Západočeská univerzita v Plzni
Fakulta aplikovaných věd

KIV/UPS

Semestrální práce

Síťová hra Žolíci

Autor: Matyáš Krejčí
Email: krejcim@gapps.zcu.cz
Předmět: KIV/UPS
Datum vytvoření: 12. 1. 2026
Repo: Github

Plzeň, 2026

Obsah

1 Zadání **3**
- 1.1 Návrh aplikace . . . 3
  - 1.1.1 Reconnect . . . 3
- 1.2 Poznámka k návrhu . . . 3

2 Protokol **4**
- 2.1 Formát zpráv . . . 4
- 2.2 Přenášené struktury, datové typy . . . 4
  - 2.2.1 **MAGIC** . . . 4
  - 2.2.2 **TYPE** . . . 4
  - 2.2.3 **LENGTH** . . . 5
  - 2.2.4 **MESSAGE** . . . 5
  - 2.2.5 **Packet** . . . 5
- 2.3 Výčet zpráv . . . 6
- 2.4 Stavový diagram protokolu . . . 7

3 Server **7**
- 3.1 Moduly . . . 7
  - 3.1.1 Makefile a CMakeLists.txt . . . 7
  - 3.1.2 client_manager (.c + .h) . . . 7
  - 3.1.3 config.h . . . 8
  - 3.1.4 game_manager (.c + .h) . . . 8
  - 3.1.5 logger (.c + .h) . . . 8
  - 3.1.6 main.c . . . 8
  - 3.1.7 protocol (.c + .h) . . . 8
  - 3.1.8 room_manager (.c + .h) . . . 8
  - 3.1.9 server_manager (.c + .h) . . . 8

4 Klient **8**
- 4.1 Moduly a třídy . . . 8
  - 4.1.1 card.py . . . 8
  - 4.1.2 clientgui.py . . . 8
  - 4.1.3 console.py . . . 8
  - 4.1.4 constants.py . . . 8
  - 4.1.5 gamestate.py . . . 9
  - 4.1.6 logger.py . . . 9
  - 4.1.7 main.py . . . 9
  - 4.1.8 message_handler.py . . . 9
  - 4.1.9 message_types.py . . . 9
  - 4.1.10 network.py . . . 9
  - 4.1.11 pages_drawer.py . . . 9
  - 4.1.12 ui_elements.py . . . 9
- 4.2 Paralelizace . . . 9
- 4.3 Verze . . . 9
  - 4.3.1 Serverová část (C) . . . 9
  - 4.3.2 Klientská část (Python) . . . 10
  - 4.3.3 Prostředí a nástroje . . . 10

5 Překlad a spuštění **10**
5.1 Klient . . . 10
5.1.1 Překlad . . . 10
5.1.2 Spuštění . . . 10
5.2 Server . . . 10
5.2.1 Překlad . . . 10
5.2.2 Spuštění . . . 11

6 Testování **11**
6.1 Test `nc` . . . 11
6.1.1 Klient . . . 11
6.1.2 Server . . . 11
6.2 Test `netstat` . . . 11
6.3 Test `InTCPtor` . . . 12
6.4 Test `dev/urandom` (server) . . . 12
6.5 Test `dev/urandom` (klient) . . . 12

7 Dosažené výsledky **12**

1 Zadání

Náplní této semestrální práce je vytvoření plně funkční síťové hry. Toto obnáší vytvoření herního serveru, který se bude starat o veškerou logiku, bude udržovat stavy hráčů, místností a her a uchovávat a spravovat informace o klientech (uživatelích). Pro grafické rozhraní je potřeba vytvořit klienta, který bude velice jednoduchou a snadno uchopitelnou grafikou zprostředkovávat uživateli informace jdoucí ze serveru. Aby bylo možné informace ze serveru přenášet po síti klientovi a naopak, je naprosto nezbytné mít definovaný komunikační protokol. Zprávy potom mohou být testovány na správnost, čímž bude možné vyhodnocovat případné chybové stavy jak na straně klienta, tak na straně serveru. Plné zadání semestrální práce, jednotlivé body zadání a specifikaci požadované funkcionality je možné nalézt na tomto odkazu.

