Univerzitet u Sarajevu

Prirodno-matematički fakultet

Odsjek za matematiku

TEHNIČKA SPECIFIKACIJA

Poker2013

Student: Nadin Zajimović Mentor: Safet Habibija

Sarajevo, juni 2013. godine

Projekat „Poker2013“ je skup aplikacija (severska i klijentska aplikacija) koji omogućava igranje popularne igrice „Texas Hold'em Poker“ preko mreže.

Pravila igrice možete naći na linku:
https://en.wikipedia.org/wiki/Texas_hold_'em

Projekat se sastoji iz serverske aplikacije, koja se pokrene na jednom računaru i koja je zadužena za opsluživanje više klijenata, na kojim je pokrenuta klijentska aplikacija. Na server se može konektovati proizvoljan broj klijenata. Serverska aplikacija je koncipirana tako da se unutar nje može napraviti proizvoljan broj soba u kojim može biti 2-5 klijenata, te se u svakoj sobi igra jedna instanca igre. Komunikacija između klijenta i servera se vrši korištenjem soketa, tj. klasa TServerSocket i TClientSocket, koje su nam dostupne u C++Builder-u. Za komunikaciju je razvijen svojevrstan protokol, tj. skup klasa koje modeliraju sve vrste „paketa“, tj. sve vrste informacija koje se prenose između klijenata i servera. To je postignuto tako što je napravljena jedna bazna klasa „Paket“. Ona služi da bi se iz nje mogle naslijediti ostale klase koje modeliraju sve vrste paketa, te da bi se tako mogao iskoristiti princip polimorfizma u C++-u.

Klasa **Paket** se sastoji iz samo jednog atributa:

- TipPaketa tip

Ovaj atribut je tipa „TipPaketa“, to je zapravo pobrojani tip podataka, u kojem sam pobrojao sve vrste paketa koje su bile potrebne za ispravnu komunikaciju. To su:

- pktKonekcijaResponse,
- pktRegistracija,
- pktRegistracijaResponse,
- pktLogin,
- pktLoginResponse,
- pktLogout,
- pktKorisnikZahtjev,
- pktKorisnikResponse,
- pktNapraviSobu,
- pktNapraviSobuResponse,
- pktZahtjevSveSobe,
- pktSveSobeResponse,
- pktUdjiUSobu,
- pktUdjiUSobuResponse,
- pktIzadjiIzSobe,
- pktIzlazakIzSobeNotifikacija,
- pktIzbacenIzSobe,
- pktIzbacenIzSobeNotifikacija,
- pktZatvoriSobu,
- pktSviIgraciUSobi,
- pktSviIgraciUSobiResponse,
- pktZapocniIgruUSobi,
- pktPocetakIgreNotifikacija,

- pktKarte,
- pktPotez

Za svaki od navedenih tipova je nasijeđena posebna klasa, i u nastavku će biti navedena definicija te opis svake od tih klasa.

Klasa „**PaketKonekcijaResponse**“ je klasa naslijeđena iz klase „Paket“ i njena svrha je da se sa servera pošalje informacija klijentu o uspješnosti njegove konekcije i tako će klijent znati da li se uspio konektovati na traženi server. Ona ima dodatne atribute:

- bool status – atribut

Od metoda, tu su dva konstruktora, koji inicijaliziraju atribut status:

- PaketKonekcijaResponse()
- PaketKonekcijaResponse(bool)

Klasa „**PaketRegistracija**“ nam služi da bi u jednom objektu, paketu, mogli slati sve podatke o registraciji novog igrača, sa klijenta na server. Na klijentskoj aplikaciji će korisnik unijeti svoje podatke, koji će preko ove instance ove klase biti poslani na server. Od dodatnih atributa, tu su:

- char ime[] – niz znakova koji predstavlja ime korisnika
- char prezime[] – prezime korisnika
- char username[] – username korisnika, mora biti jedinstven na nivou servera
- char password[] – password koji je sebi dodijelio korisnik

Tu su i konstruktoru zaduženi za inicijalizaciju atributa:

- PaketRegistracija() – inicijalizira atribute na prazne stringove, tj. „ne radi ništa“
- PaketRegistracija(const Korisnik &k) – inicijalizira atribute po uzoru na objekat klase „Korisnik“ koja će biti kasnije opisana, a koja modelira jednog korisnika

Klasa „**PaketRegistracijaResponse**“ je zadužena da korisniku koji je posla zahtjev za registraciju odgovori na isti. Tačnije, kada server obradi pristigli zahtjev, u zavisnosti od ispravnosti podataka (npr. da li je username dostupan ili zauzet), korisniku su šalje primjerak ove klase, koja ima samo još jedan atribut:

- bool status – indikator klijentu o uspješnosti registracije

Tu su i konstruktori koji inicijaliziraju atribut:

- PaketRegistracijaResponse()
- PaketRegistracijaResponse(bool)

Klasa „**PaketLogin**“ služi za slanje zahtjeva za login sa klijenta ka serveru. Ona, pored naslijeđenih atributa, ima još i atribute:

- char username[] – username koji korisnik unese prilikom prijave
- char password[] – password koji korisnik unese prilikom prijave

Tu su i konstruktori:

- PaketLogin()
- PaketLogin(const Korisnik &k)

Klasa „**PaketLoginResponse**“ je zadužena da klijentu pošalje obrađen odgovor na njegov zahtjev za prijavu(login). Ona ima atribute:

- int status – govori klijentu da li je njegova prijava uspješna, da li su podaci pogrešni, ili je već prijavljen na nekom drugom računaru
- const static int LOGIN_USPJESAN=0 – konstanta na nivou cijele klase, uvedena samo zbog čitljivijeg koda
- const static int LOGIN_POGRESAN=1
- const static int VEC_LOGOVAN=2

Za ovu, kao i za sve naredne klase koje su naslijeđene iz klase „Paket“ su razvijeni i konstruktori koji inicijaliziraju atribute, pa njih nećemo eksplicitno navoditi, možete pretpostaviti da svaka klasa koja modelira neki paket posjeduje odgovarajuće konstruktore.

Klasa „**PaketLogout**“ modelira korisnikov zahtjev za odjavu. Njeni atributi su:

- char username[] – username korisnika koji se odjavljuje
- int cipovi – iznos koji je trenutno na njegovom „računu“ a koji će biti pohranjen na serveru

Klasa „**PaketKorisnikZahtjev**“ modelira zahtjev za svim informacijama o korisniku čiji username znamo (naravno, osim passworda). Tu je atribut:

- char username[] – username korisnika čije podatke klijent želi saznati

Klasa „**PaketKorisnikResponse**“ modelira odgovor servera na klijentov zahtjev za korisnika. Atributi:

- bool status – status zahtjeva, true ukoliko korisnik postoji i obrađeni su njegovi podaci, false ukoliko ne postoji korisnik ili je došlo do neke druge grešle
- char ime[] – ime traženog korisnika
- char prezime[] – prezime traženog korisnika
- char username[] – username traženog korisnika
- int cipovi – čipovi na računu traženog korisnika

Klasa „**PaketNapraviSobu**“ modelira zahtjev konkretnog korisnika za pravljenjem sobe na serveru. Njeni atributi su:

- char ime[] – ime sobe koje je korisnik izabrao
- char imeKorisnika[] – ime korisnika koji je poslao zahtjev za pravljenje sobe
- char prezimeKorisnika[] – prezime korisnika
- char usernameKorisnika[] – username korisnika
- int cipoviKorisnika – čipovi korisnika

Klasa „**PaketNapraviSobuResponse**“ modelira severski odgovor klijentu na zahtjev za pravljenje sobe. Tu je atribut:

- int idSobe – šalje korisniku informaciju o jedinstvenom identifikatoru – cijelom broju kojim je njegova soba identificirana za druge korisnike. Ukoliko je došlo do greške, vrijednost ovog atributa je -1

Klasa „**PaketZahtjevSveSobe**“ modelria klijentski zahtjev za sve sobe koje su dostupne u datom trenutku. Ova klasa je, kao i sve prethodno opisane, naslijeđena iz klase „Paket“, te nema dodatnih atributa, jer zahtjev za sve sobe ne zahtijeva nikakve dodatne informacije, osim informacije da je tip paketa pktZahtjevSveSobe.

Klasa „**PaketSveSobeResponse**“ modelira odgovor servera koji sadrži informacije o svim dostupnim sobama. Unutar ove klase, imamo strukturu „Soba“ koja opisuje jednu sobu sa informacijama koje su potrebne klijentu.

Struktura „**PaketSveSobeResponse::Soba**“ se sastoji iz atributa:

- char ime[] – ime sobe
- int id – jedinstveni identifikator sobe
- int brojIgraca – broj igrača trenutno u sobi

Sada možemo nabrojati sve atribtue klase „PaketSveSobeResponse:

- int brojSoba – informacija o broju dostupnih soba
- Soba sobe[] – niz instanci strukture „Soba“ koji klijentu daje informaciju o svim sobama

Klasa „**PaketUdjiUSobu**“ modelira korisnički zahtjev za ulazak u izabranu sobu. Atributi:

- char ime[] – ime korisnika koji želi ući u sobu
- char prezime[] – prezime korisnika
- char username[] – username korisnika
- int cipovi – stanje na računu korisnika
- int idSobe – identifikator sobe u koju korisnik želi ući

Klasa „**PaketUdjiUSobuResponse**“ modelira severski odgovor na klijentov zahtjev. Atributi:

- int dodijeljeniID – identifikator igrača u odabranoj sobi, ovaj identifikator je jedinstven na nivou sobe. Ukoliko je došlo do greške, ovaj atribut ima vrijednost -1

Klasa „**PaketIzadjiIzSobe**“ modelira klijentov zahtjev za izlazak iz sobe u kojoj se nalazi. Atributi:

- int idSobe – identifikator sobe iz koje klijent izlazi
- int idIgracaUSobi – identifikator klijenta u izabranoj sobi

Klasa „**PaketIzbacenIzSobe**“ modelira zahtjev klijenta koji je napravio sobu za izbacivanjem jednog od „gostujućih“ igrač iz te sobe. Atributi:

- int idSobe – identifikator sobe iz koje se klijent izbacuje
- int idIgracaUSobi – identifikator igraca koji se izbacuje, na nivou sobe

Klasa „**PaketIzbacenIzSobeNotifikacija**“ modelira upozorenje servera klijentu da ga je vlasnik sobe izbacio iz sobe u kojoj se do tada nalazio. Ona nema dodatnih atributa jer nije potreno više informacija od same informacije da je to paket koji notificira korisnika o njegovom izbacivanju.

