VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

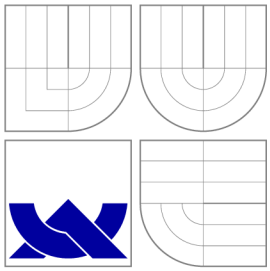
MANAŽER PRO DESKOVÉ HRY S PARALELNÍM HRANÍM TURNAJŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

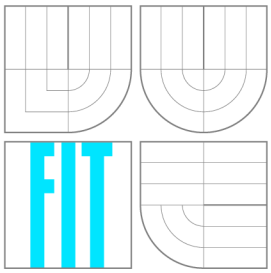
AUTOR PRÁCE
AUTHOR

TOMÁŠ ČÍŽEK

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

MANAŽER PRO DESKOVÉ HRY S PARALELNÍM HRANÍM TURNAJŮ

MANAGER FOR DESK GAMES WITH PARALLEL TOURNAMENT PLAYING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ ČÍŽEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAROSLAV ROZMAN, Ph.D.

BRNO 2015

Abstrakt

Tématem této bakalářské práce je navržení a realizace aplikace pro pořádání turnajů v klasických deskových hrách, které jsou hrány počítačovými programy. Aplikace vychází z již existujícího Manažeru deskových her a rozšiřuje ho mimo jiné o možnost paralelního hraní her v rámci turnaje. Z počátku popisuji původní Manažer deskových her. Následně se věnuji problematice programování paralelních a přenositelných aplikací. Zbytek práce shrnuje návrh a implementaci cílové aplikace.

Abstract

The theme of this work is to design and realize an application for organizing classic board game tournaments played by computer programs. The application is based on already existing Manager of board games and extends it, so there is a possibility of parallel playing games within the tournament, among other things. At the beginning I describe existing Manger of board games. Subsequently, I focus on theory of programing parallel and portable applications. The rest of the text summarizes the design and implementation of the target application.

Klíčová slova

manažer, deskové hry, Java, paralelismus, přenositelnost

Keywords

manager, board games, Java, parallelism, portability

Citace

Tomáš Čížek: Manažer pro deskové hry s paralelním hraním turnajů, bakalářská práce, Brno, FIT VUT v Brně, 2015

Manažer pro deskové hry s paralelním hraním turnajů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Čížek
20.5.2015

Poděkování

Chtěl bych poděkovat vedoucímu práce, Ing. Jaroslavu Rozmanovi, Ph.D., za inspiraci, spolehlivé vedení a bezproblémovou spolupráci.

Obsah

Kapitola 1

Úvod

Je známo, že lidé jsou a vždy byli soutěživí. Hry obecně jsou jako stvořené k uspokojování této lidské touhy. Některé se svou náročností, oblíbeností a popularitou začaly přibližovat i sportům. Pořádání turnajů ve hrách se přirozeně stalo docela častým a nevyhnulo se ani těm deskovým. V této práci budeme uvažovat hlavně klasické deskové hry. Ty mají dlouhou historii, jsou velice oblíbené a lidé v nich soupeří rádi. Příkladem jsou třeba piškvorky, dáma nebo šachy.

S příchodem počítačové éry a vývojem umělé inteligence se však nabídl i jiný druh soutěží. Zde už nemáme na mysli souboj dvou lidí, ale počítačových programů nebo případně také člověka s počítačem. V kontextu této práce se bavíme o počítačových programech vytvořených ke hraní určité deskové hry s cílem ji vyhrát. Takovému programu, který vyhodnocuje stav hry a řídí jednotlivé tahy na herní ploše se říká *brain* (tento název bude dále v tomto významu používán). Z určitého úhlu pohledu se nakonec opět jedná o souboj lidí. Tentokrát to však nejsou přímo hráči, ale programátoři, kteří vytvoří soutěžící programy. Pořádání turnajů tohoto druhu je hlavní motivací této práce.

Cílem je tedy vytvořit aplikaci umožňující pořádání turnajů ve hraní klasických deskových her, kde soutěžícími by byly brainy nebo-li jejich autoři. Přínosem této aplikace je usnadnění a hlavně urychlení realizace turnaje. Samotná hra trvá v případě souboje programů pouze okamžik. Hlavní urychlení pochází z organizace a přípravy jednotlivých duelů. Veškerý průběh turnaje je totiž starostí aplikace.

Je nutno zmínit, že na poli této problematiky již vznikly práce Jiřího Kusáka a Jana Kouřila. Jako důsledek jejich snahy vznikl docela úspěšný Manažer deskových her podporující nejen pořádání turnajů pro brainy, ale také hru člověka proti počítači. Jejich aplikace však není bezchybná a má pár podstatných nedostatků. Chtěl bych jejich práci vylepšit a posunout o krok kupředu. Ve své práci si kladu za cíl vytvořit nový manažer. Ten bude sice velice podobný tomu původnímu, avšak eliminuje některé jeho nedostatky. Hlavními vylepšeními jsou přenositelnost (aplikace je spustitelná na různých operačních systémech) a podpora paralelního hraní her v průběhu turnaje (bude možno hrát více her současně).

Zhruba první polovina tohoto dokumentu je spíše teoretická. Nejprve se podrobněji seznámíme s již existujícím Manažerem deskových her a poté jsou obsaženy kapitoly o problematice přenositelnosti aplikací a také paralelismu. Po následujícím krátkém shrnutí obsahující plán práce začíná druhá polovina. Ta se vesměs věnuje návrhu, popisu funkčnosti a implementace nově vytvořeného manažeru a také zhodnocení výsledku práce.

Kapitola 2

Manažer deskových her

V této kapitole se blíže podíváme na Manažer pro deskové hry (dále jen Manažer), který vznikl v rámci bakalářských prací Jiřího Kusáka a Jana Kouřila [16] [15]. Je to desktopová aplikace napsána v jazyce C# na platformě .NET a funguje pod operačními systémy Windows. Povíme si, které hry jsou podporovány, co Manažer umí, jak vypadá a vysvětlíme si používané typy turnajů.

2.1 Deskové hry

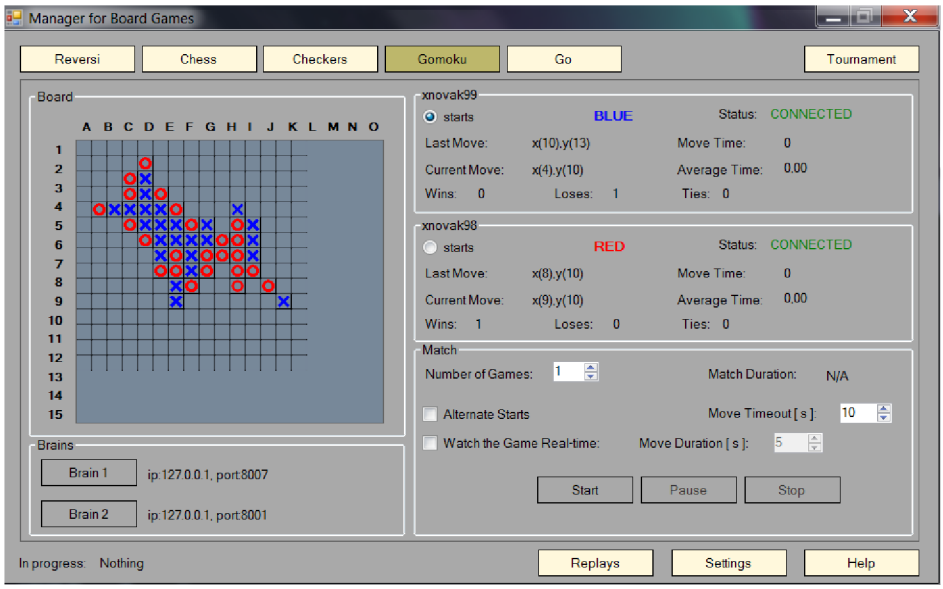
Deskové hry tu s námi byly od nepaměti. Vyznačují se tím, že jejich průběh je realizován na herním plánu. Podle délky jejich historie je můžeme rozdělit do dvou kategorií, moderní a již v úvodu zmíněné klasické deskové hry. Moderní deskové hry (např. Osadníci z Katanu nebo Monopoly) jsou relativně nové, tudíž známe jejich autora a rok vzniku. Historie těch klasických je dlouhá a jsme schopni určit nanejvýš období jejich vzniku.

Většina klasických stolních her jsou pro dva hráče, kteří soupeří mezi sebou. Mají sice poměrně jednoduchá pravidla, ale za to jsou náročné na přemýšlení a strategii. Manažer podporuje pět her, které patří mezi nejznámější: šachy (chess), dámu (checkers), piškvorky (gomoku), othello (reversi) a hru go. Tyto hry jsou rozšířené po celém světě, proto jsou v závorce uvedeny i jejich anglické názvy. Nejnáročnější z těchto her jsou šachy, o nichž se dá říct, že jejich teorie má charakter vědy. Díky tomu jsou však dnes pravděpodobně nejpopulárnější klasickou deskovou hrou. Naopak piškvorky se těší ze své obliby hlavně díky jednoduchosti.

2.2 Funkce a uživatelské rozhraní

Manažer má dva režimy fungování. Uživatel může buď spustit samostatnou partii jen pro dva hráče nebo uspořádat turnaj. Mimo to je možno ukládat a přehrávat záznamy jednotlivých her. Aplikace obsahuje i nápovědu k programu ve formě vestavěného webového prohlížeče. V té lze nalézt pravidla jednotlivých her a informace o nastavení či ovládání programu.

Na obrázku 2.1 vidíme manažer v režimu samostatné partie. V horní části okna si uživatel může zvolit jednu z pěti her. Podle zvolené hry se přizpůsobí herní deska v levé části okna aplikace, na které se zobrazuje průběh hry (na obrázku je zvolena hra piškvorky). Pravá polovina slouží k zobrazení údajů a statistik hráčů, nastavení parametrů partie a jejímu ovládání. A konečně v levém spodním rohu okna aplikace se nachází tlačítka, pomocí nichž lze vybrat účastníky duelu. Lze zvolit jak brain tak i lidského hráče, který pak volí



Obrázek 2.1: Manažer deskových her – samostatná partie (piškvorky)

své tahy na zobrazené herní desce pomocí myši. Při souboji dvou lidí se hráči musí střídat u jednoho počítače.

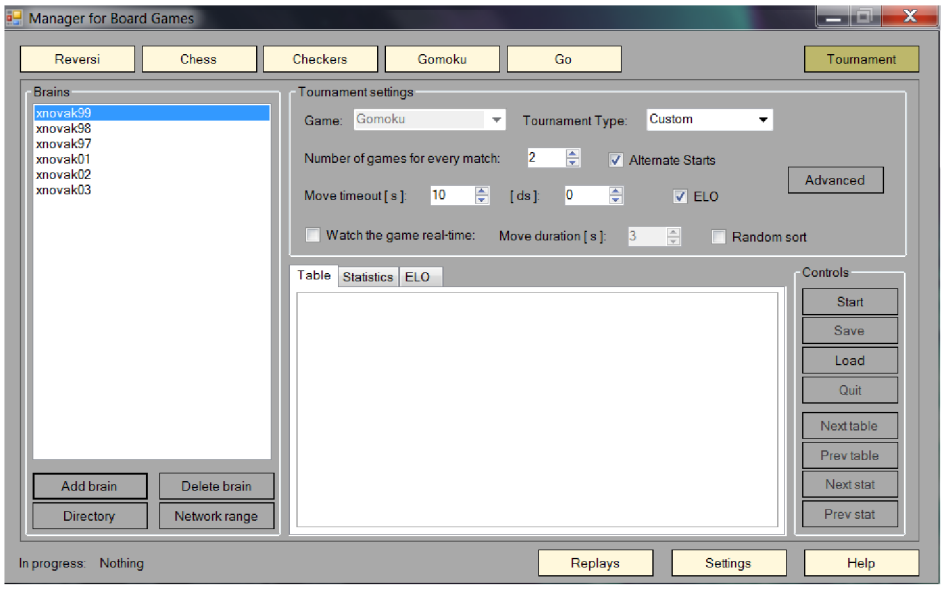
Obrázek 2.2 zobrazuje režim pořádání turnaje. Při levém okraji okna se zobrazuje seznam hráčů. V případě turnaje však už nelze přidávat lidské hráče. Turnaj se dá tedy pořádat pouze pro brainy. Pravá horní část slouží k nastavení turnaje. Obsahuje tedy mimo jiné hlavně možnost vybrat hru a typ turnaje. Protože hry v rámci turnaje se odehrávají postupně, je zde i možnost sledovat celý turnaj v reálném čase s tím, že si uživatel zvolí délku prodlevy mezi jednotlivými tahy. Zbytek okna aplikace je věnován ovládacím tlačítkům a prostoru pro výpis výsledků turnaje nebo statistik jednotlivých hráčů.

2.3 Hodnocení a typy turnajů

V průběhu turnaje hráči získávají body za jednotlivé partie. Tři body získá hráč za vítězství, jeden bod za remízu a za prohru nezískává hráč body žádné. Toto bodové hodnocení hráčů určuje umístění v turnaji.

Mimo to se každému hráči počítá jeho dlouhodobé hodnocení hodnotícím systémem ELO. ELO je hodnota, která určuje výkonnost hráče. Čím lepší hráč je, tím větší by měl mít hodnocení ELO. Tato hodnota je z počátku nastavena na 2000 a přepočítává se po každém turnaji, který hráč odehraje. Za každou prohranou partii se hráči jeho ELO snižuje a za vyhranou partii zvyšuje. Hlavní myšlenkou tohoto hodnotícího systému je, že za výhru nad lepším soupeřem se hráčovo ELO zvýší o více než za výhru nad slabším soupeřem. Stejně tak při prohře nad silnějším soupeřem neztratí tolik jako při prohře se slabším soupeřem. Více podrobností o hodnotícím systému ELO i se vzorcem pro jeho výpočet uvádějí autoři ve svých bakalářských pracích [16] [15].

Manažer podporuje tři typy turnajů: systém Každý s každým, Skupinový systém a Švýcarský systém. První jmenovaný je tím nejspravedlivějším. Jak název napovídá, všichni hráči odehrají jednu partii proti každému ze svých soupeřů. Nevýhodou toho systému je



Obrázek 2.2: Manažer deskových her – turnaj

jen potřeba odehrání velkého počtu partií (to však u turnajů brainů vůbec nevadí).

Při použití Skupinového systému jsou brainy roztříděny do herních skupin o předem definované velikosti. Lze nastavit počáteční počet skupin, jejich velikost i počet postupujících ze skupiny. Skupina se rozhoduje klasickým způsobem – hraje každý s každým. Po odehrání skupin se o konečném pořadí turnaje rozhoduje opět systémem Každý s každým, přičemž postupující hrají už jen mezi sebou a nepostupující taktéž.

Turnaje používající Švýcarský systém se hraje na kola. Principem je v každém kole proti sobě postavit hráče s co nejbližším počtem dosažených bodů. Turnaj končí po odehrání zadaného počtu kol. K určení jasného vítěze je zapotřebí odehrát $log_2 N$ kol kde N je počet účastníků turnaje. Pokud je počet hráčů turnaje lichý, řeší Manažer tuto situaci započítáním přebývajícímu hráči remízu (z takové partie však nelze počítat ELO).

