Debreceni Egyetem

Informatika Kar

Interaktív webes felület fejlesztése logikai játékokhoz

Témavezető:
Dr. Halász Gábor
egyetemi docens

Készítette:
Ujjobbágy Attila
programtervező matematikus

Debrecen
2009

1 Tartalomjegyzék

2 Bevezetés

Diplomamunkám célja egy olyan interaktív, webes felület kialakítása, ahol egyrészt a játszani szeretők találhatnak maguknak szórakozási lehetőséget, játékok egy gyűjteményét, valamint ami a fejlesztők számára egy olyan univerzális keretrendszert biztosít, aminek felhasználásával megalkothatják saját játék implementációikat.

Az elkészített alkalmazás természetesen tartalmaz majd a játékokból egy-egy megvalósítást Java applet-ként elkészítve, de ezen túl minden játékhoz nyújt egy web szolgáltatások által létrehozott interfészt, amit felhasználva el lehet készíteni a játék klienseket bármilyen platformra, ami támogatja a webszolgáltatások hívását. Ilyen módon lehetőség nyílik akár mobiltelefonra optimalizált, vagy olyan nyelvek használatával készült kliensek írására, mint a C, a különböző .NET nyelvek, vagy akár a PHP.

Manapság egy webes felület kialakításánál alapvető követelmény, hogy az szép legyen, mégis egyszerű felépítésű, gyorsan reagáló, és viszonylag kevés adatforgalmat használjon, ne töltődjenek újra minden lépésnél az oldalak. Ennek megvalósítására ma már rendelkezésre állnak olyan ingyenesen felhasználható technológiák, mint a Google Web Toolkit (GWT), ami az AJAX (Asynchronous JavaScript and XML) technikára épülve biztosít számunkra lehetőséget funkciók háttérben (a böngészőben a lap frissítése nélkül) történő hívására, valamint a legkülönfélébb előre lekódolt webes komponensek használatára. A GWT-től bővebben későbbi fejezetekben még szó lesz, konkrétan kifejtve.

Az alkalmazást a ma nagyon divatos, és logikusan átgondolt háromrétegű architektúrára alapozva készítem el. A háromrétegű architektúra gyakorlatilag három, egyenként önálló funkcionalitással bíró egységből áll, amelyek cserélhetők, a másik két rész módosítása nélkül fejleszthetők. Az első réteg az adatbázis, minden tárolandó adat, ami a rendszerben keletkezik itt található. A második réteg az alkalmazás maga, itt tárolódik az üzleti logika, valamint az adattárolással kapcsolatos folyamatokat leszámítva (hiszen azt az első réteg végzi) minden nagyobb erőforrást igénylő folyamat itt zajlik. A harmadik réteg a felhasználói felület, itt jön a képbe a GWT, ugyanis e réteg megvalósítása azzal lesz megoldva. A rétegekből egyenként akár több is lehet, lehet több féle adatbázis, vagy egy adatbázisra több féle alkalmazás szerver támaszkodhat, a lényeg, hogy a feladatok jól el vannak különítve, a felhasználó csak az üzleti

logikán keresztül képes elérni az adatbázist. Az én esetemben az adatbázis elérést a JBoss Java alkalmazásszerver biztosítja, az adatok írása/olvasása pedig a Java nyelv által adott standard eszközökkel lesz megvalósítva. Az adatok tárolása Oracle10g adatbázisban történik. Az üzleti réteg, az alkalmazáslogika Enterprise JavaBean-ek segítségével lesz megoldva, a harmadik, a prezentációs réteg pedig a már említett Google Web Toolkit alkalmazásával.

Az üzleti réteg funkcióit használó webszolgáltatások a legfelső, webes szinten vannak elhelyezve. A működéséhez szükséges leírók, segéd állományok, a hívások paramétereit illetve visszatérési értékeit becsomagoló Java osztályok mind automatikusan, a méltán népszerű Apache Axis eszköz segítségével lesznek kigenerálva, egy későbbi fejezetben ezt bővebben kifejtem.

Egy alkalmazás esetében nagyon fontos szempont a naplózás. Nagy szerepe van egyrészt a biztonság tekintetében, másrészt hibakeresés során, vagy egyszerűen, ha a futás során különféle információkat szeretnénk később visszakereshető módon tárolni. Megfelelő naplóbejegyzések segítségével akár statisztikákat is lehet készíteni az egyes játék kliensek teljesítményéről. Erre a feladatra hivatott a JBoss (és még rajta kívül szinte mindegyik) alkalmazásszerver által támogatott Log4J naplózó rendszer, mely felhasználásra kerül az alkalmazásban.

3 A fejlesztés során alkalmazott eszközök

Ebben a fejezetben a bevezetőben már említett eszközökről adok egy rövid, áttekintő leírást.

3.1 Java

A Java egy manapság igen divatos, és viszonylag könnyen megtanulható objektumorientált programozási nyelv, melyet a Sun Microsystems fejleszt a 90-es évek elejétől. Legfontosabb tulajdonsága a platformfüggetlenség, amiből kifolyólag egy Java alkalmazás bármilyen operációs rendszer alatt mindenféle változtatás nélkül képes ugyanolyan módon futni. Ez az úgynevezett JVM (Java Virtual Machine – Java virtuális gép) segítségével van megoldva. A Java fordító egy köztes kódot (byte kódot) fordít a forrásállományokból, melyet aztán a JVM futtat.

Többek között e tulajdonságok miatt választottam az alkalmazások leprogramozásához ezt a nyelvet, valamint azért, mert fejlett támogatást nyújt az összes olyan feladat megoldásához, amire szükség lesz. Értendő ezalatt a hatékony hálózatkezelés, a többszálúság támogatása, valamint az alapvető GUI komponensek egyszerű kezelése, aminek a játékok felhasználói felületének programozásakor nagy szerepe lesz.

3.2 Java Applet

Az applet-ek olyan kisalkalmazások, melyek a böngészőbe beépülő Java Plugin segítségével, weboldalba ágyazva futnak. A Java Applet fejlesztése során élvezhetjük a Java nyelv által nyújtott előnyöket. A programkódot a böngésző letölti a felhasználó számítógépére, majd ott futtatja. Emiatt persze az applet-nek sok biztonsági kritériumnak kell megfelelnie, illetve nem végezhet akármilyen műveletet a kliens gépen (nem nyithat hálózati kapcsolatot akárhová, nem használhatja a felhasználó állományait). Az alapértelmezésként tiltott műveleteket engedélyezni lehet, ha úgynevezett aláírt applet-eket használunk, és megbízunk a kisalkalmazás tanúsítványának kibocsátójában.

A megvalósított játékoknak a weboldalon játszható verziója Java applet-ként lesz implementálva.

3.3 Eclipse Framework

Az Eclipse egy nyílt forráskódú, Java nyelven íródott, ezáltal platformfüggetlen integrált fejlesztőkörnyezet (Integrated Development Environment, IDE), mely az elérhető pluginok segítségével sokféle (nem csak Java nyelvű) alkalmazás elkészítésére használható.

Az általam használt verzió sokoldalú támogatást nyújt a JBoss alkalmazásszerverhez, valamint J2EE (azaz üzleti Java) alkalmazások fejlesztéséhez.

3.4 Enterprise JavaBeans (EJB)

Az EJB egy olyan szerver oldali komponens, mely önállóan telepíthető egy elosztott, többrétegű környezetben. Mivel a szolgáltatásai az EJB specifikációban meghatározottak, bármilyen, a specifikációnak megfelelő konténerben használhatóak forráskód módosítás, és újrafordítás nélkül. A Java osztályokon kívül leíró állományok is tartoznak hozzá, amikben meghatározhatjuk az általános szerveroldali szolgáltatásokkal, például a tranzakció-kezeléssel és a biztonsággal kapcsolatos beállításokat.

Az EJB legnagyobb előnye, hogy a fejlesztőnek nem kell mélyen foglalkoznia az általános alkalmazásszerverbeli szolgáltatásokkal, mint például a tranzakció kezelés, perzisztens állapot kezelés, erőforrás gazdálkodás, hanem elegendő az üzleti logikára koncentrálnia.

Háromféle EJB létezik, ezek az

- entitás bean-ek

  Segítségükkel perzisztens objektumokat kezelhetünk, modellezhetjük az üzleti folyamatok során használt üzleti adatokat. Használata ma már körülményes, helyette szívesebben alkalmazzák a különböző ORM (object-relational mapping) eszközöket, mint a szabad forrású Hibernate.

