Katedra informatiky a výpočetní techniky

Semestrální práce z předmětu úvod do počítačových sítí

Online hra Sim

Jonáš Dufek
A21B0111P
jonasd@students.zcu.cz

17.12.2023

OBSAH

1 ZADÁNÍ

Cílem semestrální práce bylo vytvořit server a klient síťové hry pro více hráčů.

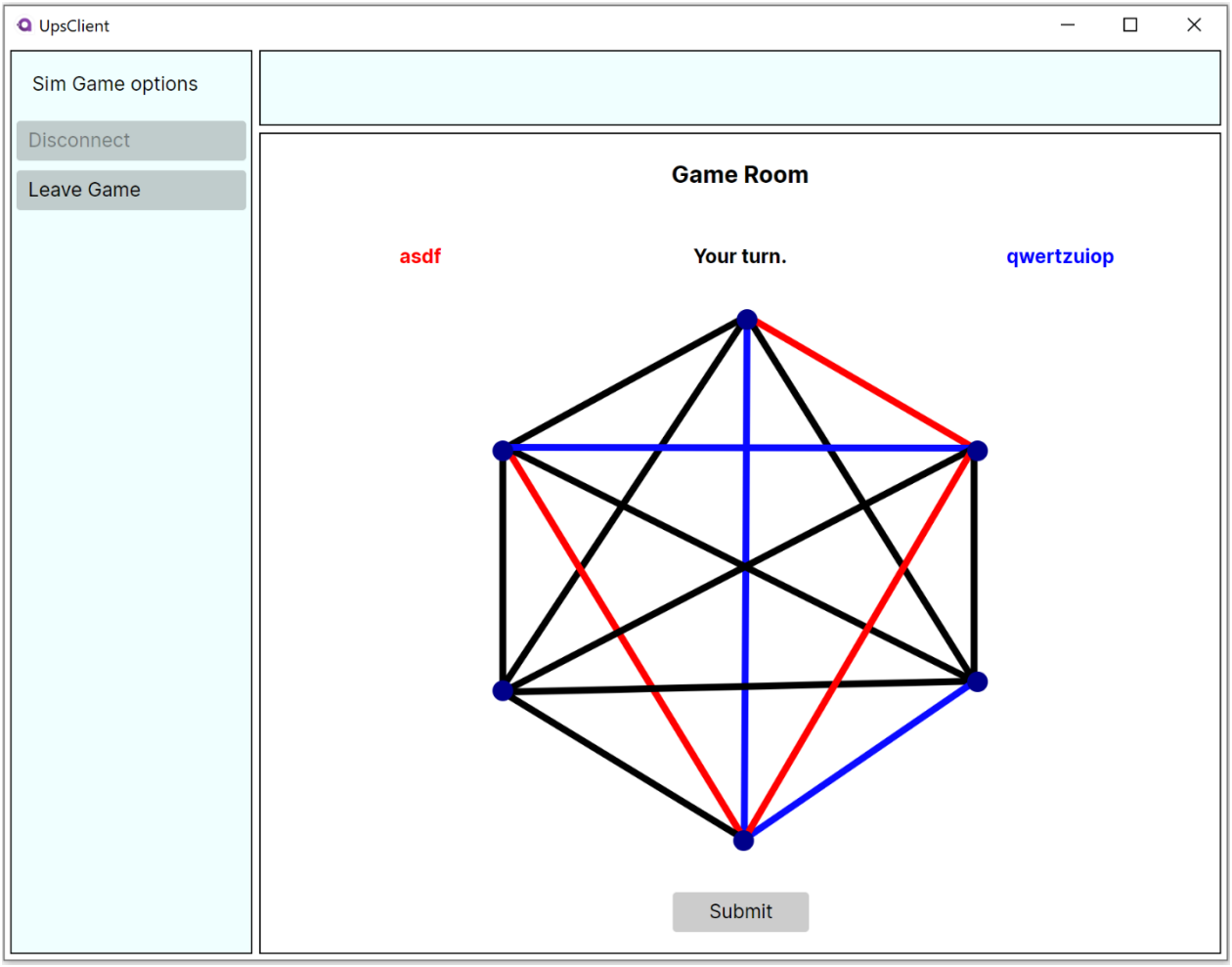
Detailní zadání viz: http://home.zcu.cz/~ublm/files/PozadavkyUPS.pdf

2 POPIS HRY A PRAVIDLA

Sim je hra založená na teorii grafů, jedná se o takzvanou „paper-and-pencil game“, k jejímu hraní tedy není potřeba nic jiného než tužka a papír.

