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: 7. ledna 2026

Obsah

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

Vzhledem k rozsáhlosti semestrální práce je rozsáhlé i zadání. 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 zprávy. Hlavička se skládala z 4 částí:

```
typedef struct {
        uint16_t size; // 2B
        uint8_t packetID; // 1B
        uint8_t clientID; // 1B
        MessageType type; // 1B (mapped to uint8_t)
} Measure;
```

Hlavička zabírala celkem 5B, zbytek byl tvořen přenášenou zprávou.

3.2.1 Detekce špatné zprávy

Při příjmu zprávy se kontroluje hlavička zprávy. Zpráva se rozdělí na hlavičku a data, Zkontrolují se validní hodnoty hlavičky zprávy, pokud jedna z hodnot není správná dojde k odpojení klienta.

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 tedy 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

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*.

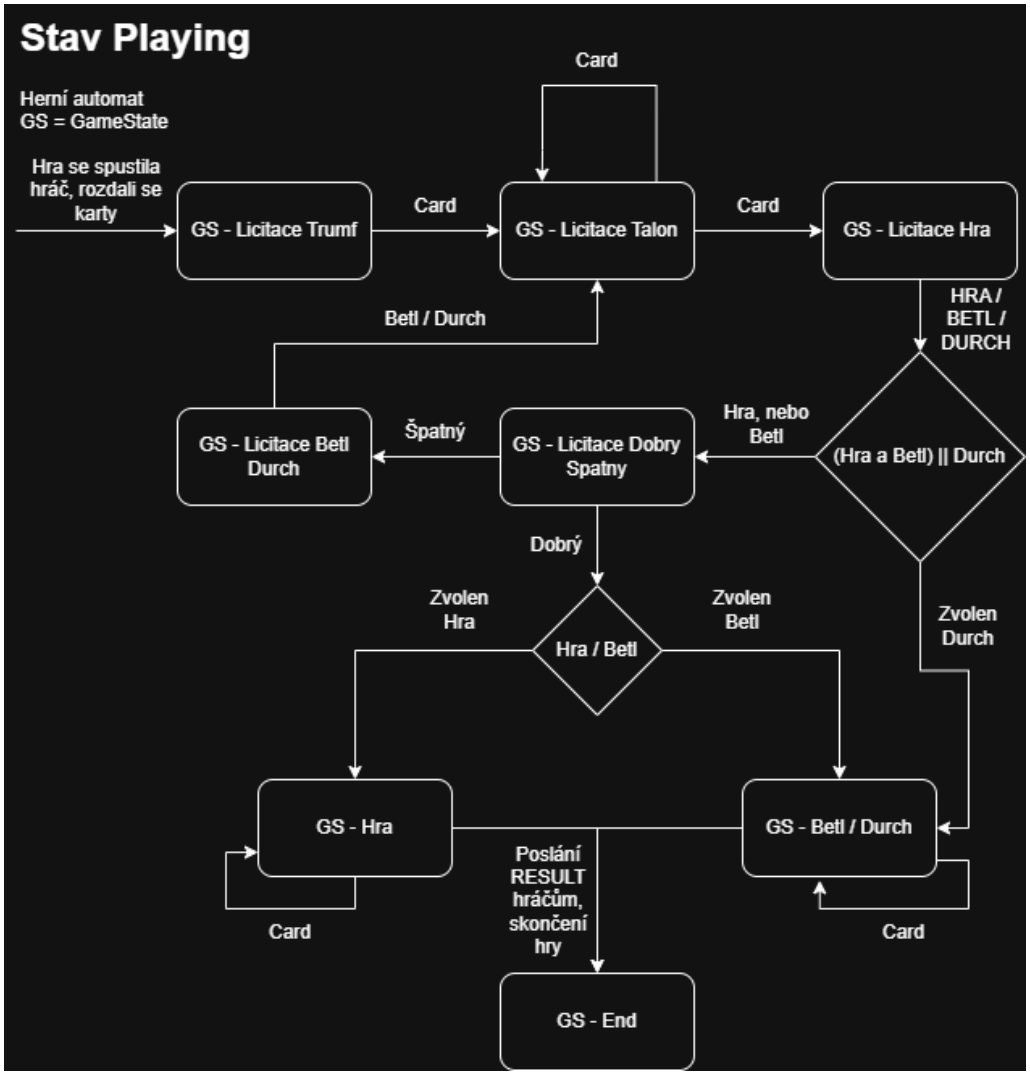
3.5 Zpětné připojení do hry (Reconnect)

Cílem zadání bylo navrhnout a implementovat mechanismus umožňující zvládnutí jak krátkodobého, tak i dlouhodobého výpadku spojení klienta se serverem. V případě detekce přerušení socketového spojení je na straně klienta automaticky spuštěn mechanismus *automatického znovupřipojení*.