1.1 Návrh aplikace

Uživatel se po spuštění aplikace dostane na přihlašovací obrazovku, kde musí zadat své jméno, adresu serveru a port, na kterým server naslouchá. Kliknutím na "připojit" odešle požadavek k připojení a server vrací zprávu, která je buď úspěšného nebo neúspěšného charakteru. Pokud se stal neúspěch, aplikace uživateli sdělí, o jakou chybu se jedná. Klientská aplikace sama kontroluje, zda-li uživatel zadal nevhodné údaje pro adresu serveru nebo číslo portu a dokud tato data nejsou korektní, žádný požadavek neodesílá. Server pro chybu může klientovi sdělit, že jméno, pod kterým chtěl vystupovat, je na serveru zabrané. Pokud ale klient odpoví úspěchem, klient je přivítán do hry a je mu zaslán unikátní token, kterým se bude prokazovat v případě krátkodobého výpadku sítě. Klient se po připojení nachází v lobby, kde má možnost vytvořit místnost nebo se připojit k již existující místnosti. Po připojení do místnosti se musí připravit a dokud není naplněna, čekat na protihráče. Pokud jsou oba hráči připraveni ke hře, uživatel, který místnost vytvořil nebo zdědil po předchozím majiteli může spustit hru. Hráči se potom dostávají do obrazovky s kartami, kde se střídají podle pravidel žolíků. Jakmile jeden hráč zavře, oba se přesouvají do obrazovky s výsledky a mají možnost hrát opakovaně, či hru opustit. Jakmile ale jeden odejde zpět do lobby, dostává se tam i druhý, který již nemá na co čekat.

1.1.1 Reconnect

Reconnect je v aplikaci řešen jak na straně serveru, tak na straně klienta pomocí heartbeat vlákna. Toto vlákno registruje příjem zpráv a obnovuje timestamp posledního kontaktu. Server navíc odesílá každé 3 vteřiny PING, který když klient přijme, odpoví PONG. Jakmile 10 sekund (tedy 3x PING/PONG) nepřijme jedna, či druhá strana jakoukoliv zprávu, odpojuje se. Pokud se v tuto situaci jedná o klienta, zahajuje vlákno reconnectu, tedy na základě exponencionálního timeoutu (počínaje t = 2, max=30) se snaží se serverem opět navázat spojení. Pakliže se spojení povede navázat, odesílá stejnou zprávu jako při prvotním připojení, ale obohacenou o token, který byl klientovi zaslán. Tímto se zabrání sebrání relace (session hijacking) a reconnect na straně serveru může proběhnout, jestliže je token správný. Při odpojení klient přepíná obrazovku na původní přihlašovací a server zasílá zprávu klientovi v místnosti, že se protihráč odhlásil → vykreslení obrazovky.

1.2 Poznámka k návrhu

Vzhledem k omezenému času ke kontrole projektu jsou některá pravidla klasických žolíků upravena, respektive zjednodušena.

1. Hráči nemusí dosáhnout určitého součtu vykládaných karet (defaultně 42).
2. Hráči mohou přikládat bez vyložených karet.
3. Hráči mohou brát vyhozené karty, kdykoliv se nějaká v balíčku nachází.
4. Hráči nemusí kartu z vyhozených karet hned použít a mohou si ji jednoduše ponechat.

2 Protokol

Aplikace běží na vlastním textovém protokolu nad transportním protokolem TCP. Textový protokol má fixní pravidla, která jsou kontrolována jak na serveru, tak na klientské straně při příjmu zprávy.

2.1 Formát zpráv

Jak již bylo zmíněno, každá zpráva má fixní formát. Obě strany tento formát očekávají, a pokud nastane jiná situace, vhodně na to reagují (tradičně se odpojí). Při příjmu je zpráva rozparsována a kontrolována po jednotlivých částech.