Základem hry je úplný neorientovaný graf o 6 uzlech, hráči se střídají v obarvování hran. Prohrává ten hráč, který jako první ze svých hran vytvoří v grafu trojúhelník (cyklus o třech hranách).

Podrobný popis pravidel viz: https://en.wikipedia.org/wiki/Sim_(pencil_game)



Obrázek 1 - Ukázka herní plochy

3 SYSTÉMOVÉ POŽADAVKY

3.1 Klient

- Programovací jazyk: C# 12.0
- Podporované platformy: Windows, Linux, MacOS, a další podle podpory .NET
- Kompilátor a běhové prostředí: Microsoft .NET 8.0
- Buildovací systém: MSBuild (součást .NET)

3.2 Server

- Programovací jazyk: C++
- Podporované platformy: Linux (verze jádra >2.6.22)
- Kompilátor: GCC (s podporou alespoň C++17)
- Buildovací systém: CMake (alespoň verze 3.20.0)

4 POSTUP PŘEKLADU

4.1 Klient

Po nainstalování .NET stačí přejít do složky `/UpsClient/UpsClient.Desktop` a spustit příkaz:

```
dotnet run
```

Dotnet stáhne externí knihovny, aplikaci zkompiluje, a spustí.

4.2 Server

Je použit soubor `CMakePresets.json`, který definuje nastavení pro debug a release build.

V kořenovém adresáři lze překlad provést pomocí následujících dvou příkazů:

```
cmake --preset release

cmake --build --preset release
```

Spustitelný soubor `main` se bude nacházet ve složce:

`UpsServer/cmake/build-release-linux/`

5 IMPLEMENTACE

5.1 Klient

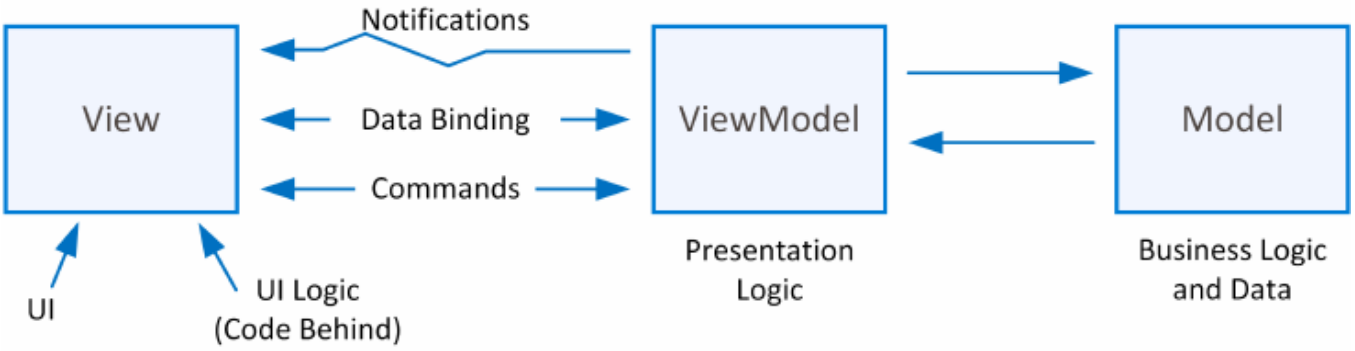
5.1.1 Dekompozice

Klient je dekomponován do 4 hlavních složek dle modelu MVVM:

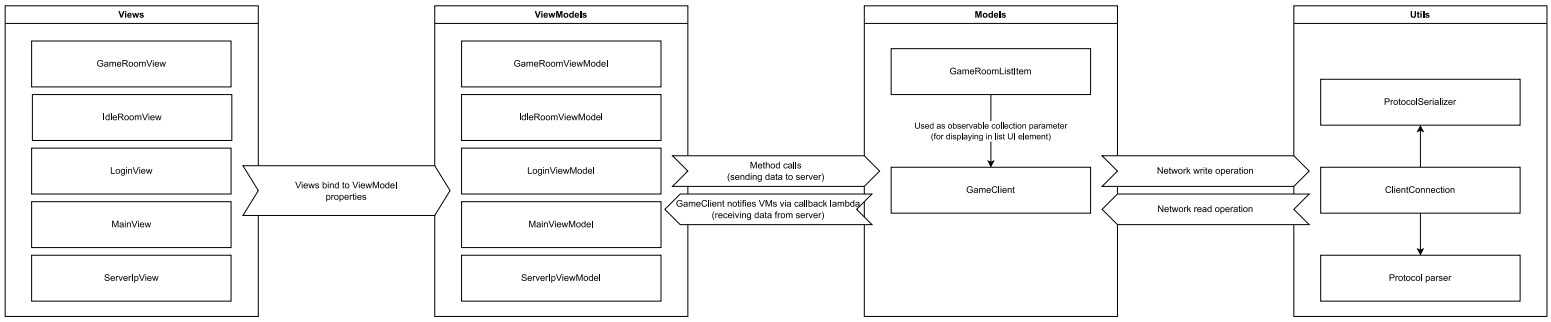
- `Views`
  - XAML soubory popisující vzhled a strukturu GUI, každá „stránka“ zde má svůj view
- `ViewModels`
  - Pro každý View existuje jeden ViewModel
  - ViewModel přebírá data od modelu a zajišťuje aktualizaci dat ve View
  - View používá tzv *„bindings“* pomocí kterých se připojí na atributy ViewModelu
- `Models`
  - V našem případě je to GameClient, který zprostředkovává komunikaci se serverem
- `Utils`
  - Třídy zaměřené na parsování a vedlejší činnosti