- üzenetvezérelt bean-ek

  Aszinkron módon hajt végre tevékenységet, üzenetet lehet neki küldeni, és erre reagál.

- session bean-ek

  Elsősorban üzleti tevékenységeket, folyamatokat megvalósító újrafelhasználható komponens. Viszonylag rövid életű, amikor egy kliense meghívja egy szolgáltatását a

konténer példányosít belőle egyet, vagy az előzőleg már gyorsító tárazott (cache, vagy pool) példányt használja, végrehajtja a kért műveletet, majd meg is szüntetheti. Az alkalmazás üzleti rétege állapotmentes session bean-ekben lesz megvalósítva. Az állapotmentesség azt jelenti, hogy az egyes metódushívások között nem őrződik meg a komponens állapota.

A session bean-ek szolgáltatásait kétféle interfészen keresztül lehet elérni.

- Az úgynevezett távoli interfészen keresztüli hívás során a kliens oldali csonk (stub) végzi el a paraméterek platformfüggetlen formára alakítását (marshaling), felépíti a hálózati kapcsolatot a szerver oldali vázzal (skeleton), aztán ezen keresztül továbbítja az IIOP protokoll szerinti hívást. Ezután a szerver oldali skeleton visszaalakítja a paramétereket (demarshaling), és továbbítja a hívást az EJB objektumnak.
- A lokális, vagy helyi interfész használata akkor javasolt, a távoli interfész helyett, ha az EJB és a kliense (ami akár egy másik EJB is lehet), egyazon környezetben, egy JVM-ben futnak. Ilyenkor ugyanis elkerülhető a felesleges marshaling/demarshaling folyamat, és a hálózati kapcsolat kialakítása.

A fejlesztés során az interfészeket és a leírókat az XDoclet eszköz segítségével, annotációk használatával fogjuk kigenerálni.

3.5 JBoss J2EE Application Server

A JBoss egy nyílt forráskódú, szabadon felhasználható, J2EE-n alapuló alkalmazásszerver, amit jelenleg a Red Hat fejleszt. Mivel teljes mértékben Java-ban íródott, a JBoss platform független, azaz minden olyan operációs rendszer alatt képes futni, amihez létezik Java futtató környezet.

Széleskörű támogatást nyújt a legtöbb J2EE szabvány használatához, mint például az EJB, JTA (Java Transaction API – tranzakció kezelés), a webes technológiák, mint a servlet-ek, JSP-k (Java Server Pages), vagy a webszolgáltatások. Emellett a naplózó szolgáltatásokat is támogatja, amik közül kiemelendő a Commons Logging, és a következő részben tárgyalt Log4J.

3.6 Log4J Logging

A Log4J egy Java alapú naplózó eszköz. A naplózás során elkülönítünk különböző szinteket, amiket a naplóbejegyzésekhez lehet rendelni. Más szintet lehet rendelni egy hibakeresést segítő bejegyzéshez, mint például egy kritikus hibát jelzőhöz. A Log4J által használt szintek a következők:

Szint	Leírás
FATAL	Komoly hibák, amik hirtelen leállást okoznak. Érdemes ezeket az üzeneteket azonnal láthatóvá tenni egy állapot konzolon.
ERROR	Egyéb futás közbeni hibák, vagy váratlan körülmények.
WARN	Olyan nem várt futás közbeni szituációk, amik nem feltétlenül hibák, inkább figyelmeztetések.
INFO	Fontos futás közbeni események.
DEBUG	Részletes információk a program futásáról, többnyire csak naplófájlokban jelenik meg, nem az állapot konzolon.
TRACE	Sokkal részletesebb információk a program futásáról.

A Log4J-t általában egy XML konfigurációs fájl segítségével állítják be. Ebben le lehet írni különböző Appender-eket, amik végrehajtják a naplóbejegyzés kiírását. Ilyen appender lehet például a konzolra író ConsoleAppender, vagy a fájlba dolgozó FileAppender, de ezen kívül a keretrendszer bővíthető bármilyen, akár adatbázisba író appender-rel.

Lehetőség van a bejegyzések kinézetét is meghatározni, ez a Layout-ok (elrendezés) segítségével történik.

A naplózási szinteket, és az appender-eket akár Java csomagokra lebontva is előre konfigurálhatjuk.

3.7 Webszolgáltatások

A webszolgáltatások leírására minden nagyobb szoftvergyártónak van definíciója, ám ezek az alapokban megegyeznek. Az IBM meghatározása a következő:

„A webszolgáltatás egy olyan felület, amely a hálózaton keresztül, szabványos XML üzenetekkel elérhető műveletek egy csoportját írja le. A webszolgáltatások kielégítik egy adott feladat vagy feladatcsoport igényeit. A webszolgáltatáshoz szabványos, formális XML leírás tartozik, amelyet a szolgáltatás leírásának nevezünk, és ez tartalmazza a szolgáltatás igénybe vételéhez szükséges összes részletet, többek között az üzenetek formátumát, az átviteli protokollokat és a szolgáltatás pontos helyét.

A felület elrejti a szolgáltatás megvalósításának részleteit, így a szolgáltatás használata független az azt megvalósító hardver- és szoftverfelülettől, illetve az adott megvalósítás megírásához használt programozási nyelvtől. Ez teszi lehetővé, hogy a webszolgáltatásokon alapuló programok közötti kapcsolat igen laza maradhasson, maguk a megvalósítások pedig elemekre épülő, többféle eljárást felhasználó megoldások legyenek. A webszolgáltatásokat használhatjuk önmagukban, illetve más webszolgáltatásokkal együtt is, és összetett üzleti tranzakciókat hajthatunk végre a segítségükkel.”

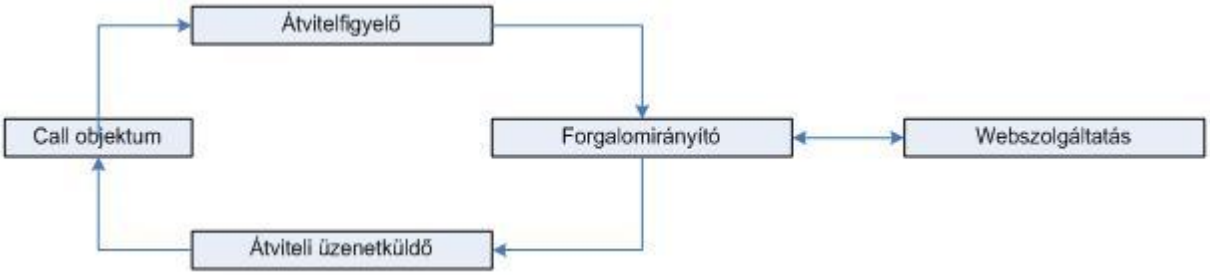
3.8 Apache Axis

Az Axis egy nyílt forráskódú, XML alapú webszolgáltatás keretrendszer. Többféle eszközt ad a webszolgáltatások kigenerálására, és telepítésére. A segítségével egymással együttműködő, elosztott alkalmazásokat készíthetünk. Működése, illetve annak menete, hogy egy kérés hogyan jut el a webszolgáltatásig, illetve a válasz vissza a szolgáltatás igénylőjéhez a következő:

1. A szolgáltatás igénylője felépít a Call objektum segítségével egy SOAP kérést (amiben meghatározza a szolgáltatás URI-jét, nevét, a hívandó metódust és annak paramétereit), majd az üzenet az Axis üzenetfeldolgozó csomópontjához kerül, ahol a hálózati protokollnak megfelelő formára alakul.

2. Az átvitelfigyelő Message objektummá alakítja az üzenetet, majd beteszi azt a MessageContext objektumba, ahol beállítja az átviteli protokollt (ez lehet például smtp, http, vagy ftp).

3. A forgalomirányító felelős a kérés webszolgáltatáshoz történő továbbításáért.
4. A webszolgáltatás végrehajtja a metódust, majd visszaadja a választ a forgalomirányítónak.
5. A forgalomirányító továbbítja a választ az átviteli üzenetküldőnek.
6. Az átviteli üzenetküldő visszaküldi az XML üzenetet a hálózati protokoll segítségével a szolgáltatás igénylőjéhez.



1. ábra – Az Axis komponensek kommunikációja

3.9 Google Web Toolkit

A GWT segítségével AJAX előtétrendszereket írhatunk Java nyelven, amit a GWT átfordít a legtöbb böngészőn helyesen futó, optimalizált JavaScript-re. Mindeközben lehetőségünk van végiglépkedni a Java kódon, illetve hibát keresni benne.

