Úvod do počítačových sítí

KIV/UPS

Dokumentace k semestrální práci – Blackjack Server, klient

Václav Honzík
A19B0674P

1 ZADÁNÍ

Vytvořte klient a server realizující vybranou hru. Server bude schopen obsluhovat více hráčů najednou. Hráči se připojí do jedné z herních místností, ve kterých se poté spustí samotná hra. Herní místnosti fungují paralelně – jsou na sobě nezávislé. Pro přenos zpráv použijte nešifrovaný textový protokol nad protokolem TCP. Server i klient jsou stabilní a nepadají po / během několika odehraných hrách.

2 ZVOLENÁ HRA A PRAVIDLA

Jako hru, kterou budu implementovat jsem si zvolil Blackjack. Blackjack je relativně rychlá a jednoduchá karetní hra. Hra se hraje mezi hráči a dealerem (krupiér, bankéř …). Každý hráč na začátku hry dostane dvě karty a pak mu dealer nabízí další karty. Hraje se s klasickými "žolíkovskými kartami". Cíl hry je dostat součet karet vyšší, než má dealer, ale zároveň nepřekročit hodnotu 21. Pokud hráč hodnotu 21 překročí automaticky prohrává. Hráči tedy nehrají proti sobě, ale proti dealerovi.

Ve hře se používají všechny karty kromě žolíků, karty 2 – 10 mají stejnou hodnotu, karty J, Q, K mají hodnotu 10 a eso (A) má hodnotu 1– tvrdý součet (**hard hand**) nebo 11 – měkký součet (**soft hand**) podle hráčova uvážení. Pokud má hráč dvojici karet A a kartu s hodnotou 10 (může být jakákoliv – 10, J, Q, K), tak má tzv. **Blackjack** (natural) – tato kombinace mu zaručí, že když bude stát, dostane minimálně svůj vklad (viz konec hry).

2.1 HRA HRÁČE

Před samotným startem hry musí každý hráč vložit nějakou částku peněz. Po uzavření všech sázek jsou hráčům rozdány dvě karty a hrají postupně po směru hodinových ručiček. Hráč má ve svém kole několik možností:

- **Hit** – hráč si vezme další kartu, pokud je busted (překročil 21), následuje hra dalšího hráče nebo dealera. Po zahrání "hitu" už nemůže hráč jít double, ale může zahrát hit znovu, pokud 21 nepřekročil.
- **Stand** – hráč zůstane stát a hraje hráč po něm popř. dealer.
- **Double** – hráč zdvojnásobí původně vsazenou sázku, vezme si kartu a jeho hra končí.

2.2 HRA DEALERA

Stejně jako normální hráč si dealer na začátku hry vezme dvě karty. Jeho druhá karta je vždy otočena obráceně, aby nešla zjistit její hodnota – odhalí se až po dohrání hráčů. Dealer hraje jako poslední. Po skončení kol všech hráčů se otočí jeho druhá karta a pokud je jeho součet

menší než 17 (při měkkém součtu) vezme si další kartu, takto pokračuje, dokud není jeho měkký součet větší jak 17.

2.3 KONEC HRY

Výhry hráčů jsou určeny podle splnění (resp. nesplnění) následujících podmínek:

- Pokud krupiér překročí hodnotu 21, vyhrávají všichni hráči, kteří hodnotu 21 nepřekročili.
- Pokud hráč překročí hodnotu 21, automaticky prohrává bez ohledu na výsledek dealera.
- Pokud hráč nemá blackjack a vyhrál, dostane dvojnásobek svého vkladu. Pokud má hráč blackjack a krupiér nikoliv vyhraje 2.5 násobek svého vkladu a pokud dealer má také blackjack, je hráči vrácen pouze jeho vklad.

