



**FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI**

**Semestrální práce z předmětu UOS
Úvod do počítačových sítí**

Multiplayerová hra Mariáš

Student:

David Wimmer

davidw@gapps.zcu.cz

Datum: 13. ledna 2026

Obsah

1	Úvod	3
2	Zadání	4
3	Analýza úlohy	5
3.1	Popis přenášených zpráv	5
3.2	Formát zpráv	5
3.2.1	Detekce špatné zprávy	6
3.3	Hra Mariáš	6
3.3.1	Vizualizace stavového diagramu hry	7
3.4	Komunikace mezi klientem a serverem	7
4	Zpětné připojení do hry (Reconnect)	8
4.1	Mechanismus na straně klienta	8
4.1.1	Detekce výpadku spojení	8
4.1.2	Heartbeat Mechanismus (PING/PONG)	9
4.1.3	Automatické znovupřipojení	9
4.2	Mechanismus na straně serveru	9
4.2.1	Detekce odpojení klienta	9
4.2.2	Správa odpojených klientů	10
4.2.3	Proces reconnectu na serveru	10
4.3	Algoritmus zpětného posílání paketů	11
4.3.1	Ukládání paketů na serveru	11
4.3.2	Proces zpětného odesílání	11
5	Popis implementace serverové části	12
5.1	Modulární struktura serveru	12
5.2	Implementace herní logiky	13
5.3	Síťová komunikace a robustnost	13
5.3.1	Vícevláknové zpracování	13
6	Popis implementace uživatelského rozhraní	15
6.1	Architektura aplikace	15
6.1.1	Struktura souborů klientského rozhraní	16
6.1.2	Diagram komponent	16
6.2	Hlavní komponenty	16
6.2.1	Gui – Hlavní třída aplikace	16
6.2.2	ClientManager – Síťová komunikace	17
6.2.3	GameManager – Správa hry	17
6.2.4	Game – Herní logika	17
6.2.5	GuiManager – Vykreslování UI	18
6.3	Zpracování zpráv	18
6.4	Optimalizace a výkon	18

6.4.1	Vykreslování	18
6.5	Validace vstupů	19
7	Spuštění a běh programu	20
7.1	Kompilace serveru	20
7.2	Spuštění serveru	20
7.3	Spuštění klienta	20
8	Závěr	21

1 Úvod

Tato semestrální práce se zabývá návrhem a implementací serverové a klientské části síťové aplikace v rámci předmětu UPS – Úvod do počítačových sítí. Hlavním cílem práce je vytvořit multiplayerovou karetní hru pro 2 až N hráčů, která bude fungovat na principu klient–server architektury.

Serverová část aplikace byla implementována v nižším programovacím jazyce C++ a je určena ke spuštění v prostředí operačního systému Linux prostřednictvím příkazové řádky. Klientská část byla naopak realizována ve vyšším programovacím jazyce Python, přičemž pro tvorbu grafického uživatelského rozhraní byla využita externí knihovna Pygame. Pro komunikaci mezi klientem a serverem byl zvolen protokol TCP, který zajišťuje spolehlivý přenos dat.

Prvním krokem bylo navržení a implementování samotné herní logiky, konkrétně karetní hry Mariáš. Následně byl navržen komunikační protokol umožňující výměnu zpráv mezi klienty a serverem. Pro účely testování a pohodlného ovládání hry bylo nezbytné vytvořit grafické uživatelské rozhraní na straně klienta.

Při implementaci serveru bylo nutné řešit několik klíčových problémů, mezi které patří správa herních relací (session), opětovné připojení klientů (reconnect), paralelní obsluha více připojených hráčů a možnost opakovaného spouštění her bez nutnosti restartu serveru.

Výsledkem této práce je plně funkční síťová aplikace umožňující hraní hry Mariáš ve více hráčích prostřednictvím klient–server architektury.

2 Zadání

Zadání semestrální práce je rozsáhlé. Celé zadání týkající se této semestrální práce najdete ve složce *docs* s názvem **PozadavkyUPS.pdf**.

3 Analýza úlohy

3.1 Popis přenášených zpráv

V rámci semestrální práce nebylo dovoleno používat knihovny usnadňující přenos zpráv mezi klientem a serverem. Z tohoto důvodu bylo nutné navrhnout vlastní formát přenášených zpráv. Inspirací byly podobné projekty, ve kterých se často využívá koncept typů zpráv (`MessageTypes`).

Každá zpráva přenášená mezi klientem a serverem začíná informací o své velikosti, což umožňuje správné načtení celého obsahu zprávy na přijímající straně. Následuje identifikátor typu zprávy a číslo paketu, které slouží k jednoznačné identifikaci přenášených dat.

