PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

BAKALÁŘSKÁ PRÁCE

Síťová hra "člověče, nezlob se!"

2011 Adam Dohnal

Anotace

Člověče, nezlob se je společenská desková hra pro více hráčů. Cílem této bakalářské práce je vytvořit serverové rozhraní, umožňující komunikaci klientů mezi sebou, včetně chatování, zakládání a sledování her. Aplikace je napsána v jazyce C# pod platformou .NET. Hra nabízí uživatelům příjemné uživatelské rozhraní s velkým množstvím herních plánů až pro šest hráčů.

Děkuji Mgr. Petru Osičkovi za odborné rady a vedení při zpracování této bakalářské práce.

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

Cílem této bakalářské práce je vytvořit dvě nezávislé aplikace. Server, který se spustí jako konzolová aplikace a na určitém portu začne naslouchat příchozí spojení od klientů. Součástí programu je textový soubor, umožňující správci serveru zadat IP adresy, se kterými by se spojení nemělo navázat. Druhá aplikace je klient s grafickým uživatelským rozhraním, který umožní uživatelům připojit se na server, komunikovat s ostatními uživately, zakládat hry a samozřejme obsahuje implementaci hry člověče, nezlob se.

Cílem nebylo vytvořit jednoduchou deskovou hru a umožnit jí hrát po síti, ale zaměřit se spíše na serverouvou část. S tím souvisí mnoho problémů jak co nejefektivněji zpracovávat požadavky klientů, tak aby server nebyl příliš zahlcen nebo neplýtval zdroji. Více k této problematice uvedu v sekci programátorská dokumentace. Hru člověce, nezlob se jsem si vybral proto, že je zároveň populární a lehká na implementaci, ale zároveň umožňuje hru až pro šest hráčů, což bylo pro mě mnohem zajímavější než hry jen pro dva hráče.

V dalších částech práce bych chtěl objastnit cíle, které jsem si stanovil. Uživatelskou dokumentaci, aby každý uživatel věděl jak aplikaci správně používat a co vše je potřeba pro její chod. Většinu práce bych chtěl věnovat programátorské dokumentaci, popisující použité postupy a problémy, se kterými jsem se setkal při implementaci.

Na závěr úvodní části bych chtěl dodat, že jsem aplikaci vypracoval, tak abych co nejvíce oddělil jednotlivé logické celky od sebe, včetně prezentační části od logické, takže serverová část je zcela nezávislá na typu klientovi. Samozřejme, že pro jiný typ klienta by se musel napsat vlastní komunikační protokol a implementovat vlastní obsluhy reagující na požadavky klienta. Jádro, které je zodpovědné za přijímání, zpracování a odesílání dat je nezávislé a vytvořeno tak aby bylo efektivní, i když to tento typ klienta příliš nevyžaduje.

2. Požadavky na práci

Aplikace se bude skládat ze dvou hlavních částí:

1. serveru, ke kterému se budou moci hráči
   - přihlásit
   - vytvářet a připojovat se do her
   - komunikovat mezi sebou pomocí chatu
2. grafického klienta, který bude komunikovat s uživatelem a serverem implementující deskovou hru člověče, nezlob se

Další požadavky:

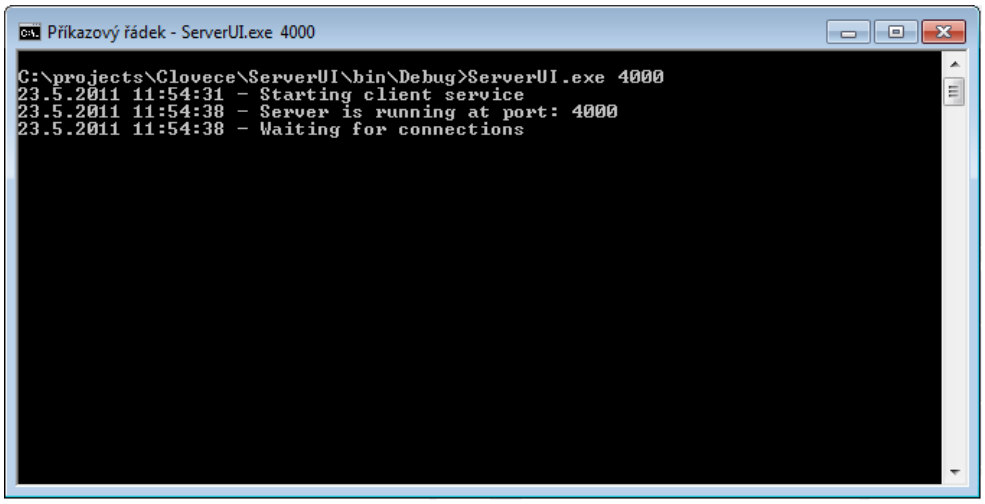
- podpora hry 2-6 hráčů
- výběr z více herních plánů
- soukromé zprávy mezi hráči

3. Uživatelská dokumentace

V této kapitole popíšu spuštění a způsob práce s aplikací, jak se připojit k serveru a popíšu uživatelské rozhraní.

3.1. Spuštění serveru

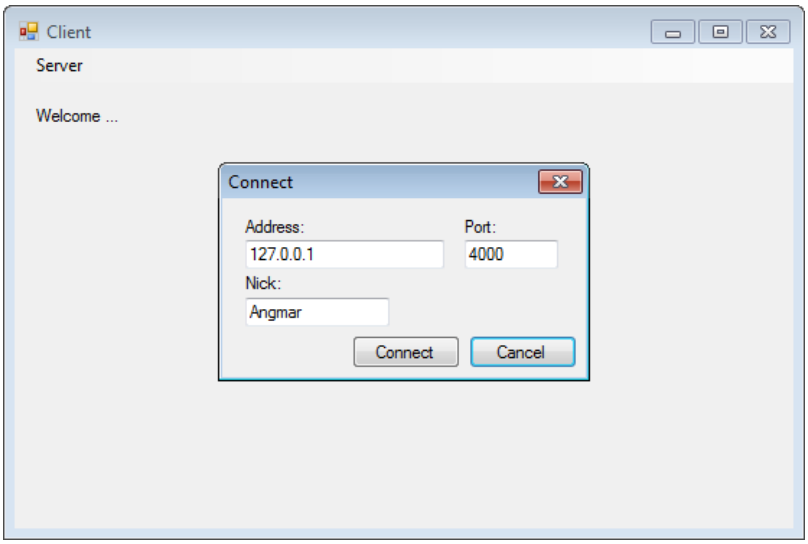
Server můžeme spustit dvěma způsoby. První způsob je spuštění spustitelného souboru ServerUI.exe. Server bude naslouchat defaultně na portu 4000. Pokud z nějakého důvodu budeme potřebovat server spustit na vlastním portu, můžeme server spustit z příkazové řádky jak ukazuje obrázek 1.. Pokud inicializace proběhla úspěšně, tak je server připraven zpracovávat požadavky od klientů.



Obrázek 1. Spuštění serveru z příkazové řádky

3.2. Spuštění klienta

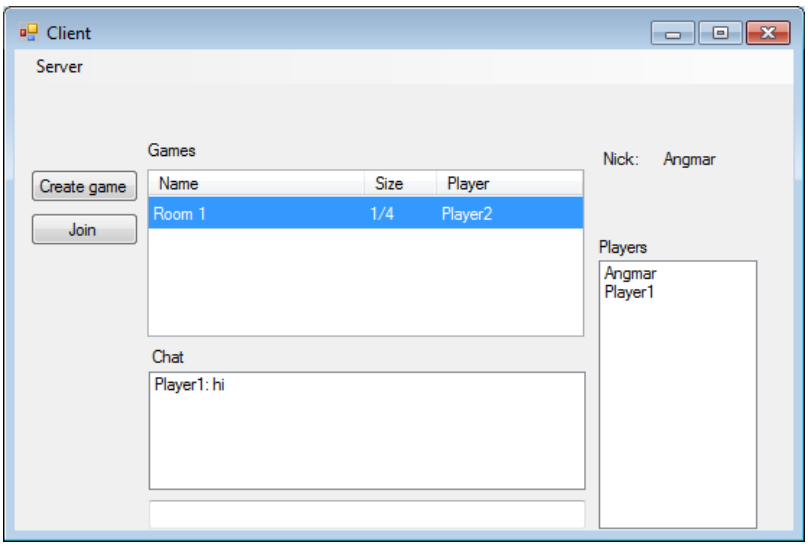
Klienta spustíme pomocí spustitelného souboru ClientUI.exe. Po spuštění se objeví standartní okno s menu. Menu Server umožňuje připojení/odpojení od serveru a ukončení aplikace. Po kliknutí na nabídku Connect se objeví dialog, který ukazuje obrázek 2.. V dialogu jsou nastaveny některé defaultní hodnoty: IP adresa lokálního rozhraní (loopback) a defaultní port 4000. Pokud by jsme server provozovali na jiném portu a na jiné veřejné IP adrese, bylo by nutné tyto údaje pozměnit.



