Studencka Pracownia Licencjackiego Projektu Programistycznego
II UWR 2010/2011

Wojciech Jedynak

GoStat

Program do wykonywania obliczeń statystycznych związanych z grą go

Dokumentacja programisty

Wrocław, 8 lipca 2011

Spis treści

1 Wprowadzenie

1.1 Cel dokumentacji

Celem niniejszego dokumentu jest takie przedstawienie struktury projektu GoStat, aby umożliwić jego modyfikacje oraz utrzymywanie programistom, którzy znają Haskella, ale nie należeli do początkowego zespołu. Wykaz użytych bibliotek powinien był pomocny, jeśli instalacja oprogramowania nie powiedzie się i konieczna będzie kompilacja programu ze źródeł. Dodatkowo życzeniem autora jest nakreślenie wykonanej pracy tak, aby zainteresowane osoby były w stanie (w razie potrzeby) na wykorzystanie opisanych tu rozwiązań w swoich projektach.

2 Organizacja projektu

2.1 Ogólny opis

Program został napisany niemal w całości w Haskellu [8]. Po uruchomieniu pliku `GoStat` wewnętrzny serwer HTTP nasłuchuje na porcie 8000, a komunikacja z użytkownikiem odbywa się za pomocą interfejsu WWW. Lista katalogów, w których znajduje się kolekcja plików SGF, zapisywana jest do pliku konfiguracyjnego, wstępnie przetworzone (*znormalizowane*) gry są przechowywane w bazie danych. Dialog z użytkownikiem może odbywać się w języku polskim bądź angielskim.

2.2 Konfiguracja programu

W programie potrzebujemy przechowywać dwie informacje: jak ma nazywać się plik bazy danych, której używa (chce używać) użytkownik i gdzie znajduje się jego kolekcja zapisów partii go, które chciałby analizować naszym programem. Do przechowywania ww. danych używamy bardzo prostego, autorskiego formatu. Funkcje związane z wczytywaniem, analizą leksykalną oraz zapisywaniem znajdują się w module *Configuration*.

2.2.1 Opis formatu pliku CONFIG

Format pliku jest opisywany przez poniższą gramatykę w postaci EBNF:

```
config ::= declaration*
declaration ::= db | dirs | '–' *anything*
db = dbserver ':' dbversion
dbversion = sqlite3 ';' path ';' | postgresql ';'
dirs = gamedirs ':' path ';'
```

Przykładowy plik konfiguracyjny:

```
-- new config
--dbserver:postgresql;
dbserver:sqlite3;/home/wojtek/db/games.db;
gamedirs:/home/wojtek/data/;
```

2.3 Baza danych

Program używa bazy danych Sqlite3 [4]. Programista nie musi zajmować się ręczną administracją bazy danych: służy do tego moduł *DB* w pliku `src/DB.hs` .

Używana jest jedna tabela o nazwie `go_stat_data` .

2.3.1 Tabela go_stat_data

Opis pól tabeli go_stat_data

Pole	Typ	NULL dozwolone?	Opis
id	PRIMARY KEY	nie	unikatowy identyfikator gry
winner	CHAR	nie	zwycięzca gry ('b' lub 'w')
moves	VARCHAR(700)	nie	znormalizowany przebieg rozgrywki
path	VARCHAR(255)	nie	bezwzględna ścieżka do gry
b_name	VARCHAR(30)	nie	pseudonim (nazwisko) czarnego
w_name	VARCHAR(30)	nie	pseudonim (nazwisko) białego
b_rank	VARCHAR(10)	tak	ranking czarnego
w_rank	VARCHAR(10)	tak	ranking białego

Jedyne pola, która nie są wymagane to pola b_rank i w_rank. Wynika to z tego, że na niektórych serwerach do gry w go nie jest wymagane podanie swojego orientacyjnego poziomu ani rankingu.

2.4 Struktura modułów

Lista modułów wchodzących w skład projektu:

2.4.1 Data.SGF.Types i Data.SGF.Parsing

```
type Move = (Int, Int)
type Moves = [Move]
type PlayerName = String
data Winner = Black | White
data Result = Unfinished | Draw | Win Winner PlayerName

parseSGF :: String -> Either String SGF
getResult :: SGF -> Result
isWithHandicap :: SGF -> Bool
getBlack, getWhite, getBlackRank, getWhiteRank, date :: SGF -> String
```

Moduły *Data.Sgf.Types* i *Data.SGF.Parsing* zawierają deklaracje typów służących do przechowywania informacji o danej partii (m. in. wynik, dane graczy, lista ruchów) oraz funkcje, które pozwalają dane te wyświetlać i analizować. Do analizy leksykalnej plików SGF wykorzystana jest biblioteka Parsec.

2.4.2 Transformations

W module *Transformations* (plik `src/Transformations.hs`) określono przekształcenia matematyczne i operację normalizacji ruchów.

```
normalizeMoves :: [Move] -> [Move]

isOnMainDiagonal :: Move -> Bool
isAboveMainDiagonal :: Move -> Bool
isBelowMainDiagonal :: Move -> Bool
isOnHorizontal :: Move -> Bool
```

```
isAboveHorizontal :: Move -> Bool
isBelowHorizontal :: Move -> Bool
horizontal :: Move -> Move
rotate90 :: Move -> Move
mainDiagonalMirror :: Move -> Move

transformIntoFirst :: Triangle -> (Move -> Move)
getTransformation :: Move -> (Move -> Move)

triangles :: [Triangle]
findTriangles :: Move -> [Triangle]
findTriangle :: Move -> Triangle
```

2.4.3 SgfBatching

Konwersja plików SGF do formatu, który będzie łatwo zapisać w bazie danych.

```
getSGFs :: [FilePath] -> IO [FilePath]
gameInfoToDB :: GameInfo -> (FilePath, Char, String, String, String, String, String)
sgfToGameInfo :: FilePath -> SGF -> Maybe GameInfo
```

2.4.4 Lang

Moduł *Lang* (w pliku `src/Lang.hs`) definiuje listę komunikatów, który wyświetlane są użytkownikowi. Komunikaty dostępne są w języku polskim i angielskim.

2.4.5 Configuration

Moduł *Configuration* odpowiada za obsługę pliku CONFIG.

```
data DbServer = Sqlite3 FilePath
data Configuration = Configuration { dbServer :: DbServer , gameDirs :: [FilePath] }

parseConfiguration :: String -> Either String Configuration
showConfiguration :: Configuration -> String
readConfig :: FilePath -> IO (Either String Configuration)
writeConfig :: Configuration -> FilePath -> IO ()
defaultConfig :: Configuration
```

2.4.6 Pages

Moduł *Pages* (plik `src/Pages.hs`) odpowiada za dynamiczne tworzenie stron WWW. Używana jest do tego biblioteka xhtml-3000 [13]. Do implementacji eleganckich efektów w JavaScript użyto bibliotek jQuery [5] oraz jQuery UI [6]. Interaktywne wyświetlanie przebiegu gry zapewnia komponent Eidogo [7].

```
– Strona główna
mainPage :: UrlBuilders -> Html

– Strona pokazująca listę możliwych ruchów
moveBrowser :: Int -> (Int, Int, Int) -> [(String, Int, Int, Int)] -> String -> UrlBuilders
-> Html
```

```
– Strona pokazująca listę gier
gameBrowserPage :: [(Int, FilePath, String, String, String, String, String)] -> (Int, Int,
Int) -> String -> UrlBuilders -> Html

– Strona pokazująca szczegóły wybranej gry
gameDetailsPage :: GameId -> SGF -> FilePath -> MovesSoFar -> UrlBuilders ->
Html

– Formularz konfiguracyjny
configForm :: Configuration -> UrlBuilders -> Html

– Strona pokazywana podczas przebudowy bazy danych
rebuildingPage :: Int -> Int -> Int -> UrlBuilders -> Html
```

2.4.7 DB

Moduł *DB* (plik `src/DB.hs`) odpowiada za zarządzanie bazą danych. Dialog z bazą danych jest możliwy dzięki HDBC i HDBC-sqlite3.

```
createDB :: GoStatM ()
deleteDB :: GoStatM ()
addFilesToDB :: GoStatM ()
queryCountDB :: GoStatM Int
queryStatsDB :: String -> GoStatM [(String, Int, Int, Int)]
queryCurrStatsDB :: String -> GoStatM (Int, Int, Int)
queryGamesListDB :: String -> Int -> GoStatM [(Int, FilePath, String, String, String,
String, String)]
queryFindGameById :: Int -> GoStatM (Maybe (String, FilePath))
```

2.4.8 Server

Moduł *Server* (plik `src/Server.hs`) udostępnia aplikacji Serwer HTTP. Korzystamy tutaj z biblioteki Happstack [12], dzięki czemu naszym zadaniem jest jedynie określenie jakie akcje mają być wykonane w odpowiedzi na dane zapytanie.

```
server :: GoStatM ()
router :: MVar Configuration -> MVar (Maybe Int) -> ServerPart Response
```

2.4.9 Main

Moduł *Main* (plik `src/Main.hs`) to punkt startowy aplikacji.

2.5 Pozostałe pliki

W katalogu `public` umieszczone zostały wszystkie pozostałe pliki konieczne do wyświetlenia strony WWW, tj. dodatkowe skrypty w JavaScript, arkusze stylów CSS i obrazki.

