SZAKDOLGOZAT

Dusza Anikó

Debreceni Egyetem
2010

Debreceni Egyetem
Informatikai Kar

KÉTSZEMÉLYES JÁTÉK WEBES FEJLESZTÉSE

Témavezető:
Kósa Márk
Egyetemi tanársegéd

Készítette:
Dusza Anikó
Programtervező informatikus

Debrecen
2010

Tartalomjegyzék

I. fejezet

Bevezetés

A „mancala” kifejezés az arab „naqala” – mozogni – szóból származik, és a kétszemélyes táblajátékok egy családját jelöli. Mancala nevű játék nincs, magát a szót Egyiptomban, Líbiában és Szíriában használják, de ezeken a helyeken sem egy konkrét játékot neveznek meg vele. Az eredete nem ismert, manapság több tucat változatát játsszák Afrikában, Ázsia egyes részein, a Nyugat-Indiai szigetvilágban és Dél-Amerikában. Afrikában a legnépszerűbb, a kontinens valamennyi térségében ismerik. Itt nem puszta táblajáték, kiemelkedő szerepet kap különböző ceremóniákon, ünnepségeken. Az uralkodó udvar legfontosabb rítusain is találkozunk vele, például az új uralkodó ezzel a játékkal bizonyította bölcsességét, stratégiai érzékét, higgadtságát. Épp ezért a táblák széles választékát megtaláljuk, az egyszerű agyagból készülteket és a drágakövekből kirakottakat egyaránt. A játék mára az Egyesült Államokban és Nyugat-Európában is népszerűvé vált.

Az elnevezések utalhatnak a tábla fajtájára (általában két- ill. négysoros), anyagára, a figurákra (magok, golyók, kavicsok), azok hangjára vagy a lépésekre. Afrikában a kétsoros táblákat általában warinak, a négysorosokat solonak nevezik. Az Egyenlítőtől északra Etiópia kivételével – ahol a háromsoros tábla terjedt el – kétsoros, délen és keleten négysoros táblákat használnak.

Feltételezések szerint a játék sumér eredetű, néhány ezer évvel ezelőtt komoly gazdasági érdekek miatt jött létre. Könyvelési célokra használhatták, az egyik térfélen a kiadásokat, a másikon a bevételeket jelölték. Az ősi egyiptomi leletek között találtak egy háromsoros, mészkőből faragott táblát, valószínűleg innen indulhatott hódító útjára. Zimbabwe, Ghána és Uganda területén sziklába, Etiópiában pedig három-négy méter magas (eredetileg falloszt formázó) ledőlt szobrokba vájtak játéktáblát. Alvarez a 16. századi abesszíniai portugál nagykövet beszámolójában megemlít egy ősinek mondott „mancal” nevű játékot. Háromszáz évvel később Bent írt le egy „gabatta” nevű változatot. A játékról Stewart Culin a következőket írja: „Három, *chachtmá*nak nevezett golyó van minden lyukban, és a játékban ezeket többször is sorra áthelyezik; ez előttünk nagyon bonyolultnak tetsző módszer szerint történt, nem is tudtunk rájönni; a lyukakat *tukul*nak, kunyhónak nevezik, és nagyon izgatottan

játszanak.” (Culin 1894, 601)

A játékhoz kapcsolódnak bizonyos szokások, hiedelmek. Például Bugandában (ma Uganda) a kabakán (uralkodó) és feleségein és nővérein kívül, a férfiak és nők ritkán játszották együtt. Ennek legfőbb oka az lehetett, hogy a férfiak büszkeségét sértette, hogy egy nő legyőzi őket. A királyi udvarhoz nem tartozó nőket a következő okokkal óvták a játéktól: rossz lesz a termés, illetve a kislányokat azzal ijesztgették, hogy nem nő meg a mellük, és senki sem fogja feleségül venni őket. Ezzel szemben az igbo (népcsoport Nigéria délkeleti részén) nők ritkán játszanak, bár nem tilos nekik, azonban a fiatal fiúk és lányok kihívhatják egymást. Ám egy igazi igbo férfi nem játszik egy asszonnyal. A Niger-deltájában élő idzso nők és lányok között népszerű az ikiokoto, ritka példája annak, hogy a nők részt vehetnek a játékban.

A győzelemért általában nem pénz, hanem megbecsülés járt, az ügyes játékos nevezetessé vált a családjában, a falujában, néha dalban is megénekelték. A mancala fontos szabálya, hogy a játékosok nem vehetik ki a gödörből a magokat azért, hogy megszámolják. A következő fejezetben taglaljuk a szakdolgozatban szereplő játék szabályait.

A mancalának az oktatásban is szerepe van. Egyszerűbb változatait puszta szerencsejátékként egészen kis gyerekek is játszhatják, számolásra készteti őket, és miközben a figurákat az egymást követő lyukakba rakják, megtanulják az egy-egy értelmű megfeleltetés elvét. A játék összes változata hasznos a gyerekek matematikai képességének fejlesztésében, hamarosan egyszerű összeadásokat is megtanulnak kiszámolni. Azokban a verziókban, amelyekben szabad visszafelé is lépni a játékos megismerheti a negatív számok és a kivonás fogalmát is.

A játék oktatásban betöltött hasznát egy valós afrikai történet is példázza. Pár évtizeddel ezelőtt Tanganyika (Tanzánia) nyugati részén élt egy Kamberage nevű fiú, aki nagyon szeretett szorót játszani apja barátjával, Ihunyo Főnökkel. A fiú állandóan nyert a játékban. Ellenfelére nagy hatással volt Kamberage ügyessége, ezért azt tanácsolta a barátjának, hogy taníttassa a fiát. Az megfogadta a tanácsot, és Kamberage a musomai kollégiumba került, ahol nemsokára ő lett az osztály legjobb tanulója. A fiút a világ Julius Nyerere néven ismerte meg, aki 1964–85 között Tanzánia elnöke volt.

II. fejezet

A játék alapjai és pár változata

Az alapok

Kellékek

A játék alapkellékei: két játékos vagy két csapat, akik a játékot játsszák. Egy tábla és figurák. Ha nincs táblánk, a célnak tökéletesen megfelel akár földbe vájt gödrök is. A figurák lehetnek kavicsok, magok, babszemek. A táblákon két-három ill. négy sor lyuk, más néven gödör vagy ház, található, néhány esetben a két szélén nagyobb, ún. gyűjtőlyukakat helyeznek el. A játékosok a gyűjtőlyukakban helyezhetik el az elnyert figurákat. A tábla mérete és a figurák elhelyezése változatonként eltérő lehet.

Célok

A cél általában az, hogy több figurát szerezzünk meg, mint az ellenfél, néha az, hogy elérjük, hogy az ellenfél ne tudjon többet lépni vagy a saját oldalunk üres legyen.

Vetés

A mancalákban a lépést vetésnek nevezik. Ez úgy történik, hogy a játékos kiválaszt egy figurákat tartalmazó gödröt. Ennek a legtöbb játékban a játékos térfelén lévő gödörnek kell lenni. A vetés úgy történik, hogy a játékos a soron következő gödrök mindegyikében elhelyez egy figurát. Az irány lehet az óra járásának irányával megegyező vagy ellentétes.

Elfogás

A játékos elfoghat bizonyos figurákat, attól függően, hogy hová esett az utolsó. Az elfogás feltételei, és az hogy mi történik az elfogott figurákkal a mancala fajtájától függ. Egyes változatokban lyukakat is el lehet fogni.

A legismertebb mancalaváltozatok céljai és szabályai

Wari (Oware)

Ez az afrikai eredetű játék talán a legismertebb a mancalák családjából. Ghána nemzeti játéka, az itteni akan anyanyelvű lakosság nevezi Owarénak. Főleg Nyugat-Afrika területén és a Karibi-térségben játsszák. Számtalan néven ismerik, a jorubák Ayo, Elefántcsontparton Awalé, Maliban Wari, a Zöld-foki Köztársaság Ouri, Ouril vagy Uril, A Karibi-térségben Warri néven ismerik, ewe nyelven Adji, ga nyelven Awélé. Angolul Awari, de a játékot tanulmányozó legelső kutató, Robert Sutherland Rattray a Wari nevet használta.

2002 májusában az Amszterdami Szabad Egyetemen két kutatója, Henri Bal és John Romein, számítógépekkel megoldották az Awarit, egy brute force megközelítést használva. Több mint 899 milliárd játékállásból határozták meg a nyerő stratégiát, amely döntetlenhez vezet. Azonban több oware-játékos megjegyezte, hogy a kísérlethez nem a nemzetközi versenyen alkalmazott abape szabályrendszert használták, hanem a Grand Slam variációt.

Tábla: soronként hat-hat lyukkal (házzal), a tábla végében egy-egy gyűjtőrekesszel, minden játékosnak a jobb kéz felé eső rekesz az övé

Játékosok: két játékos (vagy két csapat), mindkét játékos az előtte lévő hat lyukkal játszik

Figura: összesen negyvennyolc darab, egymástól nem különböztetjük meg

Cél: minimum huszonöt darab figura elfogása

Szabályok

1. Minden játékos huszonnégy darab figurával rendelkezik, amilyekből kezdetben minden lyukba négy-négy kerül.
2. A vetés úgy történik, hogy a játékos a saját térfelén lévő lyukak közül kiválaszt egy tetszőleges nem üreset, és a benne lévő figurákat az óra járásával ellentétes irányban egyenként elkezdi szétosztani.
3. Ha több mint tizenkét figura volt a lyukban (azaz egy teljes kört megtehető a táblán) akkor, azt a lyukat, amiben eredetileg a figurák voltak a vetés során kihagyjuk. Így ez a lyuk üres marad.

4. Elfogásra akkor kerül sor, ha a vetés során az utolsónak letett figura olyan lyukba kerül, amiben addig egy vagy kettő figura volt. A vetés után két, illetve három figura lesz benne. Ilyenkor ezeket a figurákat zsákmányként a gyűjtőrekeszbe tesszük. Az elfogás az előző, egymás követő lyukakra is kiterjed, amikben a vetés után két vagy három figura került.

5. Ha az ellenfél összes lyuka üressé vált, akkor a játékosnak olyan lépést kell tenni, ami az ellenfelének figurát ad. Ha ilyen lépés nem lehetséges, akkor a játékos elfogja az összes táblán lévő figurát, és a játék véget ér. Abban az esetben, ha a megmaradt figurák vég nélkül haladnak körbe a két játékos szétosztja őket egymás között.

6. Különben a játék akkor ér véget, amikor az egyik játékosnak sikerül huszonöt figurát elfognia.

7. A játékot az nyeri, aki több mint huszonnégy figurát fogott el. Ha mindkét játékosnak huszonnégy figurája van, az eredmény döntetlen.

Kalah

A Kalah-t Kalahának és Mancalának is nevezik. Amikor az Egyesült Államokban valaki Mancaláról beszél, valószínűleg a Kalah-t érti alatta. A Nyugati közönségnek a XX. század elején mutatta William Julius Champion Jr. Néha *Warri*nak vagy *Awari*nak is nevezik, bár ezek az *Oware* elnevezései. A kezdőjátékosnak van nyerő stratégiája a három-öt mag/gödör verziókban. A hat mag/gödör verzió még megoldatlan, de valószínűleg itt is a kezdőjátékos rendelkezik a nyerő stratégiával.