Obrázek 2. Připojení k serveru

3.3. Hlavní kanál

Po úspěšném připojení se v klientovi zobrazí hlavní kanál jak ukazuje obrázek 3..



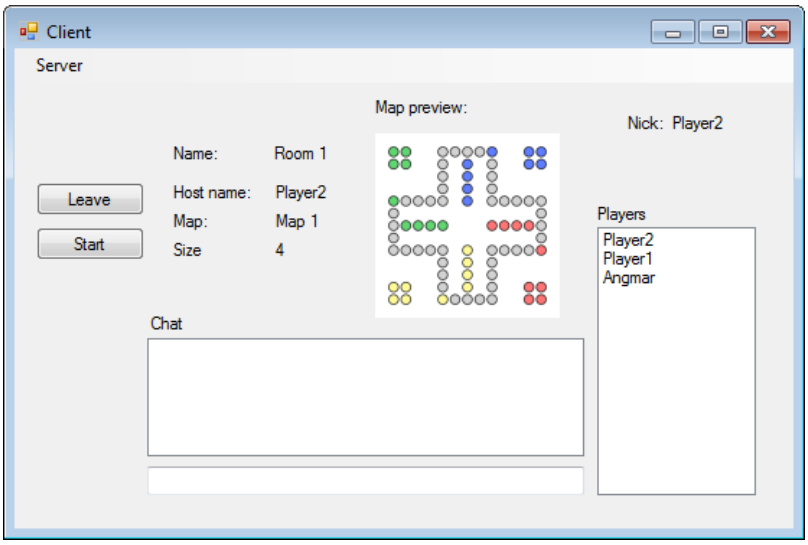
Obrázek 3. Hlavní kanál

V tomto kanále jsou vidět všichni hráči, kteří jsou připojeni do hlavního kanálu a je možné s nimi komunikovat pomocí chatu. Dále zde můžeme vidět seznam založených místností, které čekají na připojení ostatních hráčů. Do

místností je možné se připojovat, ale i vytvářet nové. Zajímavou funkcí je posílaní soukromých zpráv. Pokud víme jméno hráče, který je připojen na serveru, ale je v jiném kanále nebo nechceme aby tuto zprávu viděl někdo jiný, napíšeme do chatu `/w jméno_hráče zpráva`. Pokud chceme napsat soukromou zprávu v rámci stejného kanálu, můžeme dvakrát kliknout myší na hráče v seznamu nebo pravým tlačitkem vybrat možnost Whisper.

3.4. Místnosti

Po vytvoření nebo připojení do místnosti se klient přesune do nově vytvořeného kanálu. Tento kanál slouží ke spuštění hry. Hru může spustit pouze hráč, který místnost založil a v místnosti musí být minimálně dva hráči. V kanále je možné komunikovat pomocí chatu a posílání soukromých zpráv.

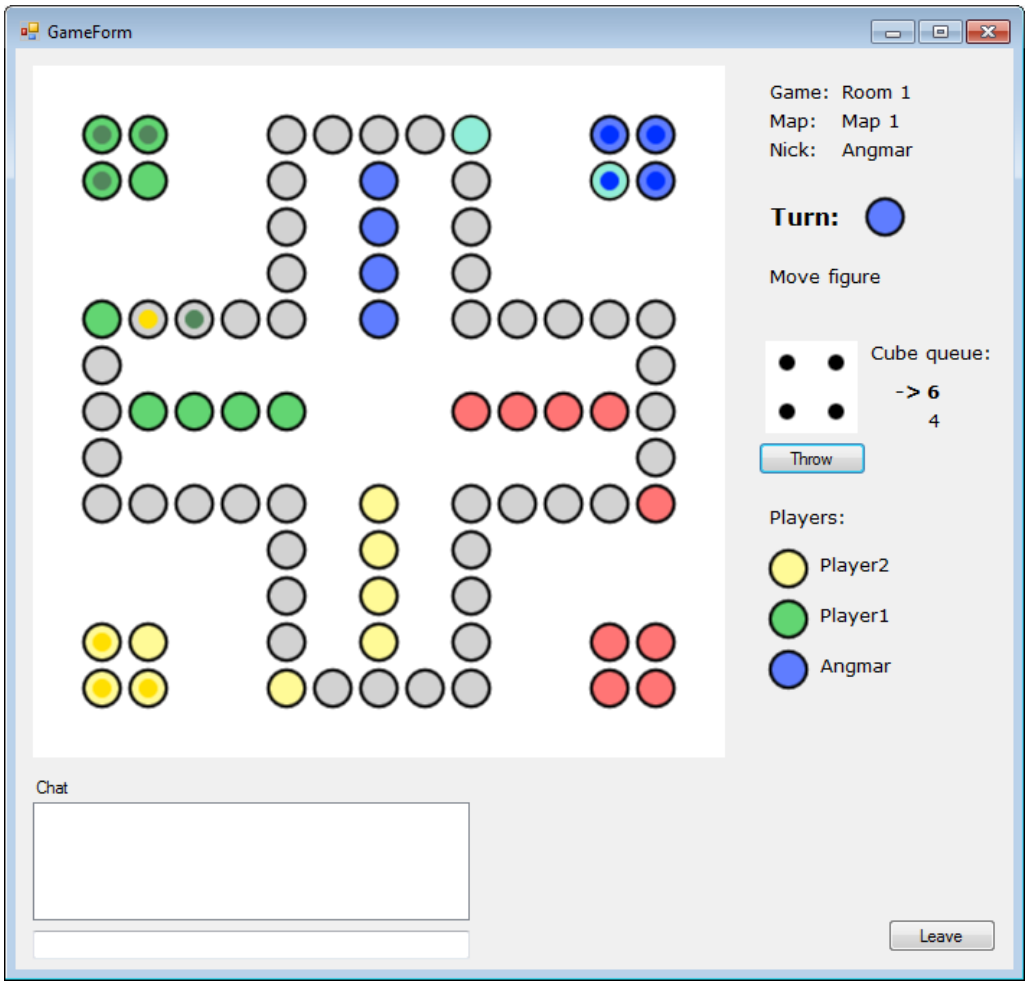


Obrázek 4. Místnost

3.5. Hra

Po spuštení hry v místnosti se serverové rozhraní schová a objeví se nové okno s inicializovanou hrou. V pravém horním rohu můžeme vidět název hry, zvolenou mapu a naše jméno. Každému hráči byla přidělena určitá barva. Seznam barev hráčů je zobrazen vpravo dole. Dále můžeme vidět, který hráč je na tahu a pod ním je zobrazena nápověda, která nám radí co máme dělat. Pokud se na nás dostane řada, musíme hodit kostkou. Klikneme na tlačítko Throw a server nám vygeneruje číslo. Máme určitý počet pokusů podle toho, jestli máme všechny

figurky v domečku nebo ne. Vpravo od hrací kostky se nám zaznamenává fronta hodů, takže když pak táhneme figurkou vidíme, jaké máme aktuální číslo hodu. Ve hře je možné zase chatovat a posílat soukromé zprávy. Hru je možné opustit a v tom případě se nám objeví hlavní kanál.