Kapitola 3

Přenositelnost programů

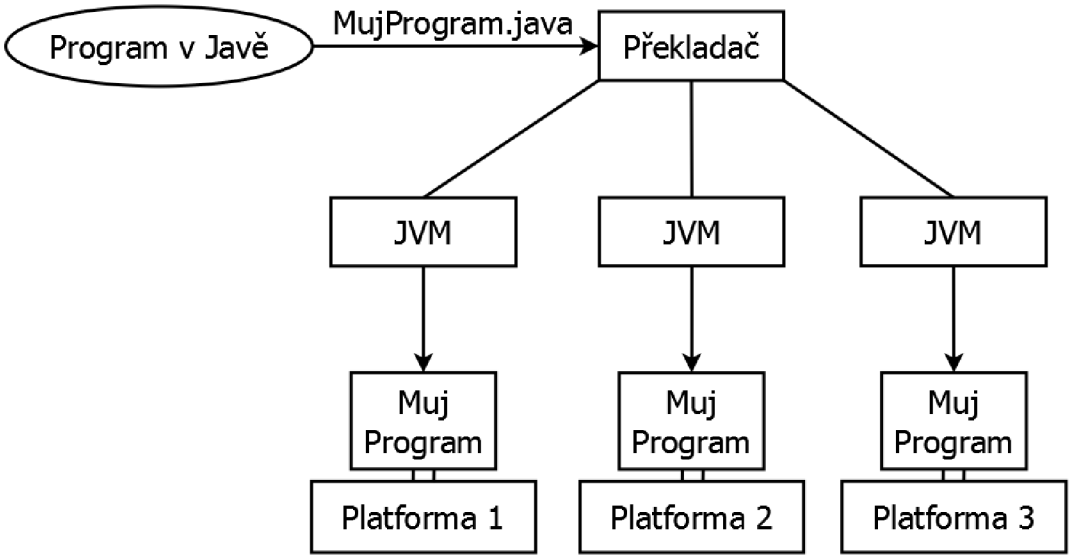
Přenositelnost [17] je v informatice chápána jako schopnost programu být přenesen na jinou platformu. Tato vlastnost se stala jedním z měřítek kvality softwaru. Program můžeme označovat jako přenositelný pokud náklady jeho přenosu na jinou platformu nejsou vyšší než náklady na jeho kompletní přepsání. Jiným označením pro takový software je multiplatformní. Tento pojem však neznamená, že program běží pod všemi platformami, ale že je spustitelný minimálně pod dvěma. Software může být distribuován jako multiplatformní více způsoby:

- je dodáván pro každou platformu zvlášť
- je distribuován formou binárního souboru, který obsahuje kód pro více platforem
- je k dispozici v mezikódu který je interpretován podle příslušné platformy (např. aplikace napsané v Javě)
- je k dispozici ve formě zdrojového kódu

Pojem počítačová platforma představuje kombinaci software a hardware počítače. Hardwarovou částí platformy se rozumí počítačová architektura, což je soustava komponent, které tvoří počítač. Příkladem hardwarových platforem jsou: x86, x86-64, ARM, PowerPC. Softwarová platforma označuje operační systém (např. Microsoft Windows, Linux, Android, MAC OS) nebo také použitý framework (např. Java platform, .NET framework, Adobe Flash, Qt).

3.1 Přenositelnost aplikací napsaných v Javě

Základ nezávislosti Java aplikací na platformě spočívá v tom, že výsledkem kompilátoru jazyka Java není spustitelný strojový kód, ale mezikód zvaný bajtový kód (bytecode). Bajtový kód aplikace je vždy stejný a nezáleží, na které platformě byl vytvořen. Překladač jazyka Java je totiž sám napsaný v Javě a je tedy multiplatformní. Bajtový kód představuje vysoce optimalizovanou sadu instrukcí, které jsou vykonávány běhovým systémem Javy nazývaným virtuální stroj (JVM – Java Virtual Machine). Na rozdíl od překladače, virtuální stroj není přenositelný a tak pro každou platformu musí existovat odlišný (obrázek 3.1) [23]. To je však přijatelná cena za přenositelnost programů. Důležité je, že i když se virtuální stroje na jednotlivých platformách liší, všechny rozumí stejnému bajtovému kódu [20].



Obrázek 3.1: Pro jednotlivé počítačové platformy jsou potřeba různé virtuální stroje (JVM)

Původní virtuální stroj Javy byl v podstatě interpret[1] pro bajtový kód. Interpretovaný program však obecně běží pomaleji, než jak by běžel stejný program při zkompilování do spustitelného kódu. Nic méně v případě Javy je toto zpoždění redukováno vysokou optimalizací bajtového kódu a technologií *HotSpot*. Tato technologie poskytuje kompilátor JIT (Just In Time – ve správný čas), který za běhu programu kompiluje vybrané části bajtového kódu do spustitelného strojového kódu. Důležité je si uvědomit, že význam má kompilovat jen některé části a ne celý program, protože virtuální stroj provádí za běhu nejrůznější kontroly proveditelné pouze za běhu programu. Zbývající kód se jednoduše interpretuje. I přesto však poskytuje kompilace JIT výrazný nárůst výkonu [20].

[1]Interpret je počítačový program, který rozpoznává příkazy ve vstupním programu, a ihned je provádí. Převádí tak vstupní program na posloupnost okamžitě prováděných akcí, aniž by ho nejprve překládal do strojového kódu.

Kapitola 4

Paralelní programování

Paralelismus v programování [1] znamená, že program vykonává více operací současně. Díky tomu může uživatel například zadat programu nějakou déle trvající úlohu a následně v něm dále pracovat, aniž by musel čekat, až se úloha dokončí. Na systémech s jednou centrální výpočetní jednotkou (CPU) toho lze dosáhnout pomocí přepínání kontextu. To znamená, že jednotlivé úlohy se na procesoru střídají. V dnešní době však už máme vícejádrové procesory (procesory s více CPU) a dokonce počítače s několika takovými procesory. Na takových architekturách lze více úloh spustit na jednotlivých CPU současně. Tyto systémy využívají paralelismus i k zrychlení, protože některé výpočty lze rozdělit na více částí a ty vypočítat samostatně na jednotlivých CPU. Dílčí úlohy jsou v paralelních programech často nazývány procesy[1] nebo vlákna[2]. Nicméně *vlákno* je obecně přijímaný termín pro dílčí úkoly.

4.1 Synchronizace paralelních výpočtů

U paralelního programování je velice důležitá synchronizace jednotlivých vláken. Obzvláště pokud používají sdílené prostředky. Špatná synchronizace může kromě zbytečných časových prodlev způsobit narušení konzistence dat nebo situace jako uváznutí[3] či stárnutí[4]. Ke komunikaci či synchronizaci se nejčastěji používá buď zasílání zpráv mezi jednotlivými vlákny (systémy s distribuovanou pamětí) nebo sdílená paměť u architektur které to umožňují. Konkrétním způsobům a metodám, jak časově sladit paralelní výpočty, se říká synchronizační primitiva. Patří mezi ně semafory, fronty zpráv a monitory (více o synchronizačních primitivech a jejich používání lze nalézt v knize [6]).

Paralelní aplikace jsou často klasifikovány podle toho, jak často jejich dílčí úkoly (vlákna) potřebují synchronizovat nebo komunikovat vzájemně mezi sebou. Této vlastnosti se říká granularita [14]. Aplikace se řadí mezi jemnozrnný paralelismus pokud vlákna spolu komunikují mnohokrát za sekundu. Pokud se jedná o hrubozrnný paralelismus nepotřebují spolu komunikovat mnohokrát za sekundu, a pokud se jedná o jednoduchý paralelismus pak spolu komunikují vzácně nebo vůbec.

[1] Proces je v informatice pojem označující spuštěný počítačový program.

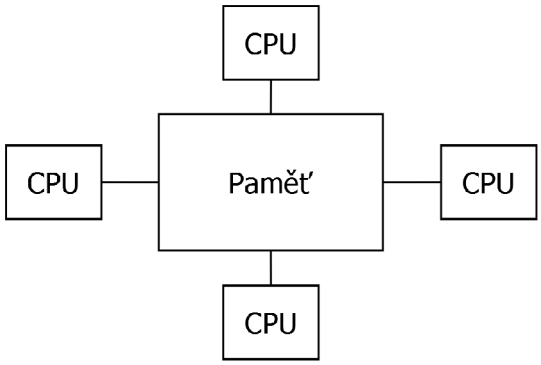
[2] Vlákna jsou základní jednotkou, které operační systém přiděluje čas procesoru, přičemž v rámci jednoho procesu může běžet i více vláken.

[3] Uváznutí nebo-li deadlock je situace, kdy každé vlákno v jisté skupině vláken čeká na akci jiného vlákna v téže skupině. Dochází tak k vzájemnému zablokování.

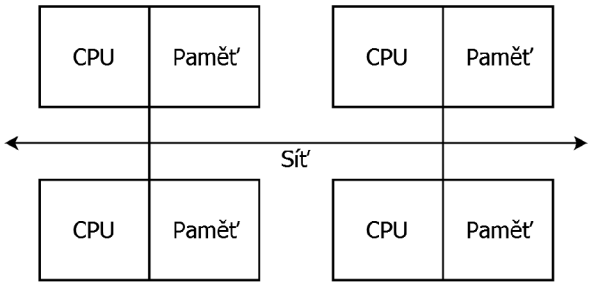
[4] Stárnutí je stav, kdy je vláknu neustále odpíráno použití sdíleného zdroje a bez tohoto zdroje nemůže vlákno dokončit svou úlohu.

4.2 Základní rozdělení paralelních architektur

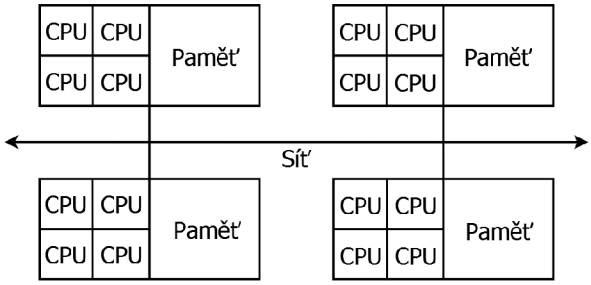
Architektury paralelních systémů můžeme rozdělit do dvou typů na základě druhu paměti (sdílená/distribuovaná). Jednotlivé procesory v systému mohou pracovat samostatně, ale používají sdílenou paměť. Takovým systémům se říká multiprocesory. Druhým typem jsou multipočítače. To jsou systémy, kde každý procesor disponuje vlastní pamětí a komunikují mezi sebou po síti. Dnes je běžná i kombinace obou typů. Všechny varianty jsou znázorněny následujícími obrázky 4.1, 4.2 a 4.3 [3].



Obrázek 4.1: Sdílená paměť (multiprocesor)



Obrázek 4.2: Distribuovaná paměť (multipočítač)



Obrázek 4.3: Kombinace distribuované a sdílené paměti

4.3 Paralelní programovací modely

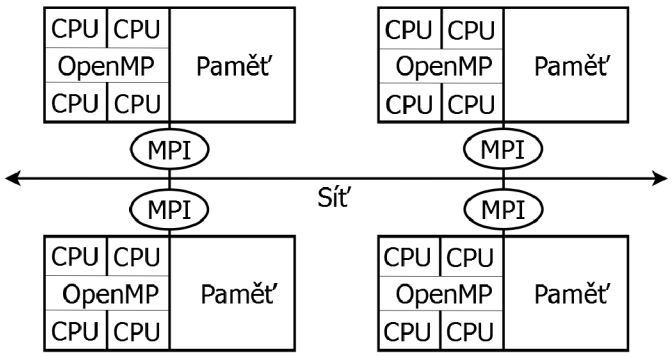
Protože je více způsobů jak propojit jednotlivé procesory a vytvořit tak paralelní systém, existují takzvané modely pro paralelní programování. Ty vyjadřují jakousi abstrakci architektury počítačového systému. Existují dva základní modely, které odpovídají architekturám uvedeným v sekci 4.2. Model reprezentující multiprocesorovou architekturu se nazývá model sdílené paměti, protože pro komunikaci mezi vlákny používá právě sdílené proměnné. Pro multipočítačovou architekturu je typický model zasílání zpráv a jak název opět napovídá, vzájemná interakce zde probíhá vyměňováním zpráv [12].

Je spousta specifikací, rozhraní a standardů pro programování paralelismu a každý z těchto přístupů lze zařadit pod nějaký model. V dnešní době nicméně převládá používání dvou přístupů k implementaci paralelismu. Jsou to realizace zmíněných dvou hlavních modelů a jedná se o OpenMP a MPI. Následující tabulka 4.1 a další dvě podkapitoly shrnují charakteristiky těchto dvou základních implementací [2].

	OpenMP	**MPI**
Programovací model	Model sdílené paměti	Model zasílaná zpráv
Architektura systemu	Architektura se sdílenou pamětí	Architektura se sdílenou nebo distribuovanou pamětí
Komunikační model	Komunikace přes sdílenou paměť	Komunikace prostřednictvím zasílání zpráv

Tabulka 4.1: Charakteristiky OpenMP a MPI

Při programování paralelismu na hybridních architekturách s kombinací sdílené a distribuované paměti se dají i oba přístupy spojit (obrázek 4.4) a využít tak výhod obou programovacích modelů.

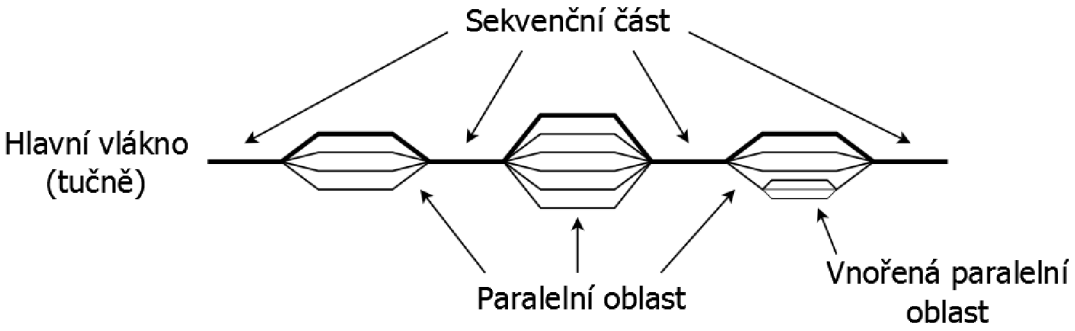


Obrázek 4.4: Hybridní model (spojení MPI a OpenMP)

4.3.1 OpenMP

OpenMP je API[5] usnadňující vytváření vícevláknových programů v programovacích jazycích Fortran, C a C++ [21]. Rozhraní je přenositelné pouze na architekturách se sdílenou pamětí (obrázek 4.1). Jedná se o soustavu direktiv pro překladač a knihovních procedur pro paralelní programování. Tyto direktivy nařizují překladači vytvářet vlákna, jak synchronizovat jednotlivé operace a spravovat sdílenou paměť.

Přepínání mezi sekvenční a paralelní části kódu se řídí *fork/join* modelem. Jedno vlákno se rozdělí do několika nezávislých vláken, které si rozdělí práci a běží současně. Až když každé z nich dokončí svůj výpočet, vlákna se sesynchronizují a spojí opět do jednoho [4]. Model je znázorněn obrázkem 4.5.



Obrázek 4.5: Fork/join model

Detaily k programování pomocí OpenMP jsou k dispozici ve specifikaci dostupné na oficiálních stránkách [19].

