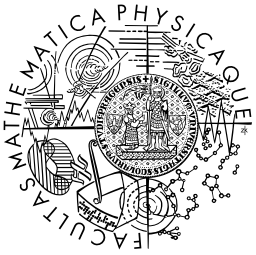


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

**BAKALÁŘSKÁ PRÁCE**

Klára Cihlářová

# Implementace hry Abaku

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: doc. RNDr. Petr Hnětynka, Ph.D.

Studijní program: Informatika

Studijní obor: IPSS

Praha 2022

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne .....  .....

Podpis autora

Ráda bych poděkovala vedoucímu mé bakalářské práce doc. RNDr. Petru Hnětynkovi, Ph.D. za trpělivost a cenné rady při sepisování této práce.

Název práce: Implementace hry Abaku

Autor: Klára Cihlářová

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: doc. RNDr. Petr Hnětynka, Ph.D., Katedra distribuovaných a spolehlivých systémů

Abstrakt: Cílem této práce bylo implementovat stolní početní hru Abaku. Abaku je hra pro 2 až 4 hráče podobná hře Scrabble, ale místo skládání slov se v této hře skládají početní příklady. Hráči se připojují k serveru pro hru mezi sebou nebo hrají pouze lokálně proti umělé inteligenci. V případě hry mezi více hráči se na závěr hry zobrazí statistiky pro dané hráče, které jsou uloženy v databázi. Dále je součástí této práce také program pro provádění operací s databází.

Klíčová slova: Abaku, desková hra, umělá inteligence, databáze

Title: Abaku game implementation

Author: Klára Cihlářová

Department: Department of Distributed and Dependable Systems

Supervisor: doc. RNDr. Petr Hnětynka, Ph.D., Department of Distributed and Dependable Systems

Abstract: The aim of this thesis was to implement a board math game Abaku. Abaku is a game for 2 to 4 players similar to Scrabble, but instead of composing words, we compose arithmetical problems. Players connect to the game server to play with each other or play only locally against artificial intelligence. In the case of a multiplayer game, at the end of the game, the statistics for the given players are displayed. They are stored in the database. Furthermore, part of this work is also a program for performing database operations.

Keywords: Abaku, board game, artificial intelligence, database

# Obsah

# Úvod

Abaku je společenská početní hra, která vznikla původně jako desková hra, a postupem času vznikly i různé počítačové implementace této hry.

Hra je velice podobná hře Scrabble. V té se vytváří slova z jednotlivých písmen a trénuje se tím slovní zásoba. V této hře se z číslic skládají početní příklady bez operátorů, které si hráči musí domyslet. Skládání příkladů trénuje práci s čísly a díky vytváření různých početních operací se procvičuje matematická dovednost uživatelů. Hráči se postupně střídají v jednotlivých tazích, každý se snaží vytvořit početní příklad za co největší množství bodů. Vítězem se stane hráč, který má na konci hry nejvíce bodů.

Cílem této práce bylo implementovat počítačovou verzi této hry, a to server i uživatelské prostředí klienta. Tento program umožňuje hru pro dva až čtyři hráče připojené k serveru. Druhou variantou je hra proti umělé inteligenci, kde je v nabídce pouze jeden typ umělé inteligence, ale program je velice snadno rozšiřitelný o další implementace. Na serveru se udržují statistiky o hře jednotlivých hráčů, k této databázi lze přistupovat i pomocí speciálního programu na práci s databází.

Práce je členěná do několika kapitol. V první kapitole práce se budeme věnovat důkladnému popisu a vysvětlení pravidel hry Abaku. Druhou kapitolu věnujeme analýze řešení. V následujících kapitolách poté popíšeme konkrétní řešení. Třetí kapitola se věnuje popisu struktury programu a vzájemné komunikaci mezi serverem a klientem. Ve čtvrté kapitole se podíváme na implementaci serveru a databáze na něm uložené. V páté kapitole se zaměříme na implementaci klienta. V šesté kapitole zhodnotíme řešení a porovnáme tuto implementaci s již existujícími implementacemi. V příloze poté najdeme také uživatelskou dokumentaci a popis přiložených souborů.

# 1. Pravidla hry

Abaku je početní desková hra, kterou lze hrát v minimálním počtu 2 hráčů a v tomto případě maximálním počtu 4 hráčů. Každý hráč po startu hry obdrží vylosované číslice do svého zásobníku. Poté se hráči střídají po jednotlivých tazích a každý má časový limit na vymyšlení a vyložení svého početního příkladu splňujícího několik základních pravidel. Pokud je příklad vytvořený správně, hráči se za něj připočítají body. Na konci hry vyhraje ten hráč, který získal za celou hru nejvíc bodů. Pravidla se liší s každým vydáním. Používala jsem následující pravidla abaku.org (a) a efko (2012), která jsem mírně upravila do vlastní verze.

## 1.1 Herní deska

Herní deska se skládá celkem z 225 políček, rozdělených do mřížky 15x15. Jednotlivá herní políčka mají různou barvu a rozlišujeme tedy následující druhy:

- *bílé políčko* - Žádný speciální význam, obyčejné políčko bez zvláštního bodového ohodnocení.
- *světle zelené políčko* - Políčko značí dvojnásobný počet bodů za číslici položenou na něj. Dvojnásobný počet bodů se počítá pouze v jednom tahu. V žádném dalším tahu už políčko nemá speciální funkci.
- *tmavě zelené políčko* - Políčko označuje trojnásobný počet bodů za danou číslici. Opět platí pouze pro první tah s číslicí na dané pozici. V žádném dalším tahu už políčko nemá speciální funkci.
- *světle modré políčko* - Toto pole označuje dvojnásobný počet bodů za celý příklad s číslicí na tomto místě. V žádném dalším tahu už opět políčko nemá speciální funkci.
- *tmavě modré políčko* - Políčko označuje trojnásobný počet bodů za celý příklad s touto číslici. V žádném dalším tahu už políčko nemá speciální funkci.
- *červené políčko* - Značí počáteční políčko, tedy na toto pole musí být položena číslice úplně prvního zahraného tahu. Jinak je první tah chybný.

Zelená políčka označují násobení počtu bodů za číslici a modrá políčka násobení bodů za celý příklad. Bílá políčka jsou základní a nemají žádnou speciální vlastnost. Červené pole je počátek. Okolo herní desky jsou rozmístěny zásobníky jednotlivých hráčů, ostatní hráči nevidí, co mají jejich soupeři za číslice.

## 1.2 Tvoření příkladů

Pro tvorbu příkladů používáme pouze kladná celá čísla a to konkrétně číslice 0-9. Výsledkem příkladů mohou být opět jen kladná celá čísla. Celkový počet číslic ve hře je 110, druhů číslic je deset a od každého druhu je jedenáct číslic. Příklady se skládají na herní pole pouze z číslic do jedné řady, operátory si je

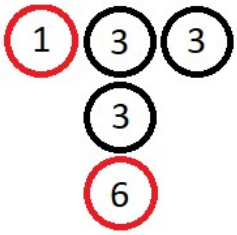
potřeba u každého příkladu domyslet. Příklady se čtou zleva doprava a shora dolů. Jako operace se uznávají pouze součet, součin, rozdíl, podíl, druhé a třetí mocniny a odmocniny celých kladných čísel. Každý příklad musí obsahovat právě jednu operaci. V žádném typu příkladu nelze použít nulu jako operand ani jako výsledek. Číslice, které jsou již položené na herní desce, lze použít do nového příkladu (ale bez bonusu pod danou číslicí). Přiložení nových kamenů podléhá speciálním pravidlům.

Přikládání nových kamenů:

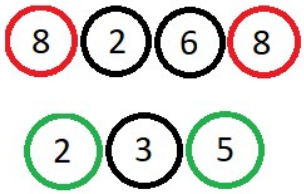
- V každém tahu je potřeba využít do příkladu alespoň jednu číslici již položenou na herní desce z předchozích tahů. Výjimkou je úplně první tah, kdy na desce není ještě žádná číslice.
- Po umístění číslic na herní desku již nelze měnit jejich umístění, nelze je přesouvat, odebírat ani je vyměnit za jinou číslici. Na desce zůstanou až do konce.
- Nelze pokládat číslice šikmo, v každém tahu lze pokládat všechny číslice jen do jednoho řádku nebo pouze do jednoho sloupce. Nelze část číslic vložit do řádku a část do sloupce, i když se jedná o jinak platný příklad viz obrázek 1.1 (červeně značíme chybně přiložené číslice, zeleně správně přiložené číslice).
- Nově přidané číslice musí dohromady tvořit jeden příklad, a to navíc společně s alespoň jedním již položeným kamenem z předchozích tahů. Jinak se jedná opět o chybný tah. Oba případy můžeme vidět na obrázku 1.2. Horní řádek ukazuje chybné přiložení číslic, dolní správné přiložení.
- Z nově přidaných číslic lze vytvořit několik příkladů v jednom tahu. Do bodového zisku se započítá každý správně vytvořený příklad, který obsahuje alespoň jednu nově přidanou číslici. Na obrázku 1.3 se nachází několik takových příkladů: $12 - 4 = 8$, $2 * 4 = 8$, $2^2 = 4$.
- Každá nově přiložená číslice musí se sousedními políčky vytvářet početní příklad. Pokud je toto pravidlo porušeno u jediné číslice, takto položené číslice nejsou správné a nemůže být tento tah uznán. Tedy pokud jsou všechny číslice vloženy do řádku, musí tvořit příklad v řádku. A navíc každá číslice, pokud sousedí s jinou číslicí ve sloupci, musí tvořit příklad ve sloupci. Toto pravidlo ilustruje obrázek 1.4. Příklady na tomto obrázku jsou rozepsané v následující sekci.

## 1.3 Bodování

Za každý odehraný tah daného hráče se spočítají body. Ty se ihned připočítají k celkovému skóre za celou hru. Body se získávají za každý správně vytvořený příklad, jehož součástí jsou právě přiložené číslice. Body jsou určeny pomocí hodnoty jednotlivých číslic v daných příkladech. Tedy číslice 0 nepřinese žádné body, číslice 9 přidává 9 bodů. Například na obrázku 1.4 za předpokladu přiložení číslic označených zeleně máme následující příklady: $3 - 1 = 2$, $3^2 = 9$, $1^2 = 1$, $\sqrt{1} = 1$, $1^3 = 1$, $\sqrt[3]{1} = 1$.

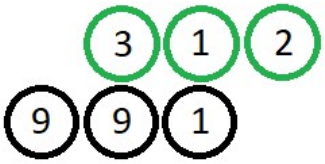



Obrázek 1.1: Chybné přiložení číslic do různých řádků a sloupců




Obrázek 1.2: Přiložení do jednoho řádku, chybné a správné


Obrázek 1.3: Několik příkladů v jednom tahu




Obrázek 1.4: Příklady v řádku i sloupci

Příklad $3 - 1 = 2$ odpovídá $3 + 1 + 2 = 6$ bodům.
Příklad $3^2 = 9$ odpovídá $3 + 9 = 12$ bodům.
Příklad $1^2 = 1$ odpovídá $1 + 1 = 2$ bodům.
Příklad $\sqrt{1} = 1$ odpovídá $1 + 1 = 2$ bodům.
Příklad $1^3 = 9$ odpovídá $1 + 1 = 2$ bodům.
Příklad $\sqrt[3]{1} = 1$ odpovídá $1 + 1 = 2$ bodům.

Dohromady dostane uživatel za takto zahraný příklad $6+12+2+2+2+2 = 26$ bodů. Pokud navíc některá ze zahraných číslic ležela na speciálním herním poli, označeným zeleně nebo modře (pole jsou popsaná v kapitole 1.1), dostane uživatel daný násobek bodů za číslici nebo příklad. Za zelená pole se násobí hodnota číslice dvěma nebo třemi. Za modrá pole se násobí počet bodů za celý příklad dvěma nebo třemi. Pokud uživatel použije dvě modrá pole v jednom tahu, počítá se bonus pouze jednou, tedy jako by bylo použito jedno modré pole. Vždy se nejdřív sečte počet bodů za celý příklad, včetně násobku bodů za číslice a až poté se vynásobí body za celý příklad. Takto spočítané body jsou výsledkem jednoho tahu hráče.

## 1.4 Průběh hry

Hra začíná rozlosováním číslic všem hráčům ve hře. Číslice jsou umístěny do zásobníků jednotlivých hráčů a nejsou viditelné ostatním. Každý dostane 5 číslic. Hru začíná první hráč. V úplně prvním tahu je nutné přiložit číslice na červené pole na herní desce. V následujících tazích už je nutné přidávat číslice k již položeným číslicím přesně dle popsaných pravidel pro přikládání číslic a vytváření příkladů. V každém tahu má uživatel 3 možnosti, co může udělat.

- Vyložit číslice ze svého zásobníku na herní plán dle pravidel, tedy utvořit početní příklad. Zkontroluje se, zda je příklad vytvořen dle pravidel. Při vyložení chybného příkladu jsou číslice vráceny zpět do zásoby hráče. Za správný tah jsou poté započteny body opět podle výše popsaných pravidel. Hráč doplní své číslice do celkového počtu 5, pokud jich ještě v zásobníku zbývá dostatek.
- Výměna všech kamenů v zásobníku hráče. Uživatel vrátí všechny číslice a dostane stejný počet nových číslic. Tah tohoto hráče touto výměnou končí a pokračuje s hrou další hráč. Body za tento tah nejsou žádné.
- Hráč nezahraje žádný tah, ale svoje číslice si chce ponechat. Následuje tah dalšího hráče v pořadí a za tento tah opět hráč nezískal žádné body.

Takto se hra střídá stále dokola, dokud nenastane jedna z ukončujících podmínek a tedy s tím i konec hry. Konec hry si popíšeme v následující kapitole.

## 1.5 Konec hry

Po skončení hry je za vítěze považován ten hráč, který za celou hru nasbíral nejvíce bodů. Pořadí za ním se seřadí podle stejného kritéria. V případě stejného počtu bodů se hráči dělí o dané místo. Ukončení hry proběhne v případě jakékoliv kombinace následujících tahů.

- Hráč vyložil příklad, který byl chybný. Tedy nezískal žádné body a číslice se vrací zpět do zásobníku.
- Hráč nezahrál nic, nechal si číslice v zásobníku, v podstatě vynechal svůj tah. Hráč opět nezískal žádné body.
- Výměna číslic z daného zásobníku. Všechny číslice se vrátí zpět a náhodně se vylosují nové. Opět je hráč bez bodů.

Každý hráč udělal během stejného kola jeden z výše popsaných tahů a žádný jiný. Tedy nikdo nezahrál tah za nějaké body. Potom je hra ukončena a je vyhlášen vítěz této hry. Pokud zůstanou hráčům na konci hry číslice v zásobníku, nic se z jejich bodového zisku neodečítá. Naopak ten hráč, který hrát může, získává body, dokud má co přikládat, i když ostatní už jen čekají na konec hry. Není žádné omezení na počet vzdání tahu jedním hráčem, hra pokračuje, dokud někdo může hrát. Pokud některému z hráčů dojdou číslice v jeho zásobníku a nemůže si již daný hráč dobrat nové číslice, je tímto způsobem také hra ukončena. Nikdo nepokračuje svým dalším tahem a je vyhlášen vítěz dle nejvyššího počtu bodů.