Obrázek 5. Hra

4. Programátorská dokumentace

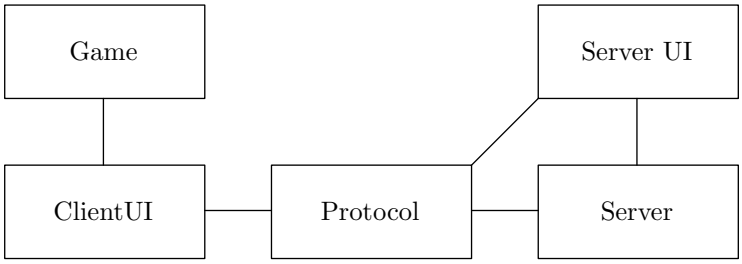
4.1. Architektura aplikace

Aplikace je napsána v programovacím jazyce C# pod platformout .NET. Byl použit objektově orientovaný návrh a snažil jsem se co nejvíce rozdělit projekt do více samostatných a nezavislých celků, které spolu spolupracují. Výhoda tohoto návrhu je větší čitelnost, odolnost vůči změnám (např. změna grafického rozhraní by neměla zasahovat do logiky aplikace) a hlavně znovupoužitelnost funkčních částí aplikace.

Celkem je tedy v projektu pět samostatných částí, jejichž závislosti ukazuje diagram na obrázku 6.. Části Server, Protocol a Game jsou preloženy jako knihovny a části ServerUI a ClientUI jsou spustitelné soubory s vlastním uživatelským rozhraním.

Následuje jednoduchý popis jednotlivých částí:

- Server - naslouchá na určitém portu připojení od dlientů, umožňuje filtrovat spojení na základě IP adresy a čte/zapisuje data do streamu. Data, která od klienta přečte posílá zpracovat do modulu ServerUI.
- ServerUI - zpracovává data od klientů a odesílá odpovědi.
- Game - obsahuje logiku hry, v tomto případě člověče, nezlob se
- ClientUI - vrstva mezi uživatelem a serverem. Zpracovává data, která přišla ze serveru, zobrazuje je a řídí chod hry.



Obrázek 6. Diagram závislostí modulů v aplikaci

4.2. Server

Cílem tohoto modulu je vytvořit vícevláknový TCP/IP server co nejvíce nezávislý na typu klienta a protokolu tak, aby byl použitelný a efektivní v co nejširší oblasti.

4.2.1. Základní požadavky

Po spuštění serveru by mela proběhnout jeho inicializace. S tím souvisí nastavení portu, načtení souboru obsahující IP adresy, které se budou filtrovat a další možnosti. Po úspěšné inicializaci by měl server naslouchat na daném portu připojení od klientů. Čekání na připojení klientů je blokující operace, což znamená, že program se zastaví a nebude pokračovat dokud se nějaký klient nepřipojí nebo se program neukončí. To znamená, že naslouchání musí probíhat ve vlastním vlákně.

Po navázání TCP/IP spojení s klientem se nad tímto spojením musí vytvořit tzv. network stream, čili datový proud umožnující čtení a zápis. Poslední částí je obsluha klientů, čili načtení dat, jejich následné zpracování jinou logickou částí aplikace a zápis odpovědi zpět klientovi.

Nastává otázka, jakým způsobem obsluhovat co nejvíc klientů zároveň tak, aby byla doba od přijmutí požadavku do jeho vyřízení co nejmenší a zároveň aby se efektivně využívali zdroje na serveru.

První způsob je mít pro každého klienta vlastní vlákno, ve kterém bude daná obsluha probíhat. Ve skutečnosti to znamená, že jakmile je akceptováno nové spojení, je vytvořeno nové vlákno a tomu je tohle spojení předáno jako parametr. Pro menší aplikace, kde se nepřipojuje mnoho klientů je toto řešení postačující. Tento způsob je jednoduchý na implementaci, ale nese s sebou určité nevýhody:

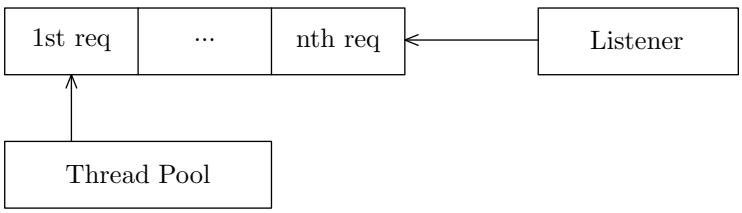
- může nastat situace, že se na server připojí mnoho klientů a počet vytvořených vláken bude rychle narůstat a celková odezva bude klesat. Navíc maximální počet vláken je v operačním systému nějak omezen. Při takové situaci je tato metoda dost neefektivní.
- druhý problém nastane tehdy, pokud je sice na server připojeno relativně málo klientů zároveň, ale často se spojení navazují a ukončují. Příkladem může být webserver, který přijme požadavek, vratí odpověď a ukončí spojení. Problém spočívá v tom, že vytváření a uvolňování vlákem je časově náročná operace.

4.2.2. Thread pooling

Lepší metoda obsluhy klientů je tzv. thread pooling. Jde o to, že máme frontu požadavků, na jejíž konec jsou vkládány požadavky od klientů v takovém pořadí v jakém přišli. Dále ještě před spuštěním serveru vytvoříme určitou množinu vláken, které přistupují do fronty, kde vyberou první požadavek, obslouží ho a jdou na další. Je důležité, aby přístup do fronty byl synchronizovaný, tzn. aby nedošlo k tomu, že více vláken vezme současně stejný požadavek. Tato metoda částečně řeší problém s neustálým vytvářením a uvolňováním vlakém po připojení nového klienta. Klienti se můžou připojovat a odpojovat libovolně, ale množina vláken zůstane stejná. Napsal jsem shchválně částečně, protože množina, která ma staticky daný počet vláken může mít vysokou odezvu.

Otázka tedy zní, jak určit velikost množiny vláken tak, aby nedocházelo ke zbytečnému vytváření a uvolňování vláken, ale doba čekání ve frontě byla co nejmenší. Představme si příklad, že množina obsahuje 3 vlákna a ve frontě čeká na vyřízení 10 požadavků. Doba čekání posledního požadavku tedy bude zhruba čtyřikrát větší než průměrná doba vyřízení požadavku. Kdyby jsme měli množinu, která obsahuje 10 vláken, doba čekání by nebyla téměř žádná. Může nás napadnout vytvořit ze začátku mnoho vláken a později se vyhneme vytváření dalších a doba čekání ve frontě bude malá, ale nastavá další problém. Co když budou k serveru připojeni jen 2 klienti a počet vláken bude 20. Obsluhu těchto dvou klientů by zvládly 1-2 vlákna podle charakteru aplikace a zbytek by nic nedělal. Takovému plýtvání zdrojů je také dobré se vyhnout a tím se dostaváme k tomu, že nejlepší způsob je dynamická množina vláken, tedy počet vláken se za běhu mění podle velikosti fronty a doby zpracování požadavku od klienta.

Algoritmů, které řeší dynamické přidávání a odstraňování vláken v množině existuje celá řada. Něco mají společně a v něčem se liší podle typu aplikace. Obecně můžeme tedy cíl takového algoritmu definovat jako omezení doby čekání na minimum, minimalizovat počet vláken, které jsou v množině zbytečné a zabránit častému vytváření nebo odstraňování vláken v množine.