4.3.2 MPI (Message Passing Interface)

V případě MPI se nejedná přímo o API založené na zasílání zpráv mezi jednotlivými uzly, ale spíše o specifikaci pro jejich vytváření. Existuje tedy více implementací MPI pro různé programovací jazyky (Fortran, c, C++, Java atd.). Původní MPI (MPI-1) bylo navrženo pouze pro architektury s distribuovanou pamětí (obrázek 4.2), které byly populární v 80. a začátkem 90. let. Avšak s vývojem multiprocesorových systémů se začaly používat spíše hybridní systémy s kombinací sdílené a distribuované paměti (obrázek 4.3). Další verze (MPI-2 a MPI-3) tedy podporují implementaci i na hybridních systémech a systémech se sdílenou pamětí [5].

MPI programy pracují vždy s procesy. Pro dosažení nejlepšího výkonu je zapotřebí přidělit každému procesu jeden procesor, odpadá tím tak zpoždění způsobené přepínáním kontextu. K tomuto mapování však nedochází v době překladu aplikace, ale v době běhu aplikace, a to prostřednictvím agenta, který MPI program spustil [22].

Důležitým pojmem v MPI jsou takzvané komunikátory. Jedná se o skupiny procesů při běhu MPI aplikace, v níž jsou procesy číslovány od nuly. V rámci takové skupiny mohou procesy komunikovat, přičemž se navzájem adresují právě přiřazeným pořadovým číslem [22].

[5] API (zkratka pro Application Programming Interface) označuje v informatice rozhraní pro programování aplikací.

Každý proces má oddělený paměťový prostor. Ke komunikaci dochází při kopírování části adresového prostoru jednoho procesu do adresového prostoru jiného. V MPI existuje více druhů komunikace: point-to-point komunikace, kolektivní komunikace, jednostranná komunikace, paralelní vstupně-výstupní operace [12]. První zmíněný druh, point-to-point, je komunikace pouze mezi dvěma procesy. Zatím co kolektivní operace komunikují s více procesy. Například se všemi ve stejném komunikátoru. Tyto dva druhy komunikace ovšem potřebují synchronizaci komunikujících procesů (jeden proces musí posílat a druhý přijímat). Jednostranná komunikace (od verze MPI-2) takové spojení odesílatel-příjemce nepotřebuje. Jedná se o vzdálený přístup do paměti (anglicky remote memory access). Poslední druh komunikace, paralelní vstupně-výstupní operace, poskytuje přístup k vnějším zařízením s využitím datových typů a komunikátoru.

Více informací o programování pomocí MPI je k dispozici v oficiálních dokumentacích [18].

4.4 Paralelismus v Javě

Dosud jsme se zabývali modely pro paralelní programování. V případě programovacího jazyka Java je však paralelní programování podporováno přímo jazykem samotným. Paralelismus se v Javě implementuje vytvářením nových vláken. Většina aplikací v Javě je spuštěna jako jeden proces a každý proces je tvořen nejméně jedním vláknem (nepočítáme-li vlákna starající se o zprávu paměti a obsluhu signálů). Toto vlákno je hlavní a má schopnost vytvářet další vlákna. Jednotlivé vlákna jsou reprezentovány instancemi třídy *Thread*.

4.4.1 Distribuovaná sdílená paměť

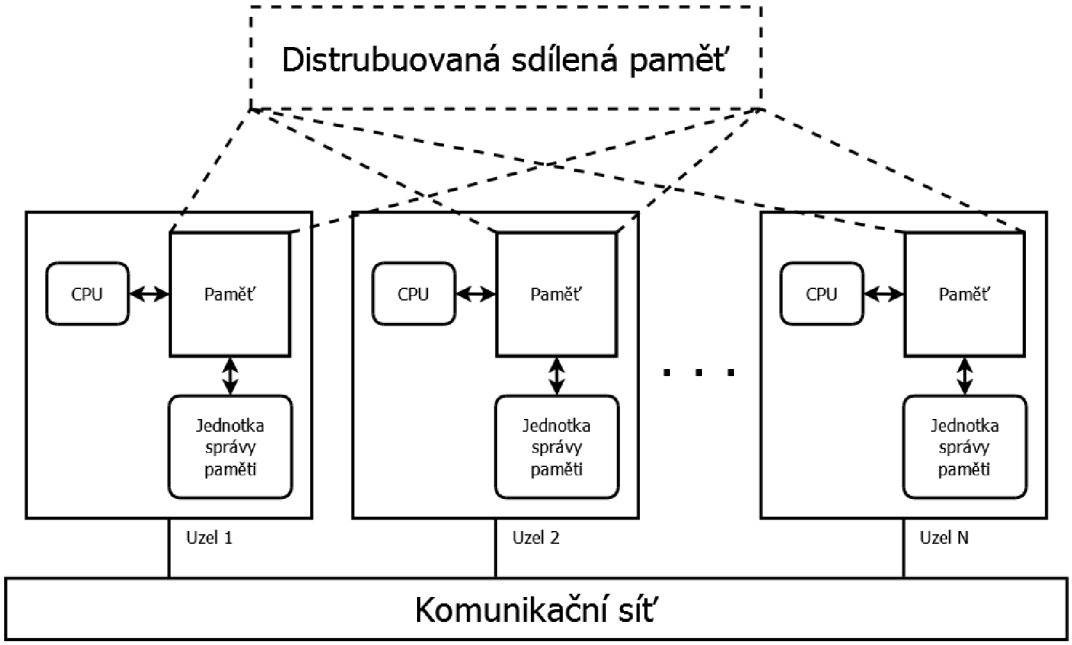
Jak vyplývá z předchozího povídání o paralelismu, moderní paralelní architektury obecně spadají buď pod systémy se sdílenou pamětí nebo systémy s distribuovanou pamětí. Systémy se sdílenou pamětí jsou sice dražší a obtížněji sestavitelnější, ale za to relativně snadno programovatelné. U distribuovaných systémů je to obráceně (jsou levnější a obtížněji programovatelné). Jazyk Java řeší toto dilema vytvořením abstrakce sdílené paměti nad systémem s pamětí distribuovanou (obrázek 4.6) [7]. Této abstrakci se říká Distribuovaná Sdílená Paměť (anglicky Distributed Shared Memory (DSM)) a umožňuje použit relativně snadnější programovací model pro sdílené paměti i na distribuovaných systémech. DSM abstrakce je dosaženo jasným mapováním odkazů do paměti na hardware používající zasílání zpráv [7].

4.4.2 Vytváření vláken v Javě

Instanci třídy *Thread* lze vytvořit dvěma způsoby [9]. Prvním z nich je vytvořit třídu implementující rozhraní *Runnable* a instanci této třídy předat jako parametr objektu třídy *Thread*. Rozhraní *Runnable* totiž definuje klíčovou metodu run, která realizuje samotnou činnost vlákna a je třeba ji implementovat. Druhým způsobem je rozšířit samotnou třídu Thread, protože ta samotná implementuje rozhraní *Runnable*. Její metoda *run* však nedělá nic, tudíž ji musí programátor redefinovat.

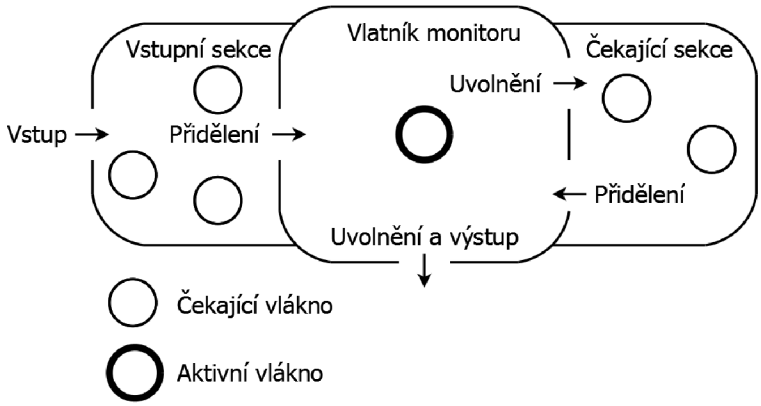
4.4.3 Synchronizace vláken v Javě

Každé vlákno má vlastní zásobník a programový čítač. Navzájem však vlákna sdílí informace o všech vytvořených objektech, načtených třídách, rozhraních a také statické proměnné. Při přístupu ke sdíleným datům je tedy nutno myslet na bezpečnost a synchronizaci



Obrázek 4.6: Distribuovaná Sdílená Paměť

aby nedošlo k chybě konzistence[6]. Hlavním synchronizačním primitivem v Javě jsou monitory. Každý objekt má automaticky přiřazen svůj monitor (obrázek 4.7) [7].



Obrázek 4.7: Monitor v Javě

Mechanismus synchronizace vláken v Javě lze rozdělit do dvou částí: výlučný přístup a součinnost vláken. Sekce s výlučným přístupem jsou úseky kódu, které nemohou být prováděny současně více vlákny. Lze je vytvářet pomocí klíčového slova *synchronized*. Jako úsek s výlučným přístupem lze označit metody, celé třídy nebo jen vybrané části kódu. O vláknu provádějící sekci s výlučným přístupem se říká, že vlastní monitor. Ostatní vlákna, které chtějí získat monitor, musí čekat ve vstupní sekci (obrázek 4.7). Až když vlákno sekci s výlučným přístupem opustí (uvolní monitor), může do ní vstoupit další vlákno.

[6]Chybou konzistence je myšlena situace, kdy dvě různá vlákna mají jiný pohled na to, co mají být stejná data [9].

Součinností vláken se rozumí používání metod *wait*, *notify* a *notifyAll*. Každé vlákno vlastnící monitor může sebe samo zastavit zavoláním metody *wait*. Tím přejde do čekající sekce (obrázek 4.7) a uvolní monitor. Takto čekající vlákno může být znovu aktivováno jiným vláknem vlastnící monitor pomocí metody *notify*. Tímto způsobem lze uvědomit čekající vlákna například o změně sdílených dat. Metoda *notify* však uvědomí pouze jedno vlákno. Chceme-li dát vědět všem vláknům z čekající sekce, že se mají aktivovat, je třeba použít metodu *notifyAll*.

4.4.4 Exekutory

Prozatím jsme si popisovali způsob paralelního programování v Javě, kdy pro každou úlohu, jenž má být provedena samostatným vláknem, musí sám programátor vytvořit instanci třídy Thread. Java však poskytuje i možnost vytváření a správu vláken od zbytku aplikace oddělit, což je vhodné pro vývoj větších a složitějších paralelních aplikací. Tato myšlenka je realizována rozhraním *Executor*. Příkladem implementace tohoto rozhraní je *fork/join* framework. Jeho princip spočívá v rekurzivním rozdělování práce na menší díly s cílem využít veškerou výpočetní sílu systému. Tato myšlenka je zřejmá i z následujícího pseudokódu (algoritmus 1), který představuje jádro *fork/join* frameworku.

Algoritmus 1: Základní pseudokód fork/join frameworku

```
if díl mé práce je dostatečně malý then
    vykonám práci přímo
else
    rozdělím práci do dvou částí, přidělím je jiným vláknům a počkám na výsledky
end
```

Rozhraní *Executor* s frameworkem *fork/join* patří k prvkům paralelismu vyšší úrovně poskytovaných balíkem *java.util.concurent* [8]. Více podrobností o rozhraní *Executor* a frameworku *fork/join* lze nalézt i s příkladem na stránkách dokumentací k jazyku Java [9].

Kapitola 5

Návrh aplikace

Tato kapitola se věnuje návrhu a detailnější specifikaci cílové aplikace. Jak už bylo řečeno v úvodu, cílem práce je vytvořit novou aplikaci pro pořádání turnajů v deskových hrách pro umělé inteligence. Tato aplikace ponese název Manažer pro deskové hry 2 (dále jen Manažer 2). Ten bude multiplatformní a jednotlivé hry v rámci turnaje budou probíhat paralelně. Tyto dva hlavní požadavky mne přivedly k volbě programovacího jazyka, kterým je Java. Důvody, proč je jazyk Java vhodný pro tvorbu přenositelných a paralelních aplikací, jsou obsaženy v sekcích 3.1 a 4.4. V následujících podkapitolách se dozvíme požadavky na funkčnost Manažeru 2, jak bude vypadat GUI, čím se bude Manažer 2 podobat původnímu Manažeru a v čem se naopak budou lišit.

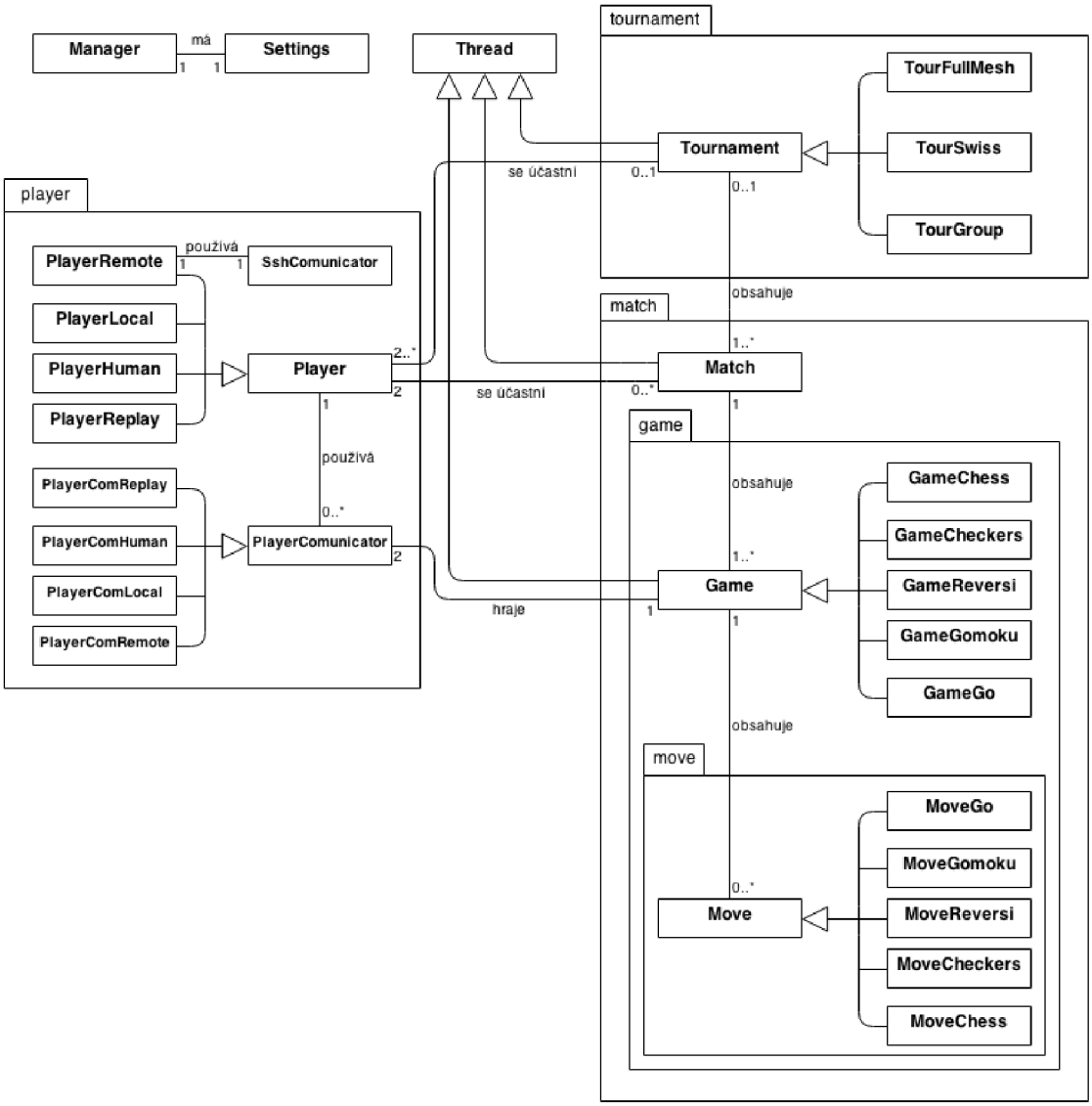
5.1 Funkčnost aplikace

Funkčnost a většina možností nastavení bude zachována z původního Manažeru (popisu původního Manažeru deskových her se věnuje kapitola 2. To znamená, že Manažer 2 by měl splňovat následující požadavky:

1. Podporovat pět deskových her (šachy, dáma, piškvorky, othello a go).
2. Obsahovat režim pořádání samostatných partií, kterých se budou moci účastnit lidští hráči, lokální brainy i brainy na vzdálených počítačích.
3. Obsahovat režim pořádání tří druhů turnajů (systém všichni proti všem, švýcarský systém a skupinový systém) pro lokální i vzdálené brainy.
4. Po dokončení turnaje umožnit prohlížení tabulek a výsledků, uložení výsledku turnaje a uložení záznamů všech her turnaje.
5. Obsahovat režim přehrávání uložených záznamů jednotlivých her.
6. Obsahovat nápovědu.