A GWT fordító analizálja és optimalizálja a kódot, általában olyan JavaScript-et gyárt, ami gyorsabban betöltődik és gyorsabban is fut, mint egy vele ekvivalens kézzel írt szkript. Például kiveszi a soha nem használt kódot, osztályokat, mezőket, metódusokat.

3.9.1 Kommunikáció a szerverrel RPC-n keresztül

A GWT sok átviteli protokollt támogat, mint a JSON, vagy az XML. A hagyományos Java RMI-hez hasonlóan (Remote Method Invocation – Távoli eljáráshívás) egyszerűen létre kell hoznunk egy interfészt, ami specifikálja a meghívandó távoli metódusokat. Amikor meghívunk egy távoli metódust a böngészőből, a GWT RPC automatikusan szerializálja a paramétereket, meghívja a metódust a szerveren, majd deszerializálás után visszaadja a visszatérési értéket.

3.9.2 Felhasználói igényekre optimalizált JavaScript

A GWT fordító minden szélesebb körben elterjedt böngészőre elkészíti az optimalizált kódot. Emellett mivel támogatja a nemzetközisítést, annyi verziót fordít, ahány nyelven készítettük el az alkalmazást. Végeredményképpen a JavaScript kód sokkal gyorsabb és kompaktabb lesz, mivel nem lesz teletűzdelve olyan ágakkal, amik böngészőtől függően más kódrészletre terelik a végrehajtást.

Viszont emiatt a fordítás természetesen sokkal időigényesebb.

3.9.3 A böngésző vissza gombjának, és az előzményeknek a támogatása

A GWT lehetőséget ad az alkalmazás állapotának tárolására az URL-ben, így képes kezelni a navigációt a böngésző navigációs gombjai segítségével.

3.9.4 Widget-ek

A widget (vagy vezérlő) a grafikus felhasználói felület egy része, ami információt jelenít meg, vagy a felhasználó által változtatható, mint egy ablak, vagy egy szövegbeviteli mező. A widget-ek alapvető vizuális építőkövek, amiket összekapcsolhatunk egy alkalmazásban. Az alapokból aztán nagyobb, összetettebb működésű komponenseket készíthetünk. A GWT a következő widget-eket tartalmazza:

- Egyszerű HTML elemek (gomb, rádiógomb, jelölőmező, szövegmező, jelszómező, szövegbeviteli terület, hiperhivatkozás, lista, táblázat, stb.)
- nyomógomb, kapcsológomb
- menüsáv
- fa
- füleket tartalmazó menü
- dialógusdoboz
- különféle panelek
- RichText beviteli terület

4 Permutációs játékok

4.1 Egyszemélyes játékok

Az alábbiakban néhány egyszemélyes permutációs játék leírását adom meg, amiket viszonylag könnyen meg lehet valósítani. Ezek közül a Tilitoli-t fogom elkészíteni.

4.1.1 Soliter

Bármely golyóval át lehet ugrani egy másikat, ha annak túloldalán nincs másik golyó. Ha egy golyót átugrottunk, akkor az eltűnik. Cél: minél több golyót eltüntetni.

4.1.2 Golyórendezős játék

A golyókat úgy kell elrendezni, hogy minden sorban, vagy minden oszlopban egyszínűek legyenek.

Itt a feladat az, hogy a meg kevert golyókat úgy rendezzük, hogy vagy minden sorban, vagy minden oszlopban a golyók azonos színűek legyenek. Valójában 16 számozott golyót is szám sorrendbe lehetne rakni, így aztán ez tényleg mindig megoldható.

4.1.3 Tilitoli

A számokat kell növekvő sorrendbe rakni tologatással.

Ennél a cél az, hogy a megkevert sorrendben lévő 16 számot úgy tegyük rendbe, hogy a 4x4-es táblázatban az üres helyre mindig egy szomszédos számlapot tolhatunk.

Fontos, hogy itt a számok sorrendjét figyeljük, az üres mezőt pedig függetlenül helyétől átugorjuk. tehát az alábbi táblázathoz tartozó permutáció például: 1, 5, 2, 3, 4, 6, 10, 7, 8, 9, 11, 12, 13, 14, 15, 16.

1			
5	2	3	4
6	10	7	8
9		11	12

13 14 15 16

Ez a permutációs játék nem mindig oldható meg.

4.1.4 Forgatós

Itt a cél az, hogy egy 3x3-as táblázatban lévő 9 darab számot helyes sorrendbe rakjuk, azaz az első sorban 1 2 3, a másodikban 4 5 6, a harmadikban 7 8 9 kell, hogy álljon. Ennek érdekében egy 2x2-es keretben lévő négy számot forgathatunk.

Az ilyen játékoknál mindig fel lehet tenni a kérdést, hogy vajon mikor oldható meg ez a feladat. Magyarán előfordulhat-e az, hogy a gép olyan sorrendet határoz meg számunkra, amit nem lehet rendbe rakni ilyen lépésekkel. Ennél a játéknál a válasz az, hogy minden létező sorrendet rendbe lehet rakni.

4.1.5 Rubik körök

A golyókat úgy kell elrendezni, hogy a baloldali íven piros, a jobboldalin zöldek legyenek

Rubik körök (alább) célja az, hogy a golyókat színük szerint osszuk szét a két körbe. (A két körnek persze van két közös pontja, ezért úgy kell rendbe rakni a golyókat, hogy a két közös pont közül az egyikben piros, a másikban zöld legyen. Lásd a játék jobb fölső sarkában lévő kis ábrát). Megszámozva mind a 46 golyót, már teljesen világos, hogy ez valóban permutáció játék. A különbség mindössze annyi, hogy nem csak egy kimenetelt tekintünk megoldásnak. Ugyan így látható, hogy a Rubik kocka is erre az elvre épül.

A Rubik körök mindig megoldható.

4.2 Kétszemélyes játékok

Többszemélyes játék gyanánt a GO egy egyszerűsített változatát készítem el.

4.2.1 Go

A GO játék egy távol-keletről származó játék, amely Kínában alakult ki mintegy 4000 éve. A játékot ketten játsszák, egy négyzet alakú táblán. A játékosok fekete illetve fehér kövekkel lépnek felváltva. A játék szabályai igen egyszerűek, könnyen tanulhatók, ugyanakkor a játék maga igen bonyolult. A játéknak, a sakkhoz hasonlóan, komoly nemzetközi szervezete van, nyilvántartva az egyes tagok játékerejét. A sakkal ellentétben a GO-hoz még nem született olyan program, amely akárcsak megközelítené egy nagymester játékerejét.

Sötét 181 db, világos 180 db követ kap. A játszma kezdetekor a 19 vízszintes és 19 függőleges vonalból rácsozott játékmező üres.

Sötét kezdési jogával, a versenyzők felváltva, egyesével rakják le köveiket a játékmező rácspontjaira (a vonalak metszéseire), arra törekedve, hogy minél nagyobb területet kerítsenek el maguknak a táblán.

A kiszemelt terület "tiszta elkerítés" útján történő megszerzése (amikor csak az elkerítő saját kövei határolják az elkerített területet) az ellenfél védekező lépései közepette, szinte soha sem sikerülhet. Az elkerítést akadályozó ellenséges köveket azonban meg lehet "ölni", halálukkor levehetők a tábláról és a végelszámoláskor: a megszerzett területért kapott pontjaink is megnövekednek majd a megölt ellenséges kövek számával.

Egy ellenséges kő (vagy egy ellenséges kőcsoport) úgy ölhető meg, hogy azt (vagy azokat) minden rácsirányban saját köveinkkel bekerítjük.

Fentieken kívül három fontos (a játék közben esetlegesen felmerülő vitákat eldöntő) lépésszabály van:

Nincsen lépéskényszer, ha a lépésre következő játékos nem talál előnyös lépést, akkor passzolhat.

Tilos az "öngyilkosság". Egy, már teljesen körbezárt terület egyetlen még üres belső pontjára tilos úgy letenni kövünket, hogy az (esetleg környező társaival együtt) "fogságba" essen. Kivétel, ha a különben fogságba eső kövünk lerakásával egyúttal ellenséges köve(ke)t is megölünk, ill. foglyul ejtünk. Annak eldöntésében, hogy "ki kinek a foglya": a támadó (aki a követ leteszi) előnyt élvez. Ilyen esetben mindig a támadó "öl először" és így a kettős helyzetet a lépéssel foglyul ejtett kő levételével egyértelművé teszi.