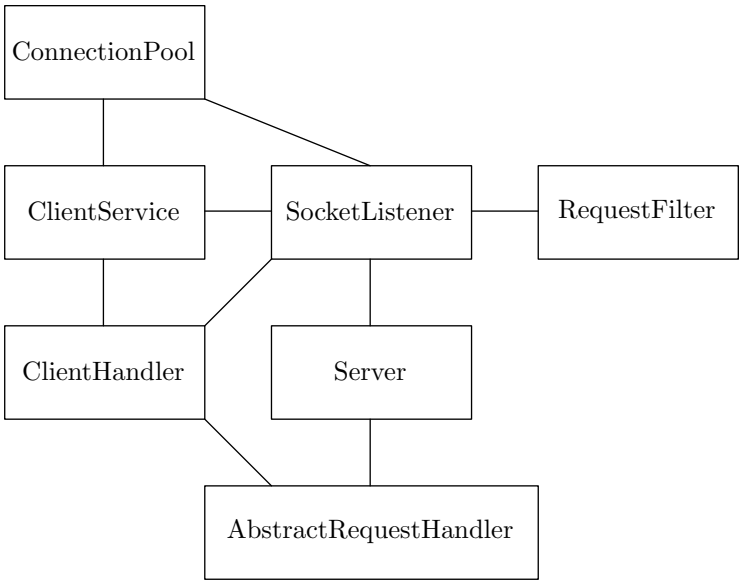


Obrázek 7. Thread Pooling

4.2.3. Implementace

Při implementaci jsem si vybral metodu thread pooling s jednoduchým algoritmem dynamického vytváření a uvolňování vláken v množině. Algoritmus zjišťuje velikost fronty požadavků, vydělí ji aktuálním počtem vláken a výsledek vynásobí průměrnou dobou zpracování požadavku, kterou si sama vlákna počítají. Výsledkem je zhruba maximální doba čekání požadavku. Pokud doba přesahuje určitou mez je potřeba vytvořit další vlákna, tak aby doba čekání ve frontě klesla pod danou hranici. Dopočítá se kolik je potřeba vytvořit vláken a ty se vytvoří. Naopak pokud do je fronta daleko menší než počet vláken, vlákna se uvolní. Dále je stanovena hranice mininum a maximum vláken, které není možno překročit.

Jěště než popíšu všechny třídy modulu, chtěl bych říct něco a zpracování požadavků. V tomto modulu je definována abstraktní třída, která zodpovídá za zpracování požadavků a odeslání odpovědí. Je čistě abstraktní takže programátor musí tuto třídu zdědit, doplnit vlastní protokol a vytvořit vlastní handler pro každou zprávu. V modulu je pak singleton, který má v sobě právě tuto uživatelsky definovanou třídu (tato třída se musí předat v konstruktoru při vytváření instance modulu), které bude modul přeposílat všechny přijaté data. Tím se použití velice zjednodušuje. Programátorovi stačí zdědit třídu, napsat svoje handlery, vytvořit novou instanci serveru a tuto třídu mu předat a vše bude fungovat. O tom jakou strukturu ma požadavek a o tom jak vypadá komunikační protokol bude řeč v některé z dalších kapitol.



Obrázek 8. Diagram závislostí v modulu Server

Client Handler

Client handler je třída, která obaluje spojení mezi serverem a klientem. Poskytuje obsluhu klienta, tzn. čtení a zápis do datového proudu.

Atributy

- `int id` - identifikátor, který se automaticky přiděluje po připojení klienta na server, je to jediná možnost jak přesně identifikovat, který objekt obsluhuje daného klienta.
- `TcpClient clientSocket` - spojení klienta ze serverem.
- `NetworkStream networkStream` - datový proud umožňující čtení/zápis.
- `string ip` - IP adresa rozhraní, ze kterého se klient připojil.
- `byte[] data` - pole bytů, kam se ukládájí příchozí data od klienta.

Metody

- `Constructor(TcpClient client)` - přidělí se ID nového spojení a vytvoří se network stream.
- `void Process()` - hlavní metoda obsluhy klienta. Pokusí se z network streamu načíst data a předat je do funkce ProcessDataRecieved, popř. pokud dojde k vyvolání vyjímky ukončí spojení a informuje o tom okolí.
- `void ProcessDataRecieved()` - data, která byli načteny rozdělí na příslušné požadavky a ty pak odešle uživateli definovanému objektu, který tyto požadavky dále zpracuje.
- `void Close()` - uzavře network stream a ukončí spojení ze serverem.
- `void SendResponse(Request response)` - odešle odpověď klientovi.

Client Connection Pool

Fronta objektů typu ClientHandler.

Request Filter

Udržuje v sobě seznam IP adres, které filtruje a nenechá připojit. Dané IP adresy načítá ze vstupního souboru, který je uložen ve stejném adresáři jako server a jeho jméno bere jako parametr.
Atributy

- `string filename` - jméno souboru s IP adresami.
- `List<string> bannedIPs` - seznam IP adres

Metody

- `Conctructor(string filename)` - pokud soubor existuje zavolá metodu Load.
- `void Load()` - načte ze souboru IP adresy a vloží je do seznamu.
- `void Save()` - uloží seznam IP adres do souboru.
- `void Add(string ip)` - přidá IP adresu do seznamu.
- `void Remove(string ip)` - odebere IP adresu ze seznamu.
- `bool IsBanned(string ip)` - zjistí jestli je daná IP adresa v seznamu.

Client Service

Tato třída má na starosti obsluhu fronty požadavků a spravuje množinu vláken.
Atributy

- `ClientConnectionPool connectionPool` - fronta požadavků.
- `Thread[] threads` - množina vláken.
- `int NumOfThreads` - počet vláken.

Metody

- `Constructor(ClientConnectionPool connPool)` - předání fronty
- `void Start()` - vytvoří a spustí vlákna, která začnou vykonávat metodu Process.
- `void Process()` - pokud není fronta prázdná, načte první požadavek (ClientHandler) a zavolá jeho metodu Process. Přístup do fronty je synchronizován pomocí zámku.
- `void Stop()` - počká až všechny vlákna dokončí aktuální požadavek a ukončí je, následně vyprázdní frontu požadavků a odešle všem klietům, že server se ukončil.

AbstractRequestHandler

Čistě abstraktní třída, která slouží ke zpracovávání požadavků od klientů. Programátor musí vytvořit vlastní instanci třídy, která tuto třídu dědí a musí přepsat její jedinou metodu.
Metody

- `virtual void HandleRequest(ClientHandler client, Request req)`

RequestHandler

Třída je navržená jako vzor singleton tak, aby byla přístupná z celého modulu a byla vytvořena jen jedna instance, která v sobě zapouzdřuje objekt uživatelsky definované třídy na obsluhu požadavků.

Socket Listener

Vstupní bod mudulu, který vytvaří frontu, množinu vláken, filtruje příchozí spojení, zabaluje je do ClientHandler objektů a ty následně vkládá do fronty.
Atributy

- `int port` - port, na kterém server naslouchá
- `string banlist` - jmeno souboru obsahujici seznam filtrovanych IP adres
- `RequestFilter filter` - filter IP adres

Metody

- `Constructor(int portNumber, string banList)` - nastaví port, vytvoří filter a vloží do něj IP adresy ze souboru
- `void StartListening()` - vytvoří prázdnou frontu, inicializuje ClientService a začne naslouchat na portu nové spojení. Po obdržení požadavku na spojení zjistí IP adresu klienta a zkontroluje, jestli není obsažená ve filtru. Poté vytvoří instanci třídy ClientHandler nad tímto spojením a umístí jej do fronty.
- `void Start()` - vytvoří nové vlákno, které začne vykonávat metodu StartListening()
- `void Stop()` - zastaví ClientService a ukončí naslouchání
- `void Restart()` - restartuje server

4.3. Protocol

Komunikační protokol je důležitá část v klient-server aplikaci. Určuje jakým způsobem jsou data uspořádána na aplikační vrstvě v paketu a tím pádem jak s nimi má aplikace naložit. Struktura dat může být libovolná, ale obě strany, jak klient, tak server by měli strukturu znát a vědět jak data zpracovat. Proto je tento modul přeložen jako knihovna, kterou využívá jak server tak klient. V této kapitole bych rozvedl strukturu komunikačního protokolu, kterou jsem navrhl, uvedl a popsal zprávy z parametry, které se můžou přenášet.