Kvůli zavedení paralelismu však ubude možnost sledovat turnaj v reálném čase, která byla v původním Manažeru k dispozici. Tato funkce stejně není příliš praktická vzhledem k tomu, že ze všech her je možno pořizovat záznamy a následně je přehrávat. Na druhou stranu bude Manažer 2 podporovat více jazyků (češtinu a angličtinu), které se budou dát přepínat v nastavení aplikace. Také přibude možnost volby velikosti herní desky u her, které to svými pravidly umožňují (piškvorky a go).

Na obrázku 5.1 si lze prohlédnout navržený diagram tříd, který z úsporných důvodů neobsahuje atributy ani metody tříd. Můžete z něj vysledovat, že všechny partie i hry budou spouštěny v samostatných vláknech, aby mohly probíhat současně a nezávisle. Také třída představující turnaj je zděděná z třídy *Thread*. To však není z důvodu, že by turnajů mohlo probíhat více, ale aby mohl uživatel v průběhu turnaje pracovat s grafickým uživatelským rozhraním.

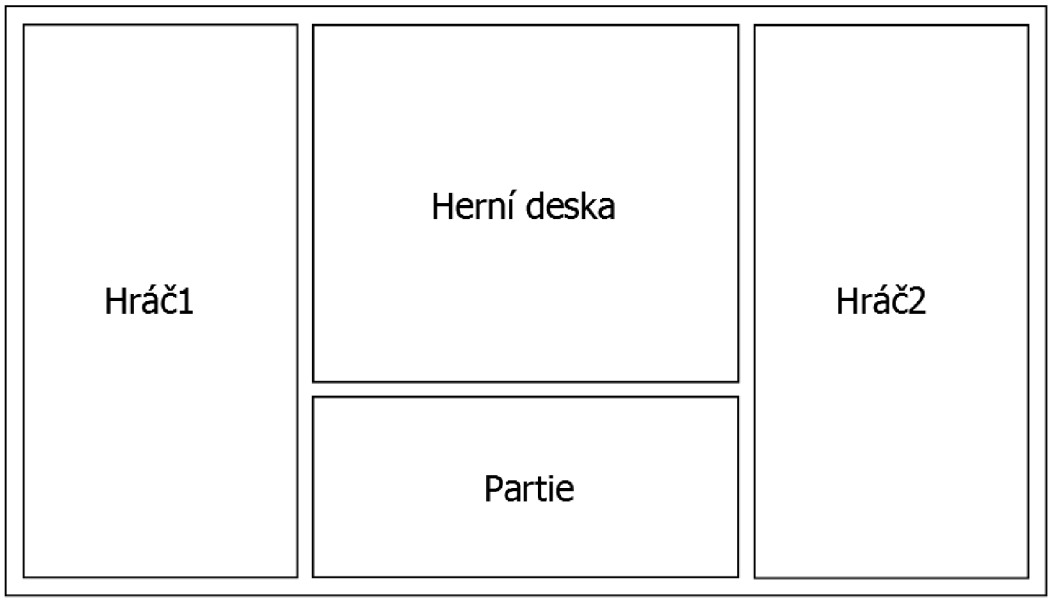


Obrázek 5.1: Diagram tříd

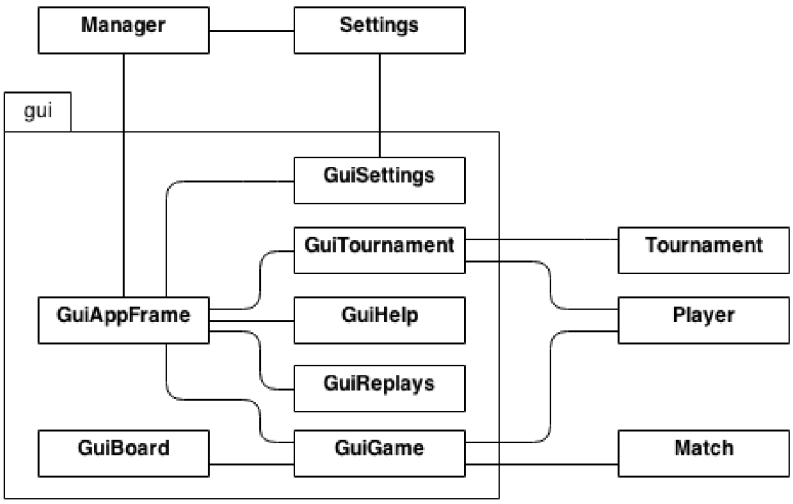
5.2 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní (GUI) Manažeru 2 bude také inspirováno původním Manažerem a lišit se bude jen mírně. Celé GUI se bude skládat pouze z jednoho hlavního okna a pár modálních dialogů. Vrchní a spodní část hlavního okna bude obsahovat podlouhlé lišty

s tlačítky, které budou sloužit k přepínání mezi jednotlivými režimy aplikace a k zobrazení nastavení nebo nápovědy. Ve zbytku okna, tedy celé středové části, se bude zobrazovat samotný obsah. Takové rozložení je ostatně použito v původním Manažeru a je vidět na obrázcích 2.1 a 2.2. Hlavní myšlenka GUI tedy zůstane. Odlišné bude pouze rozložení jednotlivých prvků obsahu hlavně v režimu pořádání samostatné partie (toto nové rozložení zobrazuje obrázek 5.2). Na obrázku 5.3 si lze prohlédnout navržený diagram tříd pro GUI Manažeru 2.



Obrázek 5.2: Návrh GUI pro režim samostatné partie

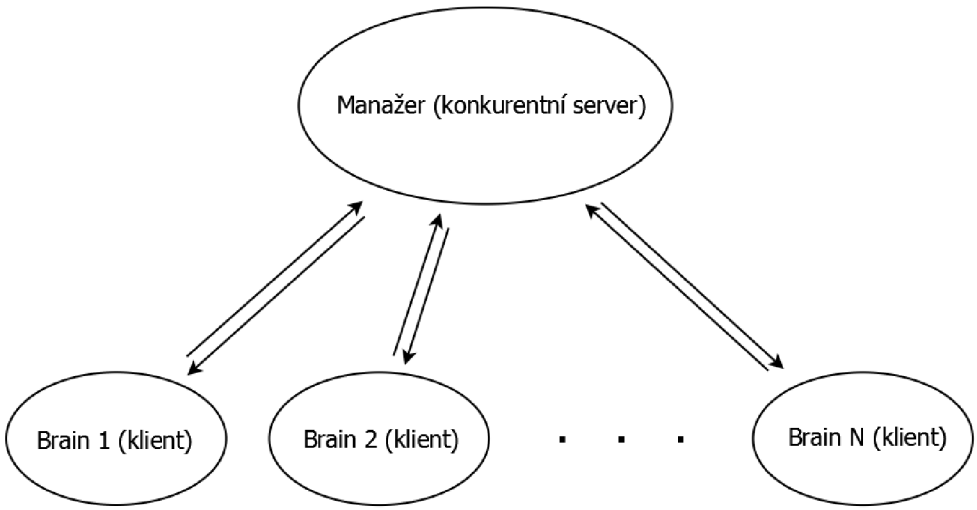


Obrázek 5.3: Diagram tříd GUI

5.3 Komunikace mezi Manažerem a brainy

Pro komunikaci mezi Manažerem 2 a brainy bude použita, stejně jako v předchozí verzi Manažeru, technologie soketů a architektura TCP/IP. Avšak z pohledu rolí v síťové komunikaci, bude nutná určitá změna.

Původní Manažer se chová jako klient, který spouští brainy fungující jako iterativní servery a následně se k nim připojuje. Problém tohoto řešení tkví v tom, že při větší zátěži procesoru spuštěný program brainu nemusí stihnout vytvořit svůj serverový soket a tím pádem pokus o připojení ze strany manažeru selže. V Manažeru tento problém při běžném využití procesoru nenastává, protože samotná aplikace Manažeru natolik procesor nezatěžuje. Navíc při připojení k vzdálenému brainu si Manažer situaci ulehčuje tím, že počítá s již předem spuštěným brainem. V Manažeru 2 by však k výskytu tohoto problému zcela jistě docházelo. Manažer 2 je totiž, jak už víme, navržen tak aby se všechny hry turnaje odehrávali paralelně. To s sebou přinese nejen požadované rapidní zrychlení průběhu turnajů, ale také mnohem větší vytížení procesoru. Tento problém se dá řešit například opakováním pokusu o připojení k brainu s nějakou časovou prodlevou. Zde se však nabízí otázka, kolikrát se o připojení pokusit a jak dlouhou časovou prodlevu mezi pokusy použít. Navíc by toto řešení s použitím časových prodlev manažer zbytečně zpomalovalo. Mnohem elegantnější a také efektivnější řešení spočívá v prohození rolí brainu a manažeru. Manažer 2 tedy bude fungovat jako konkurentní server, který bude spouštět brainy představující klienty. Poté už jen počká na navázání spojení inicializované brainem. Schéma síťové komunikace je znázorněno obrázkem 5.4.



Obrázek 5.4: Schéma komunikace manažeru s brainy

Protokol, jenž bude použitý pro komunikaci mezi Manažerem 2 a brainy, je převzat z původního Manažeru. Jednotlivé zprávy protokolu a reakce na ně jsou popsané tabulkou 5.1.

Zpráva manažeru	**Kladná a záporná odpověď brainu**	**Popis zprávy**
hello msg	hello ok hello error	Dotaz, zda je brain připraven komunikovat. Pokud brain odpoví kladně, znamená to, že je připraven příjmat další zprávy.
com msg	com ok com error	Dotaz na komunikační protokol. Zatím je implementován jen jeden protokol. Tato zpráva je určená pro možné rozšíření (msg by mohlo být nahrazeno třeba 1.0, 1.1 atd.).
game [hra]	game ok game failed	Informace o hrané hře. Pokud ji brain umí hrát, odpoví kladně.
board [číslo]	board ok board error	Manažer pomocí této zprávy informuje brain o velikosti herní plochy.
name msg	name [jméno] name error	Dotaz na jméno brainu. Jméno slouží pro identifikaci brainu v turnaji.
timeout [číslo]	timeout ok timeout error	Manažer tímto informuje brain o maximální době, jakou je ochoten čekat na odpověď. Proměnná číslo udává čas v milisekundách.
player [číslo]	player ok player error	Informace pro brain, za jakou barvu bude hrát. U her go a piškvorky tato informace není podstatná, ale ve hrách othello, dáma a šachy je tuto informaci potřeba znát. othello: 1 – bílý, 2 – černý šachy: 1 – černý, 2 – bílý dáma: 1 – černý, 2 – bílý
new game	new game ok new game error	Oznámení nové hry. Brain v tuto chvíli nainicializuje herní plochu a pokud používá, tak i své další stavové proměnné. Po této zprávě musí být připraven na novou hru.
move [tah]	move [tah] move error	Zpráva oznamující brainu tah protivníka. Pokud proměnná tah obsahuje řetězec *start*, znamená to, že je brain vyzván k provedení prvního tahu. Formáty tahů pro jednotlivé hry jsou v tabulce 5.2.
quit msg	– –	Zpráva, kterou posílá manažer brainu, když s ním už dále nebude komunikovat. Brain se po obdržení této zprávy může ukončit.

Tabulka 5.1: Zprávy komunikačního protokolu

Hra	**Formát tahu**	**Popis formátu tahu**
go, piškvorky, othello	x(X),y(Y)	X a Y jsou souřadnice na herní desce a mohou nabývat hodnot od nuly po velikost herní plochy mínus jedna. Ve hrách othello a go může brain odpovědět i řetezcem *forfeit*, což znamená, že se vzdává tahu. Příklad – x(2),y(6)
dáma	XY[KL]+	Dvojice XY jsou souřadnice počátku tahu. Další souřadnice KL označují cílové pole, kam chce brain táhnout. Tato položka se může opakovat vícekrát, neboť v dámě lze provádět vícenásobné skoky. X a K nabývají hodnot A-H a Y a L hodnot 1-8. Příklad – A1C3A5
šachy	XYKL	Využívá se stejný formát jako u dámy s tím rozdílem, že cílové pole se nemůže opakovat. Příklad – A2A3

Tabulka 5.2: Formát tahů zprávy move pro jednotlivé hry

Kapitola 6

Realizace aplikace

Tato kapitola je věnována detailnímu seznámení s novou aplikací – Manažer pro deskové hry 2. Představíme si jednotlivé části programu, jejich funkci, vzhled a zajímavosti z hlediska implementace. Hned v úvodu bych chtěl zmínit, že ačkoli Manažer 2 podporuje všech pět deskových her, plně funkční jsou zatím jen piškvorky. U ostatních her chybí doprogramovat části jako je detekce konce hry nebo kontrola platnosti tahu. Během následného popisu aplikace však budeme uvažovat, že i ostatní hry jsou uživateli přístupné.

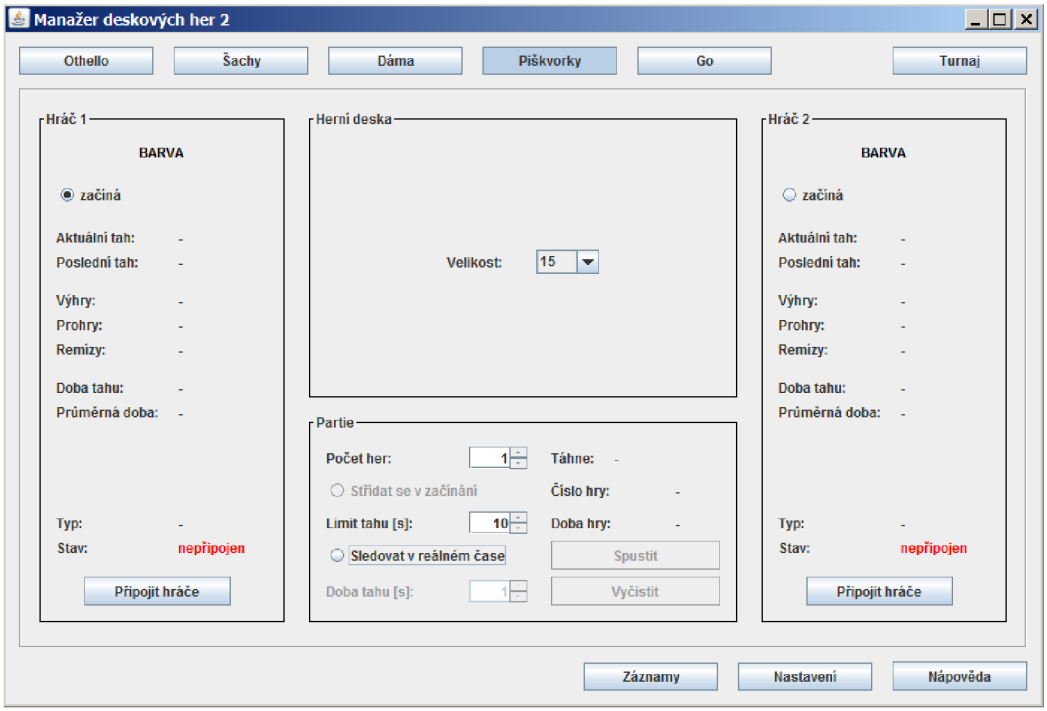
Jak jsem již zmínil v předchozí kapitole 5, celá aplikace je napsaná v programovacím jazyce Java. Grafické uživatelské rozhraní je vytvořeno s použitím Swing API. Volba vývojového prostředí tedy padla na NetBeans IDE, které podporuje různé technologie spojené s jazykem Java a obsahuje také nástroj pro tvorbu grafického uživatelského rozhraní. Brainy, na kterých byl Manažer 2 testován, jsou napsány v jazyce C/C++.

