Debreceni Egyetem

Informatika Kar

Kétszemélyes pókerjáték

Témavezető:

Dr. Horváth Géza

Egyetemi adjunktus

Készítette:

Kocsi Zoltán

Programtervező informatikus

Debrecen

2011

Tartalomjegyzék

Bevezetés

Szakdolgozatom célja megismerni és elsajátítani olyan technológiákat, melyek hasznosak lehetek a későbbi munkavégzésnél. Témaválasztásom oka, hogy napjainkban nagy népszerűségnek örvend ez a játék. Dolgozatomban be szeretném mutatni, hogy milyen módon lehet megvalósítani egy pókerjátékot a választott technológiákkal.

A szakdolgozati program tervezéséhez, kivitelezéséhez több fejlesztői környezetet is használok. A MySQL Workbench 5.0-s verziójával az adatbázissémát terveztem meg, az Enterprises Architect 7.5-ös verziójával a use case, class, az activity és a component diagramokat készítettem el. A program kliens oldalát Adobe Flash Professional CS5 és Flash Develop programok segítségével valósítottam meg, míg a szerver oldalát Netbeans 6.9.1-es verziószámú fejlesztői környezettel kiviteleztem.

Fontosnak tartottam, hogy a program használjon adatbázistáblákat is, ezért lehetőség lesz regisztrációra, a játék mentésére, a játék betöltésére valamint az eredmények megtekintésére.

A póker története, szabályai, lapkombinációk

A póker rövid története

A játék eredetére vonatkozóan több teória is létezik. Nevét valószínűleg a francia „poque” nevű játék után kapta, amely alapjául egy „nas” elnevezésű perzsáktól átvett játék szolgálhatott. Ugyanakkor a póker szerkezete hasonlít a spanyol „primero” nevű játékra is. A póker tehát feltételezhetően több játék jegyeit is magába foglalja.
A játék bár Európából és Ázsiából származtatható mégis Amerikát hódította meg leginkább. Nevada államban, New Orleansban az 1800-as évek elején kezdett elterjedni a játék, amit ekkor még 20 lappal játszottak, a játékosok pedig 3 lapot kaptak. Az osztást követően a szereplők arra licitáltak, hogy kinél vannak a legerősebb lapok. Később bővült a póker 52 laposra valamint bevezették a sort és a flusht. A játék egyre nagyobb népszerűségre tett szert, amíg illegális szerencsejátéknak nem minősítették. 1936-ban ugyan legalizálták, de a kaszinótulajdonosok nem kedvelték, mert kevés bevételt hozott.
Az igazi áttörést az 1970-ben induló World Series of Poker nevű bajnokság megrendezése hozta. A játék ettől kezdve lett igazán közkedvelt és több változata is született.
A következő mérföldkövet az Internet jelentette, amellyel lehetővé vált az online játék. 1994-ben jött létre az első online kaszinó, négy évvel később pedig Planet Poker néven az első online pókerterem.
Mára már megszámlálhatatlanul sok pokersite szolgálja ki a játékosok igényeit.

A játékszabályok

Több pókerváltozat is létezik, ilyen például az Omaha, a Razz, a Seven Card Stud és a Texas Hold’em. Ezen utóbbival fogalakozom részletesen, mivel szakdolgozatom fő témája egy Pot Limit Texas Hold’em játék megtervezése és kivitelezése.

A Pot Limit Texas Hold'em póker

A játékot mindig egy pakli 52 lapos francia kártyával játsszák, ami négy színből áll. Treff (clubs), Káró (diamonds), Kör (hearts), Pikk (spades) és a lapok értéke Ász (Ace), Király (King), Dáma (Queen), Bubi (Jack), 10, 9, 8, 7, 6, 5, 4, 3 és 2.

A Pot Limit játék esetén a feltehető összeget a kezdő tétek nagysága, a körben az előző játékosok által feltett tétek nagysága és a játékos megmaradt zsetonjainak száma határozza meg.
Maximum tét: a maximum tét vagy emelés, amelyet egy játékos megtehet az a „pot" összegével megegyező tét.

A játék menete

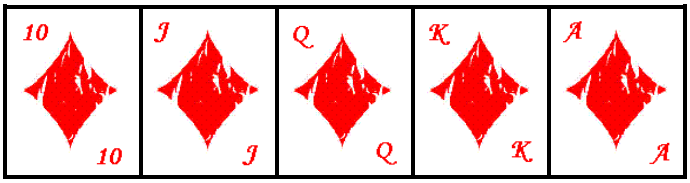
Minden játékos két lapot kap a kezébe, 5 lap pedig az asztal közepére kerül. A játékosok ebből a 7 lapból választják ki a számukra legjobb kombinációt adó 5 lapot.

Az osztás előtt az osztótól balra ülő játékos beteszi a kis vak licitet (small blind), a következő pedig a nagy vak licitet (big blind). Az osztó ezután mindenkinek két lapot oszt. Először a big blind után ülő játékos licitálhat, aki eldobhatja (fold) a lapját, megadhatja (call) a tétet (bet), illetve emelhet (raise) is. A következő játékosoknak ugyanezek a lehetőségei. Az első licitkör után az osztó három lapot tesz középre (flop). Az osztás után a small blind kezdi a licitálást, aki passzolhat (check), illetve tétet rakhat be. Miután valaki emelte a tétet, a játékostárs(ak) a következőket teheti(k): bedobja a lapját,tarthatja a tétet, emelhet.
A licitálás után az osztó leteszi a negyedik lapot (turn). Újra kezdődik a licitálás az előzőeknek megfelelően. A licit után jön az ötödik lap (river), amit egy újabb licitálás követ. A licitálások után a lapmutatás (showdown) következik. Az a játékos mutatja először a lapját, aki utoljára emelt. Az nyer, akinek a legerősebb lapkombinációja van. Döntetlen esetén el kell osztani az összegyűlt tételeket. A másik lehetőség, hogy a játékos addig emeli a tétet, hogy mindenki más eldobja a lapjait, így anélkül nyerhet, hogy meg sem mutatja a kártyáit.

A következő játék előtt az osztót jelölő buttont eggyel balra kell tenni, amihez igazítani kell a vakokat is.

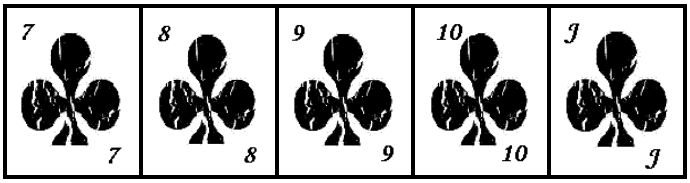
A leosztások értéke
A nyerő lapkombinációk a legerősebbtől a leggyengébbig rangsorolva.

royal flush



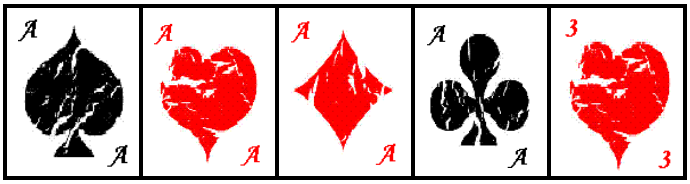
A legmagasabb leosztás a pókerben a Royal Flush: azonos színű A-K-Q-J-10.

straight flush



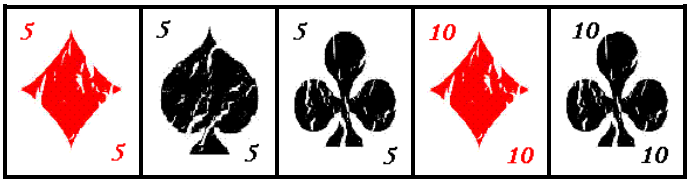
A színsor a második legmagasabb leosztás, mely öt egymást követő, azonos színű kártyalapból áll.

four of a kind (poker)



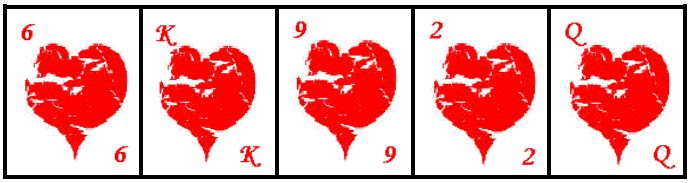
A póker a következő legmagasabb értékű leosztás. Négy azonos értékű kártyalapból és egy ötödik lapból áll, ami lehet bármilyen.

full house



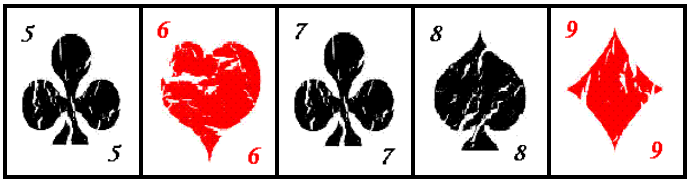
Rangsorban a full következik, mely három plusz két megegyező értékű kártyalapból áll.

flush



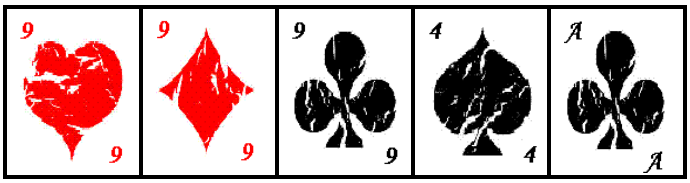
A flush azonos színű, de nem egymást követő öt kártyalapból áll.

straight



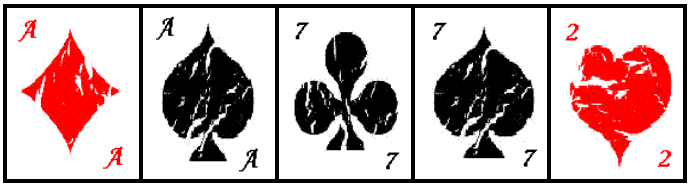
A sor öt tetszőleges színű, egymást követő lapból áll, melyben az ász lehet magas vagy alacsony értékű is. A sorhoz nem kell feltétlenül ász is, bármilyen öt egymást követő kártyalap elég. Minél nagyobb a legmagasabb lap, annál erősebb az értéke.

three of a kind (trips)



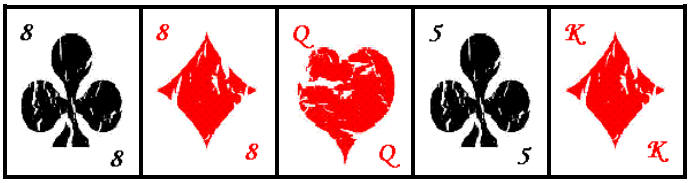
Három bármilyen azonos értékű kártyalap.

two pair



Két-két egyforma értékű kártyalapból, és egy ötödik kártyalapból áll, mely azokkal nem azonos értékű; a két pár közül a magasabb értékűvel hivatkoznak rá.

pair (one pair)



Két megegyező értékű kártyalap, plusz három másik kártyalap, melyek az előző kettővel együtt nem hoznak létre semmilyen nagyobb értékű leosztást. Minél magasabb lapokból áll, annál nagyobb az értéke is.

high card

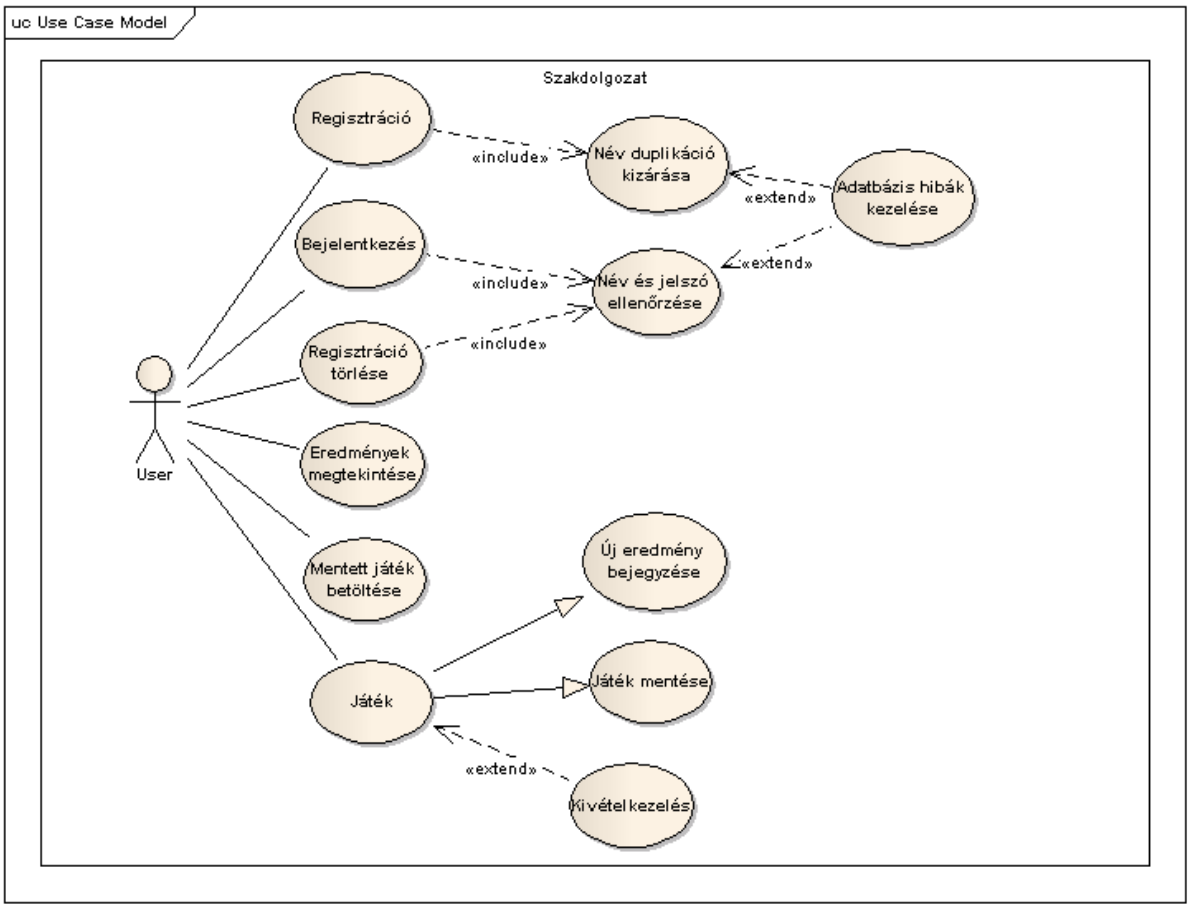
Ez a leggyengébb leosztás. A legmagasabb lap az ász, majd sorban a többi lap egészen a kettesig. Amikor két játékosnak megegyezik a legmagasabb lapja, akkor a soron következő dönt.