5.1.2 Rozvrstvení aplikace

- Využito bylo standardního C# modelu MVVM



Obrázek 2 - Obecný MVVM model (zdroj: https://www.researchgate.net/figure/The-Model-View-ViewModel-MVVM-architectural-pattern-In-MVVM-the-View-layer-is_fig3_275258051)



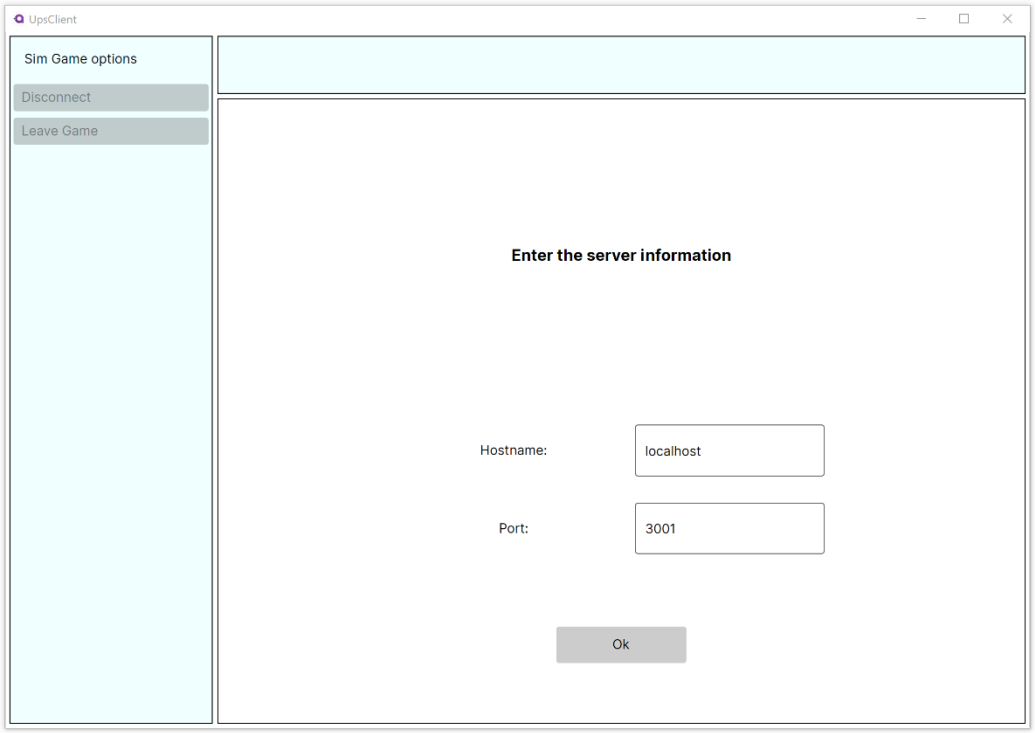
Obrázek 3 - Výsledná podoba modelu klientské aplikace

5.1.3 Použité knihovny

- AvaloniaUI – multiplatformní .NET GUI framework inspirovaný WPF
- Config.Net – settings manager pro .NET

5.1.4 Metody paralelizace

- C# asynchronní metody
- C# Thread & Task
- System.Threading.PeriodicTimer (odesílání pingu v průběhu hry)
- Standardní prostředky pro paralelizaci z knihovny AvaloniaUI



Obrázek 4 - Ukázka okna klientské aplikace

5.2 Server

5.2.1 Dekompozice

Server je dekomponován do následující složek:

- `cmake`
  - obsahuje CMake moduly využité ve skriptech
- `doc`
  - dokumentace
- `extern`
  - externí knihovny stažené pomocí CMake
- `src`
  - `config`
    - správa konfiguračních souborů a loggeru
  - `game_logic`
    - všechny třídy a soubory týkající se konkrétní implementace hry a herního serveru
  - `linux_commons_lib`
    - obalovací třídy pro linuxové prostředky jako je **epoll** a **eventfd**, umožňuje jejich využití v rámci OOP a RAII
  - `network_commons_lib`
    - abstraktní třídy pro BSD sockety, soubory ve složce game_logic z nich dědí
- `test`
  - obsahuje unit testy pro funkce serveru a parseru

5.2.2 Rozvrstvení aplikace

Samotná implementace serveru je rozdělena do tří vrstev (složka UpsServer→src→game_logic).

- `game_room`
  - vrstva implementující grafové algoritmy a logiku herní místnosti
- `game_server`
  - vrstva starající se o navázání a režii TCP připojení
- `protocol`
  - vše týkající se protokolu, tedy parsování a datové struktury

5.2.3 Použité knihovny

- Server používá knihovnu spdlog, která je automaticky stažena pomocí CMake. Umožňuje efektivní logování ve více vláknovém prostředí a záznam do souboru.

5.2.4 Metody paralelizace

5.2.4.1 std::thread

Standardní knihovna pro práci s vlákny v C++, RAII obal nad knihovnou vláken příslušného OS.

5.2.4.2 Eventfd

Linuxový synchronizační prostředek poskytující wait / notify mechanismus. Eventfd notify událost lze zachytit pomocí epoll. Pro účely synchronizace je efektivnější a rychlejší než použití pipe.

Server Eventfd využívá k implementaci thread-safe fronty EventfdQueue.

Další informace:

https://man7.org/linux/man-pages/man2/eventfd.2.html

5.2.4.3 Epoll

Linuxový prostředek pro práci s událostmi I/O zařízení.

Slouží jako náhrada starších select() a poll(). Epoll je zároveň také mnohem efektivnější než starší alternativy (výpočetní složitost $O(1)$ vs $O(n)$).

Number of descriptors monitored (*N*)	***poll()* CPU time (seconds)**	***select()* CPU time (seconds)**	***epoll* CPU time (seconds)**
10	0.61	0.73	0.41
100	2.9	3.0	0.42
1000	35	35	0.53
10000	990	930	0.66

Obrázek 5 - porovnání asynchronních prostředků (zdroj: https://suchprogramming.com/epoll-in-3-easy-steps/)

Další informace:

https://suchprogramming.com/epoll-in-3-easy-steps/

https://en.wikipedia.org/wiki/Epoll

https://www.hackingnote.com/en/versus/select-vs-poll-vs-epoll/

https://sciencesoftcode.files.wordpress.com/2018/12/the-linux-programming-interface-michael-kerrisk-1.pdf

6 POPIS PROTOKOLU

6.1 Formát zprávy

Formát zpráv je inspirovaný HTTP. Skládá se ze tří hlavních částí: názvu metody, atributů a značky ukončující zprávu. K oddělení jednotlivých atributů slouží znak \n.

Formát protokolu lze tedy popsat následovně:

```
<METHOD_NAME>\n
<attribute_1_name>:<string> | <string_list>\n
<attribute_2_name>:<string> | <string_list>\n
…
<attribute_n_name>:<string> | <string_list>\n
\r\n\r\n
```

6.2 Přenášené struktury, datové typy

Jak již bylo znázorněno v předchozí podkapitole, obsah atributu může být buď `<string>` nebo `<string_list>`.

String list je ve formátu:

$$\{item\_1, item\_2, ..., item\_n\}_1, \ldots, \{item\_1, item\_2, ..., item\_n\}_n$$

Klient i server téměř výhradně používají jako hodnotu atributu prostý string. Pouze u příkazů `GAME_COMMAND` a `GET_ROOM_LIST` je využit list.

6.3 Význam přenášených dat, kódů

Aplikace nepoužívá pro identifikaci typu zpráv kódy, ale řetězcové konstanty ve formátu `SNAKE_CASE`.