# 2. Analýza

## 2.1 Architektura

### 2.1.1 Klient a server

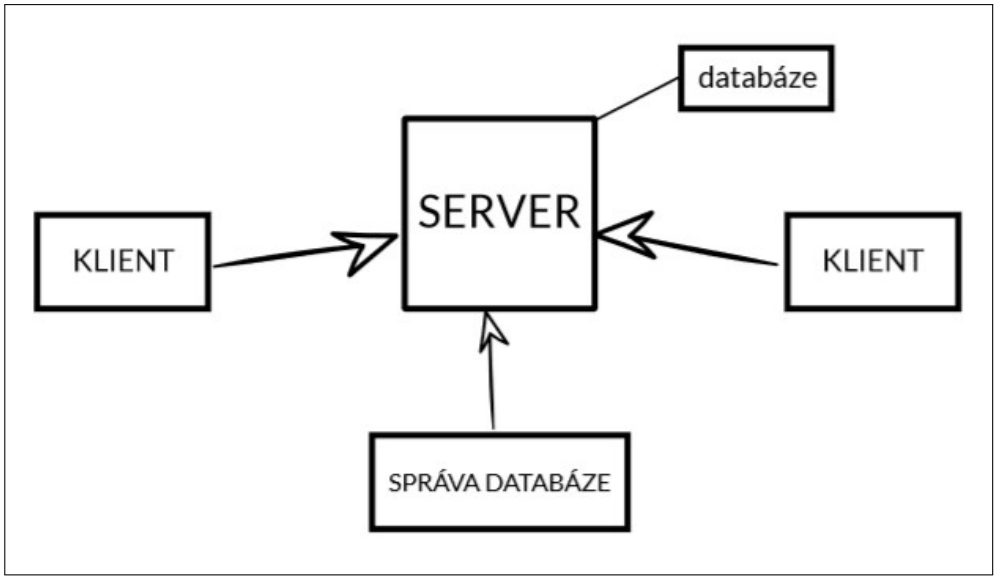
Vzhledem k tomu, že úkolem je napsat server a program pro uživatele, kteří mezi sebou chtějí hrát, je nutné práci rozdělit na 2 programy. Jeden program pro server, druhý speciálně pro klienta. Architektura hry mezi jednotlivými uživateli je na obrázku 2.1. Klient připojený k serveru by měl mít nějakou skupinu, v rámci které bude hrát. Kde uložit tyto informace? Můžeme je ukládat buď přímo v klientovi, každý klient by měl informaci o zbývajících číslicích, losoval by si sám nové číslice apod. Nebo můžu všechny tyto informace ukládat na serveru, tedy losování by probíhalo pouze na jednom místě. Tím by se zajistilo, že v případě například chyby při losování, by všichni klienti měli stejné informace o zbývajících číslicích. Na druhou stranu všechny informace o konkrétním klientovi by byly na serveru zbytečné, jako například číslice vylosované danému uživateli, protože tyto informace jsou ve většině případů serveru úplně k ničemu. Konkrétně číslice uživatel umisťuje na svůj plánek a pokud by je nechtěl vyměnit, server si je nemusí vůbec pamatovat.

Klient a server jsou tedy každý jeden program. Server by měl být spíše konzolová aplikace. Taková aplikace běží neustále, aby se k ní mohl kdokoliv a kdykoliv připojit, tak by nebylo dobré mít i grafické rozhraní a asi by to ani nedávalo smysl. Toto řešení by bylo vhodné, kdyby byla potřeba přehledně zobrazovat jednotlivé připojené skupiny a uživatele v nich.

Klient na druhou stranu může být jak konzolová aplikace, tak i grafická. Pokud by byl pouze konzolová aplikace, hra by mohla vypadat například následovně. Vytiskne se na obrazovku aktuální herní plán, uživatel zadá souřadnice a číslici, kterou chce odeslat. Poté, co je s příkladem spokojen, odešle tah. Až bude na řadě, vypíše se mu na obrazovku nová situace na herním plánu. Druhou možností je aplikace s grafickým rozhraním. Kde by se zobrazoval přehledně herní plán včetně barev pozadí, rozmístění hráčů kolem plánu, aktuálního skóre atd. Toto řešení je pro hru určitě lepší, přehlednější, pro uživatele jednodušší.

### 2.1.2 Správa databáze

Protože je požadavek na snadnou úpravu databáze nebo také zjišťování hodnot, je potřeba přiložit k práci ještě jednoduchý program, který zajistí tyto funkce. I tento program má dvě možnosti. Může to být jednoduchá konzolová aplikace s jednoduchým používáním nebo aplikace s grafickým rozhraním. Vzhledem k tomu, že nepožadujeme žádné speciální grafické prvky a jsme spokojeni pouze s čistě vypsanými daty, bylo by lepší zvolit konzolovou aplikaci. Tento program by se připojoval k serveru stejným způsobem jako ostatní klienti, tedy mohl by se připojit i dálkově přes internet. Druhou možností by bylo, že by obsluha požadavků na operace s databází byla součástí programu pro server. Kde by mimo připojování klientů byla obsluha i požadavků zadávaných uživatelem přímo na serveru. Architektura s připojováním na server je na obrázku 2.1.




Obrázek 2.1: Architektura hry na serveru

### 2.1.3 Umělá inteligence

Poslední částí, o které jsme se ještě nebavili je umělá inteligence. Můžeme ji vytvořit jako dalšího klienta, který se bude připojovat k serveru, a hra bude vypadat úplně stejně jako v případě hry proti jinému klientovi. Výhodou této implementace by bylo, že by hra vypadala úplně stejně, dalo by se ukládat i v tomto případě statistiky na server, nevýhodou by bylo, že pokud nemá uživatel přístup k internetu, nemohl by si vůbec zahrát. Proto je zde druhá varianta, agent je třída, která je součástí klienta. Tedy klient se nemusí za účelem hry nikam připojovat a hraje se pouze lokálně. Může to být bráno jako tréninková hra před hrou proti ostatním klientům na serveru.

## 2.2 Komunikace

Je důležité se podívat na to, jak bude probíhat komunikace mezi serverem a klientem. Na komunikaci budu využívat *sockety*. Na straně serveru to bude *ServerSocket*, na straně klienta *Socket*. Komunikace bude poté probíhat pomocí těchto socketů a vstupních a výstupních proudů, do kterých bude daná strana posílat data. Jak budou posílaná data a tedy komunikace vypadat? Je zřejmé, že některé zprávy budou obsahovat pole číslic vylosovaných na serveru a určených klientovi nebo řetězce obsahující jména připojených hráčů. Posílané zprávy mohou být řetězce, hodnoty z výčtových typů nebo například číslice. Lepší je posílat, pokud je to možné, hodnoty nějakého typu než řetězce. Hodnoty se lépe porovnávají na rovnost, je menší pravděpodobnost, že nastane chyba, než při porovnávání na rovnost s nějakým řetězcem. Určitě bude potřeba odlišit nějakým způsobem fázi hry nebo oznámit typ statistiky pro výpis. Tedy v některých případech se pošle hodnota z výčtového typu označující jeden z možných případů.

## 2.3 Knihovny

### 2.3.1 Uživatelské rozhraní

Na grafického klienta lze použít jednu z knihoven *Swing*, *AWT*, *JavaFX*. Všechny tyto knihovny umožňují vytvářet grafické rozhraní dané aplikace. *Swing* je ve skutečnosti rozšířením knihovny *AWT*, tedy spousta metod používá třídy z původní knihovny. Pro práci tedy budu vybírat mezi *Swing* a *JavaFX*. U *JavaFX* je nevýhodou, že se musíme zabývat moduly i v nemodulární aplikaci. *Swing* vyhovuje jak výběrem komponent, tak i po všech ostatních stránkách. I když v programu používám jak *Swing*, tak i *AWT*, nikdy nemíchám dva druhy komponent dohromady. Jen pro správné použití metod z knihovny *Swing* musíme vkládat argumenty z knihovny *AWT*.

Klient může být jak 2D, tak i 3D aplikace. V tomto případě by 3D aplikace neměla příliš význam, deska by byla pořád 2D, maximálně číslice v jednotlivých zásobnících by byly jiné ve 3D než ve 2D grafice. Pokud je tedy klient grafická aplikace, potřebujeme rozmyslet způsob, jak rozdělit jeho jednotlivé části. Můžeme vytvářet aktuální obrazovku vždy v momentě, kdy je potřeba, což může někdy být zbytečně složitější, byla by potřeba odstranit staré prvky a přidat nové na panel. Nebo si jednotlivé obrazovky předpřipravit a rozdělit na panely dle částí průběhu hry. Každý panel by tedy obsluhoval jednu část, jeden by byl pro přihlašování, druhý pro hru, třetí pro výpis statistik atd. V herním poli bude potřeba vytvořit každé políčko ze stejné komponenty. Tou komponentou může být například tlačítko nebo label. Protože máme dva požadavky na políčko, aby mělo nějakou barvu pozadí a aby na něj šla umístit číslice, stačí nám pouze na reprezentaci takového políčka label.

### 2.3.2 Logging

Další důležitou částí je vypisování chyb v programu. Je několik možností, jak je uživateli zobrazit. Můžu použít chybové dialogy z grafických knihoven, které se zobrazí uživateli přes obrazovku a které musí potvrdit. Tyto dialogy jsou vhodné pro program s grafickým rozhraním. Další možností je chyby vypisovat do konzole, odkud je konzolová aplikace spuštěna. Tedy opět se uživateli chyba vypíše přímo tam, kam kouká. Obě tyto varianty mají i svou nevýhodu. Pokud chybu potvrdíme a odklikneme ji nebo ukončíme celý běh aplikace, chybu, která nastala, už nikdy nezjistíme. Pokud by z nějakého důvodu nastala chyba, která by ukončila běh celé aplikace, kde pak najdeme důvod této chyby? Na to můžeme použít balíček *java.util.logging*. Ten umožňuje zapisování chyb do souborů, takže i po ukončení aplikace lze nalézt v daném souboru informaci o chybách.

### 2.3.3 Databáze

Poslední a neméně důležitou částí je databáze. Databázi chceme mít vytvořenou pouze jednu pro všechny vytvořené skupiny na serveru. Tato databáze musí být tedy umístěna na serveru a ten také obsluhuje požadavky jednotlivých klientů na informace z databáze. Musíme si rozmyslet ještě typ přístupu do databáze. Jednou z možností je JDBC API, které umožňuje připojit se pomocí ovladače k databázi a vytvářet SQL příkazy nad danou relační databází. Nevýhodou je,

že při vytváření SQL dotazů opravdu musíme vytvořit celý řetězec s dotazem. Druhou možností je využít nějakou techniku, která zajistí mapování do programovacího jazyka sama. Přesně touto technikou je ORM, neboli objektově relační mapování. Jak již říká název techniky, umožňuje namapovat relační databázi na objekt v daném jazyce. Výhodou v takovém případě je tedy to, že programátor vůbec nepoužívá SQL jazyk a pracuje s daty z databáze jako s objekty. Díky ORM je implementace komunikace s databází daleko jednodušší, i proto jsem ji zvolila do svého programu.

# 3. Architektura programu

V této a následujících dvou kapitolách se podíváme podrobněji na to, jakým způsobem byly řešené jednotlivé části tohoto programu. Popis tříd, metod, atributů a dělení programu na části najdeme blíže v programátorské dokumentaci přiložené k této práci. Na všechny chyby, které nastanou při spouštění některého z mých programů, používám třídu *java.util.logging*, která umožňuje zaznamenávat jednotlivé chyby i do souboru, takže po skončení programu lze zjistit chyby při spouštění daných programů. Celkově jsou v mém programu vytvořeny 3 logovací soubory. Pro server, klienta a správu databáze je vždy jeden. V těchto souborech jsou zaznamenané všechny chyby, které nastanou v průběhu a mohou stát za ukončením programu. Přesněji se tato kapitola bude věnovat tomu, jak jsem zařídila komunikaci mezi klienty a mezi klientem a serverem. Implementaci klienta a serveru rozebereme v dalších dvou kapitolách. Při psaní celého programu jsem využívala dokumentaci Oracle a knihu o vytváření her Davison (2005).

Součástí programu je v adresáři *doc* vygenerovaná dokumentace.

## 3.1 Koncept

Moje práce, jak jsem popsala výše, obsahuje tři spustitelné programy. Jedním z nich je server, druhý klient a třetí je program pro správu databáze. Hru mezi klienty bylo nutné zprostředkovat přes server, který bude s jednotlivými klienty komunikovat a umožňovat posílání zpráv ostatním klientům. Tedy na serveru se musí udržovat informace o aktuálním stavu hry, o připojených klientech apod. Klienti se připojují k serveru pouze kvůli hře proti ostatním, k serveru se připojuje také správce databáze. Bylo nutné vymyslet, jakým způsobem spolu budou komunikovat právě server a klient. Na straně klienta uživatel zadává různé informace, volí si soupeře pro hru, připojuje se k serveru pomocí IP adresy a portu, vyplňuje své přihlašovací údaje. Po spuštění jeho hry umisťuje na svém herním plánku číslice do různých příkladů a získává za ně body nebo čeká na tah soupeře. Pokud nastane konec hry, zobrazí se mu pořadí hráčů seřazené dle získaných bodů. A případně může zvolit panel statistiky, který umožní z databáze na serveru vypsat jednotlivé statistiky o všech jeho soupeřích i o něm. Tedy jsem potřebovala rozmyslet, které herní informace je nutné udržovat na serveru a které bude uživatel posílat.

Komunikace probíhá pomocí socketů. Socket je koncový bod připojení přes počítačovou síť. Je nutné na serveru vytvořit ServerSocket, který bude poslouchat na nějakém námi určeném portu. Na spojení klienta čeká v metodě **accept()**. V klientovi vytvoříme Socket, který se pokusí připojit k IP adrese a portu zadaným uživatelem. Pokud připojení proběhne v pořádku, pro komunikaci je nutné vytvořit streamy, jeden vstupní, druhý výstupní. Pak již není problém zapisovat do těchto streamů informace pro server nebo klienta.

Ještě než popíšu, jak přesně jsem vyřešila komunikaci v průběhu hry, musím popsat důležitý koncept komunikace mezi klienty. V průběhu hry je potřeba, aby jeden klient mohl poslat informace, například o svém tahu, všem ostatním klientům ve skupině. Taková metoda se jmenuje **broadcast()**, dostane jako parametr zprávu, kterou chce poslat ostatním, a vlákno odpovídající připojenému klientovi.

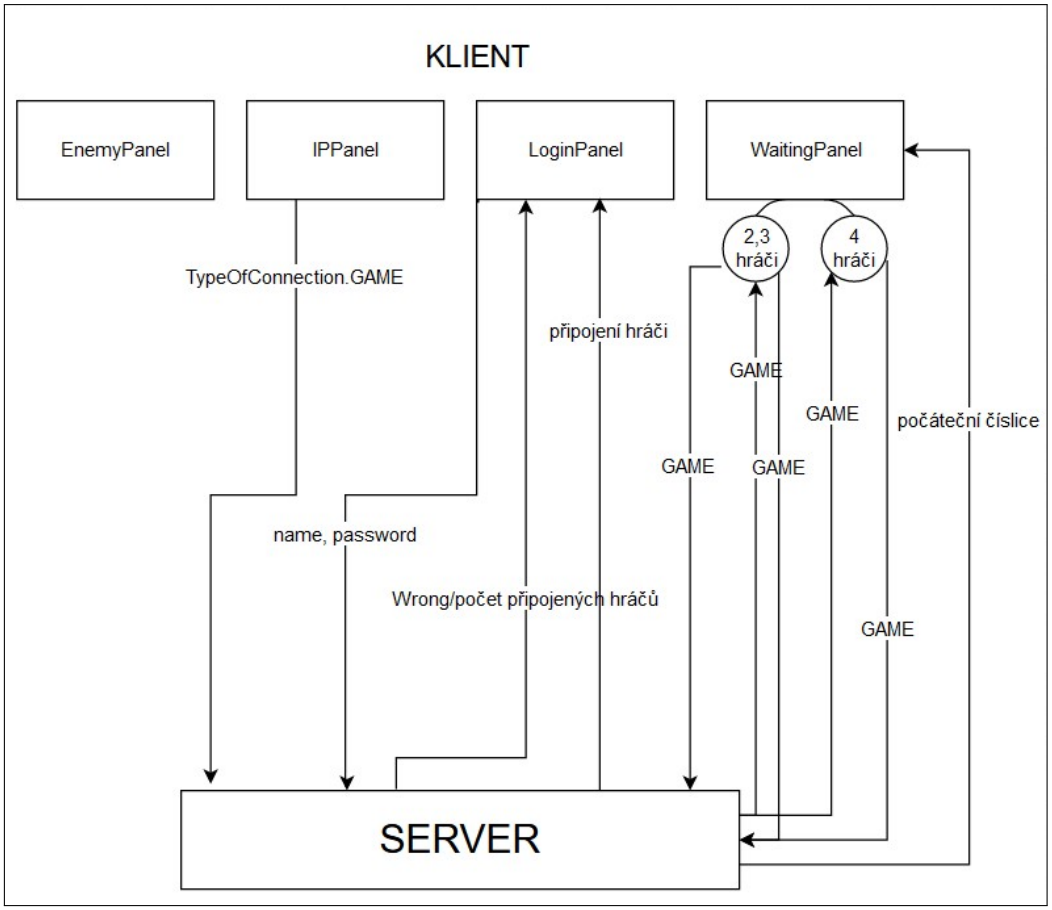
Tuto zprávu rozešle všem ostatním klientům připojených ve skupině. Tato metoda je využitá především pro rozeslání tahu jednoho klienta všem ostatním. Klient pošle informace o svém tahu na server. Ten tyto zprávy přepošle všem ostatním. Týká se to informací o počtu zahraných kamenů, poli všech umístěných číslic, jméně hráče, který hrál, s jeho počtem bodů a počtu číslic, kolik ubylo hráči ze zásobníku (tedy hráč už hraje s menším počtem než 5 číslic).