Két lépésen belül nem állhat elő azonos állás, tehát az ilyenre vezető lépés(ismétlés) tilos. (Ez a szabály nem a visszaütést tiltja, hanem a visszaütésnek azt a fajtáját, amikor egy megelőzővel azonos állás jönne létre - persze a tábla egészét tekintve -.)

Kézenfekvő, de a szabályokhoz tartozik, hogy a játékmező széle: határoló, bekerítő szerepet tölt be és így többnyire a "támadót" segíti. (A tábla szélének szorított és a tábla felőli pontjain bekerített ellenséges kövek fogságba esnek.)

A játszmának akkor van vége, ha mindkét játékos passzolt. Ekkor következik az eredmény összeszámolása: először betömjük a semleges pontokat, aztán levesszük a területünkön foglyul ejtett köveket, majd összeadjuk az általunk körbekerített (rács)pontok és a levett (megölt) ellenséges kövek számát. A pontértékek összevetése előtt, világos még 4,5 plusz pontot kap (kezdési hátrányának kiegyensúlyozására).

4.2.1.1 Egyszerűsített GO

Az egyszerűsített GO tulajdonképpen egy új szabály bevezetését jelenti, mégpedig, hogy aki először üt, az a nyertes. Ez jelentősen egyszerűsíti az egész játékot, egyértelművé téve a győztest. A nyertes személyének eldöntése a GO játék egyik igen bonyolult problémája, amellyel önmagában dolgozatokat lehetne megtölteni. E probléma nélkül is marad elég, ezért a játék problémáinak kezelése továbbra sem triviális.

5 A megvalósítás lépései

Az alábbiakban részletesen kifejtem az alkalmazás elkészítésének mozzanatait. A fejlesztés során az Eclipse Framework legújabb 3.4.1.0-ás verzióját, és a Java EE 1.5-öt használom.

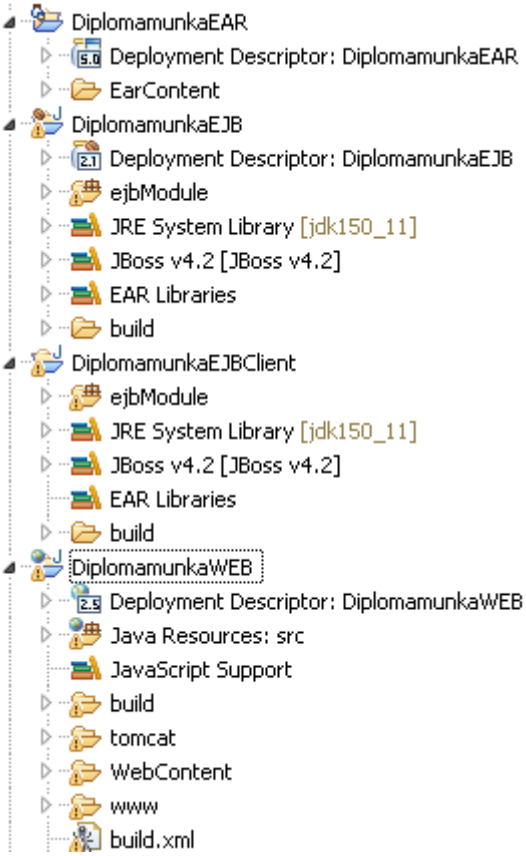
5.1 A webes alkalmazás elkészítése

A webes alkalmazás alatt a komplett felület értendő a hozzá kapcsolódó szolgáltatásokkal, webszolgáltatásokkal, naplózással, valamint az adatbázissal együtt. A játékok megvalósítását későbbi fejezetekben tárgyaljuk.

5.1.1 A projektek létrehozása

Az alkalmazáshoz négyféle projektre lesz szükség, amit az Eclipse-ben pár kattintással létre tudunk hozni.

- DiplomamunkaEAR
  Egy Enterprise Application Project, ez fogja össze az EJB, valamint a Web projekteket, illetve ez kerül betelepítésre az alkalmazásszerverbe.
- DiplomamunkaEJB
  Ez egy EJB projekt, ez tartalmazza az üzleti réteget megvalósító Enterprise JavaBean-eket.
- DiplomamunkaEJBClient
  Az EJB projekt mellé automatikusan generálódik, azokat az interfészeket és osztályokat tartalmazza, amik az üzleti logikát tartalmazó EJB-k hívásához szükségesek.
- DiplomamunkaWEB
  Ez a projekt tartalmazza a webes réteget, itt találhatók a GWT kódjai, valamint a webszolgáltatások.



2. ábra - Eclipse projektek

5.1.2 Az üzleti réteg

Az EJB-ket tartalmazó réteg. Az alkalmazásban állapot nélküli bean-eket használok az üzleti logika megvalósítására. EJB-t az Eclipse segítségével könnyen létre lehet hozni, majd kibővíteni a szolgáltatásainkkal. A használatukhoz szükséges osztályok, interfészek, és leíró XML dokumentumok az XDoclet nevű eszköz segítségével kigenerálhatók. Az XDoclet-et a kódban elhelyezett úgynevezett annotációk (jelzések, megjegyzések) használatával tudjuk munkára bírni. A következő annotáció eredménye az lesz, hogy az eszköz az ezzel jelölt metódust elérhetővé fogja tenni a lokális interfészen keresztül, és ezt az információt az EJB leíróba is beteszi:

```
     * @ejb.interface-method view-type="local"
```

Ez pedig azt fogja eredményezni, hogy a kódból egy állapotmentes session bean jön létre, amit az alkalmazásszerver „GoServer” néven tesz majd be a JNDI fába, valamint azt is kiköti, hogy az EJB tranzakcióit a szervernek kell kezelnie (konténer menedzselt tranzakciók):

```
@ejb.bean name="GoServer" description="An EJB named GoServer"
display-name="GoServer" jndi-name="GoServer" type="Stateless"
transaction-type="Container"
```

Az EJB-k az alkalmazásban a Java által biztosított adatbázis kezelő eszközök segítségével hívják meg az adatbázisban létrehozott tárolt eljárásokat, ezáltal kezelve az adatokat.

5.1.2.1 UserBusinessBean

Ez a bean valósítja meg a felhasználó-kezeléssel kapcsolatos szolgáltatásokat, melyeket a következő táblázatban foglaltam össze:

Szolgáltatás neve	**Leírás**
isUserExists	Paraméterül megkap egy felhasználónevet, visszatérési értéke egy logikai érték, ami megmondja, hogy a megadott névvel létezik-e regisztrált felhasználó.
registerUser	Beregisztrál egy felhasználót a kapott adatokkal: felhasználónév, teljes név, jelszó.
login	Megvizsgálja, hogy a kapott felhasználónévvel és jelszóval létezik-e regisztrált felhasználó. Ha igen, akkor visszaadja a felhasználó adatait.
getUserById	A megadott adatbázis azonosítóval rendelkező felhasználó adatait adja vissza, a jelszó kivételével.

5.1.2.2 GoServerBean

Ez az EJB felelős a Go játék menetéhez szükséges szolgáltatások nyújtásáért. Ezeket a funkciókat fogjuk később kipublikálni a Go kliens számára. Íme a metódusai:

Szolgáltatás neve	Leírás
makeGameRequest	A felhasználó amikor belép a Go kliensben, sikeres azonosítást követően lefut ez a szolgáltatás. Lement egy úgynevezett játék-szándékot az adatbázisba. A játék-szándékról a Go program megvalósításáról szóló fejezetben lesz szó.
getGameRequestorUsers	Visszaadja azokat a felhasználókat, akiknek van nyitott játék-szándékuk.
requestUserToPlay	A felhasználó a nyitott szándékkal rendelkezők közül kiválasztva egyet, ennek a szolgáltatásnak a segítségével hívja ki a megadott azonosítóval rendelkező játékost játékra.
getPlayRequest	Paramétere egy azonosító, feladata megmondani, hogy az adott felhasználót kihívta-e valaki játékra.
checkRequestAccepted	Ellenőrzi, hogy a játékra kihívást elfogadta-e a kiválasztott ellenfél.
acceptPlayRequest	A szolgáltatás segítségével elfogadjuk az ellenfél kihívását.
rejectPlayRequest	A szolgáltatás segítségével elutasítjuk az ellenfél kihívását.
sendStep	Megkapja az aktuális játékos színét, és hogy melyik pozícióra szeretné letenni a követ. Feladata ellenőrizni, hogy a lépés megengedett-e, illetve ha igen, akkor az új kialakult játékállapot lementése az adatbázisba.
getLastGameState	Visszaadja a legutolsó lementett játékállapotot.
deleteGameRequest	Töröltre állítja a megadott játékos játék szándékát. A kliensből kilépéskor hívjuk.
getLastTwoStates	Visszaadja a legutóbbi két lementett játékállapotot.