Formát zprávy:

MAGICTYPE_MESSAGEMESSAGE_LENGTHACTUAL_MESSAGE

Pro jednoduchost při kontrole a parsování je každá část zprávy (kromě `ACTUAL_MESSAGE`) dlouhá přesně 4 znaky. To vyúsťuje k tomu, že maximální délka zprávy může být maximálně 9999 znaků dlouhá (což na účely hry je více než dostačující – až spíše nepotřebné). Délka zprávy se kontroluje před odesláním a je zároveň kontrolována při příjmu, pokud by k odeslání skutečně došlo).

Část zprávy	Nabývající hodnoty	Popis
MAGIC	"JOKE"	Pevně definovaná hodnota na úplném začátku každé zprávy
TYPE_MESSAGE	LOGI, LOGO, OCRT, ...	Typy zpráv, které řídí aktivitu serveru a klienta
LENGTH_MESSAGE	0000	Celé číslo maximální velikosti 9999 udávající délku těla zprávy
ACTUAL_MESSAGE	"user"	Tělo zprávy – řetězec nesoucí informace z klienta nebo serveru

Tabulka 1: Popis struktury zpráv

Příklad: Při pokusu o připojení na server klient odesílá žádost o připojení zprávou s kódem `LOGI`, kde tělo zprávy obsahuje přezdívku, pod kterou chce uživatel vystupovat. Taková zpráva by potom vypadala takto:

JOKELOGI0008*uzivatel*

2.2 Přenášené struktury, datové typy

2.2.1 MAGIC

- ASCII řetězec
- Délka `4`
- Identifikace protokolu

2.2.2 TYPE

- Výčtový typ → ASCII řetězec
- Délka `4`
- Určuje typ zprávy

2.2.3 LENGTH

- Integer
- Rozsah $0000 - 9999$
- Informuje o délce zprávy

2.2.4 MESSAGE

- ASCII řetězec (ASCII / utf-8)
- Délka proměnná
- Aplikační data

2.2.5 Packet

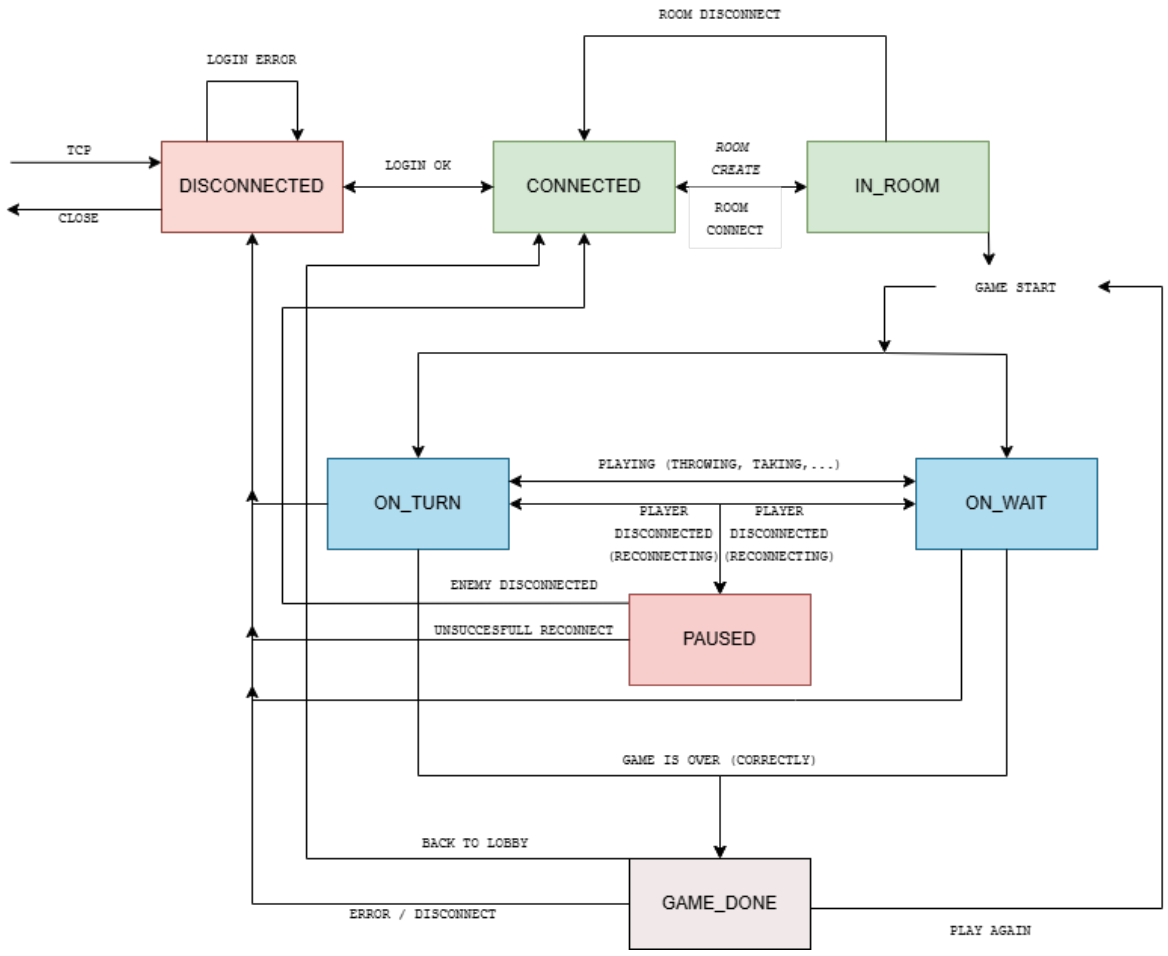
Všechny výše zmíněné části zprávy se potom poskládají do jednoho ASCII řetězce (build) a v tomto formátu se následně odesílají a na druhé straně jsou parsované, kontrolované, případně použity k dalším akcím.

2.3 Výčet zpráv

Typ zprávy	**Využívající strana**	Popis účelu
ADDC	K + S	Přidání karty do postupky
BOSS	K + S	Určení majitele místnosti
CLOS	K + S	Zavření poslední kartou
CRDS	K + S	Informace o kartách v ruce
CSEQ	K + S	Vytvoření postupky
ECNT	K + S	Chyba připojení k místnosti
ECRT	K + S	Chyba vytvoření místnosti
EDIS	K + S	Chyba odpojení z místnosti
EEDY	K + S	Chyba při pokusu o status “Připravený“ před hrou
ELIS	K + S	Chyba při pokusu o vylistování místností
ERRR	K + S	Obecná chyba využita na více místech
ESTR	K + S	Chyba při pokusu o start hry
GEND	K + S	Informace o skončení hry
LBBY	K + S	Inforamce o přesunu do lobby
LOGI	K + S	Klient odesílá požadavek o připojení
LOGO	K + S	Pokus o odhlášení
NOTI	K + S	Notifikace (obecná)
OCNT	K + S	Potvrzení o připojení k místnosti
OCRT	K + S	Potvrzení o vytvoření místnosti
ODIS	K + S	Potvrzení o odpojení z místnosti
OEDY	K + S	Potvrzení a změně stavu na “Připravený“ před hrou
OKAY	K + S	Potvrzení (obecné)
PAUS	K + S	Změna stavu hry na “pozastavená“
PING	K + S	Informace o aktivitě ze strany serveru
PLAG	K + S	Žádost/potvrzení o opakovaném hraní se stejným uživatelem
PONG	K + S	Informace o aktivitě ze strany klienta
PRDY	K + S	Žádost/odpověď s počtem připravených uživatelů
QUIT	K + S	Opuštění (obecné)
RCNT	K + S	Žádost o připojení k místnosti
RCRT	K + S	Žádost o vytvoření místnosti
RDIS	K + S	Žádost o opuštění místnosti
REDY	K + S	Žádost o změnu stavu na “připravený“
RINF	K + S	Žádost o informace o místnosti
RLIS	K + S	Žádost o vylistování místností
STAT	K + S	Informace o herním stavu (karty, karty protihráče, ...)
STRT	K + S	Žádost o start hry
TAKP	K + S	Žádost o kartu z balíčku
TAKT	K + S	Žádost o kartu z vyhozených karet
THRW	K + S	Žádost o vyhození karty
TURN	K + S	Změna aktuálního hráče na hrajícího
UNLO	K + S	Žádost o vyložení karet (postupka, set)
WAIT	K + S	Změna aktuálního hráče na čekajícího