Proměnná `max_reconnect_attempts` určuje maximální počet pokusů o navázání nového spojení se serverem. Mezi jednotlivými pokusy je zavedena časová prodleva definovaná proměnnou `reconnect_delay`. Obě tyto hodnoty jsou nastaveny na 5, což znamená, že celková maximální doba automatického znovupřipojení činí 25 sekund.

Na straně serveru je definována maximální doba, po kterou je klient považován za dočasně odpojeného, pomocí konstanty `RECONNECT_TIMEOUT_SECONDS`. Klient má k dispozici 60 sekund na opětovné navázání spojení. V případě, že automatické znovupřipojení selže, je uživatel přesměrován zpět do hlavního menu aplikace, kde má k dispozici manuální možnost opětovného připojení prostřednictvím tlačítka *Reconnect*, které iniciuje stejný proces jako automatický mechanismus.

Dlouhodobý výpadek spojení je řešen úplným odpojením klienta ze serveru. Pokud se klient nepřipojí zpět do jedné minuty od vzniku výpadku, server jej definitivně odstraní z aktivní herní relace.

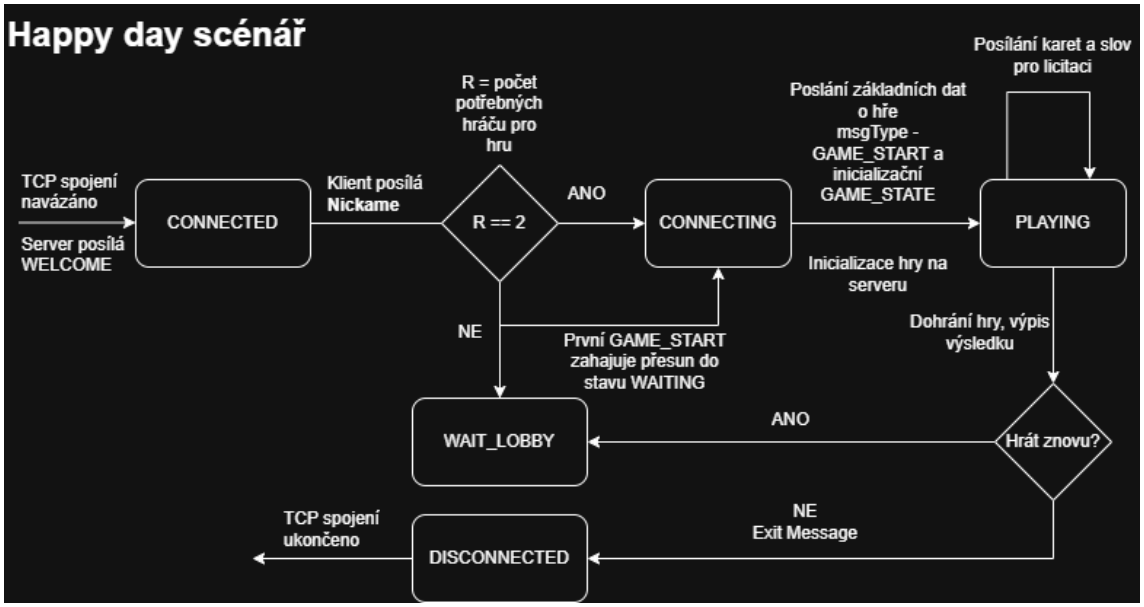


Obrázek 1: Vizualizace stavového diagramu hry

3.5.1 Algoritmus zpětného posílání paketů

Při návrhu mechanismu pro opětovné připojení klienta po výpadku konektivity bylo nutné implementovat metodu pro dodatečné zaslání ztracených dat (paketů). Vzhledem k tomu, že na straně serveru nebyla realizována průběžná potvrzení o doručení (ACK), nebylo možné využít standardní protokoly jako *Go-Back-N* nebo *Selective Repeat*.