6.1 Samostatné partie

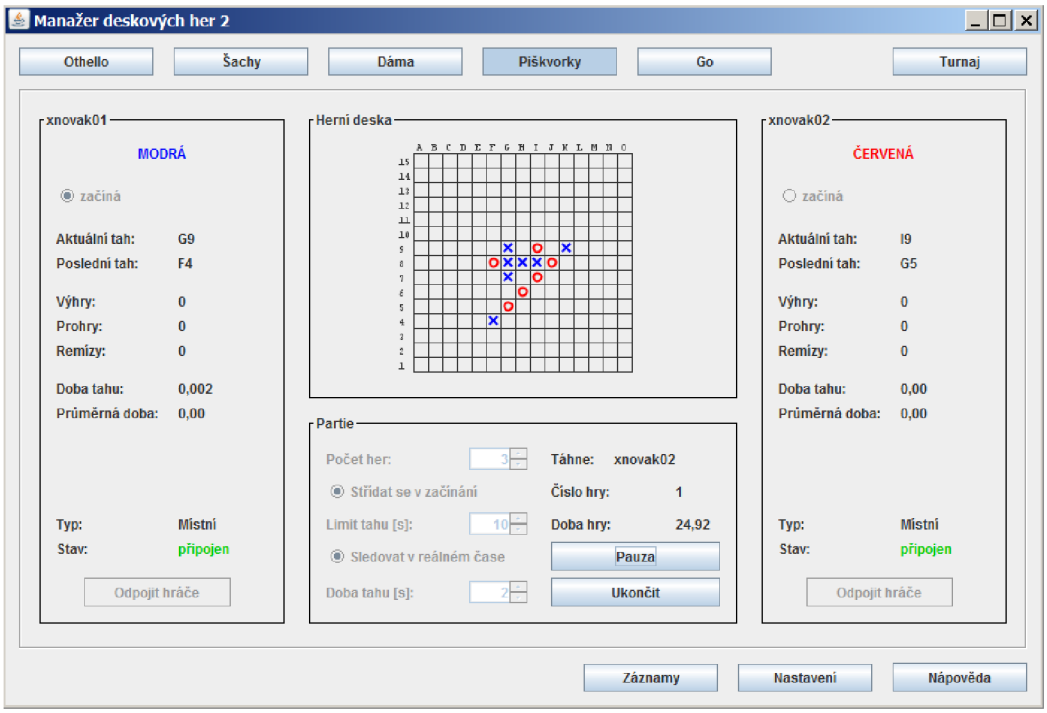
První ze tří módů aplikace slouží pro pořádání samostatných partií. Tento režim se aktivuje volbou jedné z her v horní části okna aplikace. Po výběru hry se zobrazí GUI tohoto režimu, které si můžete prohlédnout na obrázcích 6.1 a 6.2. Pojďme si tedy tento režim Manažeru 2 představit.

GUI je rozděleno do čtyř panelů. Dva po stranách jsou věnovány hráčům. Ty slouží pro zobrazení veškerých informací o hráčích, pro volbu začínajícího hráče a v obou je i tlačítko pro připojení/odpojení hráče. Mezi zobrazované informace o hráči patří: souřadnice aktuálního a předchozího tahu, počet výher, proher a remíz, časomíra zobrazující délku aktuálního tahu, průměrná doba tahu v rámci hry a typ hráče. Připojení hráče je realizované pomocí dialogu (obrázek 6.3), z něhož může uživatel připojit buď lidského hráče nebo brain. Pro výběr souboru brainu je použita speciální swing komponenta *VFSJFileChooser*, pomocí které lze procházet jak lokální, tak vzdálené souborové systémy (FTP, SFTP, WEBDAV atd.) [24]. Vzhled této komponenty lze vidět na obrázku 6.4. Manažer 2 tedy rozlišuje tři typy hráčů: lokální brain, vzdálený brain a člověka. V samostatné partii lze proti sobě postavit jakoukoli kombinaci těchto tří typů hráčů.

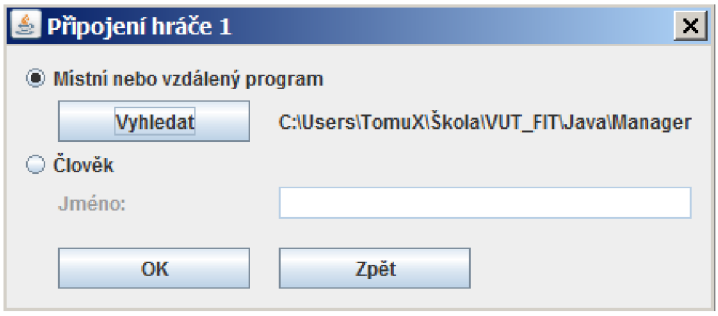
Uprostřed dole se nachází panel partie. Jeho levá půlka obsahuje prvky pro nastavení partie: počet her partie, střídání se v začínání, časový limit na tah a sledování hry v reálném čase. Sledování hry v reálném čase znamená nastavení časové prodlevy mezi jednotlivými tahy a je umožněno, pouze pokud se partie neúčastní lidský hráč. Na pravé části panelu jsou informace o probíhající hře (časomíra, číslo hry v rámci partie, jméno hráče na řadě)



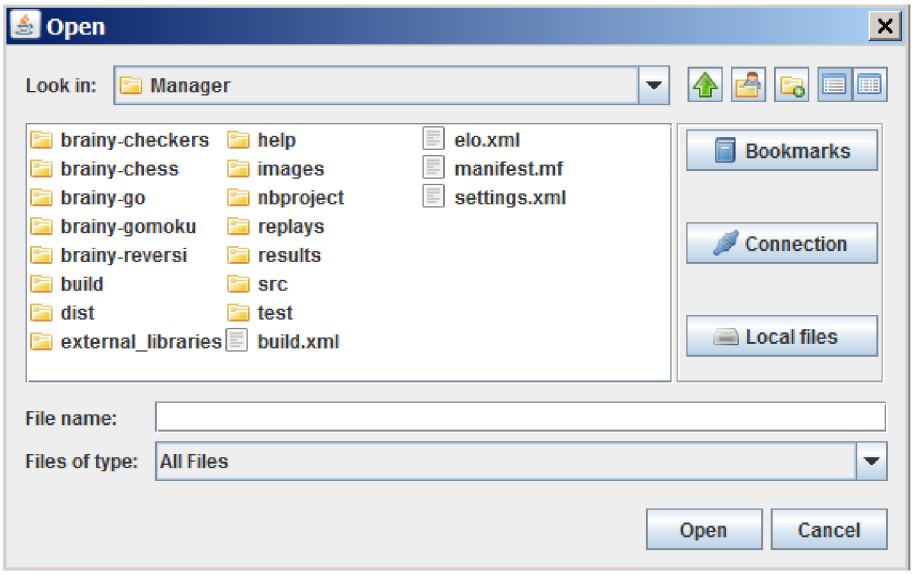
Obrázek 6.1: Manažer deskových her 2 – samostatná partie (před hrou)



Obrázek 6.2: Manažer deskových her 2 – samostatná partie (během hry)



Obrázek 6.3: Manažer deskových her 2 – dialog pro připojení hráče



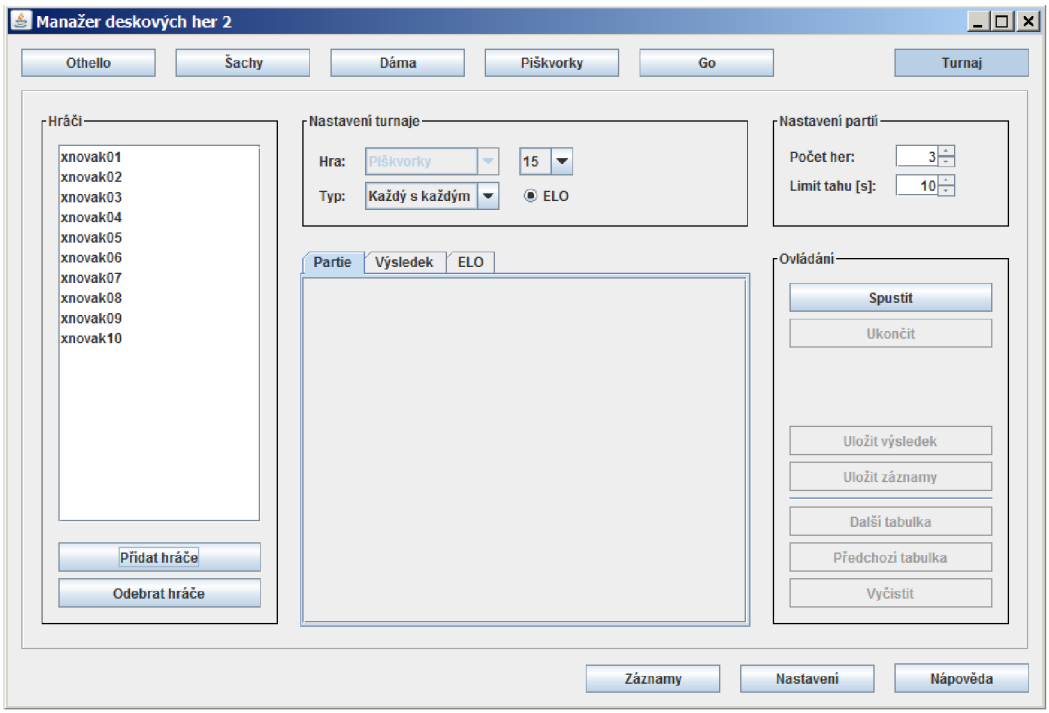
Obrázek 6.4: Swing komponenta VFSJFileChooser

a také dvě tlačítka pro ovládání. Obě mají dvě různé funkce podle toho, v jakém stavu se partie nachází. Vrchní tlačítko slouží pro odstartování partie nebo pro její pozastavení. Aby šla partie odstartovat, je třeba mít připojené oba hráče. Spodním tlačítkem lze partii ukončit a následně druhým kliknutím provést akci vyčistit. Tato akce odstraní z GUI veškeré informace spojené s ukončenou partií včetně herní desky. Hráči zůstávají připojeni.

Čtvrtým panelem je ten, v němž se zobrazuje herní deska. Nachází se ve zbylém prostoru mezi panely hráčů. Velikost tohoto panelu se zvětšuje či zmenšuje se změnou velikosti okna. Rozměry panelů hráčů se nemění a panel partie mění pouze svou šířku. Před spuštěním partie lze v panelu herní desky zvolit její velikost. U her šachy, dáma a othello je s ohledem na pravidla k dispozici pouze volba herní desky o velikosti 8x8. Po odstartování partie se již zobrazuje herní deska s aktuálním průběhem hry. Lidští hráči své tahy zadávají pomocí levého tlačítka myši přímo na zobrazené herní desce. Tu lze také přibližovat a oddalovat a to pomocí kolečka myši.

6.2 Turnaje

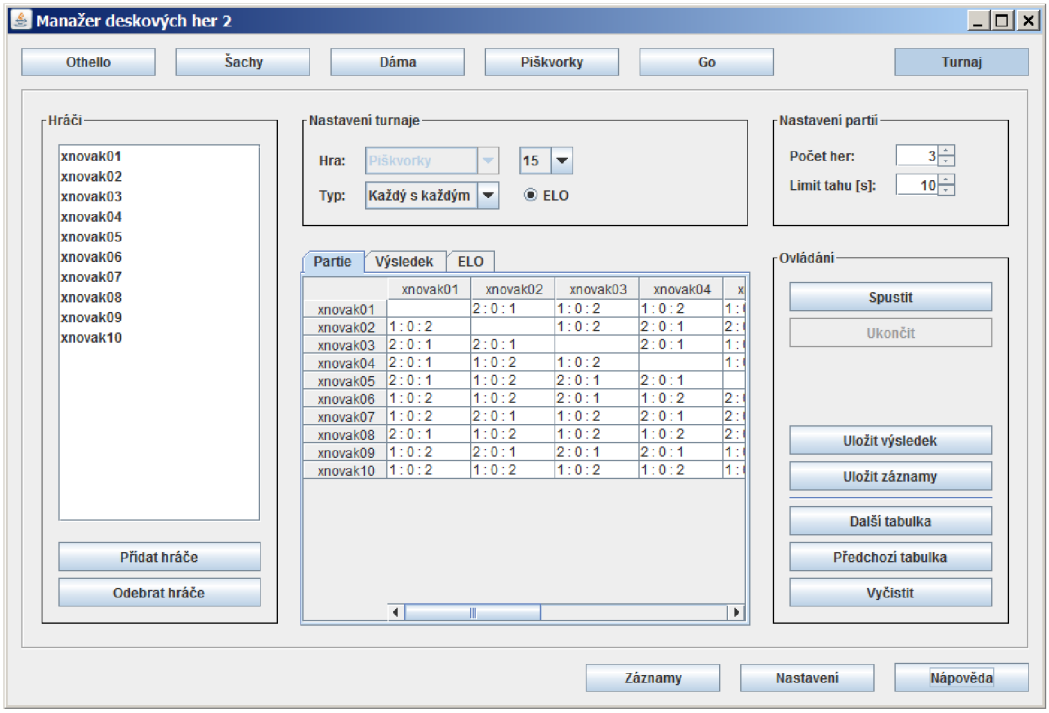
Režim pořádání turnajů je tím hlavním a nejdůležitějším. Dostaneme se do něj tlačítkem v pravém horním rohu okna. Na obrázcích 6.5 a 6.6 můžeme vidět, jak tento režim vypadá.



Obrázek 6.5: Manažer deskových her 2 – turnaj (před turnajem)

GUI tohoto režimu se opět skládá z několika panelů. Vlevo se nachází seznam hráčů účastnících se turnaje a dvě tlačítka pro editaci tohoto seznamu. V případě turnaje jsou hráči myšleny brainy, protože lidé se turnajů účastnit nemohou. Výběr brainů je realizován opět komponentou *VFSJFileChooser*, takže lze opět připojit brainy lokální i vzdálené. Do turnaje nelze připojit dva brainy se stejným jménem. Jméno brainu totiž slouží k jeho identifikaci.

