Ylläpitodokumentti

Asdf

Helsinki 5.5.2008

Ohjelmistotuotantoprojekti

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Kurssi

581260 Ohjelmistotuotantoprojekti (6 ov)

Projektiryhmä

Kuisma Sami
Louhio Jaakko
Rimpilä Raine
Urtela Mika
Vilhonen Vesa

Asiakas

Pasanen Tomi, Teemu Saukonoja

Johtoryhmä

Yrjänäinen Sampo
Simola Kimmo

Kotisivu

`http://www.cs.helsinki.fi/group/asdf`

Versiohistoria

Versio	Päiväys	Tehdyt muutokset
1.0	15.4.2008	Dokumentti luotu
1.1	17.4.2008	Dokumenttia viimeistelty
1.2	2.5.2008	Lisatty Peliaulakuvauksia

Sisältö

1 Johdanto

Asdf on Helsingin yliopiston tietojenkäsittelytieteenlaitoksen ohjelmistotuotantoprojektin 2008 kevään ryhmä, joka toteutti asiakkaan tilaaman yleiskäyttöisen pokeripalvelinohjelmiston. Tässä dokumentissa käydään läpi projektiryhmän tuottamaa ohjelmistoa, asennusta, ylläpitoa sekä ohjelmistoon liittyviä ominaisuuksia, ratkaisu- ja toteutusmalleja. Dokumentti on tarkoitettu ohjelmiston ylläpitoa varten. Projektissa keskityttiin toteuttamaan monipuolinen ja hyvin laajennettava palvelin-asiakas peliohjelmisto. Pokeripalvelin ja rajapinnat on toteutettu yleiskäyttöiseksi ja soveltuu sellaisenaan kaikkien mahdollisten vuoropohjaisten pelien tukemiseen, ja dokumentissa on painotettu ohjelmiston jatkokäyttöä sekä laajentamisen mahdollisuutta ja sen käytännön toteuttamista.

Projektin puitteissa ohjelmistoon toteutettiin palvelinohjelmisto, tietoliikenne sekä asiakasohjelmisto. Asiakasohjelmisto toteuttaa peliaulanäkymän sekä pokeripeleihin soveltuvan pelipöytänäkymän. Peliaula hakee dynaamisesti palvelimelta sinne toteutetut pelit ja mahdollistaa niihin liittymisen sekä Serverille kirjautumisen. Kaikki viestienvälitys palvelimen ja asiakasohjelmiston välillä on XML-skeemalla määritelty. Määrittelyistä luodaan pelien viestinvälityksen jälkeen pelien tarvitsemia luokkia JAXB-XML-sidontaa käyttäen.

Pokeripalvelin ohjelmiston tarkoitus on toimia yleiskäyttöisenä ympäristönä testattaessa erilaisia tekoälysovelluksia. Projektin puitteessa kuitenkin toteutettiin vain ympäristö ja rajapinnat Texas Hold'em pelaamiseen, sekä suoraviivainen toteutusmahdollisuus muille pokeripeleille. Rajapintojen suunnitteluun panostettiin erityisen paljon.

Ohjelmisto koostuu useista osajärjestelmistä, sekä useista valmiista projektissa hyödynnetyistä komponenteista. Luvussa 2 käydään läpi ohjelmiston yleiskuvaus. Luvussa 3 käydään läpi toteuttamatta jääneet toiminnallisuudet. Luvussa 4 kerrotaan ohjelmiston asennuksesta sekä käyttöönotosta. Luvuissa 5-8 käydään läpi ja kuvataan ohjelman komponentit. Komponentit on kuvattu ylläpitäjän ja/tai jatkokehittelijän tarvitsemalla tarkkuudella. Komponentteihin liittyvästä koodista löytyy yksityiskohtaiset kommentit ohjelman toiminnasta sekä Java-dokumentaatio kommentit. Luvussa 9. annetaan sanalliset kuvaukset ohjelmiston käyttämistä rajapinnoista. Liitteeksi 1. on ohjelmiston XML-protokollakuvaukset. Liite 2. on ohjelmiston asennusohje. Liite 3 on liitännäisten kehitysohje.

Ohjelmiston Java-dokumentaatiot löytyvät osoitteesta:

texttthttp://www.cs.helsinki.fi/group/asdf/javadoc

1.1 Sanasto

Sessio - Asiakkaan yhden loogisen kommunikointitarvekokonaisuuden tunniste. Erimerkiksi peliaulalla on oma sessiotunniste, ja jokaisella pelillä johon asiakas on liittynyt on oma sessiotunniste.
Pelilogiikka - Kuvaus pelitilanteesta ja sen muutoksista asiakkaiden käyttäytymisen funktiona.
JAXB - Java Architecture for XML Binding. Luo Java-luokista XML-skeeman ja XML-viestistä skeeman mukaan tuo takaisin Java-luokaksi.
XML - Projektissa käytetty kuvauskieli tiedonvälityksessä osajärjestelmien välillä.
NIO - Ohjelmistossa käytetty Javan valmis verkko-I/O toteutus. Tulee sanoista New I/O.

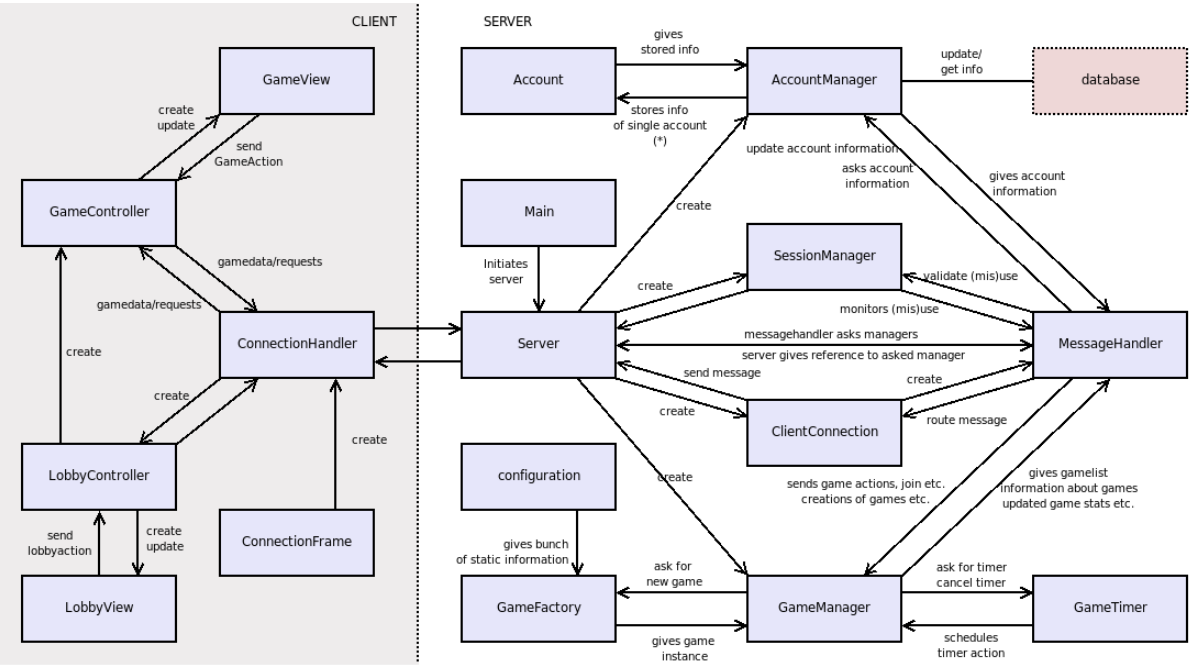
2 Ohjelmiston yleiskuvaus

Järjestelmä koostuu palvelimesta, sen kanssa keskustelevista asiakkaista ja erillään palvelimesta toimivista pelilogiikkaluokista. Palvelin pitää kirjaa asiakasyhteyksistä sekä sessioista, ja lisäksi ylläpitää rakennetta käynnissä olevista peleistä. Palvelin vastaa asiakkaiden pyyntöihin ja välittää pelipyynnöt oikeille peli-instansseille. Palvelin osaa mm. luetella palvelimelta löytyvät pelityypit, ja palvelimen tuntemat aktiiviset pelit.

Asiakas on loppukäyttäjän käyttämä järjestelmä, jolla otetaan yhteys TCP/IP protokollalla palvelimeen. Asiakkaan tulee luoda itselleen käyttäjätunnus ja kirjautua sisään ennen kuin pelaaminen on mahdollista. Asiakas voi kirjautumisen jälkeen pyytää palvelimelta peliluetteloa, tai luoda uusia palvelimen tuntemia pelejä.

Pelilogiikka toimii palvelinohjelmistossa. Pelilogiikan tulee toteuttaa projektin tuottamat pelirajapinnat. Pelilogiikka ladataan dynaamisesti palvelimella eikä peli-instanssit pääse muokkaaman palvelimen tietoja.

Asiakkaan on tärkeää kyetä näyttämään pelitilanne, ja tämän vaatima logiikka on asiakkaalla tärkeää olla. Kuitenkaan pisteytystä tai lopputuloksen laskentaa ei suoriteta asiakkaalla, koska tiedot voivat väärinkäyttäjän tapauksessa olla epärehtejä, eikä niitä siksi suositella palvelimen pelilogiikassa noteeraamaan. Palvelin tallentaa pisteytykset käyttäjien tileille. Kuvassa 1. on esitetty Peliohjelmiston komponenttien yhteistoimintakuvaus.



Kuva 1: Peliohjelmiston yhteistoimintakuvaus.

3 Toteuttamatta jääneet toiminnallisuudet

- (V6) Pelaajan nimen perusteella pelihakua ei toteutettu.
- (V12) Yleinen pelihistorian tallennus.
- (V15) Pelaajatilejä ei voi poistaa muuten kuin suoraan tietokannasta.

Parannusehdotukset

Pisteyksen tallennus tietokantaan uusille peleille on hankalaa, sillä se vaatii tietokantataulujen muokkausta tai listäystä. Tähän voisi pohtia vaihtoehtoisia menetelmiä.

Peliohjelmat voisi ladata dynaamisesti palvelimelta, niin niiden levittäminen olisi helpompaa ja turvallisempaa sekä palvelimen toiminnallisuudelle, että käyttäjien mukavuudelle.

GameControllereita asiakasohjelmistossa ei poisteta poistuessa pelistä. Tämä hukkaa muistia, mikäli käyttäjällä ei ole botteja pelaamassa pelissä. GameControllerin poistaminen aiheuttaa kaikkien pelaajan lisäämien bottien kuolemisen kyseisestä pelistä, joten sitä ei pitäisi tehdä ennen kuin kaikki pelaajan agentit ovat pelistä poistettu tai poistuneet.

Tyhjien pelien poistaminen palvelimelta ei toimi GameTimer-luokassa tapahtuvan ajastimen peruuttamisen vuoksi. Tämä tulisi korjata.

Pokerilogiikan tilannepäivityksessä lähetetään tilatieto-olio, joka välittää kaiken julkisen tiedon pelitilanteesta. Tämä on verrattain raskasta, ja tietoa voisikin lähettää päivitysten muodossa, jättäen muuttumattomat tiedot lähettämättä.

Liukulukujen käyttö rahaliikenteessä on Väärin(tm). Pitäisi olla täsmäpistelukuesitys. Liukulukuvirheitä voisi myös harkita korjaavan pyöristyksillä, mikäli on kyvykkäistä ihmisistä pulaa.

