uj.hazak[f.getUtolso()-j] = 0;
                            j++;
                        }
                        uj.gyujtottA = uj.getGyujtottA() + aratott_magok;
                    }
                 }
            }
            if( jatekos == 'B' ) {
                aratott_magok = AratB(f,uj);
//GRAND SLAM B:
                if( aratott_magok > 0 ) { //ha történt aratás
                    if( aratott_magok - GrandSlamB(f,uj) == 0 ) {
                        int ures = uresHazak();
                        for(int i = 6-ures; i >= 1; i--)
                            uj.hazak[i] = hazak[i] + 1;
                    }
//ARAT "B":
                    else {
                        aratott_hazak = szamolHazak(f,uj);
                        int j = 0;
                        while( j <= aratott_hazak - 1 ) {
                            uj.hazak[f.getUtolso() - j] = 0;
                            j++;
                        }
                        uj.gyujtottB = uj.getGyujtottB() + aratott_magok;
                    }
                 }
            }
//VÉGTELEN KÖR:
            vegtelenKor(uj,f);
//ÜRÍTÉS:
            if(urit(uj)) {
                if(jatekos == 'A') {
                    for(int i = 1; i <= 6; i++) {
                        uj.gyujtottA += uj.hazak[i];
                        uj.hazak[i] = 0;
                    }
                }
                if(jatekos=='B') {
                    for(int i = 7; i <= 12; i++) {
                        uj.gyujtottB += uj.hazak[i];
                        uj.hazak[i] = 0;
                    }
                }
            }
        }
        uj.valtJatekos();
        return uj;
    }
```

a. A vető metódus feladata az, hogy a paraméterül kapott állapotot házait megváltoztatva a felmarkolt magokat a szabályok szerint egyesével bedobja a házakba.
b. Mindkét játékos esetén létezik egy-egy arató szakasz, amelyek egymás tükörképei. A vetést követően az `aratott_magok` változó az aratást végző függvénytől kap értéket. Ha ez nulla, akkor nem történt aratás, majd pedig leellenőrizzük, hogy végtelen kör vagy ürítés következik-e, vagy csupán a másik játékos kapja meg a játszás jogát, ekkor az eddigi új állapot lesz az aktuális. Ha az `aratott_magok` pozitív, akkor viszont vizsgálnunk kell, hogy Grand Slam-et okozott-e a lépésünk, vagy valódi aratást. A Grand Slam függvény szintén lejátssza az aratást és leellenőrzi, hogy ez az ellenfél házainak kiürülését okozza-e. Ha igen, akkor a függvény visszatérési értékként ugyanannyit ad vissza, mint amennyi az `aratott_magok`. Ekkor belépünk a feltételbe, ahol az `ures` mező a 0 magot tartalmazó házak számát tartja nyilván és a for ciklus a többi ház értékét megnöveli eggyel a szabályok értelmében. Az else-ágba akkor kerül az algoritmus, ha nem Grand Slam helyzet van, hanem aratás. Megszámolja, hogy hány házat arattunk és ezeket sorra kinullázza, míg a learatott magokat az aktuális játékos gyűjtőjébe teszi.
c. A végtelen kör az a helyzet, amikor olyan kevés magja van mindkét játékosnak, hogy nem lesznek képesek egymás magjait learatni, ezért nem érdemes a játék folytatása. A meghívásra kerülő eljárás összeszámolja a tábla két felén lévő magokat és ha azokban kevesebb, mint 4-4 mag van, akkor beteszi azokat a játékosok gyűjtőibe, majd kinullázza a térfelek házait és véget ér a játék. A másik eset, hogy nincs végtelen kör hanem megvizsgáljuk, hogy táblaürítés állt-e elő, amit egy logikai függvény ellenőriz. Ha pl. az A lép, ellenőrzi, hogy B minden háza 0-e és A tud-e olyan lépést tenni, amivel magokat vet az ellenfele térfelére. Ha nem, akkor viszi a táblán lévő maradék magokat a saját gyűjtőjébe, azaz kiüríti a házait. Ekkor a játéknak vége lesz. Etetés után viszont a következő játékos folytatathatja a játékot, persze csak akkor, ha nem alakult ki végtelen kör, ami szintén a játék végét jelenti.

A vetést megvalósító függvény:

```
public Abapa vet(Felmarkol f, Abapa a) {
     int hova = f.getHonnan() + 1;
     a.hazak[f.getHonnan()] = 0;
     for( int i = f.getMennyit(); i > 0 ; i-- )
         int vethet = hova % 12;
```

```
            if(f.getMennyit() > 11) {
                if( f.getHonnan() != vethet ) {
                    a.hazak[vethet] = a.hazak[vethet] + 1;
                }
                else {
                    a.hazak[f.getHonnan()] = 0;
                    i++;
                }
                hova++;
            }
            else { // ha mennyit < 12
                a.hazak[vethet] = a.hazak[vethet] + 1;
                hova++;
            }
        }
        return a;
}
```

A `hova` jelzi annak a háznak az indexét, ahonnan a vetés indul, majd azt a házat ahonnan felmarkoltuk a magokat 0-ra állítjuk. A for addig lépked, amíg van a kezemben mag. Mivel a 12-es házat az 1-es követi, ezért alkalmazom a maradékos osztást a `hova`-ra. Az if-else-re azért van szükség, mert a szabályok azt írják elő, hogy ha a vetéssel körbe érünk a házakon, akkor a `honnan` házat kihagyjuk. Ezt a belső else oldja meg, amiben meg kell eggyel növeljük a ciklusszámlálót, hogy a kihagyás miatt a kezünkben maradó magot a következő házba tehessük. A `hova`–val pedig a következő házba lépünk, ha van még nálunk mag.

Az aratást végző függvények:

Következik két int visszatérési értékű függvény. Ezek dolga az aratás. Csak az A játékos metódusait mutatom be, mert a B-jé analóg vele. A reprezentáció Arat(i,A) és Arat_még(i,A) függvényeit valósítják meg, ahol i annak a háznak az indexe, amit aratunk.

```
public int AratA(Felmarkol f, Abapa a) {
    aratott_magok = 0;
       if( vetBfelere(f.getUtolso()) && (a.hazak[f.getUtolso()] == 2 ||
a.hazak[f.getUtolso()] == 3)) {
        aratott_magok = a.hazak[f.getUtolso()];
        for( int j = 1; j <= 5; j++ ) {
            int tmp = aratMegA(f.getUtolso() - j, a);
            if( tmp > 0 )
                aratott_magok += tmp;
            else
                break;
        }
    }
    return aratott_magok;
```

```
}

public int aratMegA(int i, Abapa a) {
    if( vetBfelere(i) && (a.hazak[i] == 2 || a.hazak[i] == 3) )
       if( vetBfelere(i+1) && (a.hazak[i+1] == 2 || a.hazak[i+1] == 3) )
            return a.hazak[i];
    return 0;
}
```

Az `AratA()` első if feltétele, ha teljesül, akkor az utolsónak vetett házat learatjuk. Ezután viszont a ciklusban meghívásra kerül az `aratMegA()`, ami megvizsgálja, hogy az ellenfél térfelén a megelőző házakat is aratjuk-e. Megnézi, hogy ha az aktuális ház aratható, akkor vajon a megelőző is az-e. Ha igen, akkor visszaadja a házban lévő magok számát a `tmp`-nek, különben pedig 0-t. Az ilyen többszörös aratáskor az aratott magok száma összeadódik és ezzel tér vissza az `AratA()`. Ha az `aratMegA()` 0-t ad vissza, akkor kilépünk a for-ból, mivel az aratási sor megszakad - ha nem 2-3 van a vizsgált házban - és nem arathatunk tovább.

A Grand Slam lépés:

```
public int GrandSlamA(Felmarkol f, Abapa a) {
    int magok = 0;
    if( vetBfelere(f.getUtolso()) && ( a.hazak[f.getUtolso()] == 2 ||
                                        a.hazak[f.getUtolso()] == 3 )) {
        int uresek = a.uresHazak();
        magok += a.hazak[f.getUtolso()];
        for( int j = 1; j <= 5-uresek; j++ ) {
           int tmp = aratMegA(f.getUtolso() - j,a);
           if(tmp > 0)
                magok += tmp;
           else return 0;
         }
     }
    return magok;
}
```

Lényegében ez is hasonlóan működik, mint az előzőek. Megszámolja, hogy az utolsónak vetett magot követően vannak-e üres házak az ellenfél térfelén. Ugyanúgy számolja a magokat, mint az aratásnál, itt is meghívja az `aratMegA()`-t majd 0-t ad vissza, ha nem Grand Slam volt a lépéssor.

4.3 A lépésajánló algoritmusok implementációja

A MinimaxAlgoritmus és NegamaxAlgoritmus osztályok:

Mint ahogy már említettem a megnevezett osztályok a `Lepesajanlo` absztrakt osztályból lettek származtatva. Belőle két konstanst és egyéb csomag-láthatóságú adattagokat örököltek. Utóbbiak a mínusz és plusz végtelent jelölik, de ezek csupán egy elég kicsi, illetve nagy integer értékek, melyekre szükség van az algoritmusokban a minimum és maximum operátorhasznosság kiválasztásához. Az örökölnek egy `operator` és egy `allapot`, továbbá három `int` típusú mezőt. Az `allasokSzama` tesztelésnél érdekes statisztikai információt hordozó tag csupán. A `melyseg` egy az algoritmus osztályok példányosításakor megadott szám, ami a fában való keresés korlátját határozza meg, a `hasznossag` pedig az `Abapa` osztályban levő minimax és negamax hasznosságfüggvények értékét hordozza.

```
package uj_algoritmus;
import uj_allapotter.*;

public class MinimaxAlgoritmus extends Lepesajanlo {

    public Operator getOperator() {
        return operator;
    }
    public int getHasznossag() {
        return hasznossag;
    }
    public int getAllasokSzama() {
        return allasokSzama;
    }

    public MinimaxAlgoritmus( Allapot a, int m ) {
        this.allapot = a;
        this.melyseg = m;
        this.allasokSzama = 1;//ezt az elso allapotot biztosan kiertekeljuk

        if( a.isVegAllapot() || m == 0 )
            this.hasznossag = a.minimaxHasznossag();
        else if( a.getJatekos() == 'A' ) {
            this.hasznossag = MINUSZ_VEGTELEN;
            for( Operator o : Allapot.getOperatorok() ) {
                if( a.isAlkalmazhato(o) ) {
                    Allapot uj = a.alkalmaz(o);
                    MinimaxAlgoritmus mini = new MinimaxAlgoritmus(uj,m-1);
```