Z důvodu možnosti opětovného odeslání ztracených paketů v případě výpadku klienta bylo zavedeno číslování paketů a identifikátor klienta (`clientID`). Tento mechanismus je využíván v algoritmu zpětného odesílání paketů, který je podrobně popsán v následující kapitole.

Vlastní datová část zprávy je oddělena pomocí speciálních znaků `|`, `-` a `:`. Tyto znaky se v přenášených datech nevyskytují, a proto není nutné řešit jejich escapování.

3.2 Formát zpráv

Zpráva byla složena z hlavičky a datové části. Hlavička se skládá z 5 částí oddělených znakem `|`:

```
// Format zpravy:  
// SIZE|PACKET_ID|CLIENT_ID|TYPE|DATA1|DATA2|... \ n  
typedef struct {  
    uint16_t size;           // Celkova velikost zpravy  
    uint8_t packetID;        // ID packetu (0-254)  
    uint8_t clientID;        // ID klienta  
    MessageType type;        // Typ zpravy (1-20)  
    vector<string> fields;    // Datova cast  
} Message;
```

Příklad zprávy:

25|42|0|14|MujNickname\n

Kde:

- 25 – Celková velikost zprávy v bajtech
- 42 – ID paketu (sekvenční číslo)
- 0 – ID klienta
- 14 – Typ zprávy (CONNECT)
- `MujNickname` – Datová část (přezdívka)
- `\n` – Terminátor zprávy

3.2.1 Detekce špatné zprávy

Při příjmu zprávy probíhá validace na několika úrovních:

1. Validace řetězce (před deserializací):

- Kontrola prázdné zprávy
- Kontrola maximální délky (65535 bajtů)
- Kontrola terminátoru `\n`
- Minimální počet delimiterů (alespoň 2)
- Kontrola neplatných znaků (null bytes, kontrolní znaky)
- Detekce podezřelých vzorů (více než 100 stejných znaků za sebou)

2. Validace zprávy (po deserializaci):

- **CLIENT_ID** – Musí odpovídat očekávanému číslu klienta a být v rozsahu 0 až (`requiredPlayers` - 1). Výjimka: při RECONNECT zprávě se `CLIENT_ID` nekontroluje.
- **MESSAGE_TYPE** – Musí být v rozsahu 1-20.
- **PACKET_ID** – Kontrola sekvence s tolerancí pro wraparound (po 255 se vrací na 0). Rozdíl větší než deset je zaznamenán pouze výpysem do konzole.
- **FIELDS** – Každé pole nesmí být delší než 1000 znaků a nesmí obsahovat delimiter, terminátor ani null bytes
- **SIZE** – Celková velikost nesmí překročit `MAX_MESSAGE_SIZE`

Pokud validace selže, klient je odpojen a obdrží zprávu typu DISCONNECT s důvodem.