A programmal szembeni felhasználói követelmények

A programnak tudnia kell kezelni bizonyos felhasználói interakciókat:

- regisztráció
- belépés
- regisztráció törlése
- eredmények megtekintése
- elmentett játék betöltése
- játék indítása
- játék elmentése
- befejezett játék eredményének eltárolása

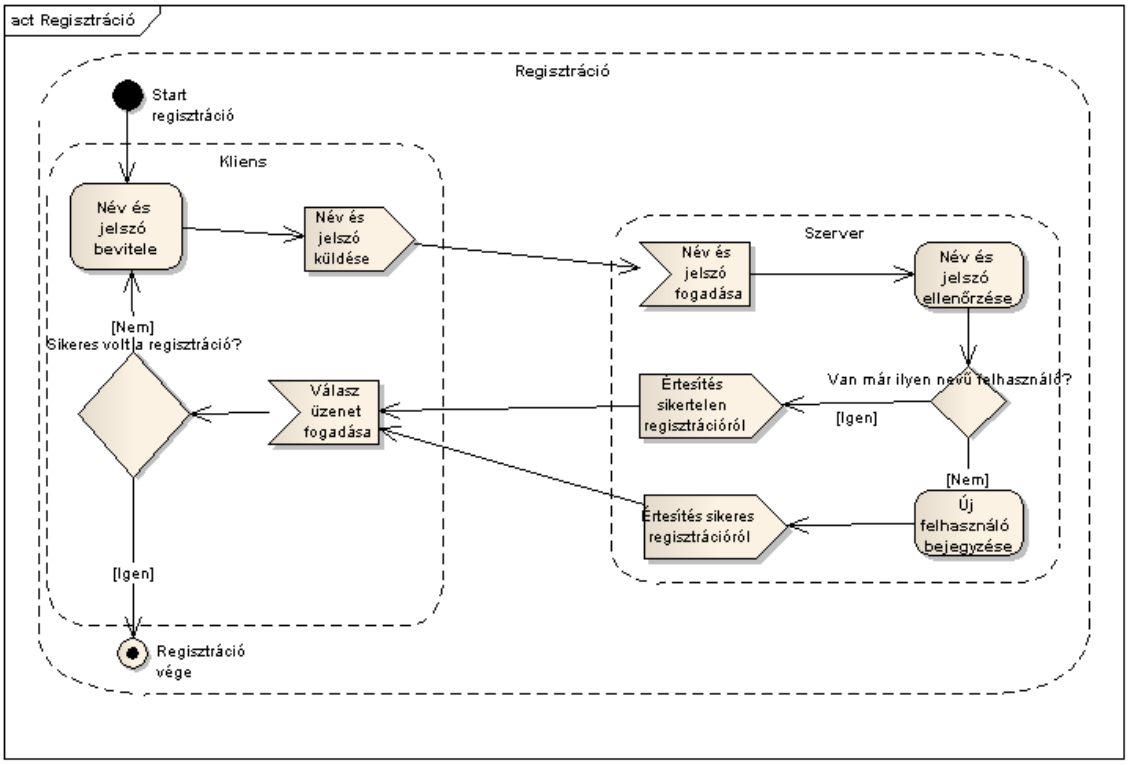
A programhoz tartozó use-case diagram:



1. sz. ábra

A felhasználó interakciói:

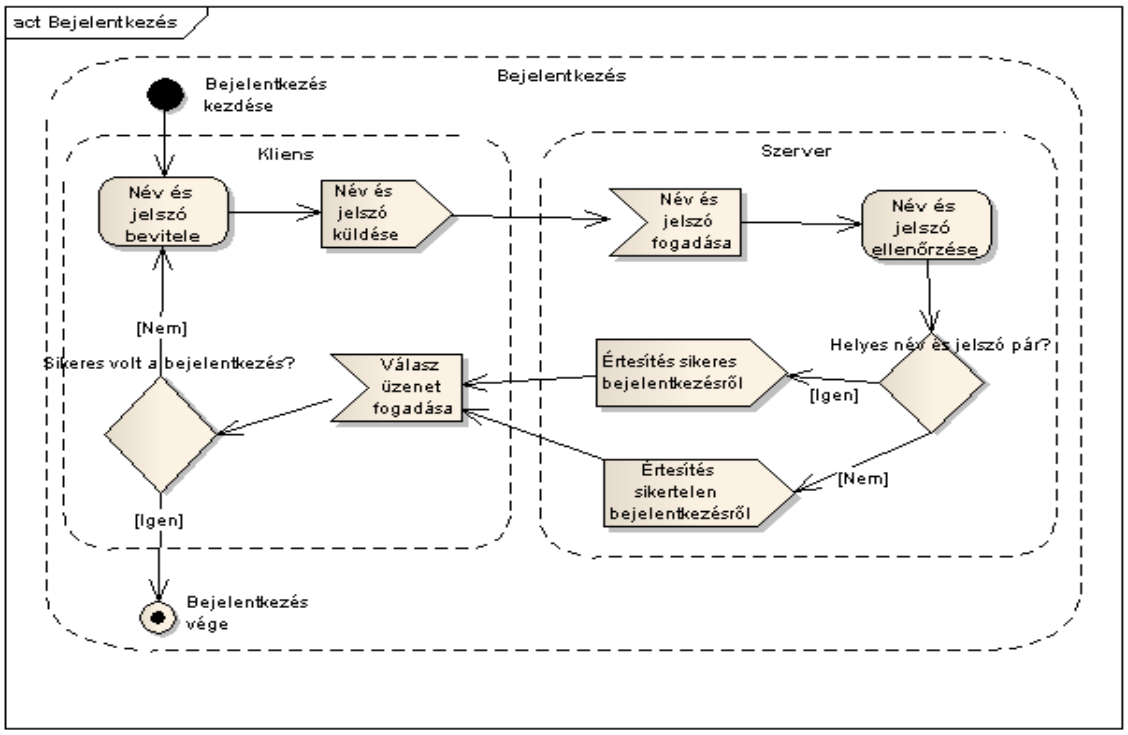
Regisztráció



2. sz. ábra

A kliens oldalon bevitelre kerül egy tetszőleges név és jelszó páros, melyet elküld a szervernek. A szerver oldalon végrehajtódik egy lekérdezés, hogy szerepel-e már az adott nevű felhasználó az adatbázisban. Amennyiben nem szerepel, úgy bekerül az adatbázisba a felhasználó az adott jelszóval. Ezt követően a szerver értesítést küld a kliensnek a sikeres regisztrációról, és egyúttal megtörténik a beléptetés is. Ha szerepel már adott nevű felhasználó, úgy a regisztráció sikertelennek mondható. Ekkor a kliens értesítést kap arról, hogy nem megfelelő felhasználónevet adott meg, ezután lehetősége lesz új név és jelszó páros megadására.

Belépés



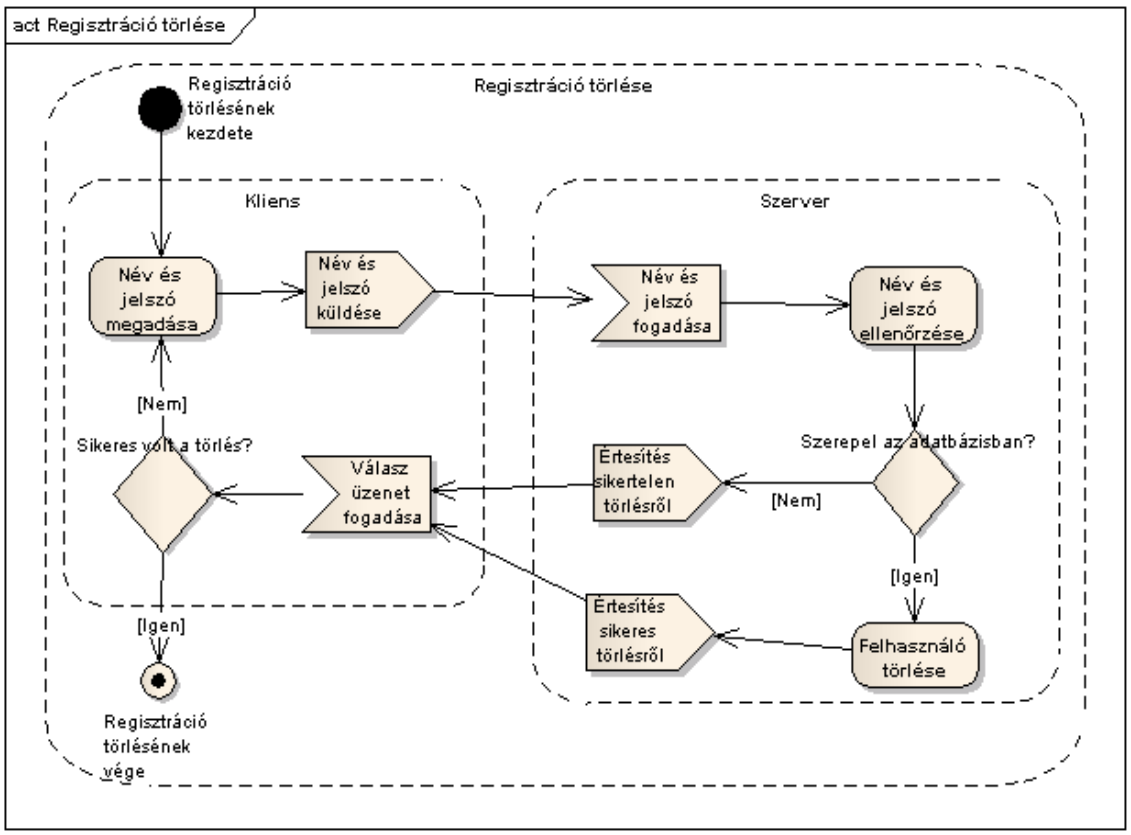
3. sz. ábra

A kliens oldalon bevitelre kerül egy név és jelszó páros, melyet elküld a szervernek. A szerver oldalon ez a név és jelszó ellenőrzésre kerül. Egy lekérdezés hajtódik végre, mely megvizsgálja, hogy szerepel-e adott nevű felhasználó az adatbázisban az adott jelszóval. Amennyiben a van ilyen felhasználó, úgy a belépés sikeresnek tekinthető, és erről egy üzenetet küld a szerver a kliensnek. Ha a lekérdezés azt eredményezi, hogy nem szerepel az adott nevű felhasználó az adott jelszóval az adatbázisban, úgy a belépés sikertelen, és erről a szerver értesíti a klienst, ezután lehetőség lesz újbóli név és jelszó páros megadásának.

Regisztráció törlése

A kliens oldalon bevitelre kerül egy név és jelszó páros, melyet átküld a szervernek. A szerver ezután végrehajt egy lekérdezést, mellyel ellenőrzi, hogy szerepel-e az adott nevű felhasználó az adatbázisban, és ha szerepel, akkor a hozzá tartozó jelszó megegyezik-e azzal a jelszóval, mely kliens oldalon bevitelre került. Amennyiben a lekérdezés azt eredményezi, hogy korrekt név és jelszó páros lett megadva, úgy az adott nevű felhasználót törli az adatbázisból, és értesítést küld a kliensnek a sikeres törlésről. A törlés végleges, és

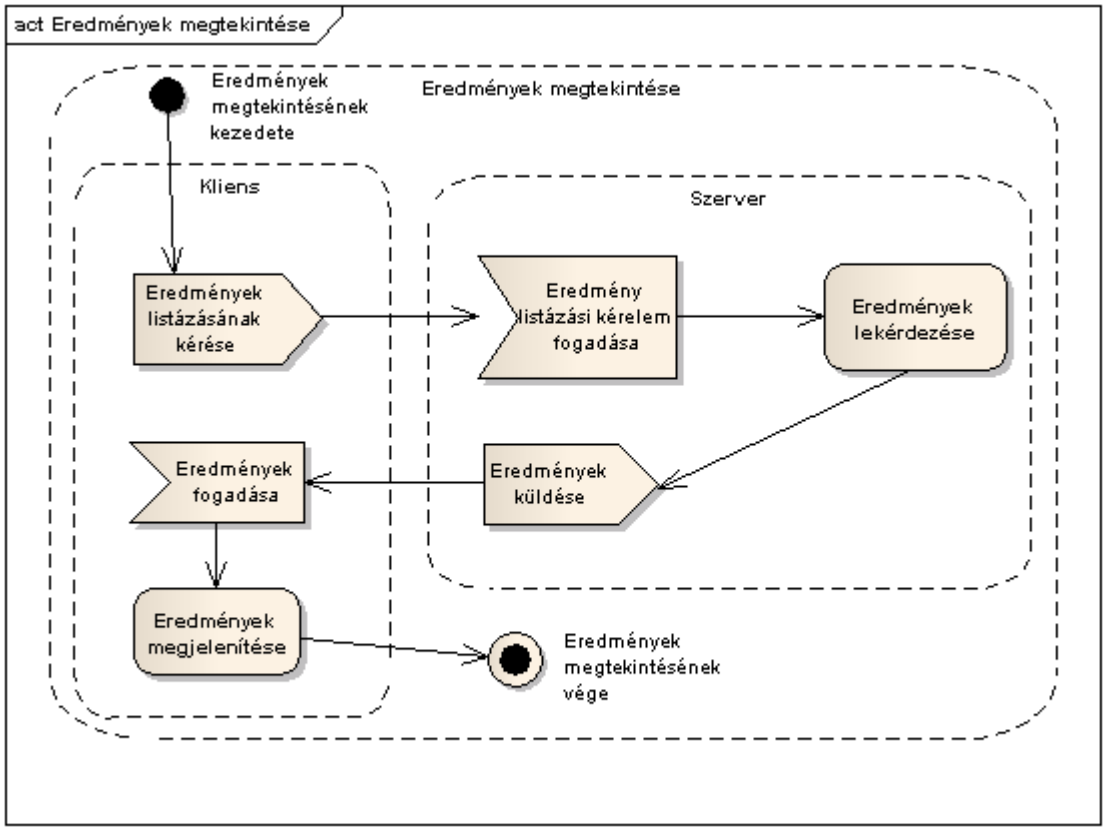
nem visszavonható. Amennyiben helytelen név és jelszó páros lett megadva, úgy a semmilyen törlés nem megy végbe, ezáltal a regisztráció törlése sikertelennek tekinthető, melyről a szerver üzenetet küld a kliensnek. Ekkor lehetőség lesz újból megpróbálni a regisztráció törlését.