4.3.1. Struktura

Jak jsem už uvedl, struktura musí být taková, aby se určitým způsobem cílová aplikace dozvěla o jakou zprávu jde, popřípadě kde jsou uloženy její parametry a kde končí. V mém projektu jsem si vybral jednoduchý způsob, který data obaluje do tagů připomínající jazyk HTML.

Obecně pak data vypadají takto:

```
<h>message_code</h><p>param 1</p><p>param 2</p>...<p>param n</p><d>
```

Tag `<h></h>` obsahuje kód zprávy, kterou odesílá a za ní jsou v tazích `<p></p>` uloženy parametry zprávy. Občas se může stát, že v jednom paketu se odešle více než jedna zpráva a cílová aplikace musí jednotlivé zprávy od sebe odlišit. K tomuto účelu slouží tag `<d>`.

Aby byla práce cílové aplikace ze strukturou těchto dat co nejlehčí, modul definuje pomocnou třídu Request, která umožňuje snadno zpracovávat a vytvářet zprávy takovéto struktury.

Request

Tato třída zapouzdřuje strukturu dat zprávy před svým okolím a pomocí svého rozhraní umožňuje aplikaci jednoduše zpracovávat a vytvářet takévéto zprávy aniž by programátor něco vědel o struktuře jakým se data budou odesílat. Pokud by programátor chtěl vytvořit vlastní strukturu zpráv, musel by přepsat tuto třídu, protože ostatní moduly, které data zpracovávají nebo vytváří se právě odkazují na tuto třídu.
Metody

- `int GetHeader()` - vrací kód zprávy, uložený mezi tagy `<h></h>`.
- `List<string> GetParams()` - vrací seznam všech parametrů, uložených mezi tagy `<p></p>`.

- `Request SetHeader(int code)` - nastaví kód zprávy.
- `Request AddParam(string param)` - přidá nový parametr zprávy.
- `Request End()` - ukončí zprávu, přidá tag `<d>`.

4.3.2. Protokol zpráv

Protokol definuje standard, podle kterého probíhá komunikace a přenos dat mezi dvěma koncovými body. V našem případě komunikace probíhá pomocí zpráv, které jsou formátovány podle výše uvedeného způsobu a přenášeny TCP segmentem. Oba konce musí přesně vědět jaké zprávy mají příjimat a co s nimi dále dělat. V této části vyjmenuji důležité zprávy i s parametry pomocí nichž probíha komunikace klientů.

- `AUTH_REQUEST nick`
  - klient žádá server o autorizaci
  - server odpoví:
    - `AUTH_RESPONSE_FAIL nick`
      - pokud je na serveru přihlášen uživatel ze stejným jménem
    - `GLOBAL_CHANNEL`
      - pokud autorizace proběhla úspěšně a hráč je přesunut do hlavního kanálu
    - `IP_BANNED nick`
      - pokud je hráč na seznamu filtrovaných IP adres
- `SERVER_QUIT`
  - server se vypíná, dá vědět všem připojeným klientům
- `PLAYER_ADD nick`
  - pokud se hráč připojí do kanálu jsou mu odeslány všechny jména hráčů v kanálu
- `PLAYER_JOIN nick`
  - pokud se hráč připojí do kanálu, všem hráčům v kanálu se odešle, že se připojil nový hráč
- `PLAYER_LEFT nick`
  - pokud se hráč odpojí z kanálu, všem hráčům v kanálu se odešle, že se hráč odpojil
- `MESSAGE nick text`
  - hráč odešle zprávu na kanál, zpráva je následně rozeslána všem hráčům na kanále

- `CREATE_ROOM name map hostname size`
  - klient odešle žádost na server o vytvoření nové místnosti/hry
  - server odpoví:
    - `CREATE_GAME_FAIL`
      - pokud místnost ze stejným jménem už existuje
    - `CREATE_GAME_OK`
      - pokud vytvoření místnosti proběhlo úspěšně, hráč hostname je odebrán z hlavního kanálu a přidán do nově vytvořené místnosti
- `ROOM_CRATED name size players size hostname`
  - po vytvoření nové místnosti je tato zpráva odeslána všem hráčům v hlavním kanále
- `JOIN_ROOM name`
  - hráč odešle žádost o připojení do místnosti
  - server odpoví:
    - `JOIN_ROOM_FAIL`
      - pokud počet hráčů v místnosti je roven maximálnímu počtu
    - `JOIN_ROOM_OK`
      - hráč je úspěšně připojen do místnosti
- `ROOM_UPDATE name players size`
  - po připojení nebo odpojení hráče v místnosti se tato zpráva odešle všem hráčům v hlavním kanále
- `LEAVE_ROOM`
  - hráč opustil místnost a je připojen na hlavní kanál
- `ROOM_HOST_LEFT`
  - pokud hráč opustil místnost a zároveň tuto místnost vytvořil, je tato zpráva odeslána všem hráčům v místnosti a jsou připojeni na hlavní kanál
- `ROOM_REMOVED name`
  - pokud místnost opustil hřáč, který ji vytvořil je místnost zrušena a tato zpráva je odeslána všem hráčům v hlavním kanále
- `GAME_START_REQUEST map players name player 1 ... player n`
  - klient žádá server o spuštění hry
  - server odpoví:
    - `GAME_START map players name player 1 ... player n`
      - je odeslána všem hráčům v místnosti

  - `GAME_RUNNING name`
    - je odeslána všem hráčům v hlavním kanále

- `GAME_END`
  - pokud ve hře zbývá poslední hráč sám, je mu odeslána tato zpráva

- `CUBE_NUMBER_GET`
  - klient žádá server o vygenerování čísla
  - server odpoví:

    - `CUBE_NUMBER number`
      - je odeslána všem hráčům ve hře

- `CHANGE_PLAYER`
  - je odeslána všem hráčům v místnosti

- `DO_MOVE x1 y1 x2 y2`
  - je odeslána všem hráčům v místnosti

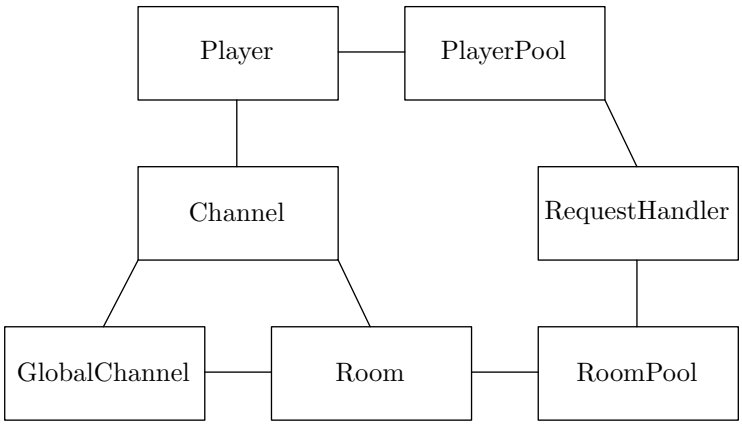
Seznam těchto zpráv je implementován jako enum, kde každý symbol ma svojí vlastní celočíselnout hodnotu.

4.4. Server UI

Modul ServerUI je výstupní částí serverové části aplikace. Je to konzolová aplikace využívají knihovny Server a Protocol. Při spuštění inicializuje a spustí SocketListener z modulu Server. Modul definuje logiku na serveru a implementuje obsluhu zpráv z modulu Protocol. Diagram tříd s jejich závislostmi je na obrázku 9.. Logikou serveru se myslí správa hráčů a místností na serveru.

Po přihlášení klienta k serveru a po jeho úspěšné autorizaci se vytvoří nová instance typu Player a uchová se v globální množině hráčů, která slouží pro rychlé hledaní hráčů podle identifikátoru jejich spojení. Každý hráč, který je připojen na serveru musí být v nějakém kanále. Po přihlášení na server je hráč automaticky připojen na hlavní kanál tzv. GlobalChannel. V tomto kanále hřáč mimo jiné může vytvářet a připojovat se místností, což jsou zase kanály. Po vytvoření místnosti je místnost přidána do množiny místnosti, která umožňuje rychlé vyhledávání podle názvu. Hráč, který místnost založil může hru spustit.

