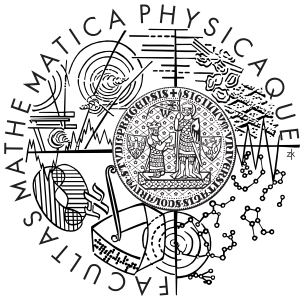
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Romana Hamplová

Použití vzorů při hře Go

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Jan Hric
Studijní program: Informatika, Programování

2009

Děkuji svým rodičům, vedoucímu bakalářské práce a příteli za veškerou pomoc a svatou trpělivost.

Prohlašuji, že jsem svou bakalářskou práci napsal(a) samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne Romana Hamplová

Obsah

Název práce: Použití vzorů při hře Go
Autor: Romana Hamplová
Katedra (ústav): Katedra teoretické informatiky a matematické logiky
Vedoucí bakalářské práce: RNDr. Jan Hric
e-mail vedoucího: Jan.Hric@mff.cuni.cz

Abstrakt: V této práci implementujeme řešič taktických cílů pro hru Go (např. zajmutí kamenů, spojení skupin). Tento řešič je založen na rozpoznávání tvarů kamenů na desce a výběru tahů podle předdefinovaných vzorů. Hlavní částí práce je analýza a návrh vhodné reprezentace vzorů pro zachycení typických situací a jejich řešení za útočníka i obránce. Uživatelské rozhraní programu umožňuje spouštění řešiče včetně možnosti zadávání druhu taktického cíle, vytváření nových vzorů a editaci celé databáze uložených vzorů. Součástí je i připravená databáze vzorů.
Klíčová slova: umělá inteligence, go, tesuji, taktické cíle

Title: Applications of Patterns in Game Go
Author: Romana Hamplová
Department: Katedra teoretické informatiky a matematické logiky
Supervisor: RNDr. Jan Hric
Supervisor's e-mail address: Jan.Hric@mff.cuni.cz

Abstract: This work implements a solver for tactical goals in the game of Go (for example capturing stones, connecting groups). The solver is based on recognizing the shapes of stones on the desk and choosing the moves from predefined patterns and the follow-up moves. The main part of this work features analysis and proposal of convenient representation of patterns which follow typical situations in the game and their solving considering both the attacker and the defender. User interface of the program allows execution of the solver including selection of the type of the tactical goal, creating new patterns and editing the whole database of patterns. The work also contains a precreated pattern database.
Keywords: Artificial Intelligence, go, tesuji, tactical goals

Kapitola 1

Úvod do problematiky

1.1 O hře go a umělé inteligenci

Hra go je nejstarší deskovou hrou na světě a i přes svá jednoduchá pravidla (viz Příloha A) je stále považována za nejkomplexnější strategickou deskovou hru. V posledních desetiletích si svou náročností získala i odborníky ze světa informatiky, protože go je poslední významná desková hra, pro kterou stále neexistuje program, který by byl schopen v rovné partii porazit profesionálního hráče. Otázkou umělé inteligence v go se zabývají profesionální týmy po celém světě. Momentálně nejúspěšnějším softwarem je MoGo [2] , založené na metodě pseudonáhodného výběru tahů pomocí metody Monte-Carlo a vyhledávacím algoritmu UCT [2]. MoGo již je schopné porazit profesionální hráče, ale s hendikepem a běžící na superpočítači.

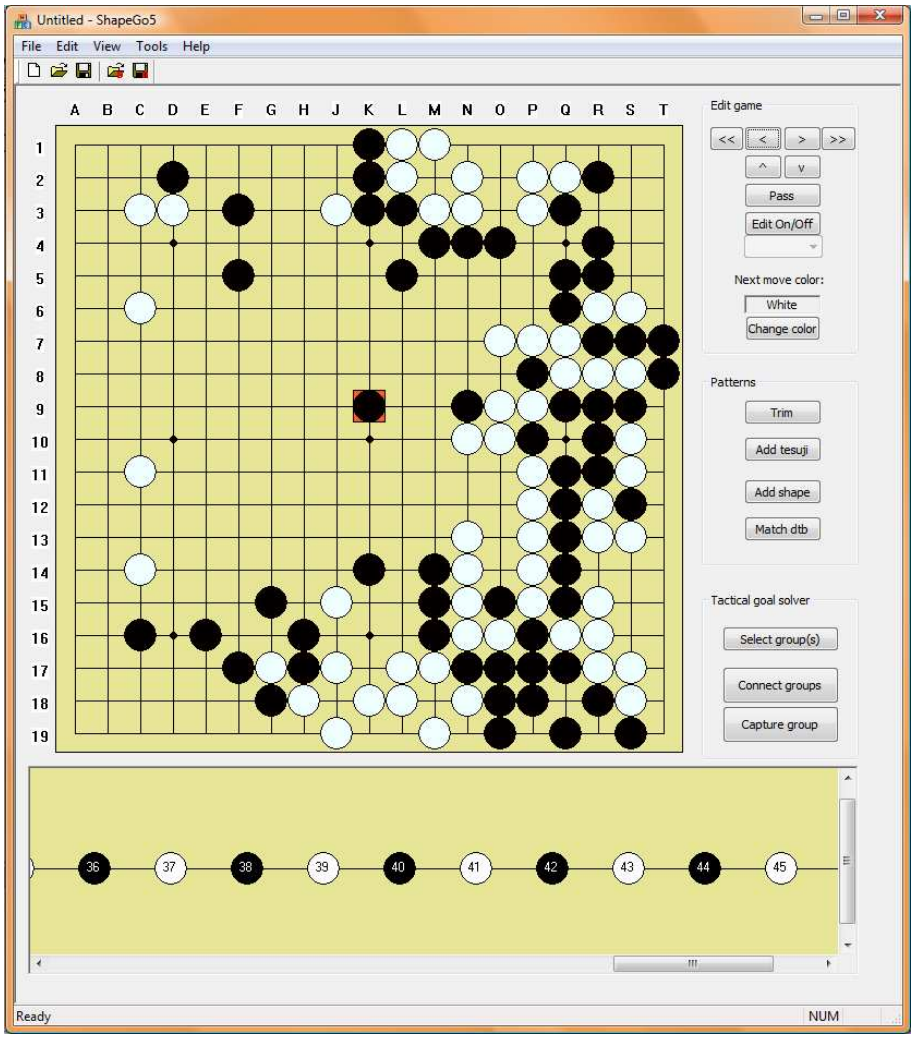
Go má vysoký koeficient větvení při zkoušení tahů a proto není možné použít jednoduché metody prohledávání všech možností. Ani metody úspěšné v jiných deskových hrách (např. v šachách) neuspěly. Pokud bychom go porovnávali s šachy, pak jsou tu tři největší odlišnosti. Šachové algoritmy v nejhorším případě propočítávají průměrných 40 možných pozic v každém tahu. Ohodnocovací funkce je poměrně jednoduše definovatelná, díky čemuž je možné alfa-betou ořezat počet možností na tah cca na 10. Navíc je možné předpřipravit určitou strategii, podle které bude počítač hrát. V go je první tah možné umístit na 361 pozic, v průměru je pro každý tah 200 možností a hra obvykle má 200 až 300 tahů, což dává příliš vysoký počet možností při prohledávání do hloubky. Výběr dobré statické ohodnocovací funkce prakticky není možný, protože tahy, které se v lokálním měřítku zdají zbytečné či nevýhodné, mohou mnohem později výrazně ovlivnit směr hry

v neprospěch protihráče. Připravená strategie (plán hry), která by výrazně ořezala koeficient větvení, také není vhodna. Připravený plán je v go spíše přítěží, protože je nejdůležitější flexibilita do té míry, kdy je nutné po jednom tahu zcela změnit strategii. Z těchto důvodů jsou nejdůležitější heuristiky přizpůsobené přímo pravidlům a zvláštnostem go.

V dalším textu se vyskytují specifické goistické termíny a proto je vhodné znát alespoň základní pravidla Go např. z Přílohy A - Stručná pravidla hry go a rejstřík goistických pojmů.

Implementace komplexní umělé inteligence je otázkou spíše diplomové práce ve velkém týmu a proto jsem se rozhodla zabývat menším celkem. V go se můžeme setkat se situacemi, kdy vítězství nebo prohra závisí na správném zvládnutí dané situace, taktického cíle. Jako příklad mohu uvést situaci, kdy je nutné spojit dvě skupiny, které by byly samy o sobě mrtvé, ale spojené jsou živé, situace, kdy je bezpodmínečně nutné zajmout nějaký soupeřův kámen, nebo koncová fáze hry, kdy je nutné ovládat nejlepší způsoby zmenšování soupeřova území. Pro svou práci jsem tedy zvolila plnění taktických cílů.