3 POPIS PROTOKOLU

3.1 FORMÁT A ZPRACOVÁVÁNÍ ZPRÁV

Formátování zpráv je z části inspirováno formátem **JSON**u a z části formátem **CSV**. Zprávu tedy můžeme chápat jako jakýsi jednoduchý objekt – má vždy nějaké pole (field – index) a hodnotu pole (field value). Pole a jeho hodnota jsou oddělené dvojtečkou, tato dvojice je dále oddělená čárkou. Příklad:

```
{pole1:hodnota1,pole2:hodnota2}\n
```

Zpráva má vždy navíc jeden ze tří "typů dat" – **request**, **response** a **ping**. **Request** znamená, že zpráva je požadavek z jedné strany – často tedy očekává odpověď - **response**. **Ping** slouží ke kontrole připojení jak klienta, tak serveru – klient posíláním pingu zjišťuje, zda-li je server naživu a server si po přijmutí pingu automaticky obnovuje čas přijetí zprávy klienta – po delší době co klient neobdrží zprávu klienta odpojí a odstraní jeho data. Některé zprávy navíc používají i středník, který slouží k oddělení více hodnot jednoho pole. Příklad:

```
{lobby1:1;0;6,response:lobbyList,lobby0:0;0;6,dataType:response}
```

Tento formát jsem si zvolil zejména kvůli snadnému zpracovávání a vytváření zprávy – pro implementaci nám stačí pouze hashovací mapa, kam budeme jednotlivé pole a jejich hodnoty. Tento formát jsem si zvolil především kvůli tomu, že se snadno serializuje a deserializuje a také je velmi přehledný při debugování. Nevýhodou je pouze velké množství speciálních znaků – pokud bychom chtěli předejít museli bychom escapovat – např používat uvozovky, jako to dělá klasický JSON.

3.2 TABULKA ZPRÁV KLIENTA A SERVERU

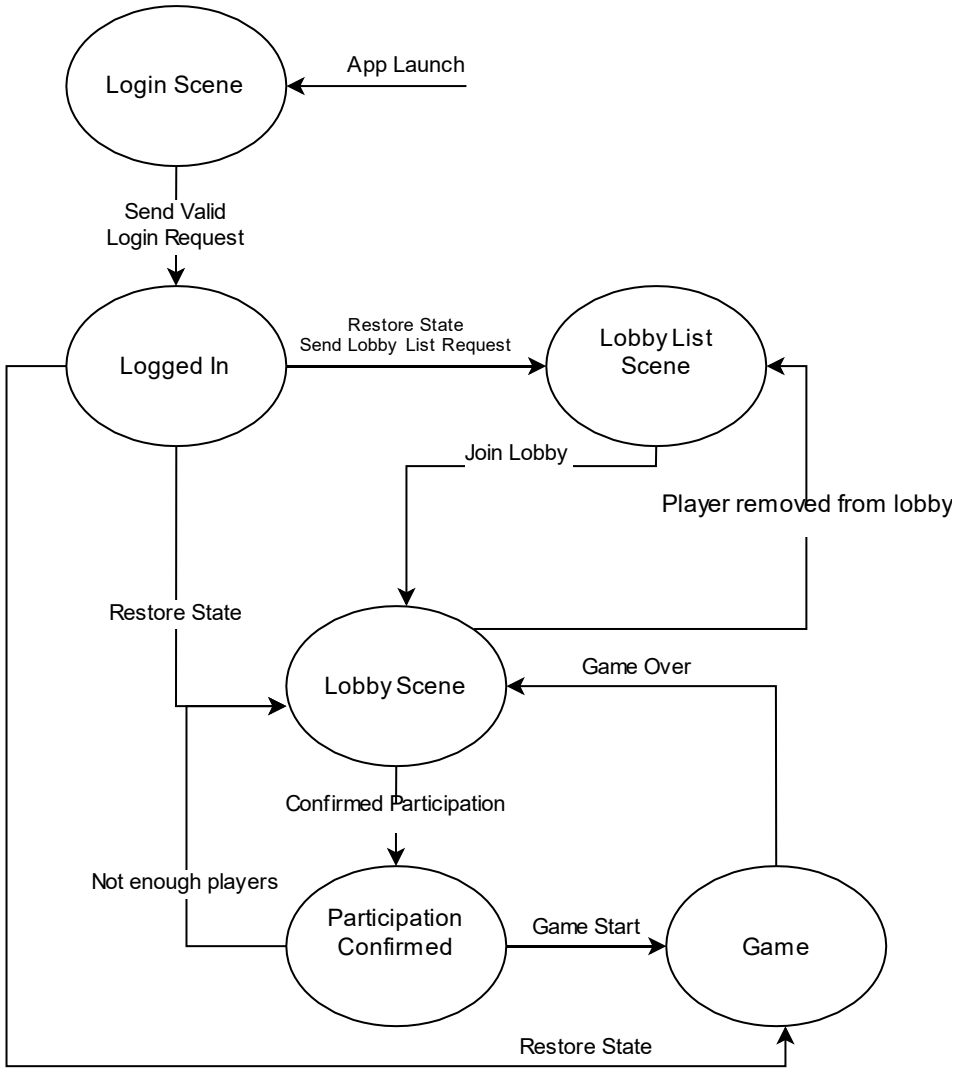
Zpráva	Význam	Odesílá	Odpověď na
`{request:login,username:username,dataType:request}`	Přihlášení	klient	
`{restoreState:bool,response:login,dataType:response}`	Odpověď na přihlášení	server	login
`{request:lobbyList,dataType:request}`	Požadavek na seznam herních místností	klient	
`{response:lobbyList,lobby0:id;playercount;playerlimit, dataType:response}`	Seznam herní místnosti	server	lobbyList
`{request:joinLobby,lobbyId:0,dataType:request}`	Požadavek na připojení do místnosti	klient	
`{isJoinable:true,response:joinLobby,dataType:response}`	Odpověď na připojení do místnosti	server	joinLobby
`{request:updatePlayerList,dataType:request}`	Požadavek na aktualizaci seznamu hráčů	server	joinLobby
`{response:updatePlayerList,dataType:response}`	Odpověď klienta s aktualizací hráčů	klient	updatePlayerList nebo sendReady
`{dataType:ping}`	Ping serveru	klient	
`{response:ping,dataType:response}`	Odpověď serveru na ping	server	ping
`{request:sendReady,dataType:request}`	Klient je připraven na hru	klient	
`{request:leaveLobby,dataType:request}`	Klient chce opustit lobby	klient	
`{leaveLobby:bool,response:leaveLobby,dataType:response}`	Klient byl odstraněn z lobby	server	leaveLobby
`{restoreState:state,response:login,dataType:response}`	Klient se znovu připojil a obnovuje si stav	server	login
`{request:confirmParticipation,dataType:request}`	Požadavek o potvrzení účasti	server	
`{request:game,dataType:request}`	Požadavek o přípravě hry	server	