4. sz. ábra

Eredmények megtekintése

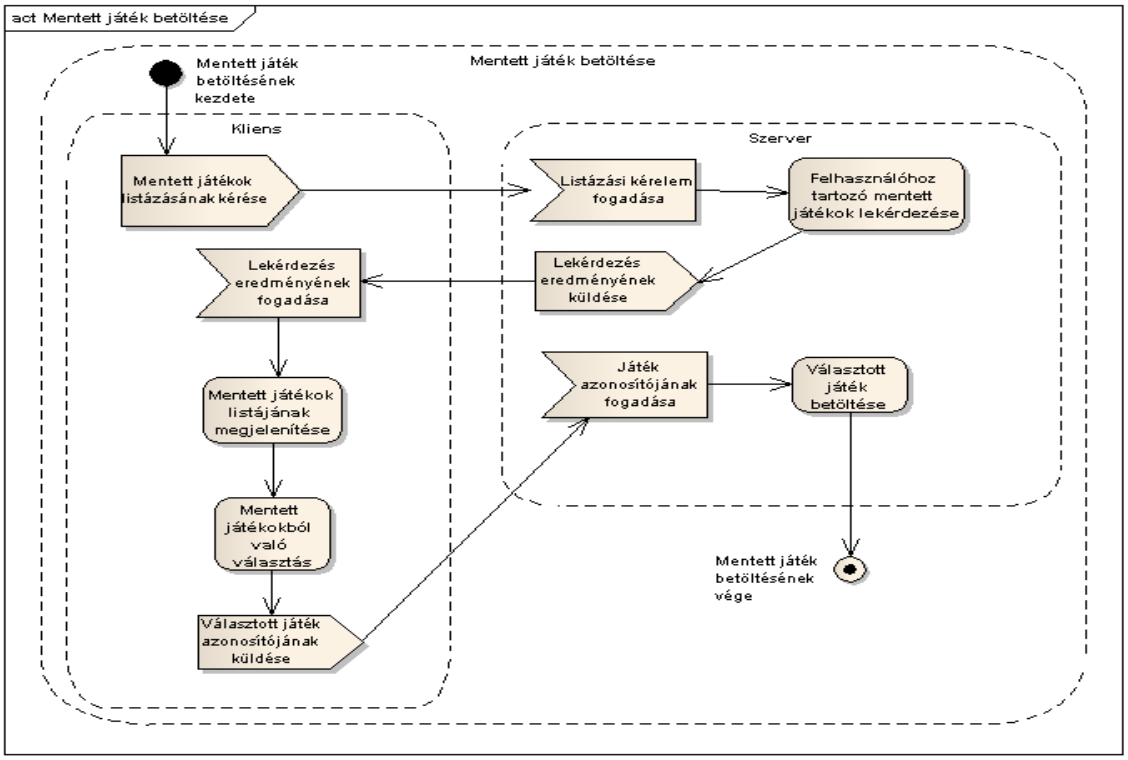
A kliens oldalon a felhasználó generál egy üzenetet, mely azt jelzi, hogy meg kívánja tekinteni az eddigi eredményeket. Ezt az üzenetet a kliens elküldi a szervernek. Mikor a szerverhez beérkezik ez az üzenet végrehajtódik egy lekérdezés az eredmények táblából, és ennek a lekérdezésnek az eredményét visszaküldi a kliensnek. A kliens dolga ezután megjeleníteni a beérkezett eredményeket.



5. sz. ábra

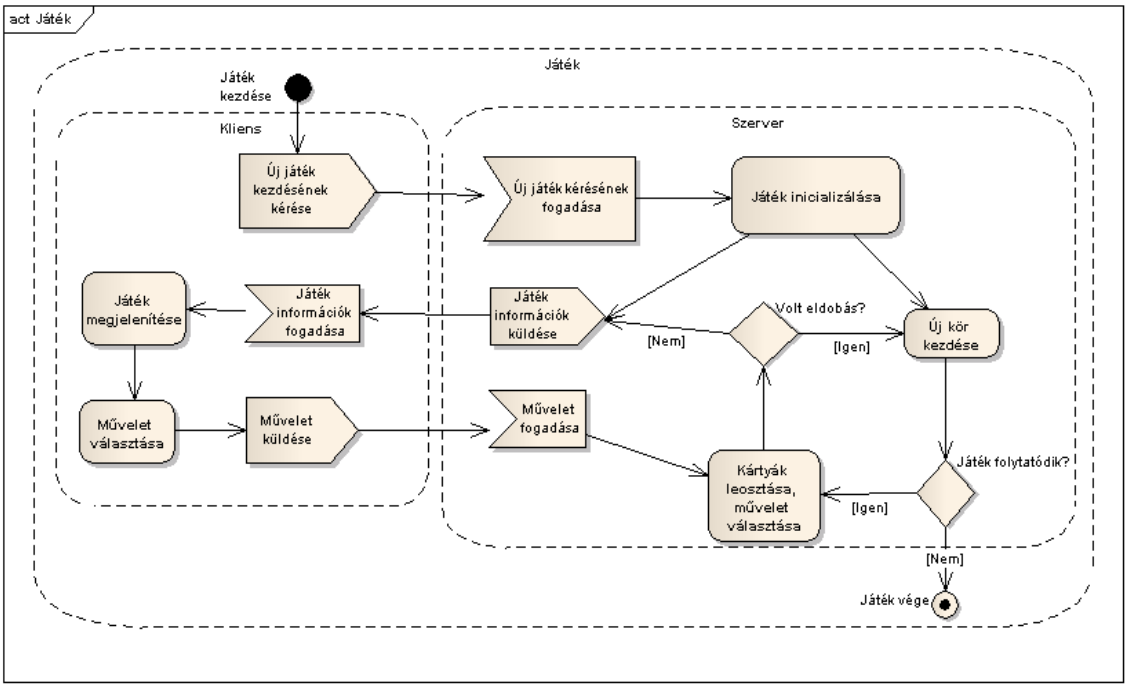
Játék betöltése

Kliens elküld egy üzenetet, melyben a felhasználó kéri a hozzá tartozó mentett játékok listáját. A szerver feldolgozza ezt az üzenet, végrehajt egy lekérdezést, mellyel lekérdezi az adott nevű játékos összes eddigi elmentett játékát. A lekérdezés eredményét pedig továbbítja a kliensnek. Ezután kliens oldalon megjelenik a felhasználóhoz tartozó mentett játékok listája. Amennyiben ez a lista nem üres, úgy a játékos választ egyet az általa elmentett játékok listájából, majd a kliens átküldi a szervernek a választott játék azonosítóját. Miután a szerver megkapta a kiválasztott azonosítót, betölti az adott játékot, és a játékos folytathatja tovább az általa korábban elmentett játékot.



6. sz. ábra

Új játék indítása

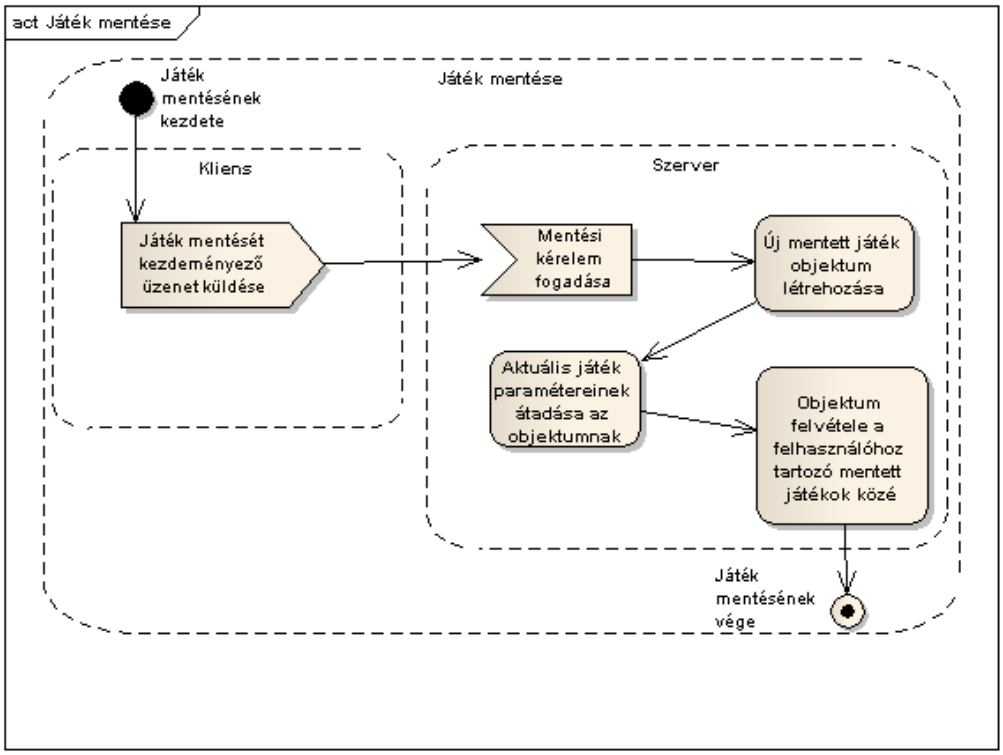


7. sz. ábra

A kliens oldalon a felhasználó generál egy üzenetet, melyben jelzi, hogy egy új játékot kíván elindítani. Ezt az üzenetet a kliens továbbítja a szervernek. Ekkor szerver oldalon megtörténik a játék inicializálása. A szerver üzenetet küld a kliensnek a játék állapotáról, melyet a kliens fogadás után megjelenít. Ezután szerver oldalon elkezdődik a játék egy új köre. Ellenőrzi, hogy folytatódik-e a játék, mely azt jelenti, hogy a játékosok zsetonjainak ellenőrzése következik. Ha mindkét játékosnak van elég zsetonja, akkor a játék állapotának megfelelő művelet következik (kártyák leosztása, operátor választása). Miután ez végbement, a szerver újból üzenetet küld a kliensnek a megváltozott játékállapotról. Ezt az állapotot a kliens megjeleníti, majd felhasználói interakció következik. A felhasználó által választott műveletet továbbítja a kliens a szerver felé. Ez addig folytatódik, amíg a játék véget nem ér. A játéknak akkor van vége, ha valamelyik játékosnak nem maradt elég zsetonja ahhoz, hogy a vaktéteket megadják.

Játék elmentése

A felhasználónak lehetősége van a játék elmentésére.

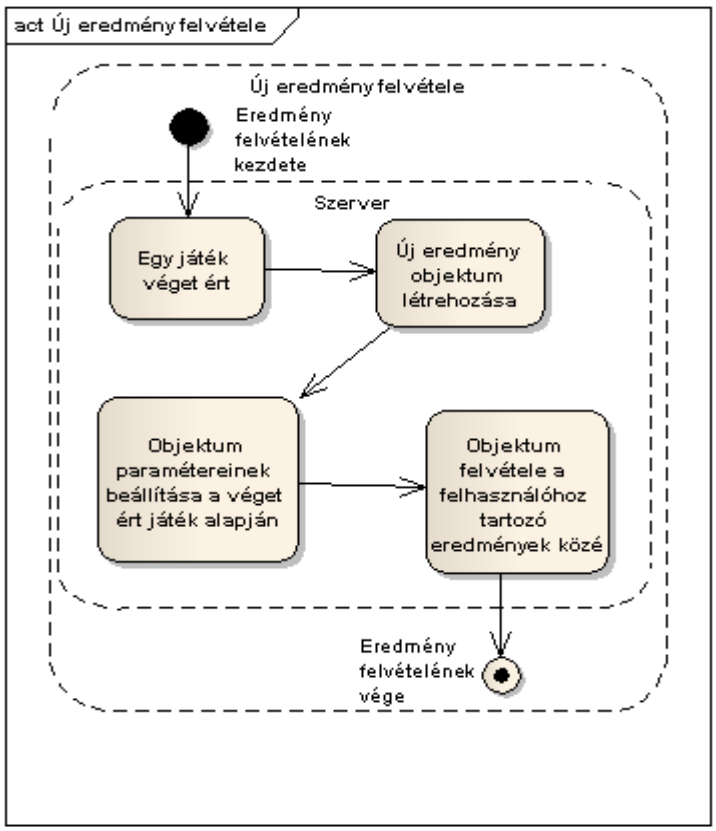


8. sz. ábra

A felhasználó kezdeményezi az aktuális játék elmentését. Szerver oldalon létrejön egy új mentett játék objektum, melynek paraméterei az aktuális játék paraméterei alapján állítódnak be. Ezután az objektum bekerül a mentett játékok táblájába.

Egy új eredmény bejegyzése az eredmény táblába

Új eredmény akkor kerülhet a táblába, ha egy játék véget ért.

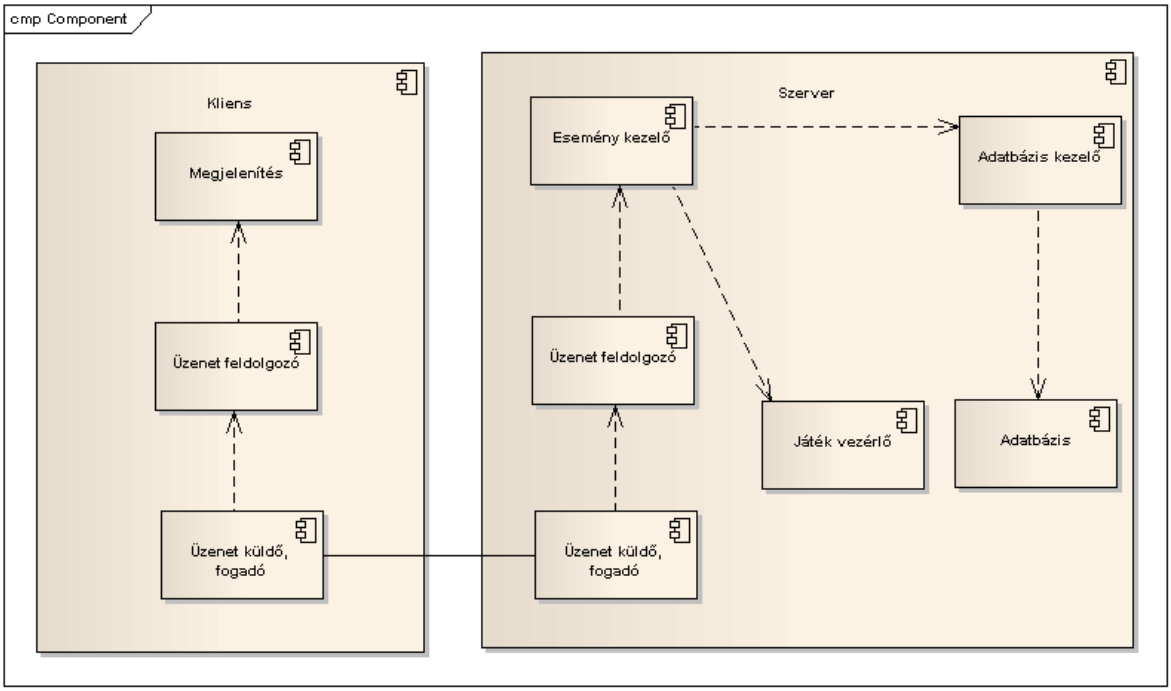


9. sz. ábra

Ez a folyamat kizárólag szerver oldalon - függetlenül a klienstől- zajlik. Amikor vége lesz egy játéknak egy új eredmény objektum jön létre, melynek paramétereit a véget ért játék paraméterei alapján határozza meg a szerver. Miután ez megtörtént, az új eredmény objektum bekerül az eredmények táblába.

A program elméleti felépítése

A programhoz tartozó komponens diagram:



10. sz. ábra

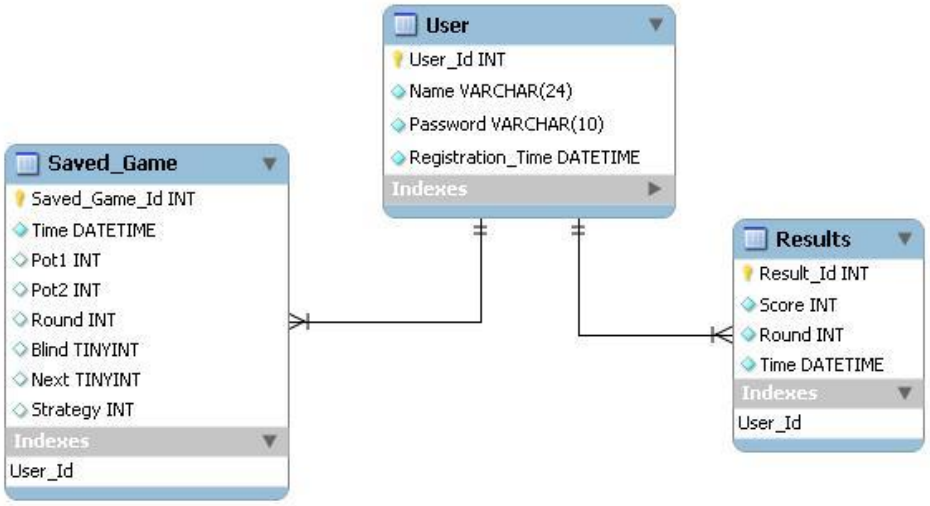
A program két fő részből épül fel, kliens illetve szerver oldalakból. A program szerver oldala tartalmaz üzenet küldését és fogadását megvalósító komponenst. Ez a komponens felel a kapcsolat biztosításáért, az üzenetek helyes elküldésért és fogadásáért. Az üzenet feldolgozó komponens a beérkezett, illetve az elküldendő üzenetek helyes átalakítását valósítja meg. Az eseménykezelő felelős a program használata közben előforduló események helyes kezeléséért, valamint a vezérlés kiosztásáért. A vezérlést a játék vezérlőnek adja át, vagy pedig az adatbázis kezelőnek. A játékvezérlő kezeli a felhasználóhoz tartozó új játék indítása és mentett játék folytatása interakciókat. Az adatbázis kezelő feladata a regisztráció, belépés, regisztráció törlése, eredmények megtekintése, új eredmény felvétele, azaz a felhasználói interakciók megvalósítása, valamint a felhasználóhoz tartozó elmentett játékok listájának elkészítése. Az adatbázis kezelő a műveletek végrehajtásához a program mögött lévő adatbázist használja. Az eseménykezelő hiba esetén válasz üzenetet is generálhat, ha a vezérlést nem tudta továbbadni.

A játékvezérlő a játék során fellépő interakciókat is kezeli, valamint játékkal kapcsolatos üzeneteket fogalmaz meg, amit átad az üzenet feldolgozónak.
Az adatbázis kezelő felelős az adatbázis műveletek végrehajtásából adódó válasz üzenetek megfogalmazásáért.
A program kliens oldalának feladata kapcsolódni a szerverhez, valamint a szervertől érkező üzenetek helyes feldolgozása, és megjelenítése. A kliens feladatai közé tartozik továbbá a szerver irányába küldendő üzenetek megfogalmazása, azok helyes átalakítása, majd elküldése, és biztosítani a felületet a felhasználói interakciókhoz.

Szakdolgozati program gyakorlati megvalósítása

A program mögött lévő adatbázis

Az adatbázis sémát MySQL Workbench 5.0 OSS programmal készítettem el.



11. sz. ábra

Az adatbázis viszonylag egyszerű, három táblából áll, melyek a programot futtató felhasználó különböző adatait tárolja.

User tábla

A felhasználó személyes adatait tartalmazza.
User_Id: egy egész értékű mező, amely a felhasználó azonosítója, illetve egy rekord elsődleges kulcsa, ami magában foglalja azt, hogy két azonos értékű azonosító nem szerepelhet a táblában. Regisztrációkor automatikusan generálódik a felhasználóhoz. *Name:* egy karakter sorozat, amely a felhasználó nevét tartalmazza. Jellemzője, hogy tetszőlegesen választható és egyedi valamint a név nem lehet „Fail” mivel, az egy lefoglalt kulcsszó a program számára. Maximum 24 karakter hosszúságú. Egy név csak egyszer szerepelhet a táblában, ugyanis a név duplikációt a program kizárja.

Password: egy olyan karakter sorozat, amely a felhasználóhoz tartozó tetszőleges jelszót tartalmazza. Terjedelme maximum 10 karakter lehet és megadása kötelező. *Registration_Time:* az a dátum, amely a regisztráció időpontját jelenti. A program regisztrációkor automatikusan hozzárendeli a felhasználóhoz, ezért külön megadása nem szükséges.

Result tábla

A felhasználó által korábban végigjátszott játékok eredményeit tartalmazó tábla. *Result_Id:* olyan egész értékű mező, amely az eredmények azonosítja és a rekordok elsődleges kulcsa, ez kizárja, hogy két rekordhoz is ugyanaz az azonosító tartozzon, továbbá automatikusan rendelődik egy érték rekordhoz.
Score : egész értékű, a játékos zsetonjainak számát mutatja meg a játék befejezése után.
Round: egész értékű, amely a körök számát jelenti. Számon tartja, hogy mennyi kör kellett ahhoz, hogy a játék befejeződjön.
Time: az az időpont, amikor a játék véget ért.

Saved_Game tábla

A játékosnak lehetősége van elmenteni a játékot, a Saved_Game tábla az elmentett játékok adatait tartalmazza.
Saved_Game_Id: egész értékű, az elmentett játék azonosítóját jelenti, valamint a táblában szereplő rekordok elsődleges kulcsa. Két rekordnak nem lehet azonos elsődleges kulcsa. Automatikusan rendelődik egy új rekordhoz.
Time: dátum, az elmentés időpontját jelenti.
P*ot1*: egész érték, az aktuális játékban mutatja meg a felhasználó zsetonjainak számát a játék elmentésének idejében.
Pot2: egész érték, a gépi játékos zsetonjainak számát jeleníti meg az elmentett játékban, a mentés időpontjában.
Round: egész értékű és azt jelzi, hogy a játék hányadik körében lett elmentve.
Blind: egész értékű, mely azt jelzi, hogy melyik játékos a nagyvak.
Next: egész értékű. Ennek értéke határozza meg azt, hogy melyik játékos következett volna az adott játékban az elmentés idejében.
Strategy: egész érték, a gépi játékos aktuális körbéli stratégiáját tárolja.

A táblák között lévő kapcsolatok

A központi tábla a User tábla, és ehhez kapcsolódik a Result és Saved_Game tábla.

User és Result táblák közötti kapcsolat:

A User és a Result táblák között egy a többhöz kapcsolat (one to many) van, amely azt jelenti, hogy egy felhasználóhoz több eredmény is tartozhat, ugyanakkor egy eredményhez kizárólag egyetlen felhasználó tartozhat. A kapcsolat a két elsődleges kulcs segítségével jön létre.

User és Saved_Game táblák közötti kapcsolat:

A User és Saved_Game táblák között szintén egy a többhöz kapcsolat (one to many) van. A kapcsolat típusa szerint egy adott felhasználóhoz több elmentett játék tartozhat, míg fordított esetben, egy elmentett játékhoz szigorúan egy felhasználó tartozhat. A kapcsolat az elsődleges kulcsok alapján jön létre.

A program kliens oldalának megvalósítása

A kliens oldali program egy flash project, melyet Adobe Flash Professional CS5, és Flash Develop fejlelsztői környezetekkel valósítottam meg.

A kliens feladatai:

- Egy socket-en keresztül kapcsolat kiépítése a szerverrel. A kliens a 3456-os számú porton keresztül kommunikál a szerverrel a „localhost”-on.
- A szerver által meghatározott funkciók grafikus megjelenítése.
- Üzenetek küldése a szerver felé.
- A szervertől érkezett üzenetek fogadása, és helyes feldolgozása.

A fent említett funkciók megvalósításához végül öt saját „ .as ” osztályt, valamint öt „ .fla ” és ezekhez tartozó öt „ .swf ” fájlt hoztam létre.

Main osztály

Privát tagjai:

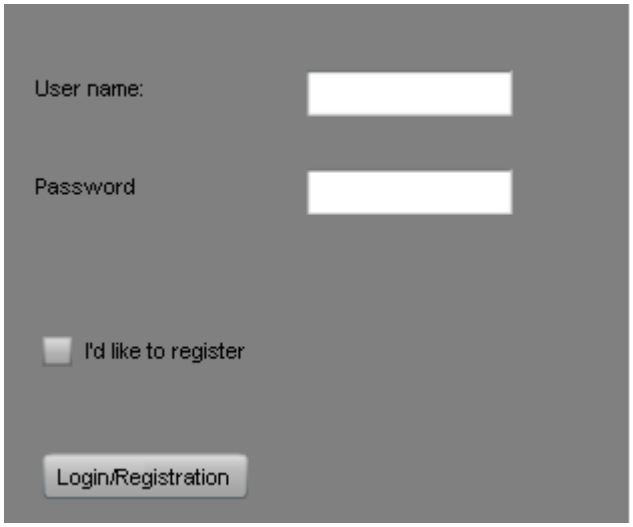
- clientSocket, amely egy xmlSocket
- loginhandler, amely egy LoginHandler

A Main osztály egy MovieClip, ezt kihasználva jeleníti meg a kliens program a különböző

felületeket, oly módon, hogy erre a MovieClip-re tölti be a „ .swf ” fájlokat, tehát amíg fut a program, az összes felület a Main által jelenik meg.
A Main osztály feladata a már várakozó szerverhez történő kapcsolódás a clientSocket elnevezésű xmlSocket–tel. Amennyiben a kapcsolat létrejött, a vezérlés és a clientSocket továbbadódik egy LoginHandler típusú objektumnak. A Main objektum önmagát is továbbadja, mivel a Main osztály egy kiterjesztett MovieClip, amin a megjelenítés történik. A Main osztálynak további feladata nincs.

LoginHandler osztály

Az osztály feladata biztosítani a felhasználó belépésének, vagy regisztrációjának grafikus felületet és ennek eseménykezelését. Ezt a felületet a Login.swf importálásával biztosítja, amely így néz ki:



12. sz. ábra

A program ezen szakaszából történő továbblépéshez szükségszerű regisztrálni, vagy belépni. Az első beviteli mezőbe kerül a felhasználónév, alá pedig a jelszó. A jelszó nem jelenik meg explicite, a felhasználó csak egy csillagokból álló karakter sorozatot lát. Ezek alatt található egy úgynevezett CheckBox, melyre rákattintva a regisztrációs szándékot lehet jelezni. A panel bal alsó sarkában egy Login/Registration gomb található. Ha a felhasználó már korábban regisztrált, akkor itt lehet belépni a játékba, ha még ezt nem tette meg, akkor ennek a gombnak a segítségével indíthatja el a regisztrációt. Ha a felhasználó klikkelt aLogin/Registration gombra, és a regisztrációjának megfelelően töltötte ki a

mezőket (nem maradtak üresen a mezők, és a felhasználónév nem „Fail”), úgy a kliens üzenetet küld a szervernek. Az üzenet Json formátumú lesz, - mint minden kliens és szerver üzenet - melynek tagjai:

- messageName: amely vagy login, vagy registration. A szerver ezt a tagot fogja vizsgálni annak megállapítására, hogy a felhasználó milyen műveletet szeretne végrehajtani.
- userName: a felhasználó által bevitt név
- password: a névhez tartozó jelszó

Az üzenetet elküldése után a kliens várakozni fog egy válasz üzenetre a szervertől, amelyben jelzi, hogy a művelet sikeres volt vagy sem.