Navržený algoritmus využívá na straně serveru vyrovnávací paměť (buffer) typu pole, do které jsou odchozí pakety ukládány. Každý paket je opatřen identifikátorem `clientID` pro určení adresáta a sekvenčním číslem `packetID` pro zajištění správného pořadí a identifikaci.



Obrázek 2: Vizualizace komunikace při šťastném scénáři

Proces obnovy dat probíhá v následujících krocích:

1. **Zaslání požadavku**: Klient po obnovení spojení odešle serveru zprávu typu `RECONNECT`.
2. **Identifikace stavu**: Součástí této zprávy je `packetID` posledního úspěšně přijatého paketu, který má klient uložen v lokální paměti.
3. **Vyhodnocení na serveru**: Server porovná ID od klienta s posledním odeslaným ID. Pokud se hodnoty shodují, stav klienta je aktuální.
4. **Opětovné poslání**: V případě neshody server vyhledá v bufferu všechny pakety příslušející danému `clientID` s vyšším sekvenčním číslem, než které uvedl klient. Tyto pakety jsou následně v původním pořadí znovu odeslány.

4 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.

4.1 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 handlery 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 přijímaní 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 soketu, 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 binárního protokolu použitého pro komunikaci. Každá zpráva se skládá z hlavičky pevné délky (5 bajtů) a těla s daty. Hlavička obsahuje:
  - **Velikost (2B)** - Celková délka zprávy.
  - **ID Paketu (1B)** - Sekvenční číslo pro detekci ztrát a správné pořadí.
  - **ID Klienta (1B)** - Identifikátor odesílatele/příjemce.
  - **Typ zprávy (1B)** - Identifikátor akce (např. `CONNECT`, `CARD`, `STATE`).

  Data jsou serializována jako řetězce oddělené znakem `'|'`.

- **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.

4.2 Implementace herní logiky (Složka *game*)

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, odebírání 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.

4.3 Síťová komunikace a robustnost

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

4.3.1 Mechanismus Heartbeat

Server pravidelně očekává od klientů zprávy typu `HEARTBEAT`. Pokud klient nepošle žádnou zprávu po definovanou dobu, server jej označí jako dočasně odpojeného a zahájí odpočet pro možnost opětovného připojení. Tímto je zajištěno, že hra nezanikne při krátkodobém výpadku Wi-Fi nebo přepnutí sítě na straně klienta.

4.3.2 Detekce ztráty paketů a retransmise

Díky sekvenčnímu číslování paketů v hlavičce protokolu může klient detekovat mezeru v přijatých zprávách. V takovém případě klient požádá server o znovuzaslání chybějících dat. Server pro tyto účely udržuje kruhový buffer posledních odeslaných paketů pro každého klienta.

4.3.3 Vícevláknové zpracování

Server využívá model "vlákno na klienta"pro přijímání dat, zatímco herní logika v `GameManager` je chráněna mutexy. To umožňuje serveru reagovat na zprávy od různých hráčů asynchronně (např. zpracování chatu nebo požadavku na stav), zatímco hlavní herní vlákno čeká na tah konkrétního hráče.

5 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

5.1 Technologie

Projekt využívá následující technologie:

- **Python 3.x** – programovací jazyk
- **Pygame** – knihovna pro grafické rozhraní a herní logiku
- **Socket** – síťová komunikace
- **Threading** – asynchronní zpracování síťových zpráv
- **Enum** – typově bezpečné výčtové typy