Jednou z méně probádaných oblastí v počítačovém go je umělá inteligence založená výhradně na rozpoznávání tvarů na desce. Hledání známých tvarů výrazně vylepšilo i zmiňované MoGo [2], ale v jednoduché podobně se nachází v každém významnějším počítačovém hráči jako například v programu GnuGo [3]. Nejčastěji je rozpoznávání tvarů využito pro knihovnu zahájení (Joseki). Ovšem málo bylo zkoumáno rozpoznávání tvarů pro hlavní výběr tahů. Proto je tato práce založena na výběru tahů a hraní podle předdefinované sady vzorů.



1.2 O programu ShapeGo

Ve této práci jsme si dali za cíl vytvořit program, který bude schopen řešit taktické cíle hry go (např. zajmi danou skupinu kamenů, spoj dvě skupiny kamenů, rozděl soupeřovy kameny, a pod.). Hlavním úkolem tedy bylo vytvoření umělé inteligence založené na bázi rozpoznavání vzorů kamenů na desce (gobanu) a hraní podle předdefinovaných nejlepších odpovědí pro daný vzor. Nejdůležitější součástí bylo vymyslet vhodnou implementaci vzorů, která by umožnila dobré uložení všech potřebných informací, lehké vyhledávání v databázi těchto tvarů a snadné porovnávání s pozicemi na celé desce.

V neposlední řadě jsme se zaměřili i na vhodné GUI, pomocí kterého je uživatel schopen definovat nové vzory případně editovat databázi persistentně uložených tvarů a i spouštět řešič taktických cílů.

Výsledkem je okenní aplikace ShapeGo určena pro platformu Windows, jejímž hlavním zaměřením je spouštění řešiče taktických cílů pro hru go. Třída uživatelů tohoto programu budou nejspíše hráči go a programátoři, kteří se o go a problematiku s ním spojenou, zajímají. Z tohoto důvodu program umožňuje načítání a kompletní integraci uložených her ve formátu sgf a vykreslení stromu hry pro pohodlné prohlížení odehrané partie, na což jsou hráči go z podobných programů již zvýklí. Pro programátory, kteří by chtěli tento program dále vyvíjet, je nejdůležitější rozšiřitelnost. Proto je implementace napsána podle standardů objektově orientovaného programování se striktním dodržováním interfaců pro všechny důležité třídy. Program navíc umožňuje spouštět řešič taktických cílů z okenního prostředí.

Kapitola 2

Hlavní problémy

V této práci bylo úkolem porovnat možné implementace nejdůležitějších částí programu, případně i vyzkoušet jejich účinnost při spouštění řešiče taktických cílů včetně rozboru nejvíce zpomalujících částí.

2.1 Ukládání vzorů

Návrh implementace vzorů hry go byl hlavním úkolem v této práci. Implementace musela umožňovat použití databáze při řešení taktických cílů, tedy rychlé prohledávání, možnost porovnání se stávající situací na gobanu a možnost zjištění, zda je daný taktický cíl splněn. Navíc bylo nutné umožnit uživateli vkládání nových vzorů a prohlížení databáze.

Podařilo se nám implementovat ukládání vzorů pomocí Zobristova hashování, které nám výrazně urychluje porovnání vzorů mezi sebou. Díky ukládání do dvou databází (viz kapitola Implementace, sekce Databáze vzorů) je usnadněno přidávání vzorů uživatelem a umožněno načítání i ukládání vzorů do souborů.

2.2 Vyhledávání vzorů na gobanu

Rychlost celého řešiče taktických cílů závisí na rychlosti vyhledávání tvarů na desce a porovnávání s databází tvarů. V základní verzi je potřeba porovnat všechny obdélníky na desce s tvary v databázi stejné velikosti. Vidíme, že zásadním faktorem zpomalení by měla být velikost databáze. Ta navíc není rovna počtu vkládaných tvarů, ale počtu vyhledávacích tvarů. Každý

vkládáný vzor je totiž nutné vyhledávat i ve všech otočeních o 90°, orotovaný po diagonále a pro obě barvy. (viz. podkapitoly Databáze vzorů a Tvary a tesuji).

Vyhledávání vzorů na gobanu může mít v nejhorším případě časovou složitost až $O(n * m * s * d)$, kde n a m je šířka a výška gobanu, s je počet velikostí vzorů uložených v databázi a d je rychlost zjištění, zda se daný tvar nachází v databázi.

Pro rychlost prohledání databáze je kritická její velikost. Velikost databáze může být v nejhorším případě $s * 8 * b + t * 8$, kde s je počet tvarů (shapes), b je počet černých kamenů ve tvaru, t je počet tesuji, a 8 je počet rotací. Oproti očekávání ovšem velikost databáze tvarů ovlivnila výslednou rychlost řešiče taktických cílů pouze logaritmicky, tedy zanedbatelně v porovnání s ostatními úkony, vykonávanými při vyhledávání tvarů na desce. Viz následující algoritmus.

V naší základní verzi prohledávání na gobanu probíhá podle následujícího algoritmu:

```
Pro každou velikost s z databáze {
   Pro každou pozici na desce {
      Vytvoř hash pro aktuální část gobanu velikosti s;
      if ( hash se nachází v databázi ) {
         Spoj strom tahů tohoto hashe s výsledným stromem;
      }
   }
}
```

2.3 Určení splnění taktického cíle

Určení splnění taktického cíle bylo nutné řešit pro každý taktický cíl zvlášť. Pro tento účel byla využita implementace gobanu pomocí pamatování si informací o skupinách. Tato implementace má v konstantním čase přístup k informacím o kameni na zadaném políčku, počtu kamenů skupiny, k ID skupiny a především k počtu svobod každé skupiny. Zjištění, zda je daná skupina kamenů zajata, či zda jsou skupiny spojeny, je díky tomu možné provést v konstantním čase, což přináší veliké zrychlení při určování, zda je cíl splněn.

Kapitola 3

Hashování vzorů

Po zanalyzování asymptotické složitosti vyhledávání vzorů na gobanu jsme se rozhodli pro nutnost implementace hashování vyhledávacích tvarů. V programu tak vznikly dvě třídy pro vytváření klíčů vzorů, které je možné ukládát do vyhledávací databáze. Jednoduché uložení vzoru v matici s hodnotami kamenů a Zobristovo hashování.

3.1 Jednoduchá matice

Matice obsahuje hodnoty černý, bílý a prazdné políčko. Původně byla implementována pouze k testování jakožto nejintuitivněji naprogramovatelný způsob klíčování tvarů. Porovnání dvou matic je v lineárním čase, avšak po jednom vylepšení při vyhledávání tvarů na velké oblasti desky se může k Zobristovu hashování blížit a proto jej ponechávámé v implementaci pro porovnání.

3.2 Zobristovo hashování

Zobristovo hashování vynalezené v [1] je starší metodou hashování, která byla výhradně určena k rozpoznávání vzorů ve hře go a v šachách. V obou těchto hrách se velice osvědčila a stále je využíváná v programech hrajících go např. GnuGo [3].

Zobristovo hashování je založeno na tom, že pro každé políčko je vygenerováno náhodné číslo pro každou hodnotu, která se na daném políčku může vyskytovat. V šachách to znamená generování náhodných čísel na jednom

políčku pro každou figurku (bílou i černou) a navíc prázdné políčko. V go se generují náhodná čísla pouze pro bílý kámen, černý kámen a prázdnou pozici. V naší reprezentaci tedy máme vygenerováno $361 * 3 = 1083$ náhodných čísel. Avšak 32 bitové číslo má příliš nízký rozsah hodnot, z čehož plyne vysoká pravděpodobnost kolizí. Z tohoto důvodu je zvláště pro go, které má velmi vysoký počet možných variant rozložení desky, vhodné použít alespoň 64 bitů pro hash, což jsme zvolili i my a reprezentovali pomocí dvou 32 bitových čísel.

Zobristovo hashování v naší implementaci využívá počateční inicializaci v prvním průchodu získávání hashe. Tato počáteční inicializace ukládá pro každé políčko tři náhodná 64bilová čísla. Hash pro část desky (víme, že je vždy obdelníková) má podobu jednoho 64bitového číslo. Toto číslo se získává průchodem inicializované reprezentace Zobrista - matice náhodných čísel. Pro každou pozici z odpovídající části v matici náhodných čísel se zavolá operace XOR s výsledným hashem. Operace XOR má tu výhodu, že u dvojitého xorování stejné cifry se výsledek nezmění a proto je možné jednoduše přidávat i odebírat tahy. Hash je tedy jedno číslo a díky tomu porovnání dvou zahashovaných částí gobanu je v konstantním čase. Vytváření hashe je lineární vůči velikosti gobanu, který se hashuje.