```
                    if( mini.hasznossag > this.hasznossag ) {
                        this.hasznossag = mini.hasznossag;
                        this.operator = o;
                    }
                    this.allasokSzama += mini.allasokSzama;
                }
            }
        }
        else { // B jatszik
            this.hasznossag = PLUSZ_VEGTELEN;
            for( Operator o : Allapot.getOperatorok() ) {
                if( a.isAlkalmazhato(o) ) {
                    Allapot uj = a.alkalmaz(o);
                    MinimaxAlgoritmus mini = new MinimaxAlgoritmus(uj,m-1);
                    if( mini.hasznossag < this.hasznossag ) {
                        this.hasznossag = mini.hasznossag;
                        this.operator = o;
                    }
                    this.allasokSzama += mini.allasokSzama;
                }
            }
        }
    }
}
```

Látható, hogy a konstruktor végzi az érdemi feladatot, aminek paraméterei egy `Allapot` típus és egy integer. Utóbbit, mint mélységet kapja meg a `Lepesajanlo`. Első lépésként teszteljük, hogy a kapott állapot esetleg végállapot-e illetve, hogy nem járunk-e a részfa gyökerében, mert ilyen esetekben az aktuális állapot hasznossága lesz a nyerő. Ha nem ez a helyzet, akkor attól függően ki a támogatott játékos a `hasznossag`-ot minimumra vagy maximumra állítjuk. Ha találunk alkalmazható operátort az aktuális állapotra, akkor azt alkalmazzuk is és az így kapott új állapotra rekurzív hívással újabb algoritmus példányt gyártunk le eggyel feljebb lépve a fában. Így lépegetve az adott szinten lévő csúcsok hasznosságértékei közül a támogatott játékos szempontjából a legerősebbet (MAX), míg az ellenfele szempontjából a leggyengébbet (MIN) választja ki az algoritmus. Ezután a kiválasztott csúcsba vezető operátort fogja megnevezni, mint ajánlott lépést.

A NegamaxAlgoritmus osztály is hasonlóképpen működik, azzal a különbséggel, hogy ennél nincs támogatott játékos, hanem a soron következőnek ajánl alkalmas lépést. Ennél mindig csak a maximális értékét keressük a gyermek csúcsok -1-el szorzott hasznosságai közül.

```
public class NegamaxAlgoritmus extends Lepesajanlo {
    ...
    public NegamaxAlgoritmus(Allapot a, int m) {
        this.allapot = a;
```

```
        this.melyseg = m;
        this.allasokSzama = 1;

        if( a.isVegAllapot() || m == 0 )
            this.hasznossag = a.negamaxHasznossag();
        else {
            this.hasznossag = MINUSZ_VEGTELEN;
            for( Operator o : Allapot.getOperatorok() ) {
                if( a.isAlkalmazhato(o) ) {
                    Allapot uj = a.alkalmaz(o);
                    NegamaxAlgoritmus nega = new NegamaxAlgoritmus(uj,m-1);
                    if( -nega.hasznossag > this.hasznossag ) {
                        this.hasznossag = -nega.hasznossag;
                        this.operator = o;
                    }
                    this.allasokSzama += nega.allasokSzama;
                }
            }
        }
    }
```

Az AlfabetaMinimaxAlgoritmus és AlfabetaNegamaxAlgoritmus osztályok:

Az alábbi kódrészletben a minimaxon alapuló Alfa-Béta vágás megvalósítása olvasható. A konstruktorban rekurzívan újra és újra meghívjuk az eljárást a megadott mélységig, vagy amíg végállapotot nem találunk. Az A a támogatott játékos, neki keressük a legjobb kezdőlépést. Rögtön az elején teszteljük, hogy az alfa ≥ béta teljesül-e, mert ha igen, akkor nem terjesztjük ki a további csúcsokat, kilépünk a ciklusból és A esetén béta (a minimum) a jobb érték, tehát az alfa csúcs alatt Béta-vágást hajtunk végre. Ugyanez a B esetét jelölő else-ágban: az aktuális csúcsban minimumot számolunk, a szülőjében maximumot, alfát kapjuk legjobbnak, így Alfa-vágást hajthatunk végre béta alatt.

```
public AlfabetaMinimaxAlgoritmus(Allapot a, int m, int alfa, int beta) {
   ...
     if (a.isVegAllapot() || m == 0)
         this.hasznossag = a.minimaxHasznossag();
     else if (a.getJatekos() == 'A') {
         for( Operator o : Allapot.getOperatorok() ) {
             if (alfa >= beta)
                 break;
             if (a.isAlkalmazhato(o)) {
                 Allapot uj = a.alkalmaz(o);
                 AlfabetaMinimaxAlgoritmus abMini = new
                           AlfabetaMinimaxAlgoritmus(uj,m-1, alfa, beta);
                 if(abMini.hasznossag > alfa) {
                     alfa = abMini.hasznossag;
                     this.operator = o;
                 }
                 ...
```