To znamená, že aby tento modul uměl zpracovávat požadavky klientů, musí si v paměti nějakým způsobem organizovat data o tom, jaký hráč je v jaké místnosti, jaké má práva apod.



Obrázek 9. Diagram tříd a závislostí v modulu ServerUI

4.4.1. Player

Tato třída představuje jednoho hráče, který je připojený na serveru. V objektu si drží nejenom informace jako jméno apod., ale i odkaz na objekt ClientHandler, který umožňuje odesílat zprávy po network streamu, který je asociován s tím to hráčem.

Atributy

- `string nick` - jméno hráče
- `ClientHandler client` - objekt obsluhy klienta
- `Channel channel` - odkaz na kanál, ke kterému je hráč připojen

4.4.2. PlayerPool

Úkolem této statické třídy je uchovávat všechny hráče na server v jedné kolekci, která by umožňovala rychlé hledaní hřáčů podle identifikátoru jejich spojení. Vždy když nám přijde zprava od určitého klienta známe identifikátor tohoto spojení a potřebujeme znát, o kterého hráče se jedná. Pro uchovávání jsem vybral kolekci Dictionary, která je vnitřně implementována jako hashovací tabulka kde klíč je identifikátor spojení a hodnota klíče je objekt typu Player, což umožňuje velice rychlé vyhledávání.

Atributy

- `Dictionary<int, Player> players` - hashovací tabulka obsahují všechny hráče na serveru

Metody

- `bool AddPlayer(Player player)` - pokusí se přidat hráče do kolekce, pokud v kolekci existuje hráč ze stejným jménem, funkce vráti false
- `void RemovePlayer(int ID)` - vymaže z kolecke hráče s identifikátorem spojení pokud existuje
- `Player GetPlayerById(int ID)` - podle identifik8toru spojení vrátí objekt hráče

4.4.3. Channel

Tato třída představuje abstraktní kanál na serveru. Definuje základní vlastnosti a funkčnost kanálu a ostatní třídy, které tuto funkčnost vyžadují musí tuto třídu dědit. Kanál si můžeme představit jako jakýsi kontejner v němž je obsažena určítá množina hráčů, kteří mají něco společné. Hráči se navzájem vidí a můžou spolu komunikovat pomocí chatu. Tato třída definuje události při připojení nového hráče na kanál, odpojení nebo odeslání zprávy. Tyto 3 vlastnosti jsou společné všem typům kanálu na serveru.

Atributy

- `List<Player> players` - seznam hráčů v kanále

Metody

- `virtual void AddPlayer(Player player)` - obsluha připojení hráče do kanálu. Všem hráčům v místnosti je odeslána zpráva `PLAYER_JOIN` se jménem hráče, který se připojil a jemu jsou pomocí zpráv typu `PLAYER_ADD` odesláni všichni hráči v kanále. Metoda je virtuální pro případ toho, že nějaká speciální třída by potřebovala pozměnit implementaci.

- `virtual void RemovePlayer(Player player)` - obsluha odpojení hráče z kanálu. Všem hráčům v kanále je odeslána zpráva `PLAYER_LEFT` se jménem hráče, který se odpojil. Metoda je rovněž virtuální.

- `void SendMessage(Player player, string message)` - nastane pokud hráč v kanále odeslal textovou zprávu. Zpráva je odeslána pomocí zprávy `MESSAGE` s danými parametry.

4.4.4. Room

Třída Room představuje místnost neboli hru na serveru. Tato třída dědí funkčnost od třídy Channel. Narozdíl od normálního kanálu má každá místnost maximální počet hráčů, které se do ní můžou připojit. Tento počet závisí na mapě, kterou zvolil hráč, který tuto místnost založil. Ten má navíc právo hru odstartovat a tím uzamknout místnost ostatním hračům, kteří ji vidí v hlavním kanále. Místnost se vytváří při požadavku klienta zprávou `CREATE_ROOM` a je zrušena pokud jí opustí poslední hráč.

Místnost má dvě fáze. První fáze nastává po vytvoření místnosti na serveru a místnost je v tzv. přípravné fázi, kdy se čeká na připojení ostatních klientů. Jakmile hráč, který místnost vytvořil odstartuje hru, fáze místnosti se změní na typ hra. Ve hře proti sobě hráči hrají hru a místnost v této fázi musí umět

zpracovávat zprávy, které souvisí s hrou. V této fází tato místnost mizí z dohledu ostatních hráčů v hlavním kanále, kteří by se do ní snad chtěli připojit.

Atributy

- `string name` - název místnosti
- `string map` - mapa
- `string hostname` - jméno hráče, který místnost vytvořil
- `int size` - maximální počet hráčů v místnosti
- `bool isGame` - příznak, určující jestli je místnost ve fázi hry

Metody

- `override void AddPlayer(Player player)` - k definici funkce v předkovi přidává vlastní implementaci, která provede aktualizaci místnosti v hlavním kanále
- `override void RemovePlayer(Player player)` - k definici funkce v předkovi přidává vlastní implementaci, která hráče, který se odpojil připojí do hlavního kanálu. Pokud je místnost ve stavu přípravy a hru opustil hráč, který ji založil, místnost je zrušena. Pokud je místnost ve stavu hry a hráč, který místnost opustil je předposlední, poslednímu hráči je odeslána zpráva `GAME_END`. Pokud místnost opustil poslední hráč, místnost je zrušena.
- `void GameStart(List<string> param)` - pokud hráč požádá o start hry, je změněna fáze místnosti a všem hráčům je odeslána zpráva `GAME_START`.
- `void PlayerThrowCube()` - poté co je serverem vygenerován hod kostkou je následně rozeslána všem hráčům zprávu `CUBE_NUMBER`.
- `void ChangePlayer()` - všem hráčům v místnosti je rozeslána zpráva `CHANGE_PLAYER`.
- `void DoMove(int fromX, int fromY, int toX, int toY)` - všem hráčům je rozeslána zpráva `DO_MOVE`.

4.4.5. RoomPool

Podobně jako PlayerPool tato třída slouží k uchovávání všech místností na serveru v hashovací tabulce, kde klíč je název místnosti a hodnota klíče je objekt typu Room. Třída umožňuje rychlé vyhledavání přidávání a odebírání místností z kolekce.

Atributy

- `Dictionary<string, Room> rooms` - hashovací tabulka

Metody

- `bool AddRoom(Room room)` - pokusí se přidat místnost do kolekce, pokud v kolekci existuje místnost ze stejným jménem, funkce vráti false
- `void RemoveRoom(Room room)` - odebere místnost z kolekce pokud existuje
- `Room GetRoomByName(string name)` - podle jména vrátí objekt typu Room

4.4.6. GlobalChannel

GlobalChannel je zvláštní typ kanálu, do kterého jsou automaticky hráči umístěni po prihlášení a autorizaci na serveru. Zvláštní je v tom, že mimo funkcí, které umožňuje obecná třída Channel, obsahuje seznam všech místností na serveru, které se nacházejí v přípravném stavu. V tomto kanále hráči můžou vytvářet nové místnosti nebo se připojovat do již vytvořených místností. Jelikož je hlavní kanál potřeba vždy jeden je tato třída implementována jako singleton.
Atributy

- `List<Room> rooms` - seznam místností

Metody

- `override void AddPlayer(Player player)` - po připojení hráče na hlavní kanál jsou mu odeslány všechny vytvořené místnosti pomocí zprávy `ROOM_CREATED`.
- `void AddRoom(Room room)` - přidá místnost do seznamu a odešle všem hráčům v hlavním kanále zprávu `ROOM_CREATED`.
- `void RemoveRoom(Room room)` - odebere místnosti ze seznamu a odešle všem hráčům zprávu `ROOM_REMOVED`.

- `void UpdateRoom(Room room)` - všem hráčům v hlavním kanále odešle zprávu `ROOM_UPDATE`.
- `void GameStart(Room room)` - všem hráčům v hlavním kanále odešle zprávu `GAME_RUNNING`.