Uprostřed nahoře je umístěn panel s nastavením turnaje. Typ hry lze měnit, pouze je-li seznam připojených hráčů prázdný. U piškvorek a go lze opět zvolit různé velikosti herní desky. Dále je zde možnost zvolit jeden ze tří typů turnaje. Tyto typy turnaje fungují, až na maličkosti, stejně jako v původním Manažeru (popis těchto turnajů naleznete v sekci 2.3). První maličkost se týká švýcarského systému. A to sice, že počet kol turnaje se nedá zvolit, ale je pevně nastaven na hodnotu $log_2 N$, kde N je počet účastníků turnaje. V důsledku toho je zaručeno získání jednoznačného vítěze turnaje pouze při počtu hráčů, který je mocninou dvojky. Při jiném počtu hráčů se může stát, že se na první příčce umístí hráčů více. Další rozdíly se týkají skupinového turnaje. Hráči, kteří ze skupin nepostoupí, už v turnaji končí a nehrají dále mezi sebou o konečné umístění. Podstatou odehrávání skupin je totiž zredukování počtu hráčů. V původním Manažeru se tedy skupinový systém příliš neliší od turnaje každý s každým. Skupinový systém si navíc vyžaduje při startu turnaje zadat počet skupin a počet postupujících z každé skupiny. Na tyto čísla je uživatel tázán v dialozích, které se objeví před spuštěním turnaje. Poslední akcí, kterou uživatel v panelu



Obrázek 6.6: Manažer deskových her 2 – turnaj (po turnaji)

s nastavením turnaje může provést, je aktivace a deaktivace hodnocení ELO. Pokud je aktivováno, přepočítává se hráčům toto hodnocení po odehrání každé partie v turnaji. ELO je dlouhodobé hodnocení hráčů. Proto se udržuje v souboru *elo.xml*. Hráčům, připojeným k Manažeru 2 poprvé, je přiřazeno počáteční hodnocení ELO 2000.

V pravém horním rohu je malý panel pro nastavení partií turnaje. Konkrétně lze zvolit počet her a limit na tah. Střídání hráčů v začínání v rámci partie je pro turnaje nastaveno automaticky. Začínající hráč první hry každé partie je vybrán náhodně. Sledovat partie v průběhu turnaje nelze, jelikož všechny hry turnaje probíhají současně.