Klasa „**PakezIzlazakIzSobeNotifikacija**“ modelira poruku servera svim klijentima koji se nalaze u jednoj sobi da je jedan od igrača izbačen. Atributi:

- int idIgracaUSobi – identifikator igrača koji se izbacuje, na osnovu kojeg će klijenti primjeniti odgovarajuće akcije

Klasa „**PaketZatvoriSobu**“ modelira zahtjev korisnika koji je otvorio neku sobu, da je zatvori. Atributi:

- int idSobe – identifikator sobe koja se zatvara

Klasa „**PaketSviIgraciUSobi**“ modelira zahtjev klijenta za informaciju o svim igračima u konkretnoj sobi. Atributi:

- int idSobe – identifikator sobe čije igrače klijent želi „upoznati“

Klasa „**PaketSviIgraciUSobiResponse**“ modelira odgovor na klijentov zahtjev za sve igrače neke sobe. Ova klasa u sebi ima strukturu „Korisnik_ID“ koja predstavlja fuziju dvije informacije, identifikator nekog korisnika na nivou sobe, te informacije ko samom korisniku koje imamo dostupne u klasi „Korisnik“ koja će kasnije biti opisana.

Dakle atributi strukture „**PaketSviIgraciUSobiResponse::Korisnik_ID**“ su:

- Korisnik korisnik – informacija o korisniku, jednom igraču u sobi
- int idUSobi – identifikator tog igrača na nivou sobe

Svi atributi klase „PaketSviIgraciResponse“ su:

- int brojIgraca – broj igrača unutar tražene sobe
- Korisnik_ID igraci[] – niz u kojem svaki elemenat sadrži informaciju o jednom igraču u sobi

Klasa „**PaketZapočniIgruUSobi**“ modelira zahtjev vlasnika sobe za početak igre u toj sobi. Atributi:

- int idSobe – identifikator sobe u kojoj počinje igra

Klasa „**PaketPocetakIgreNotifikacija**“ modelira obavještenje servera svim igračima unutar jedne sobe da je igra počela. Nema dodatnih atributa, jer samo obavještenje ne zahtijeva nikakve dodatne informacije.

Klasa „**PaketKarte**“ modelira jedan „špil“ karti. Na klijentskoj aplikaciji klijenta koji je započeo igru, kada je to potrebno, generiše se slučajno poredani špil karti koji će se koristiti u jednoj partiji igre, te se taj špil pomoću objekta ove klase šalje na server. Server po prijemu prosljeđuje taj paket svim igračima u toj sobi. Atributi:

- int brojIgraca – broj igrača u sobi, služi da bi znali koliko karti se šalje, jer nije potrebno slati cijeli špil, dovoljno je samo onoliko karti koliko je potrebno za jednu partiju
- int boje[] – niz koji klijentima daje informaciju o bojama karti
- int brojevi[] – niz koji klijentima daje informaciju o rangu karti (broju na njima)
- int idSobe – identifikator sobe u kojoj se odvija igra

Klasa „**PaketPotez**“ modelira jedan potez u toku igre. Kada klijent odigra potez, on zapravo serveru šalje primjerak ove klase, a onda server taj isti primjerak pošalje svim ostalim klijentima unutar sobe u kojoj se odvija igra. Tada svi klijenti prime taj paket i znaju kako teče tok igre. Atributi:

- TipPoteza tipPoteza – atribut koji nam govori koji je potez u pitanju. „TipPoteza“ je pobrojani tip i može imati sljedeće vrijednosti:
  - potezCall,
  - potezFold,
  - potezRaise
- int iznosRaise – ukoliko je tip poteza „potezRaise“, ovaj atribut sadrži informaciju o količini „raiseanog“, uloženog novca
- int idSobe – identifikator sobe u kojoj se odvija igra
- int idIgacaUSobi – identifikator igrača koji je odigrao potez, na nivou sobe

To je bila definicija korištenog „protokola“ za komunikaciju između serverske i klijentskih aplikacija. Opišimo sada od čega se sastoji serverska aplikacija.