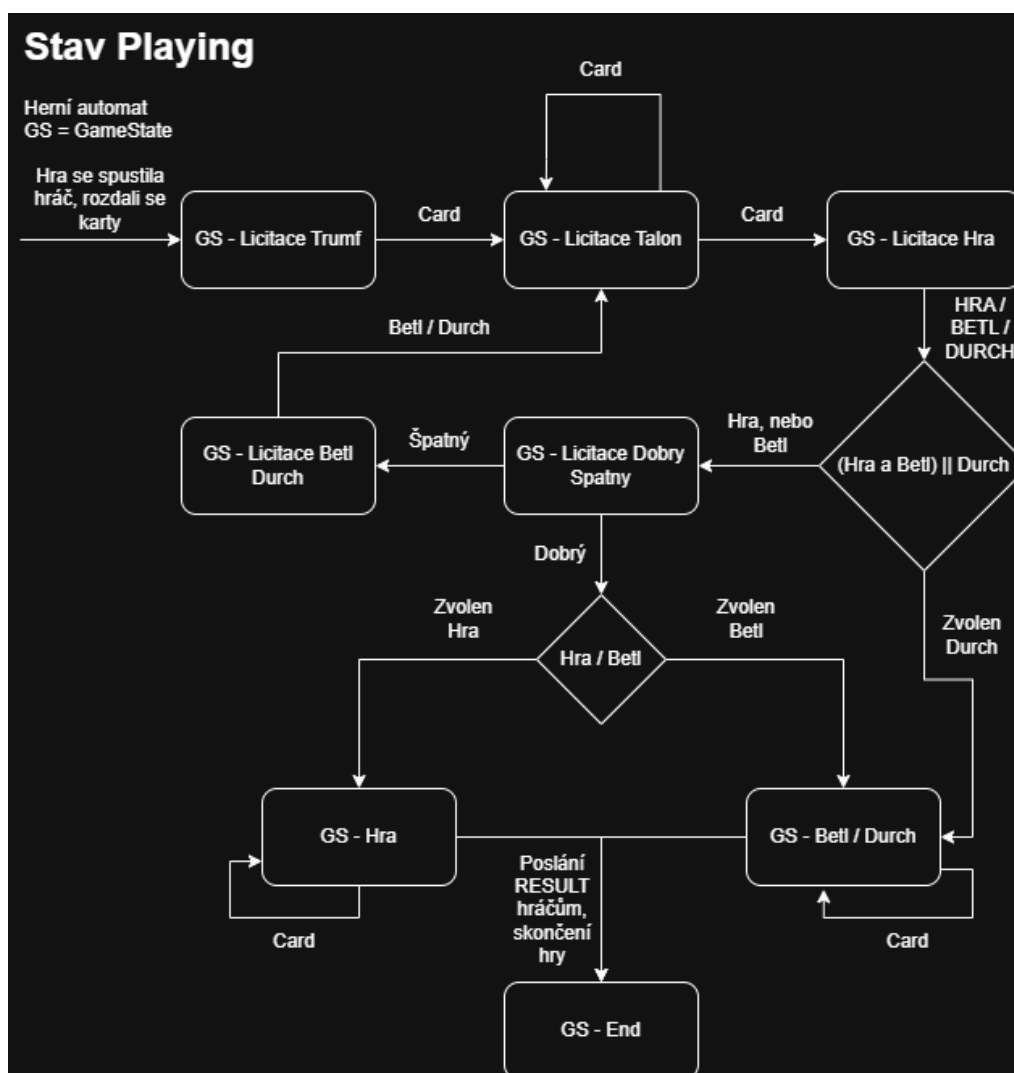
3.3 Hra Mariáš

Oproti ostatním zadáním je hra Mariáš komplexní hra s mnoha stavy. Byla proto navržena jednodušší varianta oproti celé verzi hry Mariáš. V GUI je v levém horním rohu popsáno jaké prvky nebyly zakomponovány do dané hry. Zde je podrobnější vysvětlení:

- Neexistence hlasu - Neexistuje spojení krále a svrška
- Neexistence licitování - Hra není hraná o peníze, neexistují hry Sedma, Kilo a Stosedm
- Pokud je zahlášen Betl, může být zahlášen i následujícím hráčem

Dostupné hry jsou Hra, Betl a Durch.