## 3.2 Komunikace při spouštění hry

Celou komunikaci při spouštění hry můžeme vidět na obrázku 3.1. Po připojení klienta k serveru se mu zobrazí na obrazovce pokyny k přihlášení k serveru. Po zadání jména a hesla se tyto informace pošlou serveru, který zkontroluje v databázi, zda se v ní vyskytuje dané jméno. Pokud ano, zkontroluje heslo k tomuto jménu. Ze serveru klient obdrží informaci o tom, zda bylo heslo správné. Tato informace je řetězec, protože pokud je odpověď ze serveru *Wrong*, říká to, že heslo bylo zadáno špatně. Jméno v takovém případě zůstane stejné, ale heslo se požaduje tak dlouho, dokud nebude správně. Pokud je ale odpověď číslo, odpovídá počtu připojených klientů do dané skupiny. Proto je tato informace poslána jako řetězec. Jméno nově připojeného uživatele se pošle broadcastem všem ostatním uživatelům ve skupině. Poté, co je zadáno správné heslo a klient obdrží informaci o počtu připojených uživatelů, je dále posláno ze serveru klientovi pole hráčů, kteří jsou již připojeni.

Všichni hráči ze skupiny jsou ve stejné čekací místnosti. Když se poté někdo připojí, dostanou tito hráči informaci o jeho jméně broadcastem ze serveru. Spuštění hry nastává ve dvou případech, kliknutím na tlačítko pro spuštění hry, nebo připojením čtvrtého hráče do skupiny. Aby čekající klienti odlišili počátek hry a připojení nového hráče, použila jsem řetězec ”GAME” jako informaci pro klienta, že začala hra. Řetězec jsem zvolila proto, že se rovnost na ”GAME” kontroluje v místě, kde řetězec značí jméno hráče. Tedy z toho plyne, že při vytváření jména hráče jsem musela ošetřit i variantu, že uživatel zadá jméno ”GAME”, pak by vyvolal hru pouze svým jménem. V závislosti na tom, čím byl vyvolán začátek hry, se posílá poprvé řetězec ”GAME” buď ze serveru nebo z klienta.

V případě, že se připojí čtvrtý hráč, hru vyvolá server a pošle informaci o začátku hry všem klientům. Tito klienti pošlou stejnou zprávu zpět jako potvrzení, že je hra opravdu zahájena. Pokud chce hru spustit klient, klikne na tlačítko. V obsluze kliknutí na toto tlačítko se pošle opět zpráva o začátku hry, tedy řetězec ”GAME”. Na serveru se tato informace rozešle úplně stejně jako v předchozím případě všem připojeným klientům skupiny. Tedy i tomu uživateli, který hru spustil, protože i on má spuštěné vlákno, které čeká na připojení dalších hráčů. Opět se pošle stejná zpráva zpět serveru jako potvrzení začátku hry. Tímto způsobem je spuštěna hra. Pouze v tomto případě a u potvrzování správnosti hesla se používá posílání řetězce, v ostatních případech používám především hodnoty z výčtového typu. Poté ještě server vylosuje každému klientovi jeho číslice a pošle je. Tím se kompletně připraví hra.



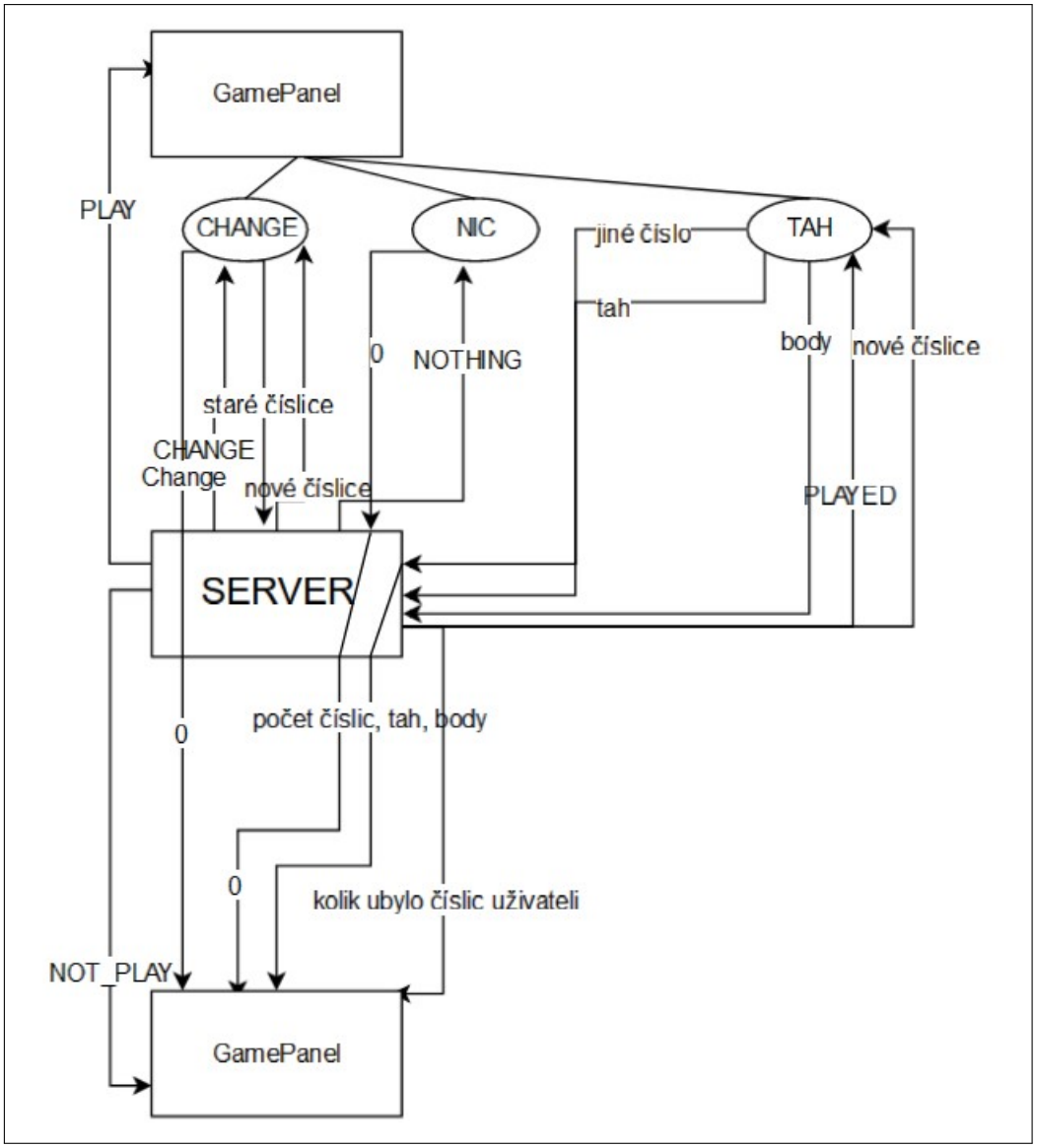

Obrázek 3.1: Komunikace při spouštění hry

## 3.3 Komunikace v průběhu hry

Celý průběh komunikace je vidět na obrázku 3.2. Při hře posílá server jeden typ informace svým klientům, a to jednu hodnotu z výčtového typu *PlayersOption*. Tyto hodnoty označují, v jaké fázi je daný hráč. Pouze jednomu klientovi v celé skupině se pošle hodnota **PLAY**, všem ostatním připojeným ve skupině se ve stejnou chvíli pošle hodnota **NOT_PLAY**. Další hodnotou je **END**, což označuje konec hry a zajistí, že se panel přepne u klienta na koncový panel. Tato hodnota se posílá ze serveru klientům v případě, že hráči došly číslice nebo všichni hráči neudělali platný tah v jednom kole. Vlákno na serveru čeká, až mu od klienta přijde informace, že je aktualizovaná informace o tom, kdo je na řadě. Tedy klient pošle na server zprávu, kdo je na řadě. Tato informace je již nastavená ve sdílené proměnné od vlákna, které odpovídá klientovi, který hrál.

Vlákno na serveru obsluhující klienta, který je právě na tahu, přijímá několik informací od klienta. Potřebuje znát počet zahraných číslic, celé pole tahu, počet bodů za tento tah (server nemá informaci o herním plánu, o barevných políčkách, pouze zprostředkovává komunikaci). Jak jsem již napsala v sekci 3.1, broadcastem se pošlou ze serveru tyto informace ostatním klientům, kteří si jednotlivé údaje zanesou do herní plochy.

Komunikace se serverem je ale důležitá nejen kvůli zprostředkovávání této komunikace mezi hráči, ale i kvůli tomu, že stav hry je uložen na serveru. Tato skupina má na serveru uloženou vlastní instanci hry, kde jsou uloženy zbývající




Obrázek 3.2: Komunikace při hře

nevylosované číslice. Server obsluhuje i losování nových číslic pro uživatele a jejich posílání klientovi. Tudíž po zahrání platného tahu se po obdržení informací z klienta a jejich rozesláním ostatním vylosují nové číslice, které server pošle klientovi, a ostatní informuje o novém počtu zbývajících číslic daného hráče v zásobníku, přesněji pošle informaci, o kolik se počet zmenšil po tomto tahu.

### 3.3.1 Komunikace serveru s hrajícím hráčem

Informace o počtu zahraných číslic se posílá po kliknutí na odeslat tah (a po kontrole správnosti, to teď neřešíme). Tato informace se na serveru rozliší na 3 různé možnosti.

Obdržíme číslici větší než nula. V tomto případě je vše přesně, jak jsem popsala v předchozích odstavcích, ale navíc server musí poslat jednu informaci hrajícímu klientovi. Server nejdříve pošle hodnotu z výčtového typu *GameConst*, přesněji hodnotu **PLAYED**. Tedy klient se dozví ve funkci **play()**, kde čeká právě na

tuto informaci, že zahrál správný tah a zde také očekává nově vylosované číslice poslané ze serveru.

V případě, že číslice je 0, uživatel nehrál platný tah. Pošle mu server hodnotu **NOTHING** a v klientovi se již po obdržení této hodnoty nic neděje.

Posledním případem je, že místo číslice přijde řetězec "Change". Uživatel si mění ve svém tahu číslice za nové, a to tak, že klikl na tlačítko pro výměnu a tento řetězec se odeslal serveru. Server pošle klientovi hodnotu **CHANGE**, která značí výměnu číslic. Klient poté čeká na obdržení svých nových číslic, které mu server pošle.

Ve všech 3 případech server pošle tomuto klientovi i informaci o tom, kdo je na řadě, kterou mu klient přepošle zpět. Tuto informaci mu posílá proto, aby klient mohl hodnotu změnit ze jména na řetězec "END". Tato změna může proběhnout pouze pokud uživatel dohrál všechny číslice a už žádné nové nedostal. Zbytek případů ukončení hry řeší server, který má informace i o ostatních hráčích.

### 3.3.2 Komunikace serveru s nehrajícími hráči

Tato část je mnohem jednodušší. Musela jsem zařídit, aby se tah odehraný jiným hráčem zobrazil na obrazovkách všech ostatních hráčů a samozřejmě aby se uživatelé správně střídali. Nehrajícím hráčům se na začátku tahu poslala informace o tom, že teď nejsou na tahu. Klienti okamžitě začali čekat pouze na informaci o počtu zahraných číslic.

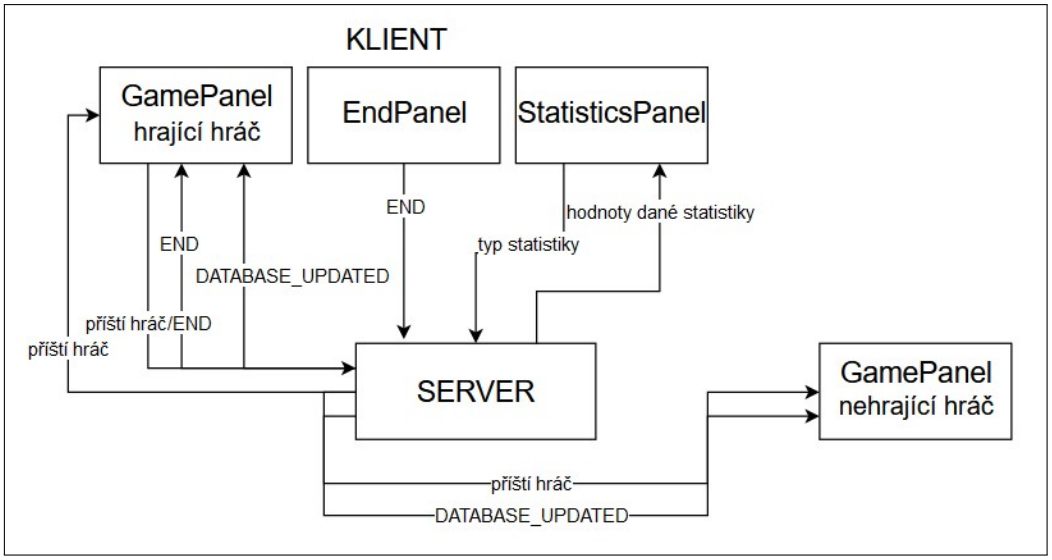
Pokud byl počet číslic nenulový, obdrží od serveru další informace o tom, jaké tahy byly přesně zahrány (tyto číslice umístí na správné pozice na své ploše), hráče, který byl na tahu, s informací o jméně a bodech. Pomocí těchto informací aktualizuje body na svém panelu u daného hráče. Poslední informací v této části, kterou dostane od serveru, je počet číslic, o který přišel hráč na tahu ze svého zásobníku (a už je není čím doplnit). Tento počet odmaže klient na své obrazovce a udržuje tak informace o zbývajících číslicích všech hráčů hned na první pohled.

Pokud byl počet číslic nula, žádné speciální informace ze serveru nebudou klientovi poslány. V obou případech se těmto klientům poté pošle informace o jméně hráče, který hraje příště, ale nic se s touto informací nedělá a jen se přepošle zpět na server. Vlákna nehrajících klientů na serveru čekají na novou příchozí zprávu z klienta a díky tomu se nespustí další tah hráče, dokud hráč na tahu nedohraje a neaktualizuje jméno hráče, který bude příště na tahu. Poté se spustí celá smyčka znovu jen s jiným hráčem na tahu, jinak vše probíhá úplně stejně.

## 3.4 Komunikace ve zbytku programu

Komunikaci ve zbytku programu lze vidět na obrázku 3.3. Pokud uživateli, který byl na tahu, došly číslice, změní jméno příštího hráče na tahu na "END". Pokud server zjistil, že daný uživatel nehrál stejně jako všichni ostatní, pošle on klientovi rovnou informaci o konci. V obou případech nastává konec hry.

Po konci hry čekají všichni klienti na informaci o aktualizaci databáze. Jak se aktualizace provádí, bude řečeno v příští kapitole. Zde řešíme pouze způsob a důvod komunikace mezi serverem a klienty. Aktualizace bude oznámena ze serveru pomocí hodnoty **DATABASE_UPDATED** z výčtového typu *Commands*.




Obrázek 3.3: Komunikace po hře

Pokud klienti obdrželi tuto informaci, mají jednotliví uživatelé na obrazovce zobrazený panel pro konec hry.

Na tomto panelu je vypsané finální skóre seřazené od uživatele s nejvyšším počtem bodů po toho, který bodů získal nejméně. A dále dvě tlačítka. Server v tuto chvíli čeká na opakované čtení hodnot typu *DatabaseGame*, tyto hodnoty značí typ statistiky, jejíž hodnotu chce klient získat od serveru. Po kliknutí na tlačítko *Ukončit*, se na server pošle zpráva **END**, která značí, že ukončujeme komunikaci se serverem (i čtení statistik). V případě, že si uživatel zvolí panel statistiky, tak po kliknutí na jednotlivé typy klient pošle danou informaci serveru. Zpráva bude jedna z následujících hodnot: **GAMES, WINS, MAX_POINTS, AVG_POINTS**. Po poslání zprávy pro ukončení se vlákno na serveru pro komunikaci s daným klientem ukončí.