Tábla: soronként hat-hat lyukkal (gödörrel), a tábla végében egy-egy gyűjtőrekesszel (kalah), minden játékosnak a jobb kéz felé eső rekesz az övé

Játékosok: két játékos (vagy két csapat), mindkét játékos az előtte lévő hat lyukkal játszik

Figura: általában 12×3 mag vagy zseton, de izgalmasabb 12×4, 12×5 vagy 12×6 változattal játszani

Cél: több magot fogjunk el, mint az ellenfél

Szabályok:

1. Általában az előző játszma nyertese kezdi a játékot.
2. A vetés úgy történik, hogy a játékos a saját térfelén lévő lyukak közül kiválaszt egy tetszőleges nem üreset, és a benne lévő figurákat az óra járásával ellentétes irányban egyenként elkezdi szétosztani. Amikor elfogytak a játékos saját gödrei, a saját gyűjtőgödrébe kell elhelyeznie egy darab babszemet. Az ellenfél gyűjtőjébe viszont nem vethet.
3. Abban az esetben, ha az utolsó babszem a saját gyűjtőbe kerül, ismételten a játékos következik.
4. Akkor, ha az utolsó babszem a játékos térfelén egy olyan gödörbe kerül, ami addig üres volt, és az ellenfél szemben lévő gödrében van legalább egy babszem, akkor mindkét gödör tartalma átkerül a játékos gyűjtőgödrébe.
5. A játék véget ér, ha az egyik játékos térfelén elfogytak a babszemek. Ilyenkor az ellenfél gödreinek tartalma átkerül a játékos gyűjtőgödrébe.

Döntetlen lehetséges.

Bao

A játékot főleg Kenya és Tanzánia területén játsszák, az egyik legbonyolultabb mancala.

Tábla: négysoros tábla, soronként nyolc gödörrel, a játékoshoz a hozzá legközelebb eső két sor tartozik. A két középső sort belső, a másik kettőt külső soroknak nevezik. A belső sorban lévő, a játékos szemszögéből balra az ötödik gödröt *nyumbá*nak, háznak, hívják, és néhány táblán különbözik az alakja. Az belső sorok első és utolsó gödrei a *kichwá*k, a másodikak és az utolsó előttiek pedig a *kimbi*k. Egyes táblákon található egy raktárnak nevezett gödör vagy a játékos jobbján vagy előtte.

Játékosok: két játékos, az egyiket északinak, a másikat délinek nevezik.

Figura: összesen hatvannégy mag, szuahéli nyelven *kete*.

Cél: Az a játékos a nyertes, amelyik előbb kiüríti ellenfele belső sorát vagy eléri, hogy az ellenfele ne tudjon többet lépni.

Szabályok

1. Vetés

Történhet az óra járásával megegyező és ellentétes irányban, de vetés közben nem

változtathatunk irányt. A vetés csak a saját gödröket érintheti, és olyan gödörből, amiben csak egy darab mag van, nem vethetünk.

2. Elfogás

Akkor történik, ha a vetés egy belső, nem üres gödörben ért véget, és az ellenfél szemben lévő belső gödre nem üres. Ilyenkor az ellenfél gödréből minden magot el kell távolítani, és el kell őket vetni a saját oldalunkon. Az ilyen vetés mindig valamelyik kichwában kezdődik, a bal oldali esetén az óra járásával megegyező, különben ellentétes irányban. Ha a bal kichwát vagy kimbit fogtuk el, a bal kichwában kezdődik a vetés, a jobb kichwa vagy kimbi esetén pedig a jobban. Különben az irány a játék szakaszától függ.

3. Lépés

Egy lépés a vetések és az elfogások sorozata. Egy lépés akkor ér véget, ha az utolsó mag egy addig üres gödörbe érkezik. Ekkor az ellenfél következik.

4. Endelea

Ha a vetés egy nem üres gödörben ért véget, viszont az elfogás nem megengedett, akkor a gödörből kivesszük az összes magot, és az addigi irányt megtartva folytatjuk a vetést. Ezt a folytonosságot nevezzük endeleának. Az endelea során, ha az utolsó mag olyan gödörbe kerül, amiben hat vagy több mag van, akkor a játékos eldöntheti, hogy folytatja-e a játékot. Az endelea véget ér, ha a vetés egy takatia gödörnél ér véget, vagy elfogás válik lehetségessé. Ekkor az elfogás szabályi szerint folytatódik a játék.

5. Takata vagy takasa

Ha a játékos nem tud elfogni, akkor azt takatának nevezzük. Ha a lépés nem elfogással kezdődik, akkor az elfogás nem megengedett a lépés során. Takata során a játékos endeleát lép, amíg vége nem lesz. A takata során az irány nem változhat. A további szabályokat az határozza meg, hogy namua vagy mtaja szakaszban vagyunk.

Namua szakasz

1. A játék kezdetétől tart addig, ameddig az összes mag a táblára kerül, és a raktárak ki nem ürülnek. A játék kezdetén huszonkét mag van a raktárban, hat a nyumbában és a kettő-kettő nyumbát jobbról követő két gödörben. Ilyenkor a déli

játékos kezd.

2. Vetés

Ebben a szakaszban a vetés úgy történik, hogy a játékos a raktárából kivesz egy magot és a saját belső sorának az egyik gödrébe helyezi. Csak akkor megengedett a nyumbába helyezni, ha az ellenféltől magokat tudunk elfogni.

3. Elfogás

Elfogás esetén, a játékosnak az a fentebb említett szabályokat kell követni. Ha a magot nem kichwába vagy kimbibe helyeztük el, a játékos eldöntheti melyik kichwából kezd el vetni.

4. Takata

A takata, amiről akkor beszélünk, amikor nem tudunk elfogni, nem kezdődhet a külső sorokban. Amikor az egyetlen nem üres gödör a kichwák valamelyike, akkor a vetés nem mehet a külső sor irányába, mert így a belső sor kiürülne, és a játékos elvesztené a játékot.

A takata nem indulhat a nyumbából, ha hat vagy több mag van benne, valamint olyan gödörből, amiben csak egy darab mag van, kivéve, ha nincs olyan gödör a belső sorban, amiben egynél több mag van, vagy a ház még mindig a játékos birtokában van (akkor is, ha hatnál kevesebb mag van benne).

5. A nyumba (ház)

A játék kezdetén mindkét játékos birtokolja a saját házát.

A játékos elveszti a házát, ha kiüríti, vagy az ellenfele elfogja.

Ha a játékos birtokolja a nyumbát, és a vetés a nyumbában ér véget, akkor a lépés takata során véget ér, ha a házban hat vagy több mag van vagy egy elfogás során nincs több lehetőség vagy nyumbában válna ez lehetségessé, és játékos azt mondja, megáll. Ha ebben az esetben játékos nem kíván megállni, és azt mondja „játsszuk ki a házat”, a ház kiürül és a magokat elvetik.

Mtaji szakasz

1. Akkor kezdődik, amikor az összes mag tehát, mind a hatvannégy játékba kerül.
2. Vetés elfogással

Egy lépés vagy a belső vagy a külső sor egyik gödréből indul, amiben egynél több, de kevesebb, mint tizenhat mag található. Egy bizonyos irányú vetés után, az utolsó

magnak lehetővé kell tenni az elfogást (elfogni kötelező). Ha ez nem a négy központi gödörből történt és óra járásával ellentétes irányban akarjuk az elfogott magokat szétosztani, akkor a jobb, különben a bal kichwából indul a vetés.

3. Takata

Ha az elfogás nem lehetséges, a játékosnak takatát kell játszani. Ha ez a belső sorban nem megoldható, akkor a külső sorban is lehet takatázni.

4. A nyumba

Ha a házat még mindig a játékos birtokolja, addig az övé marad, amíg el nem fogják. Ebben a szakaszban a namua szabályok nem érvényesek, nincs lehetőség takatiára.

5. Takatia vagy Takasia

Ha a játékos takatára kényszerül, megjátszhatja, hogy

Az ellenfél következő lépésben szintén takatára kényszerüljön

Az ellenfélnek ezután pontosan egy gödre fogható el,

Ilyenkor ez ellenfél nem ürítheti ki a takatia gödröt.

Azonban ez a gödör

Nem lehet az ellenfél egyetlen játszható gödre

Nem lehet az ellenfél háza

Omweso (Mweso, Mweisho, Coro)

Uganda nemzeti mancala játéka. Buganda 30. királyának, I. Mutasának a kedvenc játéka volt. Ő uralkodott 1862-ben, amikor az angol kutató, Spere, elérte Africa ezen partjait. A 19. század legjobb játékosa I. Mutasa és II. Mwanga miniszterelnöke Mukasa volt.

Tábla: négysoros tábla, soronként nyolc gödörrel, a játékos térfele a hozzá közelebb eső két sor

Játékosok: két személy vagy két csapat

Figurák: összesen hatvannégy mag. Kezdetben a játékoshoz eső legközelebbi sorban lévő gödreiben négy-négy darab mag.

Cél: az a játékos nyer, amelyik utoljára képes szabályos lépést megtenni, például úgy, hogy elfogja az ellenfele összes magját.

Szabályok

1. Bármelyik oldal kezdhet, ha voltak előző játszmák általában a vesztes kezd.
2. Vetés

Csak olyan gödörből vethetünk, amiben minimum két mag van. Az óra járásával ellenkező irányban történik a vetés, és csak a játékos saját gödreit érintheti. Ha az utolsó mag egy olyan gödörbe kerül, amiben korábban volt mag, akkor a gödör tartalmát elvetjük.

3. Elfogás

Ha az utolsó mag egy nem üres belső gödörbe kerül, és az ellenfél egy oszlopban lévő mindkét gödre nem üres, akkor az ezeket kiürítjük, és a saját oldalon elvetjük őket, attól a gödörtől kezdve, ahol az előző lépést befejeztük.

4. Fordított elfogás

Ahelyett, hogy az óra járásával ellentétes irányban vetnénk, a játékos az óra járásával megegyező irányban vet a négy legbaloldalibb gödrében. Ez csak akkor megengedett, ha közvetlenül elfogáshoz veled

5. Győzelmi feltételek

Okwa bulijo: a normális útja a játék megnyerésének. Ilyenkor az veszít, akinek nem tud lépni. Ez egy pontot ér.

Okutema: A győzelmet Emitwe-Ebirével (vágd le a fejét) érjük el, ha a játékos az ellenfélének mind a négy zárógödrének tartalmát elfogja. Ilyenkor két pontot kap.

Akawumbi: Az Emitwe-Ebire speciális esete, akkor beszélünk róla, ha a játékos az ellenfele összes gödrének tartalmát elfogja. Az utolsó magoknak zárógödörben kell lenniük. Ilyenkor a játékos tizenkét (egyes versenyekben csak hat) pontot kap.

Akakyala: néhány bajnokságban, a játékos nyer, ha két egymást követő lépésben elfog, és az ellenfele még nem fogta el az első magjait. Ez két pontot ér.

Okukoneeza: néhány bajnokságban, a játékos nyer, ha van egy gödre, amiben három mag van, ezt egy gödör követi, amiben két mag van, és ezt egy olyan, amiben egy mag van, és még az ellenfél nem fogta el egyetlen magját sem.

6. Az nyer, akinek előbb gyűlik ki tizenkét pont

Léteznek véget nem érő lépések. Egy bajnokságon egy játékosnak három perce van befejezni a lépést – különben a játék érvénytelen.

A vég nélküli omweso lépéseket matematikusok is elemezték.

Woaley (valéj)

A woaley az egyik legismertebb mancala változat a világon. Nevét egy afrikai faluról kapta. Három változata ismert. Ebből egyet, az alaptípust ismertetném.

Tábla: kétsoros tábla, soronként hat-hat házzal, a két oldalon gyűjtőházzal.

Játékosok: két személy, vagy két csapat (ez esetben a létszám tetszőleges), a játékos az előtte lévő hat házzal játszik, és a tőle jobbra lévő gyűjtőház a sajátja.