{bet:int,response:confirmParticipation,dataType:response}	Odpověď na sázku	klient	confirmParticipation
{playerCount:int,P1:playerUsername,P1C0:cardSuit;cardRank,P1bet:int,P1totalValue:hardHand;softHand, P1C1…,P2…, request:updateBoard,dataType:request}	Aktualizace herní plochy	server	
{request:turn,dataType:request}	Hra klienta	server	
{response:turn,turnType:hit,dataType:response}	Akce klienta – hit, double down nebo stand	klient	turn
{P0bet:betAmount;won,P0:loss, request:showResults,totalValue:hardHand;softHand, dataType:request}	Výsledek hry	server	
{time:int,request:showReturnToLobby,dataType:request}	Zobrazení limitu pro konec hry	server	
{request:showGameStartFailed,dataType:request}	Zobrazení o neodstartování hry	server	
{request:showCurrentPlayer,username:playerUsername, dataType:request}	Požadavek o zobrazení aktuálního hráče	server	
{request:showPlayerReconnected, username:playerUsername,dataType:request}	Požadavek o zobrazení znovupřipojení hráče	server	
{request:showPlayerDisconnected, username:playerUsername,dataType:request}	Požadavek o zobrazení odpojení hráče	server	
{response:notYourTurn,dataType:response}	Odpověď na pokus o tah, když není hráč na řadě	server	turn
{response:doubleAfterHit,dataType:response}	Odpověď na double down po zahrání hit	server	turn
{response:showPlayerSkipped,username:playerUsername, dataType:request}	Zobrazení, že byl hráč přeskočen	server	
{request:clientDidntConfirm,dataType:request}	Oznámení, že hráč nepotvrdil sázku	server	

{response:removedFromLobby,dataType:response}	Odpověď na odstranění uživatele z lobby po skončení sázky	server	Bet
{response:showPlayerConnected,dataType:response}	Připojení hráče	klient	showPlayerConnected
{response:showPlayerDisconnected,dataType:response}	Odpojení hráče	klient	showPlayerDisconnected
{dataType:response,response:declineParticipation}	Zrušení sázky	klient	confirmParticipation

Tabulka 1 – zprávy protokolu

3.3 STAVOVÝ DIAGRAM KLIENTA



Obrázek 1 - Stavový Diagram Klienta

Zjednodušený model klienta pracuje na stavovém diagramu v *obr. 1*. Pro přehlednost jsem některé stavy a přechody vynechal. Klient se může v jakémkoliv stavu odpojit, což by vede napřed k pokusu o připojení a po neúspěchu do stavu **Login Scene**.