Hodnota `<METHOD_NAME>` identifikující typ zprávy může nabývat následujících hodnot:

Název	Popis	Atributy	Zprávu generuje
Základní metody			
CONNECTED_OK	Server odešle klientovi po připojení, nebo po návratu do idle room	žádné	Server
REQ_ACCEPTED	Výchozí odpověď na požadavek klienta (úspěch)	záleží na situaci	Server
REQ_DENIED	Výchozí odpověď na požadavek klienta (neúspěch)	žádné	Server
TERMINATE_CONNECTION	Klient signalizuje ukončení spojení	žádné	Klient
Herní místnost			
ENTER_USERNAME	Klient serveru odesílá své uživatelské jméno	username	Klient
GET_ROOM_LIST	Pouze přístupné, pokud uživatel má jméno. Server odpoví REQ_ACCEPT s atributem room_list[1].	žádné	Klient
JOIN_GAME	Klient se připojí do herní místnosti	game_id	Klient
Metody hry			
GAME_IDLE	Server odešle klientovi po připojení do prázdné místnosti	žádné	Server
GAME_LEAVE	Klient opustí herní místnost	žádné	Klient

[1] Room_list je ve formátu {<USERNAME_1>,<USERNAME_2>,<ROOM_ID>,<GAME_STATE>}, ROOM_ID je unsigned int, GAME_STATE je buď 0 = idle, 1 = running, 2 = paused.

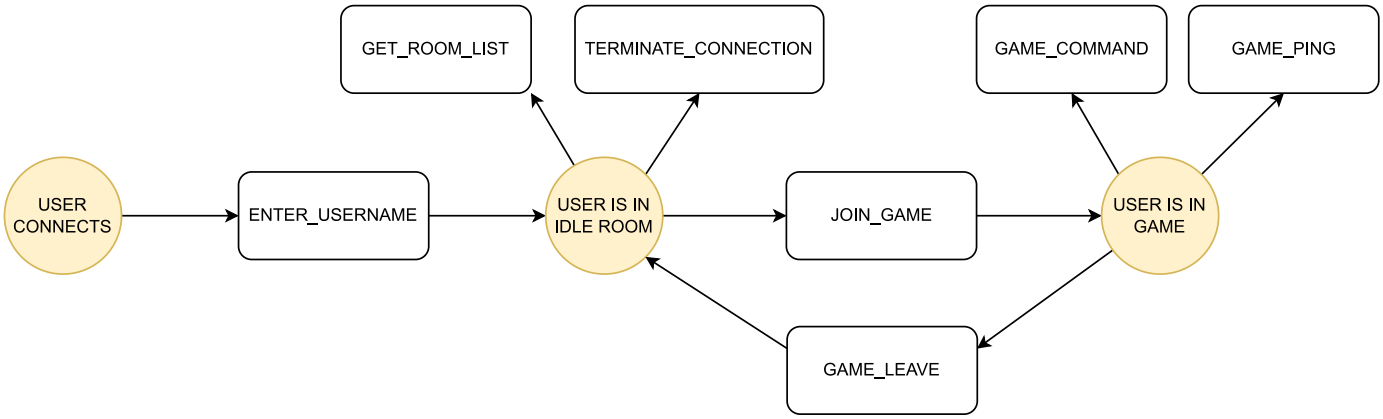
GAME_COMMAND	Klient odesílá herní příkaz	add_edge[2]	Klient
GAME_STATE	Server odesílá stav hry	on_turn, player1_edges[3], player2_username, player2_edges, player1_username, game_state	Server
GAME_PING	Klient musí odesílat, když je ve hře každých 5 sekund, jinak je odpojen. Server odpoví REQ_ACCEPTED	žádné	Klient

6.4 Omezení vstupních dat a validace hodnot

Maximální povolená délka zprávy byla nastavena zhruba na 2 MB, lze ale upravit.

Server interně validuje data a v neplatném stavu odpoví REQ_DENIED, viz kapitola Chybové stavy.

6.5 Návaznost zpráv



Obrázek 6 - Stavový diagram aplikace z pohledu klienta

[2] Ve formátu řetězce {edge_source, edge_destination}
[3] Ve formátu seznamu hran {{edge_source1, edge_destination1}, …}

6.6 Chybové stavy

V případě nesprávných atributů je klientovi poslán REQ_DENIED. Po pěti REQ_DENIED je klient odpojen.

Při porušení formátu protokolu je klient odpojen okamžitě.

7 ZÁVĚR A ZHODNOCENÍ

Všechny body zadání byly splněny, aplikace jsou funkční a správně realizují síťovou komunikaci.

Za kladnou vlastnost serveru bych považoval využití pokročilejších linuxových prostředků – epoll a eventfd.

Největší benefit klientské aplikace bych viděl v její přenositelnosti, umožňuje kompilaci pro Windows, Linux, MacOS, embedded, WebAssembly a další platformy podle podpory .NET.

Za zápornou vlastnost by se dal považovat místy na straně serveru někdy až příliš fragmentovaný kód a méně přehledná implementace parseru (zřejmě by bylo lepší využít C++ regex knihovnu).