5.2 Architektura aplikace

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

5.2.1 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
```

5.2.2 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

5.3 Hlavní komponenty

5.3.1 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

Klíčové metody

- `run()` – Hlavní herní smyčka s optimalizací vykreslování
- `setup_client_callbacks()` – Nastavení callbacků pro zprávy od serveru
- `handle_server_message()` – Zpracování příchozích zpráv
- `connect_to_server()` – Připojení k serveru s validací vstupů

5.3.2 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

Reconnect mechanismus

Při ztrátě spojení se klient automaticky pokouší o znovupřipojení:

1. Detekce výpadku (timeout při čtení nebo chyba socketu)
2. Spuštění reconnect vlákna
3. Lineární backoff (0s, 5s, 10s, 15s, 20s, ...)
4. Maximálně 5 pokusů o znovupřipojení
5. Odeslání RECONNECT zprávy po úspěšném připojení

5.3.3 GameManager – Správa hry

Třída `GameManager` spojuje herní logiku s grafickým rozhraním.

Odpovědnosti

- Vykreslování karet hráče
- Zobrazení zahraných karet na stole
- Vykreslování licitačních tlačítek
- Zpracování kliknutí na karty

- Parsování herních dat ze serveru
- Zobrazení výsledků hry

Parsování dat
GameManager obsahuje metody pro převod dat ze serveru do objektů:

- `card_reader()` – Převod stringu na objekt Card
- `player_reader()` – Vytvoření objektu Player z dat
- `state_reader()` – Aktualizace herního stavu
- `game_start_reader()` – Inicializace nové hry

5.3.4 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

END – Konec hry

5.3.5 Card – Reprezentace karty

Třída `Card` reprezentuje jednu hrací kartu.

Vlastnosti karty

- **Hodnota (rank)** – A, K, Q, J, X (10), IX (9), VIII (8), VII (7)
- **Barva (suit)** – Srdce (♡), Kule (♢), Žaludy (♣), Listy (♠)

5.3.6 Player – Reprezentace hráče

Třída `Player` reprezentuje jednoho hráče ve hře.

Třída Hand
Třída `Hand` spravuje karty v ruce hráče:

- `sort(mode)` – Seřadí karty podle barvy a hodnoty
- `find_card_in_hand()` – Najde kartu v ruce

- `add_card()` – Přidá kartu do ruky
- `remove_card()` – Odebere kartu z ruky

5.3.7 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()` – Nápověda s pravidly hry

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

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

5.4 Tok dat a komunikace

5.4.1 Připojení k serveru

1. Uživatel zadá IP, port a přezdívku v lobby
2. Validace vstupů (InputValidator)
3. Gui volá `connect_to_server()`
4. ClientManager vytvoří TCP socket a připojí se
5. Odeslání CONNECT zprávy s přezdívkou
6. Server odpoví WELCOME zprávou s číslem klienta
7. Přechod do stavu WAITING

5.4.2 Herní smyčka

1. Server odešle GAME_START zprávu
2. GameManager inicializuje hru a hráče
3. Přechod do stavu PLAYING
4. Server průběžně posílá STATE zprávy s aktualizacemi

5. Při tahu hráče server pošle YOUR_TURN
6. Hráč klikne na kartu
7. Odeslání CARD zprávy serveru
8. Server validuje tah a pošle nový STATE
9. Opakování dokud hra neskončí
10. Server pošle RESULT zprávu s výsledkem

5.5 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 vlákně
4. Volání příslušného callbacku v Gui
5. Aktualizace stavu a označení pro překreslení
6. Vykreslení změn v hlavní smyčce

5.6 Optimalizace a výkon

5.6.1 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

5.6.2 Síťová komunikace

- Asynchronní příjem v samostatném vlákně
- Fronta zpráv pro thread-safe zpracování
- Heartbeat každých 10 sekund
- Timeout 30 sekund pro detekci výpadku
- Exponenciální backoff při reconnectu

5.7 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

5.8 Zpracování chyb

5.8.1 Síťové chyby

- Timeout při připojení – zobrazení chyby v GUI
- Ztráta spojení – automatický reconnect
- Chyba při parsování zprávy – logování a pokračování
- Neplatná zpráva od serveru – zobrazení ERROR

5.8.2 Herní chyby

- Neplatný tah – server odpoví INVALID zprávou
- Zobrazení chybové hlášky v GUI
- Hráč může zkusit jiný tah

6 Spuštění a běh programu

6.1 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 *server* spuštěním následujícího příkazu:

```
$ make
```

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

```
$ make clean
```

6.2 Spuštění serveru

Server je spouštěn z příkazové řádky a nevyžaduje žádné povinné parametry. Seznam dostupných volitelných parametrů lze zobrazit pomocí přepínače **-h**. Přeložitelný soubor se nachází v hlavním adresáři *server*.

Základní spuštění serveru má následující podobu:

```
$ ./marias
```

Pro zobrazení nápovědy je možné použít příkaz:

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

6.3 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
```

7 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.