Síťová hra Pexeso

Semestrální práce z předmětu KIV/UPS

Architektura server-klient (1:N)

Textový protokol přes TCP

Server: C (BSD sockets, POSIX threads)
Klient: Java (JavaFX, java.net.Socket)

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
11. ledna 2026

Obsah

1 Úvod

Tato dokumentace popisuje implementaci síťové hry Memory (Pexeso) jako semestrální práce z předmětu KIV/UPS. Projekt implementuje kompletní client-server architekturu s podporou multiplayer režimu (2–4 hráči), lobby systémem, automatickým reconnect mechanismem a robustním textovým protokolem přes TCP.

1.1 Popis hry

Pexeso je tahová paměťová hra pro 2–4 hráče. Herní deska obsahuje sudý počet karet (typicky 16, 24, 32 nebo 36 karet) rozmístěných lícem dolů. Každá karta má přidělenou hodnotu (symbol) a každá hodnota se vyskytuje přesně 2×.

Pravidla:

- Hráči se střídají v tazích
- V každém tahu hráč otočí postupně 2 karty
- Pokud se hodnoty karet shodují → hráč získává 1 bod a hraje znovu
- Pokud se neshodují → karty se vrátí lícem dolů a přichází další hráč
- Hra končí, když jsou všechny páry nalezeny
- Vyhrává hráč(i) s nejvyšším skóre

1.2 Architektura systému

Systém se skládá ze dvou hlavních komponent:

Server (C) Spravuje herní logiku, místnosti, připojené klienty a stav her. Implementován v jazyce C s využitím BSD sockets a POSIX threads.

Klient (Java) Grafická aplikace s JavaFX pro zobrazení hry a interakci s uživatelem. Komunikuje se serverem pomocí textového protokolu.

Klíčové vlastnosti:

- Textový protokol (ASCII) přes TCP
- Podpora 2–4 hráčů současně
- Lobby systém s výběrem místností
- Automatický reconnect po výpadku
- PING/PONG keepalive mechanismus
- Validace všech síťových zpráv
- Logování na serveru i klientovi
- Paralelní běh více herních místností

2 Komunikační protokol

2.1 Základní charakteristika

Protokol je navržen jako **textový aplikační protokol** nad transportním protokolem TCP. Všechny zprávy jsou kódovány v ASCII bez diakritiky.

Parametr	Hodnota
Typ protokolu	Textový, aplikační vrstva
Transportní protokol	TCP
Kódování	ASCII (bez diakritiky)
Ukončení zprávy	`\n` (newline)
Formát zprávy	`COMMAND [PARAM1] [PARAM2] ...`
Oddělování parametrů	Jedna mezera (ASCII 0x20)

Tabulka 1: Základní parametry protokolu

2.2 Datové typy a omezení

Typ	Popis	Formát	Omezení
nickname	Přezdívka hráče	String bez mezer	1–16 znaků -> a-z, A-Z, 0-9, -
room_id	ID místnosti	Integer	1–9999
room_name	Název místnosti	String bez mezer	1–20 znaků
client_id	ID klienta	Integer	1–9999
card_id	Index karty	Integer	0–(board_size-1)
card_value	Hodnota karty	Integer	0–(board_size/2-1)
board_size	Počet karet	Integer	16, 24, 32, 36
max_players	Max. hráčů	Integer	2–4
score	Skóre hráče	Integer	≥ 0

Tabulka 2: Datové typy protokolu

2.3 Příklady zpráv

Klient → Server:

```
HELLO Alice
LIST_ROOMS
CREATE_ROOM Game1 2 16
JOIN_ROOM 42
START_GAME
FLIP 5
PONG
RECONNECT 123
```

Server → Klient:

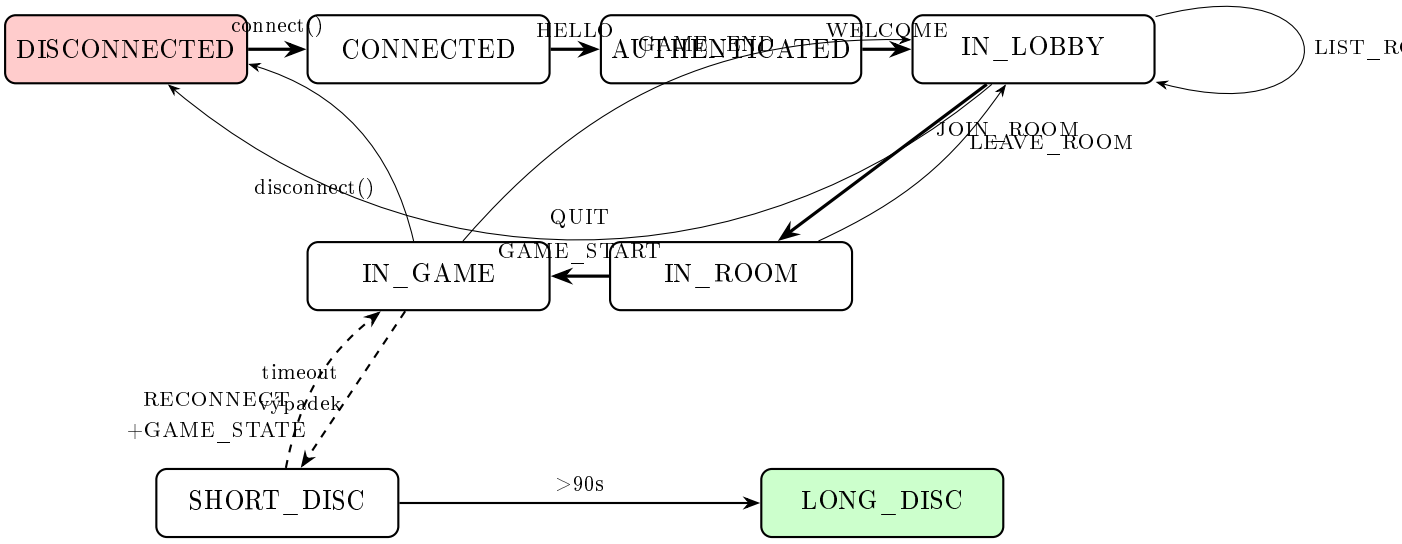
```
WELCOME 1
ROOM_LIST 2 1 Game1 2 4 WAITING 2 Game2 1 2 PLAYING
ROOM_CREATED 1 Game1
GAME_START 16 2 Alice Bob
YOUR_TURN
CARD_REVEAL 5 3 Alice
MATCH Alice 1
MISMATCH Bob
GAME_END Alice 8 Bob 6
PING
PLAYER_DISCONNECTED Bob SHORT
ERROR INVALID_COMMAND Unknown command
```

2.4 Stavový diagram klienta

Klient prochází následujícími stavy:

1. `DISCONNECTED` – Nepřipojeno k serveru
2. `CONNECTED` – TCP spojení navázáno
3. `AUTHENTICATED` – Po úspěšném `HELLO`
4. `IN_LOBBY` – V lobby (může listovat místnosti)
5. `IN_ROOM` – V místnosti (čeká na start)
6. `IN_GAME` – Hra běží
7. `SHORT_DISCONNECT` – Krátkodobý výpadek (< 90s)
8. `LONG_DISCONNECT` – Dlouhodobý výpadek (> 90s)

Přechody mezi stavy jsou řízeny protokolovými zprávami (např. `WELCOME`, `ROOM_JOINED`, `GAME_START`).