Po přihlášení (přechod pomocí **Send Valid Login Request**) obdrží klient od serveru zprávu, zda-li jeho uživatelské jméno má či nemá obnovit svůj stav – pokud se např klient na krátkou dobu odpojil.

Následná funkcionalita je zřejmá – pokud je uživatel v seznamu herních místností – **Lobby List Scene,** vybere si místnost, pokud se lze do místnosti připojit, je připojen, jinak zůstane v předchozím stavu. V samotné herní místnosti pak všichni klienti hlasují pro start a pokud se vše podaří jak má, jsou umístěni do hry. Jinak jsou upozorněni zprávou o nedostatku hráčů – pokud potvrdili svůj start, popř. odstraněni z lobby – nepotvrdili svůj start. Pokud alespoň dva hráči potvrdili start hry, je hra spuštěna. Po dohrání hry jsou všichni zúčastnění přesunuti zpět do lobby, kterou mohou opustit, nebo znovu hlasovat pro start.

4 PROGRAMÁTORSKÁ DOKUMENTACE

4.1 IMPLEMENTACE SERVERU

Server jsem implementoval v programovacím jazyce C++ (standard C++ 17), pro sestavení programu jsem použil nástroj CMake. Celý server běží na jednom vlákně, pro čtení dat od jednotlivých klientů používá funkci *select()*. Server nejprve skusí spustit svůj socket na uživatelem zadaném (nebo výchozím, v případě, že žádný nezadal) portu, poté vytvoří všechny potřebné struktury pro svojí funkčnost a následuje čtení dat od klientů.

Server obsahuje několik "logických vrstev" na kterých pracuje – třída **Server** reprezentuje samotný server – s touto třídou komunikuje klient – obsahuje cyklus se select funkcí a předává data do dalších tříd. Zprávy od klientů zpracovává několik tříd – třída **MessageHandler** je neobecnější třídou pro zpracování zpráv - analyzuje jenom požadavky pro autentifikaci a připojení do herní místnosti, ostatní zprávy předává třídě **LobbyMessageHandler**, což je třída pro zpracování zpráv samotné herní místnosti. Zprávy o hře jsou pak přes LobbyMessageHandler předány třídě **GameController**, což je třída kontrolující stav a chod hry.

4.1.1 Čtení dat od klientů

Po konfiguraci socketu serveru se spustí nekonečná smyčka, ve které je využívána funkce ***select()***, která zjistí zda-li přišlo nějaké příchozí spojení od klienta – pokud ano, klienta přijme a čeká, dokud klient něco nepošle, nebo dokud nevyprší časovač pro odpojení. V případě, že klient pošle odpověď, která není validní – špatný formát, nebo nedává smysl v daném stavu klienta, je spojení automaticky ukončeno. Samotná data klienta jsou odstraněna po cca 5

minutách, kdy od klienta nebyla obdržena žádná zpráva, klientův aktuální socket je uzavřen po cca 15 sekundách neaktivity.

Data klienta jsou uložena ve třídě **Client** – tato třída obsahuje aktuální socket pod kterým je uživatel připojen, jeho uživatelské jméno a herní data. Pro lepší přehlednost kódu jsem herní data zabalil do třídy **PlayerInfo**, v podstatě obsahují pouze to kolik vsadil a jeho aktuální karty, třída také počítá celkový součet a to zda-li je hráč *busted* nebo ne. Server si všechny klienty ukládá do seznamu (resp. vektoru) aby je mohl kdykoliv jednoduše smazat.

Přečtený řetězec ze socketu klienta je namapován na objekt **TCPData**, který zprávu deserializuje aby se dala následně použít pro zjištění požadavku či odpovědi. TCPData jak již bylo zmíněno v popisu protokolu obsahují pouze hashmapu a metody, které najdou hodnotu pro zadané pole. Pro uložení dat takto jednoduchého serveru nám bohatě postačí datový typ **string**.