Serverska aplikacije je krajnje jednostavna i sastoji se od jedne maske modelirane klasom „**TformGlavna**“. U toj klasi se, između ostalog, nalaze atributi koji modeliraju vizuelne komponente korištene na masci. Od ostalih atributa tu su:

- TServerSocket *ss_serverSocket – serverski soket koji komunicira sa klijentima
- vector<Soba*> sobe – vektor pokazivača na objekte klase „Soba“ koja modelira jednu sobu na serveru, a koja će biti opisana kasnije

- vector<AnsiString> logovaniKorisnici – vektor u kojem se čuvaju usernameovi svih logovanih korisnika. Koristi se u slučaju da se server zatvori i potrebno je odjaviti sve logovane korisnike
- unsigned long long int nextID – atribut koji vodi računa o jedinstvenom identificiranju svake novonapravljene sobe

Od metoda, tu su event-handleri za vizuelne komponente, te od ostalih, korisnički definisanih metoda, tu su:

- void __fastcall ss_OnClientRead(TObject *Sender, TCustomWinSocket *Socket) – metoda zadužena da sluša zahtjeve klijenata, kada primi zahtjev klijenta alocira potrebni prostor za prijem paketa (to se radi definisanjem pokazivača na baznu klasu „Paket“), u taj prostor smjesti primljeni paket, te u odnosu na atribut „tip“ klase „Paket“ odgonetne o kojem tipu paketa je riječ te poziva odgovarajuću funkciju za obradu pristiglog zahtjeva/paketa
- void registrujKorisnika(PaketRegistracija* paket, TCustomWinSocket* korisnik)
- void prijaviKorisnika(PaketLogin* paket, TCustomWinSocket* korisnik)
- void odjaviKorisnika(PaketLogout* paket, TCustomWinSocket* korisnik)
- void obradiKorisnikZahtjev(PaketKorisnikZahtjev* paket, TCustomWinSocket* korisnik)
- void obradiNapraviSobu(PaketNapraviSobu* paket, TCustomWinSocket* korisnik)
- void obradiZahtjevSveSobe(PaketZahtjevSveSobe* paket, TCustomWinSocket* korisnik)
- void obradiZatvoriSobu(PaketZatvoriSobu* paket, TCustomWinSocket* korisnik)
- void obradiUdjiUSobu(PaketUdjiUSobu* paket, TCustomWinSocket* korisnik)
- void obradiSviIgraciUSobi(PaketSviIgraciUSobi* paket, TCustomWinSocket* korisnik)
- void obradiIzadjiIzSobe(PaketIzadjiIzSobe* paket, TCustomWinSocket* korisnik)
- void obradiIzbacenIzSobe(PaketIzbacenIzSobe* paket, TCustomWinSocket* korisnik)
- void obradiZapocniIgruUSobi(PaketZapocniIgruUSobi* paket, TCustomWinSocket* korisnik)
- void obradiKarte(PaketKarte* paket, TCustomWinSocket* korisnik)
- void obradiPotez(PaketPotez* paket, TCustomWinSocket* korisnik)
- AnsiString trenutnoVrijeme() – metoda koja vraća trenutno vrijeme formatirano
- void odjaviSveKorisnike() – metoda koja odjavljuje sve trenutno prijavljene korisnike

Sve metode osim prve i zadnje dvije služe za obradu pristiglih paketa na server, i primaju po 2 parametra, prvi je pokazivač na paket koji je pristigao, drugi je pokazivač na Socket koji komunicira sa klijentom koji je poslao paket.

Osim ove klase, serverska aplikacija koristi i sljedeće klase:

Klasa „**Korisnik**“ služi za modeliranje jednog korisnika. Atributi:

- char ime[] – ime korisnika

- char prezime[] – prezime korisnika
- char username[] – username korisnika
- char password[] – password korisnika
- int cipovi – stanje na računu korisnika
- static const int POCETNI_BROJ_CIPOVA=100000 – konstantan atribut na nivou klase koji nam govori koliko se čipova (novaca) dodjeljuje korisniku prilikom registracije

Metode:

- Korisnik() – inicijalizira broj cipova na početni broj čipova
- Korisnik(const char[], const char[], const char [], const char []) – radi isto što i konstruktor bez parametara, uz dodatnu inicijalizaciju ostalih atributa na vrijednosti zadane parametrima
- Korisnik(const Korisnik&) – konstruktor kopije
- Korisnik& operator =(const Korisnik&) – preklopljeni operator dodjele

Klasa „**Soba**“ modelira jednu sobu na klijentu. Ona unutar sebe sadrži strukturu „IgracSocket“ koja modelira jednog igrača u sobi, igrača sa svim njegovim podacima, te soketom koji komunicira sa tim igračem.