A válasz üzenet felépítése:

- userName: sikeres művelet esetén a megadott név, sikertelen művelet esetén „Fail”
-succes: Egy olyan logikai érték, amely igaz érték esetén azt jelenti, hogy a belépés/regisztráció sikeres volt. Hamis érték esetén a művelet sikertelennek tekinthető és a felhasználónak újra meg kell adnia a nevét és a jelszavát.

Egy LoginHandler objektum addig marad aktív, amíg a belépésre vagy regisztrációra kapott válasz üzenet azt nem jelzi, hogy az adott felhasználói művelet sikeres volt. Amennyiben sikeres volt a LoginHandler objektum eltávolítja a MovieClip-ről a Login.swf fájlt, és a vezérlést továbbadja egy WelcomeHandler típusú objektumnak. A vezérlés mellett továbbadja a sikeresen belépett felhasználó nevét, valamint a clientSocket nevű xmlSocket változót.

WelcomeHandler osztály

Feladata kezelőfelületet biztosítani különböző felhasználói interakciók számára. A felület grafikus megjelenését a Welcome.swf fájl biztosítja.

Welcome.swf:

13. sz. ábra

Legfelül található egy üzenet, mely köszönti a sikeresen belépett felhasználót.
Fentről lefelé haladva sorban a következő gombok találhatóak:

- új játék indítása
- játék betöltése
- eredmények listázása
- regisztráció törlése
- kilépés

Minden gomb lenyomása egy eseményt generál. Az esemény lehet egy PlayGameHandler, DeleteRagistrationHandler objektum hívása, vagy egy üzenet generálása, amit a kliens elküld a szervernek. Az elküldött üzenetek kezelését a szerver végzi.
A New game gomb lenyomásával a felhasználó új játékot kezdhet. Ekkor egy új PlayGameHandler objektum jön létre.
A Load game gomb lenyomásával a kliens először üzenetet küld a szervernek, amelyben kéri az aktuális felhasználó elmentett játékainak listáját. Miután megkapta a választ, megjeleníti a játékok listáját a gomboktól jobbra lévő területen, amiből választhat a játékos. Ha volt megjeleníthető játék, a felhasználó a lista alatt található Load gomb megnyomásával választhat belőle, így egy új PlayGameHandler objektum létrehozásával folytatódhat a korábban elmentett játék. Ezalatt a New game, Load game, List scores,

Delete registration és Exit gombok inaktívak lesznek mindaddig, amíg a felhasználó le nem nyomja a lista alatt található Cancel gombot.

A List score gomb lenyomása egy listázási kérelmet generál, amelyben a kliens kéri az eredmények listázását. A szerver erre egy válaszlistát küld, amely a legjobb addigi lejátszott játékok eredményeit tartalmazza. A lista egy TextArea-ban jelenik meg, a gombok mellett jobbra. A listázást az Ok gombbal lehet jóváhagyni, a játék többi funkciója eközben inaktív.

A Delete registration gomb lenyomásával a felhasználónak lehetősége lesz a regisztrációját törölni. Ehhez először létrejön egy DeleteRegistrationHandler objektum, amely kezelőfelületet biztosít ehhez a művelethez. Ez az új objektum megkapja a vezérlést, és a clientSocket-et.

Az Exit gomb lenyomásával a felhasználó kilép a programból. A gomb lenyomása a szerver és a kliens közötti kapcsolat megszűnését eredményezi.

A DeleteRegistrationHandler osztály

Az osztály feladata kezelőfelületet biztosítani a felhasználó regisztrációjának törléséhez. A kezelőfelület megjelenítését a DeleteRegistration.swf fájl biztosítja:

14. sz. ábra

Az első beviteli mezőbe kerül a felhasználó neve, az alatta lévőbe pedig a felhasználó jelszava. A Delete gomb lenyomásával kezdeményezhető a regisztráció megszüntetése. A felhasználó csak a saját regisztrációját törölheti, mivel csak akkor tudja magát törölni a

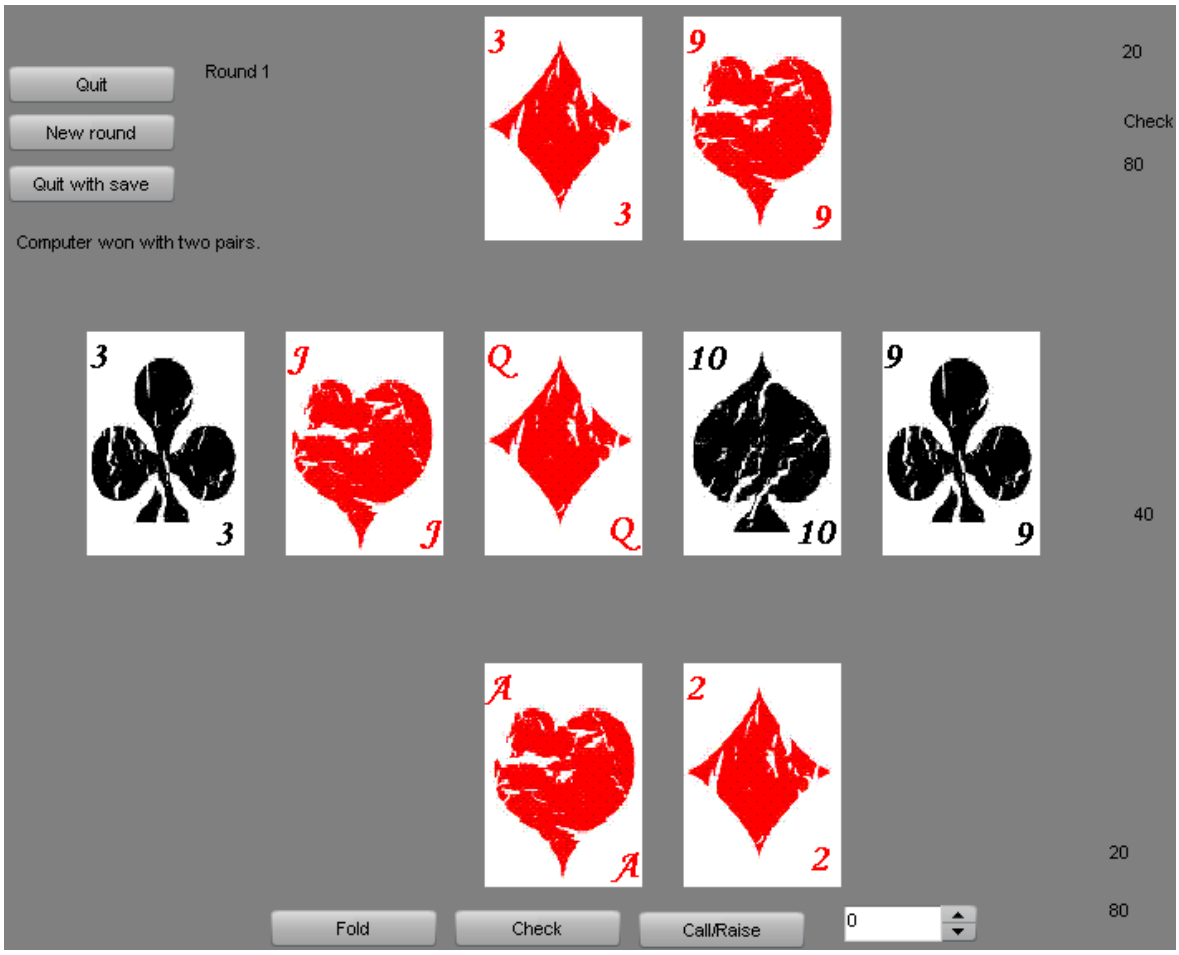
játékosok közül, ha előzőleg sikeresen bejelentkezett a játékba. Ugyanis bejelentkezéskor a kliens eltárolja a belépett felhasználó nevét és ezt a nevet hasonlítja össze a bevitt névvel. Ha a két név nem egyezik, akkor azt hibafelirattal jelzi a kliens. Amennyiben a két név azonos, (és a jelszó beviteli mező nem üres) a kliens üzenetet küld a szerver felé a felhasználó törléséről. Ennek az üzenetnek a felépítése:

messageName
userName
password

A messageName: tartalma „delete”, ezt vizsgálva fogja a szerver tudni, hogy regisztráció törlése következik.
A userName: a felhasználó neve.
A password: a felhasználó jelszava.
Ezek értékét a beviteli mezők értéke határozza meg.
Az üzenet elküldése után a kliens válaszra fog várakozni. A válasz üzenet tartalmazni fogja a művelet eredményességére vonatkozó információkat. Ha a művelet sikeres volt, egy új LoginHandler objektum generálódik, amely megkapja a vezérlést.
Ha a művelet sikertelen volt, azt a kliens egy hibaüzenettel jelzi a felhasználónak. Ekkor lehetőség lesz a név és jelszó páros újbóli megadására.
Ha a felhasználó meggondolja magát, a Cancel gomb lenyomásával visszatérhet az előző képernyőképre.

PlayGameHandler osztály

A PlayGameHandler osztály biztosítja a kezelőfelületet az új, vagy betöltött játék kezeléséhez. A felületet a Game.swf fájl biztosítja, és a következőképpen néz ki:



15. sz. ábra

Ez a kép a játék egy körének végén lévő állapotot mutatja be. Ennél a pillanatnál a Quit, New round és a Quit with save gombok aktívak. Lenyomásuk üzeneteket generálnak, melyeket a kliens továbbít a szervernek. A Quit gomb létrehoz egy olyan üzenetet, amelyben a kliens a játék végét jelzi. Ekkor szerver oldalon bizonyos beállító metódusok hajtódnak végre (lásd lent a szerver oldal működésénél). Miután elküldte az üzenetet, egy WelcomeHandler objektum kapja meg a vezérlést, a képernyőkép is ennek megfelelően változik.

A Quit with save gombbal lehet az aktuális játék elmentését kérni. Ha a szerver ilyen üzenetet kap az aktuális játék paraméterei eltárolódnak, hogy később lehessen folytatni a játékot. Ezt követően a vezérlés újra egy WelcomeHandler objektumé lesz és a képernyő is ennek megfelelően változik. A New round gombbal a játékos továbblép a következő körre.

A Quit gomb mellett található felirat az aktuális kör számát jelzi, ezt az információt a kliens a szervertől kapja.

A panel jobb oldalán található feliratok fentről lefelé haladva a következőket jelölik:

- a gépi játékos betett tétje,
- a gépi játékos utolsó művelete (Fold, Check, Call, Raise, All in),
- a gépi játékos még felhasználható zsetonjainak száma,
- az elnyerhető tét összege,
- a felhasználó bent lévő tétjének összege,
- a játékos felhasználható zsetonjainak száma.

Az ablak alsó harmadában lévő két kártya alatt található gombok a felhasználó által választható műveleteket jelölik. Ezen gombok csak akkor aktívak, ha az aktuális körben a felhasználó következik.

Fold gomb:

Lenyomásával a felhasználó eldobja lapjait. Ha a gép és a játékos tétjei egyenlőek, akkor a játék nem engedi eldobni a játékos lapjait (a „Don't fold" felirat jelenik meg a gomb mellett balra), mivel a játékosnak nem kell további zsetont felhasználnia arra, hogy a kör folytatódjon. Ha a tétek nagysága különbözik a gomb lenyomásával az adott kört a gép nyeri.

Check gomb:

Ha a két bent lévő tét nagysága egyenlő, akkor a gomb lenyomásának hatására az adott kör folytatódik. Abban az esetben ha a tétek nem egyformák a „You can't check" figyelmeztető felirat jelenik meg, tehát a felhasználónak más műveletet kell választania.

Call/Raise gomb:

A gomb lenyomásával tarthatja vagy emelheti a tétet a felhasználó. A tartás/emelés értékét a gomb mellett található, úgynevezett Stepper segítségével lehet megadni. A Stepper alapértelmezett értékeként az az összeg jelenik meg, amelyet legalább meg kell adnia a felhasználónak. A maximum értéke pedig az adott körben maximális tét lehet. Ez a tét a gépi játékos és felhasználó betett zsetonjainak különbségéből és a már bent lévő tétből áll. Ezen gombok lenyomása egy operátor üzenetet generál, amelynek felépítése:

userName
messageName
opName
pot

A userName a felhasználó neve lesz, a messageName pedig „operator”. Az opName jelzi, hogy milyen operátort választott a felhasználó (0-Fold, 1-Check, 3-Call, 4-Raise). A pot pedig a játékos betett tétjeihez hozzáadandó zsetonszámot jelenti.

PlayGameHandler osztály működése:

Ha egy PlayGameHandler objektum létrejön, akkor megkapja paraméterként a clientSocket-et, amelyen keresztül kommunikálhat a szerverrel. Egy movieClip-en keresztül jeleníti meg és megkapja a belépett felhasználó nevét. Ezek után betölti a Game.swf fájlt, majd ellenőrzi, hogy maradt-e kártya a movieClip-en. Ha maradt, akkor letörli azokat és egy új játék indítására vonatkozó üzenetet küld a szervernek, végül felkészül a szervertől érkező üzenetek fogadására. Az üzenetek típusát a counter privát tag értéke jelzi.

Counter értékei:
0: információs üzenet következik
1: kártya megjelenítésére vonatkozó üzenet
2: tétek értékeire vonatkozó üzenet
3: gépi játékos operátor üzenete
4: információ arról, hogy, ki milyen eredménnyel zárta a kört

Az információs üzenet felépítése:
- *sequence:* amely egy logikai érték. Ha hamis, akkor a játék véget ér és a vezérlést egy WelcomeHandler objektum kapja meg, a képernyő is ez alapján módosul. Amennyiben igaz, akkor a játéknak még nincs vége.
- *next:* egy logikai érték. Igaz esetén a felhasználó fogja kezdeni a kört, ha hamis, akkor pedig a gép kezd.
- *gameRound:* a játék körének száma.
- *inRound:* a belső kör értéke (1-Flop előtt, 2-Flop után, 3-Turn után, 4-River után). Ha a sequence tag igaz értékű, akkor a counter tag értéke 1-re állítódik, tehát a következő üzenet kártya üzenet lesz.

Kártya üzenet:

Az összes megjelenítendő kártyáról tartalmaz információt, azt osztályozza, hogy kié a kártya (felhasználó,gép,közös kártya). Ezek alapján jeleníti meg a megfelelő helyen a kliens a kártyákat. Ha nem történt „all in”, akkor tétek értékeire vonatkozó üzenet következik, ha volt „all in”, akkor továbbra is kártya üzenetek jönnek, végül az eredmény üzenet.