Figura: összesen negyvennyolc, tehát mindkét félnek külön-külön huszonnégy. A játék kezdetén minden nem gyűjtőházban négy-négy figura van

Cél: A játék véget ér, ha már egyik játékosnak sincs lehetősége kettő vagy négy babszemet tartalmazó ház elfogására. Tehát, addig tart, amíg mindkét térfélen található olyan ház, amiben legalább két figura van, és véget ér, amikor a házak kiürülnek, vagy csak egy figura marad bennük.

Szabályok:

1. Ugyanúgy, mint az eddig megismert változatokban, itt is az a lényeg, hogy a házakból kiemeljük a figurákat, és szétosszuk őket. A játék az óra járásával ellenkező irányba halad. Csak olyan házat választhatunk, amiben minimum két darab figura van. A saját térfelünkön hozzáérhetünk, és kézzel megszámolhatjuk a figurákat. Az ellenfél területén ez nem lehetséges.
2. A játékos a saját térfeléről kezdi a játékot. Ezután minden játékosnak onnan kell folytatnia a szétosztást, ahol az ellenfelének utolsó figurája leesett.
3. A két játékos mindig felváltva játszik.
4. Az elfogásra akkor kerül sor, ha az utolsó figura elhelyezése után, a játékos az utolsó házban lévő figurák számát kettőre vagy négyre egészíti ki. Ilyenkor ezek a gyűjtőházba kerülnek. Elfogni a saját és az ellenfél térfeléről is lehet. Zsinórban is történhet elfogás, ilyenkor a megelőző házakban is kettőre vagy négyre kell kiegészíteni a figurák számát. Ezeknek megszakítás nélkül kell követniük egymást.

A szakdolgozatban a woaley alaptípusát dolgozom ki.

III. fejezet

A reprezentáció

Milyen játék is a woaley?

Mielőtt még bármihez is kezdenénk, nem árt tisztázni, hogy egyáltalán milyen játékkal is foglalkozunk. Ezt úgy tehetjük meg, hogy különböző szempontok alapján megvizsgáljuk. Azt rögtön tisztázhatjuk, hogy a játékosok száma alapján, kétszemélyes játékról van szó.

Mivel a játékosok befolyásolni tudják a woaley kimenetét, stratégiai játékról beszélünk.

A játékosok egymás után következnek.

A véletlennek nincs szerepe.

A játékosok pontosan tudják, hogy a saját házaikban hány darab figura van, ez ellenfelét csak a szemükkel tudják megszámolni, illetve emlékezhetnek arra, hogy az egyes lépések során mennyi került bele.

A játék úgy kezdődik, hogy minden nem gyűjtőházban négy figura van, és akkor fejeződik be, amikor a nem gyűjtőházakban kettőnél kevesebb figura kerül.

A kezdőjátékos a játék kezdetén csak a saját térfeléről választhat ki egy házat. Később mindkét játékos abból a házból folytathatja, ahol az előző abbahagyta, illetve, ha ez nem lehetséges, bárhonnan, ahol nem nulla a figurák száma.

Döntetlen lehetséges, a játékos győzelmeinek és vereségeinek száma nő.

A játszmák a legtöbb esetben véges lépésben véget érnek, azonban bele lehet futni olyan szituációba is, ahol nem. Ezért érdemes megfontolni egy lépésszám korlát bevezetését vagy egy „Befejezés” opciót.

A játékban az egyes lépések állásból állásba visznek át.

A játék reprezentációja

A woaley egy kétszemélyes játék, ebből következik, hogy egy mesterséges intelligenciabeli probléma, melyeket első lépésben meg kell fogalmazni, reprezentálni kell.

Ehhez többféle módszert használhatunk, például állapottér-reprezentációt, elsőrendű logikai leíró nyelvet vagy problémaredukciót. Az én választásom a legelső lehetőségre esett, mivel tanulmányaim során ezt használtam a legtöbbet.

Először definiáljuk az állapottér-reprezentáció fogalmát. Adott egy p probléma. Azt mondjuk, hogy p problémát állapottér-reprezentáltuk, ha megadunk hozzá egy $\langle A, kezdő, C, O \rangle$ négyest, ahol

$A \neq \emptyset$

$kezdő \in A$, *kezdőállapot*

$C \subset A$, *célállapotok halmaza*

$O \neq \emptyset$, *az operátorok halmaza*

Jelölése: $p = \langle A, kezdő, C, O \rangle$

Tehát első lépésben meg kell határoznunk az állapotteret.

Tudjuk, hogy két játékosunk van. Legyenek ők „*A”* és „*B”*.

$J = \{A, B\}$, és $j \in J$, az a játékos, aki éppen lépni következik.

A játéktáblánk két soros. Ebből egyik „A”, a másik „B” sora. Minden sorban hat db háznak nevezett mélyedés van, amelyek a figurákat tartalmazzák. Egy ilyen házban 0 és 48 között változhat a figurák száma.

$$H_{A,1} = \{0,1,2,\ldots 46,47,48\}$$
$$H_{A,2} = \{0,1,2,\ldots 46,47,48\}$$
$$\vdots$$
$$H_{A,6} = \{0,1,2,\ldots 46,47,48\}$$
$$H_{B,1} = \{0,1,2,\ldots 46,47,48\}$$
$$\vdots$$
$$H_{B,6} = \{0,1,2,\ldots 46,47,48\}$$

A játékosok egy-egy gyűjtőházzal is rendelkeznek, amelyekben a figurák száma 0 és 48 között lehet.

$$H_{A,gy} = \{0,1,2\ldots 46,47,48\}$$
$$H_{B,gy} = \{0,1,2\ldots 46,47,48\}$$

Tudjuk, hogy a játékosnak ott kell folytatnia a lépést, ahol az ellenfele abbahagyta, kivéve, ha még nem léptünk vagy elfogás történt. Ezért szükséges megadni, hogy a játékos abból a pozícióból folytatja-e a játékot, ahol az előző abbahagyta, és ha igen, akkor mi volt ez a pozíció.

$H_{foly}=\{igen, nem\}$

$H_{sorpoz}=\{0,1,2\}$, ahol az 1-es „A”, a 2-es „B” sorára utal.

$H_{hazpoz}=\{0,1,2,3,4,5,6\}$

Ahol a (0,0) pozíció azt jelenti, hogy a játék még nem kezdődött el, vagy elfogás történt.

Ezeknek a halmazoknak a segítségével felírhatunk egy Descartes szorzatot, amelynek részhalmaza az állapotok halmaza.

$$A=\{H_{A,gy}\times H_{A,1}\ldots\times H_{A,6}\times H_{B,1}\ldots\times H_{B,6}\times H_{B,gy}\times H_{foly}\times H_{sorpoz}\times H_{hazpoz}\times J\}=$$

$$\left\{\left(0\begin{matrix}000000\\000000\end{matrix}0, igen, 0, 0, A\right), \left(0\begin{matrix}000000\\000000\end{matrix}0, igen, 0, 0, B\right)\ldots\left(0\begin{matrix}444444\\444444\end{matrix}0, nem, 0, 0, A\right)\ldots\right\}$$

A kényszerfeltételek megadásával tudhatjuk meg, hogy mely elemek lesznek állapotok.

1. Pontosan 48 db figurával rendelkezünk.

   $$h_{a,gy}+h_{a,1}+h_{a,2}+h_{a,3}+h_{a,4}+h_{a,5}+h_{a,6}+h_{b,gy}+h_{b,1}+h_{b,2}+h_{b,3}+h_{b,4}+h_{b,5}+h_{b,6}=48$$

2. Ha sorpozíció 0, akkor a házpozíció csak 0 lehet, és ha a házpozíció 0 akkor a sorpozíció is csak 0 lehet.

   $$h_{sorpoz}=0\supset h_{hazpoz}=0\wedge h_{hazpoz}=0\supset h_{sorpoz}=0$$

3. Ha a játékosnak nem ott kell megkezdeni meg a saját lépését, ahol az ellenfél abbahagyta, akkor a sor- és a házpozíció 0.

   $$h_{foly}=nem\supset h_{sorpoz}=0\wedge h_{hazpoz}=0$$

4. Ha a játékosnak ott kell megkezdeni meg a saját lépését, ahol az ellenfél abbahagyta, akkor a sor- és a házpozíció nem lehet.

   $$h_{foly}=igen\supset h_{sorpoz}\neq 0\wedge h_{hazpoz}\neq 0$$

5. Ha a sor- és a házpozíció nem 0, akkor a pozíciónak megfelelő ház nem lehet üres.

   $$h_{sorpoz}\neq 0\wedge h_{hazpoz}\neq 0\supset h_{h_{sorpoz},h_{hazpoz}}\neq 0$$

6. A gyűjtőkben csak páros számú figura lehet

   $$\forall i\, h_{i,gy}\ \ MOD\ \ 2=0$$

Tehát h állapot, ha

$$A=\{h|h=\left(h_{A,gy}\begin{matrix}h_{B,6}h_{B,5}h_{B,4}h_{B,3}h_{B,2}h_{B,1}h_{B,gy}, h_{foly}, h_{sorpoz}, h_{hazpoz}, j\\h_{A,1}h_{A,2}h_{A,3}h_{A,4}h_{A,5}h_{A,6}\end{matrix}\right)\in$$

$$H_{A,gy}\times H_{A,1}\ldots\times H_{A,6}\times H_{B,1}\ldots\times H_{B,6}\times H_{B,gy}\times H_{foly}\times H_{sorpoz}\times H_{hazpoz}\times J\wedge$$

$$kényszerfeltétel\,(h)\}$$

Ezzel megadtuk a probléma állapoterét.

Ezután a kezdőállapotot kell meghatároznunk. Az A játékos a kezdőjátékos, ő következik a lépésben. Kezdetben a gyűjtőkben nincs figura, minden más házban 4-4 db van, és a játék nem halad folytonosan.

$$kezdő = \left(0 \begin{matrix} 4\,4\,4\,4\,4\,4 \\ 4\,4\,4\,4\,4\,4 \end{matrix} 0, nem, 0, 0, A \right) \in A$$

Akkor érünk el célállapotot, minden nem gyűjtőházban kettőnél kevesebb figura van.

$$C = \left\{ \left(h_{A,gy} \begin{matrix} h_{B,6} h_{B,5} h_{B,4} h_{B,3} h_{B,2} h_{B,1} \\ h_{A,1} h_{A,2} h_{A,3} h_{A,4} h_{A,5} h_{A,6} \end{matrix} h_{B,gy}, h_{foly}, h_{sorpoz}, h_{hazpoz}, j \right) \vee \forall j, i \;\; i = 1...6, \;\; h_{j,i} < 2 \right\}$$

$$\subset A$$

Az a játékos a nyertes, akinek gyűjtőházában több figura van.

Végül meg kell adnunk az operátorokat.

Csupán egyetlen operátorra lesz szükségünk, mely a „vetés” nevet kapja. Három paramétere lesz, ezekben megadjuk, hogy melyik játékos vet, melyik sorból, és melyik házból.

A következő operátorok fordulhatnak elő a játék során:

$$O = \{ vetés(A,1,1), vetés(A,1,2), vetés(A,1,3), vetés(A,1,4), vetés(A,1,5), vetés(A,1,6)$$
$$vetés(B,1,1), vetés(B,1,2), vetés(B,1,3), vetés(B,1,4), vetés(B,1,5), vetés(B,1,6) \}$$

Az operátor alkalmazásainak lesznek bizonyos előfeltételei.

Ezek:

1; olyan házból nem kezdhetünk vetni, amiben nincs figura.

2; ha még egyik játékos sem tett lépést, akkor a kezdőjátékos csak a saját sorából választhat.

3; ha a játék folytonosan halad, akkor a következő vetést onnan kell kezdeni, ahonnan az ellenfél abbahagyta.