4.4.7. RequestHandler

Tato třída dědí třídu AbstractRequestHandler z modulu Server. Jsou jí přeposílány všechny zprávy, které server obdržel od klientů a je zodpovědná za jejich zpracování a odeslání odpovědi. Třída implementuje jednu metodu, ve které jsou jako parametry předány zpráva a ClientHandler spojení, ze kterého zpráva přišla. Metoda ze zprávy vytáhne kód zprávy a její parametry. Podle typu zprávy zavolá přísušnou obsluhu s danými parametry.

Obsluhy

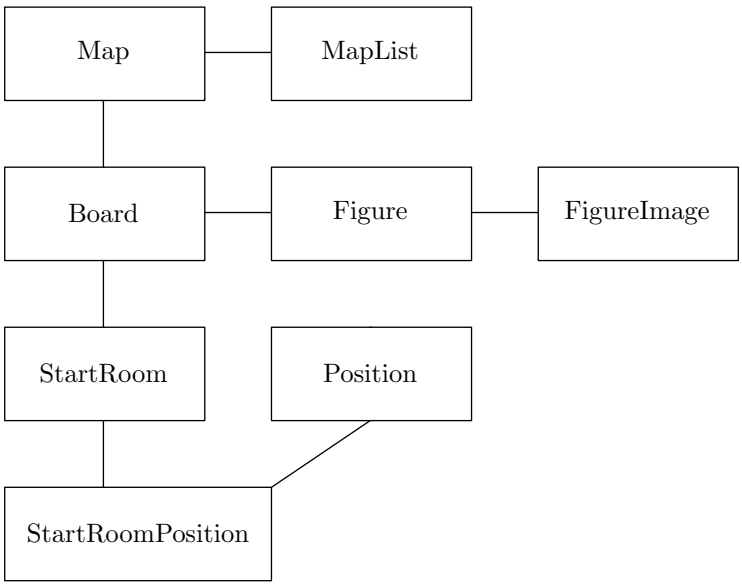
- `void AuthorizationHandler(ClientHandler client, List<string> param)` - provede autorizaci klienta
- `void CreateRoom(ClientHandler client, List<string> param)` - vytvoří na serveru novou místnost
- `void JoinRoom(ClientHandler client, List<string> param)` - přesune hráče z hlavního kanálu do místnosti
- `void LeaveRoom(ClientHandler client, List<string> param)` - přesune hráče z místnosti, ve které se nachází zpátky do hlavního kanálu
- `void GameStartHandler(ClientHandler client, List<string> param)` - v dané místnosti se odstartovala hra
- `void CubeNumberHandler(ClientHandler client)` - vygeneruje hod kostkou
- `void ChangePlayerHandler(ClientHandler client)` - v dané místnosti změní hráče na tahu
- `void DoMoveHandler(ClientHandler client, List<string> param)` - hráč vykonal tah v dané hře

4.5. Game

Funkcí tohoto modulu je implementace logiky hry, v našem případě deskové hry člověče, nezlob se. Jednou z částí tohoto modulu je způsob ukládání, načítání a práce z mapou, na které se hra hraje. Další částí je ukládání figurek na hracím poli, provádění tahů a kontrola pravidel hry. Než se pustíme do způsobu implementace jednotlivých tříd tak bych jen stručne poznamenal pravidla této hry.

Hru může hrát proti sobě 2 až 6 hráčů mezi sebou, přičemž každý hráč má odlišnou barvu figurek. Hra se hraje na určité mapě, která obsahuje cestu s políček, na kterém se nachází pro každého hráče domeček složený ze 4 políček. Každý hráč má k dispozici 4 figurky, které jsou na začátku hry umístěné na startovní pozici. Hráči se mezi sebou střídají. Hráč, který je na tahu, hází kostkou. Pokud má hráč všechny své figurku na startovní pozici má 3 hodky kostkou. V opačném případě pouze 1. Pokud hráč hodí číslo 6 má ještě jeden pokus. V případě, že hráč hodí kostkou číslo 6, může si jednu figurku nasadit. Cílem hry je dostat všechny figurky do domečku dříve než soupeři.

Modul tvoří 8 jednotlivých tříd, jejichž diagram je zobrazen na obrázku číslo 3. V následujích sekcích bych chtěl rozebrat implementaci logiky hry od struktury ukládání dat až po prácí s nimi.



Obrázek 10. Diagram tříd a závislostí v modulu Game

4.5.1. Map

Jak již název této třídy napovídá, jedná se o třídu reprezentující mapu hry. Jednotlivé mapy jsou uloženy v bínarním souboru určité struktury, o které bude reč posléze. Třída umožňuje pomocí jedné ze svých metod načíst data ze souboru a uložit si je do svých proměných tak, aby okolí mohla sdělit, na jaké pozici ma daný hráč startovní pozici, domeček nebo na jakou pozici se figurka dostane za předpokladu, že hráč hodil určité číslo. Instance této třídy tedy definuje strukturu jedné z map a logiku nechává na někom jiném.
Struktura souboru Mapa je uložena byte po bytu v binárním souboru, který je zkompilován jako zdroj s celým projektem. Tato volba neumožňuje dynamické přidávání dalších map za provozu aplikace a projekt by se musel překompilovat. Vytvořil sem určité množství map všech možných typů a nepředpokládal sem, že v budoucnu by byla potřeba přidávat mapy nové. Jak sem říkal informace jsou v souboru uloženy jako pole bytů. Pojďme se podívat jaké všechny informace tam jsou uložené:

- **velikost mapy** - počet políček na šířku a na výšku, celkem tedy 2B.
- **počet hráčů** - 1B.
- **startovní pozice** - startovní pozici tvoří 4 políčka pro každého hráče, příčemž každé políčko má souřadnici x a y, celkem tedy $pocet\_hracu * 4 * 2$ bytů.
- **délka cesty** - počet políček od místa, kde hráč nasadí figurku po poslední políčko v domečku.
- **cesty** - pro každého hráče je uložena cesta od políčka, kde hráč nasazuje figurku až po poslední políčko v jeho domečku. Cesta je uložena jako seznam políček s danými souřadnicemi.

Když už víme jak jsou data v souboru uložená, není problem je z tama načíst do paměti. Otázka zní jakou strukturu dat zvolit.

Atributy

- `string name` - název mapy
- `int size` - počet hráčů
- `int width` - šířka
- `int height` - výška

- `List<Position>[] startRoom` - seznam políček reprezentují startovní pozici pro každého hráče
- `List<Position>[] path` - seznam políček tvořící cestu každého hráče
- `byte[] data` - data mapy

Metody

- `void Load()` - načte strukturu mapy do atributů objektu z pole bytů
- `Position getStartPosition(int player)` - vrátí pozici, kde hráč nahazuje vlastní figurku
- `List<Position> getStartRoom(int player)` - vrátí seznam políček tvořící startovní pozici pro daného hráče
- `List<Position> getFinishPositions(int player)` - vrátí seznam políček tvořící domeček daného hráče
- `Position getMovePosition(Figure figure, int value)` - vrátí pozici, na kterou by se figurka dostala, kdyby hráč hodil dané číslo
- `bool isFigureInStartRoom(Figure figure)` - vrátí true, pokud je figurka na startovní pozici

4.5.2. MapList

Tato statická třída obsahuje kolekci všech vytvořených map. Slouží jako datový zdroj, který se pak snadno napojí na ovládací prvek ComboBox a umožní dynamické načítání map při změně kolekce.

Atributy

- `static List<Map> maps` - seznam map

Metody

- `static Map getMapByName(string name)` - vratí mapu na základě jejího jména

4.5.3. Figure

Instance této třídy představují jednotlivé figurky rozmístěné na aktuální mapě během hry. Každá figurka patří nějakému hráči a má svoji pozici.