# 4. Server

Mým úkolem bylo vytvořit herní server, ke kterému se budou jednotliví uživatelé připojovat. Herní server byl vytvořen jako jeden spustitelný program, přesněji konzolová aplikace. Tato aplikace bude spuštěna neustále a bude čekat na připojení klientů. V tomto případě by bylo grafické rozhraní zbytečné. Pro připojení klientů byla potřeba vytvořit ServerSocket poslouchající na daném portu. Aby si mohl uživatel port určit, načítám port z konfiguračního souboru. Jméno souboru, který používám, je *konfigurace.properties*, z tohoto souboru lze snadno získat požadovanou položku. Server poté poslouchá pouze na tomto portu, aby se klient správně připojil, musí zadat právě tento port mimo správné IP adresy.

## 4.1 Obsluha hry na serveru

Pro každého klienta je na serveru vytvořeno nové vlákno pro komunikaci s daným uživatelem. Vlastní vlákno pro každého uživatele jsem zvolila pro lepší přehled kódu, také pro jednoduchost, díky tomu může vlákno pro nehrajícího hráče čekat na zprávu v jiné části programu než vlákno s hrajícím klientem. V obou případech je vlákno daného klienta reprezentováno třídou *ClientThread*, která obsluhuje celou komunikaci s klientem. Zajistí přihlášení k serveru, vypsání ostatních připojených klientů v dané skupině, obsluhu hry i vypisování statistik po hře.

Při přihlašování uživatele přichází od klienta jméno a heslo a kontroluje se v databázi, zda heslo odpovídá danému jménu. Hesla sice přichází na server nešifrovaná, ale do databáze se ukládají hašovaná. Pokud se porovnává heslo s již uloženou hodnotou v databázi, porovnávají se haš hodnoty.

Na serveru se udržuje informace o počtu skupin a klientech v daných skupinách. Na uložení těchto informací jsem použila *List<GameService>*, skupina je reprezentována jednou instancí *GameService*, ve které jsou uložené údaje o všech uživatelích skupiny. Společná instance se využívá také ke sdílení proměnných mezi vlákny, tedy je jedna instance skupiny a změny provedené ve skupině jedním vláknem jsou vidět i v ostatních vláknech. GameService obsluhuje hru daných uživatelů ve skupině. Veškerá hra se ovládá v této třídě. Mimo tato vlákna se na serveru obsluhuje pouze přijímání nových klientů.

Jednotlivé uživatele reprezentuji na serveru jako *Clients*. Byla potřeba si zapamatovat jak údaje o hráči, tak i vlákno pro komunikaci s ním. Také byl požadavek na porovnatelnost mezi hráči kvůli rozpoznání vítěze. Klient je menší než jiný, pokud má méně bodů, stejný, pokud má stejně bodů, větší, když má větší počet bodů. Tito hráči jsou všichni uloženi v nějaké skupině. Skupinu reprezentuji třídou *Group*. V této třídě musí být uloženy všechny společné hodnoty. Tedy seznam všech hráčů ve skupině, reprezentovaný listem klientů. Dále společná instance hry udržující informace o aktuálním stavu hry, počtu zbývajících číslic apod. A také informaci, zda hra v této skupině začala, což je důležité pro to, aby se hráč nechtěl připojovat ke skupině, ve které jsou dva hráči, ale již začali v tomto počtu hrát.

Pro uložení aktuálního stavu hry jsem zvolila třídu *Game*. Jedna instance této třídy patří právě jedné skupině. V této třídě je nastaven celkový počet číslic. Pokud by se požadovala změna rozložení počtu číslic, méně nul, více dvojek apod., vše se mění v této třídě. Náhodné číslo losuji vygenerováním náhodného čísla od

0 do počtu možností. Tyto informace se vyskytují pouze na serveru a ten má za úkol pro každou skupinu obstarávat losování nových číslic.

Protože je jedna instance hry sdílená mezi klienty, byla potřeba, aby metody **drawNew()**, **saveOld()** byly *synchronized*. Synchronized metody zajistí, že pokud jedno vlákno spouští kód této metody, jiná vlákna do této metody nepustí, vlákna se pozastaví do té doby, než budou moci tuto metodu spustit. Tím se vyhneme *race condition*, protože dvě vlákna nebudou moct měnit jednu proměnnou ve stejnou chvíli, prolínat se. Race condition je typ chyby v programu, která nastává v případě, že se výsledky programu chovají nepředvídatelně, pokaždé jinak, způsobený nevhodným pořadím prováděných úloh. Nastává především ve vícevláknových programech. Po opuštění synchronized metody jsou změny provedené jedním vláknem vidět ve všech ostatních vláknech.

Obsluha hry probíhá v každém vláknu zvlášť podle toho, zda je daný klient na řadě nebo ne. Vlákna mají společné sdílené proměnné v této třídě. Přesněji obsluha hry probíhá v metodě **game()**, základní myšlenkou obsluhy je rozdělení na dvě části. S klientem, který hraje, se komunikuje jinak než s klientem, který nehraje. Nehrající klient musí poslat zprávu na server, aby dané vlákno zkontrolovalo, zda tentokrát není na řadě. Čekání na zprávu jsem tam přidala proto, aby při příští kontrole, zda je klient na řadě, už bylo nastavené nové jméno příštího hrajícího klienta, jinak probíhá komunikace dle popsaného v sekci 3.3.2. V hrajícím vláknu probíhá komunikace mezi serverem a klientem přesně podle sekce 3.3.1.

## 4.2 Obsluha po hře na serveru

Hra skončí v momentě, kdy místo jména příštího hráče bude v řetězci uloženo "END", značící, že nastal konec hry. Po konci hry je obsluha v jiné metodě, a to **afterGame()**. Na konci hry je potřeba aktualizovat databázi. Do databáze ukládám nejen počet bodů, ale také vítěze hry. Na to, abych zjistila, kdo vyhrál, je nutné mít klienty setříděné dle bodů. Protože stačí, aby aktualizaci provedlo jedno vlákno a aktualizovalo hodnoty u všech hráčů, setřídím pole klientů dle bodů pouze v tomto vlákně a toto setříděné pole předám metodě, která aktualizuje databázi. Metoda **updateDatabase()** z této třídy volá metody z databáze se správnými hodnotami. Díky tomu, že jsme si pole setřídili, na prvním místě se nachází vítěz. To využijeme pro předání informace o vítězi. Poté, co je databáze aktualizovaná, rozešle vlákno zprávu o aktualizaci ostatním. Jednotliví klienti mohou požadovat jakékoliv statistiky a databáze je již připravená vracet správné údaje.

Na výpis statistik mám na serveru smyčku pomocí while cyklu. Přečte se název statistiky, kterou uživatel požaduje, a zjistí se, zda je to opravdu název statistiky. Pokud ano, znamená to, že uživatel může požadovat další a mimo poslání její hodnoty uživateli, čekáme na další požadavek. Pokud zpráva neobsahuje název statistiky, žádá se ukončení připojení. Smyčka i vlákno se ukončí.

## 4.3 Databáze

Jak bylo již několikrát zmíněno v této práci, na serveru je spuštěna databáze, do které se ukládají hodnoty z jednotlivých her, informace o hráčích. V této sekci si popíšeme, jak tato databáze funguje, co a jak se do ní ukládá a popíšeme si i další spustitelný program, který pracuje právě s touto databází.

### 4.3.1 Vytvoření databáze

Na implementaci databáze jsem použila framework *Hibernate* viz javatpoint. Ten umožňuje objektově relační mapování (ORM). Toto mapování usnadňuje konverzi mezi relační databází a objektově orientovaným jazykem. Díky tomuto mapování se usnadňuje práce s databází. V databázi můžeme přistupovat k jednotlivým atributům pomocí metod get() a set(). Nemusíme řešit žádné vytváření dotazů na databázi, takže toto využití ORM je ideální. Dále je zajištěn i správný přístup do databáze z vícevláknových aplikací. Mapování může být určeno buď pomocí xml souboru, nebo anotacemi. Můj mapovací soubor, který jsem zvolila pro svůj program, lze nalézt v příloze na obrázku A.2. V něm si také můžeme všimnout, že každý hráč má své unikátní ID podle toho, kolikátý se daný uživatel do databáze přidal. Toto číslo se automaticky s každým dalším uživatelem zvyšuje. ID nám tak může podat informaci o tom, který uživatel byl v databázi dříve.

Aby hibernate správně fungovalo, potřebuje několik nastavení, jako JDBC driver, adresu databáze, dialekt, což je generátor SQL dotazů, cestu k mapovacím souborům apod. K tomu slouží druhý důležitý konfigurační soubor *hibernate.cfg.xml*. Ten můžeme vidět na obrázku A.1. Zvolila jsem takové nastavení databáze, která je uvnitř paměti a ne na disku, přístup k disku je pomalejší než přístup do paměti, navíc moje databáze nebude obsahovat takové množství dat, aby bylo potřeba je ukládat na disk. Dále se databáze smaže, jakmile se zruší všechna připojení k databázi. Pokud se přeruší běh serveru, u této hry není důležité uchovávat data i po vypnutí serveru. Oba soubory jsou součástí softwarové přílohy, jak popisuje příloha C.

Pokud máme oba soubory, můžeme vytvořit *Session* pro komunikaci s databází. Ve třídě *HibernateUtil* použiji jako konfiguraci mnou vytvořený konfigurační soubor a vytvořím *SessionFactory*, která slouží pro otevírání *Session*. Jedna *Session* umožňuje mazání, přidávání i výpis dat z databáze. Ale její životnost by měla být jedna transakce. Pokud je jedna transakce dlouhá, měla by být rozdělena do více transakcí. Po provedení dané transakce se *Session* uzavře.

### 4.3.2 Hráč v databázi

Hráče v databázi reprezentuje třída *DatabasePlayer*. V této třídě jsou obsaženy všechny údaje, které požadujeme, aby byly uložené v databázi. Tedy id hráče, přezdívka, heslo, průměrný počet bodů, maximální počet bodů, celkový počet výher a her. Dále jsem potřebovala vypisovat tohoto hráče na obrazovku, proto jsem přepsala metodu **toString()**. V této metodě jsem vypsala všechny informace o hráči až na heslo. Nedává smysl, aby se heslo vypisovalo na obrazovku, i když je hašované. Tento údaje by nedal vůbec žádnou důležitou informaci.

### 4.3.3 Komunikace s databází

Pro každou operaci v databázi jsem vytvořila speciální metodu. Jednotlivé metody obsluhují editaci uživatele, smazání jednoho uživatele, všech uživatelů, přejmenování atd. Při přidávání nového uživatele do databáze jsem nastavila všechny statistiky na 0.

Součástí mé práce je program pro správu databáze. Aby se i pro správu databáze šlo připojit z jiného PC, tento program se pokouší připojit k serveru dle zadané IP adresy a portu. Tento program pošle na server informaci o tom, že se jedná o program pro správu databáze, tedy zprávu *MANAGE_DATABASE*. Po přijmutí této informace server okamžitě přechází do stavu, kdy čeká na vybraný databázový příkaz a spouští obsluhu tohoto příkazu. Po připojení správce k serveru je již obsluha jednoduchá. Jednotlivé příkazy případně mohou požadovat i jiné informace ze serveru. Pro výpis uživatele pošle správce jméno serveru, pro úpravu údajů uživatele pošle správce jméno a dané hodnoty apod.

# 5. Klient

## 5.1 Návrh klienta

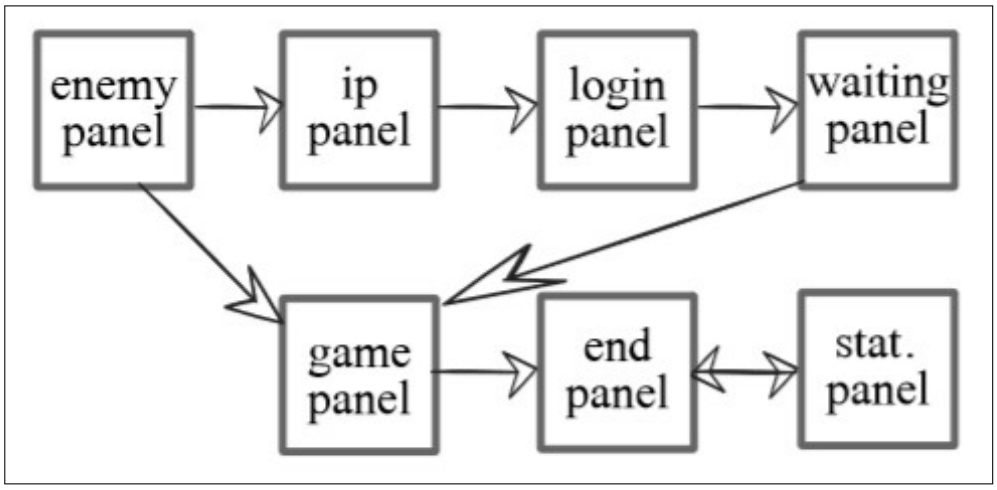
Po vytvoření serveru je potřeba vytvořit grafického klienta, který bude zobrazovat uživateli aktuální stav připojení a poté i průběh hry. Klient je opět jedna spustitelná aplikace. Grafická aplikace byla vytvořena pomocí *Swing*. Klient má několik částí, které má při svém běhu udělat. Je důležité, aby si klient vybral soupeře, proti kterému chce hrát, případně se připojil k serveru, přihlásil se ke svému uživatelskému jménu, počkal na připojení ostatních hráčů, odehrál celou hru a podíval se na výsledek hry a případně i statistiky všech hráčů.

## 5.2 Panely

Celou hru jsem si tedy rozdělila na jednotlivé panely, přesně podle jednotlivých fází průběhu této hry, přechod mezi panely lze vidět na obrázku 5.1. Panely se připraví hned při spouštění hry a přechod mezi nimi bude již jednoduchý, což se vyplatí především mezi poslední a předposlední obrazovkou, mezi kterými může uživatel přecházet, jak často chce. Panely se jednou připraví na začátku a nebude se při běhu program zdržovat předěláváním panelu. Pouze se vždy nastaví, na který panel se uživatel dívá v danou chvíli. Postupně se vždy přepnou všechny viditelné panely, někdy i podle volby uživatele. Jako první panel jsem nastavila *JScrollPane*, který zajistí, že se v okně při zmenšení pod danou velikost objeví scrollbary. Těmito scrollbary lze upravit, kterou část má uživatel zobrazenou v daném okně. Všechny herní panely přidávám do tohoto panelu.

### 5.2.1 Panely před hrou

Vzhledem k tomu, že server využívám především pro připojení kvůli hraní po internetu s jinými hráči, není potřeba se k němu zbytečně připojovat kvůli hře lokálně proti umělé inteligenci. Tedy důležitý je výběr protihráče umístit hned




Obrázek 5.1: Přechod mezi panely

po spuštění programu, aby se pokračovalo dále podle uživatelovy volby. Prvním nastaveným panelem je *EnemyPanel*. Na tomto panelu je výběr soupeře. Buď si uživatel vybere jiného hráče, nebo umělou inteligenci. V případě, že uživatel chce hrát s ostatními hráči, vybere první variantu a může volbu odkliknout, poté se pokračuje dalším panelem.

V druhém případě, hry proti umělé inteligenci, jsem přidala po zvolení této možnosti ještě zobrazení dodatečného boxu s výběrem konkrétního soupeře. Tato možnost se odkryje po zvolení právě možnosti **Umělá inteligence**, v druhém případě zůstane nezobrazená. Aktuálně jsem do programu přidala pouze jednu umělou inteligenci, ale tento box s výběrem soupeře lze snadno rozšířit přidáním další implementace umělé inteligence. Podrobněji popíšu umělou inteligenci v programu a jak jí rozšířit v sekci 5.3. Protože v tomto případě se uživatel nemusí připojovat k serveru, využiji rozdělení celého programu na jednotlivé panely a jednoduše zobrazím herní panel s hrou proti zvolenému agentovi.

Pokud uživatel zvolí první volbu, tedy hru proti jinému hráči přes server, musí se k tomu serveru nejdříve připojit. Pro připojení je vytvořený další panel ve třídě *IPPanel*. Zde se musí zadat IP adresa a port. Po jejich získání (ověřuji, že port je číslo) od uživatele vytvořím nový *Socket*, který slouží jako koncový bod komunikace v klientovi. Poté ho připojím k vytvořené adrese. Tento klient informuje server, že se jedná o herního klienta, který nespravuje databázi, to oznámí hodnotou **GAME** typu *TypeOfConnection*. Pokud se připojení podaří, nastavím další panel jako další fázi přípravy hry.