4; az operátorban megadott játékosnak és a lépésben következő játékosnak meg kell egyeznie.

Ezekhez adjuk meg a pszeudokódot!

```
FUNCTION vetés_előfeltétel(woaleyállapot, vetésoperátor)
1; IF woaleyállapot→házak[vetésoperátor→sor]   [vetésoperátor→ház]=0 THEN
2;     RETURN FALSE
3; END IF
4; IF megegyezik(woaleyállapot, kezdőállapot)=0 AND
       játékos_sora(woaleyállapot→játékos) ≠ vetésoperátor→sor THEN
5;            RETURN FALSE
6; END IF
7; IF woaleyállapot→folyamatos AND woaleyállapot→sorpozíció ≠
       vetésoperátor→sor
       AND woaleyállapot→házpozíció ≠  vetésoperátor→ház THEN
8;            RETURN FALSE
9; END IF
10; IF woaleyállapot→játékos ≠  vetésoperátor→játékos
11;    RETURN FALSE
12; END IF
13; RETURN TRUE
END FUNCTION


FUNCTION megegyezik(woaleyállapot1, woaleyállapot2)
1; IF woaleyállapot1→játékos ≠ woaleyállapot2→játékos THEN
2;     RETURN FALSE
3; END IF
4; IF woaleyállapot1→folyamatos ≠ woaleyállapot2→folyamatos THEN
5;     RETURN FALSE
6; END IF
7; IF woaleyállapot1→sorpozíció ≠ woaleyállapot2→sorpozíció THEN
8;            RETURN FALSE
9; END IF
10; IF woaleyállapot1→ házpozíció ≠ woaleyállapot2→házpozíció THEN
11;           RETURN FALSE
12; END IF
13; FOR i = 1 TO 2 DO
14; IF woaleyállapot1→gyűjtőház[i]≠woaleyállapot2→gyűjtőház[i] THEN
15;           RETURN FALSE
```

```
16;     END IF
17;     FOR j = 1 TO 6 DO
18;           IF woaleyállapot1→ házak[i][j]≠
              woaleyállapot2→házak[i][j] THEN
19;                 RETURN FALSE
20;           END IF
21;     END FOR
22; END FOR
23; RETURN TRUE
END FUNCTION


FUNCTION játékos_sora(játékos)
1; IF játékos='A' THEN
2;      RETURN 1
3; ELSE
4;      RETURN 2
5; END IF
END FUNCTION
```

Ha az előfeltételek teljesülnek, akkor alkalmazhatjuk az operátort.

```
FUNCTION alkalmaz_operátor(woaleyállapot, vetésoperátor)
1; RETURN vetés(woaleyállapot, vetésoperátor→sor, vetésoperátor→ház)
END FUNCTION


FUNCTION vetés(woaleyállapot, sorkoordináta, házkoordináta)
1; elvethető←woaleyállapot→ házak[sorkoordináta]     [házkoordináta]
2; utolsósor←sorkoordináta
3; utolsóház←házkoordináta
4; woaleyállapot→ házak[sorkoordináta][házkoordináta] ←0
5; WHILE elvethető>0 DO
6;      IF (elvethető>0) THEN
7;            woaleyállapot←vetsor(woaleyállapot, sorkoordináta,
              házkoordináta, MIN(6, elvethető))
8;            utolsósor←sorkoordináta
9;            utolsóház←MIN(6, elvethető)
```

```
10;           elvethető←elvethető – 6 + házkoordináta
11;     END IF
12;     IF (elvethető>0) THEN
13;           woaleyállapot←vetsor(woaleyállapot,
              másiksor(sorkoordináta), 1, MIN(6, elvethető))
14;           utolsósor←másiksor(sorkoordináta)
15;           utolsóház←MIN(6, elvethető)
16;           elvethető←elvethető – MIN(6, elvethető)
17;     END IF
18;     IF (elvethető>0) THEN
19;           woaleyállapot←vetsor(woaleyállapot, sorkoordináta, 1,
              MIN(házkoordináta, elvethető))
20;           utolsósor←sorkoordináta
21;           utolsóház←MIN(házkoordináta, elvethető)
22;           elvethető←elvethető – MIN(házkoordináta,elvethető))
23;     END IF
24; END WHILE
25; RETURN begyűjt(woaleyállapot, utolsósor, utolsóház)
END FUNCTION


FUNCTION vetsor(woaleyállapot, sor, honnan, meddig)
1; FOR i=honnan TO meddig DO
2;      woaleyállapot→házak[sor][i] ← woaleyállapot→házak[sor][i] + 1
3; END FOR
4; RETURN woaleyállapot
END FUNCTION


FUNCTION másiksor(sorkoordináta)
1; IF sorkoordináta=1
2;      RETURN 2
3; END IF
4; RETURN 1
END FUNCTION


FUNCTION begyűjt(woaleyállapot, utolsósor,  utolsóház)
1; IF woaleyállapot→házak[utolsósor][utolsóház]=2 OR
```

```
       woaleyállapot→házak[utolsósor][utolsóház]=4 THEN
2;     woaleyállapot→folyamatos←FALSE
3;     woaleyállapot→sorpozíció←0
4;     woaleyállapot→házpozíció←0
5;     IF begyűjtsor(woaleyállapot, woaleyállapot→játékos, utolsósor,
       utolsóház, 1)=TRUE THEN
6;           IF begyűjtsor(woaleyállapot, woaleyállapot→játékos,
             másiksor(utolsósor), 6, 1)=TRUE THEN
7;                 begyűjtsor(woaleyállapot,woaleyállapot→
                   játékos, utolsósor, 1, utolsóház+1)
8;           END IF
9;     END IF
10; ELSE
11;    woaleyállapot→folyamatos←TRUE
12;    woaleyállapot→sorpozíció←utolsósor,
13;    woaleyállapot→házpozíció← utolsóház
14; END IF
15; RETURN woaleyállapot
END FUNCTION


FUNCTION begyűjtsor(woaleyállapot, játékos, sor,  honnan, meddig)
1; FOR i=honnan DOWNTO meddig DO
2;     IF woaleyállapot→házak[utolsósor][utolsóház]=2 OR
        woaleyállapot→házak[utolsósor][utolsóház]=4 THEN
3;           woaleyállapot1→gyűjtőház[játékos_sora(játékos)] ←
       woaleyállapot1→gyűjtőház[játékos_sora(játékos)]     +
             woaleyállapot1→házak[sor][i]
4;     woaleyállapot1→házak[sor][i] ←0
5;     ELSE
6;           RETURN FALSE
7;     END IF
8; END FOR
9; RETURN TRUE
END FUNCTION
```

Mivel a *vetés(woaleyállapot, sorkoordináta, házkoordináta)* operátor állapotból állapotot állít elő, ezért a kényszerfeltételek ellenőrzését elhagyhatjuk.

Ezzel megadtuk a woaley nevű játék egy lehetséges állapottér-reprezentációját.

IV. fejezet

A keresőalgoritmus

Negamax

A játék-reprezentációnk elkészült, azonban ez nem elég ahhoz, hogy játszani tudjunk a gép ellen. Ehhez szükségünk van egy algoritmusra, ami lépést ajánl ellenfelünknek. A választásom a negamax algoritmusra esett, mely hatékony és könnyű implementálni.

A negamax hasonlít a minimaxra, és a kétszemélyes játékok azon tulajdonságán alapulnak, hogy zérusösszegűek. Úgy működik, hogy a támogatott játékosnak, nevezzük „J”-nek, egy adott állásban egy „elég jó” lépést ajánl. A definíció alapján ugyanabban a játékban az A egy játékállásának a jósága, a B azonos játékállásás jóságának a negáltja.

Az algoritmushoz meg kell adnunk a játék-reprezentációját, „J” azon „a” állását, ahol lépni következik, az állások jóságát becslő heurisztikát, és egy mélységi korlátot.

Az algoritmus pszeudokódja:

```
FUNCTION negamax( ⟨A, kezdő, C, O⟩ , állapot, mélység, h)
1; max→-∞
2; operátor→NIL
3; FOR ALL o∈O DO
4;      IF előfeltétel(állapot, o) THEN
5;             új_állapot→alkalmaz(állapot, o)
6;             v→-negamax_érték( ⟨A, kezdő, C, O⟩ , új_állapot, mélység-1, h)
7;             IF v>max THEN
8;                    max→v
9;                    operátor→o
10;            END IF;
11;     END IF;
12; END FOR
13; RETURN operátor
END FUNCTION
```

```
FUNCTION negamax_érték( ⟨A, kezdő, C, O⟩ , állapot, mélység, h)
1; IF   állapot∈V   OR mélység=0 THEN
2; RETURN h(állapot)
3; END IF
4; max→−∞
5; FOR ALL  o∈O  DO
6;      IF előfeltétel(állapot, o) THEN
7;             új_állapot→alkalmaz(állapot, o)
8;             v→−negamax_érték( ⟨A, kezdő, C, O⟩ , új_állapot, mélység-1, h)
9;             IF v>max THEN
10;                   max→v
11;            END IF;
12;     END IF;
13; END FOR
14; RETURN max
END FUNCTION
```

Alfa-béta vágás

Ezzel az algoritmussal még hatékonyabbé lehet tenni a keresést, ugyanis csökkenthetjük a keresőfában a bejárt csomópontok számát. Ezt úgy tehetjük meg, hogy amikor egyetlen olyan lehetőséget találunk, a lépés rosszabbnak bizonyul, mint egy előző lépés, leállítjuk a kiértékelést. Az alfa-béta vágás használatával nem változtatjuk meg az algoritmus végeredményét.

Az alfa-béta vágást megvalósító negamax peszeudókódja

```
FUNCTION negamaxalfabeta( ⟨A, kezdő, C, O⟩ , állapot, mélység, h)
1; max→−∞
2; operátor→NIL
3; FOR ALL  o∈O  DO
4;      IF előfeltétel(állapot, o) THEN
5;             új_állapot→alkalmaz(állapot, o)
```

```
6;            v→-negamax_értékalfabeta( ⟨A, kezdő, C, O⟩ ,
       új_állapot, mélység-1, h, -∞, ∞)
7;            IF v>max THEN
8;                   max→v
9;                   operátor→o
10;           END IF;
11;    END IF;
12; END FOR
13; RETURN operátor
END FUNCTION


FUNCTION negamax_értékalfabeta( ⟨A, kezdő, C, O⟩ , állapot, mélység, h,
alfa, beta)
1; IF  állapot∈V  OR mélység=0 THEN
2; RETRUN h(állapot)
3; END IF
4; FOR ALL o∈O DO
5;       IF előfeltétel(állapot, o) THEN
6;                new_állapot→alkalmaz(állapot, o)
7;                alfa→MAX(-negamax_érték( ⟨A, kezdő, C, O⟩ ,
         új_állapot,mélység-1, h, -beta, -alfa), alfa)
8;                IF alfa>beta THEN
9;                         RETURN alfa
10;               END IF;
11;      END IF;
12; END FOR
13; RETURN alfa
END FUNCTION
```

V. fejezet

A heurisztika

Ha kész az állapottér-reprezentáció, megvan a keresőalgoritmus, akkor csak az van hátra, hogy meghatározzuk az egyes állások jóságértékét. A játék egyszerűségét figyelembe véve ez nem lesz nehéz.