4 Ohjelmiston toimintaympäristö ja käyttöönotto

Sekä palvelin että asiakas vaativat Java 6 tai uudemman toimiakseen. Käyttöjärjestelmä voi kummassakin tapauksessa olla mikä vain, jossa on Java 6 tai uudempi, mutta palvelimen tapauksessa suositus on Linux järjestelmä, eikä toiminnan eheydelle anneta mitään takeita muissa järjestelmissä.

Seuraavassa käsitellään palvelimen asentamista uuteen tietojärjestelmään, sekä asiakkaan asentamista toiseen tietojärjestelmään.

Tietokannan asennus onnistuu asentamalla PostgreSQL tietokanta mielivaltaiseen järjestelmään, johon on mahdollista ottaa TCP/IP yhteys. Lisäksi täytyy asettaa käyttäjätunnus ja osoite -kentät tietokantaa vastaaviksi server.conf tiedostoon.

Palvelimen asentaminen onnistuu purkamalla Server.zip:n sisältö haluttuun koneeseen ja käynnistämällä runServer.sh linux -järjestelmässä tai runServer.bat windows -järjestelmässä. Palvelin tarvitsee myös asennetun tietokannan (katso edellinen kohta) ja konfiguraatiotiedoston.

Asiakas löytyy Client.zip tiedoston sisältä. Pura tämä hakemiston path_from_root_to_target alihakemistoon client. Aja samaisesta kansiosta nyt runPokerClient.sh (linux -järjestelmä) tai runPokerClient.bat (windows -järjestelmä).

5 Palvelin

Palvelimen komponenttien yhteistoimintakuvaus esiteltiin luvussa 2. Seuraavassa on selitetty komponentteihin liittyviä ominaisuuksia.

- Main

  Tämä luokka käynnistää Server -luokan ja asettaa lokiin kirjaamisen.

- Server

  Alustaa Java.nio pistokkeet yhteyksien kuuntelemiselle ja hyväksymiselle. Luokka pyörii jatkuvassa silmukassa run-metodissa, jossa kutsutaan toistuvasti valitsijaluokan select metodia. Luokka huolehtii myös erilaisten managerien alustamisesta ja välittämisestä muille luokille. Server-luokka alustaa uuden ClientConnection-luokan jokaista uutta hyväksyttyä yhteyttä kohti.

- ClientConnection

  Edustaa yksittäistä pistokeyhteyttä. Luokka tarjoaa metodit viestien lähettämiseen ja sisältää vastaanotetun viestin käsittelylogiikan. ClientConnection alustaa MessageHandler-luokan ilmentymän, joka tekee viestin korkean tason käsittelyn.

- MessageHandler

  Huolehtii viestien käsittelystä. Luokka huolehtii sessiotunnisteiden käytöstä, jotta yhdistetyt asiakkaat eivät voi yrittää huijata eivätkä välitetyt viestit mene väärille vastaanottajille. MessageHandler käyttää Serverin omistamien managereiden palveluita.

- AccountManager, Account

  Sisältää käyttäjätunnusten luonnin, poiston ja muutokset. Myös autentikointi on tehty tässä luokassa. Luokka käyttää tietokantaa autentikointiin. Jokaista kirjautunutta käyttäjää kohti alustetaan Account ja se talletetaan luokan ClientConnection jäsenmuuttujaan, josta se voidaan hakea kun halutaan tietää käyttäjän autentikoinnin tilanne ja mahdolliset käyttäjäoikeudet.

- SessionManager

  Pitää kirjaa yhteyksien sekä sessiotunnisteiden käytöstä. tällöin voidaan tarkistaa, että yhteydet yritä käyttää toistensa sessiotunnisteita.

- GameManager

  Sisältää pelien luontiin liittyvät toiminnot. Tämän luokan kautta kulkevat kaikki peleille menevät viestit. GameManagerilla on lista pelejä ja se käyttää GameFactorya uusien pelien luontiin.

- GameTimer

GameTimer on peliliitännäisille tarjottu ajastinrajapinta. Luokka on hyvin samantapainen kuin Timer-luokka, jonka se perii. GameTimer synkronoi ajastettujen tapahtumien toimitukset peleille, kun palvelin saattaa olla eri säikeestä käsittelemässä peliä samaan aikaan.

- Configuration

  Lukee käyttäjän asetukset server.conf tiedostosta työhakemistosta ja tarjoaa ne staattisten metodien välityksellä kysyjille.

Seuraavassa on kuvattu tarkemmin uuden pelin lisääminen palvelimelle, autentikoinnin toteutus sekä palvelimen säikeistyksestä.

5.1 Tietokanta

Palvelin käyttää autentikointiin PostgreSQL-tietokantaa. Tietokannan yhteysmerkkijono ja käyttäjätunnukset tulee määritellä server.conf tiedostoon. Autentikointitietokannan taulun voi luoda seuraavalla SQL-lauseella:

```
CREATE TABLE account (
name varchar(256),
password char(32),
type short,
balance decimal,
PRIMARY KEY(name)
);
```

Salasana-kentän sisältö on MD5-hajautusarvo käyttäjätunnuksen ja selväkielisen salasanan yhdisteestä heksadesimaalimerkkijonona.

5.2 Säikeistys

Palvelin on toteutettu kahden säikeen avulla. Ensimmäinen säie on vastuussa Java.nio luokkien yhteysoperaatioiden käsittelystä ja toinen ajastimien hallinnasta. Ajastimia tarjotaan liitännäisille. Synkronointi säikeiden välillä tapahtuu liitännäistä kutsuttaessa ja liitännäisestä ulospäin lähtevissä kutsuissa.

6 Tiedonsiirto

6.1 JAXB, XML

Asiakkaan ja palvelimen tiedonvälityksessä on hyödynnetty valmista JAXB-pakettia, joka muuntaa Java-luokkia XML-muotoiseksi informaatioksi, ja takaisin. Viestit kääritään XML Schema kielellä määriteltyihin viestiluokkiin, jotka voivat määrittelyn mukaan olla (kirjoittamisen hetkellä) kysely-, vastaus- tai tilannepäivitystyyppisiä viestejä. Nämä viestityypit taas jakautuvat erilaisiin viestityyppeihin, kuten peliviesteihin tai autentikointiviesteihin. Katsaus käytettyihin viestiluokkiin löytyy mukaan liitetystä XML-skeemasta ja Javadoc -dokumentaatioista. Viestien määrittely on hoidettu XML-skeemalla niin, että skeema jättää prosessoimatta valittujen viestityyppien tietynnimiset alielementit. Tämän avulla ohjelmiston liitännäisten avulla tehdyt laajennukset voivat määrittää omat valinnaiset viestityyppien parametrit, mutta viestin eheys pystytään kuitenkin tarkistamaan luotettavasti ennen liitännäisille luovuttamista. Jaxb työkalujen käyttöä rohkaistaan myös liitännäisten kehittelyssä sillä etuna on protokollan väärintulkitsemisen tai -käyttämisen huomaaminen jo kääntämisvaiheessa, sillä viestin käsittely tapahtuu perinteisten Java-luokkien kaltaisten metodien kutsuilla.

6.2 java.nio

Tietoliikennepistokkeiden kanssa käytetään Javan New I/O (Java.nio) luokkia. Luokat mahdollistavat asynkronisten pistokkeiden käyttämisen, jolla voidaan ehkäistä tilannetta, jossa jokaista asiakasyhteyttä varten on oma säie. Asiakkaan ja palvelimen välinen liikenne on toteutettu niin, että asiakas pitää koko session ajan yhteyttä auki palvelimeen. Yhteyden ollessa auki kumpi tahansa pää pystyy herättämään keskustelukumppanin jonkin tapahtuman sattuessa. Jatkuvasti auki olevan yhteyden takia viestinvälityksessä lähetetään ensin viestin koko ja sitten vasta itse viesti. Viestin koko lähetetään 32-bittisenä kokonaislukuna, jonka tavujärjestys noudattaa tietoliikenteessä käytettyä Big-Endian (Network Byte Order) tavujärjestystä.

6.3 Tietoliikenneprotokolla

Protokolla on jaettu kolmeen ylätyyppiin päivitys, kysely, vastaus. Seuraavassa on selitetty mihin kutakin tyyppiä on käytetty. Toteutettu protokolla löytyy liitteestä 2 tarkempine selityksineen.

6.3.1 Päivitys

Päivitys on serverin lähettämä pelitilanteen päivittämiseen käytetty viesti. Sisältävät päivitetyt tiedot pelitilanteesta, kuten pelitilanneolion, keskusteluviestit ja muun pelintilannesisältöön liittyvän tiedon.

6.3.2 Kysely

Kysely on asiakasohjelmiston pelipöytään sekä peliaulan käyttämät viestit. Sisältävät viestipyynnöt, joiden avulla asiakasohjelmisto pyytää palvelinta suorittamaan eri toimintoja. Peliaulan toimintoihin kuuluu pelilistausten ja pelitietojen pyytäminen, pelin luominen ja pelipöytään liittyminen, palvelimelle kirjautuminen sekä uusien tilien luonti. Pelipöytä lähettää keskusteluviestit, penkin varaukset sekä peliin liittymisen kyselyviestinä. Mahdollista on myös että pelilogiikka pyytää asiakkaalta kyselyä.

6.3.3 Vastaus

Palvelin käyttää vastausviestejä vastausviesteinä pelipöydän sekä peliaulan pyyntöihin. Lisäksi pelipöydästä välitettävät pelitapahtumat lähetetään palvelimelle vastausviesteinä.

6.4 Pokeripelissä käytetty protokolla

Asiakkaan ohjelmiston pokeripeli viestii palvelimella pyörivän pelilogiikan kanssa käyttämällä spesifistä pokeripeleille tehtyä protokollaa. Protokollan mukaiset viestit kulkevat yllä esitetyn prokollan mukaisten otsakkeiden alla. Viestien sisältö on sijoitettu erityiseen parametrielementiin, joka voi sisältää mitä tahansa. Viestit sisältävät pelitilanne-, päivitys-, käsikortti- sekä pelitapahtumaviestejä.

7 Pelilogiikka

Pelilogiikkamoduulit sijaitsevat palvelimella mutta ne on eroteltu itsenäisiksi moduuleiksi. Pelilogiikkamoduulien täytyy toteuttaa Game-rajapinta. Pelilogiikkamoduulit toteuttavat itse oman toiminnallisuutensa. Logiikkamoduuleilla on mahdollisuus käyttää palvelimen valittuja palveluita abstraktin Game-luokan tarjoamien metodien kautta. Ajastinpalveluita on pakko käyttää tätä kautta, jotta synkronointi toimii oikein.

7.1 Pokeripeli: GameModel

Projektin yhteydessä toteutettuun TexasHoldem peliin liittyvä peli-informaatio-objekti, joka toteuttaa gameinfo rajapinnan. GameModel sisältää pelin kaiken julkisen tiedon ja aksessorit niiden muunteluun. Aksessoreilla pelilogiikka päivittää pelitilannetta. Pelilogiikan ulkopuolelle pelitieto-olio (GameInfo) lähetetään ilman tietosisältöä muuttavia aksessoreita. Olioiden tietosisältönä on kaikki pelin esittämiseen tarvittava tilatieto. Metodeista kuvaukset löytyvät Javadoc-liitteestä.

7.2 Pokerilogiikka

Kuvassa 2 on esitelty ohjelmiston käyttämä yleiskäyttöinen pokeripelilogiikka tilakoneena. Sen lisäksi että uusia pelejä voidaan toteuttaa luomalla pelejä, jotka toteuttavat game-rajapinnan voidaan uusia pokeripelejä periä pokerilogiikasta vaihtelemalla jako-, panostus- sekä kortinvaihtokierrostilojen määrää ja järjestystä. Lisäksi uuteen peliin tulee toteuttaa voittajanlaskenta sekä potinjakoalgoritmi, mikäli kyseiset algoritmit poikkeavat valmiista toteutuksesta.