3.3 Porovnání a vylepšení

Vytváření klíčů je u obou reprezentací v lineárním čase, ale porovnání dvou vytvořených klíčů je u Zobrista konstantní, zatímco matice opět vyžaduje lineární čas. Obě metody jsme vyzkoušeli při 10000 násobném matchování databáze (se stejnými vstupními daty) a Zobristovo matchování nám snížilo potřebný čas na polovinu.

Matici jsme zkoušeli dále vylepšit počáteční inicializací na hodnoty celého gobanu. Matchování gobanu pak mělo mírně změněný algoritmus vytváření hashe v implementaci, kdy místo vytváření hashe si pouze matice pamatovala offset - pozici levého horního rohu - a porovnání probíhalo vůči tomuto okénku. Tato implementace nám, navzdory očekávání, pro menší databázi prohledávání zpomalila o 30%. Vylepšení by se začalo projevovat až u velkého počtu velikostí vzorů. Právě počet velikostí nám zvyšuje počet vytváření hashů.

Zobristovo hashování by bylo možné dále zrychlit metodou GetNextLine a GetNextColumn, které by nevytvářeli celý nový hash, ale pouze oparecí XOR znovu "přidali" první řádek, čímž by se díky symetrii XORu odebral, a

stejnou operací přidali řádek nový. Toto vylepšní jsme již neimplementovali, protože zrychlení by nebylo dostatečně velké pro námi používané databáze.

Kapitola 4

Implementace

4.1 Databáze vzorů

Databázi vzorů bylo vhodné uložit do paměti počítače ve dvou verzích. Hlavní a objemnější databáze je výhradně určena pro použití při vyhledávání vzorů v průběhu řešení taktických cílů a jsou v ní uložené tvary ve formátu, v jakém se budou vyhledávat na hrací desce. Pomocná databáze slouží pro rychlejší ukládání persistentních dat na disk a možnou editaci celé databáze vzorů uživatelem. Jsou v ní tedy uloženy tvary ve stejném formátu, ve kterém byly vloženy uživatelem.

Hlavní databáze vzorů je uložená v podobě hashovací tabulky, kde vyhledávcí klíč je hash vzoru, podle kterého bude vzor vyhledáván na desce, a hodnota je sestavený strom tahů, které jsou vhodnou odpovědí na daný vzor. Díky využití standardní knihovny použitého jazyka C++ a třídy map, která ukládá data do vyváženého vyhledávacího binárního stromu, je vyhledávání v tabulce v nejhorším případě s logaritmickou složitostí.

Každý tvar je nutné vyhledávat i ve všech otočeních o 90° a jejich zrdcadleních. Pro přepočítávání rotací bylo využito klasického geometrického vzorečku pro rotaci:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix}$$

Ježto se $\alpha = \frac{\pi}{2}$ a levý horní roh tvaru je vždy v počátku souřadnic, takže je nutné posunutí, vzorec se redukuje na

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} y \\ -x + h \end{pmatrix}$$

,

kde h je rovno výšce rotovaného vzoru. U tesuji je navíc potřeba rotovat strom tahům, kde se používá inverzní vzorec.

Přepočítávaní rotací by bylo v průběhu výpočtů taktických cílu veliké zdržení a proto jsou i všechny transformace vzorů uloženy v databázi stejným způsobem jako originální vzor. Z tohoto důvodu je tato databáze mnohonásobně objemnější.

Po orotování je potřeba tvary přetransformovat do podoby klíče (hashe) a v této podobě se stromem tah; vložit do databáze. V případě, že se přidávaný hash v databázi již nachází, pak to znamená, že pro jeden tvar na gobanu existuje více správných odpovědí. V takovém případě se slučuji stromy odpovědí v jeden, stejný hash se již znovu neukládá.

Pomocná databáze ukládá vzory jako rozložení kamenů na desce a to každý vzor pouze v jedné verzi. Vzory z této pomocné databáze i se stromy tahů jsou vykreslovány v programu při editaci databáze pro prohlížení uživatelem. Celá databáze je ukládána do soubory v upraveném formátu .sgf.

```
Vkládání tvaru do databáze {
   Vygeneruj rotace tvaru;
   Pro každou rotaci {
      Vygeneruj vyhledávací tvary spolu se stromy tahů;
      Pro každý vyhledávací tvar{
         Vytvoř hash;
      }
      Pro každý hash {
         If ( hash se nachází v databázi ){
            Sluč strom tahů hashe a strom tahů v databázi odpovídající hashi;
         } else {
            Vlož hash a strom tahů do databáze;
         }
      }
   }
}
```

4.2 Tvary a tesuji

Vzory jsme se rozhodli rozdělit do dvou typů. Tvar (neboli shape) je jednoduchý, přesně definovaný tvar kamenů na desce. Je to rozmístění kamenů, které je univerzalně dobré, nebo plní určitý taktický cíl. Tesuji, odpovídající goistickému pojmu (viz Rejstřík goistických pojmů), obsahuje určitou situaci na desce, za které se dá použít, a strom tahů, které je možné na tuto situaci zahrát. Tesuji obvykle i v goistických knihách obsahuje jako ukázku seznam možných reakcí soupeře a odpovědi na ně.

Shape je zadáván jako tvar, kterého je potřeba na desce dosáhnout. Při ukládání do hlavní databáze je tedy nutné vygenerovat nejen rotace, ale i všechny tvary bez jednoho černého kamene (jeden tah zpět, tzn. tvar, z kterého po zahrání jednoho tahu černého vznikne uložený shape. Až z tohoto tvaru (bez jednoho kamene) se vytváří hash a je ukládán do databáze. Hodnota databáze pro tento hash je strom s jedním tahem obsahující vymazaný kámen. Z jednoho tvaru se tedy do hashovací tabulky uloží maximálně

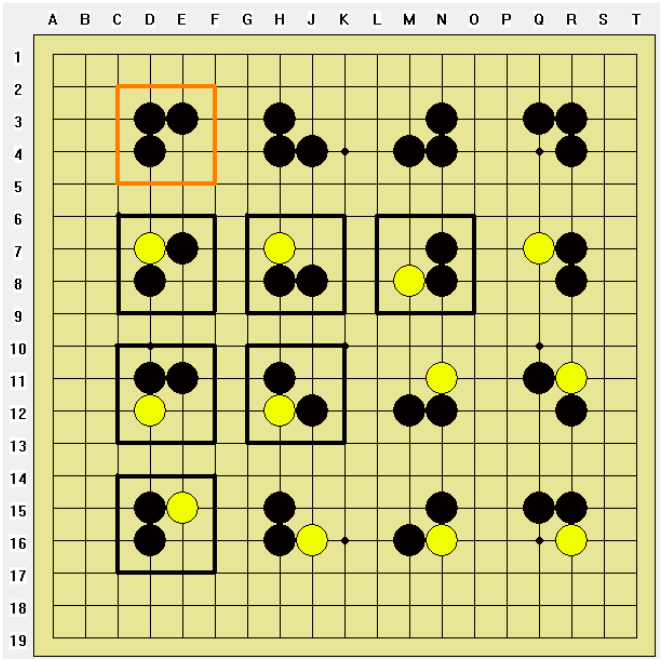
$$numOfRotations * numOfBlackStones = 8 * numOfBlackStones$$

hash klíčů, kde numOfRotations je počet vygenerovaných rotací (vždy 8) a numOfBlackStones je počet černých kamenů ve tvaru. Při ukládáni do databáze ale může docházet k tomu, že vyhledávací tvar už je jednou obsažen (např. s jiným stromem tahů). V tom případě se stejný hash znovu neukládá a dochází k slučování stromů tahů.

V následujícím obrázku je ukázka vygenerování vyhledávacích tvarů. Je použit goisty neoblíbený tvar známý jako prázdný trojúhelník. Červeně ohraničený je tvar vkládaný uživatelem. Na prvním řádku jsou vygenerované rotace. Mirroring by v tomto případě nevygeneroval žádný nový tvar oproti rotacím a proto jsou vykresleny pouze 4rotace, které jsou navzájem různé. Pod každou rotací jsou vyhledávací tvary, které vzniknou ubráním jednoho kamene.

Na obrázku můžeme vidět vzniklé duplicity. Ve výsledku se do vyhledávací databáze uloží pouze tvary orámované černým čtvercem a k nim odpovídající stromy tahů. V tomto případě pro každý ukládáný tvar bude mít strom tahů dva vrcholy. Do pomocné menší databáze, která je používána pro ukládání do souboru, bude uložen pouze tvar orámován červeně.

