UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Desková hra Go pro více klientů

Petr Lokvenc

Bakalářská práce
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Petr Lokvenc**
Osobní číslo:	**I09180**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Desková hra Go pro více klientů**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem práce je vytvořit desktopovou aplikaci pro hraní čínské deskové hry Go po místní síti s možností turnaje více hráčů.

Teoretická část:
Úvodní část bude obsahovat seznámení s pravidly hry Go a jejich použití v aplikaci, dále použití architektury klient-server pro více uživatelů s využitím tříd pro socketovou komunikaci.

Implementační část:
Implementační část bude obsahovat aplikaci Go, jež lze na místní síti založit. Aplikace umožní založit turnaj, jež může zakladatel zahájit po připojení dostatečného počtu hráčů. Průběžný stav turnaje bude graficky znázorňován turnajovým pavoukem. Aplikace bude obsahovat nápovědu s popisem pravidel a ovládání.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

SOO-HYUN, Jeong, [translated by] Janice KIM a Drawings by a LEE. The Korean Go Association's Learn to play go: a master's guide to the ultimate game. 2nd ed. New York, NY: Good Move Press, 1994. ISBN 09-644-7961-3.
BLOCH, Joshua. Java efektivně: 57 zásad softwarového experta. Vyd. 1. Praha: Grada, 2001, 230 s. ISBN 80-247-0416-1.
HAROLD, Elliotte Rusty. Java network programming. 3rd ed. Sebastopol, Calif.,: O'Reilly, c2005, xxii, 735 p. ISBN 05-960-0721-3.

Vedoucí bakalářské práce: **Ing. Zdeněk Šilar**
Katedra informačních technologií

Datum zadání bakalářské práce: **21. prosince 2012**
Termín odevzdání bakalářské práce: **10. května 2013**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 29. března 2013

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 9. 8. 2013 Petr Lokvenc

Poděkování

Chtěl bych poděkovat Ing. Zdeňku Šilarovi za odbornou pomoc při vypracování této práce. Dále bych chtěl poděkovat mému otci za psychickou a hmotnou podporu během studia a mé sestře Monice za pomoc při testování praktické části mé bakalářské práce. Děkuji mému spolužákovi Vratislavu Markovi za podporu a ochotnou pomoc během studia.

Anotace

Cílem práce je vytvořit desktopovou aplikaci pro hraní čínské deskové hry Go po místní síti s možností turnaje více hráčů. Teoretická část obsahuje seznámení s pravidly hry Go a jejich použití v aplikaci, dále použití architektury klient-server pro více uživatelů s využitím tříd pro soketovou komunikaci. Implementační část obsahuje popis aplikace Turnaj v Go. Aplikace umožňuje založit turnaj, jejž může zakladatel zahájit po připojení dostatečného počtu hráčů. Průběžný stav turnaje je graficky znázorňován turnajovým pavoukem. Aplikace obsahuje nápovědu s popisem pravidel a ovládání.

Klíčová slova

Java, Go, desková hra, sokety, datagramy, JDeveloper

Title

Go board game for multiple clients.

Annotation

The objective of bachelor’s thesis is to make application for Chinese board game Go playable via local network with option of hosting tournament for multiple players. Theoretical part describes rules of Go and their usage in application and client-server architecture for multiple clients using classes for socket communication. Implementation part describes application Turnaj v Go. Application allows hosting of tournament, which can be started after sufficient number of players joins the game. Current status of tournament is graphically represented by tournament diagram. Application contains help with description of game rules and controls.

Keywords

Java, Go, Board game, Sockets, Datagrams, JDeveloper

Obsah

Seznam zkratek

AGA	American Go Association
GUI	Graphical User Interface
IDE	IntegratedDevelopmentEnvironment
IP	Internet Protocol
JDK	Java DevelopmentKit
JVM	Java VirtualMachine
LAN	Local Area Network
TCP	TransmissionControlProtocol
UDP	User Datagram Protocol

Seznam obrázků

Úvod

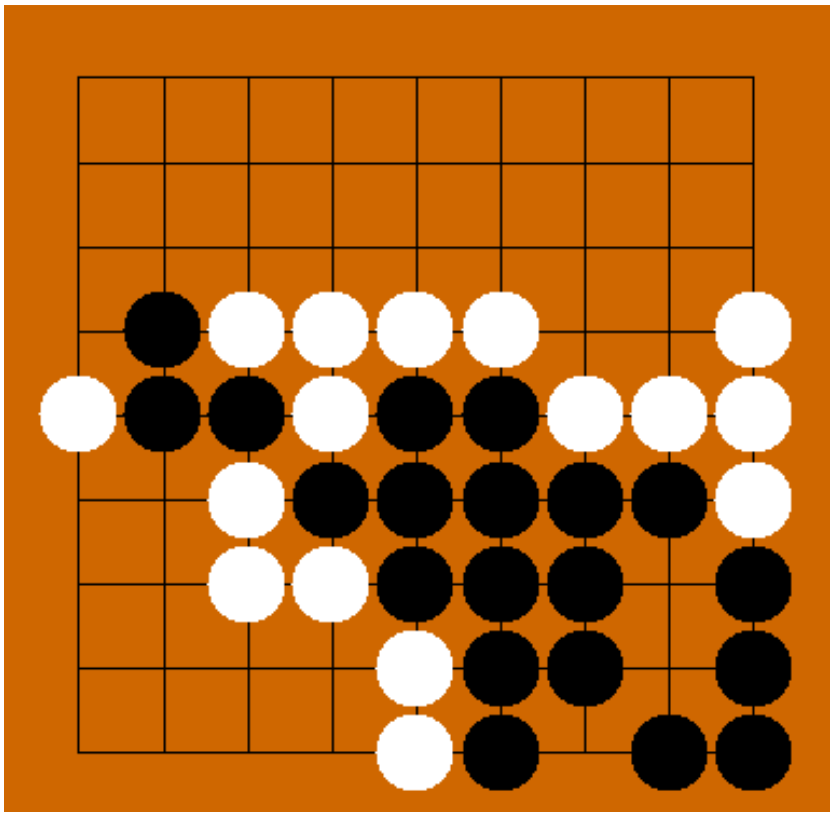
Turnaj v Go je desktopová aplikace naprogramovaná v jazyce Java, která umožňuje hrát čínskou deskovou hru Go dvěma nebo více hráči. Hru je možné hrát po místní síti (LAN). Lze založit turnaj, nebo se připojit k založenému turnaji, pro 2,4,8,16, nebo 32 hráčů. Aplikace obsahuje herní obrazovku pro zakládání a vyhledávání her na místní síti, hrací desku, grafické znázornění turnaje pomocí turnajového pavouka a nápovědu ohledně pravidel a ovládání.

V této práci budete seznámeni se základními pravidly hry Go a důležitými částmi aplikace. Dále se dozvíte, jak funguje architektura klient-server s využitím tříd pro soketovou komunikaci.Jsou zde popsány i algoritmy pro implementaci jednotlivých pravidel.

Deskovou hru Go jsem si vybral, jelikož obsahuje jednoduchá, ale přesto důmyslná pravidla.Průběh hry má mnoho variací (vzhledem k tomu, že kameny se nejen pokládají, ale i odebírají) a je nenáročná na grafickou implementaci.

1 Hra Go

Hra,jež pochází ze starověké Číny,nejčastěji známá pod japonským názvem Go, je desková tahová hra pro 2 hráče. Hraje se s černými a bílými kameny, kdy každému hráči náleží jedna barva.Cílem hry je mít na konci hry větší počet bodů než protihráč. Body hráč získá zajímáním protihráčových kamenů a za ohraničené území na konci hry.(Matthews, 1999)



Obrázek 1 – Ukázka hry Go

1.1 Důležitá pravidla

Hra Go má několik pravidel. Některá jsou nezbytná pro vlastní hru, jiná jsou doplňující a používají se na oficiálních turnajích pro vyvážení obtížnosti mezi hráčem začínajícím a pokročilým. Zde jsou uvedena ta základní.

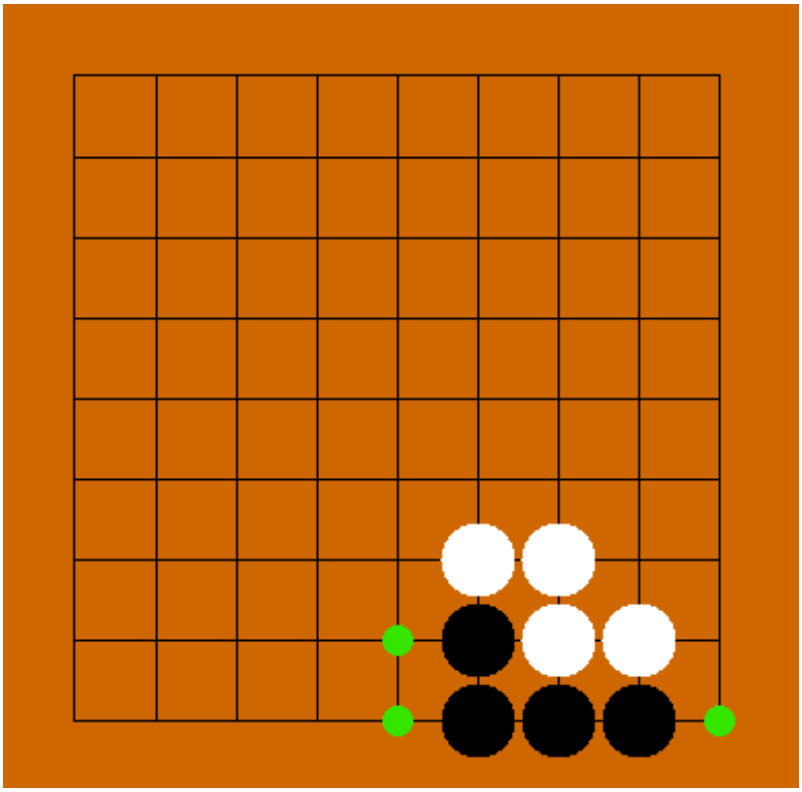
Hrací deska a tahy hráčů

Hraje se na čtvercové hrací desce zvané „*goban*“ s mřížkou tvořenou 9, 13 nebo 19 horizontálními a stejným počtem vertikálních čar.Deska s mřížkou 9x9 čar odpovídá šachové hrací desce 8x8 hracích polí. Kameny se pokládají na průsečíky čar, nikoli na hrací pole.

Hra začíná tahem hráče s černými kameny, dále se hráči v tazích střídají. Hráči postupně táhnou na jednotlivé body hrací desky. Hráč se může svého tahu vzdát a táhnout může protihráč.(Iwamoto, 1976)

Skupiny a svobody

Skupinu kamenů tvoří jeden nebo více kamenů stejné barvy, které spolu horizontálně nebo vertikálně sousedí. Důležitou částí jsou takzvané svobody, jež se vztahují na skupinu kamenů. Svobody jsou volné body horizontálně či vertikálně sousedící se skupinou kamenů, na obrázku (Obrázek 2) jsou označeny zelenou barvou.