Tétek értékeire vonatkozó üzenet:

A felhasználó és a gépi játékos zsetonjaira vonatkozó üzenet.

Az üzenet tagjai :

- cPot: a gépi játékos még felhasználható zsetonjainak száma,
- inCPot: a gépi játékos játékban lévő zsetonjainak száma,
- hPot: a felhasználó még felhasználható zsetonjai,
- inHPot: a felhasználó játékban lévő zsetonjainak száma,
- mPot: az elnyerhető zsetonok száma.

Egy ilyen üzenet után, ha a korábban kapott információs üzenet azt tartalmazta, hogy a gép kezd, akkor a counter értéke 3 lesz, tehát a kliens egy operátor üzenetet fog várni. Ha a felhasználó kezdi a kört, akkor pedig aktívvá válnak a Fold, Check, Call/Raise gombok.

Operátor üzenet:

Egy játékos egy műveletét leíró üzenete. (Felépítése és tagjainak jelentését lásd fentebb.)

Információ a nyertesről:

Az üzenet megjelenik a Quit with save gomb alatt. Arról hordoz információt, hogy ki és milyen lapkombinációval nyerte a játékot (Draw, You, Computer). Ha ilyen üzenet érkezik, akkor a gépi játékos lapjai is láthatóvá válnak.

Speciális helyzet (all in):

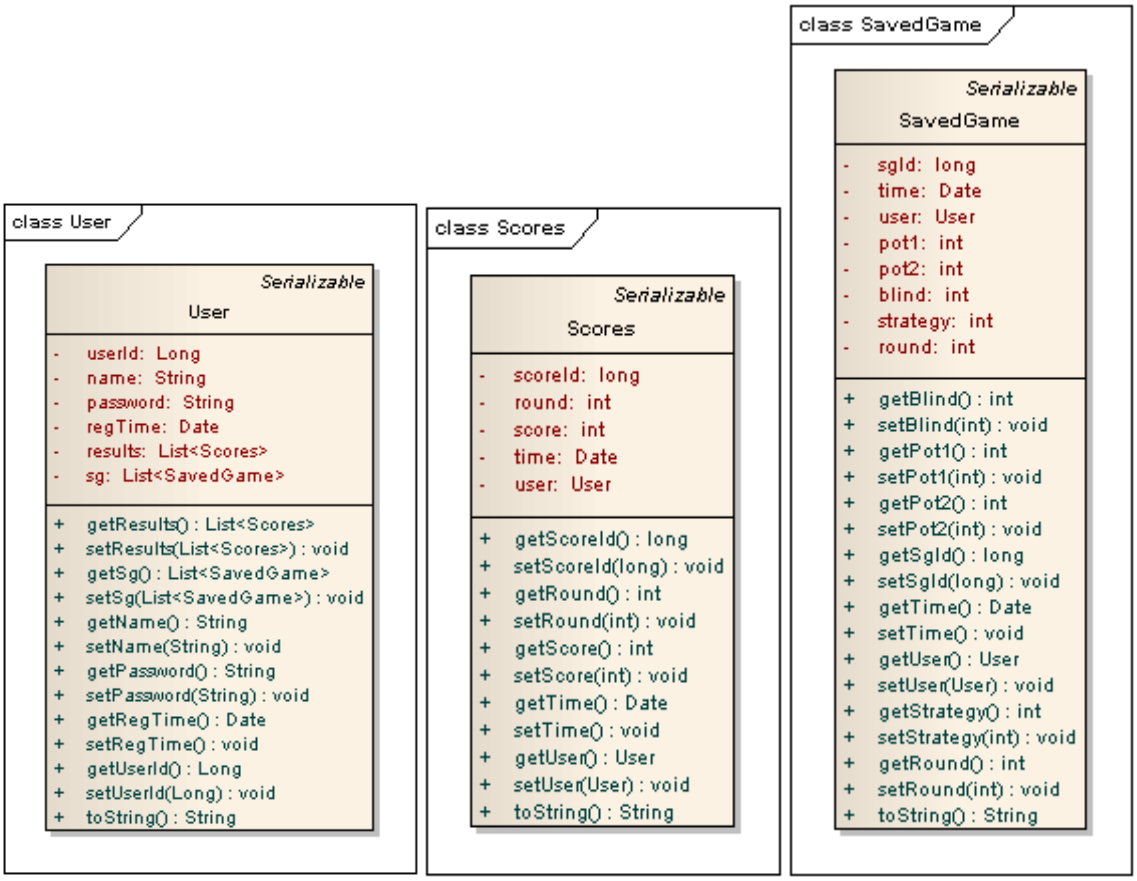
„All in”-ről abban az esetben beszélhetünk, amikor az egyik fél beteszi az összes megmaradt zsetonját a játékba és a másik fél ezt a tétet tartja. Ekkor a kliens megjeleníti a gépi játékos lapjait, a szerver pedig elküldi a még le nem osztott kártyák információit is a kliensnek, ezután közli, hogy melyik játékos nyert.

A program szerver oldalának megvalósítása

A program szerver részének elkészítéséhez Netbeans 6.9.1-es verziószámú fejlesztői környezetet használtam. A szerver program feladata kapcsolatot biztosítani a kliens számára, ezen a kapcsolaton keresztül üzenetek küldése és fogadása, a fogadott üzenet helyes feldolgozása. Továbbá a szerver program felel az adatbázis létrehozásáért és eléréséért, amit apache derby-vel valósítottam meg. Az adatbázis a szerver mappájában jön létre, tehát a felhasználó csak olyan meghajtón tudja helyesen futtatni a programot, ahol van írási jogosultsága. A szerver program 57 osztályt tartalmaz, így helyhiány miatt csak a fontosabb osztályok működését fejtem ki bővebben. Az osztályok működése csomagonkénti lebontásban a következő:

Persistance csomag

A csomag osztályait a Java Persistence Api keretrendszer segítségével készítettem el, amely lehetővé teszi, hogy a táblák egyedeit Java objektumokként kezeljem, ehhez az osztálydefiníció elé helyezett @Entity annotáció szükséges. Az adatbázison módosítást végrehajtó metódusok elé pedig a @Transactional annotáció elhelyezése szükséges.
A persistance csomagban találhatóak azok az osztályok, amelyek reprezentálják az adatbázis táblák egyedeit, valamint itt találhatók azok az osztályok, amelyek felelősek a táblákkal végzett műveletekért. A User, Scores és SavedGame osztályok az adatbázisban lévő táblák egyedeit reprezentálják.



16. sz. ábra

A többi osztály az adatbázis táblákkal történő különböző műveletekért felelősek.

A Registration osztály feladata egy új felhasználó felvétele a User táblába, azzal a megkötéssel, hogy két azonos nevű felhasználó nem szerepelhet a táblában.

Az AddScore osztály egy új eredményt hoz létre, és elhelyezi azt a Scores táblába.

A SaveAGame osztály pedig egy SavedGame objektumot ír a Saved_Game táblába.

A DeleteRegistration osztály az adott nevű felhasználót távolítja el a User táblából abban az esetben, ha szerepel adott nevű felhasználó adott jelszóval a táblában.

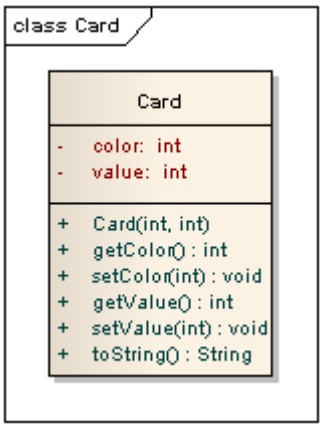
A Login, ListScores, ListSavedGame, ChooseASavedGame osztályok egy-egy lekérdezést hajtanak végre.

A táblák elérését egy EntityManager biztosítja. Az EntityManager paramétereit a spring-beans.xml, persistance.xml és az application.properties fájlok tartalmazzák.

Game csomag

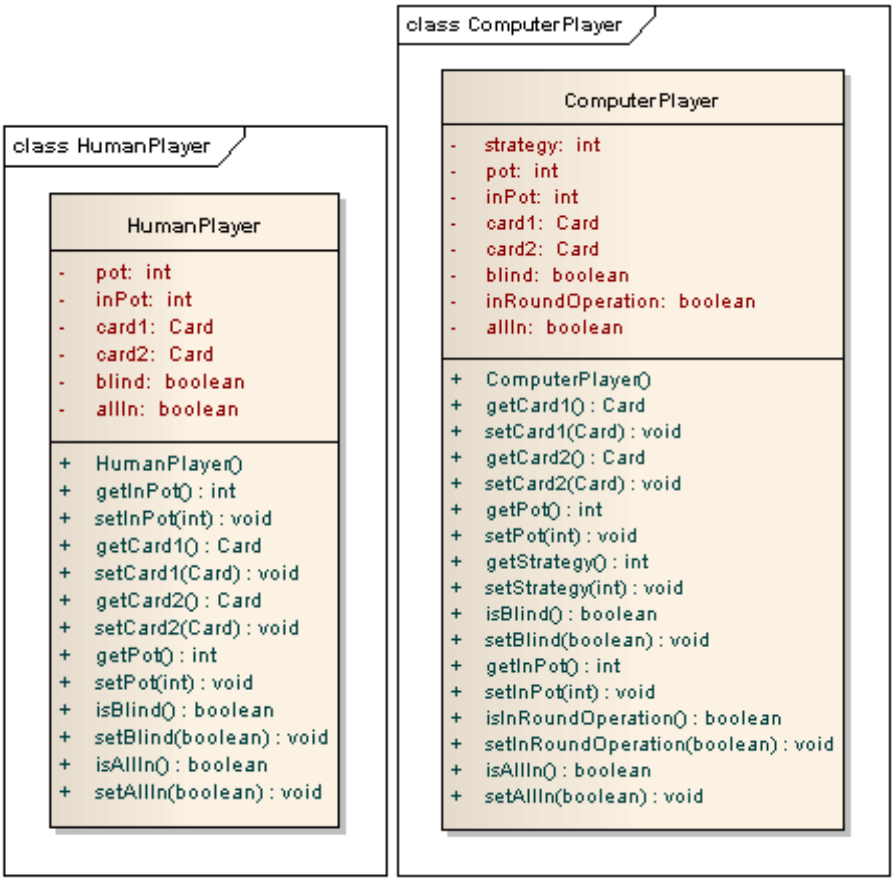
A Game csomag osztályai a játékhoz kapcsolódó osztályokat tartalmazza.

A Card osztály reprezentál egy kártyalapot. Két privát tagja van, color és value, amik egy adott kártya színét és értékét jelentik.



17. sz. ábra

A HumanPlayer osztály az emberi játékos tulajdonságait, a ComputerPlayer osztály pedig a gépi játékos jellemzőit tartalmazza.



18. sz. ábra

Mint látható a két osztály nagyon hasonló. A különbséget az adja, hogy a gépi játékosnak van stratégiája, amit a játék során határoz meg a program, valamint a játék vezérléséből adódóan a gépi játékosnál figyelni kell azt, hogy az adott körben hajtott-e már végre valamilyen műveletet. Ezt az inRoundOperation nevű privát logikai változó értéke tartalmazza. Ezeken a tagokon kívül a két osztály megegyezik, ugyanazokkal a tulajdonságokkal bírnak. A pot a játékosok zsetonjainak mennyiségét, az inPot pedig a még felhasználható zsetonjainak számát jelenti. A blind nevű logikai tag igaz érték esetén nagyvakot, ellenkező esetben kisvakot jelent. Az allIn nevű logikai tag igaz érték esetén azt jelzi, hogy a játékos már betette az összes zsetonját a játékba.

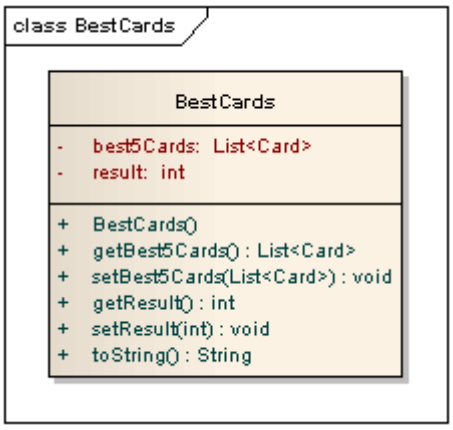
Az Operator osztály a játék során előforduló műveletek reprezentálja: dobás(fold), mehet(check), tét tartása(raise) és tét emelése(raise).

A Dealer osztály a játékban az osztó szerepét tölti be.

19. sz. ábra

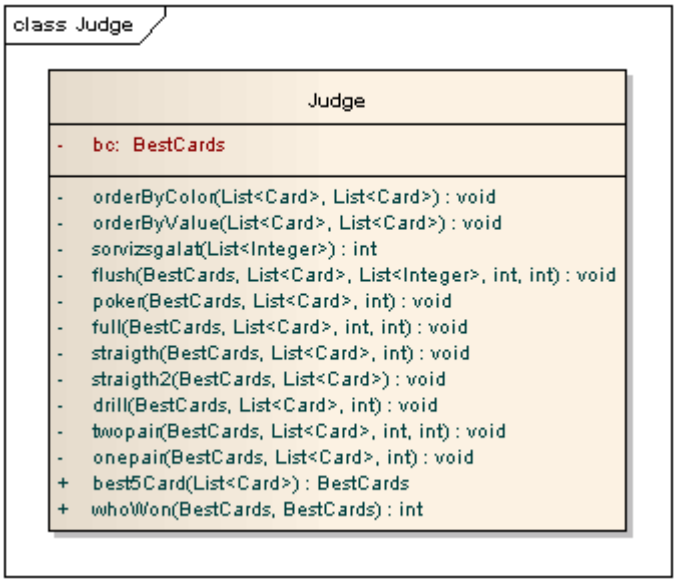
Feladata a kártyapakli előállítása, annak megkeverése, valamint a játék során a kártyák helyes kiosztása a játékosoknak.

A BestCards osztály a játékosok legjobb öt lapját valamint ennek a lapkombinációnak az erősségét tartalmazza.