Tlačítka pro ovládání jsou umístěny v panelu napravo. Nachází se zde samozřejmě tlačítka pro spuštění a ukončení turnaje. Turnaj lze spustit, jsou-li připojeni alespoň dva hráči. Ukončit ho lze kdykoli v jeho průběhu. Zbylých pět tlačítek v tomto panelu jsou zpřístupněny až po dohrání turnaje. První z nich slouží k uložení výsledku turnaje. Ten se ukládá do xml souboru a obsahuje kromě výsledku také dobu trvání turnaje v sekundách. Obsah takového souboru vypadá například takto:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<result tournament_duration="0,970">
  <player loses="0" name="xnovak03" rank="1" score="6" ties="0" wins="2"/>
  <player loses="1" name="xnovak01" rank="2" score="3" ties="0" wins="1"/>
  <player loses="2" name="xnovak02" rank="3" score="0" ties="0" wins="0"/>
</result>
```

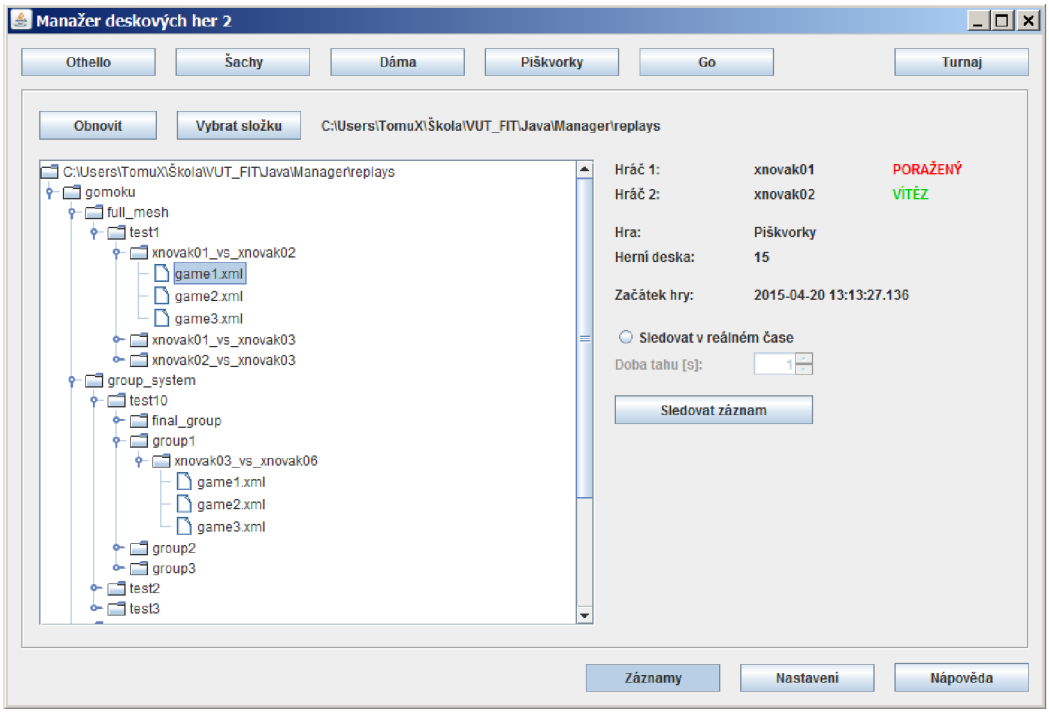
Dalším tlačítkem se dají uložit záznamy všech her turnaje (záznamům se věnuje kapitola 6.3). Před samotným uložením výsledku nebo záznamů Manažer 2 uživatele požádá o pojmenování turnaje. Tento název je pak použit pro jméno souboru s výsledkem nebo v případě

záznamů adresáře. Poslední tři tlačítka slouží k přepínání mezi turnajovými tabulkami a k jejich vymazání.

Tabulky se zobrazí po dohrání turnaje v záložkovém panelu uprostřed. Ten vyplňuje zbytek prostoru mezi ostatními panely stejně jako panel s herní deskou v režimu samostatné partie (při zvětšení okna se tedy panel s tabulkami také zvětšuje). Panel obsahuje tří záložky: partie, výsledek a ELO. První jmenovaná obsahuje tabulky s výsledky všech partií turnaje. Turnaj typu každý s každým má jen jednu, ale ostatní dva typy turnaje mají tabulek více a dá se mezi nimi přepínat již zmíněnými tlačítky. Švýcarský systém má tabulku každého kola. Po odehrání turnaje skupinovým systémem jsou vždy vytvořeny tabulky pro všechny skupiny a také finální tabulka, která je tvořená výsledky partií mezi postupujícími. Druhá záložka, výsledek, skrývá tabulku konečného umístění hráčů v turnaji. Je zde u každého hráče počet vyhraných, prohraných, remízových partií a jeho dosažené skóre, podle něhož se umístil (skóre se počítá stejným způsobem jako v původním manažeru – kapitola 2.3). Hráči se stejným skóre se umístí na stejné místo. Poslední záložka, ELO, je pro tabulku, která obsahuje změnu hodnocení ELO každého hráče (tj. ELO hráču před a po turnaji). Tato tabulka existuje pouze tehdy, bylo-li přepočítávaní hodnocení ELO aktivováno.

6.3 Záznamy

Přehrávání záznamů je poslední funkcí aplikace. Pro vstup do tohoto režimu slouží levé tlačítko ze spodní lišty okna. Na snímku 6.7 se můžete opět podívat jak tento režim vypadá.



Obrázek 6.7: Manažer deskových her 2 – záznamy

Jak si můžete všimnout, většinu prostoru zde zabírá komponenta *JTree*, která slouží k zobrazování hierarchicky uspořádaných dat. V tomto případě je použita pro procházení adresářové struktury s uloženými záznamy. Třída převádějící souborový systém na datový

model pro *JTree* komponentu se nazývá *FileSystemModel* a byla převzata z fóra věnující se programování v Javě [11]. Tlačítka nad touto komponentou slouží k aktualizaci stávajícího stromu a k výběru jiného adresáře se záznamy. Ukládat se dají pouze záznamy z turnajů a to příslušným tlačítkem po skončení turnaje (jak již bylo zmíněno v předchozí sekci).

Záznamy her se ukládají takovým způsobem, aby byly roztříděny do adresářů podle partie, ke které patří. Adresáře partií jsou rozděleny do složek podle skupin nebo kol pokud patří k turnaji skupinového nebo švýcarského typu. U turnajů typu každý s každým se partie už nijak neroztřiďují a jsou všechny přímo v adresáři turnaje, který pojmenovává uživatel. Složky jednotlivých turnajů jsou ještě také roztříděny do adresářů a to nejprve podle typu turnaje a poté podle hry.

Při označení xml souboru záznamu hry se v pravé části okna vyplní informace o hře. Konkrétně to jsou jména hráčů, informace o tom, který z nich vyhrál, typ hry, velikost herní desky a také datum a čas začátku hry. Poslední, co zde uživatel nalezne, je nastavení časové prodlevy mezi tahy, s kterou bude záznam sledován. Pak už zbývá jen kliknout na tlačítko *Sledovat záznam* a Manažer 2 uživatele vezme do režimu samostatné partie. Zde je odstartována partie o jedné hře odpovídající spuštěnému záznamu. Před stisknutím tlačítka pro sledování záznamu však musí být herní deska volná. To znamená, že v režimu samostatné partie příslušné hry, nesmí být rozehraná partie ani být připojen žádný hráč. V opačném případě je uživatel na tuto skutečnost upozorněn dialogem a záznam se nespustí. Položka *Typ*, která určuje typ právě připojeného hráče, je v případě sledování záznamu u obou hráčů vyplněna hodnotou *Záznam*.

6.4 Nastavení a nápověda

Do nastavení Manažeru 2 se uživatel dostane stlačením prostředního tlačítka ve spodní liště okna. Lze zde nastavit adresáře pro ukládání záznamů a výsledků turnajů. Při výchozím nastavení jsou použity složky *Results* a *Replays* v adresáři s aplikací.

Další věcí, kterou lze zvolit, je jazyk. Manažer 2 podporuje češtinu a angličtinu (výchozí je čeština). Multijazyčnost je realizována pomocí tříd *ResourceBundle*, *Locale* a souborů s příponou *properties*. Princip této techniky není složitý. Soubory s příponou *properties* obsahují všechny textové řetězce použité v aplikaci systémem klíč = hodnota. Pro každý podporovaný jazyk existuje jeden takový soubor. Všechny obsahují stejné klíče, akorát hodnoty (textové řetězce) se samozřejmě liší. K požadovaným řetězům se z programu přistupuje přes klíč pomocí instance třídy *ResourceBundle* [10]. Přidaní podpory dalších jazyků do Manažeru 2 tedy vyžaduje pouze překlad textových řetězců a už jen minimální zásah do zdrojového kódu aplikace.

Poslední akcí, kterou je možné v nastavení provést, je změna portu, na kterém server Manažeru 2 naslouchá. Aby se tato změna provedla, je třeba server restartovat. Pro tuto akci je zde příslušné tlačítko. Výchozí port je 9000.

Tyto nastavení se po vypnutí aplikace ukládání do souboru *settings.xml* umístěném ve stejném adresáři jako Manažer 2. Při opětovném spuštění je toto nastavení načteno a použito.

Z aplikace je přístupná i nápověda k Manažeru 2 a to skrz prohlížeč, který se pustí po stisknutí tlačítka v pravém dolním rohu okna. Nápověda tedy existuje ve formě HTML souboru, podobně jak tomu bylo i u Manažeru 1. Tato nápověda však zatím bohužel neobsahuje žádné užitečné informace a je třeba ji dopsat.

6.5 Propojení Manažeru s brainy

Spouštění brainů a následná komunikace s nimi je tou hlavní činností, kterou aplikace Manažer 2 vykonává. Pojďme si tedy tuto oblast trochu více přiblížit co se týče implementace.

Manažer 2 byl vytvořen tak, aby fungoval se stejnými brainy jako Manažer 1. Ovšem, jak už bylo zmíněno v sekci 5.3, bylo třeba je upravit, aby fungovaly jako klienti. Nyní program brainu očekává dva parametry, adresu a port, které použije pro připojení k Manažeru 2. Brainy jsou zatím vytvořeny tak, že neumí hrát více her současně. Proto, aby mohly v rámci turnaje probíhat všechny hry najednou, je třeba spustit brain tolikrát, kolik hraje her.

Po spuštění Manažeru 2 se kromě načtení nastavení aplikace a zobrazení GUI vytvoří také serverový soket (objekt třídy *ScoketServer*). Dále už Manažer 2 pouze reaguje na akce uživatele. Při připojení brainu, ať už do turnaje nebo k samostatné partii, je třeba jej poprvé spustit a navázat s ním komunikaci kvůli zjištění jména a abychom ověřili, že funguje. Poté je brain hned odpojen a ukončen.

K spouštění lokálních brainů je použita třída *ProcessBuilder* a do parametru s adresou je předán *localhost* (tj. adresa právě používaného počítače). Vzdálené brainy jsou spouštěny přes službu SSH pomocí knihovny *JSch* [13]. U připojení vzdálených brainů přes internet však nastává drobná komplikace. Ta souvisí s tím, že Manažer 2 představuje server a pokud chceme, aby byl Manažer 2 přístupný i z internetu, měl by mít počítač, na kterém běží, veřejnou IP adresu. Pokud ji nemá, musí to uživatel řešit například přesměrováním portu, na němž Manažer 2 naslouchá. Přesměrování komunikace přes určitý port se dá zařídit v nastavení směrovače (routeru). Brainu se tedy musí do parametru předat veřejná IP adresa počítače (ta pod kterou počítač vystupuje na internetu). Tu Manažer 2 zjišťuje při vytváření serverového soketu a to pomocí služby dostupné z `http://checkip.amazonaws.com/`, která má tuto funkci. Pomocí třídy *URL* se Manažer 2 spojí s tímto zdrojem a přečte z něj IP adresu, pod kterou je vidět v internetu.

Po spuštění brainu čeká Manařer 2 deset sekund na jeho připojení. Pokud se při tomto prvním připojení kvůli zjištění jména do této doby nepřipojí, je tento neúspěch oznámen uživateli chybovým dialogem. Po úspěšném připojení brainu již máme k dispozici objekt hráče s cestou k souboru a jménem, který brain reprezentuje. Tento brain je tedy považován za připojeného, protože je ověřeno, že funguje a lze jej kdykoli spustit.

Každý hráč může mít více tzv. komunikátorů. Komunikátor je v Manažeru 2 objekt, který zajišťuje komunikaci s brainem (posílá brainu informace o hře a zjišťuje jeho tahy). Každý komunikátor představuje jeden spuštěný brain nebo-li jednu momentální účast ve hře. Komunikátory jsou vytvářeny při vytvoření hry a připojovány k brainům při jejím spuštění. Po konci hry jsou brainy odpojeny a vypnuty. Pokud se brain po jeho spuštění před hrou nepřipojí do deseti sekund a jeho protihráč ano, hru prohrává. Pokud se nepřipojí ani jeden z hráčů, končí hra remízou.

Propojení brainu s komunikátorem zajišťuje serverový soket, který vyčká na připojení a vytvoří soket pro komunikaci s brainem (tuto funkci obstarává metoda *accept* třídy *ScoketServer*).

6.6 Paralelismus

Jak už bylo řečeno v sekci 5.1, paralelismu je dosaženo tím, že třídy, představující partii a hru, rozšiřují třídu *Thread*. Každá hra proto běží ve svém vlákně. Ideálně by tedy odehrání celého turnaje trvalo stejně dlouho jako jeho nejdelší hra. To však samozřejmě není reálné, protože ve skutečnosti úplně všechna vlákna běžet zároveň nemohou. Počet vláken, která

jsou doopravdy prováděna současně, závisí na počtu procesorů a jader výpočetního systému. Cenou za dosažení rychlejšího průběhu turnajů díky paralelismu jsou tedy pochopitelně větší nároky na výpočetní sílu počítače. Paralelismus však přináší jisté potíže, které bylo třeba řešit, a o těch si v této kapitole povíme.

V předchozí sekci jsme se dozvěděli, že na začátku každé hry jsou připojovány komunikátory, což znamená spouštění a připojování brainů. Serverový soket je však jen jeden a připojení brainů musí akceptovat postupně. Musí být zajištěno, aby se ke každému komunikátoru připojil správný brain. Proto je část kódu se spouštěním brainu a následným voláním metody *accept* sekcí s výlučným přístupem. Dalšími sekcemi s výlučným přístupem musejí být některé metody třídy představující hráče (brain). Patří mezi ně metody pro úpravu počtu výher, proher a remíz nebo úpravu hodnocení ELO. Tyto atributy hráče jsou totiž měněny po každé hře či partii a tudíž z více vláken současně. Také pro seznam komunikátorů hráče bylo třeba použít speciální implementaci seznamu *synchronizedList*, která zajišťuje bezpečnou práci se seznamem ve vícevláknovém prostředí.

Na začátku této sekce bylo zmíněno, že odehrávání turnajů takto masivně paralelním způsobem vyžaduje výkonnější počítač. Počet vláken, které mohou být naráz spuštěny, není nekonečný a závisí na platformě. Přesažení tohoto limitu dává najevo virtuální stroj Javy chybou *OutOfMemoryError: unable to create natiwe thread*. Tato chyba je v programu odchytávána a řešena jednoduše čekáním, dokud se některé již běžící vlákno nedokončí a neuvolní místo pro vytvoření dalšího. Stejně tak je tento způsob náročný i na paměť počítače. To hlavně proto, že každá hra si ukládá všechny tahy hráčů, aby bylo možné po skončení turnaje vytvořit její záznam. S tímto souvisí zase chyba *OutOfMemoryError: java heap space*, která už se v programu příliš ošetřit nedá. Znamená to jednoduše nedostatek paměti pro vytvoření nových objektů. Šlo by Manažer 2 optimalizovat, aby používal méně paměti. Například by se tahy her nemuseli ukládat, ale mohli by být soubory se záznamy vytvářeny rovnou při průběhu her. Nebo lze jednoduše navýšit velikost paměti, kterou virtuální stroj Javy používá pro alokaci (takto byl tento problém doposud řešen). Toho docílíme třeba tím, že Manažer 2 spustíme s přepínačem *-Xmx*. Například pokud chceme poskytnout Manažeru 2 pro alokaci paměť o velikosti 512 MB, spustíme ho s přepínačem *-Xmx512M*. Výchozí nastavení tohoto parametru virtuálního stroje Javy závisí na platformě a lze jej zjistit pomocí příkazu *java -XX:+PrintFlagsFinal -version*.

Jako nepříjemnost se projevil fakt, že brainů je při turnaji spuštěno příliš mnoho. Při vypnutí velkého turnaje při jeho průběhu totiž trvá poměrně dlouho, než se všechny brainy ukončí. Manažer 2 proto po ukončení turnaje uživateli dovolí pracovat s GUI, přestože ještě všechny brainy nejsou vypnuty. Uživatel by tento nedostatek měl brát na vědomí a nespouštět další turnaj dokud se veškeré brainy předchozího turnaje zcela neukončí. Navíc pokud aplikace Manažeru 2 z nějakého důvodu spadne v průběhu turnaje a brainy zůstanou spuštěné, je velmi nepříjemné je všechny ukončovat ručně nebo být nucen kvůli toho restartovat počítač. Některé počítače navíc mohou mít nastaveno omezení pro počet spuštěných programů. Těchto potíží by se šlo s trochou úsilí v budoucnu také zbavit. A to vytvořením brainů, které by dokázaly hrát více her současně. Manažer 2 by bylo samozřejmě nutno trochu upravit a takovým brainům přizpůsobit. Výsledek by však zato určitě stál, protože při turnaji by pak stačilo spustit každý brain pouze jednou.

