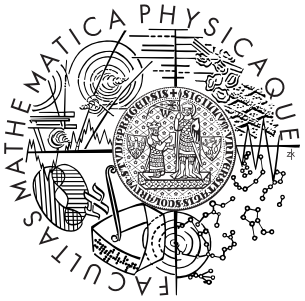
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

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

2010

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

Prohlašuji, že jsem svou bakalářskou práci napsala 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 (zajmutí kamenů, spojení skupin). Tento řešič je založen na rozpoznávání tvarů kamenů na desce a výběru tahů podle předdefinované databáze 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 a příklady pro použití.
Klíčová slova: umělá inteligence, go, vzory, 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 (capturing stones, connecting groups). The solver is based on recognizing the shapes of stones on the desk and choosing the moves from predefined database of 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 and examples of use.
Keywords: Artificial Intelligence, go, patterns, tesuji, tactical goals

Kapitola 1

Úvod do problematiky

1.1 Hra go a programové zpracování

Hra go je považována za nejstarší a nejkomplexnější strategickou deskovou hru na světě [7]. Go je zároveň poslední klasická desková hra, pro kterou stále neexistuje počítačový hráč, který by byl schopen v rovné partii porazit špičkového lidského hráče. Proto si tato hra získala v posledních desetiletích svou náročností i odborníky ze světa informatiky. Otázkou vývoje programů hrajících go na vysoké úrovni se zabývají profesionální týmy po celém světě. Momentálně nejúspěšnějším softwarem je MoGo [13], založené na pseudonáhodném výběru tahů pomocí metody Monte-Carlo a prohledávání stavového prostoru algoritmem UCT [13]. MoGo již je schopné porazit profesionální hráče, ale pouze v případě, že má vysoký hendikep[1] a je spuštěn na superpočítači.

Go má vysoký koeficient větvení při zkoušení tahů, a proto není možné použít jednoduché metody prohledávání všech možností [12]. Ani metody úspěšné v jiných deskových hrách neuspěly. Go bývá často srovnáváno s šachy. Pokud bychom tyto dvě hry porovnali, pak najdeme tři největší rozdíly. Šachové algoritmy se v nejhorším případě větví do 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. Naproti tomu v go je první tah možné umístit na 361 pozic, v průměru je pro každý tah 200 možností a hra má obvykle 200 až 300 tahů, což dává pří-

[1] Hendikep je zvýhodnění černého hráče. Podrobněji viz A.2 - Rejstřík goistických pojmů

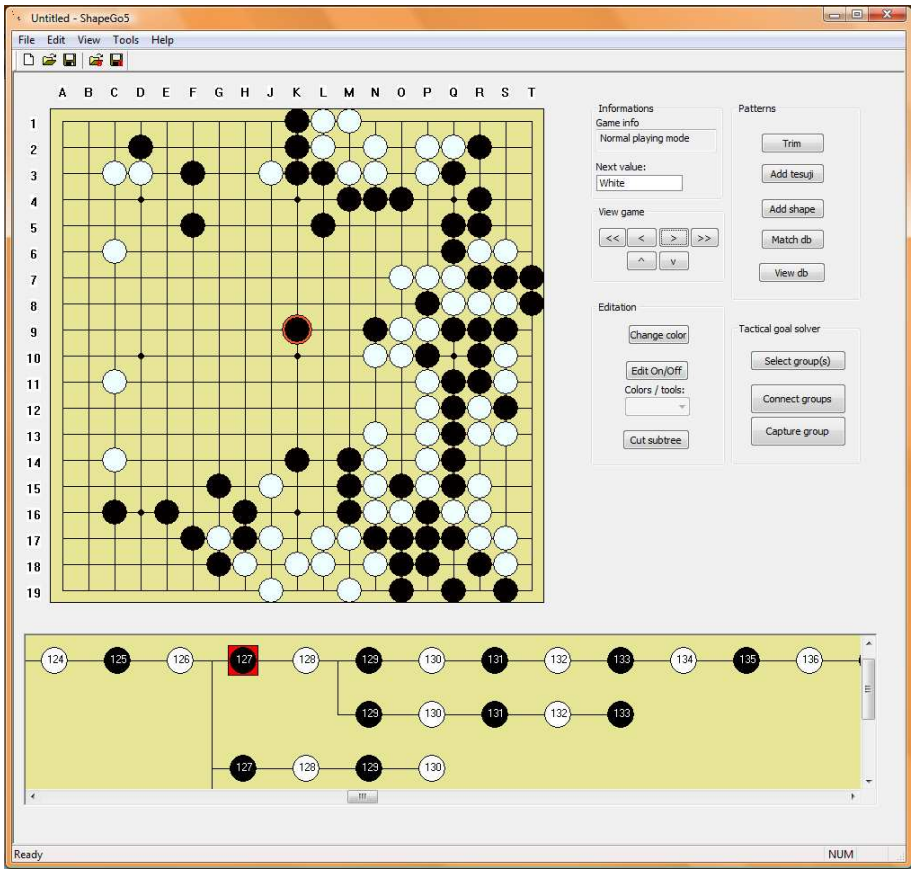
liš vysoký počet možností při prohledávání do hloubky. Výběr dobré statické ohodnocovací funkce prakticky není možný, protože tahy, které se v lokálním měřítku zdají zbytečné či nevýhodné, mohou mnohem později výrazně ovlivnit směr hry v neprospěch protihráče. Připravená strategie (plán hry), která by pomohla výrazně snížit koeficient větvení, také není vhodná. Připravený plán je v go spíše přítěží, protože je důležitá flexibilita do té míry, kdy je nutné po jednom tahu zcela změnit strategii. Proto je tak důležité zkoumat postupy specifické přímo pro hru go, mezi něž patří i rozpoznávání tvarů na desce a hraní podle pravidel, která jsou podobná způsobu rozhodování lidských hráčů.

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 komplexního herního algoritmu je otázkou spíše diplomové práce ve velkém týmu, a proto jsme se rozhodli 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. Dalším příkladem jsou situace, kdy je bezpodmínečně nutné zajmout nějaký soupeřův kámen nebo v koncové fázi hry, kdy je nutné ovládat nejlepší způsoby zmenšování soupeřova území. Pro svou práci jsem tedy zvolila plnění konkrétních taktických cílů.

Jednou z málo probádaných oblastí v počítačovém go je rozhodování založené výhradně na základě rozpoznávání tvarů na desce. Hledání známých tvarů výrazně vylepšilo i zmiňovaný program MoGo [13], ale v jednoduché podobně se nachází v každém významnějším počítačovém hráči, jako například v programu GNU GO [4]. Nejčastěji se rozpoznávání tvarů využívá v zahajovací fázi hry [8] (tzv. joseki[2]. Vzhledem k tomu, že je rozpoznávání tvarů pro hlavní výběr tahů málo prozkoumáno, je tato práce založena na výběru tahů a hraní podle předdefinované sady vzorů.

[2]Pojmem joseki je označována sekvence tahů za útočníka i obránce, která v určité situaci na desce spňuje konkrétní lokální cíl. Podrobněji viz A.2



1.2 O programu ShapeGo

1.2.1 Cíle této práce

V této práci jsme si dali za cíl vytvořit program, který bude schopen řešit taktické cíle hry go (zajmutí dané skupiny kamenů, spojení dvou skupin kamenů) na základě předpřipravených vzorů. Hlavním úkolem je vymyslet a vytvořit 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. Dalším úkolem bylo vytvoření algoritmu pro hledání nejvhodnějšího tahu na bázi rozpoznávání vzorů kamenů na desce (gobanu) a hraní podle předdefinovaných nejlepších odpovědí pro daný vzor. V neposlední řadě bylo potřeba zaměřit se na vhodné uživatelské rozhraní, pomocí kterého bude uživatel schopen definovat nové vzory, edito-

vat databázi persistentně uložených tvarů a i spouštět řešič taktických cílů. Výsledkem těchto požadavků je program ShapeGo, který budeme podrobně analyzovat v dalších kapitolách.

1.2.2 Programová omezení

Existují univerzální řešiče taktických cílů a stejně tak programy, které přímo hrají go. Náš program byl ale koncipován jako řešič konkrétních taktických cílů s přímým využitím vzorů. Nejedná se tedy o univerzálního počítačového hráče a není možné očekávat, ze program bude hrát bez dobře propracované databáze vzorů. Tzn. program nebude schopen řešit úlohy bez odpovídajících vzorů v databázi.

1.3 Přehled kapitol

Kapitola 2 se zabývá přehledem hlavních problémů, které bylo potřeba v této práci analyzovat, a navržením jejich řešení. Mezi nejdůležitější části patří implementace vzorů jak pro použití v řešiči taktických cílů, tak jejich persistentní uložení. Další významnou částí je vyhledávání vzorů na gobanu, které je zásadním zpomalujícím faktorem. A poslední částí hlavní analýzy je určení splnění taktického cíle, které se liší z programového hlediska a z pohledu hráče go.

V kapitole 3 jsou rozebrány nejpodstatnější datové struktury pro reprezentaci základních programových prvků, struktura implementace databáze i jednotlivých vzorů a použité hashovací metody pro ukládání vyhledávácích tvarů[3]. Zběžně jsou popsána i specifika struktury gobanu.

V kapitole 4 se zaměřujeme na nejvýznamnější algoritmy. Je zde podrobně rozebrána práce s databází vzorů, porovnávání a vyhledávání vzorů na gobanu, algoritmus řešiče taktických cílů včetně detekce splněnosti taktického cíle.

V závěru je shrnutí dosažené práce včetně zamyšlení se nad možnými vylepšeními.