20. sz. ábra

A result értéke 0-tól 9-ig vehet fel értéket a lapkombinációk erősségének függvényében. A 0 (a royal flush) a legerősebb, a 9 (high cards) a leggyengébb lapkombinációt jelenti.
A Judge osztály best5Cards metódusának feladata megállapítani egy játékos legjobb öt lapját hét kártyalapból kiválasztva. Ehhez először rendezi a kártyákat szín és érték szerint is. Majd az így rendezett listákat vizsgálva tudja eldönteni, hogy az adott hét kártyalapból melyik az öt legerősebb kombináció. A whoWon metódus feladata pedig eldönteni a két játékos közül azt, hogy az adott körülmények között melyikük nyert.



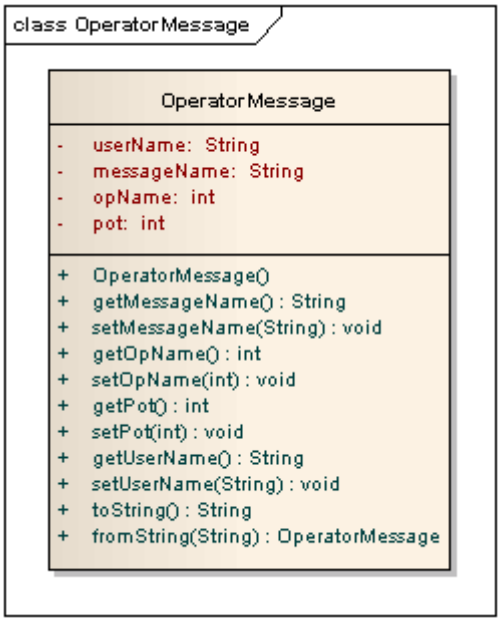
21. sz. ábra

A ChooseOperator osztály feladata megállapítani a gépi játékos számára a legmegfelelőbb operátort az adott körülményeket figyelembevételével.

A Game osztály felelős új játék indítása esetén alapértékekre állítani a játék paramétereit. Ha egy új kör kezdődik beállítja a vakokat, illetve, ha a gép kezd, akkor a ChooseOperator osztály segítségével kijelöl egy operátort a gépi játékosnak. Szintén az osztály feladata, hogy a vaktétek összegét emelje minden 10 lejátszott kör után. Ennél az osztálynál fontos megjegyezni, hogy singleton, tehát a program futása során egy Game objektum jön létre, ezáltal több helyről lehet módosítani ugyanazt a Game objektumot. Ennek a későbbiekben az OperatorHandler osztálynál lesz jelentősége.
Ha a játék tovább már nem folytatható – mivel valamelyik játékos zsetonjainak száma már a minimális tét alatt van -, akkor a befejezett játék és a játékos paraméterei alapján az AddScore osztály segítségével egy új eredményt ad hozzá a Result táblához.

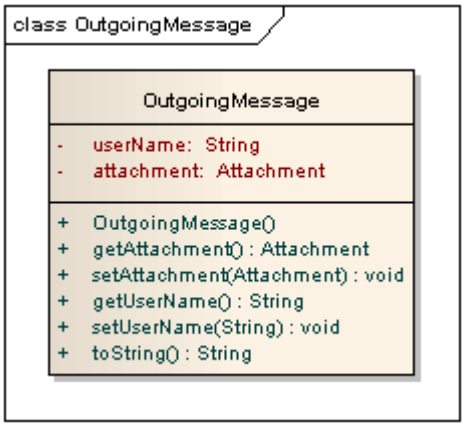
Message csomag

A csomagban található egy Attachment interface, amelyet minden üzenet implementál, ezáltal az üzenetek kezelése sokkal egyszerűbbé válik.
A klienstől érkező és a kliensnek elküldött üzenetek JSON formátumúak. Választhattam volna azt is, hogy az üzenetek xml alapúak, viszont a JSON formátumú üzenet sokkal rövidebb karakter sorozatban jelenik meg, mint egy xml formátumú üzenet. Egy üzenet JSON formátumban a következőképpen néz ki, egy OperatorMessage osztályú üzenetet alapul véve:
{”userName”:”Zoli”,”messageName”:”operator”,”opName”:0,”pot”:0}
Az üzenetek feldolgozását a későbbiekben fejtem ki.
Az itt található OperatorMessage osztály egy operátor üzenetet ábrázol, amely egyaránt lehet ki-, illetve bemenő üzenet.



22. sz. ábra

Az OutgoingMessage pedig a kimenő üzenetet ábrázolja.



23. sz. ábra

A userName a bejelentkezett felhasználó neve, akinek az üzenetet el kell küldeni, az attachment pedig maga a kimenő üzenet, amelyet valamely outgoingmessages csomagban lévő osztály reprezentál.

A csomagon belül lévő outgoingmessages csomagban található osztályok a kliens irányába továbbítandó üzenetek ábrázolják. Ezek közé tartozik a CardMessage, ami a kliensnek azt jelzi, hogy milyen kártyát kell megjelenítenie. A ConnectAnswer a belépésre, vagy a regisztrációra adott választ jelenti. A DeleteRegistrationAnswer a felhasználó törlésének sikerességét jelző üzenet. A GameInfoMessage egy játékról tartalmaz alapvető információkat (folytatódik-e még a játék, hanyadik körnél tart a játék, ki kezdi az adott

kört). A ListSavedGameAnswer a felhasználó elmentett játékainak listáját tartalmazza, amelyek közül betölthet egyet. A ListScoreAnswer az addig lejátszott játékok eredményét ábrázolja. A PotMessage a zsetonokra vonatkozó információkra vonatkozik. A WinnerInfo osztály a játék egy körének eredményét jelenti.

A csomagon belül lévő incomingmessages csomagban a klienstől érkező üzeneteket reprezentáló osztályok találhatóak. Ezek az üzenetek mutatják be a játékos interakcióit (regisztráció, belépés, játékmentése, új játék kezdése stb.)

Mivel az üzeneteket JSON formátumban küldi és fogadja a szerver, ezért a kimenő osztályok toString() metódusait úgy írtam felül, hogy egy JSON formátumú karakter sorozattal térjenek vissza, ezt a Gson.toJson függvényhívással lehet elérni. A bejövő üzeneteknél pedig írtam egy fromString(String gsonString) metódust, amely egy adott JSON formátumú karakter sorozatból előállítja a megfelelő üzenetet. Ezt a Gson.fromJson metódus biztosítja. Az üzenetek átalakítását Google Gson-nel valósítottam meg.

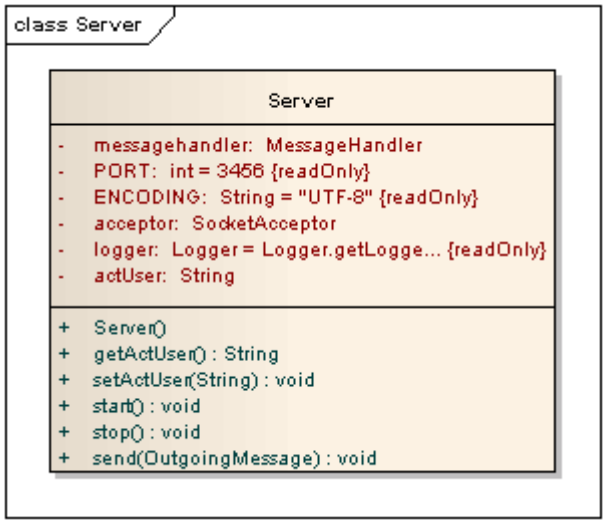
Gson csomag

Ez a csomag egyetlen osztályt tartalmaz, a GsonParser osztályt, amelynek feladata a beérkező üzenetek Google Gson formátumból történő helyes átalakítása.

24. sz. ábra

Server csomag

A szerver csomag tartalmazza a Server osztályt, amelynek a feladata a kliens és a szerver közötti kommunikációs csatorna kiépítése. A kommunikáció a 3456-os számú porton keresztül történik. A Server osztály ehhez a kapcsolathoz Java NIO library-n alapuló Apache mina keretrendszert használ. A keretrendszer használatával esemény vezérelt aszinkron kapcsolat alakítható ki a szerver és a kliens között. A framework megköveteli, hogy az osztály tartalmazzon egy kiterjesztett IoHandlerAdapter tagot, amely többek között a beérkező üzenetek feldolgozásáért is felelős. Az osztály feladata továbbá üzenetek fogadása a klienstől, valamint üzenetek küldése a kliens részére.



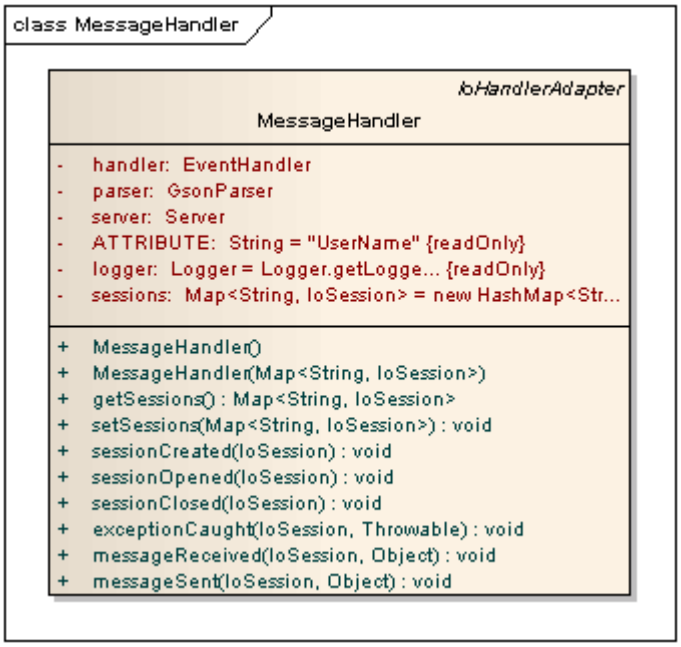
25. sz. ábra

A start() metódus hívásával történnek meg azok a beállítások, amelyek a kapcsolatra vonatkoznak. A szerver a start metódus végén kezd el várakozni a kliensre az acceptor.bind (new InetSocketAddress(PORT)) metódus hívással.

A stop() metódus meghívása a kapcsolat bontását eredményezi.

A send(OutgoingMessage message) metódus felelős egy kimenő üzenet helyes elküldéséért a kliens irányába.

A beérkező üzenetek kezelését a Server osztály messagehandler tagja végzi.

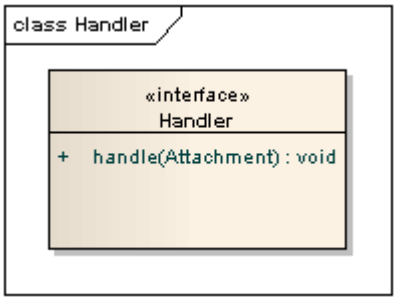


26. sz. ábra

A beérkező üzeneteket a MessageHandler osztály messageReceived metódusa érzékeli, majd átadja a parser nevű GsonParser objektumnak, amely átalakítja azt. Először egy UserConnect osztályú üzenet érkezik, ami a regisztrációt vagy a belépést takarja. Ha a regisztráció vagy belépés sikeres volt, akkor azt a session-t - amelyen keresztül az üzenet érkezett - eltárolja a MessageHandler a sessions nevű HashMap-be. A szerver a továbbiakban ezen az eltárolt session-ön keresztül fog kommunikálni a klienssel.

Eventhandling csomag

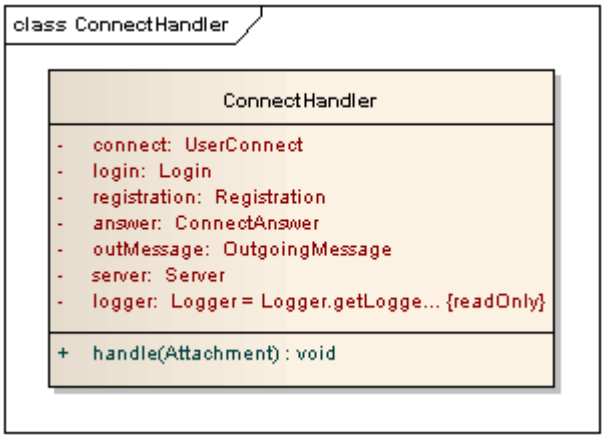
Az ebben a csomagban található osztályok feladata a beérkezett kérések teljesítése. A csomagban található egy Handler interface, amelynek egy handle(Attachment message) metódusa van. Az eseménykezelő osztályok ezt az interface-t implementálják, és a handle metódust írják felül.



27. sz. ábra

Mivel minden üzenet implementálja az Attachment interface-t, ezért az üzenetek osztályainak átalakítása hiba nélkül végrehajtható.

A ConnectHandler osztály feladata elvégezni a regisztrációt vagy belépést, valamint a megfelelő válasz üzenet megfogalmazása.

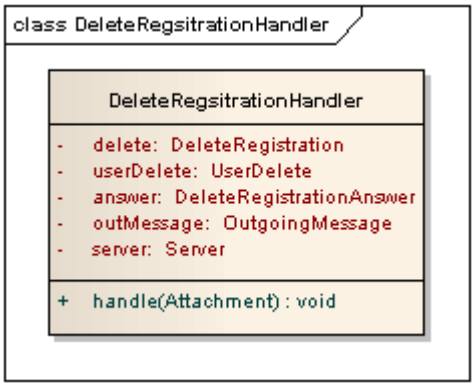


28. sz. ábra

A regisztrációhoz vagy belépéshez szükséges paramétereket egy UserConnect típusú üzenetből veszi ki. Az üzenet messageName paraméterét felhasználva megállapítja, hogy regisztráció, vagy belépés végrehajtását kérte-e a felhasználó. Ha regisztrációt, akkor a Registration osztályú registration privát tag reg(String userName, String password) metódusát hívja meg a megfelelő paraméterezéssel, és ennek megfelelően állítja be a válasz üzenet paramétereit. Ha ez a metódus a „Fail” karakter sorozattal tér vissza, akkor a regisztráció sikertelen volt. A ConnectAnswer típusú answer objektum paramétereit beállítja a regisztráció vagy belépés műveletének sikerességének függvényében, amelyet a szerver ezután elküld a kliensnek.