Tudjuk, hogy akkor nyerünk ha több figura van a gyűjtőnkben, mint az ellenfelünknek. Tehát az jó nekünk, ha az egyes állásokban több figurával rendelkezünk, mint a másik játékos, azonban az rossz, ha kevesebbel.
A heurisztikánk pszeudokódja tehát:

```
FUNCTION heurisztikanegamax (woaleyállapot)
1; RETURN woaleyállapot→gyűjtőház[játékos_sora
              (woaleyállapot→játékos)]-
       woaleyállapot→gyűjtőház[másiksor(
       játékos_sora(woaleyállapot→játékos))]
END FUNCTION
```

VI. fejezet

A játék „tanul”

Van állapotér-reprezentációnk, keresőalgoritmusunk és heurisztikánk. A játék ezekkel is elkészíthető, azonban gondoljunk bele, hogy milyen lenne, ha „tanulna” azaz megjegyezné a játék során előfordult játékállásokat és ezek jóságértékét, mondjuk úgy, hogy ezeket egy adatbázisban tárolja. Így amikor a keresőnk előállít egy új állapotot meg kell néznie, hogy az bekerült-e már az adatbázisba, ha igen, akkor kiolvassa a hozzá tartozó jóságértéket, és eldönti, hogy folytassa-e tovább a keresést vagy sem. Amennyiben az állás még nincs benne az adatbázisban, folytatja a keresést, és eldönti, hogy az állás egyáltalán bekerüljön-e. A játék folyamán először a célállapotok kerülnek be, és egyre közelebb kerülünk majd a kezdőállapothoz.

Minden állás maximum egyszer szerepelhet az adatbázisban, és minden álláshoz tartozik egy érték. Tehát kulcs-érték párokról beszélünk. Tárolhatnánk az állásunkat, úgy hogy sztringgé alakítjuk, az értéket pedig egész számként.

Én a következő algoritmust használtam az átalakításhoz.

```
FUNCTION átalakít(woaleyállapot)
1; kód←””
2; kód←összefűz(kód, woaleyállapot→játékos)
3; kód←összefűz(kód, átalakítfolyamatos(woaleyállapot→folyamatos))
4; kód←összefűz(kód, átalakítpozíció(woaleyállapot→sorpozíció,
 woaleyállapot→házpozíció))
5; kód←összefűz(kód, átalakítjátéktábla(woaleyállapot→ házak,
 woaleyállapot→gyűjtőház))
6; RETURN kód;
END FUNCTION


FUNCTION átalakítfolyamatos(folyamatos)
1; IF folyamatos = TRUE THEN
2; RETURN ”T”
3; ELSE
```

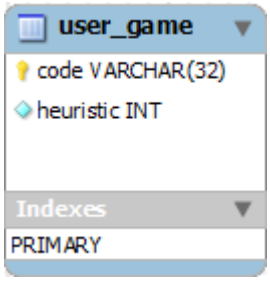
```
4; RETURN ”F”
5; END IF;
END FUNCTION


FUNCTION átalakítjátéktábla(házak[2][6], gyűjtőházak[2])
1; kód←””
2; FOR i=1 TO 2 DO
3;      FOR j=1 TO 6 DO
4;             kód összefűz(kód, átalakítszám(házak[i][j]))
5;      END FOR
6;      kód összefűz(kód, átalakítszám(gyűjtőházak[i]))
7; END FOR
8; RETURN kód
END FUNCTION

FUNCTION átalakítszám (szám)
1; szám←szám + 130;
2; a←szám DIV 2
3; b←szám DIV 2 + szám MOD 2
4; kód←összefűz(asciikód(a), asciikód(b))
5; RETURN kód
END FUNCTION
```

Ez az egyszerű algoritmus egy 32 karakter hosszú sztringet állít elő.

A tábla amibe beszúrjuk a kulcs-érték párokat sémája a következő



VII. fejezet

Mik a követelményeink?

Ha eddig eljutottunk, akkor álljunk meg, és fogalmazzunk pár követelményt a kifejlesztendő webes alkalmazással kapcsolatban. Készüljön Java nyelven. Csináljunk egy honlapot Servlet/JSP technológia felhasználásával, egy Java Appletet – ez lesz a játékunk – amit beillesztünk ebbe a honlapba. A játékunk egy adatbázistáblában tárolja a lépéseket. Ezt implementálhatjuk úgy, hogy csak egyetlen táblánk van, amit minden játékos olvashat és írhat. Másik megoldás az, hogy a táblák ideiglenesek, és csak egy session alatt állnak a felhasználó rendelkezésére. Alkalmazhatjuk azt is, hogy a honlapra először regisztrálni kell, ekkor minden felhasználó külön táblával rendelkezhet, aminek tartalmát akár törölheti is, ezzel a játék újrakezdi a tanulást. Én a legutóbbit választottam, mert így mindenki a saját szintjének megfelelően játszhat.

Így meg kell határoznunk, hogy a felhasználó milyen adataira vagyunk kíváncsiak. Legyenek ezek minimálisak, egy felhasználói név, egy e-mail cím és egy jelszó. A legutóbbit, ha akarja megváltoztathatja.

Fontos, hogy lekérdezhesse saját statisztikáját, hányszor győzött, vesztett, játszott döntetlent vagy törölte a játékállásokat tartalmazó táblájának tartalmát.

Készíthetünk számára egy összesítést is, hogy kik a legtöbb győzelemmel, vereséggel, döntetlennel rendelkező felhasználók, ill. kik törölték a legtöbbször a táblájukat.

Választanuk kell egy adatbázis-kezelő rendszert, és egy alkalmazásszervert is, legyenek ezek a MySQL 5.5 és a GlassFish v3 Prelude. A Java SE 6-os és a Java EE 6-os verzióját használom.

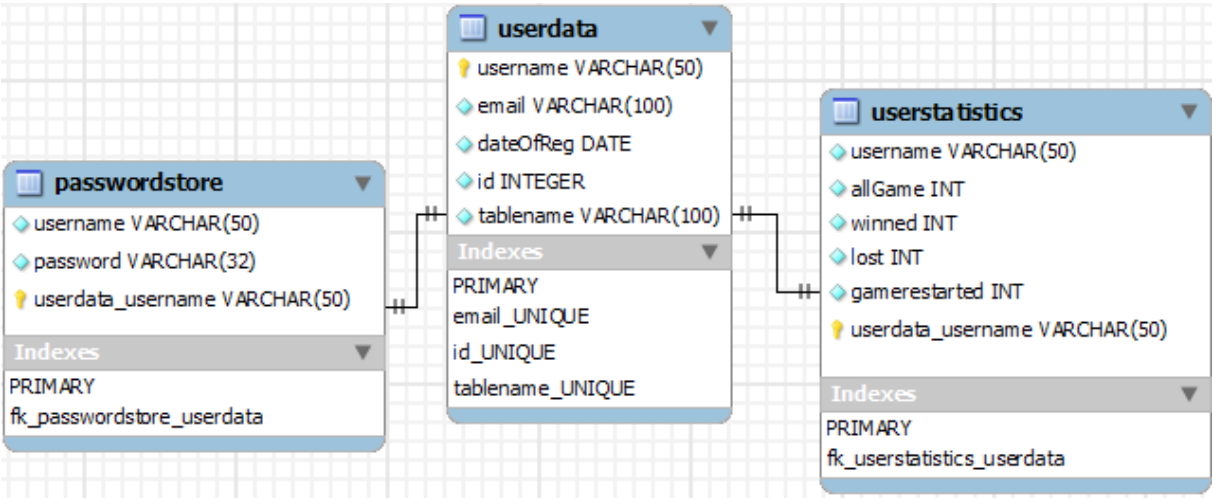
VIII. fejezet

Az adatbázis

A felhasználóknak regisztrálni kell, tehát szükségük lesz egy egyedi felhasználó névre és jelszóra. Ezeket tárolhatjuk egy táblában. Lesznek bizonyos adatok, amiket érdemes elkérni, ilyen a felhasználó e-mail címe, tárolhatjuk a regisztrációjának dátumát, egy felhasználói azonosító számot, és a hozzá tartozó tábla nevét.

Lehet egy olyan táblánk is, amiben összegyűjtöttük a felhasználók játékainak, győzelmeiknek, vereségeinek, és táblatörléseiknek számát, magyarul amiben tárolunk egy statisztikát róluk. Valamint minden felhasználónak ott lesz a játékállásokat tartalmazó táblája.

A sémánk a következő:



A Java és az adatbázis-kezelés

A JDBC API, a Java API része, ezzel tudunk hozzáférni a relációs adatbázisokban található adatokhoz. A JDBC-vel olyan alkalmazásokat tudunk írni, melyek el tudják végezni a következő tevékenységeket

1; Csatlakozni tudnak egy adatbázishoz

2; Lekérdezhetjük és frissíthetjük az adatbázist

3; Kinyerhetjük, és feldolgozhatjuk a lekérdezés során kapott adatokat.

A JDBC 4 részből áll:

1; A JDBC API

A JDBC 4.0 API része a Java platformnak, két csomagból áll, ezek a java.sql. és a javax.sql.. Mindkettő megtalálható a Java SE-ben és a Java EE-ben.

2; A JDBC Driver Manager (meghajtó)

A JDBC DriverManager osztály olyan objektumokat definiál, amelyek a Java alkalmazásokat csatlakoztatni tudják egy JDBC driverhez.

3; A JDBC Test Suit

Segítenek meghatározni, hogy JDBC driverek fognak futni a programban.

4; A JDBC-ODBC Bridge

A Java Szoftver híd JDBC elérést biztosít ODBC driveren keresztül.

Egy Java alkalmazás a következő mód tud az adatbázishoz csatlakozni JDBC segítségével:

Tegyük fel, hogy telepítettük a Java platform legfrissebb verzióját, a megfelelő JDBC drivereket, és rendelkezésünkre áll egy adatbázis, amihez csatlakozhatunk. A példákban a MySQL JDBC driver fog szerepelni.

1. lépés: Be kell töltenünk a drivert.

```
Class.forName("com.mysql.jdbc.Driver");
```

2 lépés: Csatlakoznunk kell az adatbázishoz

```
java.sql.Connection con = DriverManager.
getConnection("jdbc:mysql://domain/adatbazis", "username",
"password");
```

Ezután végezhetünk különböző műveleteket, például

Lekérdezéseket:

```
PreparedStatement pstmt = con.prepareStatement("SELECT * FROM
```

```
userdata ud WHERE ud.username = ?");
pstmt.setString(1, "Ancsika3");
ResultSet rs = pstmt.executeQuery();

while (rs.next()){
          System.out.println(rs.getString("username") + " " +
          rs.getDate("dateOfReg"));
}
```

Beszúrásokat:

```
PreparedStatement pstmt = con.
      prepareStatement("INSERT INTO user_game(code, heuristic) " +
      "VALUES(?, ?)");
pstmt.setString(1, "randomsztring");
pstmt.setInt(2, randomszám);
pstmt.execute();
```

Törléseket:

```
PreparedStatement pstmt =  con.prepareStatement("DELETE FROM
tablanev");
pstmt.execute();
```

Hibernate

A Hibernate egy objektum-relációs leképező könyvtár a Java nyelv számára, ami keretrendszert biztosít az OO-modell és a relációs adatbázis közötti leképezéshez.

A Hibernate legfőbb funkciója a Java osztályok leképzése adatbázis táblákra (és Java adattípusok leképezése SQL adattípusokra). A Hibernate egyúttal adat lekérdező és kinyerő lehetőségeket is nyújt.

A leképezés történhet egy XML-fájl vagy annotációk segítségével. Bármelyik módszert is választjuk, meg kell adnunk egy konfigurációs fájl (ez általában a hibernate.cfg.xml).

A Hibernate rendelkezik egy Hibernate Query Language (HQL) nevű SQL által inspirált nyelvvel, ami lehetővé teszi az SQL-szerű lekérdezések írását Hibernate objektumokra.