Obrázek 1: Stavový diagram klienta

2.5 Keepalive mechanismus (PING/PONG)

Server pravidelně kontroluje živost klientů pomocí PING/PONG mechanismu:

- Server posílá `PING` každých 5 sekund po přijetí `PONG`
- Klient musí odpovědět `PONG` do 5 sekund
- Pokud klient neodpoví → server označí klienta jako odpojeného
- Klient má `READ_TIMEOUT` 15 sekund – pokud od serveru nepřijde žádná zpráva, detekuje výpadek

2.6 Reconnect mechanismus

Krátkodobý výpadek (SHORT_DISCONNECT):

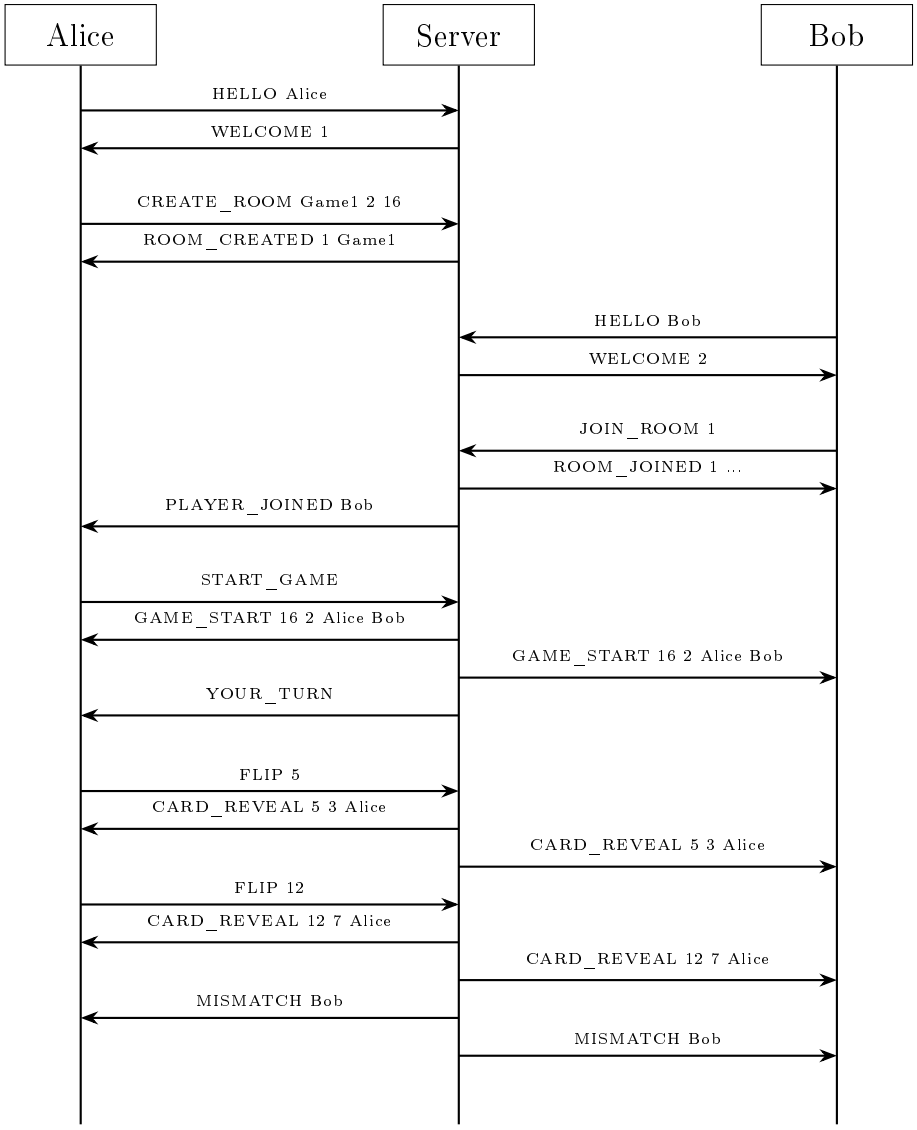
- Klient detekuje odpojení (READ_TIMEOUT 15s)
- Automatický reconnect: 7 pokusů × 10s interval = 70s celkem
- Server čeká 90 sekund na reconnect
- Po úspěšném reconnectu server pošle `GAME_STATE` s aktuálním stavem hry
- Hra pokračuje ze stejného stavu

Dlouhodobý výpadek (LONG_DISCONNECT):