Součástí tohoto textu jsou i přílohy *Stručný úvod do hry go* včetně rejstříku použitých goistických pojmů, *Uživatelská dokumentace*, *Programátorská dokumentace* a *Obsah přiloženého DVD*.

[3]Vyhledávací tvar je přímo rozložení kamenů, které bude vyhledáváno na desce. Podrobně viz 2.2

Kapitola 2

Analýza hlavních problémů

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. Hlavním cílem tedy bylo navrhnout implementace podstatných částí programu s ohledem na co nejefektivnější využití databáze vzorů při špouštění řešiče taktických cílů. Následující podkapitoly pojednávají o částech programu, na které byl kladen největší důraz.

2.1 Ukládání vzorů

Návrh struktury vzorů hry go byl jedním z hlavních úkolů. Implementace jednotlivých vzorů musí zachytit specifika hry go. Databáze těchto vzorů musí umožňovat použití při řešení taktických cílů. Navíc je nutné umožnit uživateli vkládání nových vzorů, prohlížení databáze a její základní editaci.

2.1.1 Jednotlivé vzory

Nejprve jsme se zaměřili na reprezentaci jednotlivých vzorů. V počítačových hráčích, které využívají rozpoznávání vzorů, bývá obvyklé reprezentovat vzor jako rozložení kamenů na malé části desky pevně dané velikost jako např. v [2]. Tato obyčejná reprezentace jednoho typu tvaru však byla pro naše účely příliš jednoduchá. Po analýze specifik hry go včetně toho, jak pojem vzory chápou lidští hráči, jsme všechny vzory rozdělili do dvou typů

- tvary (shapes) a tesuji[1], které pokrývají všechny možnosti reprezentace potřebné pro naše účely. Takovéto rozdělení je navíc intuitivní pro hráče go, kteří by databázi tvarů vytvářeli.

Vzory bývají dále rozděleny na útočné, obranné, případně na další kategorie jako např. v implementaci GNU GO [5]. Takovéto rozdělení bylo pro účely řešiče taktických cílů nedostatečné. Rozhodli jsme se tedy všechny vzory dělit přímo mezi konkrétní taktické cíle, které mají dané vzory plnit.

2.1.2 Databáze vzorů

Vzory bylo nutné reprezentovat v jednotné struktuře - databázi. Byla navíc potřebná rozdílná reprezentace vzorů pro účely persistentního ukládání, prohlížení a editace uživatelem a pro řešič taktických cílů. Každý z těchto účelů má jíné požedavky a podle nich byly vytvářeny jednotlivé struktury.

Primární databáze si jednoduše ukládá do seznamu jednotlivé vzory tak, jak jsou popsány v sekci 3.1. Z této hlavní databáze jsou poté odvozeny struktury pro další použití.

Nejpodstatnější byla reprezentace vzorů pro řešič taktických cílů. Pro tuto strukturu byla nejdůležitější rychlost hledání vzorů podle situace na gobanu, tedy rychlé prohledávání. Tato reprezentace musela tedy umožňovat nalezení odpovídajícího tahu a zjištění, zda je použitím nalezeného vzoru daný taktický cíl splněn. Při podrobnějším rozboru typů vzorů naší reprezentace je zřejmé, že vkládaný vzor uživatelem nemusí být vždy totožný s tvarem, který bude vyhledáván na gobanu. Navíc bude tvarů pro vyhledávání na gobanu mnohonásobně více než vkládaných vzorů. Podrobně je tato reprezentace zkoumána v sekci 2.2

Pro správu vzorů uživatelem a tedy i zobrazení v uživatelském prostředí bylo vhodné vytvořit druhou variantu implmentace databáze. Tato varianta využívá používaný způsob reprezentace her ve stromové struktuře. Stejná strukturu také využíváme pro persistentní ukládání do souborů ve formátu SGF[2] s pomocnými údaji.

[1] Oba tyto typy vzorů jsou podrobně rozebrány v kapitole3.1. Tesuji je navíc goistický pojem, který je popsán v sekci 3.1 - Rejstřík goistických pojmů

[2] SGF je Smart Game Format používaný pro reprezentaci hry go. Jeho strukturu naleznete ve specifikaci [14].

2.2 Vyhledávání vzorů na gobanu

Rychlost celého řešiče taktických cílů závisí na rychlosti vyhledávání tvarů na desce a porovnávání s databází vzorů. V základní verzi je potřeba porovnat všechny obdélníky na desce s tvary stejné velikosti, které jsou uloženy v databázi. Je zřejmé, že zásadním faktorem zpomalení by měla být velikost databáze. Ta navíc není rovna počtu vkládaných vzorů uživatelem, ale počtu vyhledávacích tvarů[3]. Každý vkládáný vzor je totiž nutné vyhledávat i ve všech jeho rotacích o 90°, reflexích po diagonále a inverzi barev. (viz kapitoly 4.2 a 3.1).

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 různých 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[4], t je počet tesuji a 8 je počet rotací. Tedy velikost odpovídá $O(s*b+t)$. Oproti očekávání ovlivnila velikost databáze tvarů výslednou rychlost řešiče taktických cílů pouze logaritmicky, tedy zanedbatelně v porovnání s ostatními úkony, vykonávanými při vyhledávání tvarů na desce.

V naší základní verzi matchování databáze s pozicí na gobanu probíhá podle následujícího algoritmu. Celková časová složitost porovnání celé desky s celou databází je složena z časových složitostí káždého z těchto řádků. Výsledkem je vytvořená stromová struktura tahů, které odpovídají vzorům nalezeným na celé desce. Pro účely řešiče taktických cílů je ovšem tento algoritmus upraven heuristikou, která je popsána v sekci 4.3.

```
1.  Pro každou velikost 'v' vzoru z databáze
2.     Pro každou pozici na desce
3.        Vytvoř hash pro aktuální část gobanu velikosti 'v';
4.        if ( hash se nachází v databázi )
5.           Spoj strom tahů tohoto hashe s výsledným stromem;
```

[3]Vyhledávací tvar je přímo takové rozložení kamenů na desce, které se bude na gobanu vyhledávat. Může se lišit od tvaru, jak jej zadal uživatel. Podrobněji viz 4.1

[4]Počet černých kamenů ve vzoru určuje počet vyhledávacích tvarů. Podrobně vysvětleno v sekci 4.1

Při použití značek z předchozího textu dostáváme následující fakta:

- Řádek 1 odpovídá s tzn. počtu všech různých velikostí uložených vzorů. Tento údaj hraje velice zásadní roli.
- Řádek 2 odpovídá $n*m$, tedy velikosti gobanu.
- Složitost řádku 3 odpovídá složitosti vytvoření 'vyhledávacího klíče' pro použitou implementace hashování.
- Řádek 4 představuje vyhledání vzoru v databázi, v naší implementaci s logaritmickou složitostí vůči velikosti databáze.
- Operace slučování stromů, tedy řádek 5, se ukázal jako nejpomalejší část celého procesu. I při použití velice propracovaného mechanismu je v nejhorším čase lineární vzhledem k velikostem těchto dvou stromů.

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, o počtu kamenů skupiny a především o 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

Datové struktury

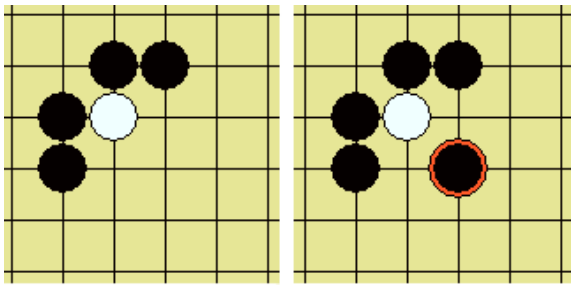
Návrh propracovaných datových struktur byl zásadním klíčem k úspěšnému vytvoření řešiče taktických cílů. Analýza možných návrhů včetně rozboru, na které vlastnosti bylo potřeba se zaměřit, je popsána výše v sekci 2. V této kapitole jsou podrobně rozebrány zvláštnosti a výhody i nevýhody implementovaného datového modelu. Mezi nejvýznamnější patří implementace jednotlivých vzorů a uložení databáze. Také je v této kapitole stručně rozebrána reprezentace gobanu, která zásadním způsobem ovlivnila rychlost určování splněnosti taktických cílů.

3.1 Tvary a tesuji

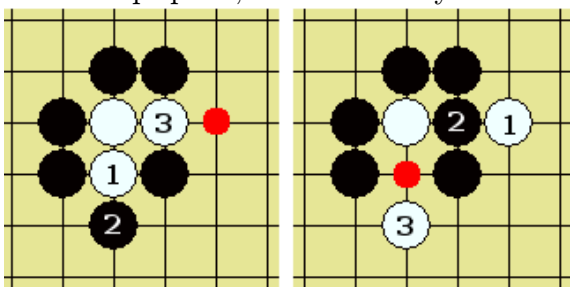
S ohledem na zvláštnosti hry go jsme se rozhodli vzory rozdělit do dvou typů. Tvar (neboli shape) je jednoduché, přesně definované lokální rozložení kamenů na desce, které má i svou přesně určenou velikost (tzn. počet políček, které obsahuje). Toto rozmístění kamenů už plní samo o sobě určitý taktický cíl. Hráč se tedy snaží tohoto tvaru dosáhnout.

Tesuji je goistický pojem (viz kapitola A.2), který popisuje tahy pro dosažení konkrétního lokálního cíle (např. zajmutí kamenů) včetně seznamu nejčastějších reakcí soupeře a vhodných odpovědí na ně. Takovéto posloupnosti tahů jsou obvykle reprezentovány stromovou strukturou. Použití tesuji je vždy podmíněno určitou situací na desce [3]. Tato výchozí situace je součástí definice tesuji a je reprezentována obdobně jako shape s tím rozdílem, že samotná tato situace je již vyhledávacím tvarem. Příkladem tesuji je tzv. síť. Ta je znázorněna na obr. 3.1, ukázky navazujících tahů jsou na obr. 3.1.