A programban a Hibernate-et és a JDBC-t is használom, előbbit, az userdata, passwordstore és userstatistic tábláknál, utóbbit a felhasználók játékállásokat tartalmazó tábláinál. A program a Hibernate 3.2.5-ös verzióját használja.

IX. fejezet

A login- és regisztrációs rendszer

Mielőtt a felhasználó bármihez is nyúlhatna regisztrálnia kell. Hozzunk ehhez létre mondjuk egy registration.jsp oldalt, ahol egy formban elkérjük a legszükségesebb adatait. Ezek:

1; A felhasználó neve

2; Az e-mail címe

3; A jelszava, és a megismételt jelszava

```
<form     name="registration"     action="<%=contextPath+"/Registration"%>"
method="POST">
        <input name="username" type="text" size="25" maxlength="50"/>
        <input name="email" type="text" size="25" maxlength="100"/>
        <input name="password" type="password" size="25" maxlength="50"/>
        <input name="passwordAgain" type="password" size="25"
        maxlength="50"/>
        <input name="ok" type="submit" value="Regisztrálok"/>
</form>
```

Miután a felhasználó megadta és nyugtázta a megadott adatokat, küldjük el ezeket a Registration.java nevű servletnek. A servlet megvizsgálja az adatokat, ha valami gondot talál, visszaküld minket a regisztrációs oldalra a megfelelő hibaüzenettel.

```
request.getSession().setAttribute("message", "hibaüzenet");
response.sendRedirect("registration.jsp");
```

A registration.jsp így vizsgálhatja meg, hogy történt-e hiba:

```
String message = (String) session.getAttribute("message");
if (message!=null){%>
        <p><%=message%></p>
<%}
```

Hiba lehet:

A felhasználói név már létezik vagy nem adták meg.

Az e-mail cím már létezik vagy nem adták meg.

A megadott e-mail cím, valójában nem lehet e-mail cím.

Az e-mail helyességét reguláris kifejezéssel tudjuk vizsgálni

```
private boolean checkEmail(String email){
     Pattern p = Pattern.compile("^[a-zA-Z0-9]+(\\.[_a-zA-Z0-
     9]+)*@" +
          "[a-zA-Z0-9]+(\\.[a-zA-Z0-9_]+)*\\.([a-z0-9A-Z]+)
          $");
     Matcher m = p.matcher(email);
     if (!m.matches()){
          return false;
     }
return true;
}
```

A két jelszó nem egyezik.

A jelszó nem elég hosszú.

Amennyiben minden rendben van, a felhasználó kap egy egyedei azonosítót, és egy táblanevet. A jelszavát egy kódolás segítségével át kell alakítunk, így ha esetleg illetéktelen személy hozzáférne a táblában tárolt jelszóhoz, nem tudja visszafejteni azt. Használhatjuk erre például az MD5-öt, amely minden adatból, függetlenül a méretétől és típusától egy 32 karakter hosszú hexadecimálist hasht hoz létre. Habár az MD5 nem visszafejthető, már több biztonsági rést is találtak rajta, és léteznek online elérhető adatbázisok amelyek tartalmazzák az MD5 hasht, és a hozzá tartozó eredeti szót. Biztonságosabbá tehetjük a használatát, ha kétszer kódoljuk le a jelszót, vagy kiegészítő karaktereket használunk, amit csak az alkalmazásunk ismer.

Ezután beszúrjuk az adatokat a megfelelő táblákba, a kezdőoldalra irányítjuk a játékost, és jelezzük neki, hogy már bejelentkezhet.

```
<%if (session.getAttribute("registered")!=null){%>
       <p>Most már bejelentkezhet!</p>
       <%session.removeAttribute("registered");
}%>
```

A bejelentkezés a regisztrációhoz hasonló.

A felhasználó egy formban megadja az adatait, a felhasználó nevét és jelszavát, amit szintén elküldjük a Login.java nevű servletnek.

```
<form name="login" action="<%=(contextPath +"/Login")%>" method="POST">
        <input name="username" type="text" size="25" maxlength="50"/>
        <input name="password" type="password" size="25" maxlength="50"/>
        <input name="ok" type="submit" value="Belépés"/>
</form>
```

Az végrehajtja a jelszavon a megfelelő átalakítást, majd megnézi, hogy létezik-e a felhasználó a megegyező jelszóval. Ha nem, jelzi a hibát, ha igen, beléptünk a rendszerbe, úgy, hogy a sessionbe tároljuk egy egy Userdata típusú objektumot ”loggedUser” néven. Ez fogja tartalmazni a felhasználónk adatait.

```
Passwordstore pw = (Passwordstore) hibernateSession.get
(Passwordstore.class,request.getParameter("username"));
if (pw!=null && pw.getPassword().equals(ConvertPassword.
       convertPassword(request.getParameter("password")))){
           request.getSession().setAttribute("loggedUser",
           pw.getUserdata());
}
```

Ezért minden lapunknál le kell kérni a sessionből loggedUser attribútumot, és ellenőrizni kell, hogy ez az érték null-e. Ha igen, akkor kiírjuk, hogy weplapot böngésző egyén, nem jogosult megtekinteni a tartalmat. Ha nem megjelenítjük az oldalt.

```
Userdata loggedUser = (Userdata) session.getAttribute("loggedUser");
if (loggedUser!=null){%>
       <!--oldal tartalma-->
<%}
else{%>
       <h1>Nincs jogosultságod az oldal megtekintésére!</h1>
<%}%>
```

X. fejezet

Egyéb funkciók

A felhasználó bármikor megnézheti a saját adatait. Ehhez hozzuk létre a mydatas.jsp-t. Lekérjük a loggedUser-t és frissítjük a statisztikáját. Ezek után csak kiíratjuk az adatait.

```
if (loggedUser!=null){
        loggedUser.setUserstatistic(RefreshUserstatistic.
        refreshUserstatistic(
        loggedUser.getUserstatistic()));
}
```

Ezen az oldalon érjük el a jelszóváltoztatás, illetve a játéktábla-törlés funkciókat.

Először a jelszóváltoztatást szeretném ismertetni.

A mydatas.jsp oldalon helyezzünk el egy formot, amiben csak egy submit típusú inputmező van.

```
<form action="changepassword.jsp">
        <input type="submit" value="Jelszó megváltoztatása"/>
</form>
```

Ez eljuttat minket a changepassword.jsp-re, ahol a következő form fogad:

```
<form action="<%=(contextPath +"/ChangePassword")%>" method="POST">
        <input type="password" name="oldPw" size="30"
        maxlength="50"/>
        <input type="password" name="newPw" size="30"
        maxlength="50"/>
        <input type="password" name="newPwAgain" size="30"
        maxlength="50"/>
        <input type="submit" name="ok"/>
</form>
```

Miután a felhasználó nyugtázta az adatokat a ChangePassword.java servlet feldolgozza ezeket. Amennyiben hibát talál, visszaküld a changepassword.jsp-re a megfelelő hibaüzenettel.

A hiba lehet:

1. Rosszul adtuk meg a régi jelszót.
2. Az új jelszó és az ismétlése nem egyezik.
3. Túl rövid az új jelszó.

Amennyiben mindent rendben talál, kicseréli a régi jelszót az újra az adatbázisban, és visszaküld a mydatas.jsp-re.

A játéktábla-törlés ennél is egyszerűbb.

Adott a mydatas.jsp-n egy form, egyetlen submit típusú inputmezővel:

```
<form action="<%=(contextPath + "/DeleteTable")%>" method="POST">
       <input value="Játékadatbázisom törlése" type="submit"/>
</form>
```

Erre rákattintva a DeleteTable.java servlet törli a felhasználó játéktáblájának tartalmát.

Másik funkció a legtöbb játékkal, győzelemmel, vereséggel és táblatörléssel rendelkező felhasználók megtekintése.

Hozzuk létre a statistic.jsp-t, és egy Statistic.java osztályt. A Statistic.java feladata lekérdezni a tíz elsőt a négy kategóriából, és az eredményeket négy List<Userstatistic> listában tárolni.

A statistic.jsp-ben csak annyit kell tennünk, hogy kiíratjuk ezeknek a listáknak a tartalmát.

XI. fejezet

Az applet

Applet vs JApplet

A játékunk GUI-ja egy Java Applet lesz. A Java Applet egy kicsi alkalmazás, programka, ami a webböngészőben fut a JVM segítségével vagy a Sun AppletViewer nevű programjában, amit appletek teszteléséhez fejlesztettek ki.

Az applet beágyazható HTML-oldalakba, amik paramétereket adhatnak át neki. Ezáltal ugyanannak az appletnek viselkedése megváltozhat, attól függően, hogy milyen paramétert adtak át neki.

Egy Java Applet a java.applet.Applet osztályból származik és ettől négy metódust örököl, amit a böngésző hív meg ezek az init(), a start(), stop() és destroy(). Kirajzoltatni a paint(Graphics g) metódussal tudunk, frissíteni az update(Graphics g)-vel vagy a repaint()-tel, ezeket a java.awt.Component osztálytól örökli. Egy appletbe java.awt.* osztályokat is fel tudunk használni.

Az appleteknek nagy előnye, hogy több platformra is könnyű elkészíteni, mert a legtöbb böngésző támogatja.

Én nem appletet, hanem JAppletet készítettem. A javax.swing.JApplet szintén a java.applet.Applet osztályból származik, azonban lehetővé teszi a swing eszközrendszer használatát. Az Applet és a JApplet nem kompatibilisek egymással, míg az applethez akármennyi gyermekelemet hozzáadhatunk, addig a JApplet csak eggyel rendelkezhet, ez a JRootPane típusú contentPane, tulajdonképpen a JApplet gyermekeinek szülője.

Tehát, az applethez az add(child) metódussal adunk gyermekelemet, a JApplethez a getContetPane().add(child) metódussal.

Mivel a swing komponenseket az esemény-irányító szál vezérli, de a böngészők által meghívott „mérföldkő” metódusok – az init(), a start(), a stop(), és a destroy(), nem ezen a szálon vannak. Ezért ezeknek a metódusoknak a SwingUtilities osztály invokeAndWait vagy, ha megfelelőbb az invokeLater metódusát kell meghívniuk.

A JApplet is örökli a paint(Graphics g) és update(Graphics g) és a repaint() metódusokat, ám, ha ezeket meghívjuk a felhasznált swing komponenseink „eltűnnek”. Erre az a megoldás, hogy létrehozunk egy JPanel-ből származtatott osztályt, és ennek írjuk felül a paint metódusát. Ezt az osztályt hozzá tudjuk adni a JApplet gyermekeihez.

Legyen az appletünk neve WoaleyJApplet, helyezzük el a gamegraphics nevű csomagban.

```
public class WoaleyJApplet extends JApplet implements Runnable{
        // TODO az osztály összes adattagja és metódusa
}
```

Mire lesz szükségünk?

Hozzunk létre olyan osztályokat, amik a JPanel osztályból származnak. Nincs más dolgunk, mint ezeket cserélgetni. Szükségünk van egy olyan panelre, ami akkor jelenik meg, ha nem vagyunk jogosultak játszani (NoAuthorityPanel). Egy olyanra, amin kiválaszthatjuk az opciókat, a számítógép kezd vagy az ember, és a nehézségi szintet, elolvashatjuk a szabályokat, és elindíthatjuk a játékot (OptionPanel). A lényeg az a panel lesz, amin játszhatunk. Itt jelenik meg a tábla, figurák, hova katttinthatunk, hány figura van egy gödörben, melyik játékos következik lépni stb. (WoaleyPanel). Miután vége a játéknak egy újabb panelre kiírjuk, hogy ki a győztes, és a pontszámokat (EndOfGamePanel). Szükségünk lesz még egy panelre, amire az esetlegesen fellépő hibákról is tájékoztathatjuk a felhasználót (SomethingFailedPanel).