Tabulka 2: Výčet zpráv

2.4 Stavový diagram protokolu



Obrázek 1: Stavový diagram

3 Server

3.1 Moduly

3.1.1 Makefile a CMakeLists.txt

Tyto soubory slouží pro sestavení serverové aplikace. `Makefile` umožňuje jednoduchou a rychlou kompilaci projektu pomocí nástroje `make`, `CMakeLists.txt` slouží k sestavení projektu pomocí `CMake` umožňuje generovat build pro různé překladače. V projektu byl použit především k debugování.

3.1.2 client_manager (.c + .h)

Soubor `client_manager` je hlavní soubor k obluze klienta. Udržuje pole struktur, které obsahují informace o jednotlivých klientech, validuje kroky klienta v jednotlivých stavech a na jejich základě odesílá zprávy. Zároveň rozhoduje o připojení klienta, respektive jeho odpojení.

3.1.3 config.h

Tento soubor by se dal nazvat konfiguračním souborem, definuje ty absolutně nejzákladnější konstanty.

3.1.4 game_manager (.c + .h)

Tento soubor obsluhuje hru – vytváří, maže, kontroluje, zda-li krok klienta byl validní či nikoliv. Na začátku inicializuje hru, určí hráče na "tahu" a hráče, který čeká, následně inicializuje balíček a rozdá karty (uchovává informace ve strukturách, `client_manager` potom rozesílá data uživatelům).

3.1.5 logger (.c + .h)

`logger` je modul, který slouží k jednoduchému zapsání stavu do loggovacího souboru. Umožňuje úrovně od nejnižší `DEBUG` po nejvyšší `FATAL`.

3.1.6 main.c

Vstupní soubor aplikace, nastavuje logger, inicializuje pole klientů, her a místností, spouští server.

3.1.7 protocol (.c + .h)

Obsahuje pole možných zpráv, má speciální metody na přijímání a odesílání zpráv – čte po částech, dokud nepřečte celou zprávu.

3.1.8 room_manager (.c + .h)

Hlavní správce místností. Vytváří místnost, čeká na uživatele, připravuje uživatele (do stavu "ready"), získává informace o místnosti, případně místnost maže, pakliže je prázdná.

3.1.9 server_manager (.c + .h)

Hlavní síťové nastavení. Nastavuje socket, adresu serveru, port na kterém naslouchá, přijímá a odmítá klienty, startuje hlavní klientské vlákno a vlákno timeout.

4 Klient

4.1 Moduly a třídy

4.1.1 card.py

Třída pro uchování objektu karty, definující kliknutený obdélník pro GUI, označování a registrování kliku na kartu.

4.1.2 clientgui.py

4.1.3 console.py

Třída definující herní konzoli, která zobrazuje některé zprávy serveru (negativní i pozitivní) a některé zprávy, které je schopen vyhodnotit sám klient.

4.1.4 constants.py

Soubor uchovávající nejdůležitější globální konstanty protokolu, velikosti herního okna, velikosti karet, cest k obrázkům a barvy používané pro grafické okno.

4.1.5 gamestate.py

Výčtový typ uchovávající možné stavy klientské strany.

4.1.6 logger.py

Stejná funkcionalita loggeru jako na straně serveru.