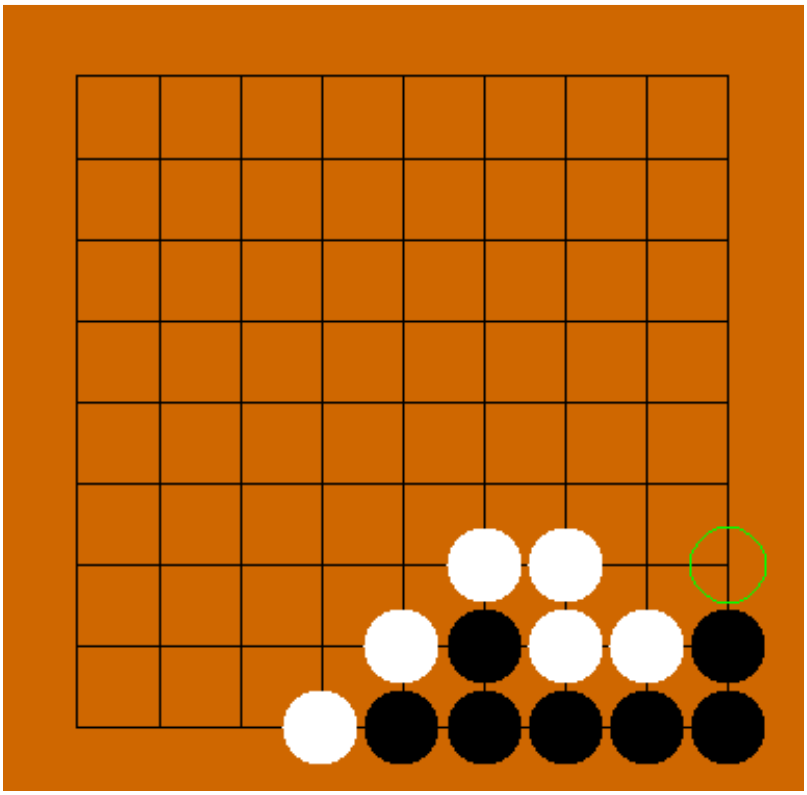


Obrázek 2 – Svobody skupiny černých kamenů

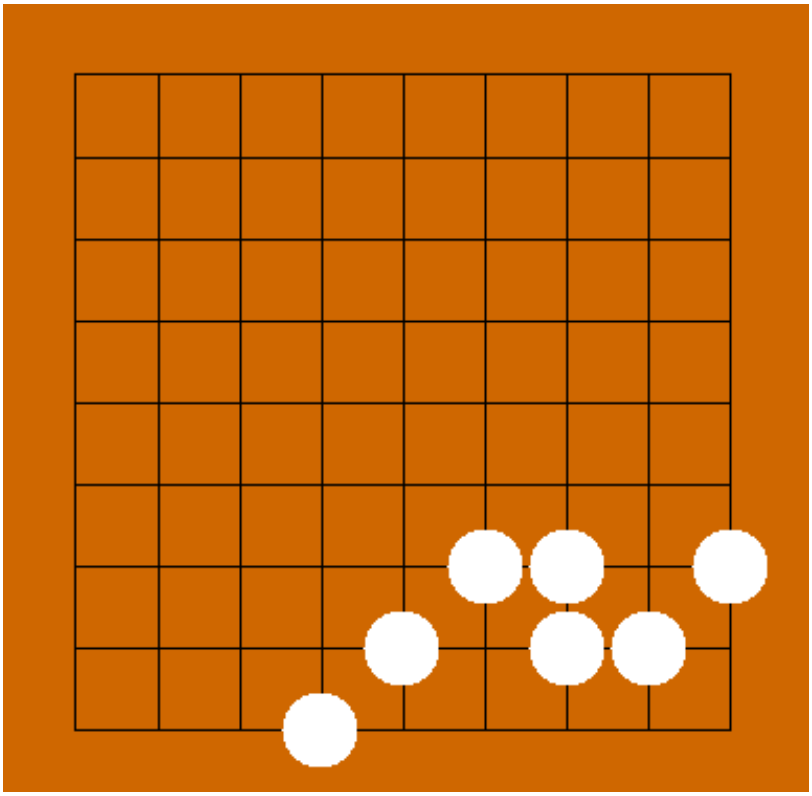
Ve hře Go je používán pojem „oko“. „Oko“ je svoboda nebo skupina svobod, které jsou ohraničené kameny jedné barvy. Je to například skupina černých kamenů viz Obrázek 5 a Obrázek 6.

Zajímání kamenů

Skupina kamenů je zajata, pokud hráč tahem zabere poslední svobodu protihráčově skupině kamenů. V takovém případě je zajatá skupina odebrána z hrací desky a hráči, který ji zajal, je přičten bod za každý kámen v zajaté skupině. Například bílý hráč na obrázcích (Obrázek 3 a Obrázek 4) zajme skupinu 7 kamenů černého hráče. Bílému hráči je tedy přičteno 7 bodů.

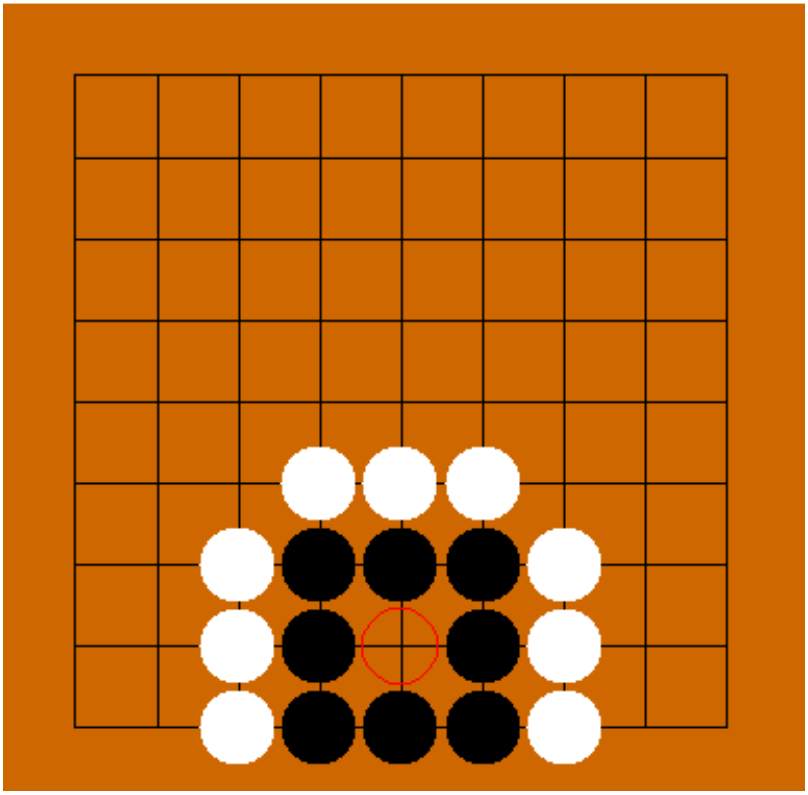


Obrázek 3 – Kameny před zajetím



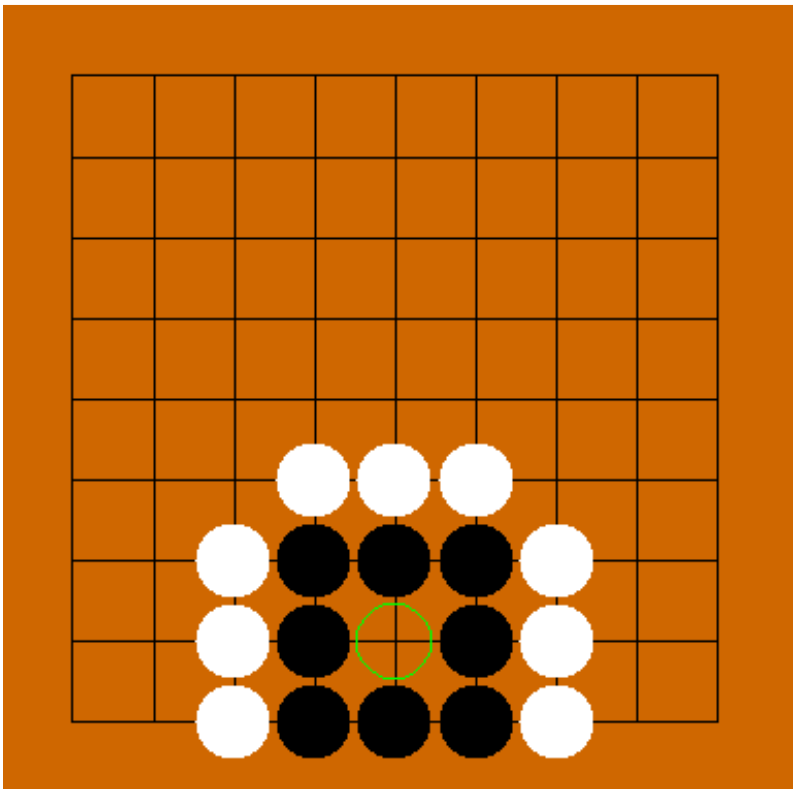
Obrázek 4 – Kameny po zajetí

Nelze spáchat „sebevraždu“, tj. zahrát tah tak, aby vlastní skupina ztratila poslední svobodu. Tomu se také říká pravidlo „kamikadze“. V takové situací musí být kámen zahrán na jiné místo na desce, nebo se musí hráč tahu vzdát.

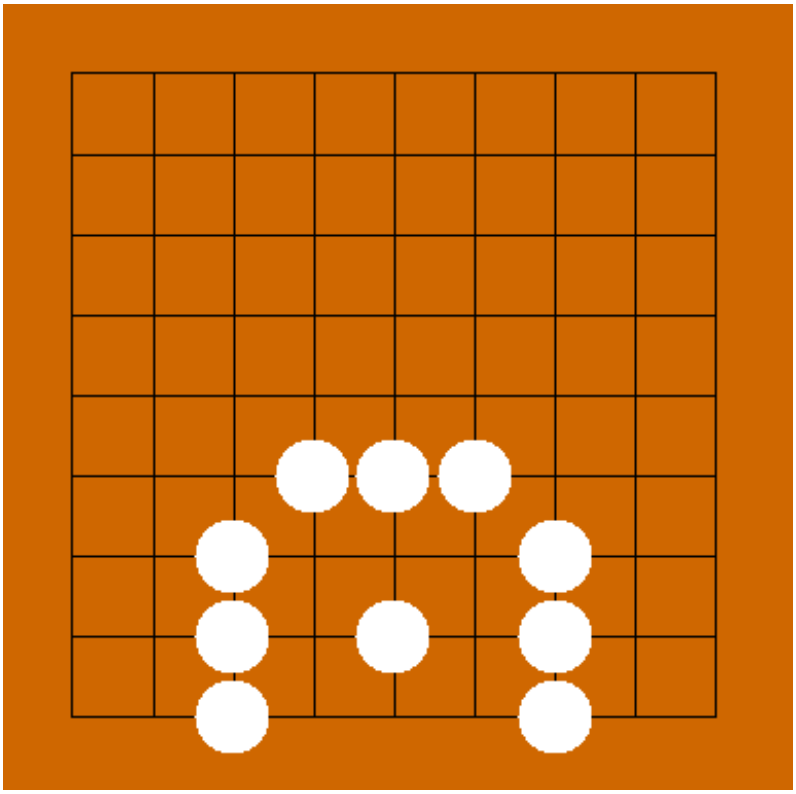


Obrázek 5 – Kamikadze

Výjimkou je případ, kdy je tímto tahem zajata protihráčova skupina kamenů tak, že po jejím odstranění zbude hráčově skupině alespoň jedna svoboda. V takovém případě jde o zajetí a nikoliv o „sebevraždu“. Pokud bílý hráč položí kámen do „oka“ černé skupiny (viz Obrázek 6),může to na první pohled vypadat jako „sebevražda“, protože bílý kámen nebude mít žádnou svobodu. Ale v tomto případě bude černá skupina zajata a bílému kameni zbydou 4 svobody, viz Obrázek 7. (Soo-hyun, 1994)