5.1.3 Naplózás

Az alkalmazásban az összes EJB szolgáltatás-hívás naplózva van, a hívás idejével, a szolgáltatás nevével, illetve paramétereivel. A naplózás a Log4J eszköz segítségével történik, mely úgy van bekonfigurálva, hogy külön naplófájlba, valamint az adatbázisba is naplózzon. A perzisztált bejegyzések később jó alapjai lesznek használati, működési statisztikák kimutatására.

5.1.4 A prezentációs réteg

Egy GWT-s webalkalmazás alapjában véve három különálló részre bontható.

- public könyvtár
  Ide kerülnek a statikus állományok, mint például a CSS stíluslapok, HTML oldalak, képek.
- client végződésű csomagok
  Ebbe a könyvtárba kell betennünk a használni kívánt saját készítésű Widget-einket, az RPC szolgáltatás interfészeket, ide értve az alap, és az aszinkron interfészt. Ezekből a forrásokból Javascript fog generálódni.
- server végződésű csomagok
  Az RPC szolgáltatás megvalósítások helye, amikből Java osztályok, úgynevezett servlet-ek fordulnak majd.

A GWT a szolgáltatás interfészek által közös felületet biztosít a szerver és a kliens oldalon, a szolgáltatás hívását pedig aszinkron módon biztosítja. Következzen egy példa, a UserService, ami a UserBusinessBean EJB szolgáltatásait használja.

5.1.4.1 UserService

Ez a közös interfész, a client csomagban található, forráskódja a következő:

```
package hu.neuron.client;

import ...

@RemoteServiceRelativePath("userService")
public interface UserService extends RemoteService {
```

```
    public static final String LOGIN_SUCCESS_SESSION_KEY =
"loginSuccess";

    public UserWebVO login(String name, String password)
throws Exception;

    public Boolean logout();

    public Boolean isUserExists(String userName) throws
Exception;

    public Boolean registerUser(String fullName, String
userName, String password) throws Exception;

}
```

5.1.4.2 UserServiceImpl

Ebből az osztályból egy Java servlet fordul, és a server csomagban kell elhelyezni.

```
package hu.neuron.server;

import ...

public class UserServiceImpl extends RemoteServiceServlet
implements UserService {
...
    public Boolean isUserExists(String userName) throws
Exception {
        // A UserBusiness EJB lokális interfészének elkérése
        UserBusinessLocalHome userBusinessHome =
UserBusinessUtil.getLocalHome();
        // EJB példány létrehozása
```

```
          UserBusinessLocal userBusiness =
userBusinessHome.create();
          // Az EJB szolgáltatás meghívása
          Boolean result = userBusiness.isUserExists(userName);

          if (result == null) {
               throw new Exception();
          }

          return result;
     }
...
}
```

5.1.4.3 UserServiceAsync

Ez az aszinkron interfész, amely a client csomagban kell helyet foglaljon. A widget-ekből ezen keresztül hívjuk a szolgáltatásokat.

```
package hu.neuron.client;

import hu.neuron.client.vo.UserWebVO;

import com.google.gwt.user.client.rpc.AsyncCallback;

public interface UserServiceAsync {

     void login(String name, String password,
AsyncCallback<UserWebVO> callback);

     void logout(AsyncCallback<Boolean> callback);

     void isUserExists(String userName, AsyncCallback<Boolean>
callback);
```

```
    void registerUser(String fullName, String userName, String
password, AsyncCallback<Boolean> callback);
}
```

5.1.4.4 Az aszinkron hívás

A konkrét RPC hívás menete:

```
final UserServiceAsync userService =
GWT.create(UserService.class);
ServiceDefTarget userServiceEndpoint = (ServiceDefTarget)
userService;
String moduleRelativeURL = GWT.getModuleBaseURL() +
"userService";
userServiceEndpoint.setServiceEntryPoint(moduleRelativeURL);

userService.login("guest", "guest", new
AsyncCallback<UserWebVO>() {

    public void onFailure(Throwable caught) {
        // a hívás sikertelenségének lekezelése
    }

    public void onSuccess(UserWebVO result) {
        // sikeres hívás esetén ez a kód fut le
    }
});
```

Mint láthatjuk, az RPC hívás az aszinkron interfészt használja, amiben a metódusok kapnak egy plusz AsyncCallback típusú paramétert. Az AsyncCallback két metódust tartalmaz, az onFailure fut le akkor, ha a hívás sikertelen volt, az onSuccess pedig akkor, ha sikeres.

5.1.4.5 TextConstants

Egy egyszerű interfész, ami a nemzetköziesítést szolgálja. A felhasználandó szöveges konstansok neveivel megegyező nevű, String típusú metódusokat kell benne felvenni, majd ezzel a névvel felvenni a sztringeket tartalmazó fájlban a szöveget. Példa:

```
package hu.neuron.client;
import com.google.gwt.i18n.client.Constants;
public interface TextConstants extends Constants {
     String loginIncorrect();
}
```

A szövegeket tartalmazó fájlban a bejegyzés:

```
loginIncorrect=Bejelentkezés sikertelen!
```

És végül a konstans felhasználása a widget-ben:

```
final static TextConstants textConstants =
GWT.create(TextConstants.class);
// kidobjuk a bejelentkezés sikertelen dialógusdobozt
Window.alert(textConstants.loginIncorrect());
```

5.1.4.6 DecoratedBoxAbs

A weboldal dobozokból fog összeállni, lesz például a bejelentkezéshez egy, a játékok kilistázásához, a regisztrációs űrlaphoz. Ennek támogatására a GWT tartalmaz egy előre elkészített panel típust, ez a DecoratedPanel, ami egy egyszerű panel, egy ízléses lekerekített sarkú oldalszegéllyel.

Az általam készített absztrakt DecoratedBoxAbs az alap DecoratedPanel-t bővíti ki olyan lehetőségekkel, aminek segítségével az egyéni dobozainkat létre tudjuk hozni. Például be lehet neki állítani a doboz címkéjét, méretét, valamint a tartalmát.

5.1.4.7 DiplomamunkaApplication

Ez az osztály az alkalmazás belépési pontja. Példányosítja a felületi komponenseket, és elhelyezi azokat a HTML oldalon.

5.1.4.8 GamesTreeBox

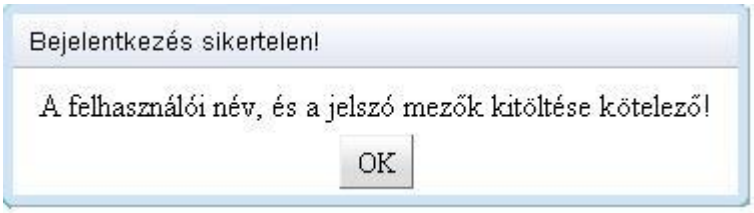
A DecoratedBoxAbs leszármazottja, ez az osztály valósítja meg a játékok listázására kihelyezett fastruktúrát. A fa tartalmazza az egyszemélyes játékok csomóponton belül a Tilitolit, a kétszemélyes játékokon belül pedig a GO-t. Definiálva van egy eseményfigyelő is, ami a játék kiválasztásakor betölti a jobboldali dobozba a játékot.

5.1.4.9 LoginBox

Ez az osztály is DecoratedBoxAbs leszármazott, és a normál bejelentkezés, a vendégként bejelentkezés, valamint a kijelentkezés kezelése a feladata. Alapértelmezésben két beviteli mezőt tartalmaz, hová a felhasználói nevet és jelszót lehet beírni, valamint egy belépés, és egy belépés vendégként gombot. A belépésre kattintva ellenőrzi, hogy minkét mező ki van-e töltve, ha igen, akkor a UserService segítségével belépteti a felhasználót. Sikeres bejelentkezés után a doboz tartalma megváltozik, egy üdvözlő szöveg jelenik meg a felhasználó teljes nevével, és egy kilépés gomb, amire kattintva kijelentkezik a felhasználó, és visszaáll a doboz alapértelmezett állapota.

5.1.4.10 MessageDialogBox

Segédosztály, egy személyre szabott párbeszédablakot valósít meg. Címkét és üzenetszöveget lehet neki megadni, ezen kívül tartalmaz egy Ok gombot, amire kattintva eltűnik. Ezt az ablakot használjuk az alkalmazás üzeneteinek közlésére.