3.3.1 Vizualizace stavového diagramu hry

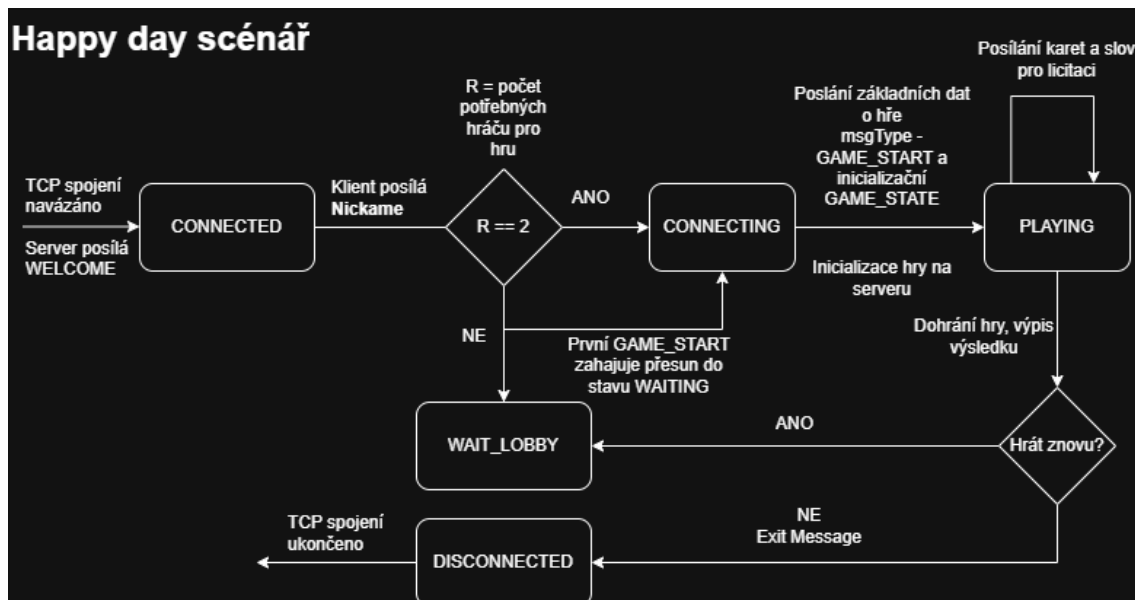


Obrázek 1: Vizualizace stavového diagramu hry

3.4 Komunikace mezi klientem a serverem

Komunikace mezi klientem a serverem byl velmi důležitý úkol na správné fungování serveru. Jak bylo zmíněno, pro komunikace se používali *Typy zpráv*, které určovali

účel dané zprávy. Celkový koncept komunikace má několik různých scénářů. Pro přehlednost byl zpracován tzv. *Happy day scenario*.



Obrázek 2: Vizualizace komunikace při šťastném scénáři

4 Zpětné připojení do hry (Reconnect)

Cílem zadání bylo navrhnout a implementovat mechanismus umožňující zvládnutí jak krátkodobého, tak i dlouhodobého výpadku spojení klienta se serverem. Implementace využívá automatické znovupřipojení na straně klienta a správu odpojených klientů na straně serveru.

4.1 Mechanismus na straně klienta

4.1.1 Detekce výpadku spojení

Klient detekuje výpadek spojení v následujících případech:

- Socket vrátí 0 bajtů při čtení (server zavřel spojení)
- Timeout při čtení ze socketu (socket.timeout)
- Chyba při odesílání dat
- Více než 3 neplatné zprávy od serveru za sebou
- **Heartbeat timeout**: Server neodpověděl zprávou PONG na PING do 12 sekund.

4.1.2 Heartbeat Mechanismus (PING/PONG)

Pro aktivní kontrolu spojení klientská aplikace implementuje aplikační heartbeat:

- Klient odesílá zprávu typu PING (ID 19) každé 3 sekundy.
- Očekává odpověď typu PONG (ID 20) od serveru.
- Pokud od poslední odpovědi PONG uběhne více než 12 sekund, klient považuje spojení za ztracené (dead peer) a zahájí reconnect.

4.1.3 Automatické znovupřipojení

Po detekci výpadku je automaticky spuštěn mechanismus znovupřipojení:

Parametry reconnectu:

- `max_reconnect_attempts` = 6 – Maximální počet pokusů
- `reconnect_delay` = 5 sekund – Prodleva mezi pokusy
- `connection_timeout` = 15 sekund – Timeout pro navázání a potvrzení spojení

Proces znovupřipojení:

1. Zavření starého socketu a vlákna
2. Vytvoření nového TCP socketu
3. Připojení k serveru pomocí `socket.connect()`
4. Spuštění listening threadu
5. Odeslání zprávy typu RECONNECT s parametry:
 - `fields[0]` – Nickname (session ID)
 - `fields[1]` – ID posledního přijatého paketu
6. Čekání na odpověď serveru (zpráva RECONNECT)
7. Pokud server odpoví kladně, reconnect je úspěšný
8. Pokud server neodpoví v limitu 15s nebo odpoví DISCONNECT, pokus selhal

Pokud všechny pokusy selžou, klient je přesměrován do hlavního menu, kde má možnost manuálního reconnectu.

4.2 Mechanismus na straně serveru

4.2.1 Detekce odpojení klienta

Server detekuje odpojení klienta v následujících případech:

- Prázdná zpráva při čtení ze socketu (TCP FIN)

- Neplatná zpráva (selhala validace)
- Timeout při autorizaci (10 sekund pro nové klienty) - netýká se reconnectujících
- Chyba na úrovni TCP KeepAlive (nastaveno na 5s idle, 5s interval)

4.2.2 Správa odpojených klientů

Pokud klient ztratí spojení během hry, server:

1. Označí klienta jako `isDisconnected = true`
2. Zavře starý socket
3. Ponechá klientovu strukturu v paměti (včetně herního stavu)
4. Spustí timeout 60 sekund (`RECONNECT_TIMEOUT_SECONDS`)
5. Informuje ostatní hráče zprávou `STATUS` s kódem 2 (odpojení)

4.2.3 Proces reconnectu na serveru

Když server obdrží zprávu typu `RECONNECT`:

1. Vyhledá odpojeného klienta podle nickname (`findDisconnectedClient`)
2. Kontrola timeoutu (musí být méně než 60 sekund od odpojení)
3. Pokud klient existuje a timeout neuplynul:
 - Odstraní dočasného klienta (nový socket) z paměti, aby neuvolňoval místo
 - Přiřadí nový socket k původní struktuře klienta
 - Nastaví `connected = true, isDisconnected = false`
 - Aktualizuje `lastSeen` timestamp
4. **Přeskočení autorizace:** Jelikož je klient již známý, server nezasílá `AUTHORIZE` ani `WAIT_LOBBY`, ale rovnou obnovuje stav.
5. Odešle klientovi ztracené pakety (viz algoritmus níže)
6. Potvrdí reconnect zprávou typu `RECONNECT`
7. Informuje ostatní hráče zprávou `STATUS` s kódem 3 (reconnect), čímž se jim v GUI zruší čekací dialog.

Pokud reconnect selže (klient nenalezen nebo timeout vypršel), server odpoví zprávou `DISCONNECT`.

4.3 Algoritmus zpětného posílání paketů

Při reconnectu server musí klientovi dodat všechny pakety, které zmeškal během výpadku. Tento mechanismus funguje následovně:

4.3.1 Ukládání paketů na serveru

`NetworkManager` udržuje kruhový buffer paketů:

- Velikost bufferu: 255 pozic (`MAXIMUM_PACKET_SIZE`)
- Každý odeslaný paket je uložen na pozici odpovídající jeho `packetID`
- `PacketID` se inkrementuje po každém odeslání: $(\text{packetID} + 1) \% 255$

4.3.2 Proces zpětného odesílání

Když klient pošle `RECONNECT` zprávu s `lastReceivedPacketID`:

1. Nalezení nejnovějšího paketu:

- Server prochází buffer odzadu
- Hledá poslední paket s `clientID` odpovídajícím klientovi

2. Kontrola aktuality:

- Pokud `lastReceivedPacketID >= latestPacketID`, klient je aktuální a nic se neposílá.

3. Sběr chybějících paketů:

- Program sebere všechny pakety v intervalu $(\text{lastReceivedPacketID} + 1)$ až `latestPacketID`.
- Algoritmus správně ošetřuje tzv. wraparound (přetečení počítadla), kdy se ID vrací z 255 na 0. V takovém případě se nejprve vyberou pakety do konce bufferu a následně od začátku.

4. Opětné odeslání:

- Pakety se posílají ve správném pořadí (chronologicky od nejstaršího).
- Mezi odesláním jednotlivých paketů je vložena umělá pauza 10 ms, aby nedošlo k zahlcení socketu nebo klienta nárazovým tokem dat.

5 Popis implementace serverové části

Serverová část aplikace je navržena jako robustní, vícevláknový systém schopný obsluhovat více herních místností (lobby) současně. Architektura klade důraz na oddělení síťové komunikace, herní logiky a správy klientů. Implementace využívá standard C++17 a knihovnu POSIX vláken pro efektivní paralelní zpracování.

5.1 Modulární struktura serveru

Server byl rozdělen na následující moduly, z nichž každý má jasně definovanou odpovědnost:

- **Main** - Vstupní bod aplikace. Zajišťuje parsování parametrů příkazové řádky pomocí přepínačů (např. `-i` pro IP adresu, `-p` pro port, `-l` pro počet lobby a `-n` pro počet hráčů). Dále inicializuje hlavní instanci **GameServer** a nastavuje handlers pro systémové signály (`SIGINT`, `SIGTERM`) pro korektní ukončení serveru.
- **Server (GameServer)** - Centrální orchestrátor celého systému. Spravuje životní cyklus ostatních komponent (**NetworkManager**, **LobbyManager**, **MessageHandler**). Obsahuje hlavní smyčku pro přijímání nových spojení (`acceptClients`) běžící ve vlastním vlákně a deleguje nově připojené klienty do dostupných herních místností.
- **LobbyManager** - Implementuje logiku pro správu více herních místností. Umožňuje dynamické přidělování hráčů do volných lobby a udržuje přehled o stavu jednotlivých her (čekání na hráče, probíhající hra). Každé lobby je reprezentováno strukturou **Lobby**, která obsahuje vlastní instance **ClientManager** a **GameManager**.
- **NetworkManager** - Zapouzdřuje nízkoúrovňové operace se sokety (Berkeley sockets). Zajišťuje inicializaci serverového socketu, vazbu na IP adresu a port (`bind`), a naslouchání (`listen`). Implementuje metody pro bezpečné odesílání a přijímání zpráv podle definovaného protokolu. Klíčovou funkcí je ukládání odeslaných paketů do historie pro možnost jejich opětovného zaslání v případě detekce ztráty na straně klienta.
- **Protokol** - Definuje strukturu vlastního textového protokolu použitého pro komunikaci. Celý popis protokolu je zmíněn v **sekci 3.2**.
- **ClientManager** - Řídí logiku a stav připojených klientů v rámci jednoho lobby. Sleduje aktivitu hráčů (heartbeat), spravuje jejich přezdívky a přidělená čísla hráčů. Implementuje mechanismus pro opětovné připojení (`reconnect`), který umožňuje hráči vrátit se do rozehrané hry po krátkodobém výpadku spojení (standardně 60 sekund).
- **GameManager** - Slouží jako most mezi síťovou vrstvou a samotnou logikou hry Mariáš. Zajišťuje synchronizaci mezi vlákny klientů a herním stavem