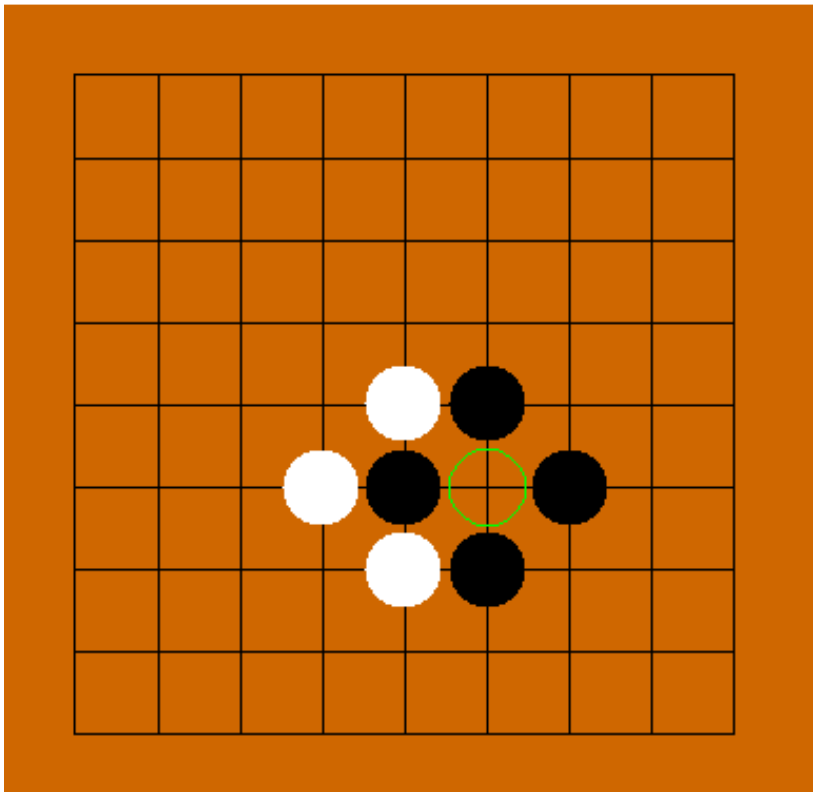
Obrázek 6 – Zajetí oka



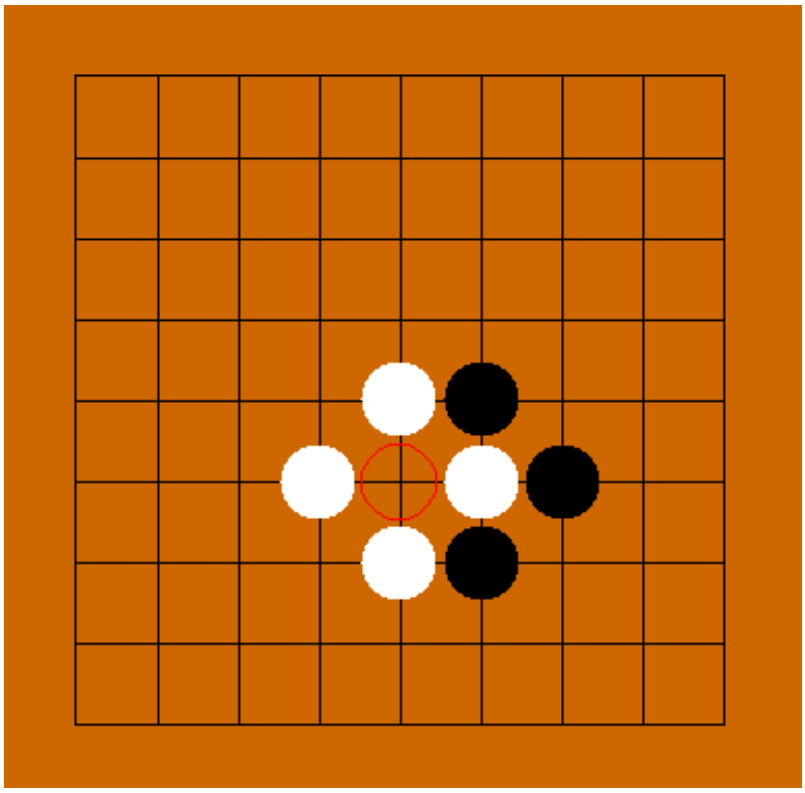
Obrázek 7 – Zajaté oko

Pravidlo o opakování

Existují situace, za kterých lze provést sérii dvou tahů tak, aby se stav na desce nezměnil. Jako příklad se uvádí situace na následujícím obrázku (Obrázek 8).

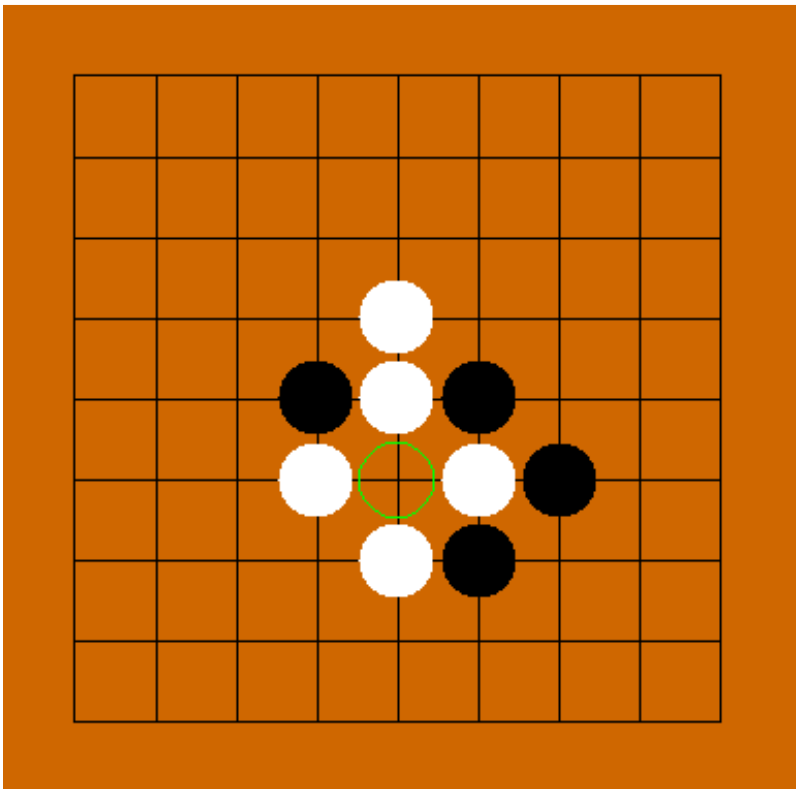


Obrázek 8 – Situace „Ko“



Obrázek 9 – „Ko“ – zakázaný tah

Proto existuje pravidlo, že tahem nesmí nastat stav hrací desky stejný, jako byl stav předcházející, toto pravidlo se jmenuje „Ko“. Pravidlo „Ko“ donutí hráče provést alespoň jeden tah na jiné místo, nebo se tahu vzdát. Tímto pravidlem je zajištěno, že se již nebude opakovat zcela stejná situace. Této situaci se říká „bitva o Ko“.

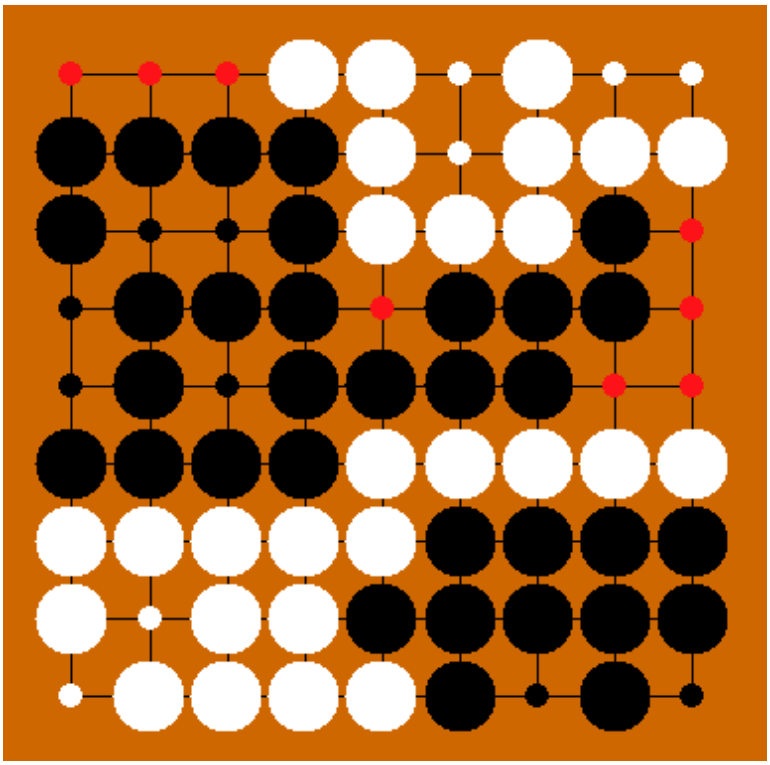


Obrázek 10 – Bitva o „Ko“

Ukončení hry a hodnocení

Hra končí ve chvíli, kdy se oba hráči po sobě vzdají tahu.

Výsledný počet bodů je součtem bodů, které hráč získal zajetím soupeřových kamenů během hry, bodů zajatých území a půlbodu, který se přičítá bílému hráči jako kompenzace za to, že černý hráč má první tah. Tomuto půlbodu se říká „Komi“ a zároveň zabraňuje remíze. Zajaté území je skupina horizontálně či vertikálně sousedících volných míst, ohraničených kameny jedné barvy. Volná místa ohraničená kameny obou barev se považují za neutrální území a hodnocení nijak neovlivňují. Zajaté oblasti jsou vyznačeny na obrázku (Obrázek 11), bílé body jsou území bílého hráče, černé body jsou oblasti černého hráče a červené body jsou neutrální oblasti. (Shotwell, 2003)



Obrázek 11 – Zajatá území

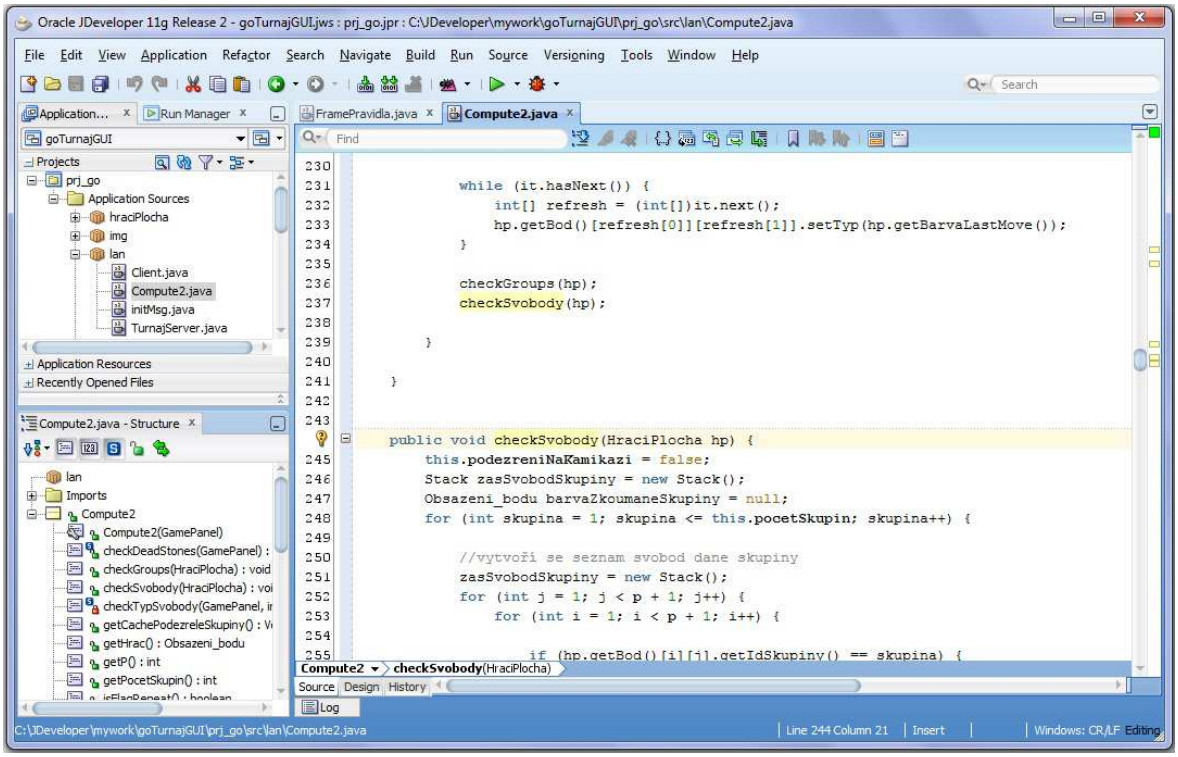
Předpokládejme, že během hry získal černý hráč 17 bodů a bílý hráč 20 bodů za zajaté kameny. Černý hráč získá za zajatá území 7 bodů a bílý hráč 6 bodů. Bílému hráči náleží navíc půlbod „Komi“. Černý hráč má tedy 24 bodů a bílý hráč vyhrává s 26,5 body.

2 Použité technologie

Pro naprogramování aplikace Turnaj v Go bylo využito IDE JDeveloper. Jsou použity technologie pro síťovou komunikaci (sokety, serverové sokety a další), dále technologie pro více-vláknový běh aplikace a pro implementaci GUI byly použity komponenty z knihovny Swing.

2.1 IDE JDeveloper 11.1.2.3.0

Pro tvorbu aplikace bylo použito vývojové prostředí JDeveloper verze 11.1.2.3.0. JDeveloper je Integrované vývojové prostředí (IDE) od společnosti Oracle. Umožňuje návrh, kódování, ladění a nasazení Javových aplikací.



Obrázek 12 – JDeveloper IDE

2.2 Třídy pro síťovou komunikaci

Použité třídy jsou z JDK balíku `java.net`, což jsou standardní nástroje pro síťovou komunikaci. Třídy `Socket` a `ServerSocket` jsou použity k vytvoření TCP spojení mezi serverem a klienty.Toto spojení je určené pro vlastní hraní turnaje. `DatgramSocket` a `MulticastSocket` je použit herním lobby pro rozesílání a přijímání informací o založených turnajích.