```
                }
            }
            this.hasznossag = alfa < beta ? alfa: beta;
        }
        else { // ha jatekos == B
            ...
            if (alfa >= beta)
                break;
            ...
            if(abMini.hasznossag < beta) {
                beta = abMini.hasznossag;
                this.operator = o;
            }
            ...
            this.hasznossag = alfa > beta ? alfa: beta;
        }
    }
```

A negamaxra írt vágás algoritmusában természetesen csak egy helyen kell mindent vizsgálni, így a vágás feltételét is. Arra kell figyelni, hogy a rekurzióban negatív előjellel hívjuk az alfa és béta értékeket és amikor a legjobb hasznosságot keressük, akkor is a negált érték maximumát keressük.

```
...
AlfabetaNegamaxAlgoritmus abNega = new
                        AlfabetaNegamaxAlgoritmus(uj,m-1, -beta, -alfa);
if( -abNega.hasznossag > alfa) {
    alfa = -abNega.hasznossag;
    this.operator = o;
}
...
```

Összefoglalás

A dolgozat elkészítése közben alkalmam volt alaposan megismerni a mancala-típusú játékokat. Az Abapa program sokszori tesztelése után elmondhatom, hogy valóban szórakoztató és gondolkodást fejlesztő játék. Érdekes lett volna nyitási- és végjáték-adatbázisokat elemezni, mint amikről pl. a sakk esetén olvastam. Az ilyen adatbázisok hasznosak, ha erős játékossal szemben nagy mélységben kell a játékfában keresnie a programnak. Ezek felállításához nagy rutin és rengeteg elemzett játszma kell, de az Abapához nem találtam ilyesmiről információt. Dolgozatom célja a mesterséges intelligencia kétszemélyes játékokhoz kapcsolódó elméletének ismertetése, erre a tudásra épülő algoritmikus technikák elemzése és az optimalizált változatok bemutatása; majd pedig felhasználva ezen ismereteket egy működő játékprogram elkészítése volt. Ezeket sikerült megvalósítanom. Az állapottér-reprezentáció mindenhol szabályos formalizmussal történő megfogalmazása hosszas fejtörést okozott. Ezen kívül a pontos játékszabályok felkutatása is gondot jelentett a zavaros ismeretanyag miatt. A különböző források itt-ott eltértek egymástól és gyakran másképp fogalmazták meg ugyanazt a mozzanatot. Ez azért lehetséges, mert a világ oly sok felén játsszák a több száz eltérő nevű variánsait és ezért mindenhol kicsit másmilyen. De végül sikerült egy egységes szabályzatot kialakítanom. Az implementációban pedig makacs dolog volt, hogy a játék tesztelgetése során előjöttek olyan állások, amelyekre a szabályrendszer kialakításakor nem is gondoltam. A forrásokat újraolvasva vettem csak észre, hogy rosszul értelmeztem az egyes szabályokat, vagy elsikkadtam felettük. Az ilyen kivételeket ezért bele kellett szerkesztenem a kódba.

A mancala játékokról a http://www.awale.info és a http://www.oware.org oldalakon lehet a legtöbbet olvasni. Jóllehet egyszerű gyerekjátéknak tűnik, de érdemes kipróbálni valamelyiket.

Köszönetnyilvánítás

Mindenek előtt köszönet illeti Mecsei Zoltánt, hogy elkészíthettem nála a diplomadolgozatomat és mindig bizalommal fordulhattam hozzá, ha elakadtam. Köszönettel tartozom páromnak, Navratil Zoltánnak, hogy segített ötleteivel a megvalósítás során felmerülő nehézségek megoldásában, végig bíztatott és támogatott. Továbbá szüleimnek, hogy felhívták a figyelmemet a hibákra. Végül, de nem utolsó sorban Várterész Magdolna tanárnőnek és Kósa Márknak, hogy lelkiismeretes tanári munkájuk során megszerettették velem a Mesterséges intelligencia 1 tárgyat.