3. ábra - MessageDialogBox

5.1.4.11 RegistrationBox

A DecoratedBoxAbs leszármazottja. Egy regisztráció gombot tartalmaz, amire kattintva megjelenik a jobboldali dobozban a regisztrációs űrlap.

5.1.4.12 SpacerTable

Egy egyszerű panelt valósít meg, aminek a célja a képernyőn elhelyezett dobozok közötti távolság biztosítása.

5.1.4.13 RegistrationForm

A regisztrációs űrlap megvalósítását tartalmazza. Négy beviteli mező kapott rajta helyet a felhasználónév, a teljes név, a jelszó, valamint a jelszó ismételt megadásához, ezen kívül egy gombot a regisztráció elindításához. A gombra kattintáskor ellenőrzi, hogy minden mező ki van-e töltve, és hogy a két jelszó mező azonos értéket tartalmaz-e. Ha minden rendben van, először a UserService isUserExist szolgáltatásának segítségével megnézi, hogy létezik-e már a beírt felhasználónévvel már beregisztrált felhasználó. Ha a felhasználónév szabad, és minden mező helyesen van kitöltve, akkor a registerUser szolgáltatás használatával bejegyzi az új felhasználót. A felület képernyőképe a regisztrációs űrlappal látható a függelékben ezen a néven: 7. ábra - Az alkalmazás képernyőképe a regisztrációs űrlappal.

5.1.4.14 WebVO-k

Az EJB-k szolgáltatásainak egy része nem egyszerű Java típusokat, csomagolóosztályokat ad vissza (Boolean, Long, stb.), hanem szükség van több attribútumot tartalmazó objektumok, úgynevezett VO-k (Value Object) használatára. Ilyen például a UserVO, ami tartalmazza a felhasználói nevet, a teljes nevet, jelszót. Azonban ezek a VO-k nem használhatók a kliens oldali GWT widget-ekben, mert ezek nem láthatók, csak a szerver oldalon. Emiatt a használt VO-kat át kell alakítanunk WebVO-kra, mégpedig a szerver oldali RPC szolgáltatás megvalósításban, és az aszinkron interfészen keresztül a WebVO-kat kell használnunk, amiből már képes a GWT fordító Javascript kódot készíteni. **FONTOS!** A GWT alkalmazásban kliens oldalon minden elkészített Java kódból Javascript generálódik!

5.1.5 Webszolgáltatások kigenerálása

A webszolgáltatások fejlesztéséhez, és használatához kétféle építkezési mód létezik. Az Axis képes meglévő Java osztályokból automatikusan webszolgáltatás leírót (WSDL) generálni, valamint kipublikálni azt az alkalmazásszerveren. Ehhez a web rétegben kellett létrehozni egy egyszerű osztályt, ami az EJB-k szolgáltatásait hívja, majd ebből az Eclipse és az Axis segítségével kigeneráltatni a szükséges fájlokat. Ez az úgynevezett BottomUp mód, amikor a kezdeti egyszerű osztályból generálunk.

A másik építési mód a TopDown, ezt a webszolgáltatást használó kliensek létrehozására használjuk. Ekkor egy megadott WSDL fájl alapján hozza létre az Axis az összes, a szolgáltatások használatához szükséges állományt. Létrehoz a szinkron, és az aszinkron hívások kezeléséhez is osztályokat, valamint a szolgáltatás paraméterek és visszatérési értékek leképezéséhez is, és segédosztályokat a konkrét hívás levezényléséhez.

A Go kliens fejlesztéséhez is szükség van a Go szerver által nyújtott szolgáltatások használatára, ezért készítettem egy külön projektet DiplomamunkaWebServiceClients néven, ami az Axis-nak a WSDL-ekből generált osztályait tartalmazza.

5.2 A játékok elkészítése

A játékok implementációit is az Eclipse Framework 3.4.1-es verziója segítségével készítem el.

5.2.1 Tilitoli

A Tilitoli játéknak egy Java Applet-ként megvalósított, és a weboldalra kihelyezett változatás programoztam le. A játék egy Apache Ant típusú projekt keretében készült, amihez egy build-fájl tartozik, aminek segítségével történik a program lefordítása, illetve a disztribúció elkészítése.

A program Java osztályait a hu.neuron.attila.ujjobbagy.tilitoli csomagban helyeztem el, a fontosabb osztályok, és működésük a következő:

5.2.1.1 ImageSelectionButton

Ez az osztály az alapértelmezett Java-s Swing JButton leszármazottja, ősétől abban különbözik, hogy tartalmaz egy plusz attribútumot imageName néven, amiben a kiválasztható kép neve van tárolva.

4. ábra - Tilitoli képválasztó képernyő

Ilyen gombok találhatók a kép kiválasztó képernyőn, ahogy a fenti ábra mutatja.

5.2.1.2 ImageSelectionScreen

Ez az osztály a képválasztó képernyő megvalósítása. Itt jönnek létre az előzőekben bemutatott ImageSelectionButton-ok, illetve itt állítom rájuk az egéresemény figyelőket, amiknek a feladata az, hogy kattintásra továbbvigye a játékost a játékmenetben a következő képernyőre.

5.2.1.3 Level

A játékban lehetőség van a nehézség megválasztására. Három nehézségi fokozat közül lehet választani, ezek abban különböznek, hogy a kép hány részre lesz feldarabolva.

Nehézségi fok neve	**Kép darabjainak száma**
EASY (könnyű)	3x4
NORMAL (normál)	6x8
HARD (nehéz)	9x12

Ezeket az információkat tartalmazza a Level osztály.

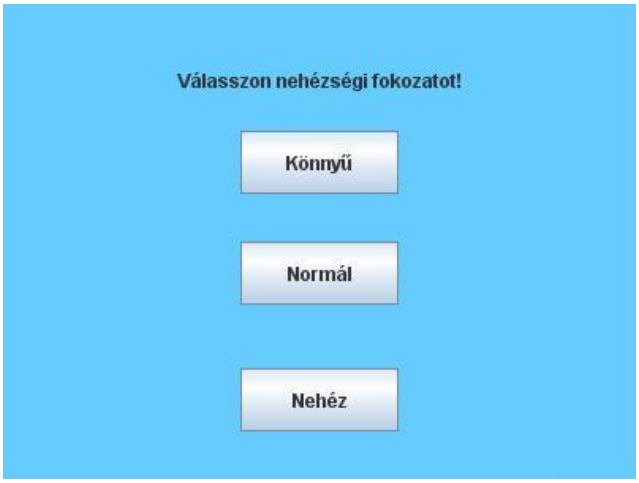
5.2.1.4 LevelSelectionButton

Hasonlóan az ImageSelectionButton-hoz, ez is a JButton osztály leszármazottja. Attól annyiban különbözik, hogy tartalmazza a hozzárendelt nehézségi szint adatait.

5.2.1.5 ImageSelectionScreen

A játékmenetben ez az a képernyő, ahol kiválasztható a nehézségi szint. Lényegében egy panel, amelyre rárakom a három LevelSelectionButton-t, a három szinttel.

A képernyő képe alant látható.



5. ábra - Tilitoli nehézségi szint kiválasztó oldal

5.2.1.6 Random

Egy lineáris kongruens modulo pszeudo-véletlenszám generátor megvalósítás.

5.2.1.7 TiliToliImageButton

A játékban a feldarabolt kép egy darabját valósítja meg. Nyilvántartom benne, hogy az adott képelem milyen koordinátákon található, hányadik sorban illetve oszlopban.

5.2.1.8 TiliToliTask

Magát a feladatot megvalósító osztály. Tartalmazza a kiválasztott szintet, és a képet. Létrehoz egy, a szintnek megfelelő méretű kétdimenziós tömböt, amibe sorban berakja a kis képelemeket. Metódusokat tartalmaz a tömb menedzselésére, illetve a képelemek mozgatására a képernyőn (moveDown, moveUp, moveRight, moveLeft). Az isDone nevű metódusa dönti el, hogy a feladat meg van-e oldva, azaz ki van-e rakva az eredeti kép. Tartalmaz egy mixThumbs nevű metódust is, aminek a feladata a képelemek összekeverése. Ez a mozgató metódusokat használja véletlenszerűen, emiatt biztosított, hogy a feladat megoldható, egyik biztos megoldása, ha a véletlenszerű mozgatást visszafelé le tudnánk játszani.

5.2.1.9 TiliToliApplet

Ez a fő osztály, a játékmenet vezérlését végzi. A kiválasztott szintnek, képnek megfelelően viszi tovább a játékost a következő képernyőre, létrehozza a TiliToliTask-ot, és kezeli az üres képelem mozgatását a billentyűzet nyílbillentyűinek figyelésével. Minden mozgatás után ellenőrzi, hogy meg van-e oldva a feladat.