Tesuji je zadáváno přímo jako tvar, který bude vyhledáván, a strom následujících tahů. Oproti tvaru jsou do databáze ukládány pouze rotace vyhledávacího tvary. Avšak, aby řešič taktických cílů byl schopen rozpoznat



Obrázek 4.1: Ukázka ukládáni tvaru (goisty neoblíbený prázdný trojůhelník) a generování odpovídajících vyhledávacích tvarů.

i tesuji v průběhu jeho odehrávání, bylo nutné ukládat do databáze i mezitvary, které se ukládají stejným způsobej, jako celé tasuji, pouze s menším stromem. Tedy každé tesuji je v databázi nejvíce $8 * bn$, kde bn je počet synů s černým kamenem.

4.3 Algoritmus řešiče taktických cílů

Strom, který je procházen v řešiči taktických cílů je implementován jako klasický and-or strom. To znamená, že při kontrole, zda je taktický cíl splněn, stačí alespoň jedno splnění v černém vrcholu (černé vrcholy se or-ují) a všechny splnění v bílém vrcholu pro dokázání funkčnosti taktického cíle. Jediné nesplnění v bílém vrcholu pak znamená nesplnění celé jedné věve podstromu. Pro každý bílý tah tedy musíme mít odpověd, která dokáže splnit cíl.

```
Vyhledej na desce všechny vzory s daným cílem,
```

```
   které se dotýkají skupiny, která je součástí cíle;
while ( Neexistuje syn v hloubce 1,
       jehož všichni potomci plní taktický cíl ){
   Získej další tvar z databáze,
      který je relevantní k dané pozici.
}
```

4.4 Rozšiřitelnost

Po celou dobu vytváření programu byl důraz kladen na rozšiřitelost a jednoduché měnění implementací. V jazyce C++ jsme proto striktně dodržovali používání interfaců. Díky takto propracovanému datovému modelu je program velice variabilní.

Jako příklad uvádím, co je potřeba udělat pro přidání nového typu hashování:

1. Vytvořit třídu pro daný hash, která bude dědit od interfacu CHashPatternI a tedy implementuje virtuální metody. Zde je jediná důležitá metoda a to LessThan(), která vrací stejnou hodnotu, jako operátor menší než. Také jsou nutné všechny konstruktory, jako v ostatních implementacích (např. CHashZobrist).

2. Ve dvou souborech projektu změnit, že nevytváříme hash CHashZobrist, ale nově vytvořenou třídu hashe.

Těmito kroky je vytvořeno a používáno nové hashování.

Obdobně snadno lze rozšířit taktické cíle, kde je hlavní fází implementace třídy, která musí být schopna určit, zda je taktický cíl korektně zadán a zda je v dané situaci na desce splněn. Dále také třídy implementující goban (a tedy kotrolující pravidla) a strom hry. Tato variabilita umožňuje použít různé implementace pravidel při řešení taktických cílů tak, aby implementace a její vlastnosti byly optimalizovány pro daný typ taktického cíle.

Kapitola 5

Závěr

Podařilo se mi implementovat program s propracovaným datovým modelem s využitím interfaců, který je možné rozšířit na použití nového hashování, jiné implementace desky a kontroly pravidel, jiné implementace uložení tvarů i stromu hry a s jednoduchým přidáváním nových taktických cílů.

5.1 Možná zlepšení

V blízké budoucnosti bych ráda přidala mnou vymyšlené hashování vzorů pomocí matice bitů, kde každá pozice má velikost 3 bity. Do těchto bitů bude možné uložit nejen hodnoty černý, bílý a prázdný, ale i černý nebo prázdný, černý nebo bílý a bílý nebo prázdný. Díky zahashování těchto hodnot do bitů by porovnávání bylo možné pomocí bitového or. Tato reprezentace by nezrychlila vyhledávání vzorů oproti již implementovaným metodám, ale umožnila by zakódování mnohem většího množství informací o okolí vzoru, které je pro správnou funkčnost vzorů zásadní. Touto změnou by bylo možné ukládat i jednotlivé tahy v tesuji do tvarů včetně tahů, které by jinak zasahovaly mimo uložený vyhledávací tvar.

Universalita celého systému ukládání tvarů by umožňovala rozšíření řešiče taktických cílů i na tzv. joseki (viz Rejstřík goistických pojmů). Vytvořená persistentní databáze tvarů by díky své slučitelnosti s formátem .sgf umožňovala nahrání celé databáze Kogo's joseki dictionary [**?**]. Navíc, díky oddělení taktických cílů, by při joseki mohla být vyhodnocována situace i v okolních rozích (např. převaha kamenů jedné barvy). Jestliže by bylo možné použít takovýto výběr tahů, pak by se blížil způsobu, jakým se o tesuji rozhodují reální hráči. Pro toto rozšíření by bylo nutné přidat nový taktický cíl a po-

změnit podmínky pro zahrání tahu z databáze, ale tento práce k tomuto tématu nebyla určena.

Paralelizace

5.2 Pár slov na konec

Program ShapeGo byl pro mne velikou výzvou a velikou zkušeností. Výsledek mé práce má k dokonalosti rozhodně daleko, ale rozhodně bych ráda na programu dále pracovala, protože je stále mnoho vylepšení, které by ShapeGo mohlo posunout za další hranice, případně přiblížit použitelnosti pro běžného uživatele.

Literatura

[1] Albert L.Zobrist: *A Hashing method with Applications for game playing*, University of Wiscon Madison, 1969

[2] http://senseis.xmp.net/?MoGo

[3] http://www.gnu.org/software/gnugo/gnugo.htm

[4] James Davies: *Elementary Go Series, Vol.3 - Tesuji*, Kiseido publishing company, 1995

[5] C. Matthews: *Shape Up*

[6] http://www.goweb.cz

Příloha A

Stručný úvod do hry go

A.1 Pravidla

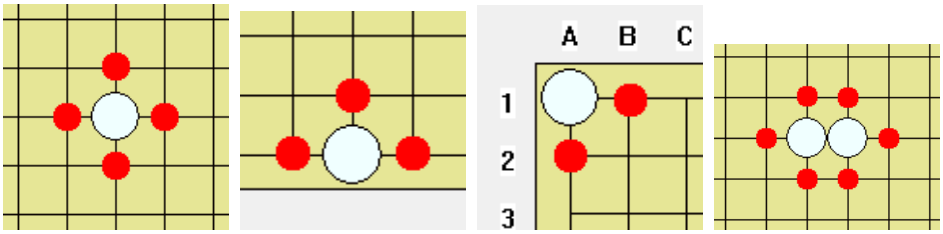
Go je známé svými jednoduchými pravidly a proto si je můžeme dovolit zde shrnout. Ke hře je potřeba hrací deska - tzv. goban - a kameny - černé a bílé. Hrají dva hráči, kteří se střídají v tazích. Začíná hráč, který má černé kameny. Tah spočívá v položení jednoho kamene na goban a to na průsečít. Po položení kamene na desku se s ním již nijak nepohybuje (kromě pravidla č.1 - viz níže).

A.1.1 Svobody

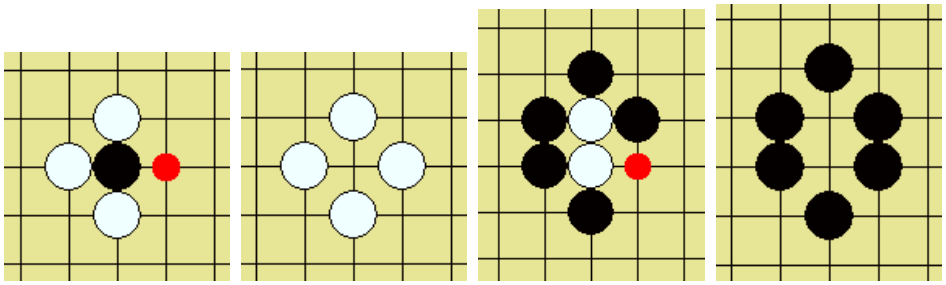
Důležitou vlastností každého kamene i každé skupiny kamenů stejné barvy na desce je počet sousedních volných průsečíku (tzv. svobod). Na následujících obrázcích jsou svobody bílých kamenů označeny červeně.

A.1.2 Tři jednoduchá pravidla go

Pravidlo č.1.:
Jestliže má kámen (či skupina kamenů) soupeře po posledním tahu 0 svobod, je zajata soupeřem a odebrána z desky.