4.1.7 main.py

Vstupní bod aplikace, nastavuje logger a spouští klientskou aplikaci.

4.1.8 message_handler.py

Soubor obsahuje funkce, které spravují příjem a odesílání zpráv, kontrolu přijatých zpráv a kontrolu zpráv odcházejících z klienta (především aby nepřesahovaly maximální délku).

4.1.9 message_types.py

Výčtový typ všech zpráv, které využívá klient a server, které slouží k porovnávání přijatých zpráv a ke konstrukci zpráv k odeslání.

4.1.10 network.py

Hlavní síťová třída, která se stará o vlákno heartbeatu a vyhodnocování připojenosti k serveru, které předá do hlavního cyklu aplikace a ta potom spouští reconnect.

4.1.11 pages_drawer.py

Třída, která se stará o vykreslování jednotlivých obrazovek dle stavu hry.

4.1.12 ui_elements.py

Třída, která má na starost vykreslování tlačítek, vstupních polí a chybových hlášení.

4.2 Paralelizace

Server využívá vícevláknovou architekturu typu **Thread-per-Client**. Hlavní vlákno serveru v nekonečné smyčce přijímá nová spojení pomocí volání `accept()` a pro každého připojeného klienta vytváří samostatné obslužné vlákno (`client_handler`), které běží v režimu detached. Kromě klientských vláken v systému běží dedikované vlákno pro správu timeoutů (`timeout_checker_thread`), které periodicky kontroluje stav všech slotů.

4.3 Verze

4.3.1 Serverová část (C)

Serverová aplikace je implementována v nízkoúrovňovém jazyce C dle standardu `C11` a běží v prostředí operačního systému Linux. Pro paralelní zpracování klientů a síťovou komunikaci jsou využity následující knihovny:

- `pthread.h` – POSIX Threads pro paralelizaci a správu vláken.
- BSD Sockets – rozhraní pro síťovou komunikaci pomocí protokolu TCP/IP.
- Standardní knihovny jazyka C (`stdio.h`, `stdlib.h`, `string.h`, `unistd.h`, `time.h`, `ctype.h`).
- `arpa/inet.h`, `netinet/in.h` – práce s IP adresami a síťovými porty.

4.3.2 Klientská část (Python)

Klientská aplikace je napsána v jazyce Python a zajišťuje uživatelské rozhraní i komunikaci se serverem. Aplikace využívá následující knihovny:

- `pygame` – implementace grafického uživatelského rozhraní a herní logiky.
- `socket` – standardní knihovna pro TCP komunikaci se serverem.
- `threading` – běh síťové komunikace v samostatném vlákně.
- `queue` – bezpečná výměna zpráv mezi síťovým a vykreslovacím vláknem.

4.3.3 Prostředí a nástroje

Vývoj a testování aplikace probíhalo v prostředí operačního systému Linux (WSL). Pro sestavení a diagnostiku byly použity následující nástroje:

- `gcc` – překladač serverové části.
- `make` – nástroj pro sestavení projektu.
- `netstat`, `nc` – diagnostické nástroje pro testování síťové komunikace.

5 Překlad a spuštění

5.1 Klient

5.1.1 Překlad

Klientská aplikace je implementována v programovacím jazyce Python, který patří mezi interpretované jazyky. Z tohoto důvodu není nutné provádět překlad do strojového kódu a není potřeba žádná kompilace.

5.1.2 Spuštění

Spuštění klientské aplikace zajišťuje modul `main.py`. Aplikaci lze spustit z příkazové řádky následujícím příkazem:

```
python3 main.py
```

Po spuštění je uživatel vyzván k zadání základních konfiguračních údajů, konkrétně přezdívky, adresy serveru a čísla portu, na kterém server naslouchá.