Další částí je, po připojení k serveru, přihlášení uživatele ke hře (k databázi). Pro přihlášení je tedy další speciální panel vytvořený ve třídě *LoginPanel*. Tento panel má opět pole pro zadání jména a hesla. Protože nechceme, aby se heslo zobrazovalo na obrazovce, použila jsem na zadávání hesla komponentu *JPasswordField*, která zajišťuje, že napsaný text jde upravit, ale nezobrazují se zadané znaky. Znak, který se má zobrazit místo zapsaných písmen, jsem nastavila pomocí metody **setEchoChar()** na $*$. V kapitole 3.2 jsem popsala komunikaci klienta se serverem i při přihlašování. Protože jsem použila řetězec *GAME* jako heslo pro spuštění hry, není dovoleno takové jméno uživatele. Řetězec se posílá kvůli tomu, že se vyčkává na připojení dalšího hráče, jméno je reprezentované řetězcem, proto je heslo řetězec. Při obdržení hesla pro spuštění hry se ukončí připojování klientů. Tudíž, pokud by bylo povoleno jméno *GAME*, ukončilo by se připojování hráčů předčasně. Jméno se poupraví a přidá se na konec náhodné číslo. Po přihlášení se na předpřipravený panel přidají všechna jména již připojených uživatelů. Jména jsou poslána ze serveru dle popsané komunikace. A dále se nastaví na obrazovku další panel, *WaitingPanel*.

Posledním panelem před samotnou hrou je tedy panel, kde hráči čekají na připojení jiných hráčů. Je potřeba, aby mimo obsluhy čekání na připojení dalších hráčů, program reagoval i na spuštění hry uživatelem. Program by nefungoval dobře, pokud by v daném vlákně v jedné metodě program cyklil, a měl by obsluhovat v jiné metodě spouštění hry. Pro čekání na nového protihráče jsem vytvořila nové vlákno, které má speciálně jediný úkol, čekat na informaci od serveru o novém hráči nebo o spuštění hry. V základním vlákně stále běží obsluha metody po stisknutí tlačítka, takže pokud uživatel spustí hru, nezávisle na čekajícím vlákně pošle tuto informaci serveru. Proto server posílá tuto informaci zpět, protože čekající vlákno čeká na tuto informaci. Na předpřipraveném herním panelu (*Ga-*

*mePanel*) se nastaví správný počet uživatelů pro hru. Aby se tato část dělala snadno, každý hráč má svůj panel s číslicemi a jménem. Pro nehrajícího hráče panel zneviditelním.

### 5.2.2 Panel hry

Další částí průběhu celého programu je samotný herní panel. Protože Abaku je hra hraná na desce 15x15, musela jsem samozřejmě začít herním plánem. Potřebuji umět rozlišit konkrétní políčko, na které uživatel chce položit svou číslici. Z komponent, které nabízí *Swing*, jsem zvolila *JLabel*, číslice jsou u jednotlivých uživatelů reprezentované pomocí *JButton*. Při zvolení dané číslice si může být uživatel jistý, že správně vybral číslici, protože kliknutí na tlačítko je viditelné. Pole potřebujeme na přidávání jednotlivých početních příkladů na políčka s určitým barevným pozadím. Tedy do pole se pouze ukládají znaky. Proto jsem zvolila *JLabel* pro reprezentaci jednotlivých políček.

Umisťování číslic do herní desky jsem naprogramovala pomocí *ActionListeneru* na jednotlivých tlačítkách pouze u číslic hráče, který na obrazovku kouká, a *MouseListeneru* na herních políčkách, tedy na label. Po kliknutí se zapamatuje, které tlačítko (a číslice) bylo zvoleno, a po kliknutí na pole se umístí text ze zvoleného tlačítka na zvolené políčko. Po pokusu o umístění se okamžitě kontroluje, zda byla daná číslice položena správně, na správné místo. Pokud ne, stejně jako v celém programu, použiji pro upozornění uživatele dialog. Pro zobrazování dialogu v celém programu používám třídu *JOptionPane* a její metodu **showMessageDialog()**, která zobrazí danou zprávu s titulkem a dle daného typu zprávy.

Číslice ostatních hráčů se taky zobrazují na panelu pomocí tlačítek, aby vypadaly stejně, ale nelze s nimi nic dělat. Každý hráč má svůj panel, na kterém jsou tedy umístěné jeho jednotlivé číslice a jméno. Dále jsem musela na obrazovku přidat aktuální bodový stav každého hráče. Toto skóre jsem umístila do levého horního rohu. Pro hru jsem přidala dvě možná tlačítka, jedno pro odeslání tahu, druhé pro výměnu číslic hráče. Komunikaci se serverem jsem rozebrala v obou případech již v kapitole 3.3.1. Pro strategii hráče je důležité, kolik číslic zbývá do konce hry. Tudíž i tento údaj je zobrazen na obrazovce. Pro hru proti umělé inteligenci jsem toto nastavení herního panelu nechala takto. Pro hru s jiným uživatelem jsem přidala i časomíru, aby hra plynula pravidelně.

Chtěla jsem, aby časomíra byla reprezentována graficky. Intuitivně, zelená barva bude reprezentovat zbývající čas a červená ten uplynulý. Chtěla jsem, aby se každá uběhnutá vteřina reprezentovala jako jeden dílek v časomíře. Na to jsem využila třídu *Timer*. Při vytváření nového časovače jsem nastavila metodu, která se má pravidelně provádět a přidávat v každém spuštění jeden dílek do *JProgressBaru*. Dále jsem nastavila, jak často se tato metoda bude provádět, čas jsem nastavila na 1000 milisekund. A časomíru jsem odstartovala. Vždy, když doběhne čas, odešle uživatel platný tah nebo vymění číslice, čas se vynuluje a běží znovu od začátku. Časomíra je spuštěna v jiném vlákně a neustále běží dokola, neřeší správnost tahů uživatele, počet bodů ani nic jiného.

Základní smyčka celé hry je v metodě *gameService*, která obdrží informaci o konci hry, případně, zda uživatel hraje, nebo ne, a podle toho spustí patřičnou metodu. Smyčka na stejné proměnné je i ve vlákně s běžícím časem. Potřebuji,

aby se vždy ve správnou chvíli vynuloval čas a běžel od začátku. Hra v klientovi je tedy rozdělená na dvě vlákna, v jednom běží čas a ve druhém se obsluhují tahy provedené hráčem. Bylo nutné, aby v případě, že uživatel nehraje, byly jeho číslice znepřístupněné a nemohl tak nic zahrávat. Když je alespoň jedna číslice umístěná na herní plochu, musí být znepřístupněné tlačítko pro výměnu číslic, aby uživatel nechtěl zahrávat příklad i výměnu zároveň. Na dvou místech v programu čekám, než se daná proměnná rovná požadované hodnotě. V takovém případě na daném místě použiji metodu *wait()* a v místě, kde měním danou proměnnou na požadovanou hodnotu, zavolám *notifyAll()*. Díky tomu se zajistí, že dané vlákno čeká, než bude probuzeno metodou a zbytečně se necyklí.

Kontrola správnosti příkladů byla nutná rozdělit do několika pomocných metod, které dlouhý složitý řetězec plný všech možných příkladů rozložily na několik výsledných příkladů a spočítaly za ně body. Nejdříve je potřeba najít celý souvislý úsek číslic na herním poli. Aby byl příklad správně, je potřeba zkontrolovat, zda je alespoň jeden příklad ve všech přiložených číslicích a včetně alespoň jedné číslice na obou krajích. Pokud takový příklad existuje, je správně a můžeme pokračovat spočítáním bodů. Z této posloupnosti číslic na herním poli vyberu všechny souvislé podposloupnosti číslic různých délek. Tyto podposloupnosti zkusím rozdělit všemi způsoby na dva operandy a výsledek nebo operand a výsledek početního příkladu. Zkusím vyzkoušet operace sčítání, odčítání, násobení, dělení, mocnění a odmocňování. Za každý takový příklad připočítám body k bodům za tento tah. Pokud je číslice na bonusovém poli, vynásobí se počet bodů za číslici, pokud je příklad na bonusovém poli pro příklad, vynásobí se na konci celý počet bodů za příklad tímto bonusem.

### 5.2.3 Panely po hře

Po konci hry jsou předpřipravené další dva panely. Prvním, který se zobrazí ihned, je *EndPanel*. Na tomto panelu jsem nechala vypsat jména seřazená dle počtu bodů a body daných hráčů. Dále jsem tam připravila dvě tlačítka, jedno má po kliknutí nastavené ukončení připojení klienta k serveru a vypnutí programu, druhé má nastavené po kliknutí přepnutí panelu na panel statistik.

Tlačítko pro ukončení je přidané i z toho důvodu, že jsem ihned po úspěšném přihlášení uživatele zakázala vypnutí celého programu po kliknutí na křížek u okna. Důvod pro zakázání byl jednoduchý, než se uživatel připojí do konkrétní skupiny, mohl aplikaci vypnout kdykoliv chtěl. Po připojení do skupiny a při běhu hry uživatel, i když nechce, musí hru dokončit buď aktivní nebo pasivní hrou. Tedy takový uživatel buď hraje dál, umisťuje číslice, nebo nechá hru běžet na pozadí a pouze nechává hru dohrát samotnou. Jeho tah bude vždy trvat maximální čas, bude prázdný, ale v pořádku se odešle i bez uživatele. Ten bude moct aplikaci vypnout až po skončení hry.

Posledním typem panelu v tomto programu je *StatisticsPanel*, který slouží k vypisování statistik z databáze. Opět je již předpřipravený od spuštění klienta, jen konkrétní jména uživatelů se tam přidají až při přepnutí na tento panel. Statistiky jsem vytvořila jako skupinu *JRadioButton*. Tedy v jednu chvíli lze vybrat pouze jednu statistiku. Opět jsem napsala obsluhu kliknutí na jednotlivá tlačítka. Podle vybraného tlačítka se ihned po kliknutí odešle zpráva serveru s typem statistiky. Ze serveru přijde zpráva s vyžádanou informací o hráči.

## 5.3 Umělá inteligence

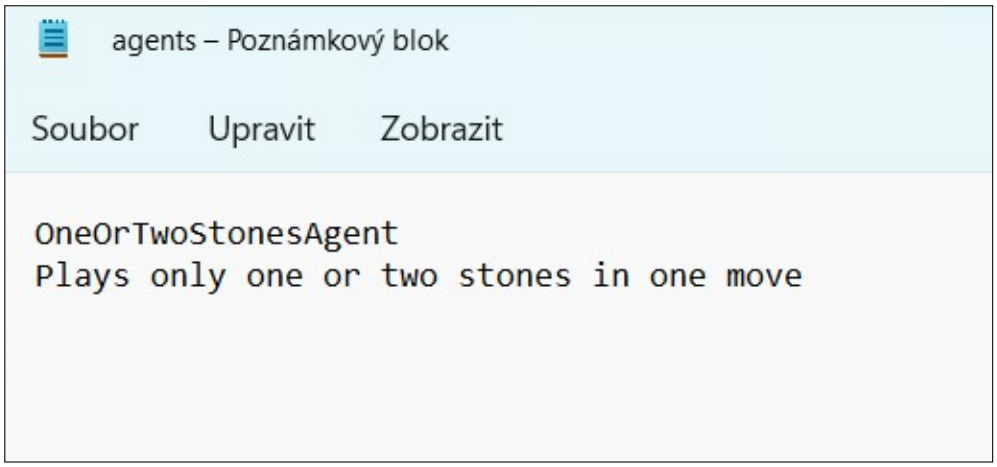
Poslední část programu, které jsem se ještě v této práci nevěnovala, je umělá inteligence. Bylo potřeba vytvořit agenta, který hraje sám podle své strategie. Vybírá vždy nejlepší možný tah, který poté zahraje. Do této hry jsem přidala pouze jednoho agenta, který hraje středně těžkou strategii.

V celém programu jsem všude, kde byla potřeba využít umělá inteligence, používala abstraktní třídu pro tuto umělou inteligenci. Abstraktní třída obsahuje jak implementované metody společné pro všechny agenty, tak i abstraktní metodu **play()**. Tuto metodu musí každý agent implementovat sám, protože se jedná o metodu, kde se obsluhuje hra a strategie agenta. Mezi společné metody patří spuštění hry (**startGame()**), kde se nastaví například počáteční kameny hráči i agentovi, dále **game()**, která se stará o běh hry, přepínání zpřístupnění a znepřístupnění panelu číslic hráče. Také mezi tyto metody patří obsluha výměny číslic, příprava koncového panelu po hře nebo počítání bodů. Všechny tyto metody mohou zůstat stejné, jediné, co se liší, je strategie, podle které bude daný agent hrát. Díky tomu, že využívám ve všech místech tuto abstraktní třídu, je velice snadné přidat do programu další implementace agentů. Do programu není nutné vůbec zasahovat.

Metodu **play()** jsem označila slovem abstract, což značí, že každá třída, která bude tuto abstraktní třídu rozšiřovat, musí obsahovat implementaci této metody. V této metodě se musí naprogramovat celá strategie hry, jakým způsobem bude daný agent zjišťovat svůj nejlepší možný tah nebo kolik číslic bude v daném tahu přikládat maximálně. Samozřejmě strategie pro hru Abaku mohou být různé. Mým cílem bylo především udělat snadno rozšiřitelnou nabídku agentů pro tuto hru.

Kdokoliv bude chtít přidat novou implementaci umělé inteligence, stačí, aby dědil od abstraktní třídy *ArtificialIntelligence* a implementoval, jak bylo popsáno výše, metodu **play()**. Jednotlivé implementace jsou do programu načítány jako pluginy. Tedy stačí, aby daná třída s novou implementací byla uložena v balíčku *cz.cuni.mff.java.abaku.klient*, kde se také daný agent hledá. Seznam všech agentů, které se vypisují do seznamu možností, se nachází v souboru *agents.txt*. Tento soubor pro každého agenta musí obsahovat dva řádky. První řádek je jméno souboru, druhý řádek je popis obtížnosti agenta nebo také popis, jak daný agent hraje. V tomto souboru se musí vyskytovat jen skutečně implementovaní agenti. Díky tomuto souboru je možné hráči přiblížit jednotlivé agenty. Můj soubor tedy vypadá tak, jak je zobrazeno na obrázku 5.2. Jak jsme si všimli, pluginy jsou snadný způsob, jak rozšířit již napsaný program o jinou funkcionalitu.

Moje implementace umělé inteligence se jmenuje *OneOrTwoStonesAgent*. Jak již říká tento název, je to agent, který umisťuje pouze jednu nebo dvě číslice v každém tahu. Tento agent vyzkouší všechny možnosti přiložení jedné a poté pouze k ní všemi způsoby druhé číslice. Ani v pokročilém stavu hry nebude počet možností tak velký, aby si agent nemohl vyzkoušet úplně všechny možnosti. Tedy vybrání úplně nejlepší tahu znamená, že zjistí u každého tahu počet bodů a vybere ten nejvyšší.




Obrázek 5.2: Soubor se seznamem agentů

# 6. Vyhodnocení

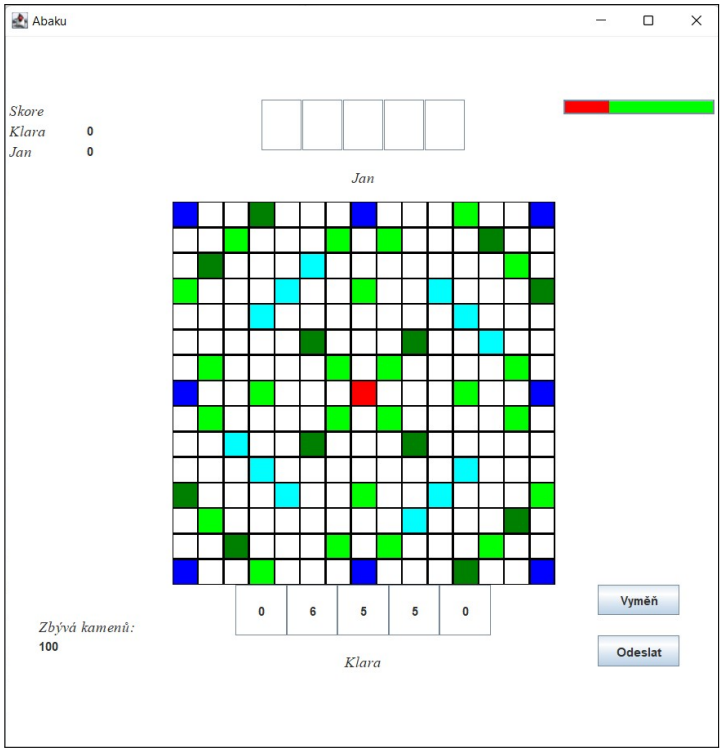
Vzhledem k tomu, že cílem bylo implementovat hru Abaku jako hru více klientů na serveru, myslím, že jsem ho úspěšně splnila. (Uživatelskou dokumentaci k této hře můžeme najít v příloze B.) Druhým typem hry je hra proti agentovi neboli umělé inteligenci. Hra byla rozdělena na několik herních panelů, které se mezi sebou přepínají dle typu hry. Jeden spustitelný soubor odpovídá serveru a druhý klientovi, který je na serveru nezávislý. Pokud si uživatel vybere hru proti jinému hráči, je nutné, aby byl server již spuštěný. První připojený klient k nově spuštěnému serveru musí při přihlašování počkat, než se vytvoří databáze na serveru. Ta se vytváří, až když jí někdo potřebuje, proto první přihlášení trvá malinko déle (cca 3 vteřiny). V čekací místnosti uživatelé vyčkávají na připojení dalších hráčů nebo na spuštění hry. Doba čekání není nijak omezená, tudíž hráč může čekat, jak dlouho chce.