Všechny vzory je vhodné ukládat jednotně v celé databázi za jednu barvu



Obrázek 3.1: Ukázka známého tesuji, tzv. sítě. Obrázek vlevo reprezentuje tvar, který bude vyhledáván na desce. Na obrázku vpravo je znázorněn správný tah za černého v případě, že chceme bílý kámen zajmout.



Obrázek 3.2: Součástí tesuji v naší implementaci jsou i další tahy, které mohou na první tah navazovat. V tomto obrázku jsou ukázány dvě další možné posloupnosti tahů, kterými se snaží soupeř zachránit svůj bílý kámen. Pořadí tahů je znázorněno očíslováním. Červeně je zde zobrazen poslední tah, kterým černý dobírá bílý kámen/bílou skupinu kamenů.

(za jednoho hráče). My jsme se rozhodli pro reprezentaci vždy za černého hráče (tzn. první tah v tesuji je za černého hráče). Při vyhledávání na gobanu poté stačí barvy invertovat.

3.2 Databáze vzorů

Vzory potřebujeme ukládat do databáze pro dva různé účely použití: pro potřeby spouštění řešiče taktických cílů a pro zobrazování a editaci uživatelem. Proto jsme se rozhodli pro dvě zcela odlišné implementace databáze.

Hlavní a objemnější databáze je určena výhradně pro použití při vyhledávání vzorů na gobanu v průběhu řešení taktických cílů. V této databázi jsou

uložené tvary ve formátu, v jakém se budou vyhledávat na hrací desce (podrobněji viz kapitola 4.1). 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.

3.2.1 Databáze pro řešič taktických cílů

Po analýze možných řešení jsme zvolili uložení hlavní databáze v podobě hashovací tabulky, což umožňuje nejefektivnější vyhledávání vzorů na gobanu. Vyhledávací klíč v této tabulce je hash vzoru, podle kterého bude vzor vyhledáván na desce[1]. Hodnota pro daný klíč (hash) v tabulce 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.

3.2.2 Databáze pro persistentní ukládání

U jednoduché implementace databáze pro účely persistentního ukládání a editování uživatelem jsme se snažili o zachování základních konceptů, na které jsou goisti zvyklí, např. z programu CGoban [9]. Toho jsme dosáhli pro hráče go intuitivní reprezentací ve stejné stromové struktuře, která se využívá pro zaznamenávání odehraných tahů ve hře. Pro tesuji toto znamená, že základ vzoru (tzn. situace, za které se dané tesuji používá) je uložen celý v jednom vrcholu stromu, na který navazují podstromy posloupností tahů. Tento vrchol je přímým potomkem kořenového vrcholu stromu. Shape je obdobně celý uložen v jednom vrcholu ve stejné hloubce stromu jako základ pro tesuji, ale nemá žádné potomky.

Pro persistentní ukládání jsme zvolili formát založený na SGF. V souladu s oficiální definicí [14] tohoto formátu jsme si nadefinovali nové značky, které ukládají informace o vzoru (velikost základního vzoru a taktický cíl, pro který je vzor určen).

[1] Více viz kapitola 3.3

3.3 Hashování vzorů

Porovnání pozice na desce a vzoru, který na desce hledáme, je nejčastěji prováděná operace při spouštění řešiče taktických cílů, a proto bylo nutné, aby bylo co nejefektivnější . Po analýze 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.

3.3.1 Jednoduchá matice

Matice obsahuje hodnoty "černý", "bílý" a "prázdné políčko". Původně byla implementována pouze k testování časové složitosti. Porovnání dvou matic je v lineárním čase, avšak při vyhledávání tvarů na velké oblasti desky se může k Zobristovu hashování blížit, a proto jej ponecháváme v implementaci pro porovnání s dalšími metodami.

3.3.2 Zobristovo hashování

Zobristovo hashování popsané v [1] je používanou metodou hashování, která byla určena k rozpoznávání vzorů ve hře go a v šachách. V obou těchto hrách se tato metoda velice osvědčila a je stále využívaná v programech hrajících go, např. GNU GO [6].

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 pro 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 počáteční inicializaci v prvním průchodu získávání hashe. Tato počáteční inicializace ukládá pro každé políčko tři náhodná 64bitová čísla. Hash pro část desky (víme, že je

vždy obdélníková) má podobu jednoho 64bitového čísla. Toto číslo se získává průchodem inicializované implementace 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, díky čemuž porovnání dvou zahashovaných částí gobanu je v konstantním čase. Vytváření hashe je lineární vůči velikosti gobanu, který se hashuje.

3.3.3 Porovnání metod a vylepšení

Porovnali jsme obě methody ukládání vyhledávaných pozic (klíčů). Vytváření klíčů je u obou reprezentací v lineárním čase vzhledem k velikosti. Ale porovnání dvou již vytvořených klíčů (tzn. klíče pro vzor v databázi a klíče pro část pozice na desce) 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 upravit metodou GetNextLine a GetNextColumn, které by nevytvářely celý nový hash, ale pouze operací XOR znovu "přidaly" první řádek, čímž by se díky symetrii XORu odebral a stejnou operací přidal řádek nový. Po analýze jsme dospěli k závěru, že by zrychlení nebylo pro naše účely dostatečné.

3.3.4 Další existující metoda vyhledávání vzorů

Dalším zajímavým způsobem vyhledávání vzorů na desce je využití vyhledávání v textu pomocí patricia tree aplikované na hru go v [11]. V této práci se autor zaměřil na odstranění zpomalení vyhledávání tvarů, které způsobuje veliké množství různě velikých vzorů.

Tato implementace se ovšem ukázala výhodná spíše pro vyhledávání tvarů větších rozměrů. Vzory používané v naší implementaci ovšem nedosahují potřebných velikostí, kdy by bylo použití této metody přínosné. Implementace vyhledávání pomocí patricia tree by mohlo být zajímavé např. pro joseki[2]. To ovšem nebylo předmětem naší práce.

3.4 Implementace Gobanu

Důležitou součástí návrhu programových komponent byla i implementace gobanu. Samotný goban musí kontrolovat pravidla hry go při každém přidání či vymazání kamene. Základní požadavek, který bylo potřeba splnit pro kontrolu splněnosti taktických cílů, bylo rychlé zjištění informací o skupině kamenů (počet svobod, počet kamenů). Pro tuto potřebu byla vytvořena složitá implementace gobanu na základě reprezentace skupin kamenů.

Každé políčko na gobanu je implementováno jako index do databáze skupin. Při každé operaci s gobanem (zahrání kamene, odebrání kamene) jsou informace o skupinách přepočítávány a aktualizovány. Základním principem je, že při každém přepočtu, který nastává pouze po zmiňovaných operacích, se propočítávají pouze sousední skupiny změněného políčka.

Tato implementace tedy umožňuje v konstantním čase zjišťování informací o skupině kamenů. Je podstatně rychlejší přidávání kamenů, které má časovou složitost $O(n + m)$, kde n je počet kamenů spojovaných skupin (v případě, že novým kamenem jsou dvě skupiny spojeny) a m počet zajmutých kamenů (v případě, že novým tahem nějaké kameny zajmeme). Jinak je čas přídání kamene konstantní. Obdobně u mazání kamenů.

[2]Joseki je goistický pojem pro sekvence tahů na začátku hry, kdy začínají první boje o roh desky. Přesné vysvětlení viz sekce A.2 - Rejstřík goistických pojmů

Kapitola 4

Algoritmy

4.1 Vytváření hlavní databáze vzorů

Hlavní databáze vzorů je určena pro potřeby jejich reprezentace. Pro řešič taktických cílů je ovšem nutné reprezentovat všechny vzory ve struktuře, která bude rychlá na vyhledávání. Jak jsme bylo popsáno v sekci 2.2, je tato struktura implementována v hashovací tabulce. Klíčem v této tabulce je vyhledávací tvar, hodnotou pak strom tahů, které danému tvaru odpovídají. V této sekci je popsáno, jak vzniká tento vyhledávací tvar a strom tahů ke každému tvaru.

Shape je uživatelem zadáván jako tvar, kterého je potřeba na desce dosáhnout. Pro potřeby řešiče taktických cílů ovšem potřebujeme databázi, která bude tvořena Při ukládání do hlavní databáze pro potřeby řešiče taktických cílů 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

$$pocetRotaci * pocetCernychKamenu = 8 * pocetCernychkamenu$$

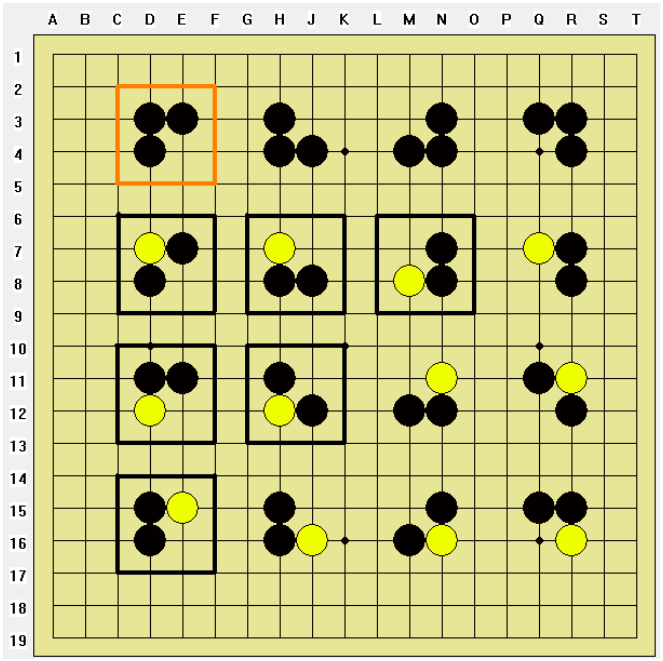
hash klíčů, počet vygenerovaných rotací je vždy 8. Při ukládáni do databáze ale může docházet k tomu, že vyhledávací tvar už je jednou obsažen (např. s jiným stromem tahů). V tom případě se stejný hash znovu neukládá a dochází ke slučování stromů tahů.

```
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;
```

V obrázku 4.1 je ukázka vygenerování vyhledávacích tvarů. Je použit goisty oblíbený 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; ten je na obrázku znázorněn žlutou barvou.