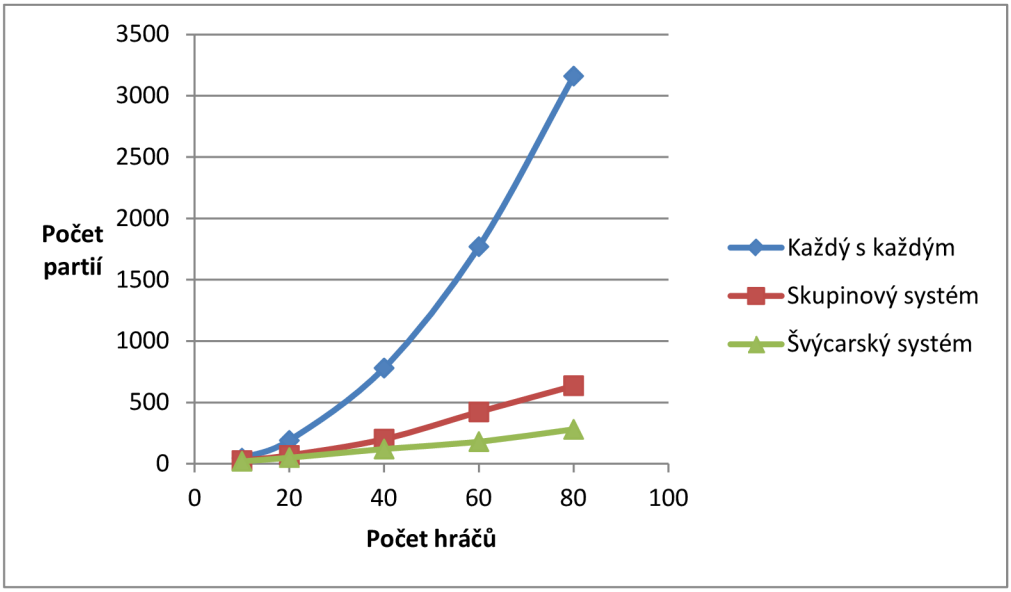
Kapitola 7

Testování

Protože v Manažeru 2 je zatím plně funkční pouze hra piškvorky, budeme se v této kapitole zabývat pouze turnaji v této hře. K dispozici jsou zatím jen dva druhy brainů, které byly převzaty z bakalářské práce Jana Kouřila a upraveny pro použití s Manažerem 2. Jeden pro operační systémy Linux a druhý pro Windows. Aby bylo možné Manažer 2 otestovat, bylo nutné upravené brainy namnožit tak, abychom získali více brainů s různým jménem. Testování tedy probíhalo s brainy, jejichž tahy jsou řízeny stejným algoritmem. Ve výsledku to znamená, že všechny hry dopadají stejně a o vítězi rozhoduje pouze to, který z brainů začíná. To však pro účel testování vůbec nevadilo. Cílem bylo zhodnotit dobu trvání turnaje v závislosti na jeho parametrech a také na počtu jader počítače.

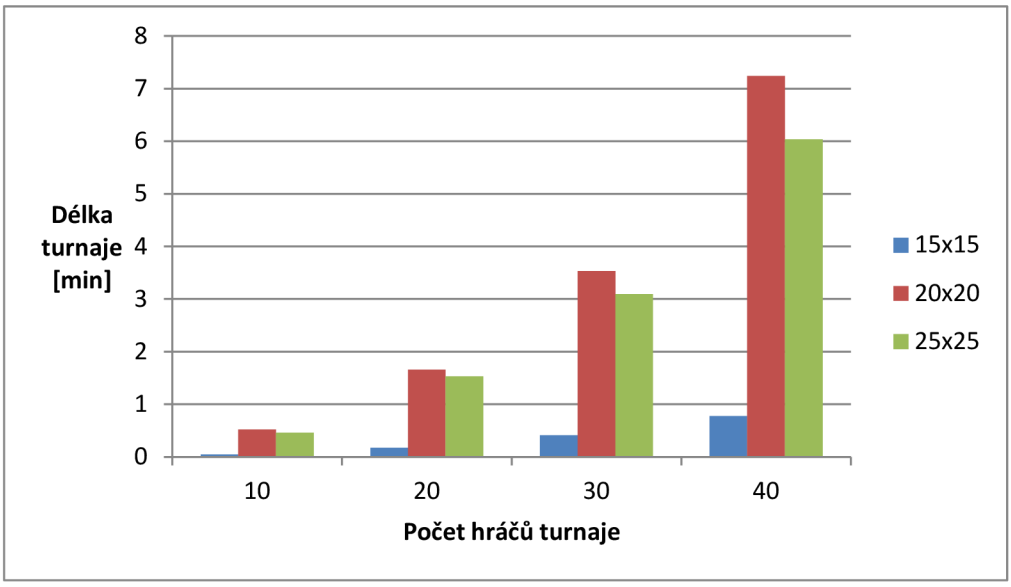
Tím hlavním parametrem turnaje, který ovlivňuje jeho délku, je počet her. Počet her turnaje však uživatel nemůže nastavit přímo, ale závisí na typu turnaje, počtu hráčů a počtu her každé partie. Typ turnaje má v tomto ohledu velký význam. Závislost počtu partií turnaje na množství hráčů je znázorněna grafem 7.1. Vidíme, že při turnaji typu každý s každým počet partií stoupá nejrychleji. Na druhou stranu se při tomto typu turnaje nejvíce uplatní paralelismus, protože mohou být odehrávány současně úplně všechny hry. Opakem je švýcarský systém turnaje. Ten využívá paralelismu nejméně, jelikož musí být odehráván po kolech. Není u něj však třeba odehrávat velké množství her. U turnaje skupinového typu hodně závisí na volbě počtu skupin a postupujících. Například kdybychom vytvořili jen jednu skupinu, turnaj by obsahoval ještě více partií než stejný turnaj typu každý s každým. Při vhodném rozdělení hráčů do skupin se však skupinový systém v počtech partií turnaje blíží více turnajům švýcarského typu. Při testování byla snaha použít co nejvhodnější počty skupin a postupujících. U turnajů pro 10, 20 a 40 hráčů byly zvoleny skupiny po pěti s dvěma postupujícími a pro 60 a 80 hráčů skupiny po deseti s třemi postupujícími.

Délka turnaje závisí také na tom, jak dlouho trvá jedna hra. To kromě algoritmu brainů lze ovlivnit časovým limitem na tah a u piškvorek také velikostí herní desky. Testovat délku turnaje v závislosti na povolené době tahu však zatím nebylo možné. Algoritmy testovacích brainů, které rozhodují další tah, jsou totiž poměrné jednoduché. Produkují odpověď příliš rychle, takže časový limit, který lze v Manažeru 2 nastavit nejméně na jednu sekundu, nestačí vyprchat. Navíc brainy sice informaci o limitu na tah přijmou, ale algoritmus na něj nebere ohled. Srovnání doby trvání turnajů odehrávaných na různě velkých herních deskách však provedeno bylo. A to s brainy pro Windows hrající turnaj typu každý s každým. Výsledky jsou zaznamenány v grafu 7.2. To, jak dlouho trvá hra na herní desce určité velikosti, je věcí algoritmu brainů. Testovací brainy hrají všechny úplně stejně, takže lze z grafu vyčíst, že hra na ploše 15x15 jim zabere nejkratší dobu a hra na ploše 20x20 jim trvá ještě déle než na ploše o velikosti 25x25. Čím více hráčů se turnaje účastní, tím více her



Graf 7.1: Závislost počtu partií turnaje na počtu hráčů turnaje

je třeba odehrát a tím více se na výsledném čase projeví velikost herní desky. Při reálném turnaji s různými brainy by však byly tyto výsledky nepředvídatelné, protože brainy by obsahovaly odlišné algoritmy a hry by jim trvaly různě dlouho.

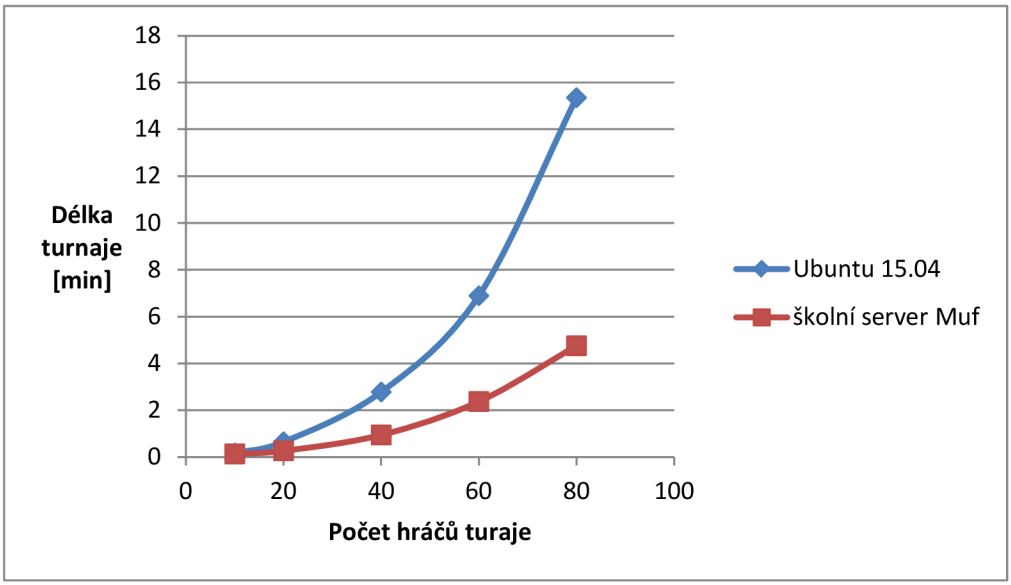


Graf 7.2: Srovnání doby trvání turnajů odehrávaných na různě velkých herních deskách

Poslední věcí, která podstatně rozhoduje o době trvání turnajů, je výkon počítače a také míra využití paralelismu. Tedy hlavně počet procesorů a jejich jader. Manažer 2 byl testován celkově na třech počítačích:

- Windows 7, 4 jádra
- Ubuntu 15.04 Linux, 4 jádra
- školní server Muf - Debian Linux, 16 jader

Testovací brainy pro Windows však srovnávat s těmi pro Linux v závislosti na počtu jader nelze. Linuxové brainy hrají totiž mnohem pomaleji. Konkrétně jedna hra na herní desce o velikosti 15x15 s testovacími brainy pro Linux trvá zhruba 4,5 s a s těmi pro Windows jen 0,1 s. Zbývá tedy možnost porovnat délky turnajů s linuxovými brainy spuštěnými na osobním počítači se čtyřmi jádry a na školním serveru Muf s šestnácti jádry. Pro tento test byl zvolen opět turnaj každý s každým, protože ten využívá paralelismu nejvíce. Výsledek je zachycen grafem 7.3. Bohužel se zatím nepodařilo sehnat přístup k více počítačům s různým počtem jader, na kterých by šlo Manažer 2 testovat. Výsledek tak nemá takovou vypovídající hodnotu jakou by mít mohl.



Graf 7.3: Srovnání doby trvání turnajů na osobním počítači a serveru Muf

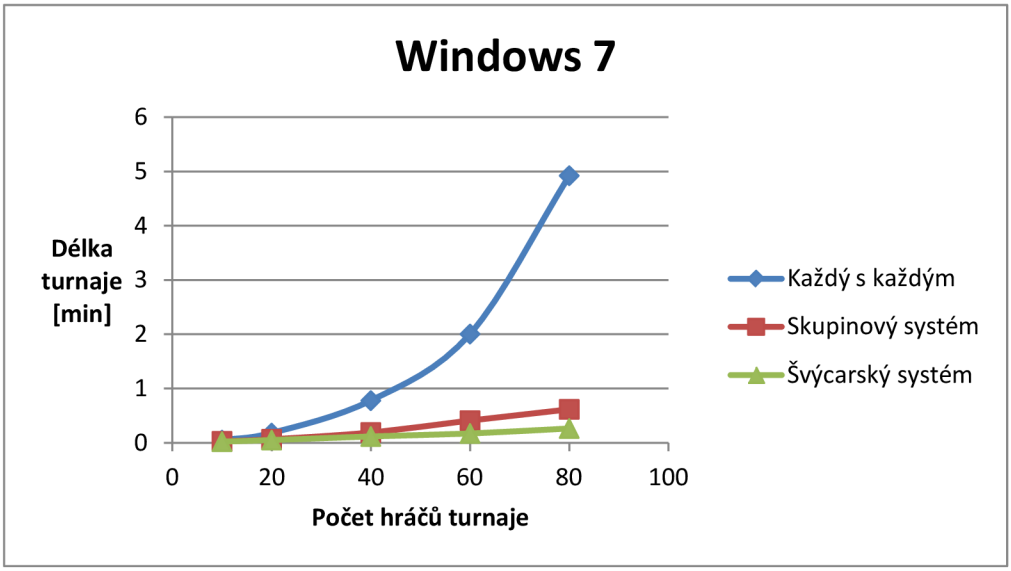
Grafy 7.4, 7.5 a 7.6 zachycují výsledky testování na jednotlivých platformách. Obsahují srovnání trvání jednotlivých typů turnajů v závislosti na počtu hráčů. Turnaje probíhaly na herní desce o velikosti 15x15 a s partiemi o jedné hře. Všimněme si, že všechny tyto tři grafy mají podobný charakter. Důležité jsou však výsledné časy, které se pohybují v jiných hodnotách. Časy na operačním systému Windows 7 a serveru Muf jsou oproti výsledkům na linuxové distribuci Ubuntu 15.04 mnohem nižší. Příčina je však různá. Zatímco na Windows 7 jsou výsledky lepší kvůli rychlejším brainům, server Muf disponuje větším počtem jader.

Když to shrneme, doba trvání turnaje závisí na třech faktorech. Především je to počet her, který je určen počtem hráčů a typem turnaje. V druhé řadě záleží na výkonnosti

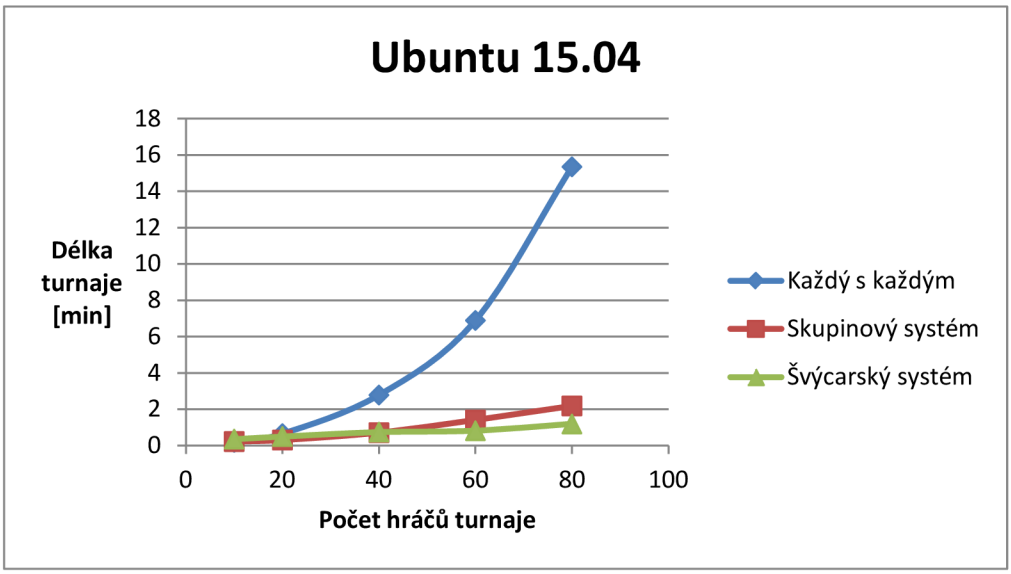
počítače, na kterém jsou Manažer 2 a brainy spouštěny. Posledním faktorem jsou brainy samotné. Jde především o počet tahů potřebný k poražení soupeře a také o to, jak dlouho jim tahy trvají. Případně jaký časový limit jim uživatel nastaví.

Užitečné by bylo, na základě nastavených parametrů turnaje a počtu jader počítače, přibližně vypočítat délku turnaje, a tuto informaci předat uživateli. To by si však zcela jistě vyžadovalo provést mnohem podrobnější testování s různými a kvalitnějšími brainy, jenž by hlavně braly v potaz zadaný limit tahu.

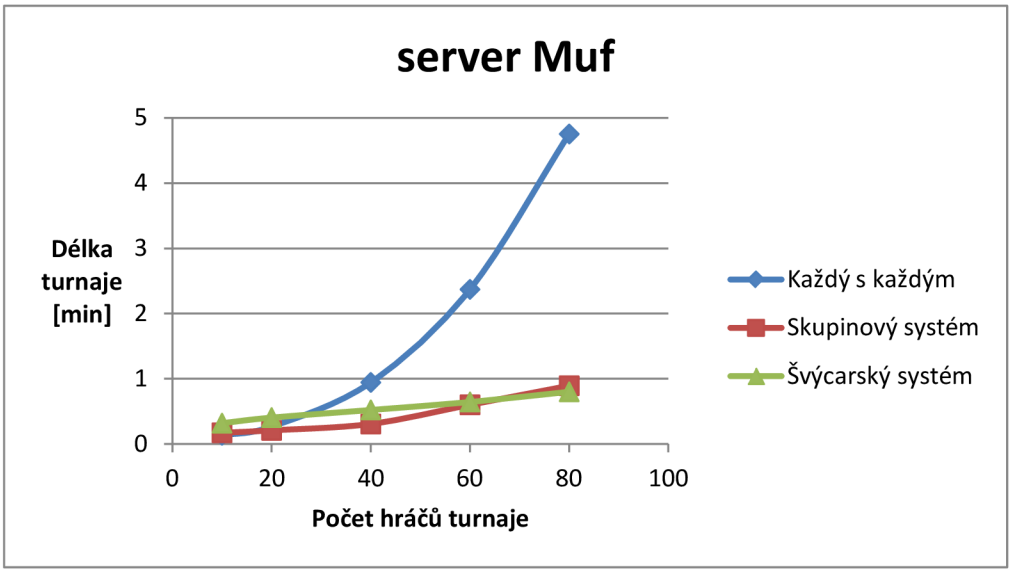
Testování velkých turnajů probíhalo pouze s brainy připojenými lokálně. Manažer 2 totiž zatím neovládá autentizaci pomocí klíčů, která je nutná pro vzdálený přístup na školní server Muf. Přístup k jinému serveru bez omezení na počet spuštěných procesů nebo programů se zatím získat nepodařilo. Proto bylo nutné testování velkých turnajů s brainy připojenými vzdáleně odložit a přenechat někomu dalšímu.



Graf 7.4: Srovnání trvání jednotlivých typů turnajů v závislosti na počtu hráčů (osobní počítač s Windows 7)



Graf 7.5: Srovnání trvání jednotlivých typů turnajů v závislosti na počtu hráčů (osobní počítač s Ubuntu 15.04)



Graf 7.6: Srovnání trvání jednotlivých typů turnajů v závislosti na počtu hráčů (školní server Muf)

Kapitola 8

Závěr

Cílem této práce bylo vytvořit novou verzi programu Manažer pro deskové hry, která by umožňovala pořádání turnajů s paralelním odehráváním her. Cíl se splnit podařilo a nový Manažer pro deskové hry 2 neztratil téměř žádnou funkci svého předchůdce. Naopak disponuje i některými drobnostmi navíc. Například je schopen běhu pod různými operačními systémy a v aplikaci je zavedena i podpora vícejazyčnosti (zatím je k dispozici jen čeština a angličtina). Manažer 2 byl v rámci práce také otestován nejen na osobním počítači, ale také na školním šestnácti jádrovém serveru Muf. Bohužel se však nepodařilo stihnout dokončit všechny podporované hry, a tak jsou v Manažeru 2 plně funkční zatím jen piškvorky. Taktéž je potřeba dopsat uživatelskou nápovědu, která je z aplikace přístupná, ale zatím neobsahuje žádné informace.

Z osobního hlediska mi tato práce přinesla nové zkušenosti a znalosti z oblasti programování v jazyce Java. Také mi umožnila se zlepšit v programování vícevláknových aplikací, s čímž jsem předtím měl jen minimální zkušenosti. Díky tomu, že samotnému programování předcházelo seznámení s předchozí verzí aplikace, jsem si plně uvědomil důležitost zásad čistého a přehledného kódu a také důležitost dobrého objektového návrhu aplikace. Přesto, že se těmto věcem stále učím, snažil jsem se být v tomto ohledu lepší než mí předchůdci. Protože jsem doposud na podobně velké aplikaci nepracoval, byla pro mne celkově tato zkušenost velkým přínosem.

Na Manažeru 2 se samozřejmě, kromě dokončení již zmíněného, dá dále pracovat. Například přidáním dalších deskových her nebo jiných typů turnajů. Především by se však mělo zapracovat v oblasti brainů a vytvořit takové, které by dokázaly hrát více her současně. Manažer 2 by mohl také podporovat více komunikačních protokolů, aby měli tvůrci brainů v tomto ohledů více volnosti. Ve vývoji Manažeru pro deskové hry 2 by se mělo určitě pokračovat, protože tento projekt jistě ještě nedosáhl svého plného potenciálu.

Literatura

[1] Almasi, G. S.; Gottlieb, A.: *Highly parallel computing*. Benjamin-Cummings Publishing Company, 1989, ISBN 0-8053-0177-1.

[2] Amitkumar, M.; Pankaj, K.; Malati, N.: A Review on New Paradigm's of Parallel Programming Models in High Performance Computing. *International Journal of Computer Science and Network*, ročník 1, č. 4, August 2012: s. 53–56.
URL http://ijcsn.org/IJCSN-2012/1-4/IJCSN-2012-1-4-41.pdf

[3] Barney, B.: Introduction to Parallel Computing [online]. 2014-11-12 [cit. 2015-01-12].
URL https://computing.llnl.gov/tutorials/parallel_comp/

[4] Barney, B.: OpenMP [online]. 2014-11-12 [cit. 2015-01-12].
URL https://computing.llnl.gov/tutorials/openMP/#ProgrammingModel

[5] Barney, B.: Message Passing Interface (MPI) [online]. 2014-12-15 [cit. 2015-01-12].
URL https://computing.llnl.gov/tutorials/mpi/#What

[6] Carver, R. H.; Tai, K.-C.: *Modern Multithreading: Implementing, Testing, and Debugging Multithreaded Java and C++/Pthreads/Win32 Programs*.
Wiley-Interscience, 2005, ISBN 978-0-471-72504-6.

[7] Fenwick, K.: *A Performance Analysis of Java Distributed Shared Memory Implementations*. diploma thesis, Adelaide University, October 2001.
URL http://www.academia.edu/551112/A_Performance_Analysis_of_Java_Distributed_Shared_Memory_Implementations

[8] Java: java.util.concurrent [online]. © 1993-2014 [cit. 2015-01-12].
URL http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/package-summary.html

[9] Java: Lesson: Concurrency [online]. © 1995-2014 [cit. 2015-01-12].
URL https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

[10] Java: About the ResourceBundle Class [online]. © 1995-2014 [cit. 2015-05-03].
URL https://docs.oracle.com/javase/tutorial/i18n/resbundle/concept.html

[11] JavaPF: How to Use JTree to create a File System Viewer Tree [online]. 2011-03-15 [cit. 2015-05-03].
URL http://www.javaprogrammingforums.com/java-swing-tutorials/7944-how-use-jtree-create-file-system-viewer-tree.html

[12] Javier, D.; Muñoz-Caro, C.; Niño, A.: A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era. *IEEE Transactions on Parallel & Distributed Systems*, ročník 23, č. 8, August 2012: s. 1369–1386, doi:10.1109/TPDS.2011.308.
URL http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.308

[13] JCraft: JSch - Java Secure Channel [online]. © 1998-2014 [cit. 2015-05-08].
URL http://www.jcraft.com/jsch/

[14] Kaldéren, G. A.; From, A.: *A comparative analysis between parallel models in C/C++ and C#/Java*. bakalářská práce, KTH Information and Communication Technology, Stockholm, Švédsko, 2013.
URL http://www.diva-portal.org/smash/get/diva2:648395/FULLTEXT01.pdf

[15] Kouřil, J.: *Manažer stolních deskových her*. bakalářská práce, FIT VUT v Brně, Brno, 2009.
URL http://www.fit.vutbr.cz/study/DP/BP.php.cs?id=8301&file=t

[16] Kusák, J.: *Manažer stolních deskových her*. bakalářská práce, FIT VUT v Brně, Brno, 2007.
URL http://www.fit.vutbr.cz/study/DP/BP.php.cs?id=5488&file=t

[17] Lehey, G.: *Porting UNIX Software*. O'Reilly, 1995, ISBN 978-1-56592-126-9.

[18] MPI: MPI Documents [online]. 2012-09-21 [cit. 2015-01-12].
URL http://www.mpi-forum.org/docs/

[19] OpenMP: OpenMP Specifications [online]. 2013-07-23 [cit. 2015-01-12].
URL http://openmp.org/wp/openmp-specifications/

[20] Schildt, H.: *Java 7 Výukový kurz*. Computer Press, 2012, ISBN 978-80-251-3748-2.

[21] Silberschatz, A.; Galvin, P. B.; Gagne, G.: *Operating system concepts (9th ed.)*. John Wiley & Sons, Inc., 2012, ISBN 978-1-118-06333-0, 181–182 s.

[22] Snir, M.; Otto, S.; Huss-Lederman, S.; aj.: MPI: The Complete Reference [online]. 1995 [cit. 2015-01-12].
URL http://www.netlib.org/utk/papers/mpi-book/mpi-book.html

[23] Taral, P.: Java is portable - reason [online]. © 2009-2014 [cit. 2015-01-12].
URL http://www.c4learn.com/java/why-java-is-platform-independent-and-portable/

[24] VFSJFileChooser team: VFSJFileChooser [online]. 2012-12-18 [cit. 2015-05-01].
URL http://vfsjfilechooser.sourceforge.net/

Příloha A

Obsah CD

- **brainy/** – Složka se zdrojovými soubory brainů.
- **kostry brainů/** – Složka s kostrami brainů.
- **Manažer/** – Složka obsahující spustitelný soubor aplikace, Manager.jar.
- **Manažer zdrojové soubory/** – Složka se zdrojovými soubory aplikace a soubory potřebnými ke spuštění ve vývojovém prostředí NetBeans IDE.
- **technická zpráva.pdf** – Technická zpráva ve formátu pdf.
- **technická zpráva zdrojové soubory/** – Složka se zdrojovými soubory technické zprávy.