4.1.2 Lobby a start hry

Pokud se klient chce do lobby připojit, pošle požadavek o připojení. Požadavek je vyhodnocen a pokud se v lobby zrovna neodehrává hra nebo není plná, je uživatel do lobby přidán. Lobby stejně jako server obsahuje seznam klientů, kam server klienta přidá, po odpojení klienta z lobby je zase ze seznamu odstraněn. Pro zajištění hry má každá svá místnost herní controller – **GameController**, který využije při zpracování zpráv týkajících se hry.

Při čtení zpráv klientů se na jednotlivé herní místnosti ukládají zprávy klientů, které se následně budou zpracovávat. Herní místnost spolu se zpracováváním zpráv provádí i kontrolu stavu ve kterém je, mohou být čtyři stavy (enum **LobbyState**):

1. **Waiting** – lobby čeká, dokud všichni připojení nepotvrdí účast.
2. **Preparing** – lobby se připravuje na start hry, přes GameController pošle všem zúčastněným klientům požadavek pro potvrzení.
3. **In-game** – lobby je ve hře, v tomto stavu pouze předává zprávy GameControlleru popř. odesílá, že se nelze do lobby připojit.
4. **Finished** – lobby čeká na návrat hráčů ze hry.

Nejprve je lobby ve stavu **Waiting**, v tomto stavu čeká na všechny klienty, dokud neodešlou, že jsou připraveni ke hře. Následně přejde do stavu **Preparing**. Všem klientům je odeslán požadavek na vložení částky. Pokud odpoví aspoň dva klienti, GameController odstartuje hru, v opačném případě se lobby vrátí zpět do stavu Waiting.

4.1.3 Hra

Při startu hry si GameController vytvoří novou instanci třídy Blackjack, pomocí které vygeneruje karty pro jednotlivé hráče. Klienti vždy hrají za roli hráče – dealer je pouze na

serveru, nedávalo by smysl aby klient "nehrál", protože to jestli dealer bere či ne je plně určeno pravidly – hráč hrající jako dealer by tak neměl žádné rozhodnutí.

Po rozdání karet hráčům pošle server prvnímu hráči požadavek, že je na tahu. Pokud hráč do 60 sekund nevykoná žádnou akci, je automaticky použit **stand** a hraje další hráč. Po dohrání všech hráčů vygeneruje server výsledné karty dealera a odešle klientům výsledky. Lobby po skončení hry přejde do stavu **Finished** a čeká cca 30s než přejde opět do stavu Waiting.

4.1.4 Řešení problémů s připojením

Pokud se klient kdykoliv odpojí, server si zachová jeho stav. Při stavu, kdy si klient prohlížel herní místnosti, nebo byl v lobby, když byla ve stavu **Waiting** (čekala na všechny pro potvrzení hraní), nebo **Preparing** (čekala na všechny na uzavření sázky) se nic obnovovat nemusí – lobby automaticky klienta odpojí – nemá smysl blokovat místo pro někoho kdo nepotvrdil že bude hrát nebo ani neodeslal požadavek pro start hry.

Pokud je lobby ve hře – **In-game** popř. zobrazuje výsledky – **Finished**, je odpojený klient – ať už vlivem špatného připojení, nebo i zavření okna zachován, dokud hra neskončí. Poté je z lobby odstraněn.

Klient také může v průběhu hry kliknout na tlačítko **Leave Game**, které ho přidá na tzv. **skip list** – seznam uživatelů, kteří budou automaticky přeskočeni – až dojde k jeho tahu je instantně použit stand a pokračuje další hráč popř. dealer.

4.2 IMPLEMENTACE KLIENTA

Aplikaci klienta jsem implementoval v programovacím jazyce **Java** (Java 13), pro sestavení celé aplikace je použit sestavovací nástroj **Maven**. Projekt obsahuje dvě knihovny – pro vizualizaci hry jsem použil knihovnu **JavaFX 13** a pro lepší přehlednost jsem použil anotační knihovnu **Lombok**, která slouží pro generování getterů, setterů a základních design patternů.