Atributi strukture „**Soba::IgracSocket**“ su:

- int id – identifikator igrača na nivou sobe
- char ime[] – ime igrača
- char prezime[] – prezime igrača
- char username[] – username igrača
- int cipovi – stanje na računu igrača
- TCustomWinSocket *igrac – soket koji komunicira sa igračem

Atributi klase „Soba“ su:

- int id – jedinstveni identifikator sobe na nivou servera
- int nextID – atribut koji se brine za jedinstvenost identifikatora igrača na nivou sobe
- int brojIgraca – broj igrača koji su u sobi
- char ime[] – ime sobe
- bool igraUToku – atribut koji nam govori da li je počela igra u sobi, ili je soba još uvijek u fazi primanja igrača
- std::vector<IgracSocket*> igraci – vektor koji sadrži pokazivače na objekte strukture „IgracSocket“, ovo su zapravo svi igrači u sobi
- const static unsigned short MAX_BROJ_IGRACA=5 – atribut koji ograničava broj igrača unutar jedne sobe

Metode klase „Soba“ su:

- Soba(unsigned short,const char []) – konstruktor koji kreira „praznu“ sobu i inicijalizira joj id i ime

- ~Soba() – destruktor koji oslobađa svu memoriju koju je objekat alocirao tokom svog života
- int dodajIgraca(TCustomWinSocket*, const char[], const char [], const char [], int, TMemo*) – metoda koja dodaje igrača u sobu po uzoru na parametre, te osim toga ispisuje informacije o procesu dodavanja na „TMemo“ objekat na koji pokazuje zadnji parametar
- void posaljiSvimaSpisakIgraca() – svim igračima u sobi šalje paket sa svim igračima u sobi
- void izbrisiIgracaPremaID(int, TMemo*) – briše iz sobe igrača čiji je identifikator na nivou sobe jednak prvom parametru, te ispisuje informacije i procesu brisanja na objekat na koji pokazuje drugi parametar
- void izbrisiIgracaPremaUsernameu(char []) – briše iz sobe igrača čiji je username jednak parametru
- void obavijestiSveIgraceOBrisanju(int) – šalje paket-notifikaciju o brisanju igrača sa ID-om proslijeđenim kao parametar svim igračima u sobi
- void obavijestiSveIgraceOZatvaranju() – šalje obavijest o zatvaranju sobe svim igračima u sobi
- void obavijestiSveIgraceOPocetku() – šalje paket-obavijest o početku igre svim igračima u sobi
- TCustomWinSocket* vratiSoketIgracaPremaID(int) – nalazi i vraća kao rezultat soket koji komunicira sa igračem čiji je identifikator jednak parametru

Pređimo sada na opis klijentske aplikacije, tj. klasa korištenim za nju. Klijentska aplikacija se sastoji iz maski za konekciju na server, prijavu, registraciju, odabir sobe, prikaz igrača u odabranoj sobi, te maske za igru. One su redom modelirane klasama „TFormConnect“, „TFormLogin“, „TFormRegistracija“, „TFormSobe“, „TFormSobaIgraci“, „TFormIgra“. Sve ove forme posjeduju atribute koji modeliraju vizuelne komponente na formama, te event-handlere za određene događaje koji kreiraju neke od navedenih paketa i šalju ih serveru na obradu, te koji primaju pakete od servera i obrađuju ih, kao i konstruktore. Opišimo sada koje bitne dodatne atribute i metode posjeduju klase koje modeliraju navedene forme.

Klasa „**TFormConnect**“

Atributi:

- TClientSocket *cs_clientSocket – klijentski soket koji komunicira sa serverom

Metode:

- bool daLiJeAdresa(const AnsiString&) – provjerava da li je unesena adresa u „IP“ formatu
- void otvoriFormuZaLogin() – nakon uspješne konekcije otvara formu za prijavu

Klasa „**TFormLogin**“

Atributi:

- TClientSocket *cs_clientSocket – klijentski soket koji komunicira sa serverom, ovaj atribut predstavlja pokazivač na soket kreiran u formi za konekciju, i pri otvaranju forme za prijavu on se postavlja da pokazuje na pokazivač kreiran u formi za konekciju. Također, prilikom otvaranja forme za prijavu, odgovarajući atributi soketa, event-handleri, se mijenjaju i postaju metode napisane u ovoj formi, tako da klijentski soket koji je „zajednički“ za sve forme, ali se samo prenosi sa forme na formu preko pokazivača zna koje funkcije treba pozivati na određene događaje
- void __fastcall (__closure *__OnRead)(TObject*, TCustomWinSocket*) – pokazivač na funkciju koja je bila event-handler za „OnRead“ događaj prije otvaranja forme za prijavu. Kao što smo i rekli, prilikom otvaranja formi se mijenjaju event-handleri klijentskog soketa, ali prilikom zatvaranja forme ti handleri se moraju vratiti na staru vrijednost. Upravo za to nam služi ovaj pokazivač na funkciju, da znamo koja je funkcija obavljala „dužnost“ event-handlera prije otvaranja forme za prijavu. Sve ostale forme sadrže ova dva atributa i oni imaju isto značenje kao i u ovom slučaju, tako da oni neće biti posebno opisivani

Metode:

- void __fastcall cs_OnRead(TObject*, TCustomWinSocket*) – metoda koja je event-handler za soketov „OnRead“ događaj za vrijeme dok je ova forma otvorena i koja služi za primanje potvrde o uspješnoj/nesupješnoj prijavi

Klasa „**TFormRegistracija**“

Atributi:

- TClientSocket *cs_clientSocket – značenje opisano ranije
- void __fastcall (__closure *__OnRead)(TObject*, TCustomWinSocket*) – značenje opisano ranije

Metode:

- void __fastcall cs_OnRead(TObject*, TCustomWinSocket*) - metoda koja je event-handler za soketov „OnRead“ događaj za vrijeme dok je ova forma otvorena i koja služi za primanje potvrde o uspješnoj/nesupješnoj registraciji

Klasa „**TFormSobe**“ služi za prikaz svih dostupnih soba i odabir one u koju želimo ući

Atributi:

- int *sobeID – niz u kojem se čuvaju identifikatori soba
- char sobeImena[][] – matrica u kojoj se čuvaju imena sobe
- int odabranaSoba – identifikator odabrane sobe, mijenja se klikom na određenu sobu
- char odabranoIme[] – ime odabrane sobe

Metode:

- void set_korisnikUsername(const char []) – metoda za postavljanje usernamea trenutnog korisnika
- void init() – metoda koja inicijalizira atribute ove klase, te šalje zahtjeve za sve sobe
- void odjaviKorisnika() – metoda koja šalje zahtjev serveru za odjavu korisnika, poziva se pri zatvaranju forme

Klasa „**TFormSobaIgraci**“ modelira formu u kojoj se prikazuju svi igrači sobe u kojoj se trenutno nalazimo, te nam daje mogućnost da izađemo iz sobe te da počnemo igru i izbacimo igrača iz sobe, ukoliko smo mi kreator sobe

Atributi:

- std::vector<Korisnik_ID> ostaliIgraci – vektor u kojem se čuvaju informacije o ostalim igračima unutar sobe (ne računajući igrača koji je prijavljen)
- int brojOstalihIgraca – broj ostalih igrača u sobi
- int izabraniIgracZaBrisanje – identifikator igrača kojeg smo označili za brisanje. Koristi se samo u slučaju da je igrač ujedno i kreator sobe
- bool daLiSamVlasnik – atribut koji nam govori da li je prijavljeni igrač napravio sobi ili je samo gost u njoj
- char imeSobe[] – ime sobe u kojoj se nalazimo
- int idSobe – identifikator sobe u kojoj se nalazimo
- int idIgracaUSobi – identifikator prijavljenog igrača na nivou sobe
- bool daLiSamIzbacen – atribut koji nam daje informaciju o tome da li smo izbačeni od strane kreatora sobe
- TForm* kreator – pokazivač na formu iz koje je otvorena ova forma. Koristi se za prikazivanje kreatorske forme, nako što se ova forma zatvori, te za manipulaciju sa klijentskim soketom i njegovim event-handlerima

Metode:

- void updateujTabeluIgraca() – nakon pristiglog paketa sa svim igračima, updateuje vizuelne komponente da budu u skladu sa pristiglim paketom
- void init() – inicijalizira atribute, te šalje zahtjev za sve igrače u sobi
- void pocniIgru() – otvara se forma za igru

Klasa „**TFormIgra**“ modelira formu u kojoj se odvija igra

Atributi:

- std::vector<Igrac*> igraci – vektor koji čuva pokazivače na objekte klase „Igrac“ koja će biti kasnije objašnjena a koja modelira jednog igrala
- Stadij stadij – stadij u kojem se nalazi igra, pobrojan tip koji prima vrijednosti iz skupa {preFlop, flop, turn, river}
- int igracNaPotezu – indeks igrača koji je trenutno na potezu
- int igracZadnjiRaise – indeks igrača koji je zadnji raiseao/uložio
- int igracBigBlind – indeks igrača koji je u sljedećem krugu „big blind“ igral

- Karta flop1, flop2, flop3, turn, river – „community cards“, karte dostupne svima. Ovo su objekti klase „Karta“ koja će nešto kasnije biti objašnjena
- std::vector<Pot*> potovi – vektor pokazivača na objekte klase „Pot“ koja će biti kasnije objašnjena a koja modelira jedan „Pot“ u igri Poker (vidi pravila igre opisana u postavljenom linku na početku ovog dokumenta)
- int indexByID(int) – vraća indeks igrača u vektoru koji ima identifikator na nivou sobe jednak parametru
- int mojID – identifikator igrača koji je logovan, na nivou sobe
- int idSobe – identifikator sobe u kojoj se odvija igra

Metode:

- int prethodniIndex(int) – vraća indeks koji dolazi nakon indeksa iz parametra, imajući u vidu da nakon zadnjeg indeksa ponovo dolazi prvi (igrači igraju puteze kružno)
- int sljedeciIndex(int) – slično kao prethodna metoda
- void dodijeliSlucajnoKarte() – slučajno raspoređuje špil i dodjeljuje karte igračima i šalje serveru paket s kartama
- void rasporediKarte(PaketKarte*) – za primljeni paket s kartama, raspoređuje karte
- int izracunajUkupanUlog(int) – računa koliko je igrač sa datim indeksom uložio
- void iscrtajFlop() – „okreće“ karte sa flopa na vidljivu stranu
- void iscrtajTurn() – „okreće“ kartu sa turna na vidljivu stranu
- void iscrtajRiver() – „okreće“ kartu sa rivera na vidljivu stranu
- void sakrijFlop() – „okreće“ karte sa flopa na nevidljivu stranu
- void sakrijTurn() – „okreće“ kartu sa turna na nevidljivu stranu
- void sakrijRiver() – „okreće“ kartu sa rivera na nevidljivu stranu
- void sakrijOstaleIgrace() – „okreće“ karte svim ostalim igračima na nevidljivu stranu
- bool provjeriValidnostRaiseIznosa(int iznos) – prvojerava da li je iznos za raise validan
- void updateujIgricu() – poziva se nakon odigranog poteza i zadatak joj je da ažurira vrijednosti atributa koji modeliraju igricu
- void zavrsiPartiju() – završava partiju, jedan krug igrice, te ispisuje pobjednike, dodjeljuje čipove pobjednicima itd.
- void odigrajCall(int) – odigrava potez „call“ za igrača čiji je identifikator zadan parametrom
- void odigrajFold(int) - odigrava potez „fold“ za igrača čiji je identifikator zadan parametrom
- void odigrajRaise(int, int) - odigrava potez „raise“ za igrača čiji je identifikator zadan parametrom, za iznos koji je zadan drugim parametrom
- void ispisiIgruUDatoteku() – vrijednosti promjenjivih koje modeliraju stanje igre ispisuje u datoteku. Koristi se za potrebe testiranja
- void init() – inicijalizira igru, atribute postavlja na početne vrijednosti
- void dodajIgraca(const Korisnik &, int) – dodaje novog igrača

Osim navedenih klasa, tu su i klase koje modeliraju logiku igre.

Klasa „**Karta**“ ima atribute:

- Boja boja – jedna od vrijednosti iz pobrojanog skipa { list, mak, srce, kocka }

- int broj – broj na karti
- TJPEGImage *slika – pokazivač na sliku karte, jednu od 52 slike za svaku kartu iz špila

Klasa „**Igrac**“:

- char ime[] – ime igrača
- char prezime[] – prezime igrača
- int cipovi – stanje na računu igrača
- int idUSobi – identifikator u sobi
- Karta karta1,karta2 – dvije karte dodijeljene igraču
- Status status – trenutni status igrača, iz skupa { igra, allIn, odustao, izasao }
- TPicture *canvasK1,*canvasK2 – pokazivači na slike na kojim se iscrtavaju slike karti
- TImage* slikaPotez – pokazivač na sliku na kojoj se iscrtava slika koja je indikator da je igrač na potezu
- TLabel *l_cipovi, *l_ulozeno – pokazivači na labele na kojim se ispisuje stanje računa igrača, te količina uloženog novca/čipova

Klasa „**Pot**“ modelira jednu „kamaru“ novca na kojoj se nalazi uloženi novac igrača koji su u istom finansijskom „rangu“. Ona ima atribute:

- int brojIgracaNaPotu
- int indexiIgracaNaPotu[] – indeksi svih igrača koji učestvuju i dalje u igri, te se nalaze na ovom potu
- int ulogPoIgracu – količina novca koju igrač mora zadovoljiti da bi ostao na potu
- int uloziIgraca[] – količina uloženog novca za svakog pojedinačnog igrača
- int brojIgraca – ukupan broj igrača u sobi
- TLabel* naslov – pokazivač na labelu na kojoj se ispisuje ime pota
- TLabel* iznos – pokazivač na labelu na kojoj se ispisuje iznos pota

Klasa „**SkupKarti**“ logički modelira skup od 7 karti koje igrač ima na raspolaganju. Za nju je definisan i operator poređenja kako bi se lako mogao utvrditi pobjednik. Skup od 7 karti potrebno je transformirati u skup od 5 karti koji predstavlja najbolju moguću poker-kombinaciju. U tu svrhu, unutar klase „SkupKarte“ je razvijena struktura „__Karta“ koja logički modelira kartu, bez njenih grafičkih elemenata, te struktura „__Hand“ koja modelira skup od 5 karata koji predstavlja najbolju moguću poker-kombinaciju.

Struktura „**__Hand**“ ima atribute:

- __Karta karte[5] – niz od 5 karti koje predstavljaju karte Handa
- TipHanda tip – tip Handa, jedna od mogućih poker kombinacija { thRoyalFlush, thStraightFlush, thPoker, ..., thOnePair, thHighCard }

te metode

- bool operator>(const __Hand&) – testira koji je hand bolje gledajući atribut „tip“. Ukoliko su istog tipa, traži se kod kojeg handa su karte većeg ranga
- bool operator==(const __Hand&) – testira jednakost dva handa