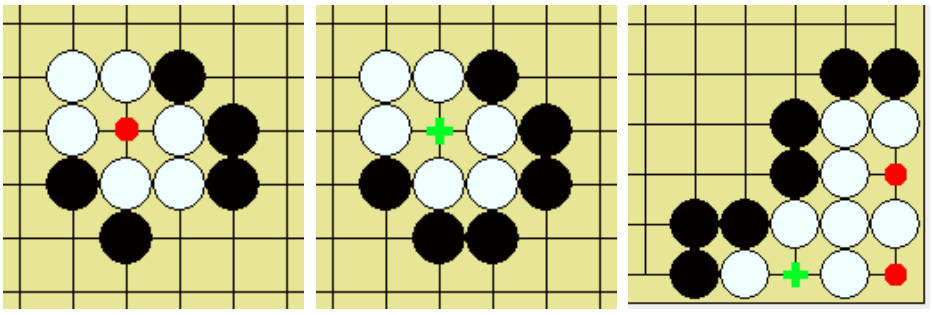


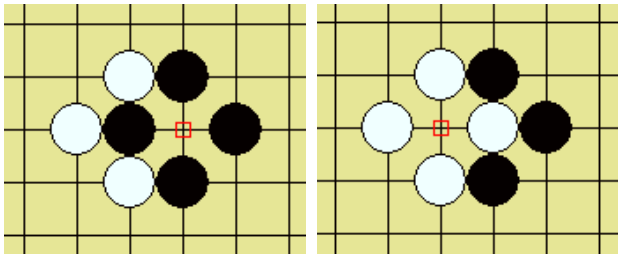
Obrázek A.1: Ukázka počtů svobod skupin kamenů. Červeně jsou označeny svobody bílých kamenů.



Obrázek A.2: Obrázky vždy bezprostředně před zajmutím kamenů (včetně označené poslední svobody) a po zajmutí skupiny a odebrání kamenů z desky

Pravidlo č.2.:
Tahem nesmíte spáchat sebevraždu - tzn. nemůžete zahrát na místo, kde by Váš kámen neměl žádnou svobodu, nebo byste tím ubrali poslední svobodu své skupině. Na takové místo ale můžete zahrát, jestliže tento nový tah vytváří nové svobody a to pravidlem č.1 (tzn. zajmutím soupeřových kamenů).





Obrázek A.3: Ukázka situace ko. Bývá zvykem značit situaci ko čtverečkem.

Pravidlo č.3.:
Ve speciální situaci může dojít k nekonečnému vzájemnému braní jednoho kamene - tzv.ko. V této situaci nesmíte zajmout kámen bezprostředně po jeho zahrání soupeřem, ale musíte alespoň jednou hrát jinam (tzv. hrozbu na ko).

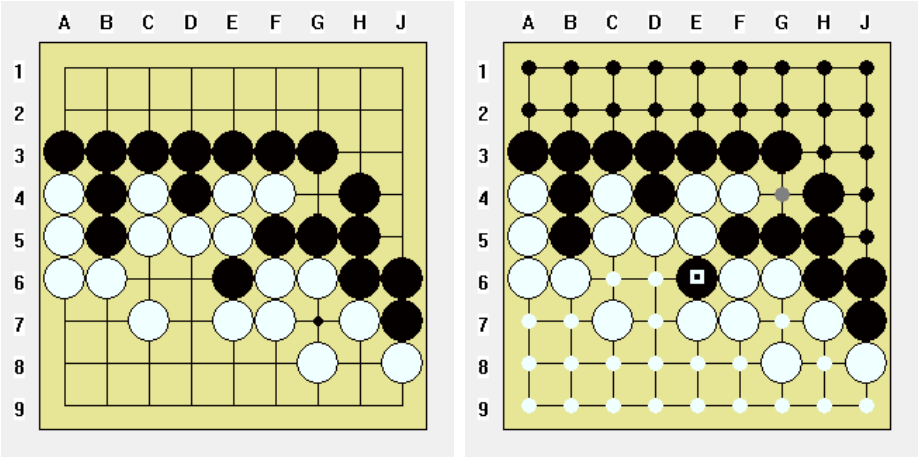
Z těchto pravidel se dají odvodit další zajímavé vlastnosti go, např. tzv. živost skupin, schody a pod. Viz rejstřík základních pojmů.

A.1.3 Cíle hry

Cílem hry je získat co nejvíce bodů ohraničeného území. Po jednom bodu získáváte za každý zajatý kámen soupeře a za každý průsečík svého území. Území můžeme nadefinovat jako prázdné působíky obklopené kameny jedné barvy. Hra končí ve chvíli, kdy se oba soupeři dohodnou, že již neexistují tahy, kterými by mohli změnit bodovou situaci, a nebo jeden ze soupeřů uzná, že již nemůže vyhrát, a vzdá se. Hráči se dohodnou na konci zahlášením pass. Hra končí, když zahrají pass oba hráči bezprostředně po sobě.

A.1.4 Ukázka partie

Partie se hraje obvykle na desce 19x19, ale jsou používané i menší desky 13x13 a 9x9, ale to je pro naše účely nepodstatné. Následuje ukázka konce hry na malé desce 9x9. Území bílého je označené bílými kolečky, území černého černými kolečky. Území "nikoho" je označené šedě. Na území bílého je jeden černý kámen, který zřejmě již nemá žádnou perspektivu a je také považován za zajatce, ačkoliv má nenulový počet svobod (je to tzv. mrtvý kámen). Bílý má navíc jednoho zajatce, kterého odebral z desky v průběhu hry. Tento kámen se původně vyskytoval na souřadnicích G-7.



Obrázek A.4: Ukázka hry. Výsledné skóre je černý 22 bodů území, bílý 23 bodů území (včetně E6) + 2 body za zajatce. Tedy černý 22 bodů, bílý 25. Bílý vyhrál o 3 body.

A.1.5 Závěrem

Ráda bych ještě zdůraznila, že v kontrastu se svými jednoduchými pravidly go vyniká svou náročností. Je to velice komplexní strategická hra (zatímco o šachách se tvrdí, že jsou hrou pouze taktickou), ve které hraje velkou roli jak schopnost předvídat a propočítávat tahy dopředu, tak intuice a hráčské zkušenosti.

A.2 Rejstřík goistických pojmů

- **Atari:** Situace, kdy má jedna skupina kamenů poslední svobodu a soupeř ji může příštím tahem dobrat.
- **Falešné oko:** Falešné oko je podobné normálnímu oku, ale platí pro něj, že v případě, kdy by soupeř obsadil všechny okolní svobody nějaké části skupiny, která k tomuto oku přiléhá, pak by tuto skupinu kamenů mohl dalším tahem zajmout a dané oko by přestalo existovat. Skupina, která sice má dvě oči, ale jedno z nich je falešné, je mrtvá skupina, protože toto falešné oko může být soupeřem zničeno.

- **Hrozba na ko:** V případě, že na desce vzniklo ko, je hráč, který právě nemůže zajmout daný kámen, nucen alespoň jednou zahrát jinam (Pravidlo č.3.). Pokud chce mít možnost ko vyhrát, pak musí zahrát takový tah, na který jeho soupeř odpoví. kdyby soupeř neodpověděl, tak by ztratil více bodů, než kolik by získal v případě výhry ko. Takovýto tah, na který soupeř určitě odpoví, se nazývá hrozbou na ko.

- **Joseki:** Sekvence tahů v zahájení partie po prvním přiblížení hráču (nejčastěji v rohu / boji o roh), která je alespoň v lokálním měřítku stejně výhodná pro oba hráče. Obvykle bývají joseki ustálené sekvence tahů, které se tak hrají již stovky let, ale se profesionálové vymýšlejí i nové.

- **Ko:** Situace, kdy je jeden kámen v atari, po jeho dobrání se ale do atari dostává soupeřův nový kámen a takto do nekonečna. Taková situace je ošetřena pravidlem č.3. Pokud chce hráč souboj o ko vyhrát, jsou potřeba "hrozby na ko". Ko končí, když jeden z hráčů neodpoví na hrozbu a zruší situaci ko např. zaplněním prázdného prusečíku.

- **Mrtvá skupina kamenů:** Skupina kamenů, která nemá, ani v ní není možné vytvořit, dvě oči. Takovou skupinu může soupeř zajmout kdykoliv. Protože nebývá nutné obsadit všechny svobody takovéto skupiny kamenů ihned, ponechává se na desce do konce hry (nebo do situace, kdy je její zajmutí nezbytné). Po konci hry se mrtvé kameny vyberou a počítají se po jednom bodu, jako by byly zajmuty ve hře.

- **Oko:** Alespoň jeden průsečík v území skupiny kamenů pevně ohraničený (tzn. ne "falešné oko"). Pokud má skupina kamenů alespoň dvě oči (mohou obsahovat i více průsečíků) nebo si je schopná dvě oči vytvořit i po tahu soupeře, pak o ní říkáme, že je živá.

- **Schody:** Situace, kdy je jedna skupina v atari, má jedinou možnost z tohoto atari uniknout jedním tahem, ale soupeř ji opět dalším tahem dostává do atari. V případě, že by hráči v této sekvenci pokračovali, vytvoří se na desce tvar, který skutečně připomíná schody. Ve hře se obvykle schody nehrají, ale je důležité, zda schody "fungují", či nikoliv, tzn. zda v případě, že by soupeři tuto sekvenci hráli, tak by útočící hráč unikající skupinu na konci zajmul, či zda by unikla.

- **Tesuji:** Sekvence tahů, které v lokálním měřítku mají plnit určitý účel s nejlepším možným výsledkem. V případě, že jsou splněny určité

podmínky, pak je i vždy úspěšná. Nejčastěji pomáhájí zajmout kámen nebo naopak s kamenem uniknout.

- **Živá skupina kamenů:** Skupina, která má alespoň dvě oči (viz "Oko"). Takovou skupinu nemůže soupeř žádným způsobem zajmout.

Příloha B

Uživatelská dokumentace

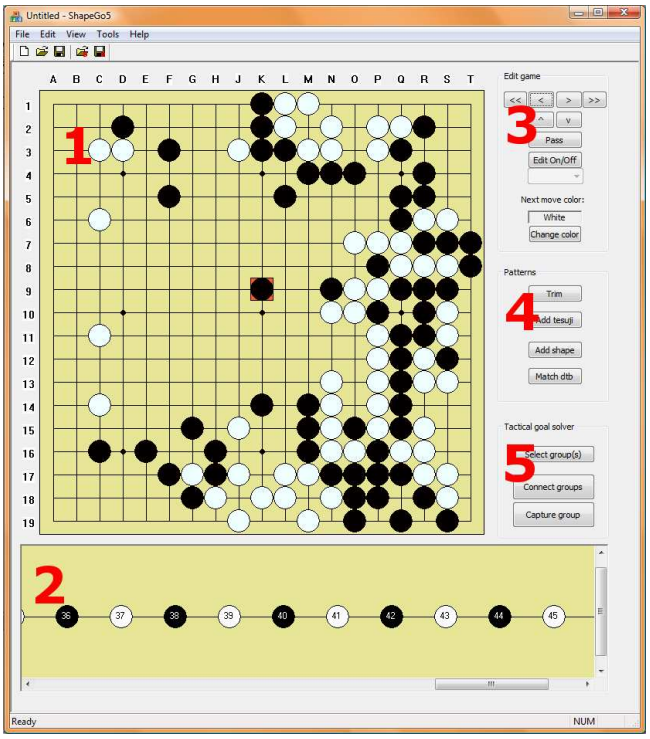
B.1 Úvod - o programu

Program ShapeGo je určen pro rozpoznávání tvarů a tesuji na desce a spouštění řešiče taktických cílů hry go založeného na rozpoznávání vzorů. Přiložená verze programu pravděpodobně nejvíce zaujme programátory, zajímající se o hru nebo naopak hráče go zajímající se o programovanání této hry.

Pro spuštění programu spusťte soubor ShapeGo.exe, který naleznete v kořenovém adresáři přiloženého CD.

B.2 Hlavní okno

Hlavní okno programu umožňuje prohlížení a editaci her včetně načítaní a ukládání souborů ve formátu sgf. Program dále zprostředkovává připravení situace pro řešič taktických cílů a rychlé vkládání tvarů do databáze vzorů.



1. Hrací deska - goban. Vykresluje momentální pozici na gobanu. Kliknutím na desku v běžném režimu odehráváte tahy. V režimu editování (tzn. po kliknutí na tlačítko "Edit" v sekci 3) vkládáte stále kameny jedné barvy. Tu je možné nastavit v combo boxu pod tlačítkem "Edit" opět v sekci 3. V režimu označování skupin (tzn. po kliknutí na tlačítko "Select groups" v sekci 5) klikáním na desku označujete skupiny kamenů stejné barvy, které jsou spojeny a opětovným kliknutím je odoznačujete.

2. Strom hry. V této oblasti se vykresluje strom hry se všemi tahy, které jste odehráli, nebo které byly nahrány ze souboru .sgf. Zobrazuje také očíslování tahu, to znamená pořadí tahu od začátku hry/editování. Kliknutím na kámen se zobrazí na gobanu situace, která tomuto kamenu odpovídá. Kameny, které jsou v tomto stromu vykresleny šedě, mají nějakou speciální vlastnost. Např. je u go zvykem v módu editování (viz. sekce 3) ukládat všechny editované kameny do jednoho vrcholu.

3. Ovládání editace. První tlačítka umožnují pohyb po stromu hry.

- "Dvojitá šipka doleva" - Posouvá ve stromu hry na úplný začátek, ještě před zahráním prvního kamene.
- "Šipka doleva" - Posouvá o jednu pozici zpět.
- "Šipka doprava" - Posouvá na první následující pozici.
- "Dvojitá šipka doprava" - Posouvá na konec sekvence prvních větvení od aktuální pozice.
- "Šipka nahoru" - Umožňuje pohyb mezi větvemi, ale pouze u větví, které mají stejného bezprostředního předchůdce. Posouvá se na vyšší větev
- "Šipka dolů" - Posouvá se na spodní větev.
- "Edit" - Zapíná a vypíná mód editace. Combobox pod tímto tlačítkem umožňuje měnit barvu přidávaného kamene.

V módu editace přidáváte všechny zahrané kameny do jednoho vrcholu. Můžete přidat na desku libovolný počet kamenů kterékoli barvy.

4. Tlačítka pro přidávání a odebírání vzorů, navíc i možnost trimování a spouštění matchování databáze tvarů.

   - "Trim" - Umožňuje otrimování (ořezání) stávajícího gobanu. Po odkliknutí se zobrazí dialog, do kterého vložíte souřadnice levého-horního a pravého-dolní rohu, které ještě mají být součástí vzniklé části gobanu
   - "Add Tesuji" - Přidání tesuji do databáze vzorů (viz Vzory, vytváření a editace)
   - "Add shape" - Přidání tvaru do databáze vzorů (viz Vzory, vytváření a editace)
   - "Match dtb" - Toto tlačítko vyvolá vyhledání všech matchů na gobanu pro hráče, který je právě na tahu. Odpovídající tahy i se stromem pokračování budou přidány do stromu hry a viditelné v sekci 2.

5. Spouštění řešiče taktických cílů, viz následující podkapitola.

B.3 Spouštění řešiče taktických cílů

Před spuštěním řešiče taktických cílů musíte mít nejprve uložené vzory v databázi vzorů. Při prvním zkoušení programu doporučujeme nejprve načíst již vytvořenou databázi přiloženou na CD.

Pro načtení databáze zvolte "Open database" v menu nebo ikonu pro otevírání se znakem červeného "D" v toolbar menu. Databázi naleznete na CD v souboru /Vzory/patternDatabase.sgf. Jestliže byla databáze načtena v pořádku, informuje Vás o tom dialog.

Před zvolením taktického cíle je potřeba označit skupinu/skupiny, na které se cíl vztahuje. Nejprve zvolte tlačítko "Select groups", které přepíná do módu označování skupin. Pozor, nemůžete vytvářet nové pozice gobanu, dokud toto tlačítko neoznačíte znovu. V módu označování skupin ihned uvidíte označenou skupinu po kliknutí na libovolný kámen.

Pro taktické cíle musíte označit odpovídající skupiny:

- pro cíl "Connect groups" - Spojení skupin - označíte právě dvě skupiny stejné barvy, které si přejete spojit.
- Pro cíl "Capture group" - Zajmutí kamene - označíte právě jednu skupinu (i jednoprvkovou), kterou si přejete zajmout.

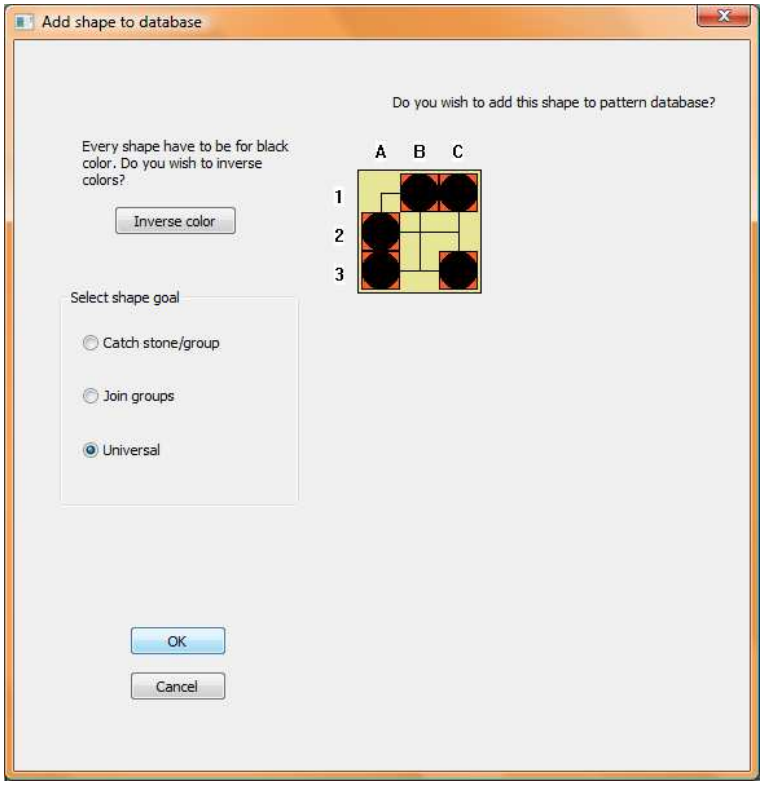
Po označení skupin stačí zvolit druh taktického cíle. To znamená buď tlačitko "Connect groups" nebo "Capture group".

B.4 Vzory a vytváření

Vzory dělíme na dva typy: Tvary a tesuji. Tvar je pouze určitý tvar na desce (např. koňský skok, bambusové spojení, stůl), tesuji je tvar na desce, který bude vyhledáván, a strom správných reakcí (tahů) na situaci za oba hráče (např. síť).

B.4.1 Přidávání tvarů

Pro vytvoření tvarů ze situace na desce v hlavním okně můžete použít tlačítko "Add shape". Objeví se trimovací dialog, v něm vyberte levý horní roh a pravý spodní roh obdelníku, který bude uložen jako nový tvar. Po potvrzení ořezání uvidíte v novém okně náhled vzoru, výběr z taktických cílů a možnost inverze barev (následující obr.).



Obrázek B.1: Dialog Add shape.

Tvary jsou vždy ukládané tak, že hráč na tahu je reprezentován černou barvou (černé kameny tvoří vzor) a bílé kameny, náležející "protihráči" jsou pouze součástí vzoru. Proto v případě opačného zbarvení použijte možnost inverze barev.

Výběrem taktického cíle určujete cíl, při jehož plnění bude tento nový vzor použit. Např tvar bambusové spojení se používá při spojování skupin, takže taktický cíl by byl Connect groups. Z taktických cílů můžete zvolit právě jeden. Pokud se jedná o univerzální tvar, případně Vám nevyhovuje žádna z možností, zvolte "Universal".

Součástí tvaru budou i veškeré prázdné průsečíky, tzn. i případné prázdné okraje, které neořežete pomocí trimování.

Po kliknutí na OK v náhledu je tvar uložen do databáze. Přejete-li si databázi uložit persistentně do souboru, zvolte "Save database" v hlavním menu nebo nástrojové liště.

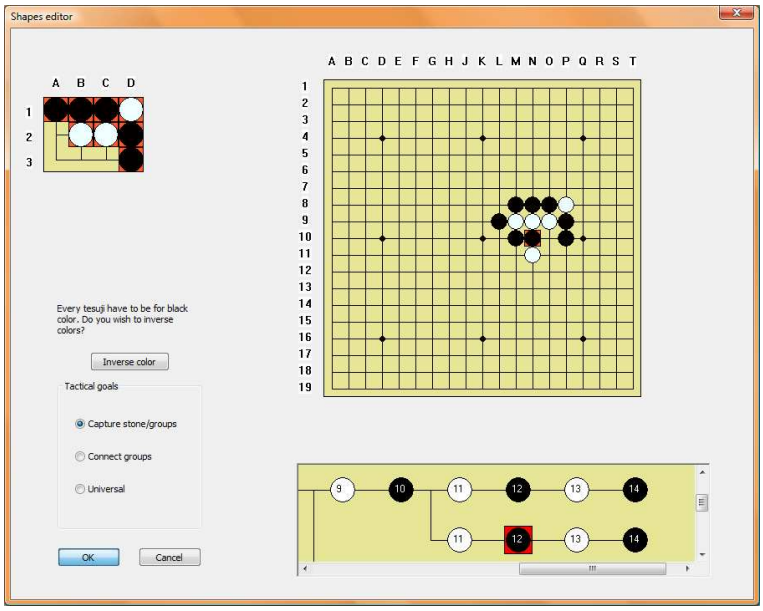
B.4.2 Přidávání Tesuji

Pro rychlé přidávání tesuji slouží tlačítko "Add tesuji". Zde je potřeba opatrnost. Jako tvar tesuji se přidá momentální situace na desce. Jako strom tahů (odpovědí na tesuji) bude uložen strom tahů, které vidíte ve vykresleném stromě hry za označeným aktuálním vrcholem.

Po kliknutí na tlačítko "Add Tesuji" se opět otevře trimovací dialog, ve kterém zvolíte levý horní roh a pravý dolní roh pro ořezání na odpovídající tvar. V tomto případě mohou být tahy, které jsou uložené ve stromě, i mimo (ořezání) ukládaný tvar.

Po potvrzení volby trimování se opět otevře okno náhledu, ale rozdílné od okna pro tvar (obr.). Tvar, který bude uložen do databáze a tedy i vyhledáván na desce, je v levém horním rohu. V pravé části okna máte možnost prohlížení stromu odpovědí. V levé části okna máte stejné možnosti, jako při přídávání tvarů. Zvolení taktického cíle a invertování barev.

Po potvrzení náhledu se tesuji uloží do databáze.



Obrázek B.2: Dialog Add tesuji.

B.5 Databáze vzorů

Databáze vzorů obsahuje všechny ukládané vzory a to buď ty, které byly ukládány za běhu programu, nebo vzory načtené ze souboru pomocí tlačítka File-¿Open database.

V případě, že si chcete prohlédnout databází vzorů, zvolte File-¿View database. Databáze bude načtena a zobrazena coby různé situace na desce včetně uložených následujících tahů.

Na přiloženém CD můžete nalézt předpřipravenou databázi vzorů v adresáři /Vzory v souboru patternDatabase.sgf.

Příloha C

Programátorská dokumentace

C.1 Úvod

Program ShapeGo je výsledkem práce na ročníkovém projektu a následně bakalářské práce. Program je určen pro řešení taktických cílů pro hru go, který je založen na rozpoznávání tvarů na desce. ShapeGo je napsán pro platformu Windows s plánem jej v budoucnu zobecnit i pro operační systém UNIX. Programovací jazyk jsme zvolili C++ s použitím knihoven STL a MFC. Program je psán podle standardů objektového programování. Tzn. až na vyjimky je veškerý kód ve třídách.

Vygenerovaná programátorská dokumentace (v anglickém jazyce) pomocí doxygenu je přílohou samotného programu v CD/doc.

Důraz byl kladen na rozšiřitelnost a použitelnost různých implementací, čehož se dosáhlo vytvořením interfacu pro každou důležitou datovou strukturu. V programu se můžete snadno orientovat i díky dodržení jmenné konvence, kdy jméno třídy vždy obsahuje k čemu je určena a buď I, jestliže se jedná o interface, nebo zkratku, která vystihuje, co je na dané implementaci interfacu důležité. Např.: CGobanI (interface pro uložení gobanu) a CGobanGroups (implementace, pamatující si informace o skupinách), CGoTreeI (interface pro strom hry) a CGoTreeVec (implementace stromu hry pomocí šablonové třídy vector) apod.

C.2 Reprezentace hry

C.2.1 Základní problémy

Odehranou hru je potřeba reprezentovat tak, aby bylo možné se kdykoliv vrátit do určité situace. Tato reprezentace je nutná jak v případě, kdy načítáme hru ze souboru .sgf, tak i při ukládání tesuji a kdykoliv, kdy si chceme prohlížet odehrané tahy. Ovšem hlavní je reprezentace desky, která se stará o přidávání nových kamenů a tedy o kontrolu pravidel go.

C.2.2 Goban - hrací plocha

Pro goban je vytvořen interface CGobanInterface. Jeho nejdůležitější metodou je metoda MakeMove, která musí kontrolovat pravidla hry a v případě jejich splnění odehrát jeden tah.
CGobanGroups: Třída implementující interface CGobanInterface. Goban je uložen jako matice, kde každý prvek je ukazatelem na skupinu, do které patří, případně NULL pointer pro prázdnou pozici. Každá skupina obsahuje informaci o počtu kamenů, které do ní patří, počtu svobod (tzn. počet svobod celé skupiny) a barvu kamenů ve skupině. Minimální velikost skupiny je jeden kámen. Skupiny se spojují při přidávání nového kamene, který sousedí se dvěmi skupinami stejné barvy. Při spojování je potřeba přepočítat počet svobod nové větší skupiny.

Tato reprezentace umožňuje v konstantním čase zjišťování informací o skupině kamenů. Je podstatně rychlejší přidávání kamenů. (v čase O(n+m)), kde n je počet kamenů spojovaných skupin v případě spojování a m počet zajmutých kamenů.)

V programu naleznete i nepoužívanou implementaci interfacu CGobanInterface, třídu CGobanMatrix. Tato reprezentace sice měla rychlejší odebírání kamenů, ale jinak pomalejší všechny ostatní operace a proto jsme ji dále nevyvýjeli.

C.2.3 Strom hry

Pro strom hry je vytvořen interface CGoTreeI, obsahující interface pro vrchol stromu CGoTreeNodeI
Nejdůležitější metody **CGoTreeI:**

- AddStone - Vytvoření a přidání vrcholu s odehraným tahem.

- AddEditStones - Vytvoření a přidání vrcholu s kamenem, který byl zahrán během editace gobanu (editování desky je zvláštní operace a kameny přidané během editování je zvykem ukládat do jednoho vrcholu).

- MergeTrees(CGoTreeI * toTree, CGoTreeI * fromTree) - Jedna z nejvyužívanějších metod při vytváření databáze. Používá se ve chvíli, se do databáze ukládaji dva stejné heshe. V takovém případě se mergují (spojují stromy). Obdobně při prohledávání gobanu se spojují stromy nalezených tvarů. Tato metoda merguje fromTree do toTree. MergeTrees volá rekurzivní metodu MergeNodes na oba rooty. Od rootů projde všechny syny a v případě, že nalezne dva, které mají stejné kameny, zmerguje tyto dva nody opět v metodě MergeNodes(). Jestliže ve fromTree oxistují nody, které se nezmergují, pak se překopírují na odpovídající místa v toTree.

- GetStonesToPrev - Tato metoda je nutná pro posunutí se při prohlížení hry o tah zpět. Vrací seznam kamenů odehraných do předchozího tahu (včetně).

- GetNext - Metoda vracející kameny na i-tém synovy aktuálního vrcholu.

CGoTreeVec: Třída implementující interface CGoTreeI. Obsahuje vnitřní třídu pro vrchol CNode, která implementuje interface CGoTreeNodeI. Tato třída implementuje n-ární strom hry pomocí template vectoru. Vector je využit v CNode pro ukládání editovaných kamenů i následovníků (dětí) každého vrcholu. Každý vrchol má navíc odkaz na svého otce pro rychlé posouvání stromem nazpět.

C.3 Tvary a tesuji

Základní interface pro libovolný vzor je CPatternI. Od něj poděděné interfacy CShapeI a CTesujiI. CShapeI je interface pro jednoduchý tvar na desce. CTesujiI musí obsahovat i strom navazujících tahů včetně odpovědí soupeře. Nejdůležitější metody **CPatternI:**

- GetHashPattern() - vracející mapu provázaných hashů tvarů, které se budou vyhledávat, a stromů tahů.

- CGobanPartValue GetGobanPattern() - Vrací tvar vzoru tak, jak byl zadán uživatelem. (v této podobě se mj. ukládá do databáze CPatternFileDb)
- CGoTreeI * GetTree() - Vrací strom tahů. V případě CShapeI vrací NULL (tvar nemá obsahovat strom tahů (viz začátek kapitoly), ale stram tahů pro tvar se vytváří při získávání vyhledávacích tvarů (match patternů).

CShapeMatrix: Třída implementující interface CShapeI. Reprezentace tvaru je pomocí matice ze StoneValue (=enum hodnot, které se mohou vyskytovat na gobanu.)
CTesujiGroups: Třída implementující interface CTesujiI. Tvar na desce je reprezentován třídou CGobanGroups. Strom tahů je uložen v CGoTreeVec.

C.4 Hashování

Interface pro hashování je CHashPatternI. Nejdůležitější požadavek, který musí každá implementace splňovat, je porovnávání - metoda LessThen, vracející stejnou hodnotu, jako by vracel operátor porovnání. Operátor porovnání je použit v implementaci třídy map z STL. Dále je nutný konstruktor s CGobanPartValue * a CGobanI *. Všechny hashe se ukládaji do databáze vzorů pomocí JAK SE RIKA TOMUHLE TYPU TRIDY? třídy CHashHelper, která v operátoru porovnání volá metodu hashe CHashPatterI::LessThen().
CHashMatrix: Jednoduché uložení tvaru v matici o třech možných hodnotách (černý, bílý, prázdný). Výraz hash je zde použit spíše pro dodržení jmenné konvence tříd, ačkoliv jeho použití jde proti základní definici výrazu hash.
CHashZobrist: Implementace Zobristova hashování. Při prvním průchodu privátní metody GetHash dochází k inicializaci hashe, kdy se vytvoří statický hash klíč pro celý goban a každou možnou pozici.

C.5 Databáze vzorů

V programu jsou implementovány dvě třídy pro databáze vzorů. Obě třídy jsou singltony.

CPatterFileDb: Databáze určena k ukládání do souboru a načítání celé databáze ze souboru. Obsahuje mapu vzorů a ukazatelů na odpovídající stromy tahů. Vzoru jsou uložené třídou CGobanPartValue, strom tahů obdobně jako v CPatternDb v CGoTreeI : CGoTreeVec. V patternFileDb jsou vzory uloženy právě tak, je je zadával uživatel. To znamená, že tvar je ve formátu správného tvaru, kterého řešič taktických cílů má dosáhnout. Ukazatel na strom tahů u tvarů je NULL. Tesuji se ukládá ve formátu, který se na desce vyhledává, a se stromem tahů zadaných uživatelem.
CPatterDb: Implementuje hashovací tabulku pro řešič taktických cílů pomocí šablonové třídy map.. Pří přidávání vzorů získává od přidávaného vzoru seznam hashů, které se budou vyhledávat na gobanu, a ukazatel na odpovídající stromy tahů. Hash vzoru je vždy interface CHashPatternI, v odevzdávané verzy programu implementován třídou CHashZobrist. Strom tahů je CGoTreeI implementovaném jako CGoTreeVec.

- AddTesuji(CPatternI *)
- AddShape(CPatternI *)
- InitFromFIleDb() - Inicializuje databázi z databáze ve formátu CPatternFileDb.

C.6 Taktické cíle

Taktický cíl reprezentují třídy CCaptureGoal a CConnectGoal, které implementují interface CTacticalGoalI. Nejdůležitější metoda je IsFullfilled, která určuje, zda je v aktuální situaci na desce splněn daný taktický cíl.

C.7 Ukládání do souborů

Ukládání do souboru je použito při ukládání i načítání her uložených ve formátu .sgf, ale i ukládání a načítání databáze vzorů v lehce upraveném formátu .sgf, který je ale stále s drobnými chybami kompatibilní s jinými .sgf prohlížeči.

Příloha D

Obsah CD

Možnosti přiloženého CD:

- Program ShapeGo5 spustíte souborem ShapeGo(.exe) v kořenovém adresáři CD.
- Dokumentace je v adresáři /doc. Zde se nachází programátorská dokumentace ve formátu html vygenerovaná pomocí doxygenu.
- Projekt pro Visual studio 2005 naleznete v adresáři VSProject/. Zdrojové kódy se nachází v adresáři /VSProject/ShapeGo5/ShapeGo5. Pro spuštění editace projektu ve Visual studiu 2005 spusťtě soubor ShapeGo5(.sln) v adresáři /VSProject/ShapeGo5. Projekt by měl být kompatibilní s Visual studiem 2005 a mladším, u spouštění ve Visual studiu 2008 může být programem Visual studio požadována konverze projektu.
- Bakalářská práce v elektronické podobě ve formátu PDF je uložena v kořenovém adresáři CD v souboru bcPrace.pdf. Součástí bakalářské práce jsou i přílohy Stručný úvod do hry go, Uživatelská dokumentace a Programátorská dokumentace.
- Zdrojový kód bakalářské práce v jazyce LaTeXse nachází v adresáři /Tex.
- Připravené databáze vzorů pro načtení programem ShapeGo jsou uložené v adresáři /Vzory

- Soubor Readme(.txt) v kořenovém adresáři obsahuje informace z této kapitoly.