2.2.1 Socket

Socket je třída implementující klientské sokety (také zvané jen „sokety“). Soket je koncový bod pro komunikaci mezi dvěma zařízeními. Vlastní úloha Soketu je vykonávána instancí třídy implementující rozhraní `SocketImpl`, která je vytvořena a nakonfigurována

například s ohledem na lokální firewall. Socket obsahuje mimo jiné atributy především IP adresu serveru a číslo portu serveru, na něž se má připojit.(Elliotte, 2005)

Práce se soketem

Vytvoření soketu s adresou a portem serveru lze docílit pomocí konstruktoru:

```
Socket s = new Socket(serverIP, port);
```

Při takovémto vytvoření je soket nakonfigurován s výchozím nastavením systému, je mu určena cílová IP adresa serveru a cílový port serveru. Po vytvoření se soket pokusí připojit na server. Volání metod pro vytváření soketu respektive pro připojení na server jsou umístěna v bloku *try-catch*, jelikož může vyvolat výjimku `IOException`. Pokud se připojení zdaří, lze pomocí vstupních a výstupních proudů (`InputStream` respektive `OutputStream`) získaných metodami `s.getInputStream()` respektive `s.getOutputStream()` zapisovat a číst ze soketu.

Po ukončení spojení je vhodné soket uzavřít, abybyly uvolněny systémové prostředky (v tomto případě port) pomocí `s.close()`.

2.2.2 ServerSocket

`ServerSocket` je implementován na serveru, čeká na žádosti o spojení od klientů, na jejichž základě vrací instanci třídy `Socket`, která je určena pro komunikaci s daným klientem. Na rozdíl od soketu serverový soket po přijetí žádosti a vytvoření spojení nadále naslouchá, dokud není ukončen, a může přijímat další žádosti o spojení.(Elliotte, 2005)

Práce se ServerSocketem

Pro správnou funkci serverového soketu mu musí být přiřazen port, na němž bude naslouchat. Uvedu příklad vytvoření objektu typu `ServerSocket`:

```
ServerSocketss = new ServerSocket();
ss.bind(new InetSocketAddress(port));
```

Tím je vytvořen objekt typu `ServerSocket`. Příkazem `ss.accept()` bude čekat na příchozí spojení. Jakmile obdrží požadavek, naváže spojení a vrátí objekt typu Socket, který poskytuje proudy pro čtení a zápis, jež jsou vázané na toto spojení.

```
Socket s = ss.accept();
```

Dále se komunikace odvíjí pomocí soketu. Pokud nebude zájem navazovat další spojení, je možné `ServerSocket` uzavřít příkazem `ss.close()`.

Obdobně jako u třídy `Socket` některé metody třídy `ServerSocket` mohou vyvolat výjimku typu `IOException`, proto je nutné zachytávat je v bloku *try-catch*.

2.2.3 DatagramSocket

`DatagramSocket` je koncový komunikační bod, který rozesílá nebo přijímá jednotlivé pakety (respektive UDP Datagramy) bez toho, aby navazoval spojení.`DatagramSocket` používám pro rozesílání informací o založené hře na vícesměrovou (multicast) IP adresu *224.0.0.1*.(Calvert, 2008)

Práce s DatagramSocketem

Vytvoření DatagramSocketu:

```
DatgramSocketds = new DatagramSocket();
```

Na rozdíl od třídy `Socket`, `DatagramSocket` nepoužívá spojení, takže není daný žádný proud, do něhož by bylo možné zapisovat. Každý `DatagramPacket,` který je třeba odeslat, musí mít určenou cílovou IP adresu, cílový port, data a délku dat. `DatagramSocket` jej pak pouze odešle.

Abychom mohli poslat objekt, musíme jej převést do pole bytů a toto pole předat DatagramPacketu.

```
DatagramPacketdp = new
DatagramPacket(data,dataLenght,multicastAddress,port);
ds.send(dp);
```

2.2.4 MulticastSocket

`MulticastSocket` je potomkem třídy `DatagramSocket` a přidává možnost připojit se do vícesměrové (multicast) skupiny. Každý, kdo je připojen do skupiny, například `224.0.0.1,` obdrží pakety, které budou na tuto adresu odeslány. (Calvert, 2008)

Práce s MulticastSocketem

Vytvořenému objektu typu `MulticastSocket` je přiřazen port, poté je připojen do žádané skupiny. Nyní může přijímat zprávy pro tuto skupinu. Odesílat zprávy do skupiny lze i pokud `DatagramSocket` nebo `MulticastSocket` nejsou členem skupiny. Práce s objektem typu `MulticastSocket` musí být opět v bloku *try-catch*.

```
MulticastSocket ms=new MulticastSocket(54775);
ms.joinGroup(InetAddress.getByName("224.0.0.1"));
```

2.3 Použití více vláken

Vlákno je součást procesu a umožňuje běh programu. Více vláken umožňuje, aby v procesu běžely různé nebo stejné části programu paralelně. Paralelní běh vláken je v reálném čase paralelní pouze u více-jádrových procesorů. Běží-li program na jednom jádře, jde o běh pseudo-paralelní, při kterém je mezi vlákny střídavě přepínán kontext.

Práce s vlákny

Vlákno lze vytvořit pomocí třídy, která je potomkem třídy `Thread`, a přetěžuje metodu `run()`. Dále lze vytvořit třídu, jež implementuje rozhraní `Runnable`, které předepisuje metodu `run()`. Objekt tohoto typu se předá nově vytvářenému objektu typu `Thread` jako parametr konstruktoru.

V obou případech je v těle metody `run()` obsažen kód, jenž bude proveden při spuštění vlákna. Vlákno je spuštěno příkazem `start()` a skončí při návratu z metody `run()`, nebo vyvoláním výjimky, která není v těle metody `run()` zachycena. (Bloch, 2001)

V aplikaci používám, jak ve vlákně klienta, tak ve vlákně serveru, cyklickou kontrolu příznaku `kill`, který, pokud je nastaven na *true*, ukončí činnost vlákna návratem z metody `run`. Tato podmínka se kontroluje vždy zachycením výjimky `SocketTimeoutException`.

Příklad třídy, jež lze spustit jako vlákno:

```
public class LobbyClientSimple implements Runnable {
      ...
      public voidrun() {
try {
ms=new MulticastSocket(54775);
            ...
            }
      }
}
```

Příklad spuštění vlákna:

```
LobbyClientSimpleclient;
Threadt_lobbyCS = newThread(client);
...
t_lobbyCS.start();
```

Pakliže z jednoho vlákna spustíme jiné vlákno, pak ve výchozím nastavení nemůže rodičovské vlákno skončit dříve než jeho potomek. Proto lze vlákno označit jako tzv. „démona“. „Démon“ je vlákno, které je ukončeno, pokud skončí vlákno rodičovské. Celý kód pro spuštění vlákna jako „démona“ vypadá takto: (Lewis, a další, 2000)

```
LobbyClientSimple client;
Thread t_lobbyCS = new Thread(client);
t_lobbyCS.setDaemon(true);
t_lobbyCS.start();
```

3 Implementace pravidel ve hře Go

Pravidla jsou implementována na straně klienta. Při obdržení hrací desky od serveru se simuluje tah na každé volné místo na hrací desce,přičemž jsou aplikovány algoritmy implementující pravidla, která rozhodnou, je-li tah na právě testované pozici pro daného hráče validní či nikoliv. Dle toho se bodu na testované pozici nastaví příznak validity. Aby bylo jasné, jak algoritmy pracují, je vhodné blíže představit některé třídy.

Hrací deska

Hrací deska je v aplikaci realizovaná jako třída `HraciPlocha`. Skládá se z dvourozměrného pole objektů typu `Bod`, které reprezentuje rozmístění kamenů na hrací desce, a několika důležitých atributů nesoucích informace o stavu hry.

```
public class HraciPlocha implements Serializable{
private Bod[][] bod;
privateintsize;
privateint[] lastMove = null;
privateObsazeni_bodubarvaLastMove;
privatebooleanwhitePass, blackPass, gameEnded;
private double whiteScore = 0;
private double blackScore = 0;
```

`HraciPlocha` se ve třídě `GamePanel` vyskytuje ve dvou instancích. První instance označená proměnnou `hp` je aktuální stav hrací plochy, druhá označená proměnnou `hpm` je předešlý stav hrací plochy viz kapitola 3.2 Pravidlo o opakování („Ko“).

Dvourozměrné pole `bod` má rozměry o 2 větší, než je velikost hrací desky, protože hratelná plocha hrací desky je ohraničena body speciálního typu.

Třída `Bod` zapouzdřuje informace o typu bodu pomocí výčtu `Obsazeni_bodu`.Pokud je prázdný (`Obsazeni_bodu.e`), je na něm bílý kámen (`Obsazeni_bodu.w`), černý kámen (`Obsazeni_bodu.b`), nebo je hranicí hrací desky (`Obsazeni_bodu.x`). Dále obsahuje pomocné atributy, které jsou využívány algoritmy pravidel.

```
public class Bod implements Serializable{
privateObsazeni_bodu typ;
privateintidSkupiny;
privateinttypBoduOka;
privateintstoneAvailible;
...
```

Třída `Compute2` obsahuje všechny algoritmy pravidel a metody na práci s hrací plochou.

3.1 Určení validního tahu

Jestliže je hráč na tahu a přejíždí kurzorem myši nad hrací plochou, zobrazuje se mu červený kruh, pokud není tah na tuto pozici povolen. Zelený kruh se hráči zobrazí v případě, kdy je na danou pozici tah možný. Tato informace je uložena v každém bodě zvlášť jako celočíselná hodnota, kdy *1* znamená validní tah a *0* nevalidní tah. Tato hodnota

je určena po obdržení hrací plochy od severu zavoláním metody `preCount(Obsazeni_bodu)`,a to s barvou hráče jako parametr. V metodě `preCount(Obsazeni_bodu)`je v cyklu pro každý bod na hrací desce volána metoda `stoneAvailible(int, int, Obsazeni_bodu)`. Předá jí pozici potenciálního tahu a barvu hráče jako vstupní parametry.

Metoda `stoneAvailible(int, int, Obsazeni_bodu)`vrací jedničku v případě validního tahu a nulu nebo dvojku v případě nevalidního tahu (*2*vpřípadě, že došlo k porušení pravidel, *0*v případě nevalidního tahu z jiného důvodu, například pozice je již obsazena jiným kamenem). Nejdříve se kontroluje, zdali je bod typu „prázdný“ (`Obsazeni_bodu.e`), dále je vytvořena kopie hrací plochy, na které je provedena simulace tahu, následně se ověřuje pravidlo o „kamikadze“, a pokud se „kamikadze“ nepotvrdí, porovná se s minulým stavem hrací plochy, jestli se nejedná o opakování. Obstojí-li simulace ve všech těchto pravidlech, je bod označen jako validní. Validní bod má hodnotu `stoneAvailible` nastavenu na *true*.

```
public int stoneAvailible(int i, int j, Obsazeni_boduhrac) {
  //return: 0 - neplatny tah, 1 - validni tah, 2 - poruseni pravidel
  // zjistuje je-li mozny tah na oznacovany bod
switch (mf.getHp().getTyp(i, j)) {
        case e:
     //pouze pokud je mistoprazdne
     // simulace se provadi na oddelenehraciplose
HraciPlocha sim = newHraciPlocha(p);
            // kopie hlavni hraci plochy
for (int k = 1; k < p + 1; k++) {
for (int m = 1; m < p + 1; m++) {
sim.getBod()[m][k].setIdSkupiny(mf.getHp().getBod()[m][k].getIdSkupiny())
;
sim.getBod()[m][k].setTyp(mf.getHp().getBod()[m][k].getTyp());
                }
            }
            //simulace polozeni kamene
sim.setTyp(i, j, hrac);
            //zaznam tohoto tahu
sim.setLastMove(newint[] { i, j }, hrac);
if (simul(sim)) {
// paklize tah projde testem na kamikazi je daletestovan aby se
//neopakovala situace na hraci ploše (pravidlo o opakovani nebo 'Ko') -
//porovnava se stav hraci plochy se stavem minulym.
//simulace predpoklada ze k opakovani dojde, prvni odchylka toto vyvrati.
flagRepeat = true;
for (int k = 1; k < p + 1; k++) {
for (int m = 1; m < p + 1; m++) {
if (sim.getBod()[k][m].getTyp() !=
this.mf.getHpm().getBod()[k][m].getTyp()) {
      // staci jeden rozdil ve stavu hraci plochy aby nedoslo k opakovani
flagRepeat = false;
                        }
                    }
                }
if (flagRepeat) {
```

```
                return 2;
            } else {
                return 1;
            }
        } else {
            return 2;
        }

    case w:
    case b:
        return 2;
    case x:
    default:
        return 0;
    }
}
```