Atributy

- `Position pos` - pozice figurky
- `int player` - identifikátor hráče, kterému figurka patří
- `FigurePosition mapPosition` - logická pozice figurky na mapě

4.5.4. FigureImage

Pomocná statická třída, která pro danou figurku vrací její bitmapu. U figurky zkoumá, kterému patří hráči, jestli je figurka označená a jaká je její aktuální pozice na mapě a podle toho vrátí bitmapu. Výsledná bitmapa má průhlednou bílou barvu. Všechny obrázky figurek jsou zakompilovány v projektu jako zdroje.
Metody

- `static Bitmap getFigureImage(Figure figure)` - vrátí pro danou figurku její bitmapu

4.5.5. Position, StartRoomPosition

Pomocné třídy reprezentují políčko na herní mapě a políčko na startovní pozici hráče. Třída Position definuje pouze atributy X a Y představující souřadnice na mapě. Třída StartRoomPosition dědí tyto vlatnosti a navíc přidává atribut, který udává jestli je pozice obsazená figurkou hráče.

4.5.6. StartRoom

Jednotlivé instance této třídy reprezentují seznam políček, na kterých jsou ze začátku hry umístěny hráčovi figurky. Třída poskytuje dvě pomocné metody pro snadnou manipulaci s figurkami.

Atributy

- `List<StartRoomPosition> positions` - seznam startovních pozic

Metody

- `StartRoomPosition getEmptyPosition()` - pomocná metody vracející první volnou pozici v seznamu
- `void moveToHome(Figure figure)` - umístí figurku na první volnou pozici
- `void moveFromHome(Figure figure)` - umístí figurku pryč ze startovní pozice

4.5.7. Board

Hlavní třída reprezentující desku hry umožňující manipulaci s figurkami a kontrolu pravidel. Obsahuje kolekci jednotlivých figurek a umožňuje hráči s nimi táhnout, přičemž kontrolue platnost tahu a ostatní pravidla např. vyhazování figurek, konec hry, nasazování atd.

Atributy

- `Map map` - aktuální mapa
- `int playersCount` - aktuální počet hráčů
- `List<StartRoom> startRooms` - seznam startovních pozic pro každého hráče
- `List<Figure> figures` - seznam jednotlivých figurek

Metody

- `Constructor(Map map, int playersCount)` - načte danou mapu a inicializuje figurky
- `void RemovePlayer(int player)` - v případě, že hrác opustil hru smažeme jeho figurky
- `void DoMove(int fromX, int fromY, int toX, int toY)` - provede tah figurky, která se nachází na pozici fromX, fromY na pozici toX,toY. V případě, že se na této pozici nachází figurka jiného hráče, je figurka vrácena zpět na startovní pozici.
- `bool IsStartRoomFull(int player)` - vrátí true pokud je startovní pozice plná, tzn. hráč má 3 hody kostkou.
- `bool winPlayer(int player)` - vratí true pokud hráč, který táhl figurkou vyhrál hru

4.6. Client UI

Posledním modulem projektu je modul Client UI. Je to klientská aplikace vystupující mezi uživatelem a serverem. Pomocí grafického rozhraní umožňuje uživateli komunikaci ze serverem a zobrazuje odpovědi. Modul se skládá ze dvou hlavních částí a několika formulářů.

4.6.1. Listener

První částí je tzv. Listener. Úkolem této třídy je načítat data ze serveru, přeposlat je ke zpracování a umožnit odeslání dat na server. Čtení dat probíha ve vlastním vlákně, aby neblokovalo smyčku GUI. Celkově je třída podobná třídě ClientHandler, který poskytoval opačný směr komunikace.

Atributy

- `TcpClient client` - klient, v našem případě server
- `NetworkStream stream` - datový proud nad spojením ze serverem
- `byte[] buffer` - buffer do kterého se ukládají příchozí data

Metody

- `void Listen()` - připojí se k serveru, autorizuje se a ve smyčce čeká na příchozí data, která zpracovává metodou ProcessDataRecieved.
- `void ProcessDataReceived()` - z bufferu načte jednotlivé zprávy a ty přepošle třídě ResponseHandler.
- `void Connect()` - hlavní metoda, vytvoří nové vlákno, které začne vykonávat metodu Listen().
- `void Close()` - uzavře spojení.
- `void SendResponse(Request response)` - odešle zprávu na server.

Handlery

Jelikož třída umožňuje odesílat zprávy na server, poskytuje několik handlerů, které se starají o spravné vytvoření zprávy a její odeslání na server a zpřehledňují tím GUI.

- `void Authorize()` - odešle zprávu o autorizaci se jménem hráče.
- `void SendMessage(string message)` - odešle zpravu na server se jménem hráče a textem zprávy.

- `void CreateRoom(string name, string map, string hostname, int size)` - odešle požadavek na vytvoření místnosti.
- `void JoinRoom(string name)` - odešle požadavek na připojení do místnosti.
- `void LeaveRoom()` - opustí místnost.
- `void GameStart(string map, int playersCount, string gameName, List<string> players)` - odešle požadavek na odstartování hry
- `void CubeThrow()` - odešle požadavek na hod kostkou.
- `void ChangePlayer()` - odešle požadavek na změnu hráče.
- `void DoMove(int fromX, int fromY, int toX, int toY)` - odešle požadavek na odehrání tahu.

4.6.2. ResponseHandler

Třída je velmi podobná tříde RequestHandler v modulu Server UI. Obsahuje jedinou metodu, které jsou předávany zprávy, které přišli ze serveru. Metoda nedělá nic jiného než že zprávy odešle určíté metodě v GUI, která zprávu obslouží. Jediný problém, který se tu nachází je jak zavolat metodu GUI z vlákna, které toto GUI nevytvořilo. Platforma .NET na tento problém poskytuje funkci `Invoke()`, kterou implementují všechny ovládací prvky včetně formulářů. Parametry se této metodě musí předávat jako pole typu `object`.

4.6.3. Forms

Zbytek modulu tvoří několik formulářů, které zobrazují data a umožňují uživateli provádět určité operace. Jednotlivé formuláře byli z uživatelského hlediska popsány v kapitole Uživatelská dokumentace. V této částí jednotlivé formuláře vypíšu a uvedu jen pár informací co se logiky tý če.

- ClientForm - hlavní formulář, který je rozdělený do 3 částí podle toho v jaké fázi se klient nachází. Při spuštení aplikace je zobrazena první část s uvítaním a nějakými dalšími informacemi. Po přihlášení je zobrazen hlavní kanál ze seznamem místností. Po vytvoření nebo připojení do místnosti je zobrazena třetí část.
- ConnectForm - dialog, který žádá od uživatele IP adresu serveru, port a jeho jméno.

- CreateRoomFOrm - dialog při vytváření místnosti, umožňuje výběr mapy a pojmenování místnosti.
- GameForm - formulář vykreslující hru člověče nezlob se.

Závěr

Cílem této bakalářské práce bylo vytvořit implementaci deskové hry člověče, nezlob se a umožnit hráčům hrát hru po síti. Práce se skládá celkem z pěti částí. Část Server je knihovna poskytující obsluhu klientů nezávisle na typu klienta a protokolu. Tuto knihovnu je možné snadno využít pro další aplikace různého druhu. Knihovna Protocol definuje standart jakým se přenáší data mezi klientem a serverem. Spustitelný soubor ServerUI obsahuje logickou část serveru a slouží k nastartování serveru. Knihovna Game poskytuje logickou vrstvu hry člověče nezlob se a spustitelný soubor ClientUI slouží jako klient připojující se k serveru. Při zpracovávání této práce jsem se dozvěděl mnoho užitečných informaci nejen z literatury, ale i z dokumentace použitých technologií.

Reference

[1] C# 2008 Programujeme profesionálně, Christian Nagel, Bill Evjen, Jay Glynn, Karli Watson, Morgan Skinner, Computer Press, a.s., 2009. Vydání první.

[2] Mistrovství v počítačových sítích, Stephen Bigelow, Computer Press, a.s.

[3] Velký průvodce protokolem TCP/IP a systémem DNS, Libor Dostálek a Alena Kabelová, Computer Press Praha 2000.