6. ábra - Tilitoli játék képernyő közepes szinten

5.2.2 GO

A Go játékot egy asztali alkalmazásként készítettem el, ami a Java Swing-en alapul. Ez egy példa kliens a Go szerver szolgáltatásainak bemutatására.

5.2.2.1 Bejelentkező képernyő

Az alkalmazás egy bejelentkező ablak nyitásával indul, ahol a webes felületen regisztrált felhasználónévvel és jelszóval lehet belépni. Sikeres belépés után a program a Go szerver makeGameRequest szolgáltatása segítségével bejegyzi az adatbázisba, hogy a bejelentkezett felhasználó játszani szeretne. A bejegyzés előtt ellenőrzi, hogy van-e már létező nyitott játék szándéka a felhasználónak, paramétere a játékos adatbázisbeli azonosítója. Egy játék-szándék a létrehozó játékos azonosítójából, a kihívott játékos azonosítójából (ha van), és egy állapotból áll, ami NEW a bejegyzés pillanatában.

A képernyőt a LoginDialogBox osztály implementálja.

5.2.2.2 Ellenfél kiválasztó képernyő

A bejelentkezés után erre a felületre jutunk. A program a getRequestorUsers szolgáltatás használatával lekérdezi az aktív játék szándékkal rendelkező játékosokat, és a nevüket berakja egy a felületen megjelenő listába. Ez megegyezik azokkal, akik éppen be vannak lépve a kliens használatával, nem játszanak éppen, nincs függőben lévő kihívásuk, és őket se hívta ki senki. Mivel ez az állapot változhat, a Lista frissítése gombbal frissíthetjük a neveket.

Itt található még a Felkérés játékra gomb. Ez a requestUserToPlay szolgáltatást használja, ami a játék szándékunkat leíró adatokhoz beírja a kiválasztott ellenfél jelölt azonosítóját. Előfordulhat, hogy a kihívott fél állapota időközben már megváltozott (kihívták, vagy már játszik), ekkor egy hibaüzenet jelenik meg, ami tájékoztat róla, hogy a játékos nem kiválasztható. Ha a felkérés sikeresen elküldésre került, egy várakozó képernyő nyílik, ami tartalmazza majd az információt, hogy a felkért elfogadta-e a kihívást. Ezt a checkRequestAccepted szolgáltatás segítségével ellenőrzi.

Az ellenfél választó képernyő megnyílásával egyidejűleg elindul egy figyelő programszál. Ennek a feladata ellenőrizni, hogy kihívott-e minket valaki játékra. Az ellenőrzést a getPlayRequest szolgáltatás végzi, mégpedig úgy, hogy megkeresi, a felhasználói azonosítónk szerepel-e nyitott játék szándékban kihívott ellenfél azonosítóként. Ha kihívtak, megjelenik egy dialógus ablak, amiben kiválaszthatjuk, hogy a kihívást elfogadjuk-e vagy sem.

Amennyiben elfogadtuk, meghívódik az acceptPlayRequest, ami mindkét fél játék szándékában a státuszt PAIRED-re állítja (tehát innentől ezek a játékosok nem választhatóak a listából). Elutasítás esetén a rejectPlayRequest kitörli az azonosítónkat az ellenfél rekordjában a kihívott fél mezőből. Ennek, és a PAIRED státusznak a vizsgálatával tudja eldönteni a másik fél, hogy mit döntöttünk.

Ha elfogadtuk a kihívást, minkét félnél megjelenik a játéktábla.

Az ellenfélválasztó képernyőt az OpponentSelectionPanel osztály valósítja meg.

A függelék tartalmaz képernyőképet az ellenfélválasztásról.

5.2.2.3 A játéktábla

Ez a felület jeleníti meg magát a játékot. A kliens jelenleg 13x13-as táblát tud kezelni, ennek megfelelően rajzolja ki a vonalhálót.

Az, hogy a fekete, vagy a fehér kövekkel játszunk attól függ, hogy ki volt a kihívó. Mindig a kihívott lesz a fekete, a kihívó pedig a fehér. Ennek megfelelően a fehér játékosnál először egy várakozó képernyő jelenik meg, a másik fél pedig kezdheti a játékot.

Amennyiben mi következünk lépni, a tábla egy pontjára kattintás alkalmával a program elküldi a kapott koordinátákat, illetve a játékos színét, és a játékmenet azonosítót a sendStep szolgáltatásnak. A sendStep első lépésben felkéri adatbázisból a legutóbbi két lépés során előállt tábla állapotot. (Az állapot tartalmazza a játékmenet azonosítót is, ami itt kap először értéket, ha még nem volt eddig.)

- A legutolsó állapotot vizsgálva (getLastTwoStates) először megnézi, hogy a kapott pozíción van-e már kő, ha igen, akkor visszatér ezzel az állapottal, és jelzi, hogy nem megengedett a lépés.
- A második vizsgálat a kettővel ezelőtti állapotot hasonlítja össze azzal, ami a kő letételével állna elő, ezzel küszöbölve ki a KO-t. Visszatérése ekkor megegyezik az előző pontéval.
- Ellenőrzi, hogy öngyilkos lépést készül-e elkövetni a játékos. Ha igen, szintén a fent említett értékkel tér vissza. Ha ez alapján illegális lenne a lépés, viszont ezzel megöl ellenséges követ, akkor engedélyezi a lépést.

Ha a lépés engedélyezve lett, meg is lépi azt, majd kövek és a blokkok életének kiszámítása után eltávolítja a tábláról a halott köveket. Az így előállt új állapotot lementi NEW státusszal, a megölt kövek számával együtt.

Sikeres lépés után várakozik a program az ellenfélre.

Amíg a játékos nem lépett, a másik félnél egy várakozóablak jelenik meg, ami elindít egy szálat, ami figyeli, hogy lépett-e már az ellenfél. Azt a getLastStep szolgáltatás segítségével ellenőrzi, ami azt nézi, hogy létrejött-e NEW státusszal játékállapot az aktuális játékmenet azonosítóval. Ha igen, akkor ezt a bejegyzést DONE-ra állítja, és a program felveszi a lépésre jogosult állapotot.

5.2.2.4 Kilépés a kliensből

Az alkalmazás bezárásakor a deleteGameRequest szolgáltatás a játékos játék szándékát törli az adatbázisból, pontosabban szólva a státuszát töröltre állítja.

6 Összefoglalás

Mint láthatjuk, ahhoz, hogy egy modern alkalmazás elkészüljön, és igazodjon a mai felhasználói igényekhez, elég nagy számú, különböző célokra specializálódott technológiát kell alkalmaznunk. Ezek használatának elsajátítása természetesen sok időt vesz igénybe, de emellett rengeteg tapasztalatot lehet szerezni általuk a fejlesztés, valamint a rendszerben gondolkodás terén. A szabadon felhasználható eszközök segítségével komplex alkalmazásokat építhetünk, mindemellett nem egyszerű feladat a megfelelő eszköz kiválasztása, az alkalmazás elkészítéséhez az internetes közösség által legjobbnak tartottakat, legnépszerűbbeket használtam.