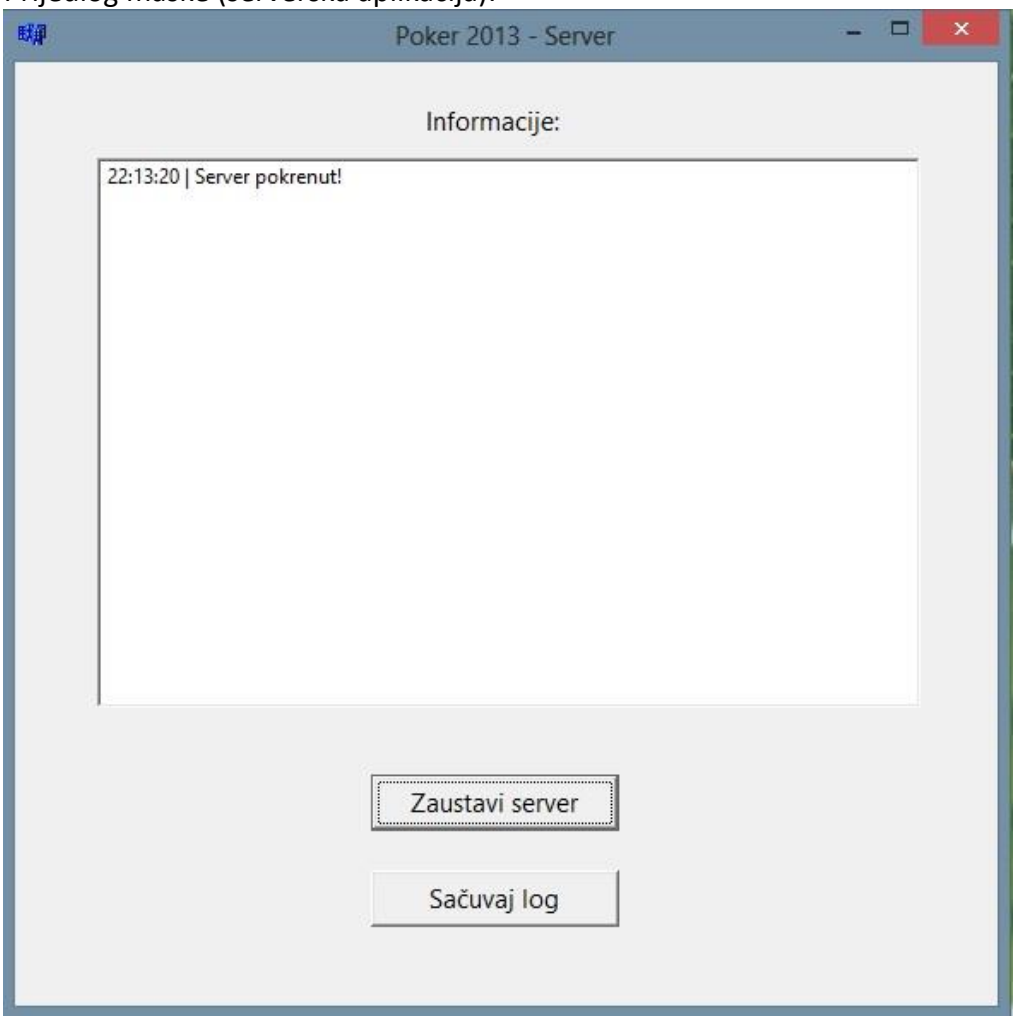
Klasa „SkupKarti“ ima atribute:

- __Karta karte[] – niz od 7 karti koji predstavlja karte dostupne igracu
- __Hand hand – hand – najbolja kombinacija od 5 karti od mogućih 7
- bool bits[] – niz koji za svaku kartu u špilu govori da li se nalazi u igraču dostupnim kartama. Služi za testiranje koji je hand u pitanju

Metode:

- SkupKarti(const Karta &, const Karta &, const Karta &, const Karta &, const Karta &k5, const Karta &, const Karta &) - konstruktor
- SkupKarti(const __Karta &, const __Karta &, const __Karta &, const __Karta &, const __Karta &, const __Karta &, const __Karta &) - konstruktor
- void konstruisiHand() – konstruise najbolji hand
- bool operator>(const SkupKarti&) – poredi dva skupa karti
- bool operator==(const SkupKarti&) – poredi jednakost dva skupa karti
- int kartaToBitIndex(const __Karta &) – pretvara kartu u njen index u nizu bits
- __Karta bitIndexToKarta(int) – pretvara index iz niza bits u kartu jedinstveno određenu indeksom
- bool provjeriRoyalFlush() – provjerava postojanost Royal flush kombinacije u skupu od 7 karti
- bool provjeriStraightFlush() - provjerava postojanost Straight flush kombinacije u skupu od 7 karti
- bool provjeriPoker() - provjerava postojanost Poker kombinacije u skupu od 7 karti
- bool provjeriFullHouse() - provjerava postojanost Full house kombinacije u skupu od 7 karti
- bool provjeriFlush() - provjerava postojanost Flush kombinacije u skupu od 7 karti
- bool provjeriStraight() - provjerava postojanost Straight kombinacije u skupu od 7 karti
- bool provjeriTriling() - provjerava postojanost Triling kombinacije u skupu od 7 karti
- void provjeriPairs() - provjerava postojanost parova u skupu od 7 karti
- AnsiString vratiImeHanda() – vraća ime handa, u zavisnosti od tipa

Prijedlog maske (serverska aplikacija):



Prijedlog maske (klijentska aplikacija) – 1:

Prijedlog maske (klijentska aplikacija) – 2: