Dokumentace ke hře Inverzní lodě

Autor: Milan Vlachovský

Obsah

1. Úvod

Cílem tohoto projektu bylo vytvořit elementární hru pro více hráčů používající síťovou komunikaci se serverovou částí v nízkoúrovňovém programovacím jazyce a klientskou částí v programovacím libovolném jazyce. Projekt vznikal jako univerzitní projekt. Autor projektu zvolil vlastní variantu hry Lodě s upravenými pravidly nazvanou *„Inverse Battleships“*. Stejně jako její předloha je hra určena pro 2 hráče, přičemž se hráčí střídají v tazích.

Byl zvolen protokol na bázi TCP. Jako programovací jazyk serveru byl zvolen jazyk Go, díky své rychlosti a nízkoúrovňovému přístupu k síťové komunikaci. Klientská část byla implementována v jazyce Python s využitím knihovny pygame pro správu vykreslování grafického prostředí hry.

2. Popis hry

Hra *Inverse Battleships* je variantou klasické hry Lodě, ve které se hráči snaží najít a zničit všechny lodě protivníka. V této variantě hráči sdílí jedno pole 9x9 a každý dostane na začátku přiřazenou jednu loď. Následně se hráči střídají v tazích, kdy každý hráč se může pokusit vykonat akci na prázdné políčko. Mohou nastat tři situace:

- Hráč zkusí akci na prázdné políčko, ve kterém se nachází nikým nezískaná loď. V tomto případě hráč loď získává a získává body.
- Hráč zkusí akci na prázdné políčko, ve kterém se nachází protivníkova loď. V tomto případě protivník o loď přichází, je zničena, hráč získává body a protivník ztrácí body.
- Hráč zkusí akci na políčko, na kterém se nic nenachází. V tomto případě hráč nezískává nic.

Hra končí, když jeden z hráčů ztratí všechny lodě. Vítězem je přeživší hráč. Body jsou pouze pro statistické účely a nemají vliv na průběh hry.

3. Popis síťového protokolu

Posílané zprávy jsou v plaintext nešifrovaném formátu. Každá zpráva je ukončena znakem nového řádku (ASCII 10). Každá zpráva musí obsahovat předem nadefinovanou hlavičku *"IBGAME"*. Následuje druh rozkazu a poté případné parametry. Každá část zprávy je rozdělena znakem ';' (případné další specifičtější oddělovače pro druh rozkazu jsou uvedeny v definici protokolu). Při nutnosti poslání oddělovače ';' přímo v parametru se použije escape sekvence pomocí znaku '\'.

Všechny přijaté zprávy jsou validovány na základě protokolu. Pokud zpráva od klienta neodpovídá protokolu, je ignorována a klient je odpojen. Pokud odpověď od serveru není validní, klient ukončí spojení se serverem a pokusí se znovu připojit. Při validním rozkazu dochází k parsování zprávy do požadovaného formátu (podle definice protokolu; např. rozkaz LOBBIES od serveru je parsován do seznamu řetězců, rozkaz ACTION od klienta je parsován do dvou celých čísel apod.), pokud parsování selže, je zpráva ignorována a klient je odpojen.

Forma	Parametry	Posílá	Vysvětlení	Část programu
HAND	<nickname>	Client	Pokus o navázání spojení	Zahájení připojení
SHAKE		Server	Odpověď od serveru	
DEAL		Client	Potvrzení od klienta	
LOBBIES		Client	Žádost o získání lobbies	Párování hráčů
LOBBIES	<id_lobby>,<id_lobby>,...	Server	Odpověď od serveru	
BRING_IT	<id_lobby>	Client	Žádost o připojení k lobby	
CREATE		Client	Žádost o založení lobby	
PAIRING	<id_lobby>	Server	Potvrzení žádosti o založení lobby	
PAIRED	<protihráčovo_jméno>	Server	Oznámení o spárování	
READY		Client	Klient připraven na inicializaci hry	
TURN	<id_hráče>	Server	Určení soupeře na tahu	Hra
BOARD	0:0:0:1:...,1:0...	Server	Zaslání stavu herního pole	
ACTION	<x>:<y>	Client	Oznámení o tahu	
WIN		Server	Oznámení výhry klienta	
LOST		Server	Oznámení prohry klienta	
LEAVE		Client	Klient se odpojuje	Odpojení
BYE		Server	Odpojení potvrzeno	
WAIT		Server	Žádost o čekání na druhého hráče	Čekání na protihráče
WAITING		Client	Potvrzení čekání	
CONTINUE	<id_lobby>;<opponent>,<player_on_turn>,0:0:0:1:...,1:0...	Server	Žádost o pokračování hry	
TKO		Server	Hra skončila chybou	Vyjímečný případ
PING		Oba		Keep Alive mechanismy
PONG		Oba		

Delimiter zpráv	Delimiter parametrů
\n	;
Header	Delimiter čísel (ACTION, BOARD, ...)
IBGAME	:
	Delimiter sekvencí (LOBBBIES, BOARD, ...)
	,

Příklad zpráv:
IBGAME;HAND;Player1\n
IBGAME;SHAKE\n
IBGAME;ACTION;5;10\n
IBGAME;PAIRED;\;)__M4st3r-_-;1\n
IBGAME;TURN;Player_01\n

Tabulka s definicí protokolu

Následující diagram znázorňuje zjednodušenou komunikaci mezi klientem a serverem.

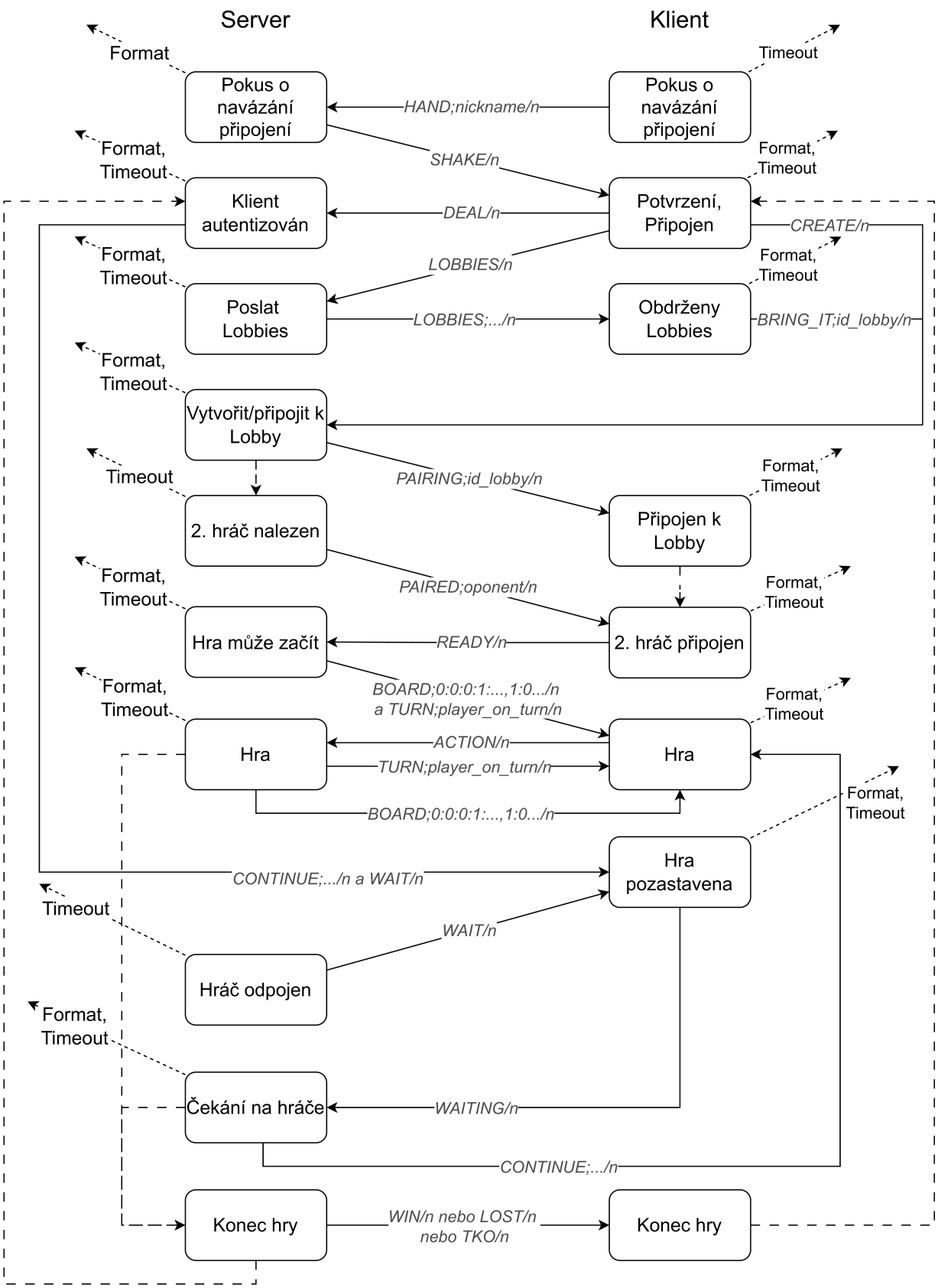

Diagram zjednoduššené komunikace mezi klientem a serverem

Zprávy jsou odlišeny šedivou barvou a kurzívou. Zprávy od klienta posouvají server do dalších stavů a naopak. Samovolné přechody (způsobené vnějšími událostmi typu nalezení soupeře, návrat soupeře, návrat z ukončeného zápasu, ...) jsou zobrazeny čárkovanou čarou. U téměř každého stavu je výstupní hrana označující selhání validace/odpojení/timeout klienta.

4. Architektura systému

Systém je rozdělen na dvě části: serverovou a klientskou. Serverová část je napsána v jazyce Go a klientská část v jazyce Python s využitím knihovny *pygame*. Serverová část je zodpovědná za správu hry, komunikaci s klienty a validaci zpráv. Klientská část je zodpovědná za zobrazení grafického rozhraní, zpracování vstupů od uživatele a komunikaci se serverem.

V obou částech je síťová komunikace zajištěna na nízké úrovni pomocí socketů (v Pythonu pomocí knihovny *socket* a v Go pomocí knihovny *net*). Jedná se o tahovou hru, kde se hráči střídají v tazích. Proto byl jako protokol zvolen TCP, který je vhodný pro tento typ hry.

Požadavky

Projekt byl vyvíjen s použitím následujících technologií:

- Python 3.12
  - pygame 2.6.0
  - pydantic 2.8.2
  - typing-extensions 4.12.2
  - termcolor 2.5.0
  - pyinstaller 6.11.1
- Go 1.23

> Za použití zmiňovaných technologií by měl být projekt bez problémů spustitelný. Spouštění na starších verzích nebylo testováno a nemusí fungovat správně.

5. Návod na zprovoznění

Pro sestavení celého projektu byly vytvořeny soubory *Makefile* a *Makefile.win*, které obsahují instrukce pro sestavení projektu na Unixových a Windows OS. Pro sestavení projektu na Unixových OS stačí spustit příkaz:

```
make
```

a pro Windows OS stačí spustit příkaz:

```
make -f Makefile.win
```

Předpokládá se, že je nainstalován program `make`; na Windows je možné použít například make z chocolatey, či jiné alternativy.

Skript sestaví spustitelné soubory ve složce *client/bin/* pro klientskou část projektu a ve složce *server/bin/* pro serverovou část projektu. Spustitelné soubory jsou pojmenovány *client* a *server*, případně na Windows *client.exe* a *server.exe*.

> Jelikož je klientská část implementována v jazyce Python, je možné ji spustit i bez sestavení. Stačí spustit soubor *client/src/main.py* v Python virtuálním prostředí s nainstalovanými závislostmi ze souboru *requirements.txt*. Spustitelné soubory pro klientskou část byly vytvořeny pomocí knihovny *pyinstaller* a jejich úspěšnost překladu bývá závislá na operačním systému a verzi Pythonu.

> Na základě standardu PEP 394 počítají soubory *Makefile* a *Makefile.win* s tím, že Python rozkaz pod Unixem je `python3` a pod Windows je `python`. V případě odlišného nastavení je nutné soubory upravit.

5.1 Klientská část

Klientská část je napsaná v jazyce Python, tedy kód je standardně interpretován řádku po řádce pomocí Python interpreteru. Pro spuštění klientské části je tedy nutné mít nainstalovaný Python 3.12 a nainstalované závislosti ze souboru *requirements.txt*.

Spuštění krok za krokem

Autor dopořučuje vytvořit si virtuální prostředí a nainstalovat závislosti pomocí následujících příkazů:

> Příkazy jsou pro Unix OS, pro Windows OS je nutné je přizpůsobit.

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
cd client/
```

Následně je možné spustit klienta pomocí příkazu:

```
python ./src/main.py
```

Je také možné spustit klienta se specifickým nastavením a nastavením logování pomocí:

```
python ./src/main.py -c ./cfg/debug_cfg.json -l
./cfg/debug_loggers_cfg.json
```

> Standardně se předpokládá spouštění ze složky *client/*

Sestavení spustitelného souboru

Kvůli charakteru jazyka není možné bez externích knihoven vytvořit spustitelný soubor. Autor proto zvolil cestu sestavení pomocí knihovny *pyinstaller*.

Pro sestavení spustitelného souboru stačí pouze spustit z kořenové složky na Unix OS příkaz:

```
make client
```

Nebo na Windows OS:

```
make -f Makefile.win client
```

Pro manuální sestavení lze použít kód z Makefile souborů.

5.2 Serverová část

Serverová část je napsaná v jazyce Go. Pro sestavení serverové části je nutné mít nainstalovaný Go 1.23 či novější. Sestavení opět probíhá za pomocí souborů *Makefile* a *Makefile.win*. Na Unix OS stačí spustit příkaz:

```
make server
```

Na Windows OS:

```
make -f Makefile.win server
```

Make sestaví spustilený soubor ve složce *server/bin/* s názvem *server*.

Spuštění serveru

Pro spuštění serveru je nutné zadat jako parametr buď IP adresu, na které bude server naslouchat (parametr *-a*), nebo specifikovat konfigurační soubor (parametr *-c*). Pro spuštění serveru na 127.0.0.1 a na portu 8080 stačí zadat:

> Příkazy jsou pro Unix OS, pro Windows OS je nutné je přizpůsobit.

```
./server/bin/server -a "127.0.0.1:8080"
```

> Při volbě IP adresy a portu je vhodné zjistit, zda je adresa a port dostupný a nejsou blokovány firewallem apod.

6. Struktura projektu

Projekt je rozdělen následovně:

- *Kořenová složka* — Obsahuje soubory pro sestavení projektu, složky *client* a *server*, složku *docs* s dokumentací a soubor *requirements.txt* s definicemi Python závislostí.
    - *client/* — Obsahuje celou klientskou část projektu včetně konfiguračních souborů, použitých textových a obrazových zdrojů a programátorské referenční dokumentace.
    - *server/* — Obsahuje celou serverovou část projektu včetně konfiguračních souborů a programátorské referenční dokumentace.

Kořenová složka

- *Makefile* — Soubor pro sestavení projektu na Unix OS.
- *Makefile.win* — Soubor pro sestavení projektu na Windows OS.
- *client/* — Složka obsahující kód s klientskou částí aplikace.
    - *client/Doxyfile* — Soubor s konfigurací pro Doxygen.
    - *client/cfg/* — Složka obsahující konfigurační soubory klientské části.
        - *client/cfg/debug_cfg.json* — Soubor s konfigurací pro debugování.
        - *client/cfg/debug_loggers_cfg.json* — Soubo s konfigurací logování pro debugování.
        - *client/cfg/debug_loggers_cfg_win.json* — Soubor s konfigurací logování pro debugování na Windows.
        - *client/cfg/default_config.json* — Soubor s výchozí konfigurací.
        - *client/cfg/default_user_config.json* — Soubor s výchozí konfigurací nového uživatele.
        - *client/cfg/loggers_config.json* — Soubor s konfigurací logování.
        - *client/cfg/users/* — Složka obsahující konfigurace uživatelů.
    - *client/docs/* — Složka obsahující dokumentaci kódu klientské části.
    - *client/res/* — Složka obsahující zdroje pro klientskou část.
        - *client/res/colors.json* — Soubor s definicemi barev použitých v GUI klienta.
        - *client/res/img/* — Složka obsahující obrázky použité ve GUI klienta.
        - *client/res/strings.json* — Soubor s definicemi textoých řetězců použitých v GUI klienta.

- *client/src/* — Složka obsahující zdrojové kódy klientské části.
  - *client/src/const/* — Složka obsahující konstanty.
    - *client/src/const/exit_codes.py* — Soubor s konstantami pro návratové kódy.
    - *client/src/const/loggers.py* — Soubor s konstantami pro logování.
    - *client/src/const/paths.py* — Soubor s cestovými konstantami.
    - *client/src/const/typedefs.py* — Soubo s definicemi požívaných objektů v kódu klienta.
  - *client/src/game/* — Složka zastřešující kód pro správu hry.
    - *client/src/game/connection_manager.py* — Soubor s kódem pro správu spojení se serverem.
    - *client/src/game/ib_game.py* — Soubor s kódem spravujícím hru.
    - *client/src/game/ib_game_state.py* — Soubor s kódem pro stav hry.
  - *client/src/graphics/* — Složka obsahující kód GUI klienta.
    - *client/src/graphics/game_session.py* — Soubor s kódem pro GUI herní session.
    - *client/src/graphics/menus/* — Složka obsahující kód GUI menu.
      - *client/src/graphics/menus/info_screen.py* — Soubor s kódem GUI informační obrazovku.
      - *client/src/graphics/menus/input_menu.py* — Soubor s kódem GUI vstupní menu.
      - *client/src/graphics/menus/lobby_select.py* — Soubor s kódem GUI výběr lobby.
      - *client/src/graphics/menus/primitives.py* — Soubor s kódem pro primitiva GUI.
      - *client/src/graphics/menus/select_menu.py* — Soubor s kódem pro výběrové menu.
      - *client/src/graphics/menus/settings_menu.py* — Soubor s kódem pro nastavení.
    - *client/src/graphics/viewport.py* — Soubor s kódem představujícím viewport pro zobrazování libovolného GUI.
  - *client/src/main.py* — Soubor s kódem pro spuštění klienta.

        - *client/src/util/* — Složka obsahující pomocné metody.
            - *client/src/util/assets_loader.py* — Soubor s kódem pro načítání zdrojů (obrázky, zvuky, ...).
            - *client/src/util/etc.py* — Soubor s vedlejšími pomocnými metodami.
            - *client/src/util/file.py* — Soubor s pomocnými metodami pro práci se soubory.
            - *client/src/util/generic_client.py* — Soubor s kódem představující generického klienta (založeném na socketech).
            - *client/src/util/graphics.py* — Soubor s pomocnými metodami pro práci s grafikou.
            - *client/src/util/init_setup.py* — Soubor s kódem pro inicializaci klienta.
            - *client/src/util/input_validators.py* — Soubor s validátory vstupu.
            - *client/src/util/loggers.py* — Soubor s kódem pro příspůsobené logování.
            - *client/src/util/path.py* — Soubor s pomocnými metodami pro práci s cestami.
- *docs/* — Složka obsahující dokumentaci.
    - *docs/doc.md* a *docs/doc.pdf* — Tento dokument ve formátu Markdown a PDF.
    - *docs/client_ref.html* — Odkaz na dokumentaci klientské části.
    - *docs/server_ref.html* — Odkaz na dokumentaci serverové části.
- *requirements.txt* — Soubor s definicemi Python závislostí.
- *server/* — Složka obsahující kód s serverovou částí aplikace.
    - *server/cfg/* — Složka obsahující konfigurační soubory serverové části.
    - *server/docs/* — Složka obsahující dokumentaci kódu serverové části.
    - *server/src/* — Složka obsahující zdrojové kódy serverové části.
        - *server/src/const/* — Složka obsahující konstanty.
            - *server/src/const/const_file/const_file.go* — Soubor s konstantami pro práci se soubory.
            - *server/src/const/custom_errors/custom_errors.go* — Soubor s definicemi chyb.
            - *server/src/const/exit_codes/exit_codes.go* — Soubor s konstantami pro návratové kódy.
            - *server/src/const/msg/msg.go* — Soubor s definicemi uživatelských zpráv.
            - *server/src/const/protocol/server_communication.go* — Soubor s definicemi síťového protokolu.
        - *server/src/go.mod* — Soubor s definicí modulů Go.
        - *server/src/logging/logging.go* — Soubor s kódem pro logování.
        - *server/src/main.go* — Soubor s kódem pro spuštění serveru.

- *server/src/server/* — Složka obsahující kód pro správu serveru.
  - *server/src/server/connection_manager.go* — Soubor s kódem pro správu spojení.
  - *server/src/server/client_manager.go* — Soubor s kódem pro správu klientů.
- *server/src/util/* — Složka obsahující pomocné funkce.
  - *server/src/util/arg_parser/arg_parser.go* — Soubor s kódem pro parsování argumentů.
  - *server/src/util/cmd_validator/cmd_validator.go* — Soubor s kódem pro validaci síťových příkazů.
  - *server/src/util/msg_parser/msg_parser.go* — Soubor s kódem pro parsování zpráv.
  - *server/src/util/util.go* — Soubor s pomocnými funkcemi.

7. Popis implementace

Do popisu implementace budou převážně zahrnuty jen ty nejdůležitější části kódu potřebné pro pochopení principu fungování aplikace. Pro podrobnější informace je možné využít programátorskou referenční dokumentaci, která je dostupná v *docs/client_ref.html* a *docs/server_ref.html* (odkazují do *client/docs/* a *server/docs/*).

> Referenční dokumentace jsou dostupné pouze v angličtině.

> **Poznámka k pojmenování/zapouzdření:**
> Na několika místech Python klient používá dvojité počáteční podtržítko (např. `__example`) k napodobení zapouzdření pomocí name-manglingu. V Pythonu je však vhodnější a běžnější používat jedno počáteční podtržítko (např. `_example`) k označení interních členů.
> Kvůli srozumitelnosti, udržovatelnosti a souladu s konvencemi jazyka Python by **veškerý budoucí vývoj nebo odvozené práce měly nahradit dvojitá podtržítka jednoduchými**. Tato volba pojmenování nemá vliv na stávající funkčnost, a proto byla v tomto projektu ponechána beze změny.

7.1 Klientská část

Klientská část aplikace byla implementována v jazyce Python s využití knihovny **pygame** pro grafické rozhraní. Hlavní součásti klienta jsou rozděleny do modulů podle jejich funkce.

> Veškerý kód se nachází pod složkou *client/src/*.

Při běhu programu pracuje vždy jedno hlavní vlákno, které se stará o logiku hry, zpracovávání vstupů a vykreslování grafického rozhraní. Pro správu asynchronní komunikace se serverem je vždy vytvořeno vlastní vlákno, které pomocí synchronizačních přístupů (Python *threading.Lock*, *threading.Event*, ...) zpracovává zprávy od serveru a zasílá zpět odpovědi.

Při každé změně stavu hry je spuštěno nové vlákno pro zpracovávání serverové komunikace odpovídající stavu hry. Pokud dostane klient neočekávanou zprávu od serveru (rozbitou, nevalidní, nesprávnou na základě stavu hry, apod.) dojde k odpojení od serveru a vypsání chybové hlášky.

Vstupní bod programu je soubor *main.py* a metoda `main()`. Tato metoda inicializuje klientskou aplikaci, načte konfiguraci, nahraje zdroje, vytvoří instanci třídy *IBGame* z *ib_game.py* a spustí hlavní smyčku hry.

Moduly klientské části

1. **game** — Hlavní modul, který zastřešuje veškerou logiku hry.
    - *connection_manager.py*
        - Spravuje připojení klienta k serveru pomocí socketů (používáním *generic_client.py*).
        - Poskytuje API pro odesílání a příjem zpráv, které využívá *ib_game.py*.
    - *ib_game.py*
        - Spravuje celou logiku hry, jako je zpracování tahů a synchronizace herního stavu se serverem.
        - Zde dochází ke správě grafického rozhraní a vstupů od uživatele.
        - Drží informace o stavu hry a komunikuje se serverem pomocí *connection_manager.py* (metody, které síťové vlákna vykonávají mají prefix `__handle_net`).
        - Obecně funguje na principu stavového automatu, kde každý stav odpovídá určité fázi hry.
            - Vždy dochází k volání metody `update()` z *main.py*, která dále volá podmetody podle aktuálního stavu hry (`__update_main_menu()`, `__update_game_session()`, ...).
            - V každé aktualizační podmetodě dochází k inicializaci grafického kontextu, případně vytvoření nového vlákna pro komunikaci se serverem (metody s prefixem `__prepare`), ke zpracování vstupů od uživatele (metoda `__proccess_input(events)`), k aktualizacím grafického rozhraní (metoda `update()` grafického kontextu), k vykreslení těchto změn (metoda `draw()` či `redraw()` grafického kontextu) a k případným aktualizacím stavu hry na základě odezvy od grafického rozhraní nebo serveru (metody s prefixem `__handle_update_feedback`).
            - Četnost volání aktualizačních metod je závislá na *tick_speed* (v případě tohoto projektu srovnatelné se snímky za vteřinu) nastavené v konfiguračním souboru pomocí volání metody `clock.tick(tick_speed)` v *main.py* hlavní smyčce.
            - Při přechodech mezi stavy hry vždy dochází k ukončování dříve spuštěných vláken.
    - *ib_game_state.py*.
        - Obsahuje třídu representující stav hry.

2. **graphics** — Modul pro vykreslování grafického rozhraní
    - *viewport.py*
        - Obsahuje třídu *Viewport*, kterou musí dědit všechny třídy, které chtějí vykreslovat do okna.
        - Třída určuje kontrakt pro vykreslování a aktualizaci, který musí být dodržen.
        - Každý potomek třídy *Viewport* musí implementovat metody:
            - `redraw()`: Pro překreslení celého obsahu okna.
            - `draw()`: Pro překreslení změněného obsahu okna.
            - `update(events)`: Pro aktualizaci obsahu okna na základě změn (např. vstupů od uživatele, zpráv od serveru, ...).
    - *game_session.py*
        - Zajišťuje vykreslování herního prostředí a interakci uživatele s herní relací.
    - *menus*
        - Obsahuje moduly pro tvorbu a správu herních menu, jako jsou vstupní obrazovky, lobby nebo nastavení.
3. **util** — Modul pro pomocné funkce
    - *assets_loader.py*: Zajišťuje načítání grafických a dalších zdrojů.
    - *generic_client.py*: Představuje generického klienta pro komunikaci se serverem používajícího socketovou komunikaci.
    - *loggers.py*: Implementuje vlastní logování pro snadněší diagnostiku a ladění.
    - *init_setup.py*: Zajišťuje inicializaci klientské aplikace.

Použité knihovny klientské části

- **pygame 2.6.0**: Pro vykreslování grafického rozhraní.
- **pydantic 2.8.2**: Pro validaci dat.
- **termcolor 2.5.0**: Pro barevné logování v terminálu.
- **pyinstaller 6.11.1**: Pro sestavení spustitelných souborů.

Paralelizace klienta

Klientská část využívá knihovny *threading* pro správu asynchronní komunikace se serverem, což umožňuje zpracovávat zprávy od serveru souběžně s vykreslováním a zpracováním vstupů uživatele. Při inicializaci každého stavu hry dochází k připravení grafického kontextu a k případnému vytvoření a spuštění vlákna pro komunikaci se serverem. Vlákno zpracovává zprávy od serveru a případně propaguje změny do hlavního vlákna:

- pomocí použití *threading.Lock* zámku (`self.net_lock`) a:
    - úpravou stavové proměnné reprezentující stav hry (`self.game_state`),
    - úpravou speciální přoměnné pro správu herní relace (`self.__game_session_updates`);
- pomocí smazání reference na grafický kontext, což vynutí inicializační metodu v dalším cyklu, která vykoná přípravu nového grafického kontextu, spuštění nového vlákna, ...

Komunikace z hlavního vlákna do síťových vláken probíhá pomocí:

- *threading.Event* (`self.__end_net_handler_thread`, `self.do_exit`), které signalizuje ukončení vlákna,
- *queue.Queue* (`self.__action_input_queue`), která slouží k předávání zpráv z herní relace do síťového vlákna.

7.2 Serverová část

Serverová část byla implementována v jazyce Go pro svou rychlost a efektivní práci s paralelizací. Byly využity pouze základní knihovny Go (pro síťovou komunikaci balíček *net*). Serverová část je rozdělena do modulů podle jejich funkcí.

> Veškerý kód se nachází pod složkou *server/src/*.

Vstupní bod serveru je soubor *main.go*, který načte konfiguraci, vytvoří instanci serveru a manažera klientů a spustí hlavní smyčku herního serveru (funkce `manageServer()`). Serverová část je rozdělena do tří hlavních modulů: *server*, *util* a *const*.

Moduly serverové části

1. **server** — Hlavní modul, který zastřešuje veškerou logiku serveru.
    - *connection_manager.go*
        - Představuje jedno spojení mezi serverem a klientem.
        - Definuje strukturu serveru, který je representován IP adresou a *net.Listenerem*.
        - Obsahuje metody pro přijímání nových přípojení, zpracování jejich zpráv a odesílání zpráv.

- *client_manager.go*
  - Spravuje všechny klienty připojené k serveru a logiku spojenou s během herního serveru.
  - Hlavní funkce je `ManageServer()`, která obsahuje hlavní smyčku serveru, která běží na hlavním vlákně.
  - Naslouchá na SIGINT a SIGTERM signály pro ukončení serveru.
  - Používá sdílené struktury (mapy) pro správu ne/ověřených klientů a lobby.

> Při každém přístupu ke sdíleným strukturám (např. klienti, lobby, ...) využívá client_manager *rwmutexy* pro zajištění bezpečného synchronního přístupu.

  - Hlavní smyčka serveru vypadá následovně:
    1. Kontroluje kontrolní proměnné pro ukončení serveru.
    2. Zavolá se funkce, která spravuje všechny aktivní lobby, které jsou hostované na serveru (funkce `ManageLobbies()`).
       - Tato funkce zajišťuje správu herních relací. Na základě stavu každého lobby ho roztřídí do front čekajících na odbavení (`lobbiesToStart`, `lobbiesToAdvance`, `lobbiesToDelete`, ...).
       - Všechny lobby se postupně pokusí odbavit a posunout do dalšího stavu, je-li to možné (funkce s prefixem `manageLobbies`).
       - Při nutnosti poslání informační zprávy klientům používá *send* zámek, který je pro každý klient definován (případně posílá zprávy paralelně pomocí Gorutin, které pouze odesílají zprávy a hned končí).
    3. Kontroluje, zda se nový klient nepokouší připojit k serveru. Pokud ano, vytvoří nové spojení a novou Gorutinu (vlákno), která bude neustále až do odpojení zpracovávat interakce od klienta.

> Každý nový klient má *recv* a *send* mutexy, které zajišťují bezpečný výlučný přístup k operacím prováděným nad sockety.

       - Nejprve se pokusí klienta ověřit a přihlásit (na základě kontraktu protokolu). Při neúspěchu klienta odpojí a odebere ze sdílených struktur.
       - Poté zpracovává zprávy od klienta a vykonává příslušné akce (např. vytvoření nové herní relace, připojení k existující herní relaci, příprava na změnu stavu lobby na základě příkazu od klienta, ...). Jedná se o funkce s prefixem *handle* (`handlePingCmd`, `handleJoinLobbyCmd`, ...). Při neplatném či nepovoleném rozkazu klienta odpojí, odebere ze sdílených struktur, případně označí klientovo lobby k odstranění.

2. **util** — Modul pro pomocné funkce.
    - *arg_parser.go*: Zajišťuje parsování argumentů při spouštění serveru.
    - *cmd_validator.go*: Validuje příkazy přijaté od klientů.
    - *msg_parser.go*: Zajišťuje správné formátování zpráv při odesílání a příjmu.
3. **const** — Modul pro konstanty.
    - *protocol/server_communication.go*: Obsahuje definice protokolu a formátu zpráv mezi klientem a serverem.
4. **logging** — Modul pro logování.
    - *logging.go*: Implementuje logování pro snadnější diagnostiku a ladění.

Použité knihovny serverové části

- Standardní knihovny Go: *net*, *os*, *fmt*, *sync*, *time*, ...

Paralelizace

Server využívá gorutiny pro souběžné zpracování klientských požadavků. Pro synchronizaci dat jsou používány mutexy a kanály (*channels*). Každý nový klient má své vlastní mutexy pro zajištění bezpečného přístupu k socketům. Server využívá *sync.RWMutex* pro zajištění bezpečného přístupu k sdíleným strukturám (mapy klientů, lobby, ...).

7.3 Detaily implementace

Síťová komunikace

Protože posílání a příjem zpráv přes BSD sockety je ovlivněn implementací, kterou používá OS, bylo potřeba řešit situace, kdy mohlo přijít více zpráv najednou, nebo kdy byla zpráva rozdělena na více částí. Jak klient, tak server si pro každé připojení drží buffer, do kterého umí ukládat přebytečná data a zpracovávat je až při příští žádost o příjem zprávy. Pokud nepřijde celá zpráva, žádost o příjem se opakuje, dokud není zpráva kompletní nebo nevyprší časový limit pro zpracování celé zprávy. Nevýhodou je, že pokud by přisla zpráva neuplná a hned na ni jiná tentokrát validní zpráva, spojení by se považovalo za nevalidní a došlo by k odpojení. Tento postup je pro projekt takové velikosti dostatečný, ale pro větší projekty by bylo vhodné implementovat žádost o znovuodeslání zprávy.

8. Závěr

Vytvoření elementární síťové hry pro dva hráče se podařilo. Hra má název *Inverse Battleships* a je inspirována klasickou hrou Lodě. Hra je rozdělena na klientskou a serverovou část. Klientská část je napsána v jazyce Python a serverová část v jazyce Go. V projektu bylo nutné řešit asynchronní komunikaci pomocí použití vláken na straně klienta a využití gorutin na straně serveru.

Pro síťovou komunikaci byl použit vlastní protokol založený na TCP. Spojení je realizováno na nízké úrovni pomocí BSD socketů. Přes protokol jsou posílány nešifrované zprávy v plain text formátu, rozdělené a parsované na základě delimiterů a pravidel z protokolu.

Hra je plně funkční a umožňuje hrát hru mezi dvěma hráči. Hra reaguje na vstupy od hráčů a zobrazuje herní stav v grafickém rozhraní. Hra umí i reagovat na neočekávané vstupy (odpojením) a zotavovat se z krátkodobé nedostupnosti serveru či klienta.

Tento projekt poskytuje základ pro další rozšíření, jako je implementace šifrování zpráv, větší množství herních pravidel apod. Zároveň slouží jako užitečný studijní materiál pro pochopení základních principů síťové komunikace a herního designu.