Od chvíle, kdy se uživatel připojí do skupiny, už nemůže vypnout hru. Přijde mi to jako nejlepší řešení v případě, kdy se uživatel rozhodne přestat hrát. Nechtěla jsem ho mazat z herní desky všech hráčů i kvůli tomu, že ostatním hráčům by to nebylo příjemné. Dále rozhodnutí, jak by se v takovém případě počítaly jeho statistiky, by také nebylo jednoduché. Nebylo by fér psát mu prohru, když hru nedohrál z různých důvodů (i třeba kvůli kliknutí omylem nebo technickým problémům), a nezapsat mu prohru by znamenalo, že kdykoliv uživatel vidí, že prohraje, raději vypne hru. Rozhodla jsem se pro řešení, kdy uživatel hru dohrát nemusí (aktivně), pokud nechce, ale dává mu to možnost se ke hře vrátit v případě, že si to rozmyslí. Tedy hra běží i bez tahu uživatele, ten nemusí na nic klikat. Může dělat cokoliv jiného a pokud se rozhodne vrátit, hra stále pokračuje. Herní panel můžeme vidět na obrázku 6.1.

Zahrané příklady se nezobrazují nijak speciálně na obrazovce. Uživatel, který příklad zahrál, je ví, protože příklady vytvářel, a soupeřovi zas běží čas na jeho tah, proto ho nebudeme zdržovat a vyrušovat od přemýšlení. Pokud by to bylo velice žádané, nebyl by problém všechny příklady uživateli vypsat z příslušné části programu na obrazovku. Uživatel má několik možností, co zahrát. Vyměnit všechny číslice, vynechat tah a nezahrát nic nebo zahrát platný příklad, za který získá body. Po hře se na obrazovce ukáže výsledné skóre a je také možné zobrazit panel se statistikami jednotlivých hráčů. V této fázi hry je již možné program ukončit.

Mnou přidaná umělá inteligence se jmenuje OneOrTwoStonesAgent. Tento agent zahrává, jak říká název, v každém tahu jeden nebo dva kameny. Počet možností, které může zahrát není tak veliký, tudíž zkouší všechny takové možnosti. Každý jeho tah je provedený během vteřiny a uživatel může znovu hrát. Je tedy rychlý a navíc i velmi těžký soupeř. Ve většině případů porazí i velmi dobrého hráče. Tato hra není nijak časově omezená, tedy uživatel může rozmýšlet svůj tah, jak dlouho chce. Celá hra proti umělé inteligenci probíhá v klientovi a není nutné se kvůli ní nikam připojovat ani přihlašovat.

Jakákoliv chyba, která znamená špatně zadané údaje, se zobrazuje jako chybový dialog. Chyby, které způsobí například i vypnutí programu, se zapisují do souboru, který si uživatel může zobrazit. Na serveru je spuštěna databáze, pokud se server vypne, databáze se smaže z paměti a při opětovném spuštění serveru




Obrázek 6.1: Počáteční nastavení herní obrazovky

bude prázdná. Pokud bychom chtěli, aby databáze zůstala v paměti i po vypnutí serveru, museli bychom změnit nastaveni v konfiguračním programu databáze.

## 6.1 Porovnání s jinými implementacemi

Na internetu jsem při hledání nenašla příliš implementací této hry, ale s některými nalezenými mohu provést srovnání. Obě nalezené verze této hry, které budu popisovat, využívají techniku "drag and drop". Toto umožňuje, že se číslice může přesunout chytnutím a přetažením na správné místo. Také se v obou případech počítají přehledně body za jednotlivé příklady.

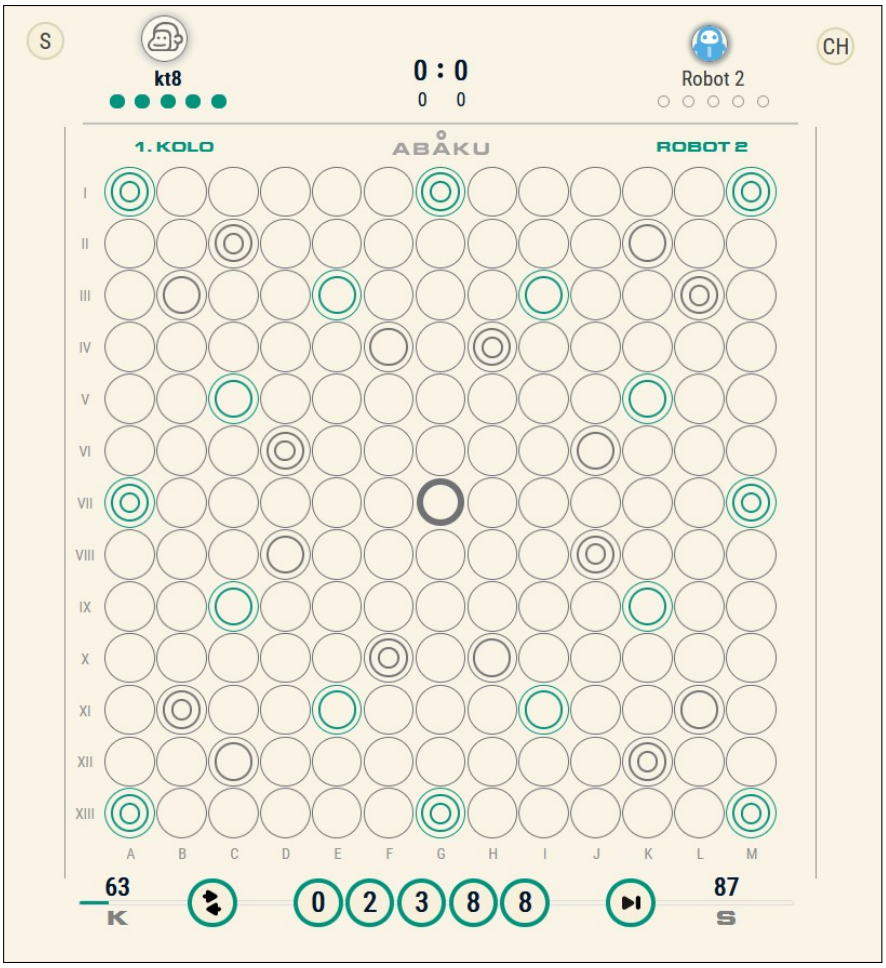
První takovou verzí hry je Abaku ze stránky seznam.cz. Tato hra již nelze spustit, ale podle toho, jak si ji pamatuji, umožňovala tato verze hru pouze proti jednomu hráči. Grafika herní desky je velice podobná té v mojí práci. Ukázku herní obrazovky můžeme vidět na obrázku 6.2. K této hře nebylo nutné se nijak přihlašovat, na konci hry se nevypisovaly žádné statistiky o jednotlivých hráčích získané z jeho všech her. Jak přesně probíhaly jednotlivé tahy, zda bylo v této implementaci možné hrát proti nějaké umělé inteligenci tréninkově už nemohu zjistit. Tato verze byla první hra, kterou jsem hrála, a díky ní jsem se inspirovala na vytvoření své vlastní verze.

Druhou verzí, kterou jsem na internetu našla je z abaku.org (b) a jmenuje se Abaku Play. Tato hra je určená veřejnosti, pro školy je speciální verze Lab ze stejné stránky. Vyžaduje se přihlášení uživatele na začátku. Uživatel má velký výběr, zda hrát proti jinému hráči, umělé inteligenci a může si i nastavit parametry hry: jak dlouho bude trvat jeden tah, kolik hráčů bude hrát, celkový počet kamenů ve hře, velikost herní plochy, a dokonce i které operace může uživatel


Obrázek 6.2: Ukázka herní plochy z seznam.cz

ve hře používat. Tato nabídka parametrů hry je tedy velice rozsáhlá. Herní plán je jednoduchý a decentní, označení bonusových polí není tak výrazné, jak jsem ho udělala ve své implementaci. V porovnání s touto verzí se mi líbí moje umělá inteligence, která hraje hodně dobře a velice rychle. Tato verze hry umožňuje výměnu jen vybraných číslic. V průběhu hry je možné si rozkliknout různé statistiky obou hráčů v jejich společné hře. Ukázka herní plochy je na obrázku 6.3. Tato implementace je daleko rozsáhlejší než moje díky tomu, že umožňuje spoustu různých nastavení. Ta však nejsou v deskové verzi hry možná, proto jsem je při implementaci nebrala v potaz.

Moji verzi hry je snadné, bez přílišného zásahu, upravit na různý počet číslic na začátku hry, dokonce i na různý počet od jednotlivých typů číslic. Do hry lze snadno přidat i další implementaci umělé inteligence. Také by bylo možné povolit, na žádost uživatele, jen některé početní operace tak, jak tomu je v druhé verzi hry. Moje verze oproti výše popsaným nevyužívá techniku "Drag and drop". I když je možná přirozenější číslici přetáhnout z jednoho místa na druhé, dle mého názoru je snazší dvakrát kliknout, nejdříve na zdrojovou číslici, poté na cílové políčko.




Obrázek 6.3: Ukázka herní plochy z abaku.org (b)

# Závěr

Tato práce se zabývala implementací deskové početní hry Abaku. Cílem bylo vytvořit server, ke kterému se budou moct uživatelé připojovat a hrát na něm. Na serveru také běží databáze, ve které jsou uložené různé statistiky jednotlivých hráčů. Máme informaci jak o celkovém počtu her, tak o těch vyhraných, průměrném počtu bodů z jednotlivých her a maximálním počtu bodů. Tato práce také obsahuje hru proti umělé inteligenci, která je brána jako tréninková hra. Není potřeba se kvůli ní připojovat k serveru, na tuto hru je neomezeně času, aby uživatel mohl v klidu vyzkoušet tvoření i složitějších příkladů. Agenta, proti kterému chce uživatel hrát, si vybere na začátku hry. Dále práce obsahuje program pro správu databáze, díky kterému lze získané statistiky ze hry upravovat, dále mazat uživatele nebo naopak uživatele přidat. Práce se nezaměřuje na grafiku hry, ale především na možnost hrát hru přes server a moct spravovat databázi.

Cílem bylo také udělat základní implementaci umělé inteligence a její snadnou rozšiřitelnost. Součástí práce je naprogramovaná pouze jedna umělá inteligence. V tomto ohledu jde práce velice dobře rozšířit. Umělá inteligence se do programu načítá jako plugin, tudíž každý nově naprogramovaný agent lze do programu snadno přidat bez přílišného zásahu do programu. Stačí přidat do seznamu agentů tohoto nového agenta a jeho jednoduchý popis, načtení je již naprogramováno. Tento nový agent by měl rozšiřovat abstraktní třídu *ArtificialIntelligence* a tedy implementovat abstraktní metodu **play()**. Takže je možné přidat do práce agenty, které budou hrát hůře nebo naopak ještě lépe.

# Seznam použité literatury

ABAKU.ORG. Pravidla hry. URL https://abaku.org/abaku-lab-pravidla-hry.

ABAKU.ORG. Abaku play. URL https://abakuplay.com/.

DAVISON, A. (2005). *Killer Game Programming in Java*. První vydání. O'REILLY, USA. ISBN 978-0-596-00730-0.

EFKO (2012). Pravidla hry abaku. desková hra.

JAVATPOINT. Hibernate tutorial. URL https://www.javatpoint.com/hibernate-tutorial.

ORACLE. Api documentation. URL https://docs.oracle.com/en/java/javase/18/docs/api/index.html.

SEZNAM.CZ. Abaku. URL https://hry.seznam.cz/hra/abaku.

# Seznam obrázků

# A. Soubory pro konfiguraci databáze

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class">org.h2.Driver</property>
    <property name="hibernate.connection.url">jdbc:h2:mem:test</property>
    <property name="hibernate.connection.username">sa</property>
    <property name="hibernate.connection.password">sa</property>
    <property name="hibernate.dialect">org.hibernate.dialect.H2Dialect</property>
    <property name="show_sql">false</property>
    <property name="hbm2ddl.auto">update</property>
    <mapping class="cz.cuni.mff.java.abaku.server.DatabasePlayer" resource="players.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Obrázek A.1: Soubor pro konfiguraci databáze

```
<?xml version='1.0' encoding='UTF-8'?>
    <!DOCTYPE hibernate-mapping PUBLIC
     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
      <class name="cz.cuni.mff.java.abaku.server.DatabasePlayer" table="PLAYERS">
        <id name="id" type="integer" column="ID">
         <generator class="increment"></generator>
        </id>

        <property name="nickname"></property>
        <property name="password"></property>
        <property name="avgPoints"></property>
        <property name="maxPoints"></property>
        <property name="wins"></property>
        <property name="games"></property>

      </class>

</hibernate-mapping>
```

Obrázek A.2: Soubor s mapováním

# B. Uživatelská dokumentace

V této kapitole se věnujeme uživatelské dokumentaci počítačové hry Abaku. Program ke svému běhu potřebuje nainstalovanou Javu 15 nebo vyšší (lze stáhnout na www.oracle.com). Pokud máte javu na svém počítači nainstalovanou, můžete spouštět JAR soubory, pokud ne, jsou přiloženy EXE soubory. Hra se skládá ze 3 spustitelných programů. Abaku-server.jar (exe) je program pro spuštění serveru. Abaku-client.jar (exe) obsahuje uživatelské rozhraní klienta. Abaku-manage.jar (exe) komunikuje s databází a vypisuje z ní informace. Program umožňuje dvě základní varianty hry. První možnost je hra proti ostatním uživatelům po připojení k serveru, druhá možnost je hraní proti umělé inteligenci. Součástí je pro správce databáze program, který komunikuje s databází, vypisuje uživatele, umožňuje je smazat apod. Všechny chyby, které nastanou, jsou zapsané v textových souborech odpovídajících spuštěnému programu.

## B.1 Základní pravidla hry

Tato hra se zakládá na vytváření početních příkladů. Každý hráč má svůj zásobník číslic a ve svém tahu přikládá na herní desku číslice a vytváří z nich příklady. Za každý příklad dostane hráč určitý počet bodů, které se připočítají k celkovému počtu jeho bodů. Vítězem je hráč s nejvyšším počtem bodů za celou hru. Herní pole je klasická deska 15x15. Na desce se vyskytují pole různé barvy. Modrá políčka značí násobení bodů za celý příklad. Zelená políčka označují násobení počtu bodů za danou číslici. Tmavší odstín představuje násobení třemi, světlejší násobení dvěma. Červené pole je počáteční pole pro první položený příklad ve hře. Bílá políčka jsou všechna ostatní, obyčejná pole. Okolo herního pole jsou umístěni hráči. Do zásobníků jednotlivých hráčů ostatní hráči nevidí. Po celou hru se udržuje informace o zbývajících číslicích v balíčku, které nejsou rozlosované hráčům. Hráči se střídají postupně dokola, každý hráč má několik možností, co může zahrát. Vyložit platný početní příklad:

- Všechny číslice se musí přikládat buď do jednoho sloupce, nebo do jednoho řádku.
- Pokud je to první vyložený příklad, musí využít červené políčko.
- Jinak musí vyložený příklad využívat nějakou již dříve vyloženou číslici. S touto číslicí musí být všechny vyložené číslice součástí správného početního příkladu.
- V jednom vyloženém příkladě může být více dalších příkladů.
- Každá číslice musí být se všemi sousedními políčky součástí nějakého početního příkladu.