pomocí mutexů a podmínkových proměnných (`std::condition_variable`). Serializuje stav hry pro odeslání klientům a validuje jejich tahy.

- **MessageHandler** - Dekóduje přijaté zprávy od **NetworkManager** a na základě jejich typu volá příslušné metody v **GameManager** nebo **ClientManager**. Tímto oddělením je zajištěno, že logika zpracování zpráv je nezávislá na způsobu jejich přenosu.

5.2 Implementace herní logiky

Aby bylo zamezeno podvádění ze strany klientů, veškerá autoritativní logika hry běží výhradně na serveru. Klienti dostávají pouze informace, které jsou pro ně v daný moment viditelné (např. své karty a karty již zahrané na stole).

Modul *game* se skládá z následujících tříd:

- **Card** - Reprezentuje hrací kartu. Uchovává informace o barvě (červené, listy, žaludy, kule) a hodnotě (7, 8, 9, 10, spodek, svršek, král, eso). Obsahuje logiku pro porovnávání karet s ohledem na trumfovou barvu.
- **Deck** - Implementuje balíček 32 mariášových karet. Zajišťuje náhodné míchání (`std::shuffle`) a korektní rozdávání karet hráčům a do talonu podle pravidel.
- **Hand** - Kontejner pro karty, které má hráč aktuálně v ruce. Poskytuje metody pro přidávání, odebrání a validaci hratelnosti karty podle pravidel Mariáše (přiznávání barvy, přebíjení).
- **Player** - Reprezentuje hráče z pohledu herní logiky. Udržuje informace o jeho skóre, získaných štyších a stavu (zda je licitátor, zda nahlásil vyšší hru).
- **GameLogic** - Obsahuje pomocné algoritmy pro vyhodnocování pravidel, která nejsou přímo vázána na stavový automat. Patří sem určení vítěze štychu, výpočet bodové hodnoty získaných karet a nastavení trumfové barvy.
- **Game** - Hlavní třída implementující stavový automat hry. Řídí přechody mezi jednotlivými stavy hry mariáše.

5.3 Síťová komunikace a robustnost

Vzhledem k povaze síťového prostředí (možnost ztráty paketů, latence) server implementuje několik mechanismů pro zajištění plynulého herního zážitku.

5.3.1 Vícevláknové zpracování

Server je navržen jako vícevláknová aplikace umožňující paralelní zpracování jednotlivých částí systému. Samostatná vlákna jsou využívána pro obsluhu herní logiky, správu jednotlivých herních místností a pro periodickou detekci odpojených klientů. Přijímání nových klientských spojení probíhá v jednom dedikovaném vlákně,

přičemž po navázání spojení je každému klientovi přiřazeno vlastní vlákno zajišťující síťovou komunikaci se serverem.

Vzhledem k paralelnímu přístupu více vláken ke sdíleným datovým strukturám jsou k zajištění konzistence dat použity synchronizační mechanismy, zejména mutexy. Tyto mechanismy zabráňují vzniku datových závodů a nekonzistentních stavů při souběžném čtení a zápisu dat.

Vícevláknová architektura umožňuje serveru provádět více činností současně, čímž dochází ke zvýšení výkonu a lepší škálovatelnosti systému. Zároveň tento přístup významně zjednodušuje implementaci herní logiky ve srovnání s řešením založeným na čistě sekvenčním zpracování, které by bylo obtížně rozšiřitelné a hůře udržitelné.

6 Popis implementace uživatelského rozhraní

Klientská aplikace slouží jako uživatelské rozhraní pro hru Mariáš, která umožňuje:

- Připojení k hernímu serveru přes TCP/IP
- Grafické zobrazení herního stavu
- Interakci s hráčem prostřednictvím GUI
- Automatické znovupřipojení při výpadku spojení
- Zobrazení pravidel hry

6.1 Architektura aplikace

Aplikace je navržena podle principů objektově orientovaného programování a je rozdělena do několika logických celků.