5.2 Server

5.2.1 Překlad

Serverová aplikace je implementována v programovacím jazyce C, a proto je před jejím spuštěním nutné provést překlad zdrojových souborů. Součástí projektu je soubor `Makefile`, který automatizuje proces překladu všech potřebných zdrojových souborů do výsledného spustitelného souboru.
Překlad serveru se provede příkazem:

```
make
```

5.2.2 Spuštění

Po úspěšném překladu lze server spustit příkazem:

```
./zolik_server
```

Server se po spuštění začne naslouchat na předem definovaném portu a je připraven přijímat připojení klientských aplikací.

6 Testování

6.1 Test `nc`

6.1.1 Klient

```
nc -C localhost 10000
```

K serveru se lze připojit pomocí příkazu `nc localhost 10000` nebo místo `localhost` zvolit konkrétní adresu nastavenou v konfiguračním souboru. Server potom vyžaduje přihlášení a další akce přesně tak, jak jsou definované v uživatelském rozhraní. Posloupnost takových příkazů potom může vypadat následovně:

1. **JOKELOGI**0004*user* // přihlášení uživatele
2. **JOKERCRT**0008*roomName* // vytvoření místnosti roomName
3. **JOKEREDY**0001*1* // status připravenosti
4. **JOKESTRT**0000 // start hry

6.1.2 Server

```
nc -l -p 10000
```

Vytvoří server na portu 10000, na který se lze připojit klientem a testovat pomocí protokolových zpráv, které by odpovídal server.

Klient	Odpověď serveru
JOKELOGI0004user	JOKEOKAY0010tokentoken
JOKERCRT0004room	JOKEOCRT00010 + JOKEBOSS00011
JOKEREDY00011	JOKERINF0017user|READY|OWNER,
JOKESTRT00011	JOKESTRT0014Hra začíná!

Tabulka 3: Komunikace klient - server

6.2 Test `netstat`

```
watch -n 1 "netstat -a 10000 | grep tcp"
```

Server se v konzoli bude zobrazovat jako LISTEN, jakýkoliv připojený klient má status ESTABLISHED. Tímto způsobem lze diagnostikovat konektivitu aplikace a sledovat životní cyklus jednotlivých TCP spojení od jejich navázání až po ukončení

6.3 Test `InTCPtor`

Interceptor je knihovna, která zachytává volání funkcí BSD socketů souvisejících s protokolem TCP a zavádí do nich zpoždění, fragmentaci zpráv a další prvky. Byla vytvořena pro účely předmětu KIV/UPS na Západočeské univerzitě v Plzni a je šířena pod licencí MIT. Veškerou dokumentaci a konfiguraci lze najít na odkazu v první větě.

InTCPtor byl nad touto aplikací testován a aplikace je plně funkční.

6.4 Test `dev/urandom` (server)

```
cat /dev/urandom | nc 127.0.0.1 10000
```

6.5 Test `dev/urandom` (klient)

```
cat /dev/urandom | nc -l 127.0.0.1 -p 10000
```

7 Dosažené výsledky

Aplikace úspěšně navazuje spojení mezi serverem a klientem, server dokáže klienty paralelně obsluhovat, rozdělovat je do místností a zpřístupnit hru, kterou po celou dobu kontroluje (správné tahy apod.). Aplikace úspěšně reaguje na odpojení klienta/klientů, ponechává si krátkodobě jejich informace a pokud se stihnout připojit do časového limitu [2 minuty], napojí je zpět do hry. Strana serveru i klienta kontroluje vstupní data uživatele. Server i klient rozhoduje o správnosti protokolových zpráv a vhodně reaguje na porušení protokolu. Taktéž obě strany čtou zprávy, dokud nejsou přečtené celé. Co se týče pravidel hry, byly zjednodušeny pro účely aplikace. Hráči nejsou povinni dosáhnout určitého součtu vykládaných karet při prvním vyložení (standardně 42). Hráči mohou přikládat bez vyložených karet. Vyhozené karty lze brát i dříve než po uplynutí třech kol a karty si hráči mohou nechávat (nemusí ji ihned použít k vyložení). Aplikace zároveň byla testována hned několika nástroji za účelem dosažení maximální funkcionality, která je v moment odevzdávání plně potvrzena.