Klient je narozdíl od serveru **vícevláknový**. Obsahuje jedno vlákno pro grafické prostředí – main, jedno pro čtení zpráv a jedno pro kontrolu připojení. Klientská data jsou uložena ve třídě **Client**, ta obsahuje zejména metody pro přípravu frontendu a připojení se k serveru. Při startu programu se spustí konstruktor klienta, který připraví scénu s přihlášením uživatele – **LoginScene**. Aplikace obsahuje několik dalších scén, které se připraví v závislosti na stavu klienta – **Lobbies** je scéna se seznamem všech herních místností, **Lobby** je scéna místnosti a **Game** je scéna s hrou. Jednotlivé scény jsou upravovány pomocí **controller**ů – třídy, které mají metody pro úpravu komponent ve scéně, které volá třída Client v závislosti na požadavcích serveru nebo uživatele.

4.2.1 Připojení na server

Klient při přihlašování zadá své uživatelské jméno a ip adresu s portem serveru. Při úspěšném připojení se spustí další vlákna programu. První takové vlákno, které je pro funkčnost aplikace potřeba je čtení zpráv od serveru. To je vytvořeno ve třídě **MessageReader**. MessageReader čte data ze serveru a reaguje na ně. Pokud jakákoliv data nejsou validní, automaticky klienta odpojí.

Synchronizaci s hlavním vláknem jsem provedl JavaFX metodou *Platform.RunLater()*. Ta nám zajistí, že se jednotlivé akce budou vykreslovat popořadě. Přečtená data se stejně jako u serveru mapují na třídu **TCPData**, která opět používá hashovací mapu pro deserializaci, implementace je totožná.

Druhé vlákno, které se spustí při připojení na serveru je implementováno ve třídě **PingService**. To zajišťuje kontrolu připojení k serveru – každou sekundu posílá ping dotaz na socket serveru a pokud není od serveru obdržená zpráva do několika vteřin, celý klientský socket smaže a zkouší se připojit znovu. Pokud se nepřipojí do tří pokusů upozorní uživatele přes třídu Client o ztracení spojení.

4.2.2 Odesílání zpráv

Odesílání zpráv serveru řeší třída **MessageWriter**, která je vlastně jenom obalovací třída output streamu socketu. Obsahuje tedy pouze metody s jednotlivými zprávami pro server.

5 UŽIVATELSKÁ DOKUMENTACE

5.1 PŘEKLAD A SPUŠTĚNÍ SERVERU S KLIENTEM

Pro překlad serveru je nutný kompilátor **GCC** a nástroj **CMake**, server je funkční pouze na operačním systému **Linux** – pro Windows je potřeba nainstalovat **Windows Subsystem Linux 2.** K překladu pak stačí pouze v příkazové řádce přesměrované ve složce projektu zadat příkazy:

```
Cmake .
make
```

Samotnou aplikaci serveru pak spustíme pomocí příkazu:

```
./UPSServer
```

Můžeme využít i argumenty pro spuštění serveru na nějaké ip adrese, portu a zvolení počtu místností. Následující příkaz se pokusí spustit server na localhost na portu 4431 s 5 herními místnostmi:

```
./UPSServer 127.0.0.1 4431 5
```

Můžeme použít i **dva argumenty**, kdy **první je číslo portu** a **druhý je počet herních místností**. IP adresa bude pak nastavena na localhost, pokud použijeme pouze jeden argument, program automaticky očekává pouze číslo portu a nastaví ip na localhost a počet herních místností na 3.

Pro překlad klienta je potřeba mít Javu 13 a vyšší a maven. Vytvoření jaru provedeme pomocí příkazu:

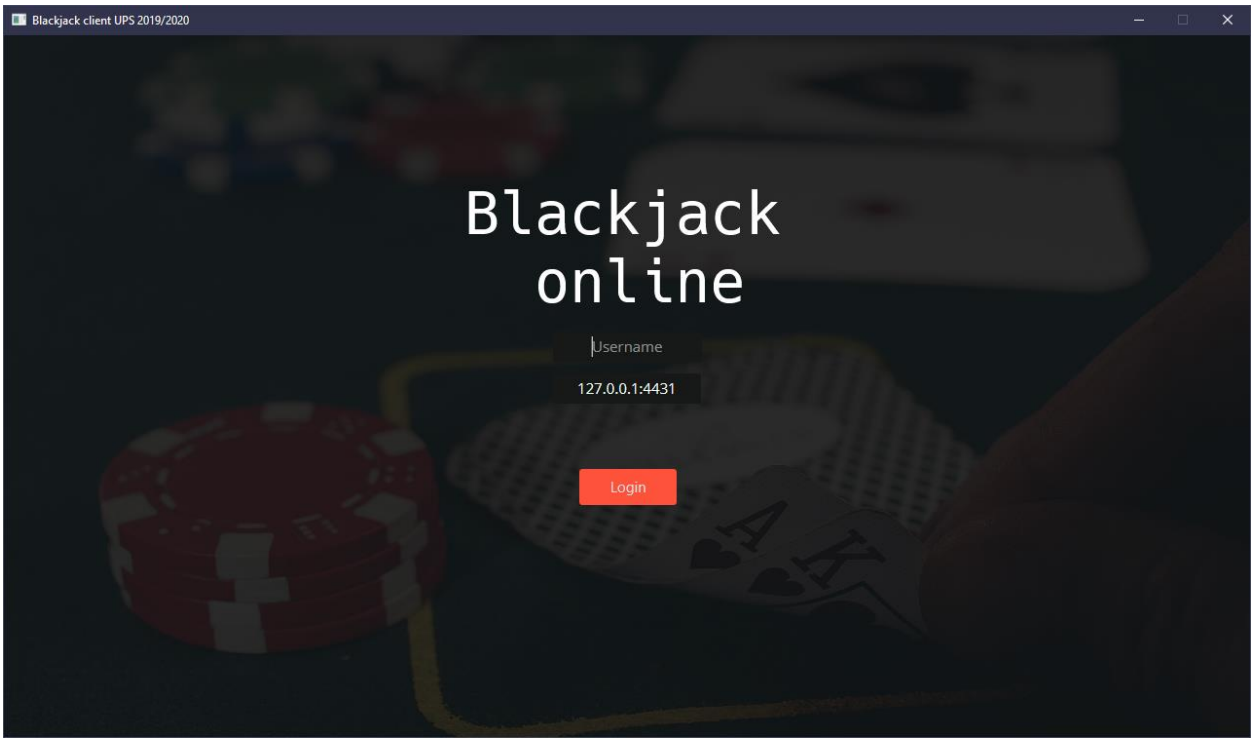
```
mvn clean install package
```

Spuštění pomocí příkazu:

```
java -jar jar/UPSClient.jar
```

5.2 Obsluha GUI

Po spuštění programu klienta se zobrazí menu se zadáním adresy a uživatelského jména:

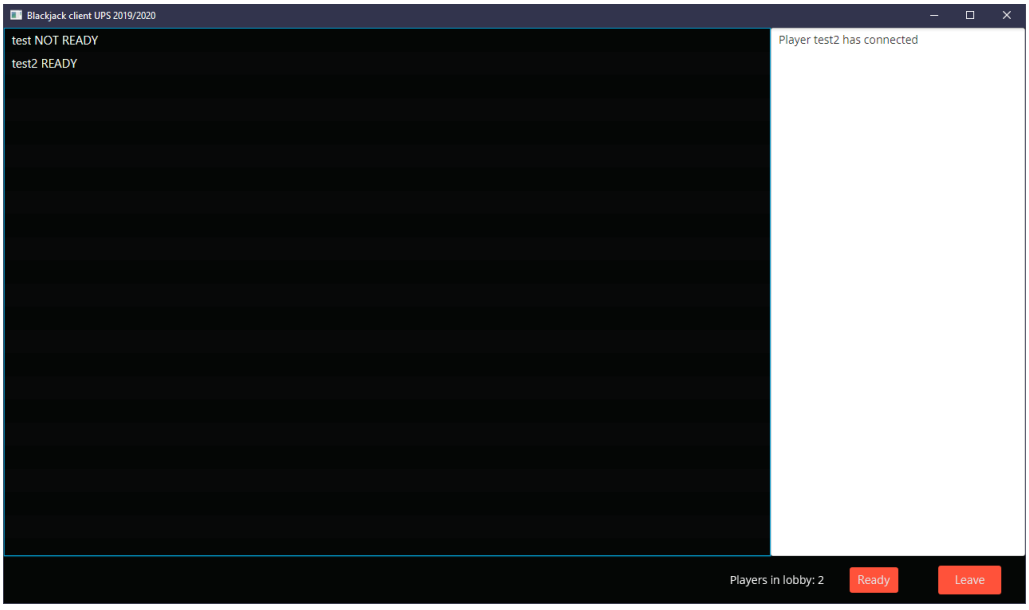


Obrázek 2 - Login scéna

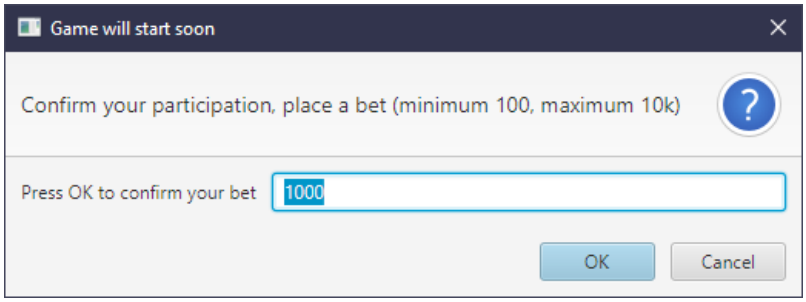
Po úspěšném přihlášení si uživatel může zvolit herní místnost, kterou zvolí kliknutím na tlačítko **Join Lobby**:

Obrázek 3 - scéna se seznamem herních místností

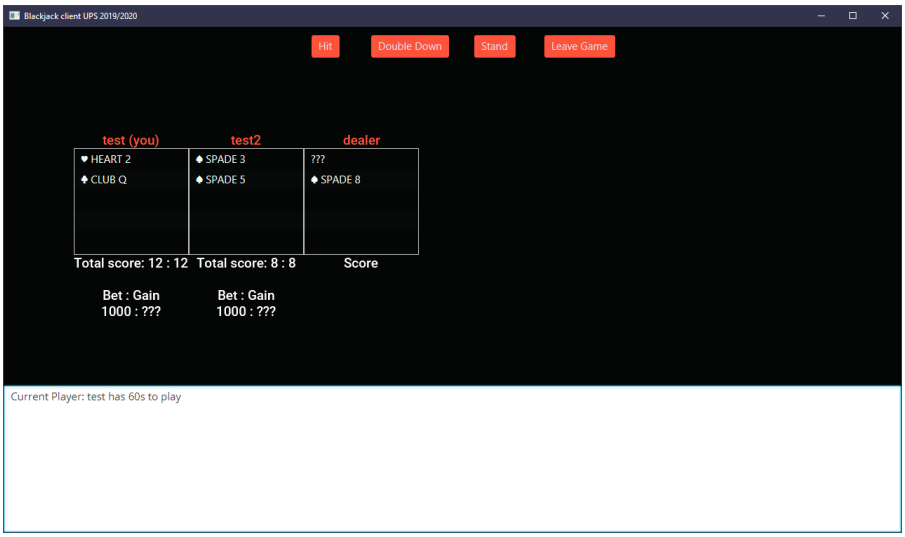
Po připojení do místnosti se hra spustí, pokud v ní jsou alespoň dva hráči, kteří potvrdili pomocí **READY** tlačítka a do časového limitu uzavřeli svoji částku:



Obrázek 4 - Scéna s herní místností



Obrázek 5 - Dialogové okno se sázkou



Obrázek 6 - Scéna se hrou

Hráči si poté vybírají z akcí pokud jsou na řadě. Taktéž mohou hru opustit – v takovém případě budou přeskočeni a nemohou se již do hry vrátit. Po skončení hry jsou opět vráceni do lobby.

6 ZÁVĚR

Aplikace je funkční pro malé množství uživatelů, aby fungovala pro větší počet, museli bychom použít jiný způsob řešení – select neumožňuje více než 1021 připojení.

Práce splňuje zadání – umí zpracovávat požadavky několika klientů najednou, herní místnosti jsou na sobě nezávislé. Server nepadá při neplatném vstupu, stejně tak jako klient.

Klient lze spustit na Windows 10, Linux i macOS za předpokladu že je na systému nainstalovaná Java 13 a maven (pokud potřebujeme znovu sestavit jar, jinak stačí poslední verze JRE).

Server lze spustit pouze na OS Linux popř. Windows za předpokladu, že použijeme WSL 2.