6.1.1 Struktura souborů klientského rozhraní

```
client/
├── main.py                # Vstupní bod aplikace
├── images/                # Grafické assety (karty, pozadí)
└── src/
    ├── Gui.py             # Hlavní třída GUI a stavový automat
    ├── GameManager.py     # Správa herní logiky a vykreslování
    ├── Client/
    │   ├── ClientManager.py # Síťová komunikace
    │   └── Protocol.py     # Definice protokolu
    ├── Game/
    │   ├── Game.py        # Herní stav a pravidla
    │   ├── Card.py        # Reprezentace karet
    │   └── Player.py       # Reprezentace hráčů
    └── View/
        ├── GuiManager.py  # Vykreslování UI komponent
        ├── Obstacles.py   # UI prvky (tlačítka, inputy)
        └── Validator.py   # Validace uživatelských vstupů
```

6.1.2 Diagram komponent

Aplikace se skládá z následujících hlavních komponent:

- **Gui** – Řídící třída, která spravuje stavový automat aplikace
- **ClientManager** – Zajišťuje síťovou komunikaci se serverem
- **GameManager** – Spravuje herní logiku a vykreslování hry
- **GuiManager** – Vykresluje UI komponenty (lobby, čekání, hra)
- **Game** – Obsahuje herní stav a pravidla Mariáše

6.2 Hlavní komponenty

6.2.1 Gui – Hlavní třída aplikace

Třída **Gui** je centrálním bodem aplikace a implementuje stavový automat s následujícími stavy:

LOBBY – Úvodní obrazovka pro zadání IP, portu a přezdívky

CONNECTING – Probíhá připojování ke hře (Nastaveno 5 sekund)

WAITING – Čekání na ostatní hráče

PLAYING – Probíhá hra

RECONNECTING – Probíhá znovupřipojení po výpadku

DISCONNECTION – Odpojení od serveru

HELP – Zobrazení nápovědy a pravidel

6.2.2 ClientManager – Síťová komunikace

Třída **ClientManager** zajišťuje veškerou síťovou komunikaci s herním serverem.

Funkce

- Připojení k serveru přes TCP socket
- Asynchronní příjem zpráv v samostatném vlákně
- Automatické znovupřipojení při výpadku spojení
- Heartbeat mechanismus pro detekci výpadků
- Fronta zpráv pro thread-safe zpracování

6.2.3 GameManager – Správa hry

Třída **GameManager** spojuje herní logiku s grafickým rozhraním.

Odpovědnosti

- Vykreslování karet hráče
- Zobrazení zahráných karet na stole
- Vykreslování licitačních tlačítek
- Zpracování kliknutí na karty
- Parsování herních dat ze serveru
- Zobrazení výsledků hry

6.2.4 Game – Herní logika

Třída **Game** reprezentuje stav hry Mariáš.

Herní stavy

LICITACE_TRUMF – Licitace trumfové barvy

LICITACE_TALON – Licitace talonu

LICITACE_HRA – Licitace typu hry

LICITACE_DOBRY_SPATNY – Licitace dobrý/špatný

LICITACE_BETL_DURCH – Licitace betl/durch

HRA – Probíhá hra

BETL – Probíhá BETL

DURCH – Probíhá DURCH

END – Konec hry

6.2.5 GuiManager – Vykreslování UI

Třída `GuiManager` zajišťuje vykreslování všech UI komponent.

Obrazovky

- `draw_lobby()` – Lobby s input fieldy
- `draw_connecting()` – Obrazovka připojování
- `draw_waiting()` – Čekání na hráče s progress barem
- `draw_reconnecting()` – Reconnecting s animací
- `draw_help()` – Náповěda s pravidly hry

UI komponenty `GuiManager` používá následující komponenty z modulu `Obstacles`:

- `InputBox` – Textové vstupní pole
- `Button` – Klikací tlačítko
- `HelpButton` – Tlačítko nápovědy

6.3 Zpracování zpráv

Všechny zprávy od serveru procházejí následujícím tokem:

1. `ClientManager` přijme data v `_listen_loop()`
2. Zpráva je vložena do thread-safe fronty
3. `_process_message_queue()` zpracuje zprávu v hlavním vlákne
4. Volání příslušného callbacku v `Gui`
5. Aktualizace stavu a označení pro překreslení
6. Vykreslení změn v hlavní smyčce

6.4 Optimalizace a výkon

6.4.1 Vykreslování

Aplikace používá optimalizované vykreslování:

- Cachování pozadí (vytvoří se pouze jednou)
- Dirty flag systém – překreslení pouze při změně

- Omezení FPS na 60 pro snížení zátěže CPU
- Vykreslování pouze viditelných prvků

6.5 Validace vstupů

Třída `InputValidator` zajišťuje validaci uživatelských vstupů:

- **IP adresa** – Kontrola formátu IPv4 (regex)
- **Port** – Číslo v rozsahu 1-65535
- **Přezdívka** – Délka 1-20 znaků, alfanumerické znaky

7 Spuštění a běh programu

7.1 Kompilace serveru

Serverovou část aplikace je možné spustit na platformě **Linux** nebo ve virtuálním prostředí systému Windows, například pomocí **WSL** (Ubuntu). Pro úspěšnou kompilaci je nutné mít nainstalovaný kompilátor **gcc** nebo **clang** a nástroj **make**. Projekt je psán v jazyce **C++** a vyžaduje podporu standardu **C++17**.

Kompilace se provádí v adresáři UPS spuštěním následujícího příkazu:

```
$ make
```

Po úspěšném překladu je vytvořen adresář *build*, který obsahuje zkompilevané soubory. Tento adresář lze odstranit pomocí příkazu:

```
$ make clean
```

7.2 Spuštění serveru

Po přeložení se vytvoří zkompilevaný soubor **marias.exe**. Tento soubor je spouštěn z příkazové řádky a nevyžaduje žádné povinné parametry. Seznam dostupných volitelných parametrů lze zobrazit pomocí přepínače **-h**.

Základní spuštění serveru má následující podobu:

```
$ ./marias
```

Pro zobrazení nápovědy je možné použít příkaz:

```
$ ./marias -h
```

7.3 Spuštění klienta

Klientská část aplikace je spustitelná jak na operačním systému **Linux**, tak na **Windows**. Pro spuštění klienta je nutné vytvořit virtuální prostředí a nainstalovat externí knihovnu **Pygame**, bez které není možné aplikaci spustit. Požadovaná verze jazyka Python je **3.10** nebo vyšší.

Klient nepodporuje žádné vstupní parametry. Po splnění všech výše uvedených podmínek lze klienta spustit následujícím příkazem:

```
$ python main.py
```

8 Závěr

Cílem této semestrální práce bylo navrhnout a implementovat síťovou multiplayerovou aplikaci založenou na klient–server architektuře, která umožňuje hraní karetní hry Mariáš pro více hráčů. Práce se zaměřovala nejen na samotnou herní logiku, ale především na návrh spolehlivé síťové komunikace, správu herních relací a řešení výpadků spojení, které jsou v reálném síťovém prostředí běžným jevem.

V rámci práce byl úspěšně implementován server v jazyce C++ a klientská aplikace v jazyce Python s grafickým uživatelským rozhraním postaveným na knihovně Pygame. Komunikace mezi klientem a serverem probíhá prostřednictvím vlastního binárního protokolu nad protokolem TCP, jehož návrh zohledňuje potřebu detekce chyb, sekvenčního řazení paketů a možnosti jejich zpětného zaslání. Zvláštní důraz byl kladen na robustnost aplikace, zejména na mechanismus automatického znovupřipojení klienta a obnovy herního stavu po krátkodobém i dlouhodobém výpadku spojení.

Herní logika byla navržena tak, aby veškeré autoritativní rozhodování probíhalo výhradně na straně serveru, čímž je zamezeno možnosti podvádění ze strany klientů. Stav hry je klientům průběžně synchronizován pomocí stavových zpráv, což zajišťuje konzistentní zobrazení hry u všech hráčů. Vícevláknová architektura serveru umožňuje paralelní obsluhu více klientů a herních místností, aniž by docházelo k blokování herní logiky.

Výsledkem práce je plně funkční síťová hra, která splňuje zadání semestrální práce a demonstruje praktické využití znalostí z oblasti počítačových sítí, paralelního programování a návrhu klient–server aplikací. Implementované řešení je modulární, rozšiřitelné a připravené na další vývoj.

Mezi možná budoucí rozšíření patří například implementace plných pravidel hry Mariáš, podpora šifrované komunikace, přidání perzistentního ukládání herních statistik nebo optimalizace síťového protokolu. Další možností je rovněž rozšíření klientské části o pokročilejší grafické efekty či podporu mobilních platforem.

Tato práce tak poskytuje solidní základ pro další rozvoj síťových herních aplikací a zároveň slouží jako praktická ukázka návrhu a implementace robustní klient–server komunikace v reálném prostředí.