Počet bodů za daný příklad odpovídá hodnotě číslic v daném příkladu. Případně se násobí dvěma nebo třemi podle případného bonusového pole pod příkladem. Počítají se body za všechny nově vytvořené příklady. Po odeslání tahu se barevná pole přebarví na bílá a tedy jsou již bez bonusových bodů. Další možností je

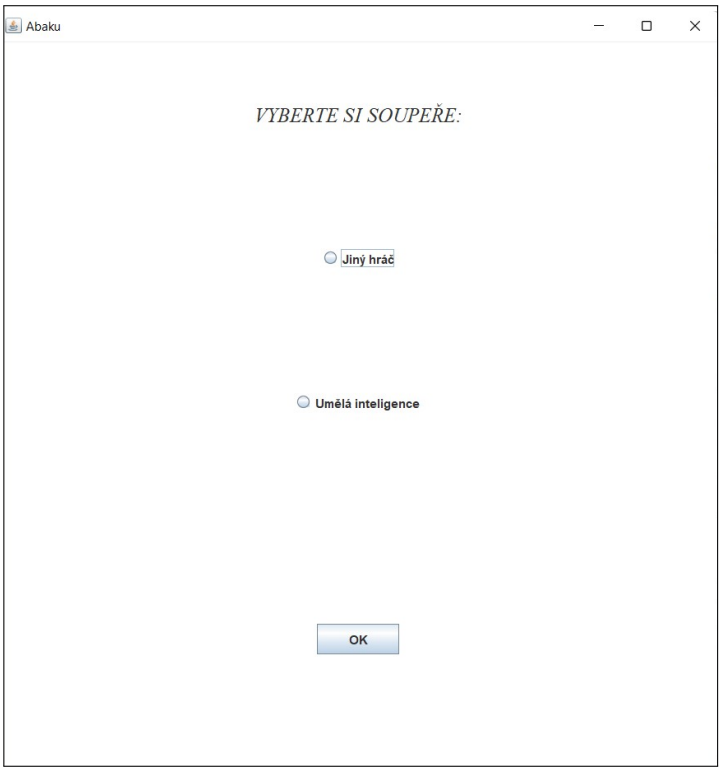
výměna všech kamenů, které má uživatel ve svém zásobníku. Tím končí tah tohoto uživatele. Pokud uživatel nestihne zahrát v daném časovém limitu žádný korektní příklad nebo si nevymění číslice, tah také končí a uživateli zůstanou stejné číslice v zásobníku. Poté hraje další hráč. Hra končí, pokud nikdo z hráčů nezahraje platný početní příklad (tedy pokud všichni vymění své číslice nebo nestihnou zahrát žádný tah). Hráč s nejvyšším počtem bodů vyhrává. V případě rovnosti bodů vyhrává hráč připojený ke hře dříve. Podrobnější pravidla lze najít v kapitole 1.

## B.2 Příprava hry na serveru

Aby se jednotliví uživatelé mohli připojovat ke hře je potřeba nejdřív spustit server. V souboru konfigurace.properties je nastaven port, na kterém bude server poslouchat. Implicitně je nastaven port 1. Pokud chcete nastavit jiný port, stačí ho v souboru přepsat. Poté je potřeba spustit server. K tomu slouží aplikace Abaku-server.jar (exe). Na Windows 11 si stačí otevřít příkazový řádek a tam přejít do správného adresáře a tuto aplikaci spustit jako *java -jar Abaku-server.jar konfigurace.properties (Abaku-server.exe konfigurace.properties)*. Aplikace potřebuje argument, který specifikuje soubor s informací o portu, proto programu předáváme náš soubor konfigurace.properties. Pokud bude program spuštěn správně, uvidíme, že neustále běží a nic nevypisuje. Máme tedy správně spuštěný server.

## B.3 Připojení klienta k serveru

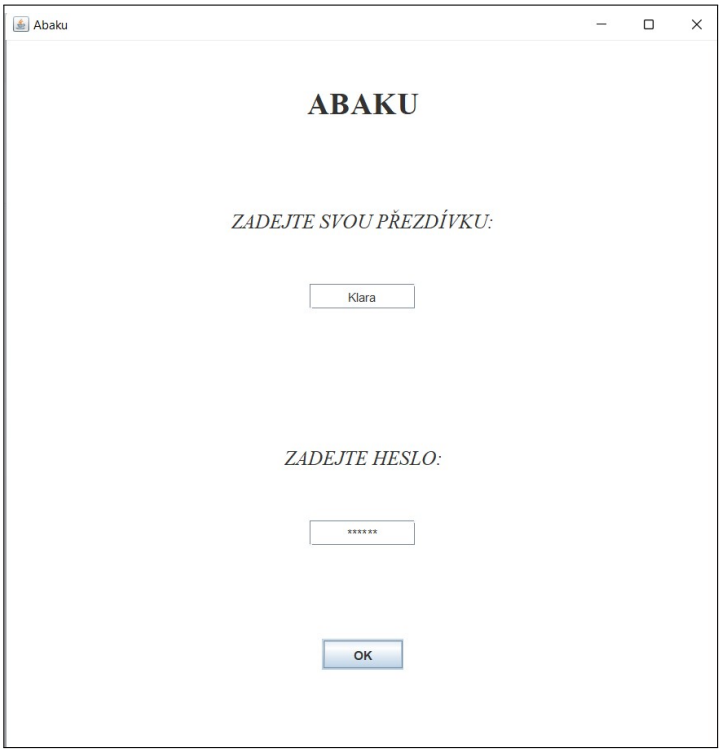
Pro připojení klienta si musíme spustit aplikaci Abaku-client.jar (exe). Postačí dvojité kliknutí na tento soubor. Spustí se nám obrazovka dle ukázky na obrázku B.1. My se chceme připojovat k serveru za účelem hry s jiným hráčem, zvolíme tedy možnost *jiný hráč* a potvrdíme výběr tlačítkem *OK*. Druhou možnost rozebereme později (B.6). Zobrazí se další obrazovka, vidíme ji na obrázku B.2, která vyžaduje informace o serveru, ke kterému se chceme připojit. Port známe z konfigurace.properties. IP adresu serveru lze snadno zjistit a pokud jsme spouštěli server na stejném PC jako klienta, stačí napsat do řádku IP adresy *localhost*. Pokud jsou obě informace zadané správně, zobrazí se nám další obrazovka, pokud ne, zobrazí se dialog oznamující, jaká chyba nastala, viz obrázek B.3. Další obrazovku lze vidět na obrázku B.4. Tato obrazovka si žádá přihlášení uživatele kvůli hře. Pokud se uživatel přihlašuje poprvé, stačí zadat jméno a jakékoliv heslo, heslo se spáruje s tímto jménem a při příštím přihlašování pod tímto jménem se bude vyžadovat toto heslo. Není dovoleno prázdné jméno ani prázdné heslo. Pokud bude zadáno při příštím přihlašování špatné heslo, zobrazí se chybová hláška jako je na obrázku B.5. Po úspěšném přihlášení se zobrazí čekací místnost, kde uživatel čeká na připojení ostatních hráčů. První připojený z dané skupiny má na panelu tlačítko pro spuštění hry. On určí, zda se začne hrát hra ve dvou, ve třech nebo ve čtyřech hráčích. Tedy zda budou první dva připojení hráči čekat na další, nebo jdou hrát ve dvou. Pokud je uživatel připojený sám, nemůže začít hru, vyskočí mu v takovém případě chybová hláška. Čekací panel je vidět na obrázku B.6. Hru již uživatel nemůže ukončit, musí ji dohrát. Nebo případně jí prostě nechat doběhnout samotnou.



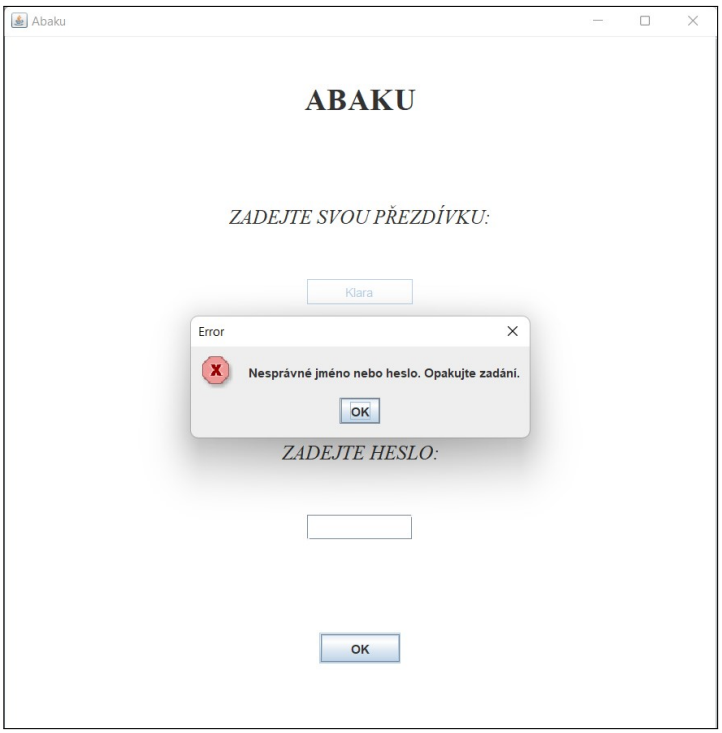

Obrázek B.1: Obrazovka pro vybrání soupeře


Obrázek B.2: Obrazovka pro zadání IP adresy a portu

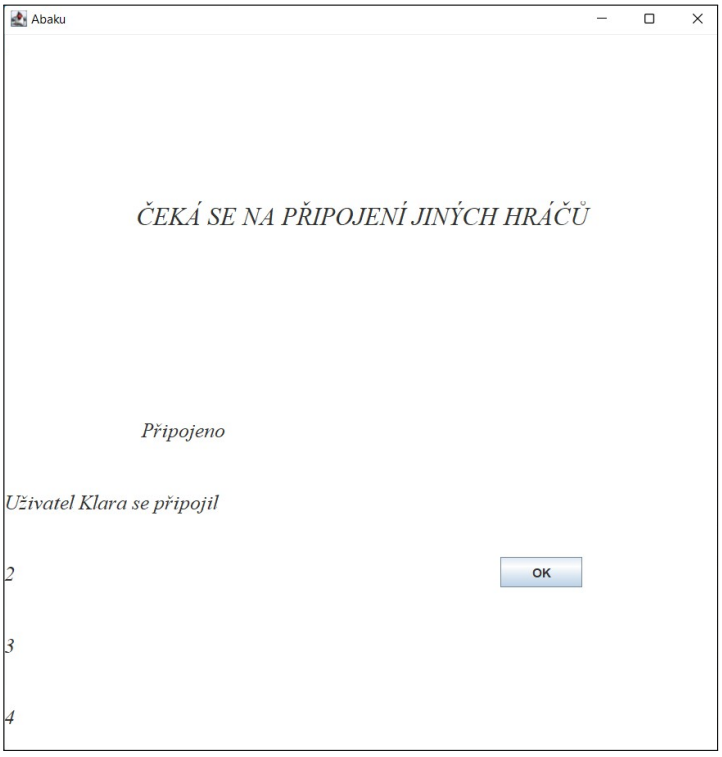

Obrázek B.3: Chybně zadaná IP adresa a port




Obrázek B.4: Obrazovka pro přihlášení k serveru




Obrázek B.5: Chybně zadané heslo




Obrázek B.6: Čekací panel s prvním připojeným hráčem

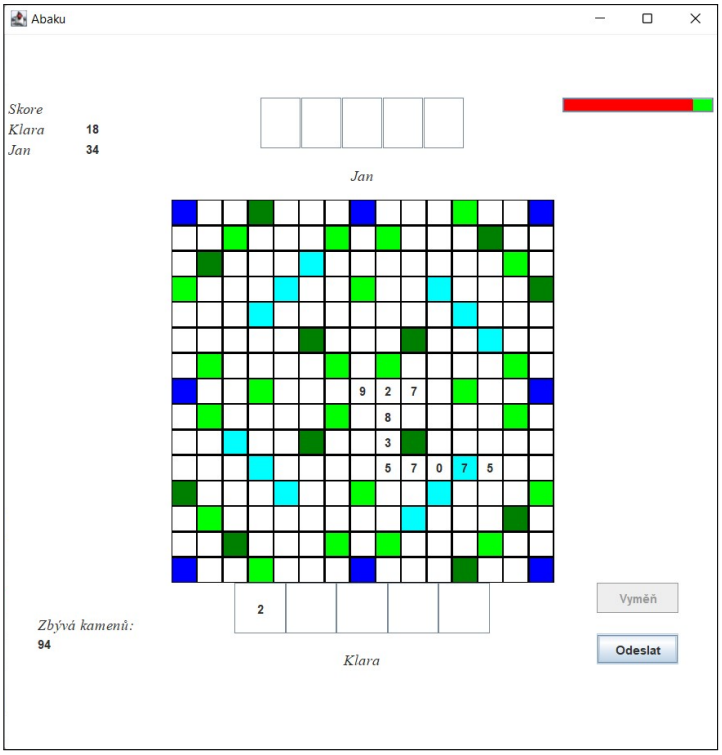
## B.4 Průběh hry na serveru

Uživatel spustil hru a objevila se obrazovka s herní deskou. Na obrázku 6.1 můžeme vidět hru mezi dvěma hráči. Hráč, který na obrazovku kouká, má své číslice dole, soupeř je nahoře. V případě 4 hráčů má vždy daný hráč číslice dole, soupeře, který hraje po něm, nahoře, dalšího vlevo a poslední hráč je vpravo. Také si můžeme všimnout rozložení barevných políček. Vpravo nahoře běží časomíra, která odpovídá 30 vteřinám. Po vypršení času končí tah daného hráče. Vlevo nahoře je vždy aktuální bodový stav všech hráčů ve hře. Vlevo dole najdeme zbývající počet ještě nevylosovaných číslic. Vpravo dole se nachází dvě tlačítka. Jedno pro výměnu všech kamenů uživatele, druhé pro odeslání právě položeného příkladu. Hráč, který je na řadě, má číslice černě, může na ně kliknout a umisťovat je do herního pole. Naopak hráči, kteří nejsou na řadě, mají číslice ve svém zásobníku šedé.

Hráč na řadě má dle pravidel několik možností. Vyměnit své číslice kliknutím na tlačítko *Vyměň*, číslice se vrátí mezi ostatní nevylosované a ze všech těchto číslic se vylosují nové číslice pro uživatele. Druhou možností je nechat doběhnout čas a pokud nepoložil žádné číslice, rovnou hraje soupeř. Pokud hráč vyložil příklad a nechal doběhnout čas, příklad se vyhodnotí úplně stejně jako při stisknutí tlačítka *Odeslat*. V obou případech vyskočí na uživatele chybová hláška, pokud příklad není správný nebo nesplňuje jinou podmínku z pravidel. Pokud je příklad správně, uživateli se připočtou body do jeho aktuálního celkového bodového zisku.

Teď si popíšeme způsob, jak vyložit platný příklad. Je nutné, aby uživatel vždy přikládal své číslice vedle již dříve přiložené číslice. Také se ověřuje, zda hráč nevyložil číslice do více sloupců a řádků zároveň. Vykládání probíhá jednoduše, uživatel klikne na číslici, kterou chce umístit, a klikne na pole na herním plánu, kam ji chce umístit. Číslice se přenese ze zásobníku na dané pole. Jeden takový tah můžeme vidět na obrázku B.7. Číslice bylo nutné přiložit v pořadí: 7, 0, 7, 5. Tedy vždy vedle již vyložené číslice. V zásobníku hráče zůstávají prázdná pole po vyložených číslicích. Po odeslání tahu tlačítkem nebo vypršením času se uživateli automaticky doplní nové číslice do zásobníku. Pokud jich již není dostatek na doplnění, odstraní se okénka pro číslice u daného hráče, aby odpovídala počtu číslic v zásobě. Můžeme tedy ke konci hry pozorovat, kolik má kdo ještě číslic v zásobníku. Po odehrání tahu je na řadě hráč naproti. Ten opět pozná, že je na řadě podle barvy jeho číslic. Když hráč není na řadě, nemůže umisťovat číslice do herního plánku.

Jak můžete také vidět na obrázku B.7, po přiložení číslice do herního plánu zešedne tlačítko na výměnu číslic. To proto, že nelze vykládat příklad a k tomu si vyměnit číslice. Pokud ale zahrajete chybný příklad a chcete zahrané číslice opravit nebo je úplně vyměnit, stačí kliknout na tlačítko pro odeslání tahu. Poté vyskočí na obrazovku informace, že byl zahrán špatný příklad, ale všechny číslice se vrátí zpět do zásobníku. Pokud je tedy ještě dostatek času, můžeme vyložit správný příklad nebo číslice vyměnit.