3.2 Algoritmus zajímání kamenů

Pakliže je proveden tah, zavolá se z třídy `Compute2` metoda `reCount(HraciPlocha)`, která zaktualizuje stav hrací desky a počty bodů obou hráčů. (Pozor, neplést s metodou `preCount(HraciPlocha)`, jež určuje validitu tahu.)

```
public void reCount(HraciPlochahp) {
        //očíslování skupin
checkGroups(hp);
        //zjištění svobod a odstranění kamenů
checkSvobody(hp);
if (podezreniNaKamikazi) {
//jde o realizaci validního tahu, tudíž nemůže jít o kamikadze, pouze se
//obnoví kameny zajímající skupiny
Iteratorit = cachePodezreleSkupiny.iterator();
while (it.hasNext()) {
int[] refresh = (int[])it.next();
hp.getBod()[refresh[0]][refresh[1]].setTyp(hp.getBarvaLastMove());
            }
        }
    }
```

Metoda `checkGroups(HraciPlocha)`označí skupiny kamenů tak, že nastaví číslo skupiny všech bodů na -1 (tzn. „neočíslováno“) a postupně testuje body hrací desky, zdali jsou, či nejsou očíslované. Je-li testovaný bod neočíslovaný, vloží jej do zásobníku. Následně v cyklu odebírá body ze zásobníku, přičemž nastaví číslo skupiny na kladnou hodnotu `pocetSkupin`, jež je na začátku průchodu nastavena na 1 a vloží do něj sousedící body stejného typu. Jakmile je zásobník prázdný, je celá skupina očíslována, inkrementuje se hodnota `pocetSkupin` a testuje se následující bod hrací desky.

Dále metoda `checkSvobody(HraciPlocha)`podobným způsobem vybere volné body bezprostředně sousedící s kterýmkoliv kamenem dané skupiny. Může nastat situace, kdy jedna svoboda sousedí s více kameny téže skupiny a bude do zásobníku přidána vícekrát, testována je ale pouze skutečnost, jestli má skupina alespoň jednu svobodu.

Jestliže skupina nemá žádnou svobodu, je určena k odebrání a hráči opačné barvy je připočten příslušný počet bodů. Pokud má tato skupina stejnou barvu jako poslední tah, je nastaven příznak `podezreniNaKamikazis` na *true*. Nepotvrdí-li se podezření, je původně odebraná skupina vrácena na hrací desku, jedná se o zajetí „oka“.

Tento algoritmus je použit ve dvou případech, buď při aktualizaci stavu plochy po tahu hráčem, nebo jako pravidlo pro určení validity tahu.(viz kapitola 3.4 Detekce pravidla „kamikadze“)

3.3 Pravidlo o opakování („Ko“)

Implementace pravidla o opakování je velice jednoduchá, ve třídě `GamePanel` je uchovávána druhá instance hrací plochy, která zaznamenává stav desky před posledním tahem. Při simulaci tahů po obdržení hrací desky se porovnává výsledný stav se stavem minulým. Pokud se shodují, jde o opakování a tah není validní.

3.4 Detekce pravidla „kamikadze“

Způsobí-li náš poslední tah situaci, že skupina kamenů, k níž patří kámen tohoto tahu, ztratí poslední svobodu, může se jednat o jeden ze dvou případů: buď se jedná o „kamikadze“ a ta není povolena, anebo jde o zajetí oka a to je povolený tah.

Detekce kamikadze probíhá ve dvou fázích. V první fázi se zavolají metody `checkGroups(HraciPlocha)` a `checkSvobody(HraciPlocha)` a jsou odstraněny všechny skupiny, které nemají žádnou svobodu. Pokud skupina související s posledním tahem nemá žádnou svobodu, je také odstraněna, ale pozice kamenů se uloží do seznamu a nastaví se příznak `podezreniNaKamikazi` na *true*. Pokud je nastaven příznak `podezreniNaKamikazi`, tak se pouze skupina podezřelá z kamikadze vrátí na desku a znovu se aplikují metody `checkGroups(HraciPlocha)` a `checkSvobody(HraciPlocha)`. Jestliže je určená k vyřazení i nadále, tj. `podezreniNaKamikazije` opět nastaveno, pak opravdu jde o „kamikadze“. Rovněž při odebrání všech zajatých kamenů bude tato skupina opět odebrána. Pokud ovšem `podezreniNaKamikazi` nastaveno není, znamená to, že zajetím jiné skupiny získala tato skupina alespoň jednu svobodu a jedná se o zajetí „oka“. Takový tah je označen jako validní.

3.5 Ukončení hry

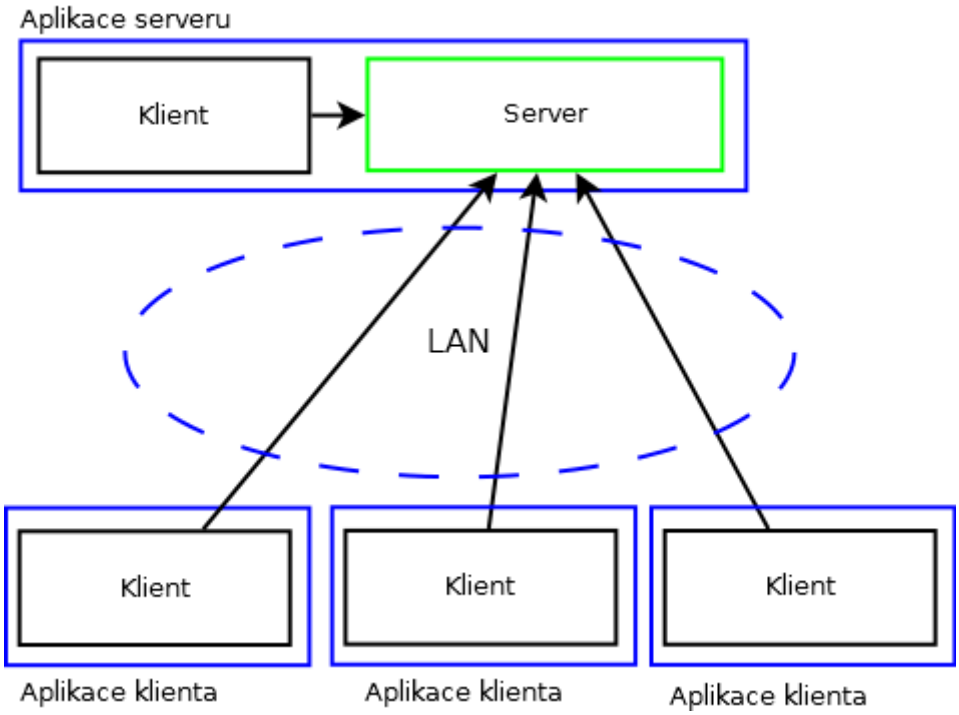
Hra končí, pokud se oba hráči po sobě vzdají tahu. Informace o vzdání se tahu je uložena v hrací ploše pod booleovskými proměnnými `whitePass` a `blackPass`. Při druhém vzdání se zavolá metoda `checkTerritory(HraciPlocha)`, která spočítá body za zajatá území. Stejně jako se číslují skupiny kamenů u hráčů,tak se očíslují skupiny svobod. Poté se prochází okolí těchto svobod a testuje se, zda sousedí s kameny bílé, černé, nebo obou barev. Hráči je přičten příslušný počet bodů,pokud sousedí s kameny jedné barvy.

Na konec se porovnají body obou hráčů a je vybrán vítěz. Klient pak pošle serveru příslušnou zprávu o konci hry.

4 Implementace síťové komunikace

4.1 Princip

Turnaj je založen na architektuře klient-server. Server organizuje utkání pomocí přesměrovávání komunikace mezi klienty. Aplikace, jež zakládá turnaj, vytvoří server, který běží jako samostatné vlákno, na tento server se aplikace sama připojí jako klient.



Obrázek 13 – Architektura

4.2 Klient

Klient se připojuje na server pomocí třídy `Client`, která běží jako samostatné vlákno a čeká na zprávy od serveru, nebo posílá zprávy na server. Po připojení na server klient pošle serveru jméno hráče v podobě objektu typu `String`.

```
s = newSocket(this.serverIP, this.port);
oos = new ObjectOutputStream(new
BufferedOutputStream(s.getOutputStream()));
oos.flush();
s.setSoTimeout(200);
ois = new ObjectInputStream(newBufferedInputStream(s.getInputStream()));
oos.writeObject(name);
oos.flush();
```

Dále čeká, než mu server oznámí začátek turnaje, mezitím přijímá pouze zprávy o počtu připojených hráčů, které jsou složeny z pole dvou celočíselných hodnot, první je počet připojených hráčů a druhá očekávaný počet hráčů. Pokud je první hodnota záporná, je to signál, že jsou připojeni všichni hráči. V tu chvíli je možné začít turnaj.

```
while (waitingForPlayers) {
...
message = ois.readObject();
...
if (message instanceof Integer[]) {
Integer[] it = (Integer[])message;
if (it[0] < 0)
waitingForPlayers = false;
if (it[0] > 0) {
gamePanel.setPocetPripojenych(it[0], it[1]);
} }}
```

Dále klient čeká na příchozí data od serveru. Příchozí data jsou postupně testována pomocí *instanceof* na datový typ, a v případě typu `String` na text, který obsahují.

Informace o začátku utkání jsou obsaženy v objektu typu `InitMsg`, to je třída, jež zapouzdřuje informaci o tom, jak velká je hrací plocha a jaká barva je hráči přiřazena. Další typy zpráv mohou být: aktualizace turnajového diagramu (turnajového „pavouka“) přijetím objektu typu `TurnajDiagram`, zpráva o změně vyřazení ze hry přijetím objektu typu `String` obsahující jeden z textů „*out*“, „*in*“, nebo „*inOdp*“. (viz kapitola 5.1 Výběr protihráčů) Dále přijímá objekt typu `HracíPlocha`, jež reprezentuje stav hrací desky po tahu protihráče, nebo oznámení o konci turnaje v podobě objektu typu `String` s textem „*WIN*“ v případě výhry, nebo „*end*“ v případě prohry.

Operace se sokety jsou umístěny v bloku *try-catch*, kde jsou zachytávány výjimky typu `SocketTimeoutException`, kdy následuje v těle *catch* příkaz *continue* a pokračuje se dalším cyklem.Dále výjimky `EOFException`, tu vyvolá metoda `Socket.readObject()`při pokusu o čtení objektu ze soketu, který je nedostupný, a `SocketException` se zprávou „*Connection reset*“ při přerušení spojení.

Vlákno klienta běží jako „démon“, tj. je ukončeno s ukončením aplikace, nebo pokud se hráč rozhodne začít novou hru při již rozehrané hře, je nastaven příznak `kill`, a klient bude ukončen návratem z metody `run`.

4.3 Server

Turnajový server je v samostatném vlákně běžící třída `TurnajServer`, která čeká na připojení dostatečného počtu klientů.Na každý nový socket nastaví server socket timeout, což je doba, po kterou bude operace čtení z proudu soketu blokovací. Pokud do této doby nebudou k dispozici příchozí data pro čtení, bude vyvolána `SocketTimeoutException`. Doba blokování je určena následovně, kde `hosts` je celkový počet hráčů.

```
s[hostsAccepted].setSoTimeout((Integer)(320 / hosts));
```

Konstanta 320 je určena tak, že při dvou hráčích bude doba blokování na jeden socket160 ms, a při 32 hráčích 10 ms. V obou případech bude každý socket čekat maximálně 320 ms, než bude opět obsloužen. Důsledkem je menší zátěž serveru při

menším počtu hráčů a z pohledu klienta konstantní doba odezvy serveru nezávislá na počtu hráčů.

Po každém připojení nového klienta server čeká, dokud mu klient nepošle svoje jméno hráče, následně jej zařadí do turnajové dvojice a rozešle aktualizovaný počet připojených hráčů všem dosud připojeným klientům.Jakmile je připojen dostatečný počet klientů, rozešle všem klientům objekt `InitMsg`, který obsahuje informace o barvě hráče a velikosti hrací desky, čímž dojde u klientů k zahájení vlastního utkání.