Készítsünk még el két segédosztályt, az Option.java-t és a WinnerStatistics.java-t, előbbi természetesen a felhasználó által választott opciókat írja le, utóbbi azt, hogy ki győzött vagy döntetlennel zárult-e a játék, és az egyes játékosok hány figurát szereztek.

Még egy osztályra szükségünk lesz, ez a Data.java, ebben tartjuk nyilván, hogy ki a játékosunk, melyik tábla tartozik hozzá, és az aktuális játék végén milyen eredményt ért el.

Az appletben képeket is kell majd megjelenítenünk, amiket be kell olvasnunk. Kell egy osztály, ami beolvassa képeket, legyen ez az ImageRead.

Három képet kell majd feldolgoznunk, egy játéktáblát, egy zöld-lámpát, ami jelzi, hogy az adott házból indulhat vetés, és egy pirosat, ami azt jelzi, hogy nem.

A játéktábla képen elhelyezkednek a házak és a gyűjtőházak. Ezek kör alakúak. Legyen

egy House osztályunk, ami a Canvas osztályból származik, és ebből származzon CollectorHouse nevű osztály is. Adjuk át ezeknek a képen található házak koordinátáit. Hozzunk létre egy Figure osztályt, ami szintén a Canvas osztályból származik, ez fogja kirajzolni a figurákat, a megadott koordinátákra. A House osztály rendelkezik egy figuralistával és egy koordinátalistával, ez szabja meg hogy melyik házba hány figurát rajzolhatunk ki, és ezeket melyik koordinátára.

WoaleyJApplet

Az applet egy paraméterben kapja meg, hogy ki akar vele játszani. Ekkor a biztonság kedvérért ellenőrzi, hogy ez a felhasználó egyáltalán létezik-e. Ezt úgy teszi, hogy a felhasználó nevét elküldi a „Game” nevű servletnek, ami visszaküldi a táblájának nevét, illetve, ha nem létezik null-t.

Az applet ellenőrzi, hogy a servlettől null-t kapott-e. Ha igen, bekövetkezik egy NoAuthorityException. Amikor ezt a kivételt elkapjuk, megjelenik egy NoAuthorityPanel típusú objektum, ami tájékoztatja az imposztort, hogy nem játszhat. Ha nem null-t kap, tehát létezik a felhasználó, és rendelkezik egy játékállásokat tartalmazó táblával, akkor meghívjuk az alábbi metódust:

```
javax.swing.SwingUtilities.invokeAndWait(new Runnable() {
        public void run() {
                createGUI();
        }
});
```

Ez elindítja a WoaleyJApplet szálat is.

Ez a szál fog felelni a panelek cseréléséért. Legelőször egy OptionPanel jelenik meg. Ezen a kezdőjátékos és a nehézségi fok kiválasztása egy-egy ButtonGroup-ba foglalt JRadioButtonok segítségével történik. A játék akkor kezdődik el, amikor a felhasználó a „Start” feliratú JButtonra kattint. Ekkor jelenik meg a következő panel egy WoaleyPanel. Ezt tájékoztatnunk kell, a játékos választásairól, úgy, hogy egy Option típusú objektumot paraméterként átadunk neki.

Amikor elértük vagy száz lépés után sem sikerült elérnünk a végállapotot, befejezzük a játékot, kiértékeljük a játszmát. Ezután a WoaleyJPanel elküldi a Game.java servletnek, hogy ki győzött, ez pedig frissíti az adatbázist. Ezután megjelenik az EndOfGamePanel, ami tájékoztatja a játékost arról, hogy vége a játéknak, valamint ki írja, hogy ki győzött vagy esetleg döntetlen született-e, és hány pontot szerzett a játékos, illetve a számítógép. Ezt leokézva újra egy OptionPanel jelenik meg.

Az applet aláírása

Az appletek felvetnek bizonyos biztonsági problémákat. A hálózaton keresztül letöltött appleteket általában nem megbízható Java programoknak tekintik, tehát a fejlesztők korlátozták a mozgásterüket. Ezért a fejlesztés során ezekkel számolni kell, bár egyes böngészők lehetővé teszik a korlátok enyhítését. A JDK 1.1 lehetővé teszi a megbízható forrásból származó appletek digitális aláírását. Az aláírt appletekre a korlátozások nem vonatkoznak.

Pár nem megbízható appletre vonatkozó korlátozás:

- Nem férhetnek hozzá az őket letöltő számítógép fájlrendszerében tárolt információkhoz.
- Csak azzal a számtógéppel kommunikálhatnak, amiről letöltöttük őket.
- Nem használhatják az awt csomag egyes részeit, illetve nem hozhatnak létre új ablakokat.

Attól, hogy még az applet aláírt, lehet ártalmas!

Összefoglalás

Nincs a Földön olyan ember, aki soha életében ne játszott volna. Játszunk, mert élvezzük, unatkozunk, nincs mást csinálnunk. Amíg mi uralkodunk a játék felett, és nem ő rajtunk, addig semmi gond sincs, azonban egyre több függő van a világon. Gondoljunk csak a különböző kártyajátékokra, mint amilyen a póker, táblajátékokra, mint az amőba (igen, vannak amőbafüggők!) vagy a különböző számítógépes és webes játékokra.

Nem tudom, fogok-e még kétszemélyes játékot írni, de ha igen, valószínűleg teljesen máshogy csinálom majd. Ha úgy döntenék, hogy újraírom ezt a játékot, biztos, hogy másképp csinálnám. Azonban webre még szeretnék fejleszteni. Még nem tudom mit, azt hogy hogyan végképp nem, de az biztos, hogy ez a szakdolgozat hasznos tapasztalat volt hozzá. Átéltem az ember által érezhető érzelmek összes skáláját, haragot, gyűlöletet, idegességet, ijedtséget, örömet, szeretet... Talán ezek a programozás ártalmai.

Azt az embert, aki eddig elolvasta ezt a szakdolgozatot most arra kérném, hogy, ha szép az idő, akkor tegyen egy sétát. Jó dolog az olvasás, és a játék is, de nem ülhetünk egész nap egy szobában a gép előtt görcsölve.

Irodalomjegyzék

Internetes források

I. fejezet – Bevezetés

[1] http://terebess.hu/keletkultinfo/mankala2.html

II. fejezet – A játék alapjai és pár változata

[2] http://www.gamerz.net/pbmserv/wari.html

[3] http://terebess.hu/keletkultinfo/mankala2.html

[4] http://en.wikipedia.org/wiki/Wari_(game)

[5] http://www.jatek.hu/szabalyok/hlp_mankala.html

[6] http://en.wikipedia.org/wiki/Kalah

[7] http://gamecabinet.com/rules/Bao.html

[8] http://en.wikipedia.org/wiki/Omweso

[9] http://www.kibao.org/

[10] http://www.wikimanqala.org/wiki/Bao

[11] http://www.tradgames.org.uk/games/Mancala.htm

III. fejezet – A reprezentáció

[12] Dr. Várterész Magda: Mesterséges Intelligencia 1

IV. fejezet – A keresőalgoritmus

[13] Dr. Várterész Magda: Mesterséges Intelligencia 1

[14] http://en.wikipedia.org/wiki/Negamax

[15] http://en.wikipedia.org/wiki/Alpha-beta_pruning

[16] http://hu.wikipedia.org/wiki/Alfa-béta_vágás

V. fejezet – A heurisztika

[17] Dr. Várterész Magda: Mesterséges Intelligencia 1

VI. fejezet – A játék tanul

A kép a MySQL Workbench 5.2 nevű szoftverrel készült.

VII. fejezet – Mik a követelményeink?

-

VIII. fejezet – Az adatbázis

A kép a MySQL Workbench 5.2 nevű szoftverrel készült.

[18] http://java.sun.com/docs/books/tutorial/jdbc/index.html

[19] http://en.wikipedia.org/wiki/Hibernate_(Java)

[21] http://www.hibernate.org/

[22] http://docs.jboss.org/hibernate/stable/core/reference/en/html/tutorial.html

[23] http://docs.jboss.org/hibernate/stable/annotations/reference/en/html/entity.html

IX. fejezet – A login- és regisztrációs rendszer

[24] http://java.sun.com/docs/books/tutorial/essential/regex/index.html

[25] http://hu.wikipedia.org/wiki/MD5

[26] http://weblabor.hu/blog/20050822/md5tamadasok

X. fejezet – Egyéb funkciók

-

XI. fejezet – Az applet

[27] http://en.wikipedia.org/wiki/Applet

[28] http://en.wikipedia.org/wiki/Java_applet

[29] http://en.wikipedia.org/wiki/Swing_(Java)

[30] http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JApplet.html

[31] http://java.sun.com/docs/books/tutorial/deployment/applet/index.html

[32] http://java.sun.com/docs/books/tutorial/uiswing/components/applet.html

[33] http://fi.inf.elte.hu/~csb/b5/node29.html

Függelék

I. fejezet – Bevezetés

-

II. fejezet – A játék alapjai és pár változata

A Wari a következő linken próbálható ki:

http://tablajatekos.hu/zz2007/mancala_vari.html

A Kalah a következő linkeken próbálható ki:

http://www.fupa.com/play/BoardGame-free-games/ancient-kalah.html

http://tablajatekos.hu/z2005/cmancala.html

http://tablajatekos.hu/uj2001/0mancala/mancala.html

http://tablajatekos.hu/z2005/bantumi.html

Kalah csigákkal :)

http://tablajatekos.hu/uj2001/_1mancala.html

A Bao nevű játék a következő linken próbálható ki:

http://www.fdg.unimaas.nl/educ/donkers/games/Bao/

http://www.baogame.com/

III. fejezet – A reprezentáció

Olvasnivaló több játékról:

http://en.wikipedia.org/wiki/Game_complexity

IV. fejezet – A keresőalgoritmus

http://en.wikipedia.org/wiki/Artificial_intelligence

http://en.wikipedia.org/wiki/Minimax

V. fejezet – A heurisztika

http://en.wikipedia.org/wiki/Heuristics

VI. fejezet – A játék tanul

-

VII. fejezet – Mik a követelményeink?

-

VIII. fejezet – Az adatbázis

A Hibernate-hez használt konfigurációs XML-dokumentum:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE hibernate-configuration (View Source for full doctype...)>
<hibernate-configuration>
<session-factory>
	<property name="hibernate.dialect">
		org.hibernate.dialect.MySQLDialect
	</property>
	<property name="hibernate.connection.driver_class">
		com.mysql.jdbc.Driver
	</property>
	<property name="hibernate.connection.url">
		jdbc:mysql://domain:port/adatbázis
	</property>
	<property name="hibernate.connection.username">
		username
	</property>
	<property name="hibernate.connection.password">
		password
	</property>
</session-factory>
</hibernate-configuration>
```

A Hibernate-hez használt osztályok.