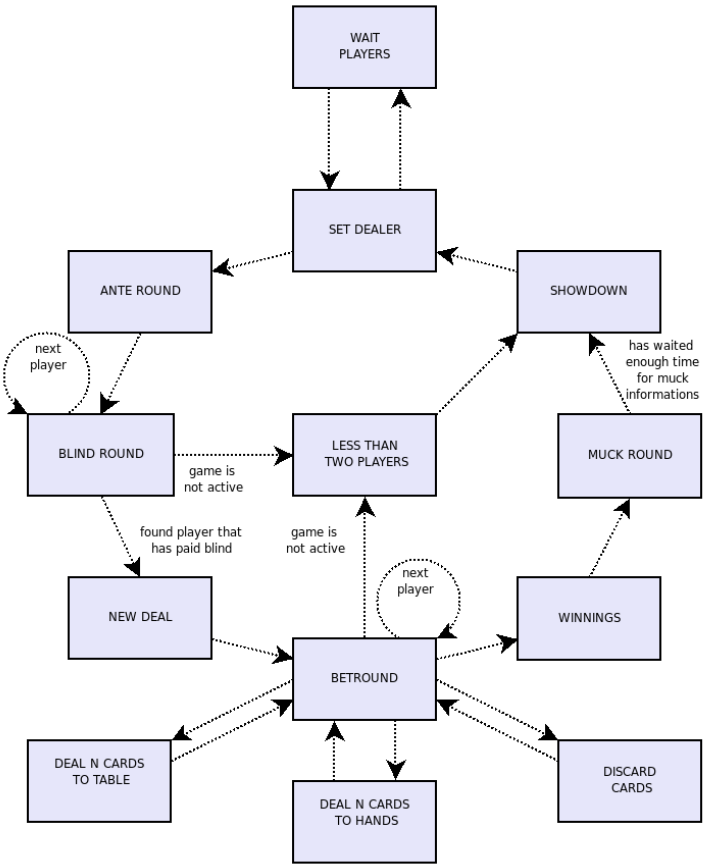
Tilakoneessa usein edellinen tila asettaa seuraavaksi toimivan pelaajan kohdalleen, jotta itseään toistavat tilat voidaan suorittaa useasti peräjälkeen.

Toimintaviestit asiakasohjelmistosta

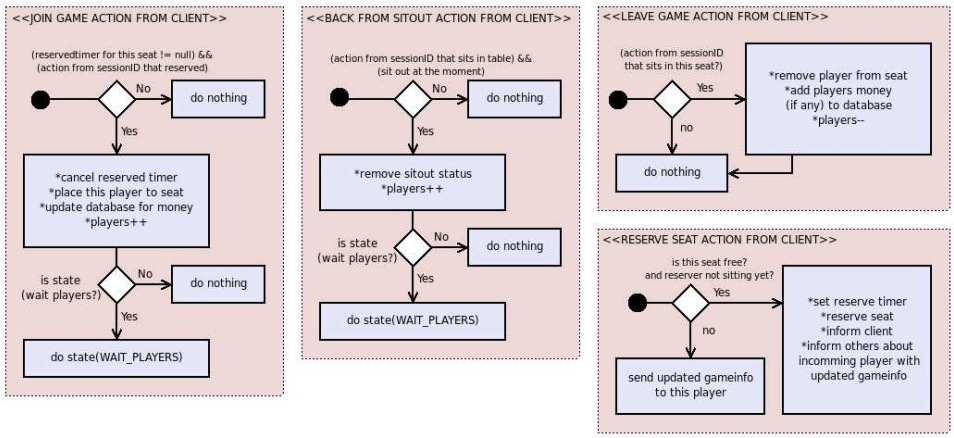
Pokerilogiikkatilakoneen tiloihin, kuten sokkopanosten keräys, panostuskierrokset, ja korttienvaihto, liittyy vuorovaikutus asiakkaan kanssa. Edellä mainituissa tiloissa pelilogiikka pyytää palvelimelta ajastinpalvelua, jonka ajan odotellaan pelaajalta pelitilanteeseen liittyvää toimintoa. Lisäksi on viestejä jotka voivat tulla missä tahansa tilakoneen tilassa, kuten peliin liittyminen, pelistä poistuminen. Kuvassa 3 on yleiskuva viesteistä, jotka voivat tulla mihin tahansa aikaan. Lisäksi kunkin tilan kuvassa on tietoa kyseisen tilan toiminnasta ja siihen liittyvästä viestinnästä.

waitplayers

Waitplayers on tila, jossa odotellaan minimimäärä pelaajia eikä peli ole käynnissä. Aina pelin alkaessa pelaajien minimimääräksi asetetaan kaksi. Pelin alkaessa odotuksen jälkeen



Kuva 2: Pokerilogiikkatilakone

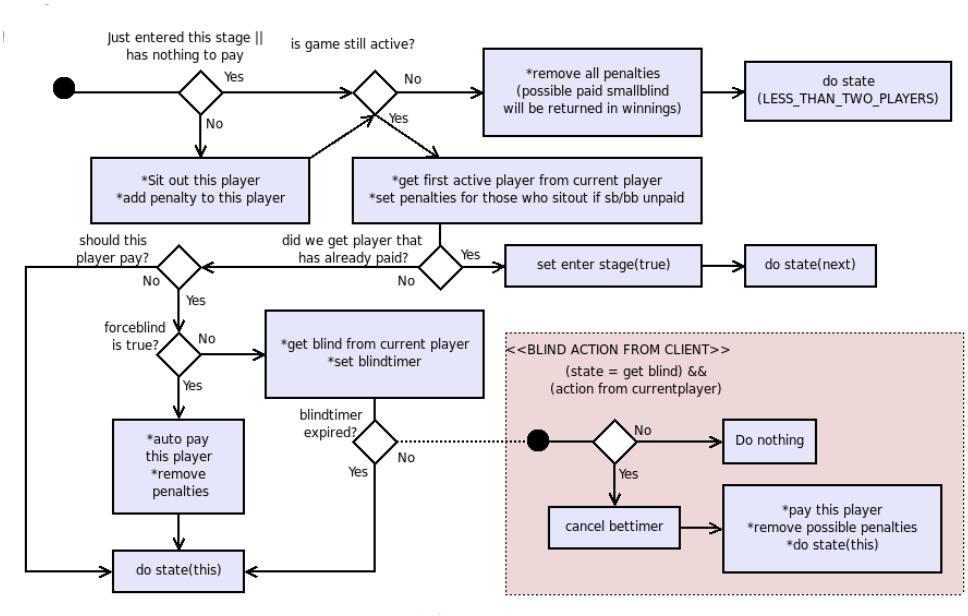


Kuva 3: Pokerilogiikka: Yleiset toimintaviestit

arvotaan uusi jakaja.

blind round

Blind round on tila, joka suoritetaan jokaiselle aktiiviselle pelaajalle pöydässä. Pelaajalta kysytään kuvan 4 mukaisesti maksettavia panostuksia. Lisäksi kysytään sakkopanokset tauolta palanneilta pelaajilta sekä peliin liittymiseen liittyviä panostuksia.



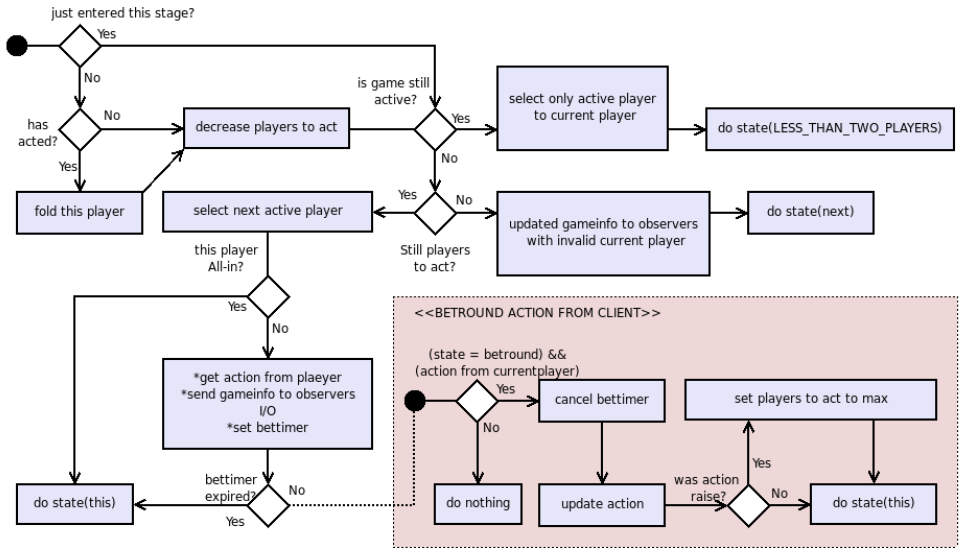
Kuva 4: Pokerilogiikka: blind round

new deal

Jakokierros suoritetaan pelin alussa ja aloitetaan luomalla uusi korttipakka. Korttipakasta jaetaan aktiivisille pelaajille käsikortit. Käsitiedoista paljastetaan pelaajille vain kortit, jotka ovat kyseiselle pelaajalle julkista tai henkilökohtaista tietoa. Muista korteista esitetään vain tausta. Jakokierroksen lopussa asetetaan pelitilaolion aktiiviseksi pelaajaksi panostuskierroksella ensimmäistä edellinen pelaaja.

bet round

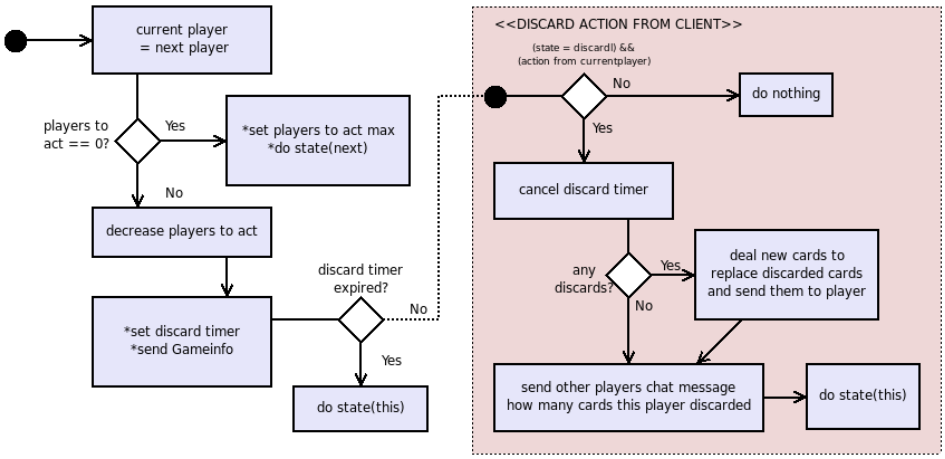
Panostuskierroksella kerätään panoksia kuvan 5 algoritmin mukaan. Tila on iteratiivinen ja suoritetaan jokaiselle aktiiviselle pelaajalle vuorollaan kunnes päästään lopputilaan. Lopputilassa kaikki ovat joko maksaneet vaaditut panostukset, heittäneet kortit pois tai panostaneet kaikkensa. Tämän jälkeen siirrytään seuraavaan tilaan. Mikäli pelissä on vain yksi pelaaja, on sekin lopputila ja siirrytään less-than-two-players tilaan.



Kuva 5: pokerilogiikka: bet round

discard round

Discard round eli korttienvaihtokierros ei kuulu texas holdemiin, mutta pokerilogiikka toteuttaa kyseisen tilan muita pelejä varten. Tilassa käydään kaikki aktiiviset pelaajat kerran läpi ja jaetaan poisheitettyjen korttien tilalle uudet kortit. Tilan vaiheet on esitetty kuvassa 6.



Kuva 6: Pokerilogiikka: discard round

deal card round

Korttienjako -tiloissa jaetaan pöytään tai pelaajille lisää kortteja, tämän jälkeen asetetaan pelitilaolion aktiiviseksi pelaajaksi seuraavassa tilassa ensimmäisenä toimivaa pelaajaa edellinen pelaaja. Pelaajan asetuksessa huomattavaa on, että *heads up* tilanteessa ei ensimmäisenä toimiva pelaaja ole aina buttonista seuraava aktiivinen pelaaja.

Less than two players

Tähän tilanteeseen tulessa pelissä on vähemmän kuin kaksi pelaajaa. Tilanteeseen voidaan päästä kahdesta eri tilasta, sokkopanoskierrokselta tai panostuskierroksilta. Sokkopanoskierrokselta tultaessa joko kukaan ei ole liittynyt peliin tai pelkästään toinen sokkopanoksista on maksettu, tällöin sokkopanos palautetaan ja siirrytään eteenpäin showdown tilaan.

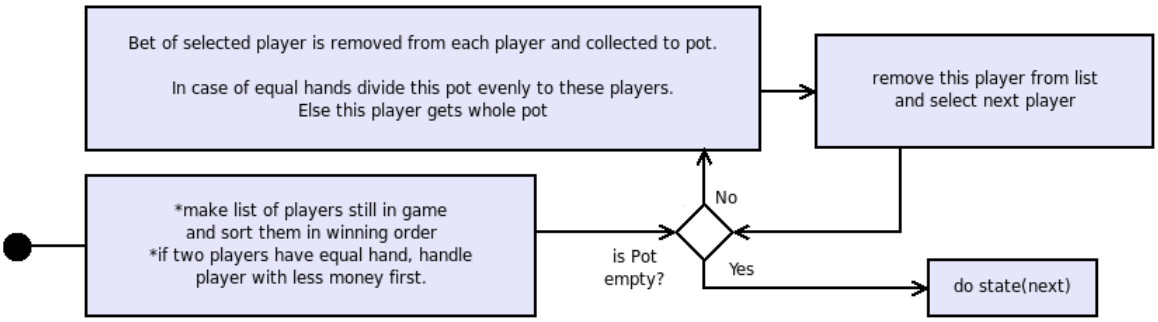
Panostuskierrokselta tultaessa tähän tilaan pelissä on enää yksi pelaaja joka voittaa kierroksen. Voittajaa varten asetetaan ajastin, jonka ajan hänellä on mahdollisuus näyttää tai piilottaa voittokortit muilta pelaajilta. Mikäli ajastin laukeaa, kortteja ei näytetä. Ajastimen laukeamisen tai korttien näyttämispäätöksen jälkeen siirrytään showdown tilaan.

winnings

Viiden kortin pokerikäden arvon laskemiseen on käytetty nimimerkillä ”Cactus Kev” esiintyvän henkilön kehittämää algoritmia, jonka kuvaus löytyy osoitteesta:

`http://www.suffecool.net/poker/evaluator.html`

Paras pokerikäsi seitsemästä kortista on laskettu sijoittamalla vuorotellen kaikki mahdolliset viiden kortin yhdistelmät algoritmiin. Kun pokerikädet on saatu, arvojärjestykseen jaetaan potti voittajien kesken kuvan 7 algoritmia käyttäen.



Kuva 7: Pokerilogiikka: kuinka voitto potti jaetaan

muck round

Voittajan ratkomisen jälkeen häviäjille suodaan vielä mahdollisuus kätkeä tai näyttää korttinsa muck-ajastimen ajan, jonka jälkeen siirrytään showdown-tilaan.

showdown

Showdown-tilassa lähetetään kaikille peliä seuraaville päivitetty pelitilanneolio ja odotetaan 4s, jotta ihmispelaajilla on aikaa omaksua lopputulos. Samalla kerätään tieto asiakasohjelmilta, ketkä pelaajista eivät osallistu seuraavalle kierrokselle.

8 Asiakasohjelmisto

Asiakasohjelmisto on loppukäyttäjän koneella pyörivä komponentti. Asiakasohjelmisto koostuu palvelimelle kirjautumisesta, peliaulasta, pelipöydästä sekä pelipöytään ja aulaan liittyvistä ohjain-olioista (GameController, LobbyController). Peliaula sekä pelipöytä toteuttavat ohjelmiston käyttöliittymän javan swing kirjastolla. Tietosisältö on upotettu Controller-olioihin, jotka kommunikoivat yleisen yhteysrajapinnan kautta palvelimen ja käyttötymä komponenttien välillä.

8.1 Peliaula

Käyttäjän kirjauduttua palvelimelle saavutaan peliaulaan. Peliaulan ylälaidassa hoidetaan uusien käyttäjienluonti, sisä- ja uloskirjautuminen sekä näytetään kuka on kirjautunut aulaan. Loput ikkunasta on jaettu kolmeen osaan. Vasemmalla sijaitsee pelivalikko palvelimella olevista pelityypeistä. Keskellä ruutua ovat pelilistaukset aktiivisista peleistä sekä listauksien tarkentamista helpottavaa ja toiminnallisuutta. Oikeassa laidassa ruutua on yksityiskohtaista tietoa valitusta pelistä. Jos yhtään peliä ei ole valittu, on oikeassa reunassa uuden pelin luontiin tarvittavat lomaketiedot. Keskellä olevaa pelilistaustietoa voidaan järjestää laskevaan ja nousevaan järjestykseen painamalla harmaan otsakerivin sarakkeiden nimiä. Lisäksi tyhjät ja täydet pöydät pystytään piilottamaan pelilistauksesta.

Oikealla laidassa sijaitsevasta yksityiskohtaisesta pelitiedosta selviää kuka pelaa valitussa pöydässä missäkin paikassa ja paljonko heillä pöydässä rahaa. Pelitietojen alla on esitetty pelin kuvaus sekä pelinliittymispainike, joka avaa valitun pelin uuteen ikkunaan. Kuvassa 8 on esitetty peliaulan graafisten komponenttien suhteet aulanhallinta-olioon sekä esitetty miten käyttöliittymäpaneelit sijoittuvat kehykseen. Kuvassa 9 on kuva itse peliaulasta.

Seuraavassa on kuvattu lyhyesti asiakasohjelmiston eri komponentit. Javadoc -dokumentaatio sisältää tarkemmat kuvaukset luokista.

- ConnectionHandler

  Java NIO -kirjastoa hyödyntävä yhteysluokka. Pyörii itsenäisessä säikeessä kuunnellen sockettiin tulevaa dataa ja sockettiin lähetettävän datan pyyntöjä. Käyttää avukeseen ChangeRequest olioita ohjatakseen socketin toimintaa lukemisen ja kirjoittamisen välillä. Sisältää metodit viestien lähettämiseen ja metodin viestien kuuntelijaksi rekisteröitymiseen. Rekisteröityneille kuuntelijoille välitetään viestejä niiden sessionID:n perusteella.

- Message

  Rajapinta, joka mahdollistaa ConnectionHandlerille saapuvien viestien kuuntelijaksi rekisteröitymisen. Sisältää rajapinta metodit, joiden avulla viestien välittäminen kuuntelijalle onnistuu.

- LobbyController

Asiakasohjelmiston aulaa ja pelihakua hallinnoiva osio. Tämän luokan kautta kulkevat viestit käyttöliittymäluokasta LobbyFrame palvelimelle ja takaisin. Huolehtii myös peliohjelmien käynnistämisestä.

- LobbyModel

  Pitää yllä aulaohjelman tietosisältöä. Tallettaa tuoreimmat päivitykset pelilistoista ja pitää yllä asiakkaan kirjautumista. Sisältää apumetodeita LobbyControllerille tietojen kaivamiseen palvelimelta lähetetyistä viesteistä. Tietoja käyttäjästä ylläpidetään luokan User avulla.

- LobbyFrame

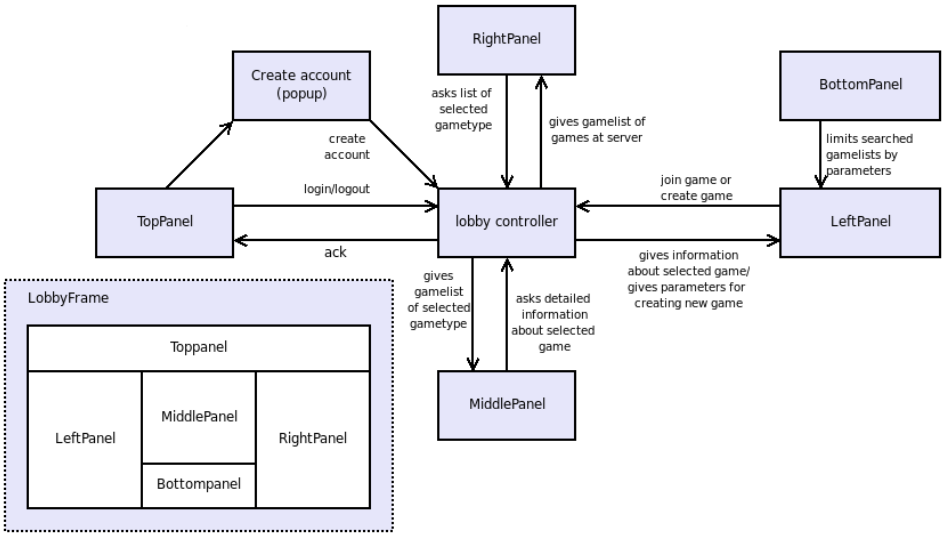
  Graafinen käyttöliittymä aulaohjelmalle. Komennot käyttöliittymästä kulkevat LobbyControllerille. Rakentuu useista osapaneeleista.

- PluginLoader

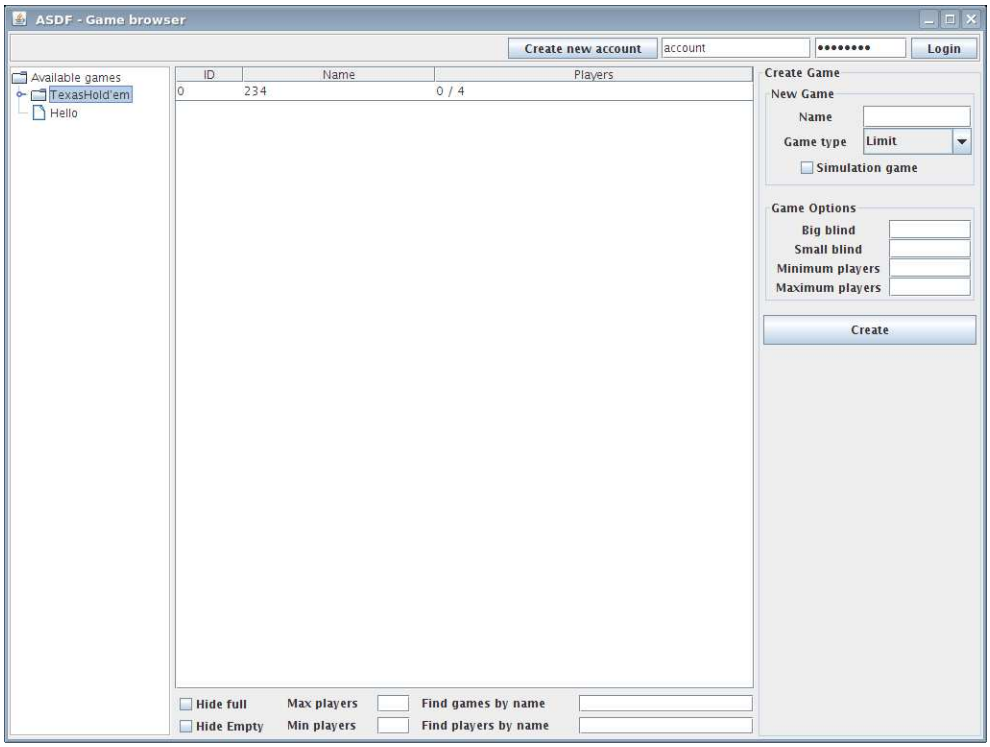
  Apuluokka, joka huolehtii luokkien dynaamisesta lataamisesta. Käytetään muun muassa peliohjelmien dynaamiseen lataamiseen ja pokeripelissä tekoälypelaajien lataamiseen. Configuration luokka pitää yllä hakemistot, joista lisäosat ja tekoälypelaajat tulee ladata.

- GameController

  Abstrakti yläluokka, jota laajentamalla voidaan toteuttaa uusia peliohjelmia, jotka aulaohjelma osaa ladata dynaamisesti. Sisältää useita valmiiksi toteutettuja metodeita ja abstraktit metodit, joiden avulla kommunikointi palvelimen välillä ConnectionHandler luokan kautta tulee tapahtumaan. Toteuttaa Message -rajapinnan.



Kuva 8: Peliaulan komponenttien suhteet toisiinsa.

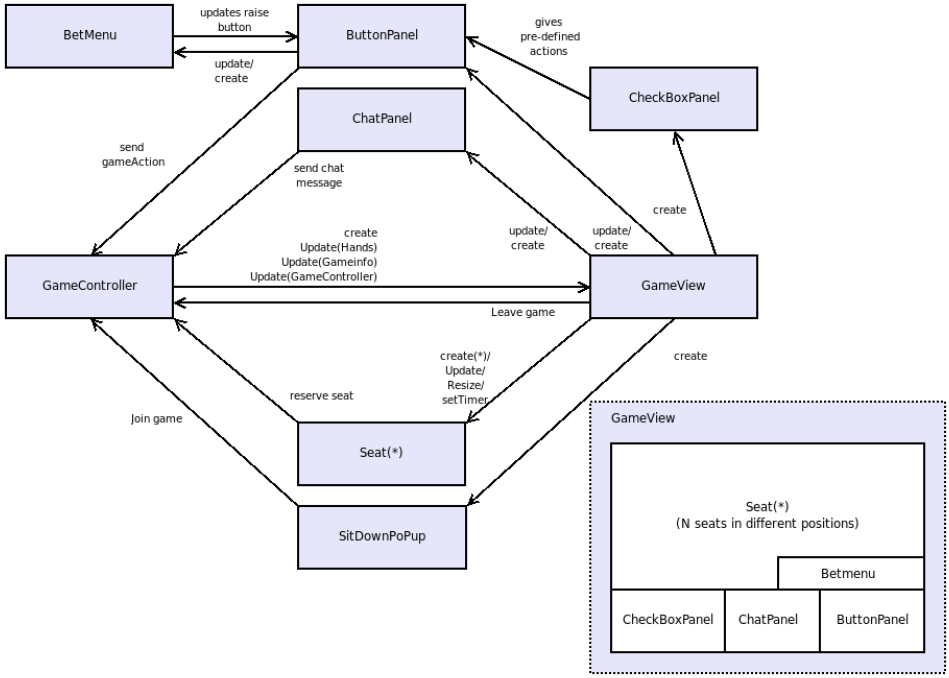


Kuva 9: Kuvankaappaus peliaulasta.

8.2 Pelipöytä

Ohjelmistossa on toteutettu pelipöytä pokeripeleille. Pelinhallintaolio välittää ajantasaista pelitietoa käyttöliittymäkomponenteille kuvan 10 esittämällä tavalla. Pelitilanteen hallintaan liittyvät graafiset komponentit on sijoitettu Javan Layoutmanagerien avulla. Kuitenkin pelinäkymään liittyvät peli-istuimet ja niihin läheisesti liittyvät informaatiota sisältävien tietorakenteiden esitykset, kuten ajastimet, jakajanapit, kortit, paikanvarausnapit sekä panostukset on sijoitettu kerrostusruutuun (JLayeredPane). Kerrostusruudun käyttö mahdollistaa dynaamisen istuimien sijoittelun sekä pelaajille jaettavan korttimäärän vaihtelun. Myös pelikuvakkeet kuten kortit, jakajanapit, jne. skaalauntuvat peli-ikkunan koonmuutosten mukaan, mahdollistaen useamman pelipöydän sovittamisen pienellekin näytölle.

Pelipöydän käyttöliittymän alareuna on jaettu kolmeen osaan. Vasemmanlaidan valintaruudut sisältävät etukäteistietoa sokkopanosten maksusta, *sitoutista* sekä voittavien ja häviävien korttien pois heitosta. Vasemmanlaidan painikkeista saa peliaulan näkyviin, tai suljettua pelipöydän. Keskellä alareunassa sijaitsee keskusteluviesti-ikkuna. Oikeassa alareunassa on näkyvissä pelissä oleva potti sekä nykyisen kierroksen tarvittava panostusmäärä sekä pelitilanteeseen liittyvät toimintapainikkeet. Toimintapainikkeita liittyy panostuskierrokseen, sokkopanoskierrokseen sekä pelaajan ollessa sitout tilassa. Oikeassa laidassa sijaitsee myös valintaruudut, joilla pelaaja voi antaa etukäteistietoa miten aikoo toimia omalla kierroksellaan. Kuvassa 11 on kuva pelipöydästä



Kuva 10: Pelipöydän komponenttien suhteet toisiinsa.

Kuva 11: Kuvankaappaus pelipöydästä.

9 Rajapintakuvaukset

- Game: Game -rajapinta määrittelee tavan, jolla yleisen pelin tulisi tehdä yhteistyötä palvelimen kanssa. Rajapinnan määrittelemät metodit antavat pelille mahdollisuuden käsitellä palvelimelta saapuvia viestejä ja lähettää niihin vastauksia. Lisäksi rajapinta ohjeistaa selkeällä tavalla kuinka pelaajia tulee liittymään peliin ja miten pelaajat voivat peliin vaikuttaa. Rajapinnan metodit peliohjelmoijan tulee kuitenkin toteuttaa itse.
- PokerGame: Rajapinta toteuttaa kaikkien pokeripelien haluamat toiminnot. Laajentamalla tätä luokkaa, voidaan helposti toteuttaa muita pokeripelejä.
- MessageHandler: Rajapinta, joka mahdollistaa asiakaspäässä olevien peliasiakasohjelmien kommunikoinnin palvelimen kanssa ConnectionHandler luokan kautta. MessageHandler rajapinnan toteuttavat luokat voivat rekisteröityä niille rajapinnan kautta annettuun yhteisolioon. Tällöin niille välitetään yhteys oliolle tulleita viestejä niiden sessiotunnisteen perusteella.
- GameController: Rajapinta määrittelee tavan, jolla peliaulasta voidaan käynnistää tietynpelin asiakasohjelma sekä miten viestinvälitys asiakasohjelman ja palvelimella toimivan pelilogiikan välillä tapahtuu. GameController toteuttaa MessageHandler -rajapinnan.
- PlayerInfo: Tarjoaa metodit pelaajakohtaisen julkisen tiedon pyytämiseen.
- GameModel: Tarjoaa metodit pelikohtaisen julkisen tiedon pyytämiseen.

Liite 1. Protokollakuvaukset

Table of Contents

Message

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="Message">
    <xs:complexType>
      <xs:choice>
        <xs:element name="Request" type="Request"/>
        <xs:element name="Response" type="Response"/>
        <xs:element name="StateUpdate" type="StateUpdate"/>
        <xs:element name="Info" type="RequireParameters"/>
      </xs:choice>
      <xs:attribute name="sessionID" type="xs:long" default="0"/>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="StateUpdate">
       <xs:sequence>
              <xs:choice>
              <xs:element name="Game" type="GameStateUpdate"/>
              <xs:element name="Lobby" type="LobbyStateUpdate"/>
                     <xs:element name="General" type="GenericStateUpdate"/>
              </xs:choice>
       </xs:sequence>
  </xs:complexType>

  <xs:complexType name="GameStateUpdate">
       <xs:sequence>
              <xs:choice>
                     <xs:element name="UpdatedGameState" type="GenericStateUpdate"/>
                     <xs:element name="Chat" type="ChatMessage"/>
                     <xs:element name="Voice" type="xs:string"/>
                     <xs:element name="Custom" type="Custom"/>
              </xs:choice>
       </xs:sequence>
       <xs:attribute name="gameID" type="xs:long" default="0"/>
  </xs:complexType>

  <xs:complexType name="ChatMessage">
         <xs:sequence>
                <xs:choice>
                      <xs:element name="Message" type="xs:string"/>
                </xs:choice>
         </xs:sequence>
         <xs:attribute name="sender" type="xs:string" use="required"/>
         <xs:attribute name="type" type="xs:int" default="0"/>
  </xs:complexType>

  <xs:complexType name="LobbyStateUpdate">
       <xs:sequence>
              <xs:choice>
                     <xs:element name="UpdatedLobbyState" type="GenericStateUpdate"/>
                     <xs:element name="Custom" type="Custom"/>
              </xs:choice>
       </xs:sequence>
  </xs:complexType>

  <xs:complexType name="GenericStateUpdate">
     <xs:sequence>
      <xs:element name="Description" minOccurs="0" type="xs:string"/>
      <xs:element name="Parameters" type="Parameters" minOccurs="0"/>
         <xs:element name="Custom" minOccurs="0" type="Custom"/>
    </xs:sequence>
    <xs:attribute name="success" type="xs:boolean" use="required"/>
    <xs:attribute name="errorCode" type="xs:long" use="optional"/>
  </xs:complexType>

  <xs:complexType name="RequireParameters">
    <xs:sequence>
      <xs:element name="Parameters" type="Parameters"/>
```

```
  </xs:sequence>
</xs:complexType>

<xs:complexType name="OptionalParameters">
  <xs:sequence>
    <xs:element name="Parameters" type="Parameters" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Parameters">
  <xs:sequence>
    <xs:any processContents="skip" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="RequireStringParameters">
  <xs:sequence>
    <xs:element name="String" type="xs:string" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="OptionalStringParameters">
  <xs:sequence>
    <xs:element name="String" minOccurs="0" type="xs:string" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Request">
  <xs:choice>
    <xs:element name="AccountManagement" type="AccountManagementRequest"/>
    <xs:element name="Authentication" type="AuthenticationRequest"/>
    <xs:element name="Game" type="GameRequest"/>
    <xs:element name="Lobby" type="LobbyRequest"/>
  </xs:choice>
</xs:complexType>

<xs:complexType name="AccountManagementRequest">
  <xs:choice>
    <xs:element name="Create">
      <xs:complexType>
        <xs:attribute name="accountName" type="xs:string" use="required"/>
        <xs:attribute name="password" type="xs:string" use="required"/>
        <xs:attribute name="accountType" type="xs:int" use="required"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="Delete">
      <xs:complexType>
        <xs:attribute name="accountName" type="xs:string" use="required"/>
        <xs:attribute name="password" type="xs:string" use="required"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="ChangePassword">
      <xs:complexType>
        <xs:attribute name="accountName" type="xs:string" use="required"/>
        <xs:attribute name="oldPassword" type="xs:string" use="required"/>
        <xs:attribute name="newPassword" type="xs:string" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>

<xs:complexType name="AuthenticationRequest">
  <xs:choice>
    <xs:element name="Login">
      <xs:complexType>
        <xs:attribute name="accountName" type="xs:string" use="required"/>
        <xs:attribute name="password" type="xs:string" use="required"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="Logout"/>
  </xs:choice>
</xs:complexType>

<xs:complexType name="GameRequest">
  <xs:choice>
    <xs:element name="Action" type="RequireParameters"/>
    <xs:element name="Create" type="CreateGameRequest"/>
```

```
    <xs:element name="Delete" type="OptionalParameters"/>
    <xs:element name="Join" type="OptionalParameters"/>
    <xs:element name="Part" type="OptionalParameters"/>
    <xs:element name="Chat" type="ChatMessage"/>
        <xs:element name="Voice" type="xs:string"/>
    <xs:element name="Custom" minOccurs="0" type="Custom"/>
  </xs:choice>
  <xs:attribute name="gameID" type="xs:long" default="0"/>
</xs:complexType>

   <xs:complexType name="Custom">
    <xs:sequence>
       <xs:any processContents="skip"/>
    </xs:sequence>
   </xs:complexType>

<xs:complexType name="CreateGameRequest">
  <xs:sequence>
    <xs:element name="Parameters" type="Parameters" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="gameType" type="xs:string" use="required"/>
  <xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="LobbyRequest">
  <xs:choice>
    <xs:element name="GameList">
      <xs:complexType>
        <xs:attribute name="gameType" type="xs:string" default="com.asdf.games.Game"/>
        <xs:attribute name="gameVariant" type="xs:string"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="AvailableGames"/>
    <xs:element name="GameDetails">
      <xs:complexType>
        <xs:attribute name="gameID" type="xs:long" default="0"/>
      </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>

<xs:complexType name="Response">
  <xs:choice>
    <xs:element name="AccountManagement" type="AccountManagementResponse"/>
    <xs:element name="Authentication" type="AuthenticationResponse"/>
    <xs:element name="Game" type="GameResponse"/>
    <xs:element name="Lobby" type="LobbyResponse"/>
    <xs:element name="General" type="GenericResponse"/>
  </xs:choice>
</xs:complexType>

<xs:complexType name="LobbyResponse">
  <xs:choice>
    <xs:element name="GameDetails" type="GameDetailsResponse"/>
    <xs:element name="GameList" type="GameListResponse"/>
    <xs:element name="AvailableGames" type="AvailableGamesResponse"/>
  </xs:choice>
</xs:complexType>

<xs:complexType name="GameDetailsResponse">
  <xs:sequence>
    <xs:element name="ColumnNames" type="OptionalStringParameters" minOccurs="0"/>
    <xs:element name="Players" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Player" maxOccurs="unbounded" minOccurs="0">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Info" type="OptionalParameters"/>
              </xs:sequence>
               <!--<xs:attribute name="type" default="-">
                  <xs:simpleType>
                      <xs:restriction base="xs:string">
                          <xs:enumeration value="human"/>
                          <xs:enumeration value="bot"/>
                          <xs:enumeration value="-"/>
                      </xs:restriction>
```

```
                    </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="name" use="required"/>-->
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Rules" type="OptionalStringParameters" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="GameListResponse">
    <xs:sequence>
    <!-- ColumnNames defines which parameters of Other in AdditionalInfo will be listed. -->
      <xs:element name="ColumnNames" type="OptionalStringParameters"/>
      <xs:element name="Games">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Game" maxOccurs="unbounded" minOccurs="0">
              <xs:complexType>
                <xs:sequence>
                <!-- Other optional info to be listed. -->
                    <xs:element name="AdditionalInfo" minOccurs="0">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="Other" type="OptionalParameters"/>
                          <!-- Players playing this game for player search. -->
                          <xs:element name="Players" minOccurs="0">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element name="Player" minOccurs="0" maxOccurs="unbounded"
type="xs:string"/>
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                <!-- Required info to be listed. This info is used by game search filters.-->
                <xs:attribute name="gameID" type="xs:long" use="required"/>
                <xs:attribute name="players" type="xs:int" use="required"/>
                <xs:attribute name="maxPlayers" type="xs:int" use="required"/>
                <xs:attribute name="name" type="xs:string" use="required"/>
                <xs:attribute name="gameType" type="xs:string" use="required"/>
                <xs:attribute name="gameVariant" type="xs:string" use="required"/>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="requestedGameType" type="xs:string" use="required"/>
          <xs:attribute name="requestedGameVariant" type="xs:string"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="GameListGameResponse">
    <xs:sequence>
      <xs:element name="Info" type="RequireParameters"/>
    </xs:sequence>
    <xs:attribute name="gameID" type="xs:long" default="0"/>
  </xs:complexType>

  <xs:complexType name="AvailableGamesResponse">
    <xs:sequence>
      <xs:element name="Game" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Variants" type="OptionalStringParameters" minOccurs="0"/>
          </xs:sequence>
          <xs:attribute name="type" type="xs:string" use="required"/>
          <xs:attribute name="clientGame" type="xs:string" use="required"/>
                <xs:attribute name="createGamePanel" type="xs:string" use="required"/>
                <xs:attribute name="serverGame" type="xs:string" use="required"/>
        </xs:complexType>
      </xs:element>
```

```
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="PlayerList">
    <xs:sequence>
      <xs:element name="Player" maxOccurs="unbounded" type="RequireParameters"/>
    </xs:sequence>
  </xs:complexType>
```

Texas Message

```
  <xs:complexType name="AccountManagementResponse">
    <xs:choice>
      <xs:element name="Create" type="GenericResponse"/>
      <xs:element name="Delete" type="GenericResponse"/>
      <xs:element name="ChangePassword" type="GenericResponse"/>
    </xs:choice>
  </xs:complexType>

  <xs:complexType name="AuthenticationResponse">
    <xs:choice>
      <xs:element name="Login" type="LoginResponse"/>
      <xs:element name="Logout" type="GenericResponse"/>
    </xs:choice>
  </xs:complexType>

  <xs:complexType name="GameResponse">
    <xs:choice>
      <xs:element name="Action" type="GenericResponse"/>
      <xs:element name="Create" type="CreateGameResponse"/>
      <xs:element name="Delete" type="GenericResponse"/>
      <xs:element name="Join" type="GenericResponse"/>
      <xs:element name="Part" type="GenericResponse"/>
      <xs:element name="Custom" type="Custom"/>
      <xs:element name="Event" type="GenericResponse"/>
    </xs:choice>
    <xs:attribute name="gameID" type="xs:long" default="0"/>
  </xs:complexType>

  <xs:complexType name="GenericResponse">
    <xs:sequence>
      <xs:element name="Description" minOccurs="0" type="xs:string"/>
      <xs:element name="Parameters" type="Parameters" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="success" type="xs:boolean" use="required"/>
    <xs:attribute name="errorCode" type="xs:long" use="optional"/>
  </xs:complexType>

  <xs:complexType name="CreateGameResponse">
    <xs:sequence>
      <xs:element name="Description" minOccurs="0" type="xs:string"/>
      <xs:element name="Parameters" type="Parameters" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="success" type="xs:boolean" use="required"/>
    <xs:attribute name="errorCode" type="xs:long" use="optional"/>
    <xs:attribute name="gameID" type="xs:long"/>
  </xs:complexType>

  <xs:complexType name="LoginResponse">
    <xs:sequence>
      <xs:element name="Description" minOccurs="0" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="success" type="xs:boolean" use="required"/>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="type" type="xs:short" use="required"/>
    <xs:attribute name="balance" type="xs:double"/>
  </xs:complexType>
</xs:schema>
```

Texas Message

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:element name="TexasMessage">
		<xs:complexType>
			<xs:choice>
				<xs:element name="Action" type="ActionMessage"/>
				<xs:element name="GameInfo" type="GameModelInfoMessage"/>
				<xs:element name="Hand" type="ParserHand"/>
				<xs:element name="Event" type="TexasEventMessage"/>
			</xs:choice>
		</xs:complexType>
	</xs:element>
	<xs:complexType name="ActionMessage">
		<xs:sequence>
			<xs:element name="index" type="xs:int" minOccurs="0"/>
			<xs:element name="amount" type="xs:double" minOccurs="0"/>
			<xs:element name="discards" type="xs:int" minOccurs="0"/>
			<xs:element name="discarded" type="xs:int" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
			<xs:element name="name" type="xs:string" minOccurs="0"/>
			<xs:element name="seat" type="xs:int" minOccurs="0"/>
		</xs:sequence>
	</xs:complexType>

	<xs:complexType name="TexasEventMessage">
		<xs:sequence>
			<xs:element name="CardPositions" type="xs:int" minOccurs="0"
maxOccurs="unbounded"/>
			<xs:element name="Action" type="ActionMessage" minOccurs="0"/>
		</xs:sequence>
		<xs:attribute name="type" type="xs:int" use="required"/>
		<xs:attribute name="amount" type="xs:double" default="0.0"/>
		<xs:attribute name="seat" type="xs:int" use="required"/>
	</xs:complexType>

	<xs:complexType name="PlayerInfoFieldsMessage">
		<xs:sequence>
			<xs:element name="bankRoll" type="xs:double"/>
			<xs:element name="bankRollAtStartOfHand" type="xs:double"/>
			<xs:element name="hasActedThisRound" type="xs:boolean"/>
			<xs:element name="hasToPostBlind" type="ParserBlind" minOccurs="0"/>
			<xs:element name="holeCards" type="ParserHand" minOccurs="0"/>
			<xs:element name="inPot" type="xs:double"/>
			<xs:element name="inPotThisRound" type="xs:double"/>
			<xs:element name="lastAction" type="xs:int"/>
			<xs:element name="lastGameInfo" type="GameModelInfoMessage" minOccurs="0"/>
			<xs:element name="myName" type="xs:string"/>
			<xs:element name="mySeat" type="xs:int"/>
			<xs:element name="myState" type="xs:int"/>
			<xs:element name="accountName" type="xs:string"/>
		</xs:sequence>
	</xs:complexType>

	<xs:complexType name="ParserHand">
		<xs:sequence>
			<xs:element name="cards" type="ParserCard" nillable="true" minOccurs="0"
maxOccurs="unbounded"/>
		</xs:sequence>
		<xs:attribute name="seat" type="xs:int" use="required"/>
	</xs:complexType>

	<xs:complexType name="ParserCard">
		<xs:sequence>
			<xs:element name="card" type="xs:int"/>
		</xs:sequence>
	</xs:complexType>

	<xs:complexType name="GameModelInfoMessage">
		<xs:sequence>
			<xs:element name="ante" type="xs:int"/>
			<xs:element name="bigBlind" type="xs:double"/>
```

```
                        <xs:element name="boardCards" type="ParserHand" minOccurs="0"/>
                        <xs:element name="buttonSeat" type="xs:int"/>
                        <xs:element name="currentPlayerSeat" type="xs:int"/>
                        <xs:element name="gameID" type="xs:long"/>
                        <xs:element name="limitStyle" type="xs:int"/>
                        <xs:element name="minPlayers" type="xs:int"/>
                        <xs:element name="minRaise" type="xs:double"/>
                        <xs:element name="players" type="PlayerInfoFieldsMessage" nillable="true"
minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element name="playersToAct" type="xs:int"/>
                        <xs:element name="pot" type="xs:double"/>
                        <xs:element name="raisesThisRound" type="xs:int"/>
                        <xs:element name="reverseBlinds" type="xs:boolean"/>
                        <xs:element name="simulation" type="xs:boolean"/>
                        <xs:element name="smallBlind" type="xs:double"/>
                        <xs:element name="stage" type="xs:int"/>
                        <xs:element name="zipMode" type="xs:boolean"/>
                        <xs:element name="winners" type="xs:int"/>
                        <xs:element name="bigBlindSeat" type="xs:int"/>
                        <xs:element name="smallBlindSeat" type="xs:int"/>
                </xs:sequence>
        </xs:complexType>

        <xs:simpleType name="ParserBlind">
                <xs:restriction base="xs:string">
                        <xs:enumeration value="SMALL_BLIND"/>
                        <xs:enumeration value="BIG_BLIND"/>
                        <xs:enumeration value="BOTH_BLINDS"/>
                        <xs:enumeration value="NO_BLINDS"/>
                </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

Plugins Client

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Addons">
	<xs:complexType>
		<xs:sequence>
		<xs:element name="Plugins">
			<xs:complexType>
				<xs:sequence>
					<xs:element name="Plugin" minOccurs="0" maxOccurs="unbounded">
					<xs:complexType>
					<xs:attribute name="uid" type="xs:string" use="required"/>
					<xs:attribute name="package" type="xs:string" use="required"/>
					<xs:attribute name="gameController" type="xs:string"
use="required"/>
					<xs:attribute name="createGamePanel" type="xs:string"
use="required"/>
					</xs:complexType>
					</xs:element>
				</xs:sequence>
			</xs:complexType>
		</xs:element>
		<xs:element name="AI">
		</xs:element>
		</xs:sequence>
	</xs:complexType>
</xs:element>
</xs:schema>
```

Liite 2. Asennusohje

Short manual for ASDF - pokeripalvelin

Requirements

- Java JRE version 6 or newer. You can get Java from: http://www.java.com/getjava/
- Runs at least on Linux and Windows XP, but should run on any platform with Java JRE version 6 or newer.

Installation

Just unzip ASDF-pokeripalvelin.zip to any directory.

Running

1. **Server**

   You can run 'pokeripalvelin' server in console in Linux or in command prompt in Windows. Just go into the directory you just unzipped ASDF-pokeripalvelin.zip and type:

   java -jar bin/server.jar

   Alternatively you can run server in Linux by: server.sh

   Note that it runs server as a background process.

   Or in Windows you can use: server.bat

2. **Client**

   You can run 'pokeripalvelin' client in console in Linux or in command prompt in Windows. Just go into the directory you just unzipped ASDF-pokeripalvelin.zip and type:

   java -jar bin/client.jar

   Alternatively you can run server in Linux by: client.sh

   Or in Windows you can use: client.bat

3. **Administration tool**

   There is a administration tool but we don't cover it in this manual. If you consider running your own 'pokeripalvelin' server, you should learn to use it.

   It is made with dot.net framework which means that if you want to run it in Linux you need to have mono framework installed. See http://www.mono-project.com/ for further details.

   In Windows you can run it just by double clicking.

Configuring Server

'pokeripalvelin' game server needs an SQL database to function. SQL database is used to store user accounts and their balance. It is recommended to use PostgreSQL database, since 'pokeripalvelin' is configured ready to work with it. But you can use other databases with a little extra effort. 'pokeripalvelin' is configured to use brutopia's PostgreSQL database server which might be online, but there are no guarantees so it is recommended to start your own database server.

Server.conf file is found in the root directory of 'ASDF-pokeripalvelin' and looks like this:

```
loggingconf = "logging.properties"
pluginpaths = file:../Plugins/bin/poker.jar,file:../Plugins/bin/hello.jar

# Authentication database settings
# This database contains accounts and their balances and is
# only used from server
auth_db_conn=jdbc:postgresql://brutopia.mine.nu:80/account
auth_db_username=ernesti
auth_db_password=salakala

# Game logging database
# This database is used with game plugins
game_db_conn=jdbc:postgresql://brutopia.mine.nu:80/statistics
game_db_username=ernesti
game_db_password=salakala
```

It should be a pretty straightforward process to set those parameters after you have configured your database correctly. See tutorials and documentation of PostgreSQL (or any other database you would like to use) for installation instructions.

You can find information about PostgreSQL and download it here if you choose to use it:

http://www.postgresql.org/

1. The create table statement for user accounts:

```
CREATE TABLE account (
name varchar(256),
password char(32),
type short,
balance decimal,
PRIMARY KEY(name)
);
```

2. And if you want to configure a database for statistics, here are two create table statements you need:

```
DROP TABLE hand;
DROP TABLE player;

CREATE TABLE hand (
      hand_id integer NOT NULL,
      timestamp timestamp NOT NULL,
      dealer smallint NOT NULL,
      flop varchar(256),
      turn varchar(256),
      river varchar(256),
      showdown varchar(256),
      card1 char(3),
      card2 char(3),
      card3 char(3),
      card4 char(3),
      card5 char(3),
      PRIMARY KEY (hand_id)
);
```

```
CREATE TABLE player (
      hand_id integer NOT NULL,
      account varchar(256) NOT NULL,
      name varchar(256) NOT NULL,
      position smallint NOT NULL,
      flop varchar(256),
      turn varchar(256),
      river varchar(256),
      money_in decimal,
      winnings decimal,
      card1 char(3),
      card2 char(3),
      PRIMARY KEY (hand_id, account, name)
);
```

Adding a new game plugins

Since our 'pokeripalvelin' is made from the start to be extremely versatile and extensible it is possible to add new game plugins to be used with the server. That means 'pokeripalvelin' is also able to play other games than only poker games. If you got interested in developing a new game to be used with 'pokeripalvelin' see Plugin_tutorial.pdf for more details.

In this manual we only look into how to add those plugins into your game directory. That should be pretty straightforward if the maker of the plugin has included instructions with it. If not, then just unzip/unrar/etc. The plugin into subdirectory called 'plugins' that should be found in the directory where you have installed 'pokeripalvelin'. Now you should have a nameoftheplugin.jar file in the plugins directory. After that you should check also that in subdirectory called 'meta' of directory 'plugins' contains nameoftheplugin.meta file.

Adding a new bot

It is possible to use AI players with our poker games implementation so it is a brilliant platform for poker AI development. Bots are added exactly the same way as game plugins. Only that bots are added in subdirectory called 'bots' in 'pokeripalvelin' directory. They also have the same kind of meta file in 'meta' subdirectory of 'bots' directory.

Liite 3. Plugins tutorial

PokeriPalvelin plugin tutorial

Hands-on guide to development and deployment of plugins

Contents

Overview

PokeriPalvelin is a client-server software allowing users to play different kind of multiplayer games. Software is designed from the ground up to be extensible on every aspect. There are few designed points for extension on the architecture but user is at the end free to replace any component if needed.

Typical plugins allow new game types or variations for existing ones. PokeriPalvelin is distributed with plugins for certain poker variants. Poker plugin itself also contains interface for programming new bot players. Software is written in Java and it is encouraged to be used to extend it. In the case where other languages are desired to be used unsupported options are available. There are few popular languages that compile to Java bytecode such as Scala and Groovy. These languages can be used directly with extension API and in case of other languages Java offers JNI (Java Native Interfaces) library to access native languages outside of JVM. Other options include using JSR-223 (Scripting for the Java Platform) that offers easy interfaces to certain scripting languages.

This document describes how to develop and deploy plugins using predefined extension points with Java. Programming poker bots and new variants are also covered.

Prerequisites

Interface API jar package

When project's main sources are compiled ant script generates extension API package named extension_api.jar in the directory Plugins/lib. This package contains interfaces required to develop new plugins.

Compilation

Compiling new plugins requires only Java SDK and ant version 1.70 or newer if you want to use our build files to compile your plugins. Plugins directory distributed with the system use Java 5 EE webservices-tools.jar package containing XJC ant task used to generate Java code directly from XML schema files.

Development

This document assumes that user is using Java as a language and is using Plugins directory as template for new plugins.

New poker bot AI

New bot is done by implementing com.asdf.plugins.pokergames.Player interface. Interface definition is documented with Javadoc styled comments that explain purpose of every to be implemented method. There exists a method for every interesting event happening in poker game. Bot is queried for action with the getAction() method. Bot is passed information about game events through various methods defined by the Player interface. You should read through very carefully our API documentation for extensions. Also we have included two example bots that can be used as a reference.

New poker variant

New poker variant is done by extending abstract com.asdf.plugins.pokergames.PokerGame class. User is free to override any method needed. There are few abstract methods that are mandatory. In addition of PokerGame user is required to implement com.asdf.gui.lobby.CreateGamePanel to have custom create game panel in client portion of the application.

Package com.asdf.plugins.pokergames.gui contains a graphical poker table implementation which should work with most common poker variants without any modifications. You should consider using it with your own poker game implementation at client side.

New game

New game is developed by inheriting com.asdf.common.Game class. Class contains methods for receiving GameRequest and GameResponse objects. Games are sending responses with send method found from Game class. On join games get client session ID, which is used to identify different players and clients. On send session ID is required to be specified. Games should not be starting own threads. Everything should be done by storing game state in it's member variables. To use timer functions developer must use setTimer methods from the Game class. These timers synchronize thread access properly.

In addition developer must inherit com.asdf.client.GameController class that defines logic for parsing messages on the client side and CreateGamePanel described above.

Deployment

Poker variants and bots should be packaged to existing poker.jar package. This happens automatically if you place your extensions on existing Plugins sources in the package com.asdf.plugins or it's subpackages it's automatically included in poker.jar. If you place it outside you must modify the compile task in the Plugins/build.xml ant script. New games can copy Plugins structure and change appropriate names in build.xml file.

Alternatively you can compile your bots or game plugins anyway you like. Then you should add them to .jar package and place the jar files in /bots or /plugins directory. It is not strictly necessary to add your class files in jar package. Placing your class files in subdirectory of plugins or bots will also work if you add that path to to client.conf and server.conf

There must be plugins/meta directory containing .meta files that describe available plugins in the working directory where application is started. There exists info.txt in that directory describing exact syntax for meta files. Basically files include fully qualified class names for Game, GameController and CreateGamePanel implementations and possible variants of the plugin.

Poker bots should be described in bots/meta directory also in application's working directory. Meta files contain fully qualified name of the bot and human readable name visible in GUI.

Client.conf and server.conf should contain entries for new plugins if placed somewhere else than in existing jar files.

Example: Hello game

Hello game distributed in plugins directory contains very basic implementations of all required interfaces. You can use these classes as a base for your own implementation.

1. Directory structure

   Lets look at the directory structure of our game first:

   pokeripalvelin – this is the workin directory

   pokeripalvelin\bin – this directory contains the jar files of the main program, server.jar, client.jar

   pokeripalvelin\lib – contains library files, including plugins_api.jar, which you should get familiarized with

   pokeripalvelin\plugins – this is the directory where all the plugins go and it should have hellogame.jar in it

   pokeripalvelin\plugins\meta – here are the metafiles for plugins and it should have hellogame.meta in it

   pokeripalvelin\bots - this is the directory where all the bots go (we do not need to care about that in this tutorial)

   pokeripalvelin\bots\meta - this is the directory where all the metafiles for bots go (we do not need to care about that in this tutorial)

2. Meta file

   Our meta file for hello game, *hellogame.meta*, is in directory *\plugins\meta* and it looks like this:

```
# Example game provided by ASDF-group with the server/client software
#
gamename Hello
panelname com.asdf.plugins.hello.CreatePanel
servergame com.asdf.plugins.hello.ServerSideHelloGameLogic
clientgame com.asdf.plugins.hello.HelloGame
variantnames ExampleGame
```

   After the comments, marked with # is the interesting stuff. We will get to all of these components in detail later, for now we just introduce them shorty.