Server nadále zpracovává příchozí data od klientů, jde-li o objekt typu `HraciPlocha`, pouze je přepošle klientu protihráče. Server dále testuje příchozí objekty typu `String`, jestli nejsou ukončeny zprávou o konci utkání a to „*vytez*“ v případě výhry, nebo „*end*“ v případě prohry. Server také rozesílá upozornění klientu, pokud se protihráč odpojí, podobně jako na straně klienta zachytává výjimky `EOFException` a `SocketException` se zprávou „`Connection reset`“pro ošetření ukončení spojení.Po ukončení turnajového kola, tj. když všechny dvojice ohlásí konec hry, na základě výsledků turnajový server vytvoří nové dvojice hráčů a rozesláním `InitMsg` zahájí další kolo.

Vlákno serveru též běží jako „démon“. Rozhodne-li se hráč zahájit novou hru při rozehraném turnaji a potvrdí ukončení turnaje, je v objektu typu TurnajServer nastaven příznak `kill`, a server bude ukončen návratem z metody `run`.

5 Implementace turnaje

Turnaj je sestavován tak, že z připojených klientů jsou vytvořeny dvojice, které spolu hrají.Dohrají-li všechny dvojice, jsou z vítězů sestaveny dvojice pro následující kolo turnaje. Pokud postupuje pouze jediný hráč, je turnaj u konce a tento hráč je vítězem turnaje.

Organizace turnaje je součástí třídy `TurnajServer`. Turnajový server se nezabývá pravidly samotného zápasu, pouze určuje kdo s kým bude hrát a rozesílá informace týkající se stavu turnaje, například turnajového „pavouka“.

5.1 Výběr protihráčů

Protihráči jsou vybráni v pořadí, v jakém se připojili k serveru. To znamená, že bude hrát první s druhým, třetí se čtvrtým apod. Po ukončení kola turnaje prochází vítěze opět v pořadí, v jakém se připojili, a vytváří dvojice pro následující kolo.

Při připojení je každý klient identifikován indexem, který označuje pořadí, v jakém se připojil, počínaje od nuly v případě prvního klienta. Třída `TurnajServer` má několik polí atributů, kdy index pole identifikuje konkrétního klienta.

```
public classTurnajServerimplementsRunnable {
    ...
private int[] binder;
private Socket[] s;
private boolean[] hostActive;
private boolean[] isPlaying;
private boolean[] dalsiKolo;
private String[] names;
private ObjectInputStream ois[];
private ObjectOutputStream oos[];
    ...
```

Pole `binder` obsahuje index protihráče, pole `hostActive` údaj, jestli je klient připojen, `isPlaying` zda stále hraje zápas, `dalsiKolo` určuje, jestli klient postupuje do dalšího kola a `names` je seznam jmen hráčů. Inicializace vypadá následovně:

```
while (hostsAccepted<hosts) {
      ...
try{
        s[hostsAccepted] = ss.accept(); // cekani na klienta!
      } catch (SocketTimeoutException e) {
continue;
      }
oos[hostsAccepted] = new
ObjectOutputStream(newBufferedOutputStream(s[hostsAccepted].getOutputStre
am()));
oos[hostsAccepted].flush();
ois[hostsAccepted] = new
ObjectInputStream(newBufferedInputStream(s[hostsAccepted].getInputStream(
)));
Object o;
try {
```

```
         o = ois[hostsAccepted].readObject();
if (o instanceofString) {
names[hostsAccepted] = o.toString();
        }
      } catch (ClassNotFoundException e) {
      }
      s[hostsAccepted].setSoTimeout((Integer)(320 / hosts));
for (int i = 0; i <= hostsAccepted; i++) {
oos[i].writeObject(newInteger[] { hostsAccepted + 1, hosts });
oos[i].flush();
               }
      //přiřazení protihráčů
if (hostsAccepted % 2 == 0) {
binder[hostsAccepted] = hostsAccepted + 1;
      } else {
binder[hostsAccepted] = hostsAccepted - 1;
      }
dalsiKolo[hostsAccepted] = false;
isPlaying[hostsAccepted] = true;
hostActive[hostsAccepted] = true;
hostsAccepted++;
}
```

Po inicializaci se zahájí první kolo turnaje rozesláním objektu typu `InitMsg`. Od této chvíle server přeposílá hrací plochy mezi hráči a čeká na zprávu o konci utkání. Zjednodušený příklad přeposílání hrací plochy:

```
for (int i = 0; i <hosts; i++){
...
message = ois[i].readObject();
...
oos[binder[i]].writeObject(message);
oos[binder[i]].flush();
}
```

Zpráva o konci zápasu je objekt typu `String` obsahující text „*vyhra*“v případě výhry, nebo „*end*“ v případě prohry. Zprávu o konci zasílají oba klienti, turnajový server zprávu „*end*“ ignoruje, pouze ji přepošle druhému klientovi, aby také ukončil hru a spočítal body. Vzhledem k půlbodu, který je přidělen bílému hráči navíc, nemůže nastat remíza, a tak vždy jeden z dvojice hráčů pošle zprávu „*vyhra*“.

Při přijetí zprávy o výhře server odešle správu o ukončení „*end*“ protihráči, výherce zařadí do dalšího kola a u obou hráčů nastaví, že zápas byl dohrán.

```
if
(messageinstanceofString&&message.toString().equalsIgnoreCase("vyhra")) {
//hrac vyhral posilam protihraci oznameni
oos[i].writeObject(newString("end"));
oos[i].flush();
   //hrac co vyhral postupuje
setDalsiKoloStatus(i, true);
setDalsiKoloStatus(binder[i], false);
isPlaying[i] = false;
isPlaying[binder[i]] = false;
continue;
}
```

Pokud jsou všechny zápasy dohrány, je zavolána metoda `reorganizuj`(), která sestaví z pokračujících hráčů nové dvojice a zkontroluje, jestli se nějaký hráč neodpojil.

```
private int reorganizuj() {
boolean first = true;
int bindFirst = -1;
for (int i = 0; i <hosts; i++) {
if (dalsiKolo[i]) {
if (first) { //first
bindFirst = i;
first = false;
                } else {
                    //druhy
binder[bindFirst] = i;
binder[i] = bindFirst;
first = true;
                    //ověř pripojeni hracu

if (!hostActive[bindFirst]) {
disconHandleX(bindFirst);
                    }
if (!hostActive[i]) {
disconHandleX(i);
                    }

if (hostActive[bindFirst] &&hostActive[i]) {
isPlaying[bindFirst] = true;
isPlaying[i] = true;
                    }

                }
            } else {
binder[i] = -1;
            }
dalsiKolo[i] = false;
        }
      ...
```

Po novém vytvoření dvojic nastavením pole `binder`, začne rozesláním `InitMsg` další kolo.Pokud bude mít po reorganizaci proměnná `first` hodnotu `false`, znamená to, že zbyl lichý počet postupujících (nikoli nutně připojených) hráčů. Tato situace nastane pouze tehdy, pokud zbyl jediný vítěz – vítěz turnaje.V takovém případě je rozeslána poslední aktualizace stavu turnaje,objekt typu `String` s textem „*TournamentEnd*“, a turnaj končí.

5.2 Prezentace stavu turnaje

Stav turnaje je prezentován turnajovým diagramem nebo též turnajovým „pavoukem“. Turnajový server si uchovává informace o hierarchii turnaje v objektu třídy `TurnajDiagram`, která zapouzdřuje pole objektů třídy `TurnajDiagramNode`. `TurnajDiagramNode` obsahuje jméno hráče a informaci o jeho připojení. Pole ve třídě `TurnajDiagram` je binární strom implementován na poli. Toto pole je aktualizováno při reorganizaci či při odpojení některého z klientů.

```
private int updateDiagram() {
...
for (int i = 0; i <hostsAccepted; i++) {
if (isPlaying[i]) {
turnajDiagram.getTurnajDiagramNode(offset + x).setJmenoHrace(names[i]);
if (hostActive[i]) {
      turnajDiagram.getTurnajDiagramNode(offset + x).setAktivni(true);
      } else {
      turnajDiagram.getTurnajDiagramNode(offset + x).setAktivni(false);
      }
      x++;
      }
   }
...
```

Když klient obdrží diagram, předá ho objektu `GamePanel`, ze kterého je poté vykreslen na komponentu typu `GrafickýPanelPavouk`. (viz Obrázek 18)

5.3 Ošetření odpojení hráče

Jestliže se během turnaje odpojí některý z klientů, postupuje protihráč automaticky do dalšího kola. V situaci, kdy je zápas u konce, ale zatím nenastalo další kolo, je při odpojení výherce původně poražený hráč určen k postupu jako náhrada.

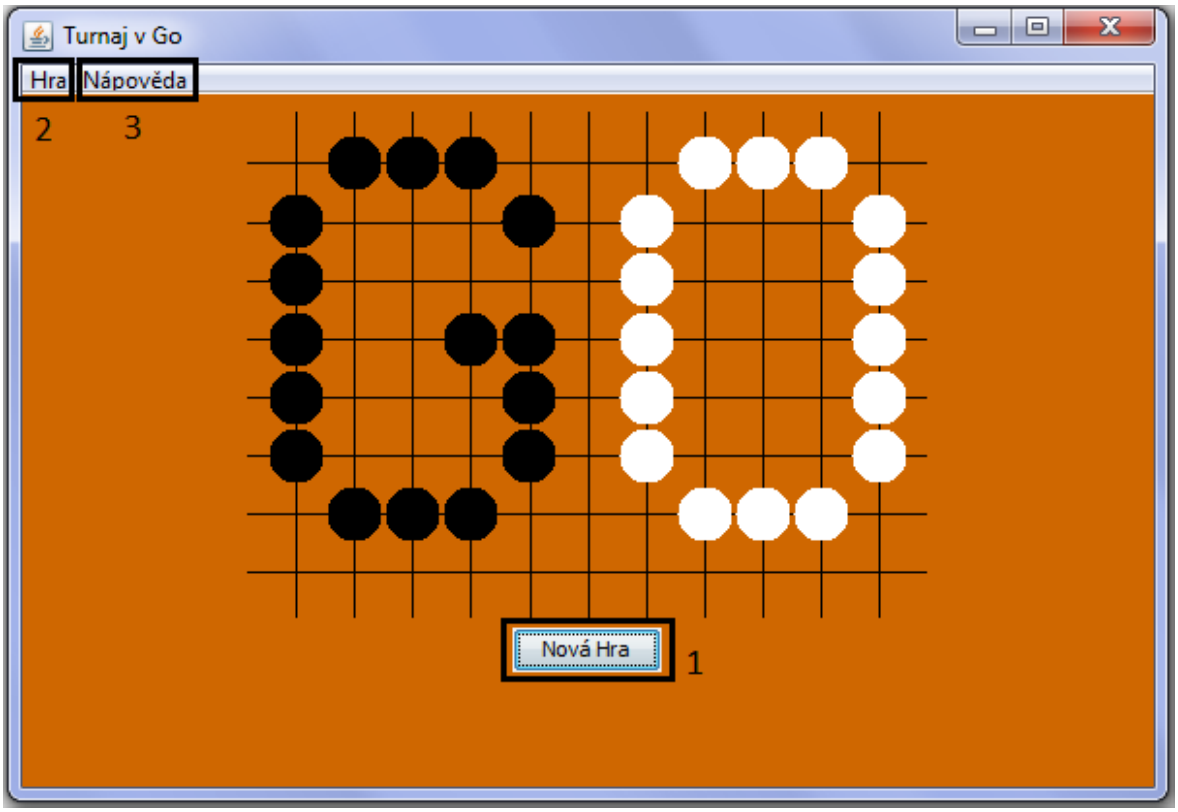
O obstarání odpojených hráčů se stará metoda `disconHandleX(int)`. Ukončí utkání, které měli hráči rozehráno, případně zařadí protihráče zpět do turnaje, pokud v současném kole prohrál. Metoda je volána v případě zachycení výjimky `EOFException`nebo `SocketException` se zprávou „*Connection reset*“.

```
private int disconHandleX(int x) {
if (x < 0) {
            return 1; //neplatnyhrac
        }
hostActive[x] = false;
isPlaying[x] = false;
setDalsiKoloStatus(x, false);
if (binder[x] < 0) {
            return 2; //neplatnyprotihrac
        }
setDalsiKoloStatus(binder[x], true, true);
isPlaying[binder[x]] = false;
        return 0;
    }
```

6 Ovládání hry

Úvodní obrazovka

Po spuštění aplikace Turnaj v Go uživatele přivítá úvodní obrazovka (viz Obrázek 14).