- Klient zkouší reconnect 70s, poté vzdá
- Server po 90s bez reconnectu odstraní hráče
- Server pošle `PLAYER_DISCONNECTED <nick> LONG`
- Pokud zbývá < 2 hráči → hra končí (`GAME_END_FORFEIT`)
- Pokud zbývá ≥ 2 hráči → hra pokračuje bez odpojeného hráče

2.7 Návaznost zpráv (sekvenční diagram)

Následující diagram ukazuje typickou komunikaci při připojení, vytvoření hry a hraní:



Obrázek 2: Sekvenční diagram typické komunikace (připojení, vytvoření hry, herní tah)

2.8 Validace a chybové stavy

Server validuje všechny příchozí zprávy a odpovídá chybovými kódy:

Error kód	**Popis**
INVALID_COMMAND	Neznámý příkaz
INVALID_PARAMS	Špatný počet/formát parametrů
NOT_AUTHENTICATED	Příkaz vyžaduje autentizaci
NICK_IN_USE	Přezdívka je již používána
ROOM_NOT_FOUND	Místnost neexistuje
ROOM_FULL	Místnost je plná
NOT_IN_ROOM	Hráč není v místnosti
NOT_ROOM_OWNER	Pouze vlastník může startovat hru
NOT_YOUR_TURN	Není na tahu tento hráč
INVALID_CARD	Neplatné ID karty

Tabulka 3: Chybové kódy protokolu

Server odpojí klienta po **3 nevalidních zprávách** (konfigurovatelné).

3 Implementace serveru (C)

3.1 Architektura a moduly

Server je implementován v jazyce C s následující modulární strukturou:

Modul	Odpovědnost
`main.c`	Vstupní bod, parsování argumentů, inicializace serveru
`server.c`	Listening socket, accept loop, vytváření threadů
`client_handler.c`	Obsluha jednotlivých klientů, zpracování zpráv
`client_list.c`	Správa seznamu připojených klientů
`room.c`	Správa lobby a herních místností
`game.c`	Logika hry Pexeso (deska, tahy, vyhodnocení)
`logger.c`	Thread-safe logování do souboru
`protocol.h`	Definice protokolových konstant

Tabulka 4: Moduly serveru

3.2 Datové struktury

Struktura klienta:

```c
typedef struct client_t {
    int client_id;            // Unikatni ID
    int socket_fd;            // Socket file descriptor
    char nickname[MAX_NICK_LENGTH];
    client_state_t state;     // CONNECTED/AUTHENTICATED/...
    time_t last_activity;     // Posledni aktivita
    int invalid_message_count; // Pocet chyb
    int is_disconnected;      // Flag odpojeni
    pthread_mutex_t mutex;    // Thread safety
} client_t;
```

Struktura místnosti:

```c
typedef struct room_t {
    int room_id;
    char name[MAX_ROOM_NAME_LENGTH];
    client_t *players[MAX_PLAYERS_PER_ROOM];
    int player_count;
    int max_players;
    client_t *owner;
    room_state_t state;       // WAITING/PLAYING
    game_t *game;             // Hra (pokud bezi)
    pthread_mutex_t mutex;
} room_t;
```

Struktura hry:

```c
typedef struct game_t {
    int board_size;          // Pocet karet
    card_t *cards;           // Pole karet
    client_t **players;      // Hraci
    int player_count;
    int current_player;      // Index aktualniho hrace
    int *scores;             // Skore hracu
    int flipped_count;       // Pocet otevenych karet
    int first_card_id;       // ID prvni karty
} game_t;
```

3.3 Paralelizace – POSIX threads

Server používá **thread-per-client** model:

- Hlavní thread běží v `server_run()` – nekonečná accept loop
- Pro každého nového klienta se vytvoří vlastní thread (`pthread_create`)
- Thread se detachuje (`pthread_detach`) → automatický cleanup
- Každý thread má vlastní `client_t` strukturu
- Logger používá `pthread_mutex_t` pro thread-safe zápis
- Room broadcast operace jsou synchronizované mutexem

Výhody thread-per-client:

- Jednoduchá implementace – každý klient má vlastní kontext
- Blokující I/O operace neovlivňují ostatní klienty
- Přirozená izolace mezi klienty
- Snadný debugging

3.4 Použité knihovny a API

BSD Sockets (POSIX):

```c
#include <sys/socket.h>    // socket(), bind(), listen(), accept()
#include <netinet/in.h>    // struct sockaddr_in
#include <arpa/inet.h>     // inet_pton(), inet_ntop()
```

POSIX Threads:

```c
#include <pthread.h>       // pthread_create(), pthread_mutex_t
```

Standardní knihovna:

```c
#include <unistd.h>        // close(), read(), write()
#include <string.h>        // strcpy(), strncpy(), strcmp()
#include <stdlib.h>        // malloc(), free(), atoi()
#include <stdio.h>         // printf(), snprintf(), fopen()
#include <time.h>          // time(), localtime()
```

Všechny použité knihovny jsou součástí POSIX standardu. Nebyly použity žádné externí networking knihovny.

3.5 Konfigurace serveru

Server přijímá konfiguraci z příkazové řádky:

```
./server <IP> <PORT> <MAX_ROOMS> <MAX_CLIENTS>
```

Příklad:

```
./server 0.0.0.0 10000 10 50
```

Parametry:

- `IP` – IP adresa pro bind (např. `0.0.0.0` pro všechna rozhraní)
- `PORT` – Port pro listening (např. `10000`)
- `MAX_ROOMS` – Maximální počet místností (např. `10`)
- `MAX_CLIENTS` – Maximální počet klientů celkem (např. `50`)

3.6 Logování

Server loguje všechny důležité události do souboru `server.log`:

- Připojení/odpojení klientů (IP, port, čas)
- Autentizace (nickname, client_id)
- Vytvoření a zrušení místností
- Start a konec her
- Výpadky (SHORT/LONG disconnect)
- Reconnect události
- Nevalidní zprávy (příkaz, client_id)
- Interní chyby

Příklad logu:

```
[2026-01-11 12:00:00] [INFO] Server initialized on 0.0.0.0:10000
[2026-01-11 12:00:05] [INFO] Client 1 connected from
   127.0.0.1:54321
[2026-01-11 12:00:06] [INFO] Client 1 authenticated as 'Alice'
[2026-01-11 12:00:10] [INFO] Room 1 created: Game1 (max 2 players)
[2026-01-11 12:00:15] [INFO] Client 2 joined room 1
[2026-01-11 12:00:20] [INFO] Game started in room 1
[2026-01-11 12:01:30] [WARNING] Client 1: Invalid command 'INVALID'
[2026-01-11 12:02:00] [INFO] Game ended in room 1: Alice 8, Bob 6
```

4 Implementace klienta (Java + JavaFX)

4.1 Architektura MVC

Klient je implementován v Javě s využitím JavaFX a architekturou Model-View-Controller:

Balíček/Třída	Odpovědnost
`Main.java`	Vstupní bod, správa scén, inicializace loggeru
`network/ClientConnection`	TCP spojení, async čtení, auto-reconnect
`network/MessageListener`	Rozhraní pro příjem zpráv
`protocol/ProtocolConstants`	Definice protokolu, timeouty
`model/Room`	Model herní místnosti
`controller/LoginController`	Ovládání přihlašovací obrazovky
`controller/LobbyController`	Ovládání lobby (seznam místností)
`controller/GameController`	Ovládání herní obrazovky
`util/Logger`	Thread-safe logování do `client.log`

Tabulka 5: Moduly klienta

4.2 Síťová komunikace

Asynchronní model:

Klient používá oddělené vlákno pro síťovou komunikaci, aby GUI nezamrzalo:

```
// Cteni v oddelnem vlakne
private void readerLoop() {
    while (running && (line = in.readLine()) != null) {
        final String message = line.trim();
        if (listener != null) {
            // GUI aktualizace pres Platform.runLater()
            listener.onMessageReceived(message);
        }
    }
}
```

Schéma komunikace:

```
[GUI Thread] -> sendMessage() -> socket
[Reader Thread] -> readLine() -> Platform.runLater() -> onMessageReceived()
```

4.3 Použité knihovny a API

Java Standard Library (java.net):

```
import java.net.Socket;              // TCP socket
import java.net.InetSocketAddress;   // Adresa serveru
```

Java I/O:

```
import java.io.BufferedReader;       // Cteni textu
import java.io.InputStreamReader;    // Stream -> Reader
import java.io.PrintWriter;          // Zapis textu
```

JavaFX (GUI):

```
import javafx.application.Application;
import javafx.application.Platform;  // runLater()
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.fxml.FXMLLoader;
```

Všechny použité knihovny jsou součástí Java Standard Edition. Nebyly použity žádné externí networking knihovny.

4.4 Timeouty a reconnect

Konstanta	Hodnota	Popis
CONNECTION_TIMEOUT	5s	Timeout při navazování spojení
READ_TIMEOUT	15s	Timeout při čtení ze socketu
RECONNECT_INTERVAL	10s	Interval mezi reconnect pokusy
MAX_RECONNECT_ATTEMPTS	7	Počet pokusů (= 70s celkem)

Tabulka 6: Timeouty klienta

4.5 GUI a uživatelské rozhraní

Klient implementuje **grafické rozhraní pomocí JavaFX** (ne konzole):

Login obrazovka – Zadání IP, portu a přezdívky

Lobby obrazovka – Seznam místností, možnost vytvořit/připojit se

Game obrazovka – Herní deska, karty, skóre, informace o hráčích

Klíčové vlastnosti GUI:

- Non-blocking – síťové operace v odděleném threadu
- Aktuální stav hry – herní pole, přezdívky, kdo je na tahu
- Indikace nedostupnosti – server/protihráč offline
- Validace vstupů – IP adresa, port, nickname, herní tahy

4.6 Logování

Klient loguje události do souboru `client.log`:

- Připojení k serveru
- Autentizace (client_id)
- Reconnect události
- Chyby (síťové, protokolové)
- Změny stavů

Příklad logu:

```
[2026-01-11 12:00:00] [INFO] Logger initialized
[2026-01-11 12:00:00] [INFO] Application starting...
[2026-01-11 12:00:05] [INFO] Connecting to 127.0.0.1:10000
[2026-01-11 12:00:05] [INFO] Connected successfully
[2026-01-11 12:00:06] [INFO] Client authenticated (ID: 1)
[2026-01-11 12:01:30] [WARNING] Connection timeout - attempting
   reconnect
[2026-01-11 12:01:40] [INFO] Reconnect successful after 1 attempts
```

5 Překlad a spuštění

5.1 Požadavky na prostředí

Server (C):

- GNU/Linux
- GCC 7.5+ nebo Clang 6.0+
- GNU Make 4.1+
- POSIX threads (libpthread)

Klient (Java):

- Java SE 17+ (OpenJDK nebo Oracle JDK)
- Apache Maven 3.6+
- JavaFX 17+ (stáhne Maven automaticky)

5.2 Překlad serveru

Pomocí Makefile:

```
cd server_src
make clean    # Vycistit stare buildy
make          # Zkompilovat server
```

Makefile:

```
CC = gcc
CFLAGS = -Wall -Wextra -pthread -g
SOURCES = main.c server.c client_handler.c client_list.c \
          logger.c room.c game.c
TARGET = server

all:
    $(CC) $(CFLAGS) $(SOURCES) -o $(TARGET)

clean:
    rm -f $(TARGET)
```

5.3 Překlad klienta

Pomocí Maven:

```
cd client_src
mvn clean package
```

Maven vytvoří spustitelný JAR:

```
target/pexeso-client-1.0-SNAPSHOT.jar
```

5.4 Spuštění

Server:

```
./server <IP> <PORT> <MAX_ROOMS> <MAX_CLIENTS>

# Priklad:
./server 0.0.0.0 10000 10 50
```

Klient:

```
java -jar target/pexeso-client-1.0-SNAPSHOT.jar

# Nebo pres Maven:
mvn javafx:run
```

6 Testování

6.1 Testovací scénáře

1. Základní hra (2 hráči):

1. Alice se připojí, vytvoří místnost
2. Bob se připojí, vstoupí do místnosti
3. Alice spustí hru
4. Hráči střídavě otáčejí karty
5. Hra skončí, zobrazí se výsledky

2. Fragmentace zpráv (InTCPtor):

- Zprávy přicházejí po 1–2 bajtech
- Server/klient správně skládá fragmenty
- Hra běží bez chyb (jen pomaleji)

3. Reconnect mechanismus:

- Simulace krátkodobého výpadku (< 90s)
- Automatický reconnect klienta
- Obnovení stavu hry (GAME_STATE)
- Simulace dlouhodobého výpadku (> 90s)
- Odebrání hráče ze hry

4. Nevalidní zprávy:

- Náhodná data: `cat /dev/urandom | nc 127.0.0.1 10000`
- Malformed příkazy: `HELLO`, `FLIP` (bez parametrů)
- Příkazy ve špatném stavu: `FLIP` před `HELLO`
- Server loguje chyby a odpojí po 3 chybách

5. Memory leaks (Valgrind):

```
valgrind --leak-check=full ./server 0.0.0.0 10000 10 50
```

Očekávaný výsledek: `0 bytes definitely lost`

6.2 Výsledky testování

Test	Výsledek
Kompletní hra (2–4 hráči)	✓
Fragmentace zpráv (extrémní)	✓
Reconnect SHORT (< 90s)	✓
Reconnect LONG (> 90s)	✓
Nevalidní zprávy	✓
Memory leaks (valgrind)	✓ (0 leaks)
Paralelní místnosti	✓
PING/PONG keepalive	✓
GUI non-blocking	✓

Tabulka 7: Výsledky testování

7 Závěr

7.1 Dosažené výsledky

Projekt splňuje všechny požadavky zadání:

- **Kompletní síťová hra** – 2–4 hráči, lobby, místnosti
- **Textový protokol** – ASCII přes TCP, čitelný a rozšiřitelný
- **Robustní implementace** – validace zpráv, error handling, logování
- **Reconnect mechanismus** – automatický při krátkodobém výpadku
- **Paralelní běh** – více místností současně bez vzájemného ovlivňování
- **Stabilita** – bez memory leaks (ověřeno valgrind), bez segfaultů
- **Modulární kód** – čitelná struktura, dokumentované komentáři
- **Standardní build** – Makefile (server), Maven (klient)

7.2 Použité technologie

Server:

- Jazyk: C
- Síťování: BSD sockets (POSIX)
- Paralelizace: POSIX threads (thread-per-client)
- Build: GNU Make

Klient:

- Jazyk: Java 17
- Síťování: java.net.Socket (Java SE)
- GUI: JavaFX 17
- Build: Apache Maven

7.3 Možná rozšíření

V budoucnosti lze projekt rozšířit o:

- Chat mezi hráči v místnosti
- Statistiky hráčů (počet her, výher, průměrné skóre)
- Žebříček nejlepších hráčů
- Spectator režim (sledování hry bez účasti)
- Registrace a autentizace hráčů (přezdívka + heslo)
- Různé velikosti herních desek (konfigurovatelné při vytvoření místnosti)

7.4 Zhodnocení

Implementace prokázala schopnost navrhnout a realizovat kompletní síťovou aplikaci s robustním protokolem, automatickým zotavením po výpadcích a stabilním během. Projekt splňuje všechny požadavky zadání a byl úspěšně otestován na různých scénářích včetně extrémní fragmentace zpráv a síťových výpadků.

Klíčové poznatky:

- Důležitost správného buffering při fragmentaci zpráv

- Nutnost thread-safe operací při paralelním běhu
- Význam keepalive mechanismu pro detekci výpadků
- Potřeba důkladné validace všech vstupů (bezpečnost)