Ha az üzenet messageName paramétere azt jelzi, hogy a felhasználó belépést kíván végrehajtani, akkor a Login osztályú login privát tag login(String userName, String password) metódusa hajtódik végre. A válasz üzenet beállítása ugyanúgy történik, mint regisztráció esetén, tehát a metódus egy karakter sorozattal tér vissza, amely ha „Fail”, akkor a belépés sikertelen volt, ellenkező esetben sikeres.

A DeleteRegistrationHandler osztály felelős a klienstől érkező a regisztráció megszüntetését kérő üzenet végrehajtásáért.



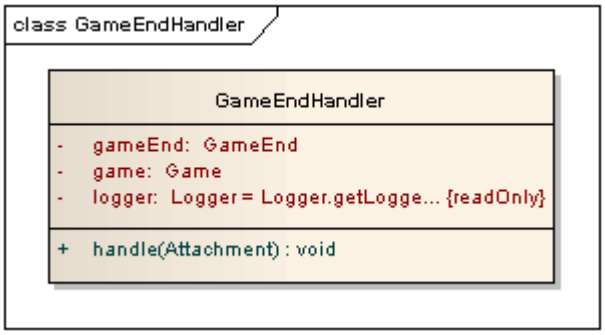
29. sz. ábra

A handle metódus paraméterének konkrét osztálya UserDelete lesz. Az üzenet tagjait felhasználva hajtja végre a regisztáció törlését a DeleteRegistration osztályú delete privát tag deleteReg(String userName, String password) metódusának hívásával. A metódus visszaad egy karakter sorozatot, amely, ha „succes”, akkor a törlés sikeres volt, ellenkező esetben sikertelen. Ez alapján beállítja a DeleteRegistrationAnswer típusú answer objektum paramétereit, amelyet ezután a szerver elküld a kliensnek.

A GameStartHandler osztály felelős egy játék egy új körének elindításáért. Amennyiben a játék első körét indítja el, úgy a Game típusú game objektum paramétereit alapértékre állítja, majd meghívja a playGame() metódusát.

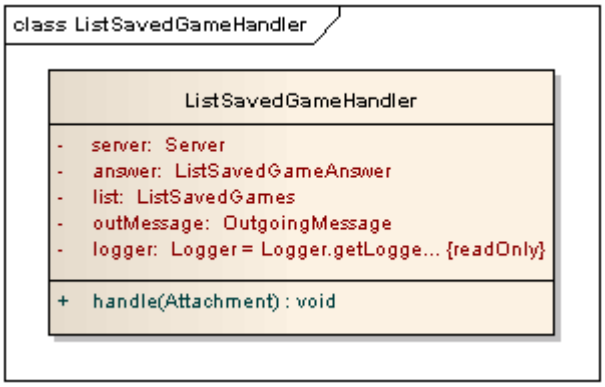
30. sz. ábra

A GameEndHandler osztály feladata a Game típusú game objektum paramétereinek visszaállítása alapértékre. Mivel a game tag singleton, ezért ez az osztály ugyanazt a Game típusú objektumot kezeli, mint a GameStartHandler osztály.



31. sz. ábra

A ListSavedGameHandler osztály feladata a belépett felhasználó által küldött ListSavedGameMessage típusú üzenet kezelése.



32. sz. ábra

A handle() metódus lekérdezi az adott felhasználó elmentett játékainak felsorolását a list ListSavedGames osztályú privát tag segítségével. A megkapott listát a szerver elküldi a kliensnek.

A ListScoreHandler osztály feladata a Result táblában lévő eredmények lekérdezése.

33. sz. ábra

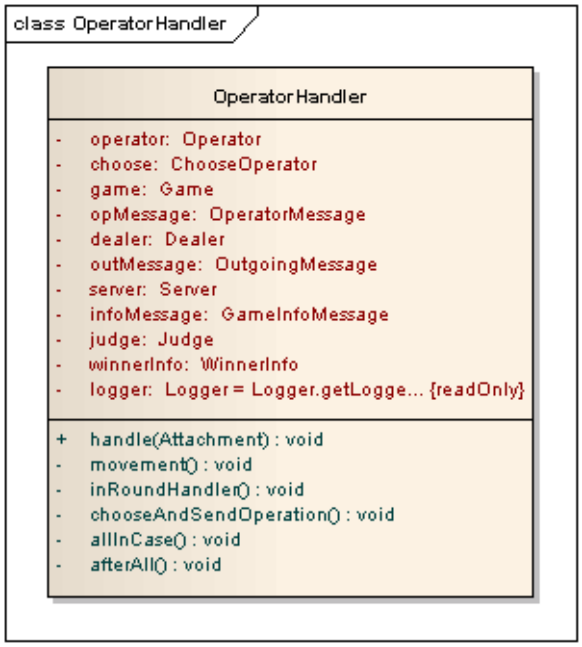
A handle metódusban a list ListScores osztályú privát tag végrehajtja a listScores() metódusát. A lista az eredmény eléréséhez szükséges körszámok és a dátum szerint növekvő sorrendben, az elért pontszámok alapján pedig csökkenő sorrendben rangsorolja a felhasználókat. Az eredményül kapott listát a szerver elküldi a kliensnek.

A LoadGameHandler osztály feladata a LoadGameMessage típusú üzenetben megadott azonosítójú elmentett játék betöltése.

34. sz. ábra

A handle() metódusban a choose.chooseSavedGame(loadMessage.getGameId()) függvényhívással egy lekérdezés ellenőrzi, hogy létezik-e a megadott azonosítójú elmentett játék. Ha létezik, akkor a game Game osztályú objektum paramétereit beállítja a kiválasztott elmentett játék paraméterei alapján.

Az OperatorHandler osztály feladata a klienstől érkező OperatorMessage típusú üzenet helyes értelmezése, feldolgozása, valamint az üzenetnek megfelelő válasz megfogalmazása.



35. sz. ábra

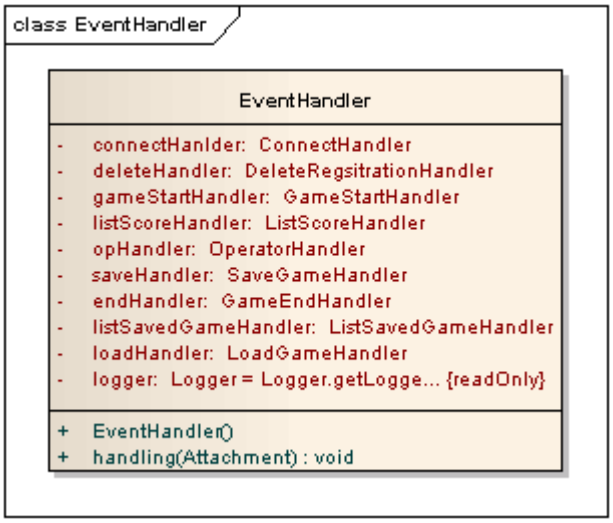
Ha egy OperatorMessage típusú üzenet érkezik, akkor a game Game típusú objektum paraméterei az üzenet paraméterei alapján megváltoznak. Mivel a game singleton, ezért itt ugyanarról a game objektumról van szó, mint a GameStartHandler osztály esetén. A felhasználó művelete, valamint a játék állapota alapján különböző válasz üzeneteket fogalmaz meg. A válasz üzenet lehet OperatorMessage, CardMessage, PotMessage vagy WinnerInfo osztályú objektum. A válasz üzenet típusa attól függ, hogy a játék éppen hol tart, és a beérkezett OperatorMessage milyen értékű paramétereket tartalmazott.

A SaveGameHandler osztály feladata a klienstől érkező SaveGameMessage típusú üzenet feldolgozása.

36. sz. ábra

Első lépésben a handle() metódusban létrejön egy új SavedGame típusú objektum. Ennek paramétereit az aktuális játék paraméterei alapján beállítja, majd a save SavedAGame osztályú privát tag save(String userName, SavedGame sg) metódusának végrehajtásával a Saved_Game táblába egy új rekordot vesz fel. Mivel a game singleton, ezért az aktuális, játék paramétereit tartalmazza.

Az EventHandler osztály feladata eldönteni, hogy az adott üzenetnek megfelelő eseménykezelő metódusa hajtódjon végre.



37. sz. ábra

A handling metódus a beérkezett üzenet típusának vizsgálatával eldönti, hogy melyik eseménykezelő handle() metódusának meghívása szükséges.

Main csomag

A csomag tartalmazza a Main osztályt, amely elindítja a szerver programot. A Main osztály main metódusa tartalmazza az ApplicationContext ctx = new ClassPathXmlApplicationContext("spring-beans.xml") függvényhívást, elvégzi a program inicializálását, legenerálja a @Component annotációval ellátott osztályok singleton példányait, valamint kiépíti a kapcsolatot a spring-beans.xml-ben megadott adatbázissal. Továbbá a main metódus tartalmazza a következő sorokat:

Server server = ctx.getBean(Server.class), amely elérhetővé teszi a server objektumot

server.start(), amely elindítja a server objektumot.

Összegzés

A program természetesen továbbfejleszthető. Ilyen továbbfejlesztés lehet például, ha a játék nem két, hanem több játékos között zajlana. A program egy másik lehetséges evolúciója az lehetne, ha a szerver program nem a felhasználó számítógépén futna, hanem egy távoli gépen. Így a kapcsolat webes alapúvá válna. A játékot valósághűvé lehetne tenni azzal, ha a gépi játékos valamilyen mesterséges intelligencián alapuló algoritmust használna a műveletei kiválasztásához.

Szakdolgozatom célja egy kétszemélyes póker játék megvalósítása volt. A hangsúlyt a program megvalósításához szükséges technológiákra helyeztem.

Az első fejezetben kifejtettem a programmal szembeni felhasználói követelményeket, amelyek a következők voltak:

- regisztráció
- belépés
- regisztráció törlése
- eredmények megtekintése
- játék betöltése
- új játék indítása
- játék elmentése
- új eredmény bejegyzése

Az elkészült program hiba nélkül megvalósítja ezeket a felhasználói követelményeket. A program minden funkcióját többször is teszteltem, magát a játékok megközelítőleg 50-szer játszottam végig.

A program szerver oldalának megvalósításához többek között Java-t, azon belül Spring, Apache mina, JPA keretrendszereket használtam. Valamint az üzenetek formátumának JSON-t választottam. Ezek a technológiák nagyon hasznosak, és helyes használat mellett leegyszerűsítik a programozó munkáját.

A Spring framework használata leegyszerűsíti a fejlesztést, a Spring contex elvégzi a Spring komponensek közötti függőségek kezelését.

Az Apache mina Java NIO library-n alapuló keretrendszer, használatával esemény vezérelt aszinkron kommunikáció valósítható meg.

A JSON formátumú üzenetek kezelése egyszerű, a küldött és fogadott üzenetek lényegesen rövidebbek, mintha xml alapúak lennének. Az üzenetek kódolásához, dekódolásához Google Gson-t használtam.

A JPA keretrendszer használatával az adatbázis táblák egyedeit Java objektumokként lehet kezelni, valamint egyszerűen lehet lekérdezéseket, módosításokat, törléseket végrehajtani.
A kliens oldal flash alapokon nyugszik, mellyel ez előtt még nem dolgoztam, mégis sikerült megvalósítanom, azokat amiket elterveztem.
Úgy érzem, hogy a szakdolgozatommal elértem a kitűzött céljaimat, és sikerült megismerkednem a választott technológiákkal. Persze nem állíthatom, hogy az ismereteim átfogóak lennének, de mindenképpen sikerült rengeteget tanulnom a dolgozatom elkészítése folyamán.

Köszönetnyilvánítás

Ezúton szeretném megköszönni Dr. Horváth Géza tanár úrnak, hogy elvállalta a szakdolgozatom témavezetői feladatait.

Továbbá szeretném megköszönni a rengeteg elméleti segítséget testvéremnek, Kocsi Jánosnak, aki a szakdolgozatom külső témavezetője volt. Nélküle biztosan nem készült volna el ez a szakdolgozat.

Irodalomjegyzék

1. A programmer's guide to Java SCJP certification: a comprehensive primer/Khalid A. Maghal, Rolf W. Rasmussen.- 3rd ed. - New York: Apress, cop. 2009. 1089. p.

2. Pro JPA 2: Mastering the Java Persistance API/ Mike Keith, Merrick Schincariol. – New York: Apress, cop. 2009. 538. p.

3. Spring Recipes/ Gary Mak, Josh Long, Daniel Rubio.- 2nd ed. – New York: Apress, cop. 2010. 1105. p.

4. Adobe Flex 3: Developer Guide, - San Jose: Adobe Systems Incorporated, cop. 2008. 581. p.

Internetes források

1. http://www.poker.zsuga.com/poker_tortenete/poker_tortenete.php (2010. november 16.)
2. http://www.onlinepokerinfo.hu/poker-tortenelem.php (2010. november 16.)
3. http://www.pokerpro.hu/poker-tortenete/1298/a-poker-toertenete (2010. november 16.)

Plágium - Nyilatkozat

Szakdolgozat készítésére vonatkozó szabályok betartásáról nyilatkozat

Alulírott (Neptunkód: ……) jelen nyilatkozat aláírásával kijelentem, hogy a

……..

című szakdolgozat/diplomamunka

(a továbbiakban: dolgozat) önálló munkám, a dolgozat készítése során betartottam a szerzői jogról szóló 1999. évi LXXVI. tv. szabályait, valamint az egyetem által előírt, a dolgozat készítésére vonatkozó szabályokat, különösen a hivatkozások és idézések tekintetében.

Kijelentem továbbá, hogy a dolgozat készítése során az önálló munka kitétel tekintetében a konzulenst, illetve a feladatot kiadó oktatót nem tévesztettem meg.

Jelen nyilatkozat aláírásával tudomásul veszem, hogy amennyiben bizonyítható, hogy a dolgozatot nem magam készítettem vagy a dolgozattal kapcsolatban szerzői jogsértés ténye merül fel, a Debreceni Egyetem megtagadja a dolgozat befogadását és ellenem fegyelmi eljárást indíthat.

A dolgozat befogadásának megtagadása és a fegyelmi eljárás indítása nem érinti a szerzői jogsértés miatti egyéb (polgári jogi, szabálysértési jogi, büntetőjogi) jogkövetkezményeket.

hallgató

Debrecen,