Obrázek 14 – Úvodní obrazovka

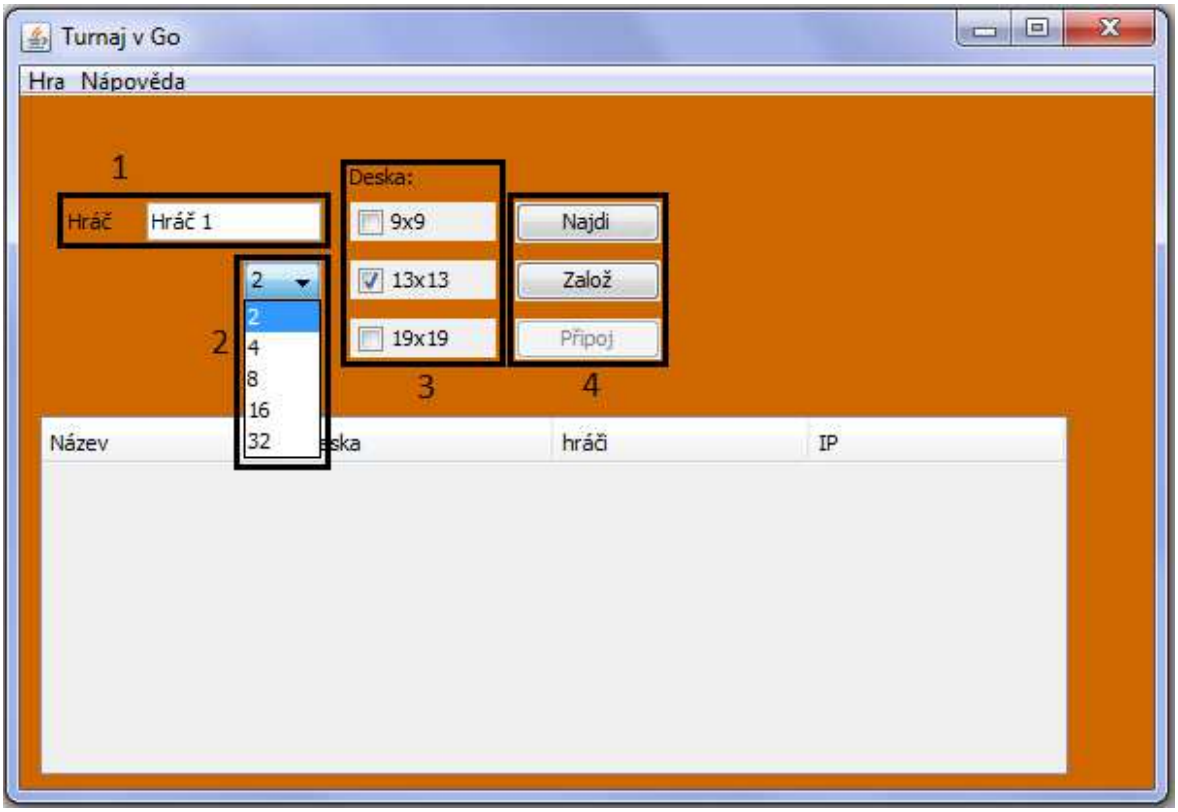
1 –Tlačítko„*Nová hra*“zobrazí uživateli herní obrazovku pro zakládání a vyhledávání turnajů.

2 – Menu„*Hra*“ nabízí položku „*Nová hra*“, jež umožňuje založit novou hru (viz 1) a položku „Konec“, která aplikaci ukončí.

3 – Menu „*Nápověda*“nabízí položky„*Pravidla*“, kde se nachází popis pravidel hry Go, „*Ovládání*“, kde se nachází popis ovládání aplikace a „*O Aplikaci*“, kde je uveden název aplikace, verze aplikace a jméno autora.

Vytvoření turnaje a připojení se k existujícímu turnaji

Pro vytváření a vyhledávání turnajů slouží herní obrazovka někdy nazývaná též „*lobby*“. (viz Obrázek 15 a Obrázek 16)



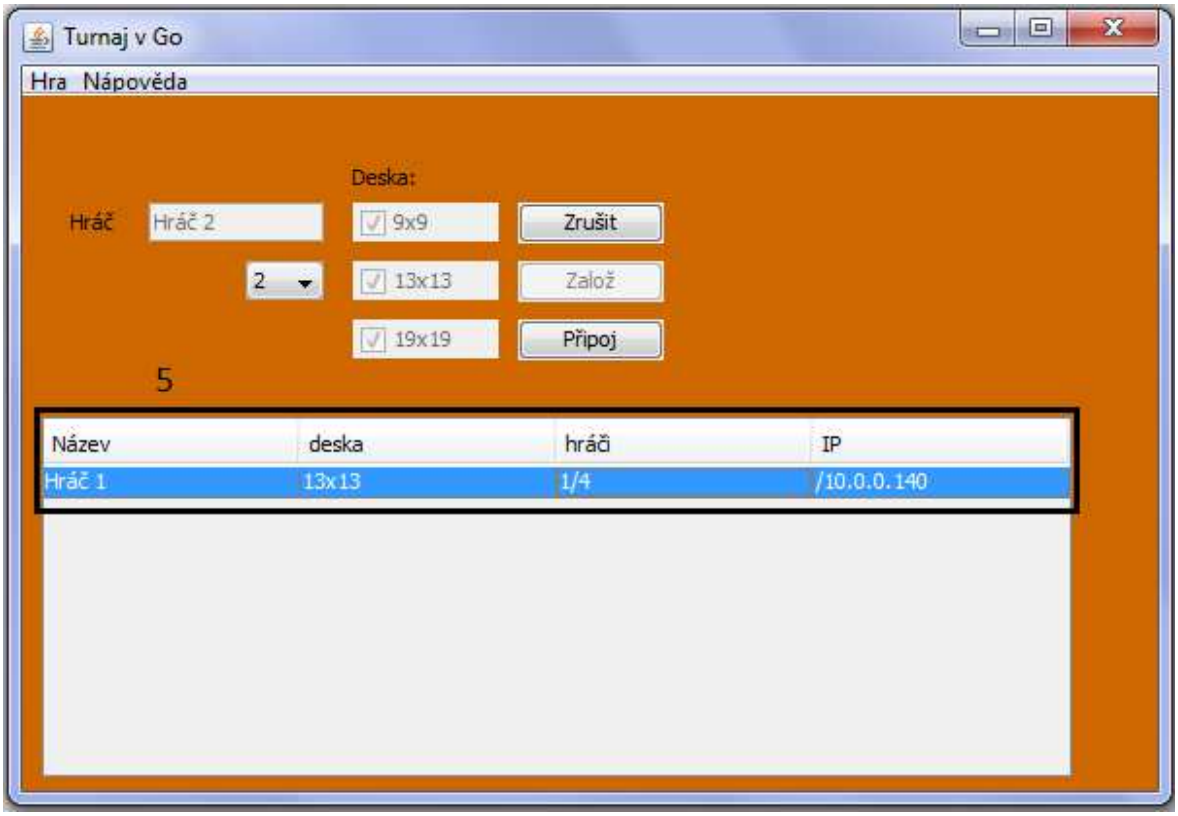
Obrázek 15 – Herní obrazovka pro založení turnaje

1 –Vstupní pole pro zadání jména hráče. Pokud je turnaj založen, zobrazí se toto jméno i jako název turnaje.

2 –Určuje požadovaný počet hráčů, aby mohl být spuštěn turnaj. Týká se pouze zakládání turnajů, vyhledávání probíhá bez ohledu na počet hráčů požadovaných pro turnaj.

3– Výběr velikosti hrací desky.Pro založení turnaje je nutné vybrat právě jednu velikost hrací desky, která bude používána pro celou dobu turnaje.Pokud jsou turnaje vyhledávány, je nutné,aby byla zaškrtnuta alespoň jedna velikost.Zobrazí se pouze turnaje, jež mají hráči vybranou velikost hrací desky.

4 –Tlačítka akcí. Po vybrání podmínek je možné zde založit turnaj tlačítkem „*Založ*“, nebo vyhledávat turnaje tlačítkem „*Najdi*“. V obou případech lze akci přerušit kliknutím na totéž tlačítko, jehož nápis se změnil na „*Zrušit*“ (viz 5). Pokud bude vhodný turnaj nalezen a vybrán, lze se k němu připojit kliknutím na tlačítko „*Připojit*“ (viz 5).



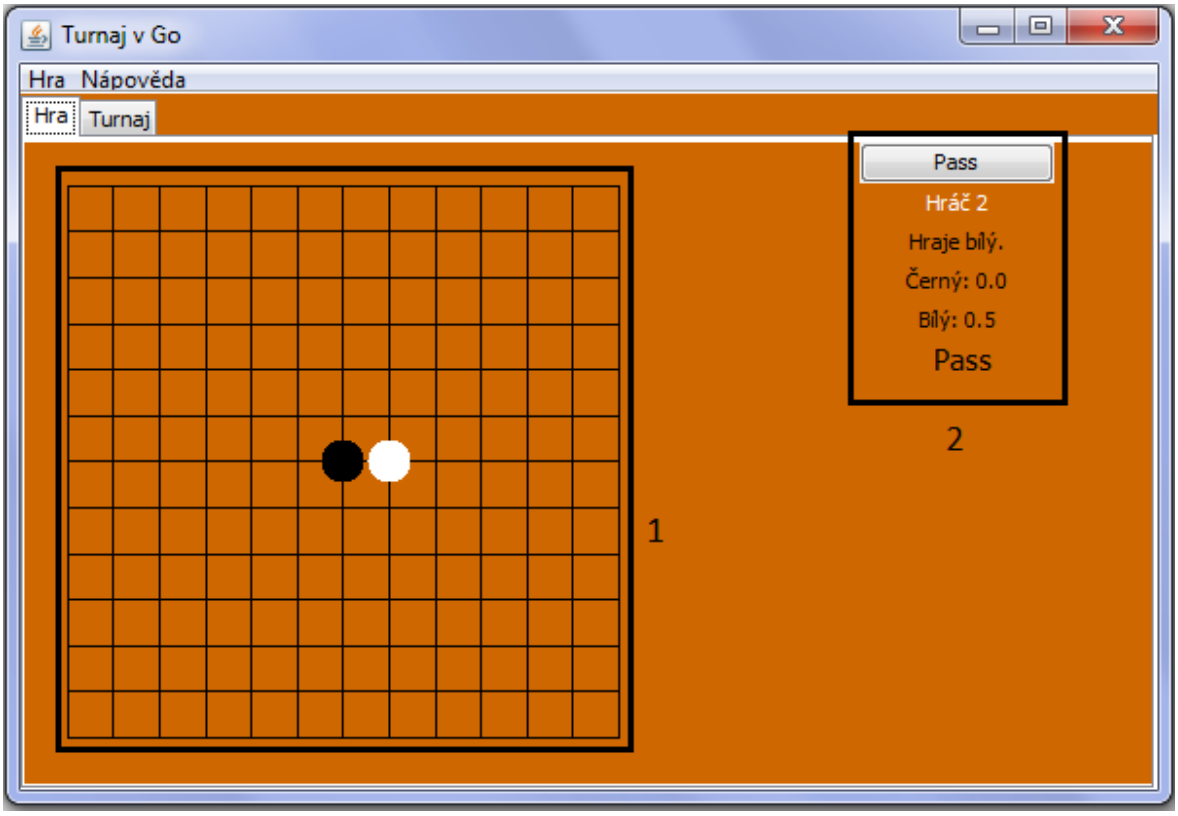
Obrázek 16 – Herní obrazovka pro vybrání turnaje

5 –Turnaje s vyhovující velikostí hrací desky jsou zobrazeny zde. Položka „*Název*“ obsahuje jméno hráče, jež turnaj zakládá, položka „*deska*“obsahuje velikost hrací desky. Položka „*hráči*“ obsahuje dvě hodnoty, první je počet připojených hráčů, druhá je požadovaný počet hráčů. Položka „*IP*“ obsahuje IP adresu turnajového serveru, aby bylo možné rozeznat servery v případě shodných názvů.

Kliknutím na řádek se založeným turnajem jej lze označit a připojit se k němu kliknutím na tlačítko „*Připojit*“ Pokud je turnaj zakládán, zobrazí se zde pouze informace o tomto turnaji, na který bude klient automaticky připojen, tudíž v položce „*hráči*“ bude pro **n** hráčů hodnota nejméně „1 / **n**“.