```
@Entity
@Table(name="userdata")
public class Userdata implements Serializable{@Id
    @Column(name="username", length=50, nullable=false, unique=false)
    private String username;
    @Column(name="email", length=100, nullable=false)
    private String email;
    @Column(name="dateOfReg")
    private Date dateOFReg;
    @Column(name="id", nullable=false)
    private Integer id;
    @Column(name="tablename", nullable=false, length=100)
    private String tablename;
    @OneToOne(mappedBy="userdata")
    @JoinColumn(name="username")
    private Userstatistic userstatistic;
    @OneToOne(mappedBy="userdata")
    @JoinColumn(name="username")
    private Passwordstore password;


    public Userdata() {

    }

    public Userdata(String username, String password, String email, Integer
id, Date                dateOFReg, String tablename) {
        this.username = username;
        this.email = email;
        this.dateOFReg = dateOFReg;
        this.id = id;
        this.tablename = tablename;
```

```
        this.password = new Passwordstore(this, password);
        this.userstatistic = new Userstatistic(this, 0, 0, 0, 0);
    }

//TODO getters and setters

}
```

```
@Entity
@Table(name="passwordstore")
public class Passwordstore implements Serializable {
    @Column(name="username", length=50, nullable=false)
    @Id
    private String username;
    @Column(name="password", length=50, nullable=false)
    private String password;
    @OneToOne
    @JoinColumn(name="username")
    private Userdata userdata;

    public Passwordstore() {
    }



    public Passwordstore(Userdata user, String password) {
        this.userdata = user;
        this.username = user.getUsername();
        this.password = password;
    }

    //TODO getters and setters


}
```

```
@Entity
@Table(name= "userstatistic")
public class Userstatistic implements Serializable{
    @OneToOne
    @JoinColumn(name="username")
    private Userdata userdata;
    @Id
    @Column(name="username", nullable=false, length=50)
    private String username;
    @Column(name="allgame", nullable=false)
    private Integer allGame;
    @Column(name="winned", nullable=false)
    private Integer winned;
    @Column(name="lost", nullable=false)
    private Integer lost;
    @Column(name="gamerestarted", nullable=false)
    private Integer gamerestarted;


    public Userstatistic(Userdata user, Integer allGame, Integer winned,
```

```
Integer lost, Integer         gamerestarted) {
        this.userdata = user;
        this.username = user.getUsername();
        this.allGame = allGame;
        this.winned = winned;
        this.lost = lost;
        this.gamerestarted = gamerestarted;
    }

    public Userstatistic() {
    }

    //TODO getters and setters
}
```

A használathoz elengedhetetlen HibernateUtil osztály

```
import org.hibernate.*;
import org.hibernate.cfg.*;

public class HibernateUtil {

    private static final SessionFactory sessionFactory;

    static {
      try {
       System.out.println(hibernate.Userdata.class.getPackage().getName());
            sessionFactory = new
            AnnotationConfiguration().addPackage("hibernate").
                    addAnnotatedClass(hibernate.Userdata.class).
                    addAnnotatedClass(hibernate.Userstatistic.class).
                    addAnnotatedClass(hibernate.Passwordstore.class).
                    configure().buildSessionFactory();
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }

}
```

IX. fejezet – A login- és regisztrációs rendszer

MD5 hasheket tartalmazó adatbázisok

http://md5.my-addr.com/

http://md5.igrkio.info/md5-hash-database.html

X. fejezet – Egyéb funkciók

```
public class RefreshUserstatistic {
    public static Userstatistic refreshUserstatistic(Userstatistic
```

```
userstatistic){
        Session session = null;
        try{
            session = HibernateUtil.getSessionFactory().openSession();
            session.beginTransaction();

            Query query = session.createQuery("FROM Userstatistic us WHERE
      us.username=:username");
            query.setString("username", userstatistic.getUsername());

            return (Userstatistic) query.uniqueResult();
        }finally{
            if (session!=null)
                session.close();
        }
    }

}


public class ChangePassword extends HttpServlet {

    /**
     * Processes requests for both HTTP <code>GET</code> and
<code>POST</code>       methods.
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
protected void processRequest(HttpServletRequest request,
HttpServletResponse     response) throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        try {
            Userdata ud = (Userdata)
request.getSession().getAttribute("loggedUser");
            if (!
ud.getPassword().getPassword().equals(ConvertPassword.convertPassword(
            request.getParameter("oldPw")))){
                request.getSession().setAttribute("message", "Nem ez volt a
\"régi\" jelszavad!");
            }
            else if (!
request.getParameter("newPw").equals(request.getParameter(
            "newPwAgain"))){
                request.getSession().setAttribute("message", "A jelszavak
nem egyeznek!");
            }
            else if (request.getParameter("newPw").length()<6){
                request.getSession().setAttribute("message", "A jelszó nem
elég hosszú!");
            }
            else{
```

```
                ud.getPassword().setPassword(ConvertPassword.convertPasswor
d(request.
            getParameter("newPw")));
                Session hsession = (Session)
HibernateUtil.getSessionFactory().openSession();
                hsession.beginTransaction();
                hsession.update(ud.getPassword());
                hsession.getTransaction().commit();
                hsession.close();
                request.getSession().setAttribute("message", "Sikeres
jelszóváltoztatás!");
            }
            response.sendRedirect("changepassword.jsp");
        } catch(Exception ex){

        } finally {
            out.close();
        }
    }

    //TODO other methods

}


public class DeleteTable extends HttpServlet {

    /**
     * Processes requests for both HTTP <code>GET</code> and
<code>POST</code>       methods.
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            Userdata ud = (Userdata)
request.getSession().getAttribute("loggedUser");
            DeletePlayerTable.deletePlayerTableContent(ud.getTablename());
            ud.getUserstatistic().setGamerestarted(ud.getUserstatistic().ge
tGamerestarted()+1);
            Session hsession = (Session)
HibernateUtil.getSessionFactory().openSession();
            hsession.beginTransaction();
            hsession.update(ud.getUserstatistic());
            hsession.getTransaction().commit();
            hsession.close();
            request.getSession().setAttribute("deleted", "Sikeres
törlés!");
            response.sendRedirect("mydatas.jsp");
        } catch(Exception ex){
        } finally {
```

```
            out.close();
        }
    }
//TODO other methods
}


public class Statistic {
    private List<Userstatistic> mostGameWinned;
    private List<Userstatistic> mostGameLost;
    private List<Userstatistic> mostGamePlayed;
    private List<Userstatistic> mostGameRestarted;

    public Statistic() {
        createStatistic();
    }

    private void createStatistic(){
        Session session = HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();

        Query query = session.createQuery("FROM Userstatistic us " +
                    "ORDER BY us.winned DESC");
        query.setMaxResults(10);
        mostGameWinned = query.list();

        query = session.createQuery("FROM Userstatistic us " +
                    "ORDER BY us.lost DESC");
        query.setMaxResults(10);
        mostGameLost = query.list();

        query = session.createQuery("FROM Userstatistic us " +
                    "ORDER BY us.allGame DESC");
        query.setMaxResults(10);
        mostGamePlayed = query.list();

        query = session.createQuery("FROM Userstatistic us " +
                    "ORDER BY us.gamerestarted DESC");
        query.setMaxResults(10);
        mostGameRestarted = query.list();

        session.getTransaction().commit();
        session.close();
    }
//TODO getters, setters
}
```

XI. fejezet – Az applet

Az applet a game.jsp-be ágyazzuk bele.

```
<applet code="gamegraphics.WoaleyJApplet" archive="game.jar" width="800"
```

```
height="500">
      <param name="username" value="<%=loggedUser.getUsername()%>"/>
</applet>
```

Az applet és a servlet közötti kommunikáció az applet részéről.

```
public URLConnection createConnection() throws MalformedURLException,
IOException{
      URL urlServlet = new URL(getCodeBase(), "Game");
      URLConnection servletConnection = urlServlet.openConnection();
      servletConnection.setDoInput(true);
      servletConnection.setDoOutput(true);
      servletConnection.setUseCaches(false);
      servletConnection.setRequestProperty("Content-Type","application/x-
java-serialized-        object");
      return servletConnection;
}

private void pickUser() throws NoAuthorityException,
IOException,ClassNotFoundException{

      String username = getParameter("username");
      URLConnection servletConnection = createConnection();
      ObjectOutputStream out = new
            ObjectOutputStream(servletConnection.getOutputStream());
      out.writeObject(username);
      out.flush();
      out.close();

      ObjectInputStream in = new
ObjectInputStream(servletConnection.getInputStream());
      String tablename = (String) in.readObject();
      in.close();

        if (tablename==null){
            throw new NoAuthorityException();
        }

        data = new Data(username, tablename);
}


private void updateUserstatistic() throws IOException,
ClassNotFoundException{
        URLConnection servletConnection = createConnection();

        if (wp.getWinnerStatistic().isWinner()){
            data.setGame(Data.WINNED);
        }
        else if (!wp.getWinnerStatistic().isWinner()){
            data.setGame(Data.LOST);
        }
        else{
            data.setGame(Data.DRAWN);
```

```
        }

      ObjectOutputStream out = new
      ObjectOutputStream(servletConnection.getOutputStream());
      out.writeObject(data);
      out.flush();
      out.close();

      ObjectInputStream in = new
      ObjectInputStream(servletConnection.getInputStream());
      String message = (String) in.readObject();
      in.close();
}
```

Az applet és a servlet közötti kommunikáció a servlet részéről

```
protected void processRequest(HttpServletRequest request,
HttpServletResponse response){
        ObjectInputStream in = null;
        try {
            response.setContentType("application/x-java-serialized-
object");
            in = new ObjectInputStream(request.getInputStream());
            ObjectOutputStream out = new
ObjectOutputStream(response.getOutputStream());
            Object obj = in.readObject();

            if (obj instanceof String){
                out.writeObject(isUserLoggedIn((String) obj));
            }
            else if (obj instanceof Data){
                out.writeObject(refreshDatabase((Data) obj));
            }

            out.flush();
            out.close();
        } catch (ClassNotFoundException ex) {

        } catch (IOException ex) {

        } catch (Exception ex){

        } finally {
            try {
                in.close();
            } catch (IOException ex) {

            }
        }
 }

private String isUserLoggedIn(String username){
         Session hibernateSession =
HibernateUtil.getSessionFactory().openSession();
```

```
         hibernateSession.beginTransaction();
         Userdata ud = (Userdata) hibernateSession.get(Userdata.class,
username);

         if (ud==null){
             hibernateSession.close();
             return null;
         }
         hibernateSession.close();
         return ud.getTablename();
    }

    private String refreshDatabase(Data data) throws IOException{
        Session hibernateSession =
HibernateUtil.getSessionFactory().openSession();
        Transaction tr = hibernateSession.beginTransaction();

        Userdata ud = (Userdata) hibernateSession.get(Userdata.class,
data.getUsername());

         if (ud==null){
             hibernateSession.close();
             return null;
         }

        Userstatistic us = ud.getUserstatistic();

        us.setAllGame(us.getAllGame()+1);
        if (data.getGame()==Data.WINNED){
            us.setWinned(us.getWinned()+1);
        }
        else if (data.getGame()==Data.LOST){
            us.setLost(us.getLost()+1);
        }
        hibernateSession.update(us);
        tr.commit();
        hibernateSession.close();

        return "ok";
}
```

A WoaleyJApplet osztály run metódusa:

```
public void run() {
         while(Thread.currentThread()==game){
             try {
                 while (op.getOption() == null) {

                 }
                 initWoaleyPanel();
                 wp.startNewGame(op.getOption());
                 removePreviousAndAddNextPanel(op, wp);
                 getContentPane().add(wp);
                 while (wp.getWinnerStatistic() == null) {
```

```
                }
                updateUserstatistic();
                eog.setWs(wp.getWinnerStatistic());
                removePreviousAndAddNextPanel(wp, eog);
                while (!eog.isOkclicked()) {

                }
                removePreviousAndAddNextPanel(eog, op);
                op.setOption(null);
                eog.setOkclicked(false);
            } catch (Exception ex) {
                getContentPane().removeAll();
                getContentPane().add(new
                SomethingFailedPanel(ex.getMessage()));
            }
        }
}
```

Köszönetnyilvánítás

Ezt a szakdolgozatot elsősorban szeretném megköszönni témavezetőmnek, Kósa Márknak.

Hálával tartozom mindazon egyéneknek, akik végighallgatták sirámaim, panaszaim, biztattak, támogattak, és képekkel, csokival, dekorkövekkel szolgálták lelkem épülését.