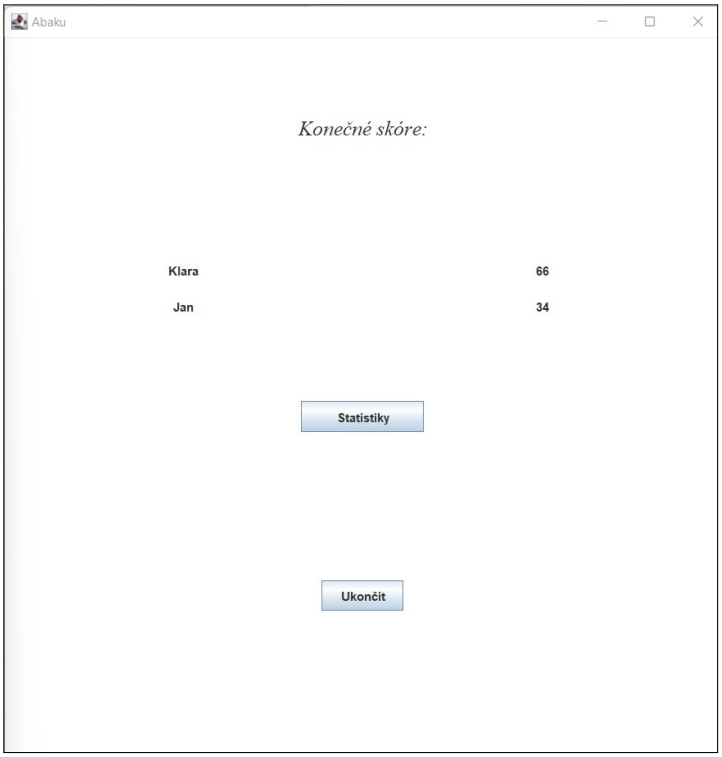

Obrázek B.7: Příklad jednoho možného tahu (přiložené číslice 7075)

## B.5 Konec hry na serveru

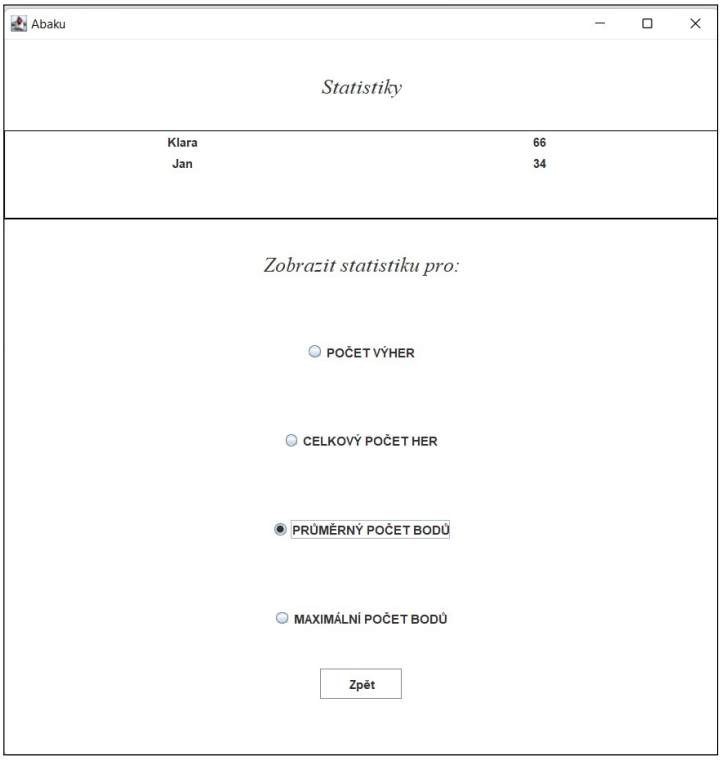
Pokud nastane konec hry, všem hráčům se zobrazí koncová obrazovka. Na ní jsou seřazeni jednotliví hráči dle získaných bodů. Hráč na nejvyšší příčce je vítěz. Příklad takové obrazovky ukazuje obrázek B.8. Na obrazovce s koncem hry se nachází dvě tlačítka. Tlačítko *Ukončit* vypíná celou aplikaci. Tlačítko *Statistiky* přepíná obrazovku na panel statistik, který vidíme na obrázku B.9. Na panelu statistik je potřeba vybrat jednu z nabízených statistik, teprve pak naskočí jména hráčů a odpovídající hodnoty. Na výběr je počet výher, celkový počet her, průměrný počet bodů, maximální počet bodů získaný v nějaké z hráčových her. V horní části jsou jména hráčů dle pořadí z této hry a hodnoty odpovídající zvolené statistice. Na obrazovce se také nachází tlačítko *Zpět* pro vrácení se na předchozí obrazovku.

## B.6 Hra proti UI

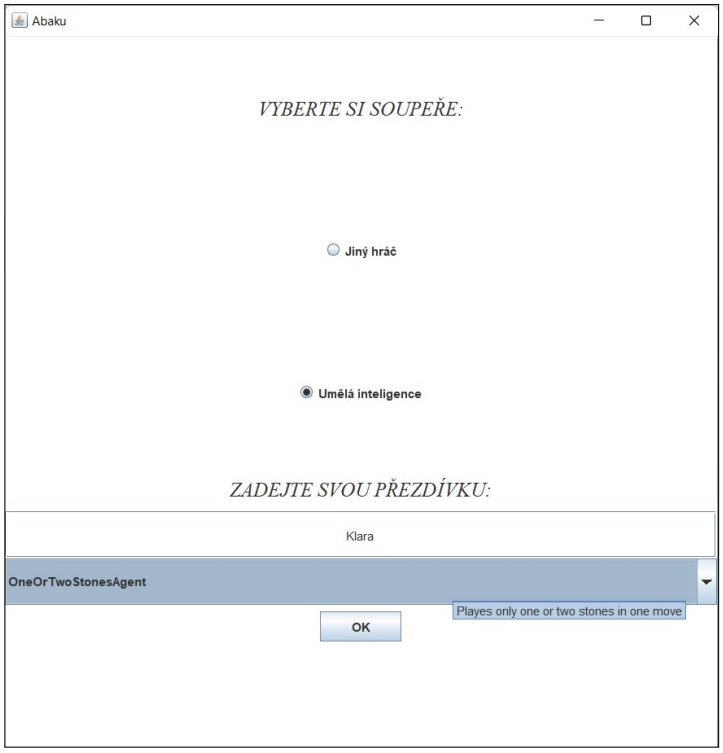
V sekci B.3 jsme si popsali hru proti ostatním uživatelům, kteří se připojují ke stejnému serveru. V této sekci si popíšeme hru proti UI. Spustíme stejný soubor jako předtím, ale liší se výběr hned na první obrazovce. Tentokrát vybereme druhou možnost *Umělá inteligence*, viz obrázek B.10. Po kliknutí na druhou možnost se zobrazí výzva pro zadání přezdívky a výběr konkrétního soupeře. Přezdívku zadáme do okénka, pod tím si kliknutím vybereme soupeře. Soupeř *OneOrTwoStonesAgent* je specifický tím, že v každém tahu zahrává pouze jednu nebo dvě číslice. Tedy nikdy nezahrává složitější příklady z více číslic, ale i tak hraje velmi dobře. Tuto informaci o agentovi nalezneme na obrazovce po najetí myší na ná-




Obrázek B.8: Obrazovka konce hry




Obrázek B.9: Obrazovka se statistikou průměrného počtu bodů




Obrázek B.10: Obrazovka s výběrem soupeře - UI

zev agenta. Pokud jsme s přezdívkou a soupeřem spokojeni, klikneme na tlačítko *OK*. Spustí se hra stejně jako v případě hry proti jiným hráčům. Jméno soupeře je jméno zvoleného agenta a jediný rozdíl ve vzhledu oproti obrázku 6.1 je v zobrazování času. Hra proti agentovi je braná jako trénink, proto odtud zmizela časomíra a je možné si tah rozmýšlet tak dlouho, jak je potřeba. Tedy nikdy nevyprší čas na tah a jediné dva možné tahy jsou zahrát příklad nebo vyměnit číslice. Nelze vynechat tah. Hra probíhá opět střídáním tahů hráče a agenta. Konec hry nastane opět za stejných podmínek. Uživateli se zobrazí konečná obrazovka, tentokrát bez statistik. Hráč není připojený k serveru a hraje pouze tréninkovou hru proti agentovi, tyto body se do statistik nezapočítávají. Po celou dobu lze hru ukončit křížkem v pravém horním rohu.

## B.7 Správa databáze

Pro získání informací z databáze o jednotlivých uživatelích je součástí program Abaku-manage.jar (exe). Po spuštění z příkazového řádku *java -jar Abaku-manage.jar (Abaku-manage.exe)* naskočí výzva k zadání IP adresy serveru a portu, na kterém server s databází naslouchá. IP adresu zadáváme úplně stejně jako v případě připojování hráče k serveru. Můžeme opět zadat *localhost* a port je dle souboru *konfigurace.properties*, v tomto případě dle obrázku B.11 je port 1. Pokud proběhne připojení v pořádku, naskočí, stejně jako na obrázku, výzva k zadání příkazu. V opačném případě je na obrazovce oznámena chyba připojení. Uživateli se nabízí několik možností, co může s databází udělat.

- *DELETE* - Tento příkaz slouží k vymazání jakéhokoliv uživatele z databáze. Po zapsání tohoto příkazu se program zeptá na jméno uživatele. Pokud

zadáme jméno z databáze, uživatel se smaže, pokud ne, databáze se nijak nezmění.

- *SELECT* - Tento příkaz vypíše uživatele, kterého uživatel požaduje. Program se zeptá na jméno hráče k vypsání. Pokud je dané jméno v databázi, vypíšou se jeho statistiky a informace o něm, pokud ne, objeví se hláška: *Tato osoba není v databázi.*

- *DELETE_ALL* - Slouží k vymazání všech hodnot z databáze. Pokud v databázi nic není, nijak se nezmění, v opačném případě se smažou všechna data. Nevyžaduje žádné dodatečné informace.

- *RENAME* - Slouží k přejmenování vybraného uživatele z původního jména na nové. Zeptá se uživatele na původní jméno daného hráče. Pokud žádný takový hráč není v databázi, opět se zobrazí hláška: *Tato osoba není v databázi.* Pokud takový hráč existuje, zeptá se program na nové jméno uživatele.

- *LIST_ALL* - Vypisuje všechny uživatele z databáze i s jejich informacemi a statistikami. Pokud není žádný hráč v databázi, vypíše se hláška: *Žádný hráč není v databázi.*

- *EDIT* - Tento příkaz zajišťuje změnu údajů u jednotlivých hráčů v databázi. Nejdříve se zeptá na jméno uživatele, kterému chceme změnit údaje v databázi. Poté zjišťuje od uživatele jednotlivé údaje: nové heslo, maximální počet bodů, průměrný počet bodů, počet odehraných her, počet výher. Mohou se zanechat stávající údaje, pokud necháme prázdný řádek. Pokud uživatel s daným jménem v databázi neexistuje, vytvoří se.

- *END* - Příkaz ukončující práci s databází. Odpojí se od serveru s databází, program tím skončí.

Vybraný příkaz stačí zapsat stejně, jako je napsán na obrazovce. Příklady některých operací jsou vidět na obrázku B.11.

```
Address of server with database:
localhost
Port:
1
Zvolte příkaz: DELETE, SELECT, DELETE_ALL, RENAME, LIST_ALL, EDIT, END
LIST_ALL
Přezdívka: Klara, id: 1, průměrný počet bodů: 499, maximální počet bodů: 741, počet vítězství: 2, celkový počet her: 3
Přezdívka: Jan, id: 2, průměrný počet bodů: 496, maximální počet bodů: 744, počet vítězství: 1, celkový počet her: 3

Zapiště další příkaz (DELETE, SELECT, DELETE_ALL, RENAME, LIST_ALL, EDIT, END):
SELECT
Zapište jméno osoby, kterou chcete vypsat.
Klara
Přezdívka: Klara, id: 1, průměrný počet bodů: 499, maximální počet bodů: 741, počet vítězství: 2, celkový počet her: 3

Zapiště další příkaz (DELETE, SELECT, DELETE_ALL, RENAME, LIST_ALL, EDIT, END):
DELETE
Zapište jméno osoby, kterou chcete smazat.
Jan

Zapiště další příkaz (DELETE, SELECT, DELETE_ALL, RENAME, LIST_ALL, EDIT, END):
RENAME
Zapište starou přezdívku, kterou chcete změnit.
Klara
Zapište novou přezdívku pro danou osobu:
KLARA

Zapiště další příkaz (DELETE, SELECT, DELETE_ALL, RENAME, LIST_ALL, EDIT, END):
LIST_ALL
Přezdívka: KLARA, id: 1, průměrný počet bodů: 499, maximální počet bodů: 741, počet vítězství: 2, celkový počet her: 3

Zapiště další příkaz (DELETE, SELECT, DELETE_ALL, RENAME, LIST_ALL, EDIT, END):
END
```

Obrázek B.11: Příklad operací s databází

# C. Obsah přiložených souborů

V souboru README.txt lze nalézt stejný popis přiložených souborů jako v této příloze. Dále se v něm také vyskytuje detailní popis spouštění programu. Součástí práce jsou mimo tohoto textu také další soubory.

Struktura přiloženého adresáře:

**Složky:**

- doc - Adresář s vygenerovanou dokumentací z programu. Nejjednodušeji si ji zobrazíme rozkliknutím souboru index.html. Otevře přehledně v internetovém prohlížeči přehled balíčků, na které lze kliknout a dále je prozkoumávat.
- jre - Poskytuje knihovny a další soubory pro spouštění EXE souborů vytvořených v Javě na cílovém počítači. Tedy slouží pro spuštění programu na těch zařízeních, která nemají nainstalovanou správnou verzi Javy.
- src - Obsahuje ve složce main/java jednotlivé balíčky s programem a ve složce main/resources lze nalézt konfigurační soubory pro vytvoření databáze, soubory hibernate.cfg.xml a players.hbm.xml.

**Soubory:**

- README.txt - Obsahuje stejné informace jako tato příloha, součástí je navíc návod na spuštění všech tří částí programu.
- Abaku-client.exe - Aplikace pro klienta, tento klient se připojuje k serveru, jedná se o aplikaci s grafickým rozhraním. Lze spustit i na počítači bez Javy.
- Abaku-client.jar - Aplikace pro klienta, tento klient se připojuje k serveru, jedná se o aplikaci s grafickým rozhraním. Lze spustit na počítači s nainstalovanou Javou.
- Abaku-manage.exe - Aplikace pro správu databáze. Připojuje se k serveru a obsluhuje jednotlivé operace s databází. Lze spustit na počítači bez Javy.
- Abaku-manage.jar - Aplikace pro správu databáze. Připojuje se k serveru a obsluhuje jednotlivé operace s databází. Lze spustit pouze na počítači s nainstalovanou Javou.
- Abaku-server.exe - Aplikace pro vytvoření serveru, ke kterému se připojují jednotliví klienti. Lze spustit i bez nainstalované Javy.
- Abaku-server.jar - Aplikace pro vytvoření serveru, ke kterému se připojují jednotliví klienti. Lze spustit pouze s nainstalovanou Javou.
- agents.txt - Textový soubor se seznamem možných agentů pro hru a popisem, jak daní agenti hrají. Popis by měl být výstižný, protože se zobrazuje jako nápověda na panelu pro výběr soupeře.
- konfigurace.properties - Soubor s konfigurací portu pro spuštění serveru. Na daném portu server čeká na připojení všech klientů. Tento port musí zadat klienti pro úspěšné připojení ke hře.

- pom.xml - Soubor s informacemi o projektu a konfiguračními detaily, které použije Maven při sestavování projektu.
- pom.xsd - Schéma, které popisuje pom.xml.
- hibernate-configuration-3.0.dtd - Popis schématu vytvořeného konfiguračního souboru hibernate.cfg.xml.
- hibernate-mapping-3.0.dtd - Popis schématu vytvořeného players.hbm.xml.
- Programátorská_dokumentace.pdf - Programátorská dokumentace s popisem jednotlivých tříd a jejich důležitých metod a atributů. Vygenerovaná dokumentace z programu se nachází v adresáři *doc*.