Ovládací prvky utkání

Po zahájení turnaje má uživatel k dispozici dvě záložky. Je to záložka „*Hra*“, kde probíhá vlastní utkání, a záložka „*Turnaj*“, kde jsou zobrazeny informace o turnaji.



Obrázek 17 – Ovládací prvky utkání

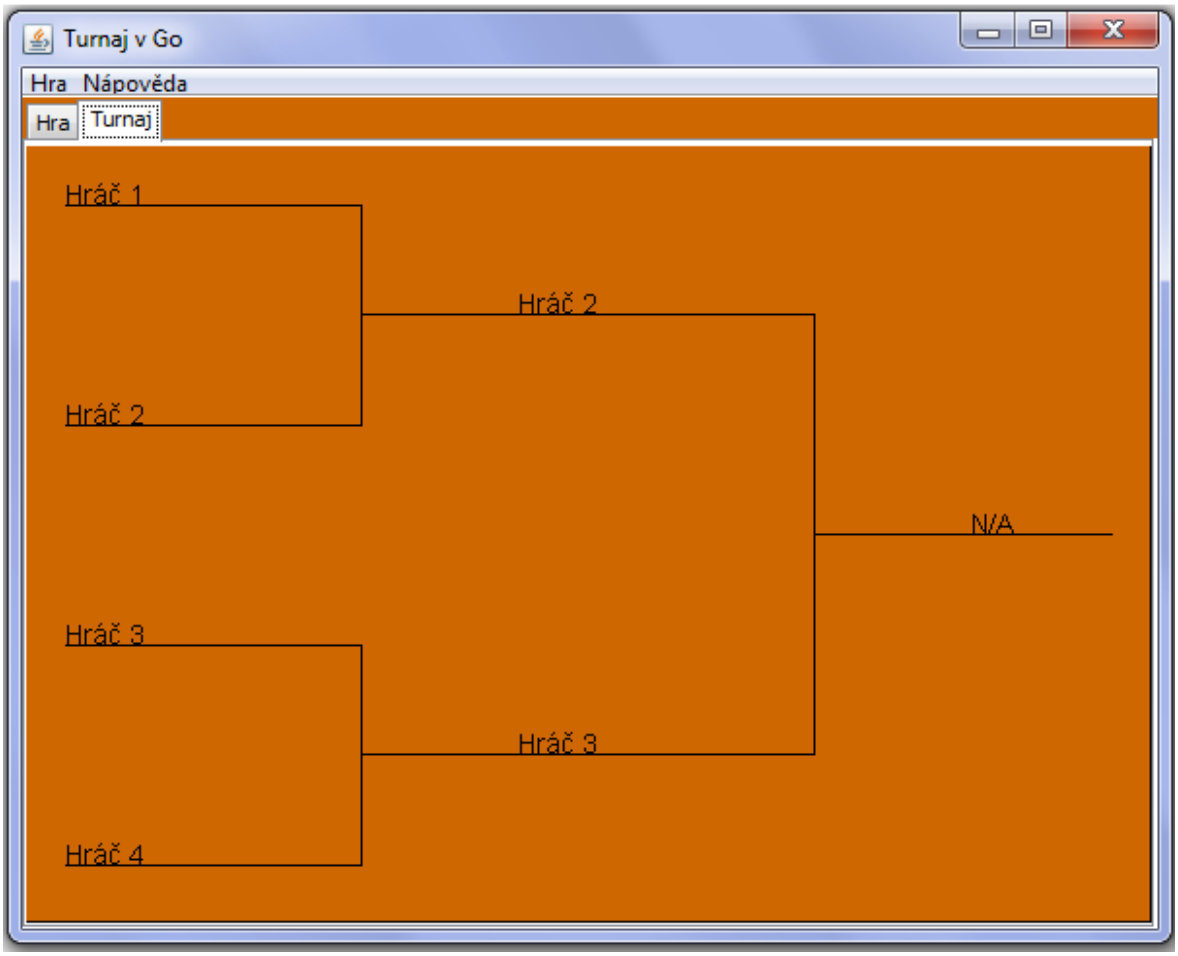
1 – Hrací deska. Kameny se pokládají kliknutím levého tlačítka myši tam, kde je povolený tah. Povolené tahy jsou znázorněné zeleným kroužkem (viz Obrázek 3), nepovolené tahy červeným kroužkem (viz Obrázek 5).

2 –Tlačítko „*Pass*“. Kliknutím na tlačítko „*Pass*“ se hráč vzdá tahu, pokud se vzdá tahu poté, co se vzdal tahu protihráč, ukončí tím utkání.

První nápis pod tlačítkem „*Pass*“ je jméno hráče zobrazené v barvě hráčových kamenů. Druhý nápis informuje hráče o tom, kdo je nyní na tahu. Následující dva číselné údaje jsou počty bodů obou hráčů. Poslední údaj je upozornění, že se hráč vzdal tahu, opět v barvě hráče.

Informace o průběhu turnaje

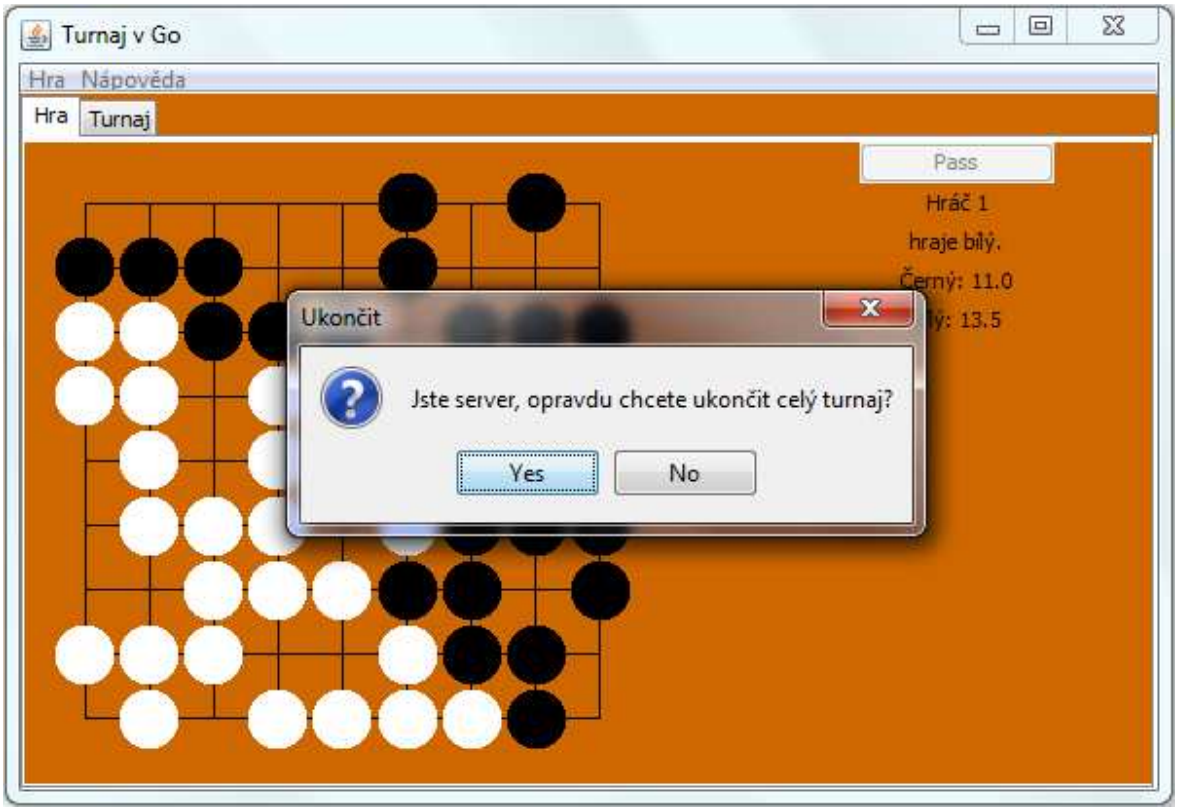
Informace o turnaji jsou dostupné na záložce „*Turnaj*“ během celého turnaje, a to i pro hráče, jež byl v některém z kol vyřazen a již se turnaje neúčastní.



Obrázek 18 – Grafické znázornění turnaje

Turnaj je graficky znázorněn ve formě turnajového „pavouka“. Diagram je orientován zleva doprava, kdy každá úroveň představuje jedno kolo turnaje. Na daných pozicích jsou zobrazena jména hráčů, nebo, není-li znám vítěz z předešlého kola, je zde zobrazen text „*N/A*“.

Ukončení aplikace



Obrázek 19 – Ukončení serveru

Ukončit aplikaci lze zavřením okna tlačítkem „x“, nebo z menu Hra – Konec. Pokud je běžící aplikace pouze klient, hra se ukončí. Pokud se odpojí klient, může turnaj dále pokračovat, pokud je ale aplikace zároveň server, zobrazí se dotazovací dialog, zda chce uživatel aplikaci opravdu ukončit, jelikož ukončením serveru bude zrušen celý turnaj.

Závěr

Mým cílem bylo vytvořit aplikaci Turnaj v Go pro hraní čínské deskové hry Go pro více hráčů. Zároveň jsem se chtěl pokusit o implementaci herních mechanismů a vytvořit aplikaci, která obsluhuje více hráčů přes místní síť.Vývoj této aplikace mi přinesl cenné zkušenosti v oblasti návrhu desktopových aplikací, síťové komunikace a práce s grafickými komponentami z knihovny Swing.

Vývoj provázely některé komplikace, například nutnost ošetření odpojení hráčů a detekce ztráty spojení. Toho bylo docíleno pomocí zachytávání výjimek. Další komplikace byla s přiřazením portu k ServerSocketu. Pokud byl jednou port přiřazen, tak nebyl JVM uvolněn, dokud nebyly ukončeny všechny Javovské aplikace. A to přes použití příkazu `setReuseAddress(true)` a řádné uzavření ServerSocketu. Toto znemožňovalo založit server podruhé bez restartu aplikace. Tento problém byl částečně vyřešen tak, že byl ServerSocket vytvořen a inicializován hned po startu aplikace a uzavřen až při ukončení aplikace. Další komplikací bylo nastavení hodnoty posuvníku u oken „Ovládání“ a „Pravidla“, jelikož při zobrazení byl posuvník umístěn dole u poslední vkládané komponenty, a to nebylo žádoucí. Tento problém byl vyřešen použitím nástroje `SwingUtilities.invokeLater`, který nastavil hodnotu posuvníku po dokončení inicializace komponent.

Turnaj v Go má hodně prostoru pro vylepšení, ať po funkční, nebo grafické stránce. Lze rozšířit možnosti obrazovky zakládání hry („lobby“) a turnaje o doplňková pravidla používaná na oficiálních turnajích AGA. Ta zahrnují mimo jiné úroveň hráčů, definují zvýhodnění pro slabšího hráče v podobě předem umístěných kamenů na hrací desce, nebo různé hodnoty „Komi“ dle rozdílu úrovně obou hráčů. Dále je možné upravit vzhled turnajového pavoukakvůli větší přehlednosti a implementovat okamžité aktualizace stavu turnaje, namísto aktualizace pouze po konci turnajového kola. Bylo by vhodnépřidat možnost (po odehrání vlastního utkání nebo po vyřazení z turnaje) sledovat rozehrané hry mezi ostatními hráči.

7 Citovaná literatura

Bloch, Joshua. 2001.*Java efektivně: 57 zásad softwarového experta.* Praha : Grada, 2001. ISBN 80-247-0416-1.

Calvert, Kenneth L. 2008.*TCP/IP sockets in Java: practical guide for programmers.* Amsterdam : Elsevier, 2008. ISBN 978-0123742551.

Elliotte, Rusty Harold. 2005.*Java network programming.* Sebastopol, Calif., : O'Reilly, 2005. ISBN 05-960-0721-3.

Iwamoto, Kaoru. 1976.*Go for beginners.* Harmondsworth : Penguin Books, 1976. ISBN 978-0394733319.

Lewis, Bil a Berg, Daniel J. 2000.*Multithreaded programming with Java technology.* Upper Saddle River, NJ : Prentice Hall, 2000. ISBN 978-0130170071.

Matthews, Charles. 1999.*Go.* Lincolnwood, Ill : NTC/Contemporary Pub, 1999. ISBN 978-0844226774.

Shotwell, Peter. 2003.*Go!: more than a game.* Boston : Tuttle Pub., 2003. ISBN 978-0804834759.

Soo-hyun, Jeong. 1994.*The Korean Go Association's Learn to play go.* New York, NY : Good Move Press, 1994. ISBN 09-644-7961-3.