Irodalomjegyzék

Elektronikus források:

- http://www.oware.org
- http://www.gamesmuseum.uwaterloo.ca/VirtualExhibits/countcap/pages/kalah.html
- http://www.chilton-computing.org.uk/acl/literature/reports/p003.htm
- http://www.awale.info/?lang=en
- http://www.terebess.hu/keletkultinfo/mankala2.html
- Magyar Elektronikus Könyvtár: Láng Attila D. – Játékvilág
- Várterész Magda: Mesterséges intelligencia előadás anyag

Könyvek:

- Russell-Norvig: Mesterséges intelligencia modern megközelítésben, Panem Könyvkiadó, 2000.
- Starkné Werner Ágnes: Mesterséges intelligencia, Veszprémi Egyetemi Kiadó, 2004.
- Fekete István: Bevezetés a mesterséges intelligenciába, ELTE Eötvös Kiadó, 2006.

Folyóiratcikkek:

- Melrose Free Press, 1963. december 19-ei száma
- Time Magazine, 1963. június 14-ei száma

Képek:

- [1. kép]: http://www.la-diag.com/index.php?etape=menu20&category=8
- [2. kép]: http://www.awale.info/wp-content/uploads/awale_o_manqala.jpg
- [3. kép]: http://www.inf.unideb.hu/~varteres/mi/part4/problema_redukcio.htm
- [5, 6, 7. kép]: Fekete István: Bevezetés a mesterséges intelligenciába – 128, 129, 132. oldal ábrái

- [8. kép]: http://www.websters-online-dictionary.org/definitions/alpha-beta+pruning

Online játék linkek:

- http://www.awale.info/joc/en/index.html
- http://www.dst-corp.com/james/Awari.html
- http://www.ethnoludie.com/index_en.html
- http://www.rocketsnail.com/mancala/classic.htm
- http://www.freeonlinepcgames.net/hu/play/ancient-kalah-classic-mancala-game/flash-game/

Szómagyarázat az (1.1)-es fejezethez:

- 1: Az idézet a Time magazin 1963. június 14-ei számából való fordítás.
- 2-3: Mindkettő akan nyelvű ghánai uralkodó osztály.

Függelék

Láng Attila D.: Játékvilág

Vari (awari, mankola, mancala)

2 személy

Tábla: mindkét játékos térfelén hat mező, amelyeket a játékos szemszögéből nézve balról jobbra számozunk: az első játékos mezői 1-6, a másodiké 7-12 számot viselnek; az 1-essel szemközt a 12-es van

Figurák: 72, közös

Alapállás: 6-6 bábu az összes mezőre

A világ egyik legrégibb játéka; talán Egyiptomból, talán Afrika más részeiről származik. Ma Afrika-szerte népszerű.

Cél a bábuk többségének megszerzése.

Lépni úgy lehet, hogy a saját térfelünkön levő egyik mezőről felvesszük az összes bábut és egyesével lerakjuk azokat a következő mezőtől kezdve, a számozás növekvő sorrendjében (a 12-es után az 1-es jön).

Ha az utolsónak letett bábu olyan mezőre került, ahol ennek eredményeképpen 2, 4 vagy 6 bábu van, akkor ennek a mezőnek a tartalmát kivesszük és saját zsákmányunkként eltesszük.

Ha az így kiürített mező az ellenfél térfelén van, akkor az ellenfél térfelének megelőző mezőinek tartalmát is megkapjuk, ha azok a mostani lépésünkkel 2, 4 vagy 6 bábura egészültek ki.

Ha valakinek nincs bábu a térfelén, akkor nem tud lépni és a játék véget ért; a megmaradt összes bábu a másik játékosé lesz. Nem az nyer, akinek utolsónak maradt bábu a térfelén, hanem aki többet zsákmányolt.