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 tvaru. Avšak, aby řešič taktických cílů byl schopen rozpoznat i tesuji v průběhu jeho odehrávání, bylo nutné ukládat do databáze i mezitvary, které se ukládají stejným způsobem jako celé tesuji, tedy pouze s menším stromem. Tedy každé tesuji je v databázi nejvíce $8 * bn$, kde bn je počet synů s černým kamenem.



Obrázek 4.1: Ukázka ukládání tvaru (goisty oblíbený prázdný trojúhelník) a generování odpovídajících vyhledávacích tvarů. Pozn.:Součástí vyhledávacího tvaru jsou i případná prázdná políčka

4.2 Práce s databází

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

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

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

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

.

V tomto vzorci je h rovno výšce rotovaného vzoru, x a y jsou původní souřadnice, x' a y' jsou výsledné souřadnice po rotaci. U tesuji je navíc potřeba rotovat strom tahům, kde se používá inverzní vzorec.

Přepočítávaní rotací by bylo v průběhu výpočtů taktických cílů 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 tahů 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 stejný hash již znovu neukládá, ale přidávají se nové možné odpovědi k existujícímu vzoru (slučují se stromy tahů).

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 souboru v upraveném formátu SGF.

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 všechna splnění v bílém vrcholu pro dokázání funkčnosti taktického cíle. Jediné nesplnění v bílém vrcholu pak znamená nesplnění celé jedné větve podstromu. Pro každý bílý tah tedy musíme mít odpověď, která dokáže splnit cíl.

Hlavním výsledkem našeho datového modelu je veliké omezení možných větvení. V prvé řadě je strom tahů za útočníka vytvářen pouze z tahů uvedených v databázi vzorů, nikoliv ze všech možných tahů na desce. Další heuristikou je omezení hledání tahů pouze na okolí místa, kde má být splněn taktický cíl. V naší práci se zabýváme použitím výhradně vzorů, proto je potřeba hledat dané vzory pouze do takové vzdálenosti od cílové skupiny kamenů, v jaké existuje nějaký vzor, který se dané cílové skupiny kamenů dotýká. Okolí cílové skupiny se tedy vybírá jako okolí do vzdálenosti největšího vzoru v databázi.

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

```
2. které se dotýkají skupiny, která je součástí cíle;
3. while ( Neexistuje syn v hloubce 1,
4. jehož všichni potomci plní taktický cíl )
5.    Prozkoumej další tvar z databáze,
6.       který je relevantní k dané pozici.
```

Řádek 1. je tedy modifikací algoritmu popsaného v kapitole 2.2. Tato modifikace spočívá v omezení na okolí cílové skupiny, což je popsané v předchozím odstavci.

Kapitola 5

Závěr

5.1 Shrnutí práce

Cílem projektu bylo navrhnout datové struktury pro vhodnou reprezentaci vzorů a vytvoření programu, který je schopen řešit taktické cíle hry go na základě definované databáze vzorů.

Podařilo se nám implementovat aplikaci s propracovaným datovým modelem a uživatelsky přívětivým prostředím. Program umožňuje načítání a kompletní integraci uložených her ve formátu SGF (Smart Game Format [14]) a vykreslení stromu hry pro pohodlné prohlížení odehrané partie, na což jsou hráči go z podobných programů již zvyklí. V neposlední řadě aplikace nabízí propracované uživatelské prostředí pro kompletní editaci souborů a spouštění řešiče taktických cílů, které bude jistě zajímavé i pro uživatele z řad goistů.

Pro programátory, kteří by chtěli tento program dále vyvíjet, je nejdůležitější jeho rozšiřitelnost. Proto je implementace napsána podle standardů objektově orientovaného programování se striktním dodržováním oddělení rozhraní od implementace pro všechny důležité třídy. Díky tomu je možné aplikaci rozšířit o nové implementace gobanu, hashování, uložení tvarů i stromu hry. Navíc je velmi jednoduché přidávání nových taktických cílů. Dokonce je možné integrovat i nové typy vzorů.

Celkovou výhodu ShapeGo oproti jiným aplikacím zaměřeným na počítačové go vidím v možnosti si matchování vzorů přímo vyzkoušet. Uživatelské prostředí také umožňuje okamžitě vidět dopad změny databáze vzorů na výsledky řešiče taktických cílů. Ukázkové matchování databáze vůči pozici na desce názorně ukazuje, jak je velký výsledný strom vhodných tahů.

5.2 Možná zlepšení

V budoucnosti bychom rádi přidali nové 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 řetězce bitů by porovnávání bylo možné pomocí bitového or. Tato implementace by nezrychlila vyhledávání vzorů oproti již implementovaným metodám, ale umožnila by zakódování mnohem většího množství informací o okolí vzoru, které může zlepšit funkčnost tesuji.

Universalita celého systému ukládání tvarů umožňuje rozšíření řešiče taktických cílů i na tzv. joseki (viz Rejstřík goistických pojmů). Vytvořená persistentní databáze tvarů navíc umožní díky své slučitelnosti s formátem SGF nahrání celé databáze Kogo's joseki dictionary [10]. Díky implementaci 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 rozhodují reální hráči. Pro toto rozšíření by bylo nutné přidat nový taktický cíl a pozměnit podmínky pro zahrání tahu z databáze. Předložená práce však k tomuto tématu nebyla určena.

5.3 Pár slov na konec

Program ShapeGo byl velikou výzvou a velkou zkušeností. Výsledek této práce je prvním krokem k vytvoření komplexního programu pro testování použitelnosti vzorů v různých fázích hry. Rozhodně bychom rádi na programu dále pracovali, protože existuje stále mnoho vylepšení, které by ShapeGo mohlo posunout za další hranice, případně přiblížit použitelnost běžnému uživateli.

Literatura

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

[2] D. Silver, R. Sutton, and M. Müller. Reinforcement learning of local shape in the game of Go. In Twentieth International Joint Conference on Artificial Intelligence (IJCAI), pages 1053-1058, Hyderabad, India, 2007, `http://webdocs.cs.ualberta.ca/~mmueller/ps/silver-ijcai2007.pdf`

[3] Charles Matthews, Seong-june Kim: *Shape Up!*, 2003, online book: `http://gobase.org/studying/articles/matthews/shape_up/`

[4] GNU GO, Official website: `http://www.gnu.org/software/gnugo/gnugo.htm`

[5] GNU GO, Documentation, The Pattern Code: `http://www.gnu.org/software/gnugo/gnugo_9.html#SEC105`

[6] GNU GO, Documentation, Hashing of positions: `http://www.gnu.org/software/gnugo/gnugo_11.html#SEC139`

[7] Historie hry go: `http://goweb.cz/ogo/historie`

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

[9] Kiseido Go Server - CGoban 3 download page: `http://www.gokgs.com/download.jsp`

[10] Kogo's Joseki Dictionary: `http://waterfire.us/joseki.htm`

[11] Martin Müller: *Pattern matching in explorer*, Game Playing System Workshop, ICOT, Tokyo, 1991, `http://www.cs.ualberta.ca/~mmueller/ps/mueller91b.ps`

[12] Martin Müller: *Not like other games - why tree search in Go is different*, Fifth Joint Conference on Information Sciences, JCIS 2000, `http://www.cs.ualberta.ca/~mmueller/ps/jcis2000.ps.gz`

[13] MoGo, Facts about MoGo: `http://senseis.xmp.net/?MoGo`

[14] SGF, Specifikace formátu SGF verze 4: `http://www.red-bean.com/sgf/`

[15] T. Cazenave: *Theorem Proving in the Game of Go*, First International Conference on the Scientific Study of Go, Seoul 2001

Příloha A

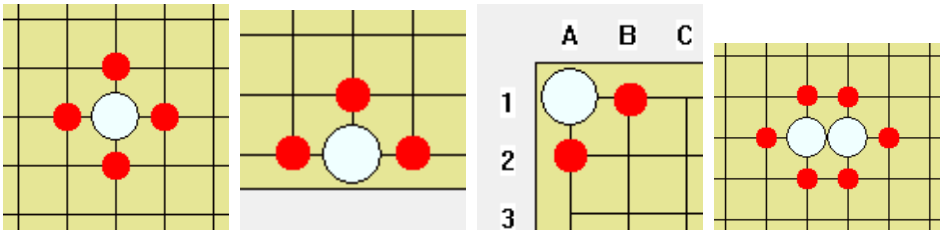
Stručný úvod do hry go

A.1 Pravidla

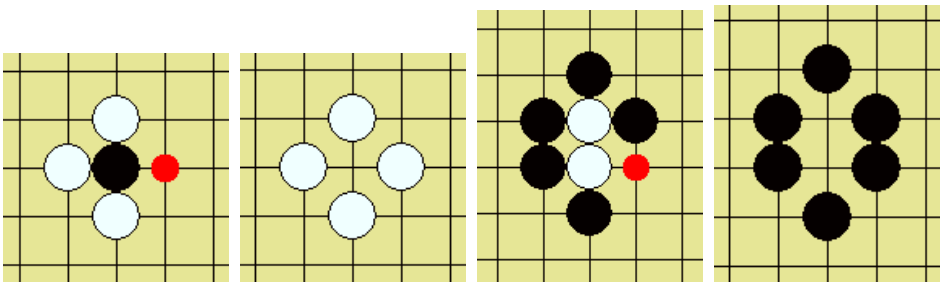
K pochopení této bakalářské práce je potřeba znát pravidla hry go, která, na rozdíl od strategie hry samotné, nejsou složitá. Ke hře je potřebná hrací deska, tzv. goban, a kameny - černé a bílé. Hrají dva hráči, kteří se střídají v tazích. Začíná hráč, který má černé kameny. Tah spočívá v položení jednoho kamene na goban, a to na průsečík. Po položení kamene na desku se s ním již nijak nepohybuje (kromě pravidla č.1 - viz níže).