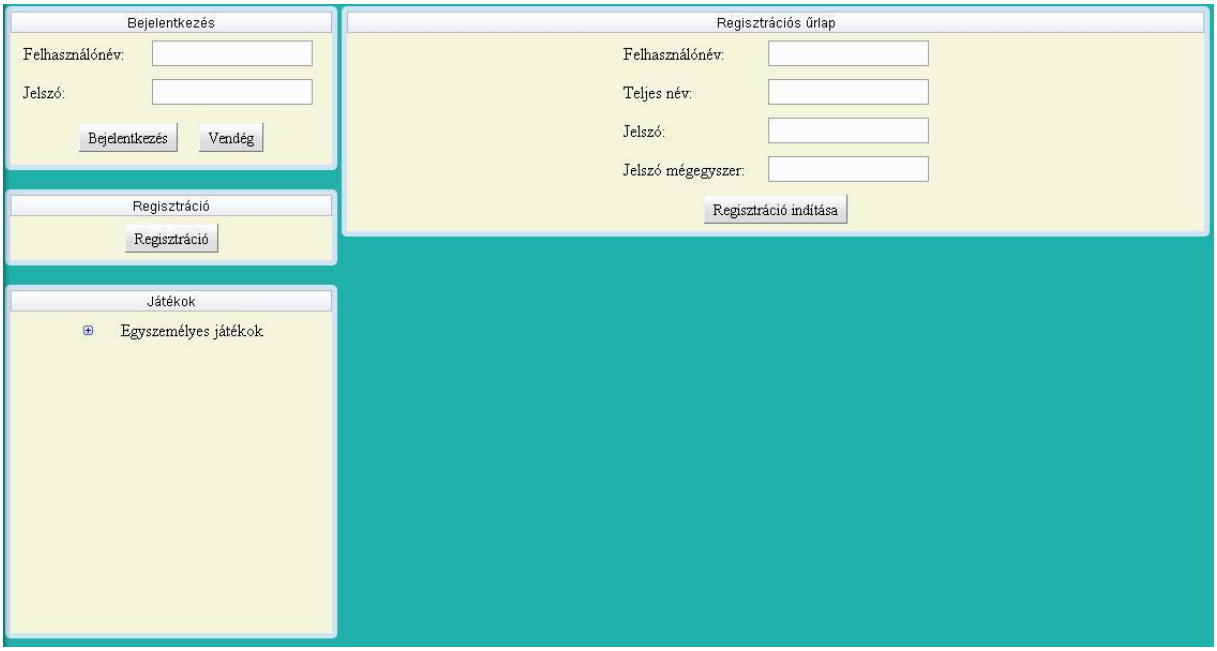
Az alkalmazás jelen állapotában közel sem végleges, de egy jó alapot ad a későbbi fejlesztésekhez, például új játékok megvalósításához.

A design során az egyszerűségre, könnyen kezelhető minimalizmusra törekedtem, ez is változtatható a későbbiekben kisebb fejlesztéssel, vagy akár a nélkül is, a CSS stíluslapok ügyes használatával.

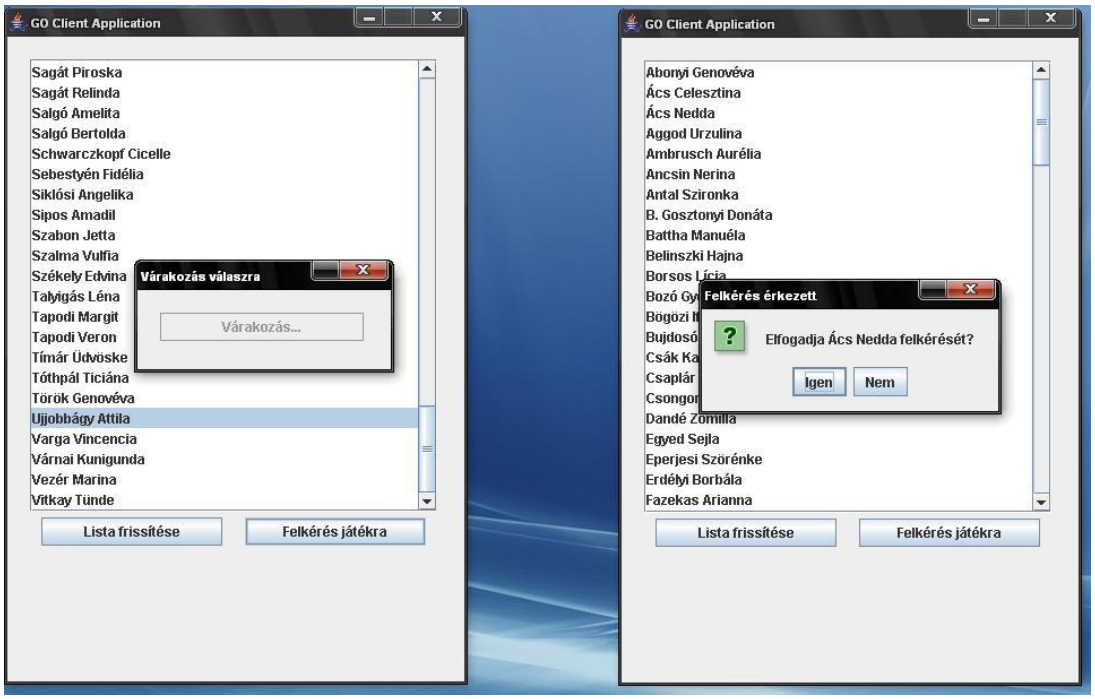
Diplomamunkám elkészülésében nagy szerepe volt témavezetőmnek, Dr. Halász Gábornak, valamint a jelenlegi munkahelyemen webfejlesztéssel eltöltött két és fél év tapasztalatának.

7 Függelékek

7.1 Nagyméretű ábrák



7. ábra - Az alkalmazás képernyőképe a regisztrációs űrlappal



8. ábra - Go kliens - felkérés játékra, a két játékos szemszögéből

7.2 Adatbázis séma

7.2.1 USR_USER tábla

A regisztrált felhasználókat tartalmazza.

Oszlopnév	Leírás	Típus
ID	Elsődleges kulcs	NUMBER(19)
FULL_NAME	Felhasználó teljes neve	VARCHAR2(255)
USER_NAME	Felhasználói név	VARCHAR2(50)
PASSWORD	Jelszó	VARCHAR2(50)

7.2.2 LOG_EVENT tábla

A naplóbejegyzéseket tartalmazza.

Oszlopnév	Leírás	Típus
LOG_DATE	A naplóbejegyzés dátuma	VARCHAR2(50)
LOG_LEVEL	A bejegyzést naplózási szintje	VARCHAR2(5)
LOG_MESSAGE	Maga a bejegyzés	VARCHAR2(4000)

7.2.3 GO_GAME_REQUEST tábla

A Go játékban rögzített játék szándékok.

Oszlopnév	Leírás	Típus
ID	Elsődleges kulcs	NUMBER(19)
USER_ID	Felhasználó azonosítója	NUMBER(19)
STATE	A szándék állapota	VARCHAR2(50)
REQUESTED_USER_ID	Kihívott felhasználó azonosítója	NUMBER(19)

7.2.4 GO_GAME_STATE tábla

A Go játék állapotait tartalmazza.

Oszlopnév	Leírás	Típus
ID	Elsődleges kulcs	NUMBER(19)
SESSION_ID	Játékmenet azonosítója	NUMBER(19)
GAME_STATE	A táblát reprezentálja	VARCHAR2(500)
RECORD_STATE	Az állapot bejegyzés állapota	VARCHAR2(50)
KILLED_BLACK_NUM	Megölt fekete kövek száma	NUMBER(19)
KILLED_WHITE_NUM	Megölt fehér kövek szám	NUMBER(19)

7.2.5 Szekvenciák

Az azonosítók generálására szolgálnak.

Név	Leírás
USR_USER_ID_SEQ	Az elsődleges azonosító generálására szolgál az USR_USER táblában.
GO_GAME_REQUEST_ID_SEQ	Az elsődleges azonosító generálására szolgál az GO_GAME_REQUEST táblában.
GO_GAME_SESSION_ID_SEQ	Játékmenet azonosító generálására szolgál az GO_GAME_STATE táblában.
GO_GAME_STATE_ID_SEQ	Az elsődleges azonosító generálására szolgál a GO_GAME_STATE táblában.

7.2.6 GO_PACKAGE csomag

A Go szerver feladatait segítő funkciókat és eljárásokat tartalmazza.

7.3 Irodalomjegyzék

Diplomamunka / szerző Rácz Árpád.

Google Web Toolkit [Online]. - http://code.google.com/intl/hu-HU/webtoolkit/overview.html.

Háromrétegű architektúra [Online]. - ComBase Kft.. - http://www.combase.hu/3part_hu.html.

J2EE útikalauz Java programozóknak [Könyv] / szerző Nyékyné G. Judit. - Budapest : ELTE TTK Hallgatói alapítvány, 2002.

Java alapú webszolgáltatások [Könyv] / szerző Graham Steve [és mtsai.]. - Budapest : Kiskapu Kft., 2002.

Táblajátékos [Online]. - www.tablajatekos.hu/uj2001.

Webszolgáltatások, XML alapú kommunikáció az interneten [Könyv] / szerző Gottdank Tibor. - Budapest : ComputerBooks, 2003.

Wikipedia [Online]. - http://en.wikipedia.org/wiki/Widget_(computing).

Wikipedia [Online]. - http://en.wikipedia.org/wiki/Apache_Axis.

Wikipedia [Online]. - http://en.wikipedia.org/wiki/Log4j.

Wikipedia [Online]. - http://hu.wikipedia.org/wiki/Eclipse.

Wikipedia [Online]. - http://hu.wikipedia.org/wiki/Java_(programoz%C3%A1si_nyelv).

7.4 Ábrajegyzék