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)

Autor: Oldřich Jan Švehla
Osobní číslo: A23B0234P

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
15. ledna 2026

Obsah

1 Úvod

Tato dokumentace popisuje implementaci síťové hry Pexeso (Memory) jako semestrální práce z předmětu KIV/UPS. Projekt implementuje kompletní client-server architekturu s textovým protokolem nad TCP.

1.1 Popis hry

Pexeso je tahová paměťová hra pro 2–4 hráče. Herní deska obsahuje sudý počet karet (16, 36 nebo 64) rozmístěných lícem dolů v mřížce. Každá karta má přidělenou hodnotu (číslo/symbol) a každá hodnota se na desce vyskytuje přesně dvakrát.

Pravidla hry:

1. Hráči se střídají v tazích podle pořadí určeného serverem
2. V každém tahu hráč otočí postupně 2 karty (příkazy `FLIP`)
3. Pokud se hodnoty obou karet shodují:
   - Hráč získává 1 bod
   - Karty zůstávají otočené (jsou odebrány ze hry)
   - Hráč pokračuje dalším tahem
4. Pokud se hodnoty neshodují:
   - Karty se po krátké prodlevě vrátí lícem dolů
   - Na tahu je další hráč v pořadí
5. Hra končí, když jsou nalezeny všechny páry
6. Vyhrává hráč s nejvyšším skóre (při remíze více vítězů)

1.2 Architektura systému

Systém používá architekturu **server-klient (1:N)** s centrálním serverem obsluhujícím více klientů současně.

Server (C) Autoritativní server spravující veškerou herní logiku, místnosti, připojené klienty a stavy 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 přes TCP.

Klíčové vlastnosti systému:

- Textový protokol (ASCII) přes TCP – čitelný, snadno debugovatelný
- Podpora 2–4 hráčů v jedné herní místnosti
- Lobby systém s výběrem a vytvářením místností
- Automatický reconnect při krátkodobém výpadku (< 90s)
- PING/PONG keepalive mechanismus pro detekci výpadků
- Validace všech síťových zpráv (ochrana proti nevalidním datům)
- Logování událostí na serveru i klientovi
- Paralelní běh více herních místností bez vzájemného ovlivňování

2 Komunikační protokol

Tato sekce popisuje protokol dostatečně pro implementaci alternativního klienta nebo serveru.

2.1 Formát zpráv

Protokol je navržen jako **textový aplikační protokol** nad transportním protokolem TCP.

Parametr	Hodnota
Typ protokolu	Textový, aplikační vrstva
Transportní protokol	TCP
Kódování	ASCII (bez diakritiky, znaky 0x20–0x7E)
Ukončení zprávy	`\n` (newline, 0x0A)
Formát zprávy	`COMMAND [PARAM1] [PARAM2] ...`
Oddělování parametrů	Jedna mezera (ASCII 0x20)
Maximální délka	8192 bajtů

Tabulka 1: Základní parametry protokolu

Příklad zprávy: `FLIP 5\n` – hráč otáčí kartu s indexem 5.

2.2 Přenášené datové typy

Typ	Formát	Omezení
nickname	String bez mezer	1–16 znaků, povolené: a-z, A-Z, 0-9, _
room_name	String bez mezer	1–20 znaků
room_id	Celé číslo (Integer)	1–9999
client_id	Celé číslo (Integer)	1–9999
card_id	Celé číslo (Integer)	0 až (board_size - 1)
card_value	Celé číslo (Integer)	0 až (board_size / 2 - 1)
board_size	Celé číslo (Integer)	16, 36, nebo 64 (4×4, 6×6, 8×8)
max_players	Celé číslo (Integer)	2–4
score	Celé číslo (Integer)	≥ 0
state	String	WAITING, PLAYING, FINISHED

Tabulka 2: Přenášené datové typy a jejich omezení

2.3 Příkazy klient → server (10 příkazů)

Příkaz	Formát	Význam
HELLO	`HELLO <nick>`	Autentizace hráče s přezdívkou
LIST_ROOMS	`LIST_ROOMS`	Žádost o seznam místností
CREATE_ROOM	`CREATE_ROOM <name> <max> <size>`	Vytvoření nové místnosti
JOIN_ROOM	`JOIN_ROOM <room_id>`	Vstup do existující místnosti
LEAVE_ROOM	`LEAVE_ROOM`	Odchod z místnosti
START_GAME	`START_GAME`	Spuštění hry (pouze owner)
READY	`READY`	Potvrzení připravenosti ke hře
FLIP	`FLIP <card_id>`	Otočení karty na dané pozici
PONG	`PONG`	Odpověď na PING (keepalive)
RECONNECT	`RECONNECT <client_id>`	Znovupřipojení po výpadku

Tabulka 3: Příkazy od klienta k serveru

2.4 Příkazy server → klient (23 příkazů)

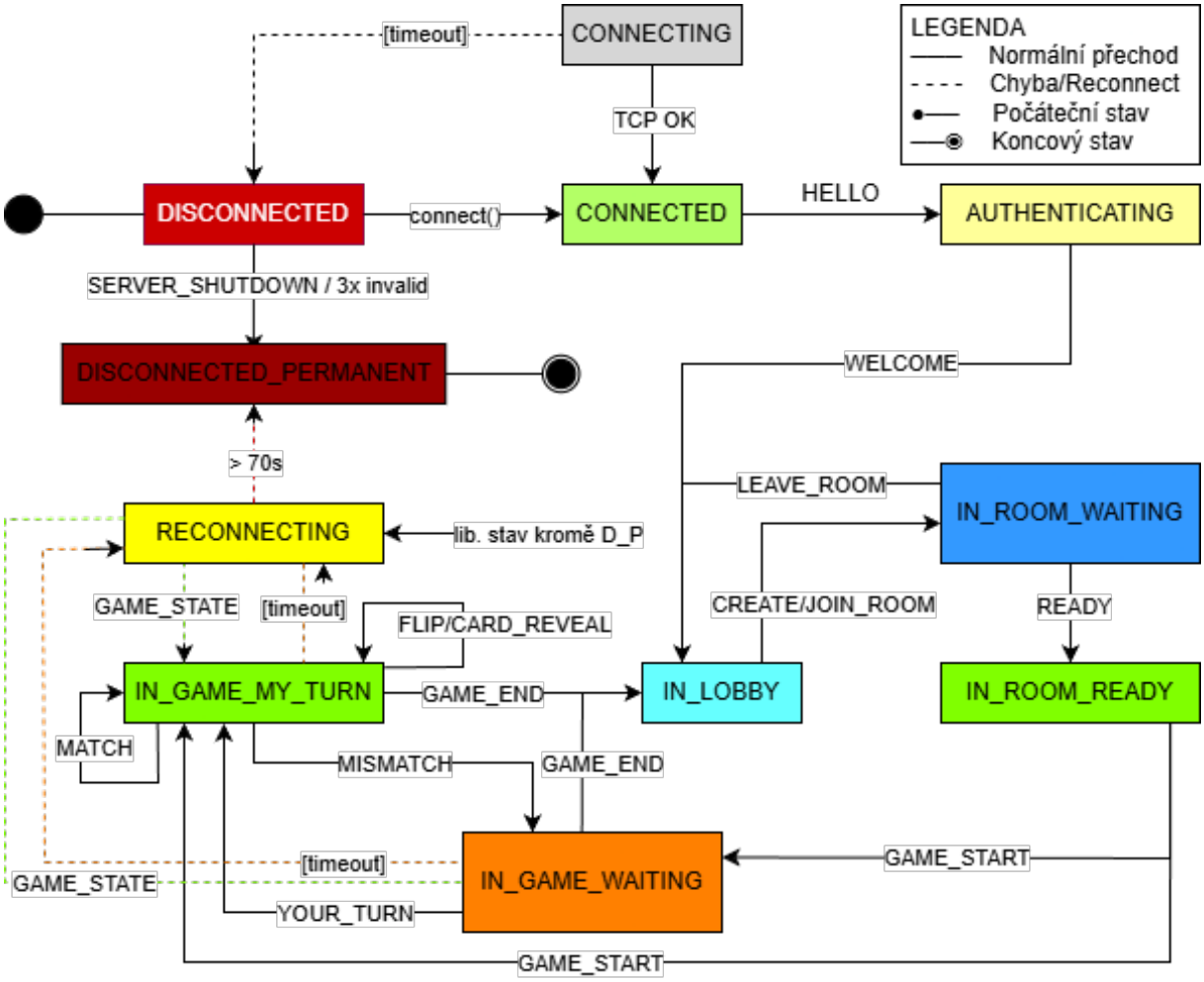
Příkaz	Formát a význam
WELCOME	`WELCOME <client_id>` – Úspěšná autentizace, přidělené ID
ROOM_LIST	`ROOM_LIST <count> [<id> <name> <curr> <max> <state> <size>]*`
ROOM_CREATED	`ROOM_CREATED <room_id> <name>` – Místnost vytvořena
ROOM_JOINED	`ROOM_JOINED <room_id> <name>` – Úspěšný vstup
PLAYER_JOINED	`PLAYER_JOINED <nick>` – Jiný hráč vstoupil
PLAYER_LEFT	`PLAYER_LEFT <nick>` – Jiný hráč odešel
PLAYER_READY	`PLAYER_READY <nick>` – Hráč je připraven
PLAYER_RECONNECTED	`PLAYER_RECONNECTED <nick>` – Hráč se znovu připojil
PLAYER_DISCONNECTED	`PLAYER_DISCONNECTED <nick> SHORT|LONG` – Výpadek
ROOM_OWNER_CHANGED	`ROOM_OWNER_CHANGED <nick>` – Nový vlastník místnosti
GAME_CREATED	`GAME_CREATED <board_size>` – Hra připravena
GAME_START	`GAME_START <size> <player1> <player2> ...` – Hra začíná
GAME_STATE	`GAME_STATE <size> <curr> <p1> <s1> ... <cards>` – Stav hry
GAME_END	`GAME_END <p1> <s1> <p2> <s2> ...` – Konec hry
GAME_END_FORFEIT	`GAME_END_FORFEIT <p1> <s1> ...` – Konec vzdáním
YOUR_TURN	`YOUR_TURN` – Jsi na tahu
CARD_REVEAL	`CARD_REVEAL <index> <value> <player>` – Otočená karta
MATCH	`MATCH <player> <score>` – Shoda karet, nové skóre
MISMATCH	`MISMATCH <next_player>` – Neshoda, další hráč
LEFT_ROOM	`LEFT_ROOM` – Potvrzení odchodu z místnosti
PING	`PING` – Keepalive dotaz
SERVER_SHUTDOWN	`SERVER_SHUTDOWN` – Server se vypíná
ERROR	`ERROR <code> <message>` – Chybová zpráva

Tabulka 4: Příkazy od serveru ke klientovi

2.5 Stavový diagram klienta

Klient prochází následujícími stavy na základě protokolových zpráv:

1. `DISCONNECTED` – Nepřipojeno k serveru (počáteční stav)
2. `CONNECTING` – Probíhá navazování TCP spojení
3. `CONNECTED` – TCP spojení navázáno
4. `AUTHENTICATING` – Odesláno `HELLO`, čeká se na `WELCOME`
5. `IN_LOBBY` – V lobby, může listovat/vytvářet místnosti
6. `IN_ROOM_WAITING` – V místnosti, čeká na další hráče
7. `IN_ROOM_READY` – Hráč potvrdil připravenost (`READY`)
8. `IN_GAME_WAITING` – Hra běží, čeká na svůj tah
9. `IN_GAME_MY_TURN` – Hra běží, hráč je na tahu
10. `RECONNECTING` – Pokus o automatické znovupřipojení
11. `DISCONNECTED_PERMANENT` – Trvalé odpojení (koncový stav)



Obrázek 1: Stavový diagram klienta – návaznost zpráv

2.6 Keepalive mechanismus (PING/PONG)

Server pravidelně kontroluje živost klientů:

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

2.7 Reconnect mechanismus

Krátkodobý výpadek ($<$ 90s):

- Klient automaticky zkouší reconnect: 7 pokusů $\times$ 10s = 70s celkem
- Server čeká 90 sekund na reconnect
- Ostatní hráči dostávají `PLAYER_DISCONNECTED <nick> SHORT`
- Po úspěšném reconnectu (`RECONNECT <id>`) server pošle `GAME_STATE`
- Hra pokračuje ze stejného stavu

Dlouhodobý výpadek ($>$ 90s):

- Server odstraní hráče ze hry
- Ostatní dostávají `PLAYER_DISCONNECTED <nick> LONG`
- Pokud zbývá $<$ 2 hráči, hra končí (`GAME_END_FORFEIT`)
- Klient musí provést novou autentizaci (`HELLO`)

2.8 Omezení vstupních hodnot a validace dat

Obě aplikace validují všechny příchozí zprávy na několika úrovních.

2.8.1 Validace na úrovni spojení

- **Binární/neplatná data** – Pouze ASCII tisknutelné znaky (0x20–0x7E)
- **Maximální délka zprávy** – 8192 bajtů (ochrana proti DoS)
- **Známý příkaz** – Zpráva musí začínat validním protokolovým příkazem
- **Počítadlo chyb** – Po 3 nevalidních zprávách je klient/server odpojen

2.8.2 Validace hodnot parametrů

Parametr	Validace	Příklad nevalidní hodnoty
card_id	$0 \leq$ card_id $<$ board_size	`FLIP -1`, `FLIP 999`
card_value	$0 \leq$ value $<$ board_size/2	záporná hodnota
score	score ≥ 0	záporné skóre
board_size	hodnota 16, 36, nebo 64	`GAME_START 5 ...`
nickname	neprázdný, 1–16 znaků	prázdný řetězec
room_id	> 0	`JOIN_ROOM 0`

Tabulka 5: Validace hodnot parametrů

2.9 Chybové stavy a jejich hlášení

Server odpovídá na chyby zprávou `ERROR <code> <message>`:

Kód chyby	Význam a kdy nastává
INVALID_COMMAND	Neznámý příkaz (překlep, neexistující příkaz)
INVALID_SYNTAX	Chybná syntaxe příkazu (chybějící parametr)
INVALID_PARAMS	Špatný počet nebo formát parametrů
NOT_AUTHENTICATED	Příkaz vyžaduje předchozí `HELLO`
ALREADY_AUTHENTICATED	Opakované `HELLO`
ROOM_NOT_FOUND	Místnost s daným ID neexistuje
ROOM_FULL	Místnost dosáhla limitu hráčů
NOT_IN_ROOM	Hráč není v žádné místnosti
NOT_ROOM_OWNER	Pouze vlastník může spustit hru
INVALID_MOVE	Neplatný herní tah (mimo tah, mimo hranice)
INVALID_CARD	Neplatný index karty nebo již otočená karta
GAME_NOT_STARTED	Příkaz `FLIP` mimo hru

Tabulka 6: Chybové kódy protokolu

Reakce na nevalidní data:

- Server/klient loguje chybu (bez obsahu dat z bezpečnostních důvodů)
- Inkrementuje počítadlo chyb
- Po 3 chybách odpojí protistranu (klient bez auto-reconnect)
- Server odpovídá `ERROR`, klient ignoruje nevalidní zprávy

3 Implementace serveru (C)

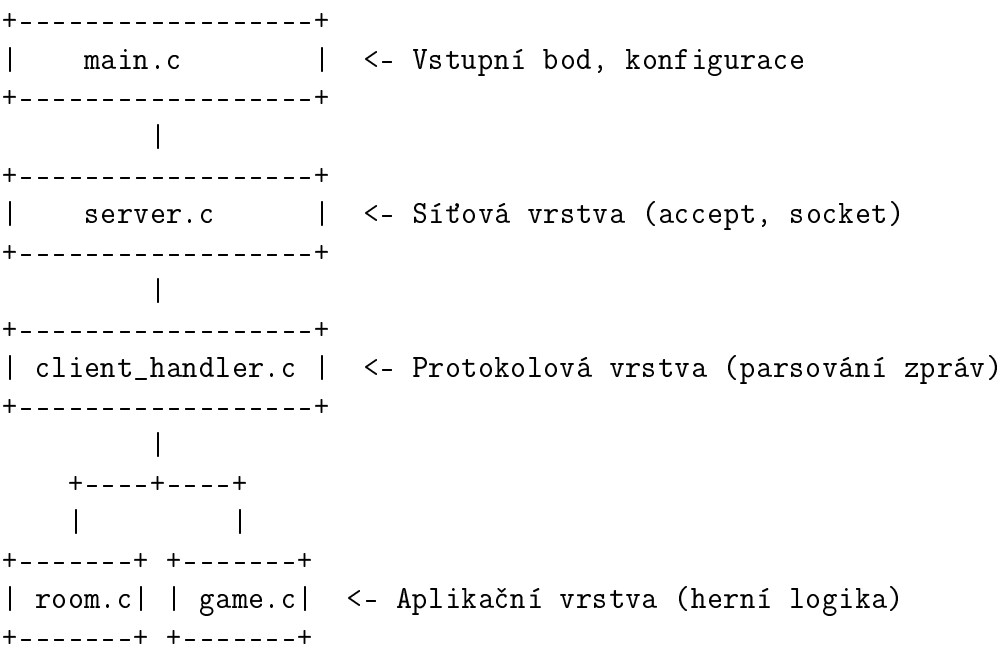
3.1 Dekompozice do modulů

Server je implementován v jazyce C s následující modulární strukturou:

Modul (soubor)	Odpovědnost
`main.c`	Vstupní bod aplikace, parsování argumentů příkazové řádky, inicializace a spuštění serveru
`server.c/.h`	Správa listening socketu, accept loop pro příchozí spojení, vytváření klientských threadů
`client_handler.c/.h`	Obsluha jednotlivých klientů, zpracování a parsování protokolových zpráv, state machine klienta
`client_list.c/.h`	Správa seznamu všech připojených klientů, vyhledávání podle ID/nickname
`room.c/.h`	Správa herních místností, lobby funkcionalita, broadcast zpráv hráčům v místnosti
`game.c/.h`	Herní logika Pexeso – inicializace desky, zpracování tahů, vyhodnocení shody, skóre
`logger.c/.h`	Thread-safe logování do souboru `server.log`
`protocol.h`	Definice protokolových konstant a příkazů

Tabulka 7: Moduly serveru a jejich odpovědnosti

3.2 Rozvrstvení aplikace



3.3 Metoda paralelizace – POSIX threads

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

- **Hlavní thread** – běží v `server_run()`, nekonečná accept loop čekající na nová spojení
- **Klientské thready** – pro každého nového klienta se vytvoří vlastní thread pomocí `pthread_create()`
- Thread se detachuje (`pthread_detach`) pro automatický cleanup po ukončení
- Každý thread má vlastní `client_t` strukturu s kontextem klienta

Synchronizace:

- Logger používá `pthread_mutex_t` pro thread-safe zápis do souboru
- Room broadcast operace jsou chráněny mutexem místnosti
- Seznam klientů je chráněn globálním mutexem

3.4 Použité knihovny

- **BSD Sockets (POSIX):** `sys/socket.h`, `netinet/in.h`, `arpa/inet.h` – `socket()`, `bind()`, `listen()`, `accept()`, `send()`, `recv()`
- **POSIX Threads:** `pthread.h` – `pthread_create()`, `pthread_detach()`, `pthread_mutex_t`
- **Standardní knihovna C:** `unistd.h`, `string.h`, `stdlib.h`, `stdio.h`, `time.h`

Žádné externí knihovny pro síťování nebo serializaci nebyly použity.

3.5 Konfigurace serveru

Server přijímá konfiguraci z příkazové řádky:

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

- `IP` – IP adresa pro bind (`0.0.0.0` pro všechna rozhraní)
- `PORT` – Port pro naslouchání (např. `10000`)
- `MAX_ROOMS` – Maximální počet herních místností
- `MAX_CLIENTS` – Celkový limit připojených klientů

4 Implementace klienta (Java + JavaFX)

4.1 Dekompozice do tříd

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

Třída/Balíček	Odpovědnost
`Main`	Vstupní bod aplikace, inicializace JavaFX, správa scén
`network/ClientConnection`	TCP spojení, asynchronní čtení v odděleném threadu, auto-reconnect logika
`network/MessageListener`	Rozhraní (interface) pro příjem zpráv a událostí
`protocol/ProtocolConstants`	Definice protokolových příkazů a timeoutů
`model/Room`	Datový model herní místnosti
`controller/LoginController`	Ovládání přihlašovací obrazovky
`controller/LobbyController`	Ovládání lobby (seznam místností)
`controller/GameController`	Ovládání herní obrazovky, zpracování herních zpráv
`util/Logger`	Thread-safe logování do `client.log`

Tabulka 8: Třídy klienta a jejich odpovědnosti

4.2 Rozvrstvení aplikace (MVC)

```
+---------------------------+
|     View (FXML + CSS)     |  <- UI definice
+---------------------------+
            |
+---------------------------+
|   Controller (JavaFX)     |  <- Logika UI, reakce na události
+---------------------------+
            |
+---------------------------+
|   Model + Network         |  <- Datové struktury, síťová komunikace
+---------------------------+
```

4.3 Síťová komunikace

Klient používá **blokující I/O** s odděleným vláknem pro čtení (reader thread):

```
// Reader thread - bezi nezavisle na GUI
private void readerLoop() {
    while (running && (line = in.readLine()) != null) {
        final String message = line.trim();
        if (isValidMessage(message) && isValidProtocolCommand(
           message)) {
            if (listener != null) {
                listener.onMessageReceived(message);
            }
        }
    }
}
```

GUI aktualizace probíhají přes `Platform.runLater()` pro thread-safety.

4.4 Použité knihovny

- **Java Standard Library:** `java.net.Socket`, `java.net.InetSocketAddress`, `java.io.BufferedReader`, `java.io.PrintWriter`
- **JavaFX 17:** `javafx.application.*`, `javafx.scene.*`, `javafx.fxml.*`, `javafx.stage.*`

Žádné externí knihovny pro síťování nebo serializaci nebyly použity.

4.5 Timeouty

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

Tabulka 9: Timeouty klienta

4.6 GUI – grafické uživatelské rozhraní

Klient implementuje grafické rozhraní pomocí JavaFX (ne konzole):

Login obrazovka – Zadání IP adresy/hostname, portu a přezdívky
Lobby obrazovka – Seznam místností, vytvoření nové, připojení k existující
Game obrazovka – Herní deska s kartami, skóre hráčů, indikace tahu

Klíčové vlastnosti GUI:

- **Non-blocking** – síťové operace v odděleném threadu, GUI nezamrzá
- **Aktuální stav** – zobrazuje hrací pole, přezdívky, kdo je na tahu
- **Indikace výpadků** – informuje o nedostupnosti serveru/protihráče
- **Validace vstupů** – kontrola IP, portu, přezdívky před odesláním

5 Požadavky na překlad a spuštění

5.1 Požadavky na prostředí

Server (C):

- Operační systém: GNU/Linux
- Kompilátor: GCC 7.5+ nebo Clang 6.0+
- Build systém: GNU Make 4.1+
- Knihovny: POSIX threads (libpthread) – součást systému

Klient (Java):

- Java SE 17+ (OpenJDK nebo Oracle JDK)
- Apache Maven 3.6+
- JavaFX 17+ (staženo automaticky přes Maven)
- Operační systém: GNU/Linux, MS Windows, macOS

5.2 Postup překladu serveru

```
cd server_src
make clean     # Vycistit predchozi build
make           # Zkompilovat server
```

5.3 Postup překladu klienta

```
cd client_src
mvn clean package     # Prelozit a zabalit do JAR
```

Maven vytvoří spustitelný JAR: `target/pexeso-client-1.0-SNAPSHOT.jar`

5.4 Spuštění aplikací

Server:

```
./server 0.0.0.0 10000 10 50
# IP=0.0.0.0, PORT=10000, MAX_ROOMS=10, MAX_CLIENTS=50
```

Klient:

```
java -jar target/pexeso-client-1.0-SNAPSHOT.jar
# nebo pres Maven:
mvn javafx:run
# nebo:
mvn exec:java
```

6 Závěr

6.1 Dosažené výsledky

Projekt splňuje všechny požadavky zadání:

- **Kompletní síťová hra** – 2–4 hráči, lobby systém, herní místnosti
- **Textový protokol** – ASCII přes TCP, čitelný a snadno 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, komentáře
- **Standardní build** – Makefile (server), Maven (klient)

6.2 Použité technologie (souhrn)

	Server	**Klient**
Jazyk	C	Java 17
Síťování	BSD sockets (POSIX)	java.net.Socket
Paralelizace	POSIX threads	Java threads
GUI	–	JavaFX 17
Build	GNU Make	Apache Maven

Tabulka 10: Přehled použitých technologií

6.3 Zhodnocení

Implementace prokázala schopnost navrhnout a realizovat kompletní síťovou aplikaci s robustním protokolem, automatickým zotavením po výpadcích a stabilním během. Aplikace byla otestována na scénářích zahrnujících: kompletní hru 2–4 hráčů, extrémní fragmentaci TCP zpráv, krátkodobé i dlouhodobé výpadky spojení, nevalidní/binární data a memory leaks (valgrind).

Klíčové poznatky:

- Důležitost správného bufferingu při fragmentaci TCP zpráv
- Nutnost thread-safe operací při paralelním běhu (mutexy)
- Význam keepalive mechanismu pro včasnou detekci výpadků
- Potřeba důkladné validace všech vstupů (bezpečnost)