   - **gamename** is used by client and server when querying this game and also this is what is shown to clients in the lobby.
   - **panelname** is the fully qualified classname of the CreateGamePanel implementation (a class extending com.asdf.client. CreateGamePanel). This is read from the metafile and send by server to clients when available games are queried. You do not need to implement your own CreateGamePanel, but it is strongly recommended since otherwise it will be impossible to create games to our server.
   - **servergame** is the fully qualified classname of the server-side Game implementation. This is the logic that actually runs the game in the server. Or at least the class that starts it. This tells server which class to load when client creates a new hellogame.

- **clientgame** is the fully qualified classname of the client-side GameController implementation. This is the game client that runs in the client side. GameController does not actually need to contain full game logic as server-side Game already does so. GameController only needs to take clients input, send it to server which runs the actual game and receive moves made by other player or whatever. But you could run the full game logic at client-side in GameController and then the server-side Game would act as a hub between clients. This tells lobby which class to load when client joins a game of this type running at server.
- **variantnames** is not strictly required but if you have different variants of your game you can use this field to tell server which those are. These will be shown at lobby and can be used by clients to create games of different variants and also filter results of game search. Note that you can give several **variantnames**, you only need to separate them with whitespace.

3. The main components

This chapter deals with the actual implementation of our interfaces and abstract classes when creating a new game for our server. We start from the server-side and then move on to the client-side. Example code will be shown along with comments explaining what is what, what it does and why it is necessary. Full example code of hellogame is included with the distribution package.

1. com.asdf.plugins.hello.ServerSideHelloGameLogic

This one is quite obvious starting point when creating a new game to this server from scratch – Using the abstract skeleton Game which the creators of this server have kindly made for us.

```
public class ServerSideHelloGameLogic extends Game {
```

`columnNames()` is used to list those columns we want to show at the gamelist for clients. `description()` is used to to set the information contained by those columns. This is done per game basis so that every game fills its own details to an `com.asdf.parser.GameListResponse.Games.Game` which is then returned to the server. You should check details about which fields you need to fill and which are optional from our protocol description message.xsd when developing your own game. If you forget to fill a mandatory field, don't worry. Our schema parser will tell you where you made a mistake when you first try to run or compile your game.

```
@Override
protected String[] columnNames() {
      return new String[]{"Jou", "Jeah", "Jaa", "123"};
}
```

```
/**
 * A small description about the game to the game listing.
 * The bare minimum of information is sent for the purposes of
 * clarity.
 */
@Override
protected com.asdf.parser.GameListResponse.Games.Game description(){

      com.asdf.parser.GameListResponse.Games.Game game = new
com.asdf.parser.GameListResponse.Games.Game();
      game.setGameID(this.getId());
      game.setName(this.getName());
      game.setGameType("Example game");
      game.setMaxPlayers(1);
      game.setPlayers(0);

      // Variant name is required, and affects the categories
      //in client game tree, in which this game can be found.
      game.setGameVariant("ExampleGame");

      return game;
}
```

`details(GameDetailsResponse a)` works the same way as `description()` above, except it is passed an object of type GameDetailsResponse which it needs to fill with the information we want to be displayed when client asks details about this game.You should check details about which fields you need to fill and which are optional from our protocol description message.xsd when developing your own game. If you forget to fill a mandatory field, don't worry. Our schema parser will tell you where you made a mistake when you first try to run or compile your game.

```
/**
 * Sends details about this game instance to the clients.
 * The example is not very interested in telling things about
 * it self,
 * as there is very little to say. So the bare minimum of
 * information is sent.
 */
@Override
protected void details(GameDetailsResponse a) {
      OptionalStringParameters b = new
OptionalStringParameters();
      Players plrs = new Players();

      a.setColumnNames(b);
      a.setPlayers(plrs);
      a.setRules(b);
}
```

Rest of the methods are quite well covered by the comments in the code and they actually are very game specific and might do whatever you like them to do. So there is no further need to comment these.

```
/**
 * Kills the game program. If a game is deemed as trash (no
 * players playing it for one minute)
 * it is killed [note that when a game is created, it has an
 * extra 1 minute of time when it is
 * not checked for trashyness at all, giving it a minimum of 2
 * minutes life time].
 * This method is called to give the game a chance to make any
 * final deeds
 * that might be necessary for proper client service.
 */
@Override
public void die() {
      // I'm ready to die ANYTIME. As soon as Server deems me
      // as trash,
      // I will die. (About 2 minutes after creation)
}

/**
 * User sends a request. Reacting to those requests should be
 * done in this method.
 * In this example we just check if the client wants to join
 * the game, and if so,
 * send a response that he is welcome to start the example
 * application. The client is not registered as a player,
 * as this example game doesn't really have players. If it
 * did, it would have to keep track
 * of clients in it's own data structures and use that
 *information in game details responses
 * and game list responses.
 */
@Override
protected void handleRequest(long sessionID, GameRequest request) {

      GameResponse msg = new GameResponse();
      GenericResponse gr = new GenericResponse();
      msg.setJoin(gr);

      gr.setSuccess(true);
      gr.setDescription("HelloGame");

      if (request.getJoin() != null)
            send(sessionID, msg);

}


/**
 * If we want to react to clients' responses, this is the
 * place.
 * This example will not cover any specific things one might
 * wish to do in this method, your hands are free. Try things
 * out and build your perfect game.
 */
@Override
protected void handleResponse(long sessionID, GameResponse response)
{
}
```

2. com.asdf.plugins.hello.HelloGame

Starting point is abstract class GameController and we begin by extending it. It provides few useful ready made methods and a bunch of abstract methods you need to implement. You should check the javadoc API documentation of class GameController when starting your own implementation. It gives you a good information about every method you might need.

```
/**
 * An example game framework provided by the ASDF group to get you a
* quick start
 * on creating your own great games in our game server system. Intended
* to function
 * as a first tutorial to working with our server software, not as a
 * tutorial in game programming.
 */

public class HelloGame extends GameController {

   // ConnectionHandlers will make your life a lot easier. See the
   // javadoc for greater understanding.
   ConnectionHandler connection = null;

   // The game window.
   JFrame helloframe = null;

   /**
    * When the game is started, this method is called. Gives you the
    * chance to do what ever
    * initializing you might wish to do.
    */
   @Override
   public void start() {
         helloframe = new JFrame();
         helloframe.setTitle("Hello World!");
         helloframe.setPreferredSize(new Dimension(640,480));
         /*For now we dont need anything else but the title saying
         Hello World!*/
```

handleResponse, handleRequest and handleStateUpdate are your main ways of communicating with the master game running at the server. Server sends you these kinds of messages and you write what you want to do with them in these methods. Also you can send messages to server using the send method in connectionHandler which you can find using getConnection() method.

```
   /**
    * Game logics has sent a response. If you wish to react in some
    * way, write some code here.
    */
   @Override
   public void handleResponse(Response rsp) {

   }
```

```
/**
 * Game logics requests something of you, if you want to do
 * something about it, write some code here.
 */
@Override
public void handleRequest(Request req) {
}

/**
 * Game logics has sent a state update message, informing you of the
 * new state of the game world.
 * If you wish to react in some way, for example store the
 * information somewhere, or process your
 * own game world model, this would be a good time to do so.
 */
@Override
public void handleStateUpdate(StateUpdate stateUp) {
}
```

This method works like a constructor. Parameters which you might need when starting your game client are passed as GameResponse object sent by the game running at server-side.

```
/**
 * When the client is joining the game for the first time, if he
 * wants to something special about it,
 * this is the place.
 */
@Override
public void init(GameResponse arg0, String arg1, AccountType arg2) {
}
```

3. com.asdf.plugins.hello.CreatePanel

Once again we should start with the abstract class that has been given as so kindly by the makers of this server program. This time we actually don't have a choice or our panel will not work in the lobby program. So we begin by extending CreateGamePanel and implementing all the abstract methods. You should really check the javadoc api of CreateGamePanel to get the idea of all the ready made methods you get by extending CreateGamePanel. Also you should check the documentation of LobbyController since all the communication you can do is through it.

```
/**
 * Game creation panel for HelloGame game example. It is recommended to
* create panels that can
 * be used by several games, instead of tailoring a single one for
* each. See pre created panels
 * before creating your own, might save a lot of trouble.
 *
 * Game type is not decided in the game creation panel, but on the
* upper level of hierarchy.
 * This means that the panels can be used by any game, as long as the
* transmitted information
 * suffices for all games that use it.
 */
```

```
public class CreatePanel extends CreateGamePanel{

   private static final long serialVersionUID = -7975533206336492104L;
   private JLabel helloLabel;

   /**
    * Creates a simple game creation panel.
    *
    */
   public CreatePanel() {
         super();
         initGUI();
         setInitialized(true);
   }


   /**
    * Creates a HelloGame creation panel. The name of the classes to
    * launch with game start
    * are transmitted automatically when the game meta files are
    * properly in place.
    *
    * This means that you can use the same game creation panel for
    * different games that ask
    * for similar parameters for creation from the user. For example
    * many poker games can
    * easily share their game creation panel.
    *
    * This game creation panel asks for no information from the user,
    * and doesn't suit for
    * games that have some creation options (for example time per turn,
    * number of players
    * before starting game, required player level, difficulty, game
    * name, etc..). The only thing the panel
    * does, is listen to mouse clicks. When the mouse is clicked on the
    * panel, it sends a
    * game creation request to server.
    */
   public void initGUI() {

         helloLabel = new JLabel();
         this.add(helloLabel);
         helloLabel.setText("Hello World!");
         helloLabel.setPreferredSize(new java.awt.Dimension(218, 178));
         //And other components we want to add to this panel...
```

When we click a mouse in this panel, we send a message to server that we want to create a new HelloGame. Notice that the only way we can get communication to server is through LobbyController which we can find with getController() method. We then request LobbyController that it would create a new game for us. We pass CreateGameRequest as a parameter to that method which contains all the information server needs to create a new HelloGame.

```
helloLabel.addMouseListener(new MouseListener(){
      @Override
      public void mouseClicked(MouseEvent e) {
            CreateGameRequest createReq = new CreateGameRequest();

            String gameName = "Helloworld";
            createReq.setName(gameName);

            Parameters param = new Parameters();
            param.getAny().add(new BasicElement("nimi"));

            createReq.setParameters(param);
            getController().requestCreateGame(createReq);
      }
});
```

4. What else should I know?

You should definitely familiarize yourself with the API documentation of the extension_api.jar package. Especially you should pay attention to the key classes which we went through in the example game. API documentation contains a lot of information about which methods you can use to communicate with the server and what kind of methods are ready made. Especially you should pay attention to our timer mechanism when implementing a new server-side game. Also you might want to spend some time figuring out how our XML message protocol which is defined by message.xsd -schema file actually works. It should be pretty versatile so you actually shouldn't need to do anything else but use the accessors of the message “objects” passed to you or the ones that you create and pass on to be sent. There are “any” and “custom” fields which can contain any kind of payload. You can insert a Java object or even a picture or sound file in it!