A.1.1 Svobody

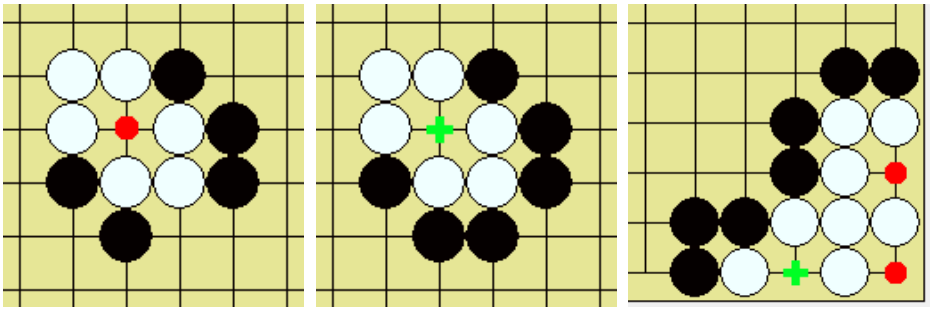
Důležitou vlastností každého kamene i každé skupiny kamenů stejné barvy na desce je počet sousedních volných průsečíku (tzv. svobod). Na obrázcích obrázcích A.1.1 jsou znázorněny svobody bílých kamenů červeně.



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



Obrázek A.2: Obrázky představují situaci vždy bezprostředně před zajmutím kamenů (jejich poslední svoboda je označena červeně) a po zajmutí skupiny a odebrání kamenů z desky

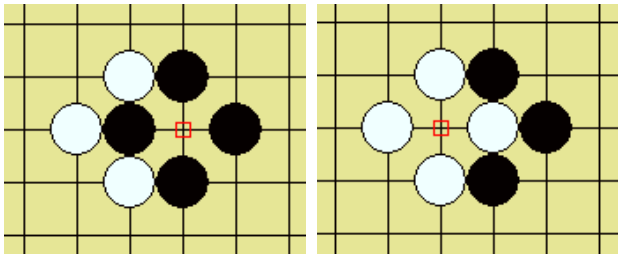


Obrázek A.3: Ukázka 2.pravidla. Červeně (kolečko) jsou označena místa, kam černý v následujícím tahu nemůže zahrát, a zeleně (křížek) místa, kam díky zajmutí soupeřových kamenů zahrát může.

A.1.2 Tři jednoduchá pravidla go

Pravidlo č.1.:
Jestliže má kámen (či skupina kamenů) po posledním tahu soupeře 0 svobod, je zajat soupeřem a odebrán (odebrána) z desky. Ukázka zajmutí kamenů je na obr. A.1.2

Pravidlo č.2.:
Tahem se nesmí spáchat sebevražda - tzn. nemůže se zahrát na místo, kde by tento nový kámen neměl žádnou svobodu, nebo by se tímto tahem ubrala poslední svoboda vlastní skupiny. Na takové místo však je možné zahrát, jestliže tento nový tah vytváří nové svobody, a to pravidlem č.1 (tzn. zajmutím soupeřových kamenů). Názorná ukázka je na obr. A.1.2.



Obrázek A.4: 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 se nesmí zajmout kámen bezprostředně po jeho zahrání soupeřem, ale musí se alespoň jednou hrát na jinou pozici (tzv. hrozba na ko). Na obr. A.1.2 je znázorněna typická situace ko.

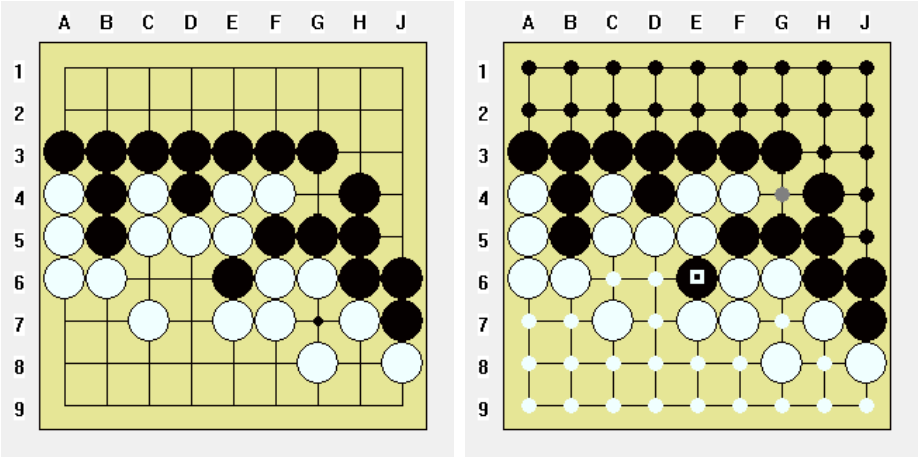
Toto jsou veškerá pravidla. Dají se z nich ale odvodit další zajímavé vlastnosti go, např. tzv. živost skupin, schody a pod. Viz Rejstřík goistických pojmů.

A.1.3 Cíle hry

Cílem hry je získat co nejvíce bodů. Každý zajatý kámen soupeře a každý průsečík vlastního území představuje zisk jednoho bodu. Území jednoho hráče můžeme nadefinovat jako prázdné průsečíky obklopené kameny jedné barvy - daného hráče. Během hry se oba hráči snaží o získání co největšího území, o které spolu na desce "bojují" či se přetlačují. Zajímání kamenů je pouze prostředkem k získávání území.

Na desce můžeme rozlišovat lokální bitvy s lokální taktikou boje, jejichž výsledek se vždy snažíme využít k celodeskové strategii. Go je silně strategická hra, ve které i lokální prohra může v celodeskovém kontextu přinést výhodu.

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.



Obrázek A.5: 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.1.4 Ukázka partie

Partie se hraje obvykle na desce 19x19, ale jsou používané i menší desky 13x13 a 9x9. Následuje ukázka konce hry na malé desce 9x9 na obr. A.1.4. 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). Když oba soupeří vidí, že je nějaký kámen (skupina kamenů) mrtvý, pak jej již není potřeba obkličovat a odebírat mu všechny svobody. Bílý má v tét oukázkové hře navíc jednoho zajatce, kterého odebral z desky v průběhu hry. Tento kámen se původně vyskytoval na souřadnicích G-7.

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, mít přehled o celé situaci na desce a využívat i nečekaných lokálních proher, 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.
- **Hendikep:** Kameny, které černý hráč (slabší hráč) umístí na goban před prvním tahem hry. Je to výhoda pro černého hráče v případě, že je slabší než bílý hráč. Když je hrána hra s hendikepem, tak je umístění hendikepu považováno za první tah hry a na tahu je bílý hráč. Hendikep zajišťuje vyrovnanou hru i dvou velmi rozdílně silných hráčů.
- **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 by bylo možné vzájemné zajímání kamenů do nekonečna. Taková situace je ošetřena pravidlem hry 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 za určité lokální situace na desce mají plnit určitý účel s nejlepším možným výsledkem pro hráče, který tesuji zahraje. V případě, že jsou splněny předem dané podmínky, pak je i vždy úspěšná. Nejčastějším příkladem je zajmutí kamene (popř. skupiny kamenů) nebo naopak únik s kamenem (popř. se skupinou kamenů) z hrozícího zajmutí.

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

Příloha B

Uživatelská dokumentace

B.1 Úvod - o programu

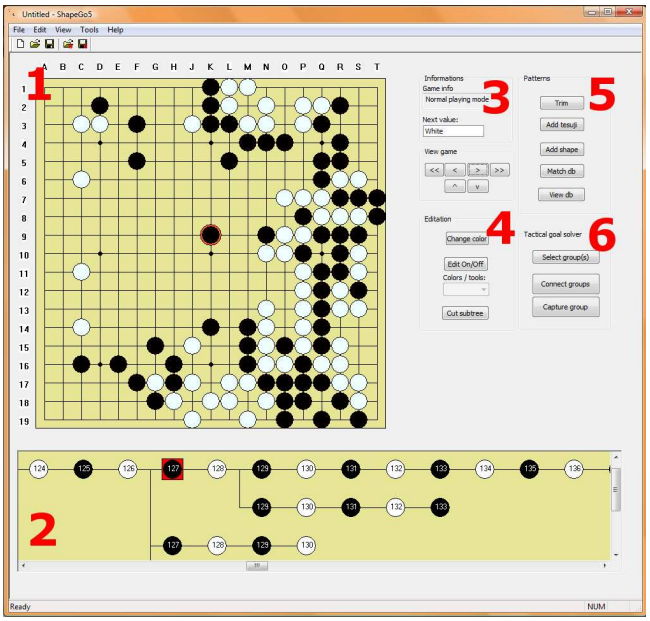
Program ShapeGo je určen pro spouštění řešiče taktických cílů hry go založeného na rozpoznávání vzorů. Jeho hlavní náplní je správa databáze vzorů a jejich možná editace uživatelem. Zároveň nabízí uživatelské rozhraní pro prohlížení a editaci souborů ve formátu SGF, na které jsou hráči go zvyklí z jiných programů. Přiložená verze programu pravděpodobně nejvíce zaujme programátory zajímající se o hru nebo naopak hráče go zajímající se o programování této hry.

B.2 Spuštění programu a hlavní okno

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

Hlavní okno by se dalo rozdělit do několika hlavních částí, kde každá slouží k jinému účelu. Pochopení tohoto rozdělení je zásadní pro správné ovládání programu a proto jej teď stručně shrneme. Hlavní okno spolu s označením oblastí je na obr. B.2.

- Oblast 1 a 2 slouží k prohlížení hry. Hlavní částí je samotný goban, druhou částí je strom tahů. Po kliknutí na nějaký vrchol stromu se na gobanu zobrazí pozice odpovídající tomuto tahu.
- Oblast 3 má převážně informativní charakter. najdete zde informace o tom, v jakém módu se program právě nachází i jaký hráč je právě



Obrázek B.1: Ukázka programu se znázorněnými oblastmi podle účelu použití.

na tahu/jaký editační nástroj je právě zvolen. V této oblasti se také nacházejí ovládací prvky pro posun stromem hry.

- Oblast 4 nabízí všechny editační prvky. V základním módu můžete využít tlačítko pro změnu barvy. Je zde také umístěno spouštění editačního módu. V něm můžete nejen přidávat libovolné množství kamenů jedné barvy, ale i kameny mazat. V neposlední řadě je zde umožněno odřezávání celé větve stromu (včetně aktuálního vrcholu).

- Oblast 5 je určena k vytváření vzorů a spouštění prohlížení databáze. Postup vytvoření vzoru je popsán dále v sekci B.4.

- Oblast 6 slouží ke spouštění řešiče taktických cílů. Přesný postup je popsán dále v sekci B.5.

Přesný popis jednotlivých ovládacích prvků naleznete na konci *Uživatelské dokumentace* v sekci B.6.

B.3 Databáze vzorů

Pro spouštění řešiče taktických cílů je nezbytná databáze vzorů. Tu je možné buď načíst již předpřipravenou ze souboru nebo vytvořit novou. Vytváření databáze probíhá postupným vytvářením vzorů, které je popsáno v sekci B.4. V této sekci se tedy budeme dále zabývat načtením již vytvořené databáze.

V programu je dále používána vnitřní databáze pro řešič taktických cílů. Ta je vytvářena automaticky z načtené databáze při spouštění řešiče taktických cílů. Tato struktura je pro uživatele transparentní. Podrobnosti můžete nalézt v textu bakalářské práce např. v sekci 3.2.

B.3.1 Načítání a prohlížení databáze vzorů

Databáze se do programu načte pomocí menu *Database - Load database*. Takto načtená databáze nebude zobrazena uživateli, ale o jejím úspěšném nahrání bude informovat Messagebox. Pro načtení databáze můžete také využít tlačitko toolbaru se symbolem otevíraní a červeným "D".

Pro prohlížení databáze vzorů zvolte menu *File - View database*. Takto otevřená databáze vzorů se zobrazí ve stromu hry. Každý vrchol ihned pod kořenovým vrcholem zobrazuje jeden vzor. Shapes (tvary) jsou reprezentovany jediným vrcholem, který reprezentuje daný tvar ve stejné podobě, v jaké byl uložen uživatelem. Tesuji jsou vždy zobrazeny v jednom vrcholu, kde je uložen vyhledávací tvar (počáteční situace) a následující strom tahů s možnými odpověďmi včetně reakcí soupeře.

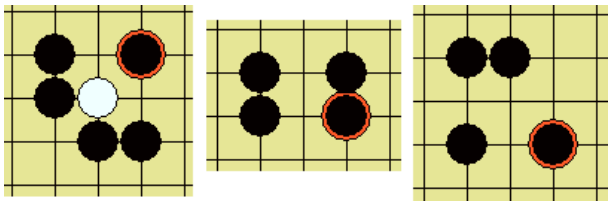
Pro vyzkoušení programu doporučuji využít předpřipravené databáze a ukázkové příklady, které jsou na přiloženém DVD v adresáři *Vzory a priklady*.

B.4 Vzory a vytváření

Vzory dělíme na dva typy: Tvary a tesuji.

- Tvar, nebo-li "shape", je pouze určitý tvar na desce (např. koňský skok, bambusové spojení, stůl - viz obr. B.4).
- Tesuji je tvar na desce, který bude vyhledáván, a strom správných reakcí (tahů) na situaci za oba hráče (např. síť B.4 nebo 3.1 a 3.1).

Pro podrobnější vysvětlení tvarů a tesuji viz kapitola 3.1.



Obrázek B.2: Ukázka známých vzorů spolu s určením taktických cílů, které tyto vzory plní. Zleva: síť (zajmutí kamene - tesuji), bambusový kloub (spojení dvou skupin - shape), částečný stůl (univerzálně dobrý vzor nebo spojení dvou skupin - shape). Tyto vzory jsou popsány např. v [3]

.

Vytváření nových vzorů a jejich ukládání do databáze vždy probíhá ve dvou fázích:

1. Fáze přípravy vzoru v hlavním okně aplikace. Nejprve je nutné vytvořit v hlavním okně požadovaný vzor včetně sekvence tahů ve stromě v případě tesuji. Uložen poté bude tvar, který je právě zobrazen na zvolené části desky.

2. Uložení vzoru do právě používané databáze. Ukládání probíhá po zvolení *Add shape* nebo *Add tesuji* pomocí dialogů k tomu určených (pro každý tip vzoru je jeden dialog).

Nový vzor se vždy uloží do databáze, která je právě načtena v programu. Tato databáze je nutná pro potřeby řešiče taktických cílů[1]. Jestliže v paměti není žádná databáze, pak se automaticky vytvoří nová.

Nasledují podrobné kroky pro jednotlivé vzory.

B.4.1 Přidávání tvaru

Tvar do databáze přidáte v následujících jednoduchých krocích:

1. Pomocí editačních nástrojů na desce vytvoříme požadovaný tvar. Tzn. např. přímo bambusový kloub.

2. Stiskneme tlačítko "Add shape".

[1]Podrobné vysvětlení funkčnosti databáze viz kapitola 3.2

3. Objeví se trimovací dialog, v něm vyberte levý horní roh a pravý spodní roh obdélníku, který bude uložen jako nový tvar.

4. Po potvrzení ořezání uvidíte v novém dialogu pro přidávání tvarů ("Add shape") náhled vzoru.

5. Vyberte taktický cíl, pro který je vzor určen. V případě, že se jedná o univerzálně dobrý tvar, tak zvolte "Universal".

6. V případě potřeby invertujte barvy vzoru (Vzor musí být vždy ukládán za černou barvu).

7. Nyní můžete potvrdit a shape bude uložen do interní databáze, kterou doporučujeme ihned uložit i persistentně pomocí volby menu *Database - Save database* v hlavním okně programu.

B.4.2 Přidávání Tesuji

Tesuji se od tvaru liší v tom, že nejprve se vytvoří tvar výchozí pozice a poté musí být vytvořen celý strom tahů. Tesuji do databáze přidáme v následujících krocích:

1. Pomocí editačních nástrojů na desce vytvoříme požadovaný výchozí tvar.

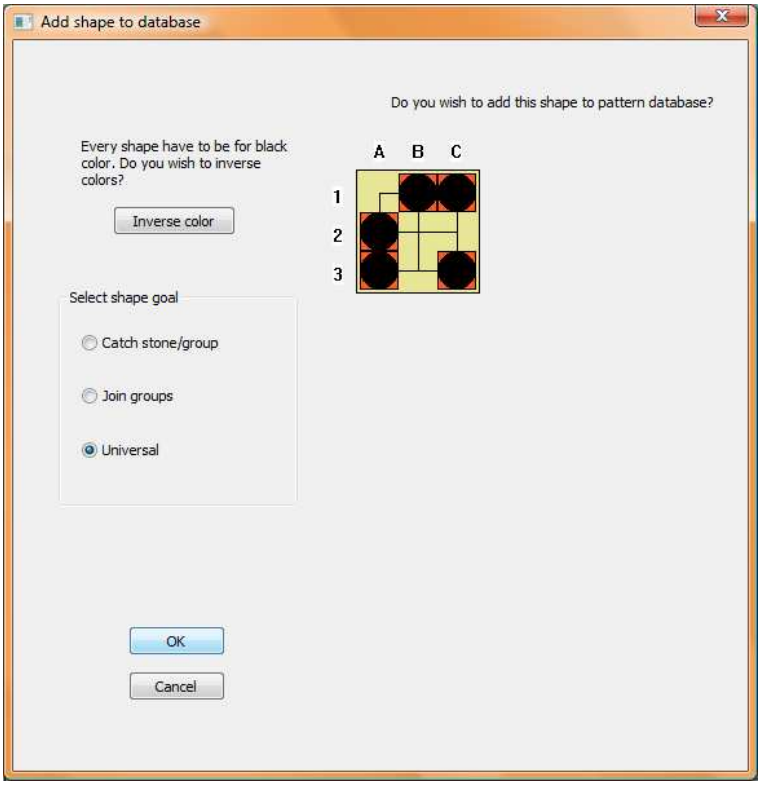
2. Pomocí normálního módu odehrávání tahů vytvořte celý strom tahů, který odpovídá danému tesuji. Větve, které neodpovídají, můžete odstřihnout pomocí tlačítka "Cut subtree".

3. Vrťte se ve stromě hry na počáteční pozici, která má být uložena jako výchozí pozice a stiskněte "Add tesuji".

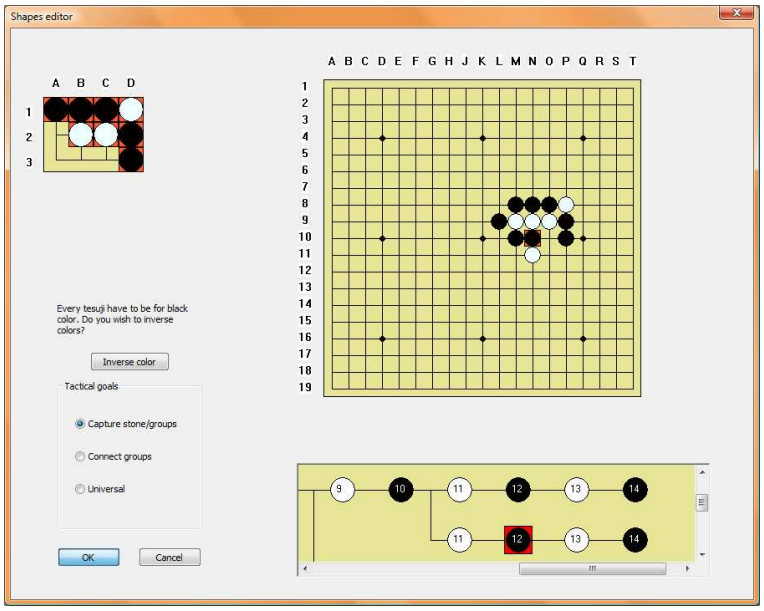
4. Objeví se trimovací dialog, v něm vyberte levý horní roh a pravý spodní roh obdélníku, který bude uložen jako vyhledávací tvar.

5. Po potvrzení ořezání uvidíte v novém dialogu pro přidávání tvarů ("Add shape") náhled tesuji.

6. Vyberte taktický cíl, pro který je vzor určen. V případě, že se jedná o univerzálně dobré tesuji, tak zvolte "Universal".



Obrázek B.3: Dialog Add shape.



Obrázek B.4: Dialog Add tesuji.

7. V případě potřeby invertujte barvy vzoru (Vzor musí být vždy ukládán za černou barvu).

8. Nyní můžete potvrdit a tesuji bude uloženo do interní databáze, kterou doporučujeme ihned uložit i persistentně pomocí volby menu *Database - Save database* v hlavním okně programu.

B.4.3 Obecné informace

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

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 žádná 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 přídávání vzoru je tvar uložen do databáze, která je právě načtena v paměti, případně se vytvoří nová, pokud ještě žádná načtena nebyla.

Pro persistentní uložení databáze zvolte v hlavním okně "Save database" v hlavním menu File nebo na nástrojové liště.

B.5 Spouštění řešiče taktických cílů

Před spuštěním řešiče taktických cílů musíte mít nejprve načtenou databázi vzorů (viz sekce B.3). Při prvním zkoušení programu doporučujeme nejprve načíst již vytvořenou databázi přiloženou na DVD ve složce *Vzory a priklady*, kde naleznete i ukázkové příklady pro souštění řešiče.

Řešič taktických cílů se spouští pomocí následujících kroků:

1. Načtěte databázi vzorů do programu (viz sekce B.3.1)
2. Na gobanu vytvořte situaci, na kterou má být řešič taktických cílů spuštěn. Doporučuji nejprve využít předpřipravených příkladů, která naleznete na přiloženém DVD ve složce *Vzory a priklady*.
3. Jestliže je program v módu editace, tak ji vypněte.
4. Každý taktický cíl je spouštěn za hráče, který je právě na tahu. Proto je nutné se ujistit, že příští tah je za barvu, kterou si přejete. V opačném případě můžete barvu změnit tlačítkem *Change color*.
5. Pro každý taktický cíl je potřeba vybrat skupinu/skupiny kamenů, na kterou/které má být daný taktický cíl aplikován. Skupinu vyberete tak, že zvolíte tlačítko "Highlighting groups", tím je zapnut mód vybírání skupin. Na gobanu můžete nyní vybírat skupiny kamenů. Ty, které budou vybrány, budou ohraničené modrou barvou.
6. Nyní je možné kliknout na vybraný taktický cíl.
7. Výsledkem je jeden vybraný tah, který řeší daný taktický cíl.

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.

B.6 Přehled ovládacích prvků

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, nebo kameny mazat (i jenom samostatný vymazaný kámen bude ve stromě zaznamenán v jednom samostatném vrcholu stromu.

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.

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 až navyjímky psán podle standardů objektového programování.

Vygenerovaná programátorská dokumentace (v anglickém jazyce) pomocí doxygenu je přílożena na DVD va složce */doc*.

Během implementace programu byl důraz 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.

Následující dokumentace je rozdělena do několika hlavních celků: Reprezentace gobanu a stromu hry, Implementace a provázanost tvarů, persistence dat a vlastní tvar souborů s databází vzorů, řešič taktických cílů a implementace uživatelského prostředí.

C.2 Reprezentace hry

C.2.1 Goban - hrací plocha

Pro goban je vytvořen interface CGobanI. Implementace gobanu musí sama kontrolovat pravidla hry go a nabízet i všechny potřebné metody pro zjišťování situace na desce a editaci kamenů. Dalo by se říct, že goban je základním stavebním kamenem.

CGobanGroups: Třída implementující interface CGobanI. 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 skupiny.

Tato implementace 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 verze sice měla rychlejší odebírání kamenů, ale jinak znatelně pomalejší všechny ostatní operace, a proto jsme ji dále nevyvýjeli. Navíc získávání informací o skupinach bylo v této implementaci v linearním čase.

Nejdůležitější metody CGobanI a jejich implementace v CGobanGroups:

- $boolMakeMove(constSPositionposition, constStoneValuecolor)$ - Přidá na goban jeden kámen dané barvy na danou pozici. Zároveň kontroluje veškerá pravidla hry go (tzn. zda je možné daný tah zahrát a zda daným tahem není zajat nějaký kámen). Také přepočítává informace o skupinách kamenů. Vrací boolean s informací, zda bylo možné kámen přidat na základě pravidel go.

- $boolDeleteStone(constSPositionposition)$ - Vymaže z gobanu kámen na dané pozici. Zároveň přepočítává všechny informace o skupinách (tzn. i zda nebyly skupiny stejné barvy rozděleny). Vrací boolean s informací, zda bylo možné kámen vymazat.

- $CGoTreeI * GetPatternMoves(StoneValuenextMoveColor)$ - Tato metoda vytvoří hash pro všechny malé podčásti gobanu (podle velikostí vzorů v databázi) a pro tyto hashe získá stromy tahů z databáze. Tyto stromy sloučí v jediný, který je návratovou hodnotou této metody. Jedná se tedy o primární metody při spouštění řešiče taktických cílů.

C.2.2 Strom hry

Pro strom hry je vytvořen interface CGoTreeI, obsahující interface pro vrchol stromu CGoTreeNodeI. Strom hry je důležitý nejen pro reprezentaci hry, ale i pro reprezantací databáze vzorů. Navíc ve formátu stromu je databáze vzorů persistentně ukládána.

CGoTreeVec: Třída implementující interface CGoTreeI pomocí vectorů. Obsahuje vnitřní třídu pro vrchol CNode, která implementuje interface CGoTreeNodeI. Tato třída implementuje n-ární strom hry pomocí STD 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. Důležité jsou také odkazy na kořenový, aktuální a poslední vrchol.

Nejdůležitější metody CGoTreeI a jejich implementace v CGoTreeVec:

- $voidAddStone(constStoneValuecolor, constSPositionnewPosition)$ - Vytvoří nový vrchol s přidávaným kamenem. Tento vrchol přidá jako nového syna za aktuání vrchol. Aktuální vrchol nastaví na tento nově přidaný vrchol.

- $voidAddEditStone(constStoneValuecolor, constSPositionnewPosition)$ - Přidává nový editovaný vrchol. Zásadní rozdíl je v tom, že všechny editace prováděné v jednom celku jsou ukládané v jednom vrcholu. Pokud aktuální vrchol již má editované kameny, pak tato metoda přidá kámen do seznamu editovaných kamenů pro daný vrchol. Jinak vytvoří nový vrchol obdobně jako v AddStone s tím rozdílem, že nový kámen bude přidán do seznamu editovaných kamenů.

- $voidMergeTrees(CGoTreeI*toTree, CGoTreeI*fromTree)$ - 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 kořeny (root). Od kořenů projde všechny syny a v případě, že nalezne dva, které mají stejné kameny, zmerguje tyto dva nody opět v metodě MergeNodes(). Jestliže ve fromTree existují vrcholy, které se nespojí, pak se překopírují na odpovídající místa v toTree.

- $voidGetPrev(vector < SStone > *stonesToPrev)$ - Tato metoda je nutná pro posunutí se při prohlížení hry o tah zpět. V parametru stonesToPrev vrací seznam kamenů odehraných do předchozího tahu (včetně). Zároveň se tato metoda stará o nastavení ukazatele na aktuální vrchol.

- $boolGetNext(vector < SStone > *nextStones, size_tn)$ - Metoda vracející kameny v i-tém synovy aktuálního vrcholu.

- $vector < CGoTreeI* > GenerateRotations(SGobanSizegobanSize)$ - Metoda je potřebná při generování rotací tesuji (při vytváření databáze vzorů pro účely řešiče taktických cílů). Tato metoda projde celý strom tahů a vygeneruje všechny potřebné rotace. K tomu využívá privátní metody PosMirrorHoriz a PosRotation90.

C.3 Tvary a tesuji

Základní interface pro libovolný vzor je CPatternI. Od něj poděděné rozhraní CShapeI a CTesujiI. CShapeI je interface pro jednoduchý tvar na desce. CTesujiI musí obsahovat i strom navazujících tahů včetně odpovědí soupeře. Oba typy vzorů jsou podrobněji rozebrány např. v kapitole 3.1.

Tyto implementace vzorů jsou používány v základní reprezentaci databáze CPaternDb.

Nejdůležitější metody CPatternI:

- $map < CHashHelper, CGoTreeI* > GetHashPattern()$ - vrací mapu provázaných hashů tvarů, které se budou vyhledávat, a stromů tahů pro tento vzor.

- $CGobanPartValueGetGobanPattern()$ - Vrací tvar vzoru tak, jak byl zadán uživatelem. (v této podobě se mj. ukládá do databáze CPatternFileDb)

- $CGoTreeI * GetTree()$ - Vrací strom tahů navazujících na tento vzor. V případě CShapeI vrací NULL (tvar - shape - neobsahuje strom tahů (viz začátek kapitoly).

Program obsahuje následující implementace těchto rozhraní:

- **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. Dále je nutný konstruktor s CGobanPartValue * a CGobanI *. Všechny hashe se ukládaji do databáze vzorů pro účely řešiče taktických cílů (tzn. do hashovací tabulky) pomocí singletonu CHashHelper, která v operátoru porovnání volá metodu hashe CHashPatterI::LessThen().

- $boolLessThan(constCHashPatternI\&m1)$ - Vracejí stejnou hodnotu, jakou by vracel operátor porovnání <=. Tento operátor porovnání je použit v implementaci třídy map z STL, ve které je uložena celá databáze pro potřeby řešiče taktických cílů.

C.4.1 CHashMatrix

Implementace 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. Tato implementace zůstala v programu spíše pro testovací účely.

C.4.2 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. Poté se již pracuje s tímto hashem gobanu a pouze se porovnává s databází. Tedy vytvoření hashe pro goban je v lieárním čase v době spuštení řešiče taktických cílů.

C.5 Databáze vzorů

V programu jsou dvě třídy pro databáze vzorů. Obě třídy jsou implementovány jako návrhový vzor singleton.

C.5.1 CPatterDb

Implementuje hashovací tabulku pro řešič taktických cílů pomocí STD 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 verzi programu implementován třídou $CHashZobrist$. Strom tahů je $CGoTreeI$ implementován $CGoTreeVec$. Ve vnitřní implementaci si databáze pamatuje zvlášť vzory pro každý typ taktického cíle. Používá vždycky ty, které plní právě počítaný taktický cíl.

- $voidAddShape(constCPatternI * shape)$ - Přidá tvar (shape) do databáze tvarů pro řešič taktických cílů. Vytvoří odpovídající hashe pro všechny rotace tvaru. Zároveň o "jeden tah zpět" tzn. bez jednoho černého kamene. K výsledným hashům, které jsou reprezentovány CHashI, přidá jako hodnotu složený strom tahů CGoTreeI. Jestliže v tabulce je již uložen daný hash, pak se stromy tahů slučují v jeden.

- $voidAddTesuji(constCTesujiI * tesuji)$ - Přidá tesuji do databáze tvarů pro řešič taktických cílů. Vytvoří hashe pro všechny rotace daného vyhledávacího tvaru. Zároveň také rotuje strom tahů, který je reprezentován jako CGoTreeI. Jestliže v tabulce je již uložen daný hash, pak se stromy tahů slučují v jeden.

- $CGoTreeI*GetMatchingPattern(constCHashZobrist*hash)$ - Vrací strom tahů odpovídajícího hashe v databázi.

- $voidUseTacticalGoal(Tacticgoal)$ - Nastavuje, který taktický cíl bude nyní vyhledáván (tzn. která databáze bude využívána).

- $InitFromFIleDb()$ - Inicializuje databázi z databáze ve formátu CPatternFileDb, tzn. načtenou ze souboru.

C.5.2 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 je obdobně jako v $CPatternDb$ reprezentován $CGoTreeVec$. 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.

- $voidAddPattern(constCGobanPartValue * g, constCGoTreeI * tree, constTacticgoal)$ - Toto je zásadní metoda CPatternFileDb, která přidává do této reprezentace databáze nový vzor ve formátu, v jakém byl zadán uživatelem.

C.6 Taktické cíle

Taktický cíl reprezentuje interface CTacticalGoal. V našem programu jsou řídy CCaptureGoal a CConnectGoal, které implementují interface CTacticalGoalI. Pro hledání řešení pro taktický cíl je použita třída CTacticalGoalSolver.

Dúležité metody CTacticalGoalI:

- $boolIsFulfilled()$ - Metoda, která zjišťuje splnění taktického cíle.
- $boolIsCorrect()$ - Tato metoda kontroluje korektní zadání taktického cíle.
- $SPosition * CheckBasicSolving()$ - Metoda, která zkouší jednoduchá řešení taktického cíle. Jedná se pouze o zabrání kamene v případě, že má už pouze jednu svobodu, a spojení dvou skupin kamenů, pokud jsou od sebe vzdáleny pouze jedno políčko.

Důležité metody CTacticalGoalSolver:

- $SPosition*CTacticalGoalSolver :: GetMove()$ - Metoda, která pouští řešič taktických cílů. Vrací tah jakožto řešení taktického cíle.
- $SPosition*CTacticalGoalSolver :: GetSimpleMove()$ - Metoda která hledá jednoduchá řešení v případě, že řešič nenalezne řešení na základě databáze tvarů. Volá metodu CheckBasicSolving třídy CTacticalGoal.
- Nejdůležitější metoda je IsFullfilled, která určuje, zda je v aktuální situaci na desce splněn daný taktický cíl.
- $CTacticalGoalSolver(CGobanGroups*goban, CTacticalGoalI*goal, StoneValuenextMoveColor, CGroupSurroundings*surroundings)$ - V tomto případě je konstruktor velmi důležitou metodou. Zde se předávají všechny potřebné informace a zadání taktického cíle. V paramterech se zadává pozice na gobanu, taktický cíl, který se bude řešit, barva hráče za kterého se bude hledat řešení a všechny okolní pozice (okolí, ve kterém má ještě smysl hledat vzor - vysvětleno v sekci 4.3).
- $boolProving(CGoTreeI :: CGoTreeNodeI*node)$ - Důležitá primární metoda, která reprezentuje hladinu stromu, ve které se snažíme splnit taktický cíl.
- $boolDisproving(CGoTreeI :: CGoTreeNodeI * node)$ - Důležitá primární metoda, která se snaží nesplnit taktický cíl.

C.7 Ukládání do souborů

Ukládání do souboru je použito při ukládání i načítání her uložených ve formátu SGF, ale i pro ukládání a načítání databáze vzorů v lehce upraveném formátu SGF, který je ale stále s drobnými chybami kompatibilní s jinými prohlížeči her go. pro účely persistentního uložení databáze vzorů jsme do formátu SGF přidali parametry:

- GL[číslo] reprezentující taktický cíl. Tento parametr musí být uveden na začátku reprezentace daného vzoru.
- PS[x y] udávající velikost vzoru, kde x udává šířku a y výšku.

Strukturou je persistentní databáze vzorů reprezentována jako strom tahů, kde první syn kořenového adresáře obsahuje seznam kamenů, které

tvoří základ vzoru (u tesuji vyhledávaci vzor, u tvaru samotný shape zadávaný uživatelem). Podrobně viz sekce **??**.

Pro ukládání do souboru formátu SGF jsou použity následující přetížené metody:

- $ostream\&operator << (ostream\&out, const CGameInfo\&info)$ - Game info reprezentuje informace o hře. V běžných goistických programech obsahuje jména a sílu hráčů, výsledek hry a další informace o partii.
- $ostream\&operator << (ostream\&out, const CGoTreeI :: CGoTreeNodeI\&node)$ - Vypsání jednoho vrcholu stromu (tzn. kamenů zahraných v tomho tahu, včetně editovaných).
- $ostream\&operator << (ostream\&out, const CGoTreeI\&tree)$ - Přetížený operátor pro vypsání stromu do formátu SGF přesně podle SGF specifikace (která je definována v [14]). Tento operátor využívá metody pro vypsání jednoho vrcholu stromu.
- $ostream\&operator << (ostream\&out, const CGobanPartValue\&partValue)$ - Vypsání jednoduché reprezentace gobanu. Využívá se stejná struktura jako kdyby všechny kameny na gobanu byly přidány při editaci.
- $ostream\&operator << (ostream\&out, const CPatternFileDb\&patternDb)$ - Vypsání celé databáze vzorů. Tato metoda využívá předchozí metody vypsání stromu a gobanu.

C.8 Rozšiřování implementace

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

Jako příklad uvedeme, 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 je také možné rozšířit třídy implementující goban (a tedy kontrolující 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.

Rozšiřitelné jsou také typy vzorů. Pro jejich rozšíření už je ale potřeba i změna uživatelského rozhraní.

Příloha D

Obsah přiloženého DVD

Možnosti přiloženého DVD:

- Program ShapeGo5 spustíte souborem ShapeGo(.exe) v kořenovém adresáři DVD.
- 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 DVD v souboru *bcPraceRHamplova.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 ukázkové databáze vzorů a odpovídající příklady pro načtení programem ShapeGo jsou uložené v adresáři */Vzory a priklady*.

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