3 Kompilacja

W głȍwnym katalogu znajduje się plik `GoStat.cabal` . Pozwala on na wykorzystanie do kompilacji narzędzia Cabal [11], dzięki czemu:

1. Nie musimy sami dbać o to, aby zainstalowane zostały odpowiednie wersje pakietów z serwisu Hackage [10].
2. Cały projekt możemy skonfigurować i skompilować jednym poleceniem (`cabal configure && cabal build`).
3. Program można zainstalować jednym poleceniem (`cabal install`).
4. Jednym poleceniem możemy utworzyć archiwum zawierające wszystkie pliki wykorzystywane w projekcie (`cabal sdist`).
5. Wszystkie pliki tworzone podczas kompilacji (*.o, *.hi) są umieszczane w katalogu `dist` – pozwala to na utrzymanie ładu w strukturze katalogów należących do projektu.

Dodatkowo, aby zautomatyzować i uprościć pewne często wykonywane czynności, utworzono plik `Makefile` , który pozwala na wydanie następujących poleceń:

- `make` – Kompilacja projektu, zbudowanie programu wynikowego
- `make run` – Uruchomienie programu
- `make test` – Wykonanie wszystkich testów
- `make install` – Instalacja programu ze źródeł
- `make windows-release` – Utworzenie pliku `dist/GoStat-binary-windows.tar.gz`
- `make linux-release` – Utworzenie pliku `dist/GoStat-binary-linux.tar.gz`

Archiwa utworzone poprzez `make {windows, linux}-release` zawierają dokumentację, plik wykonywalny `GoStat` oraz wszystkie pliki dodatkowe, niezbędne do uruchomienia programu (pliki .css, .js, obrazki itp).

Dokumentacja tworzona jest przy pomocy programu `pdflatex` .

4 Testy

4.1 Biblioteki do testowania

Użyto następujących bibliotek do tworzenia i przeprowadzania testów:

1. HUnit
2. QuickCheck
3. test-framework

Poniżej krótko charakteryzujemy i pokazujemy przykłady użycia każdej z nich.

4.1.1 HUnit

Jest to narzędzie do tworzenia testów zwanych *testami jednostkowymi*, gdzie specyfikujemy działanie testowanego systemu poprzez wykazanie jak ma się zachowywać w danych, konkretnych sytuacjach.

Przykładowy test jednostkowy z projektu (plik `tests/transformations/tests.hs`):

```
test_rotate90_fixpoint :: Assertion
test_rotate90_fixpoint = (5,5) @?= rotate90 (5,5)
```

Operator @?= należy czytać tutaj jako "powinno być równe", funkcja `rotate90` to obrót o 90 stopni, cały test zaś specyfikuje, że punkt (5,5) [środkowy punkt planszy] powinien być niezmienniczy względem obrotu.

4.1.2 QuickCheck

QuickCheck to narzędzie pozwalający wyrażać specyfikacje w sposób ogólniejszy niż jest to możliwe przy pomocy testów jednostkowych. Możemy bowiem określać ogólne *własności* systemu; np. "dla *dowolnego* punktu p na planszy, czterokrotne wykonanie obrotu o 90 stopni daje w wyniku początkowy punkt p".

Powyższa własność zapisana w Haskellu wygląda następująco:

```
property_rotate90_4_times :: Move -> Bool
property_rotate90_4_times m = m == rotate90 (rotate90 (rotate90 (rotate90 m)))
```

Jak w praktyce sprawdzane są własności takie jak ta? Zwróćmy uwagę, że nie możemy po prostu sprawdzić wszystkich możliwości, gdyż punktów na płaszczyźnie jest nieskończenie wiele! W systemach dowodzenia twierdzeń, takich jak Coq [14], czy Agda [15] możemy taką własność *udowodnić* (np. poprzez indukcję). Wymaga to jednak zazwyczaj dość sporo czasu i umiejętności.

Twórcy QuickChecka zdecydowali się na bardzo praktyczne podejście: podane własności są testowane na danych *losowych* i każdy test jest powtarzany wielokrotnie, np. 100 razy. Aby sprawdzić daną własność należy (w programie napisanym w Haskellu) wywołać funkcję quickCheck.

Przykładowe wywołanie `quickCheck property_rotate90_4_times` da nam odpowiedź:

```
+++ OK, passed 100 tests.
```

Jeśli podamy własność, która nie zachodzi to (o ile będziemy mieli szczęście) otrzymamy:

```
*** Failed! Falsifiable (after 1 test and 1 shrink):
(0,0)
```

Oznaczałoby to, że dla wartości (0,0) własność nie jest spełniona.

4.1.3 test-framework

Test-framework to biblioteka, która w wygodny sposób pozwala nam łączyć zalety testów jednostkowych i sprawdzania własności. Umożliwia ona grupowanie (katalogowanie) testów obu rodzajów, uruchamianie całych grup jednocześnie oraz automatycznie generowanie raportów z wykonanych testów. Przykładowy raport znajduje się w podrozdziale *Uruchamianie testów i analiza wyników* poniżej, w następnym podrozdziale omówimy na przykładzie projektu jak używać test-framework w praktyce.

4.2 Struktura testów

Hierarchia plików zawierających testy jest równoległa do hierarchii plików zawierających definicje modułów, przy czym testy znajdują się w katalogu `test` , właściwie moduły zaś w katalogu `src` . Przykładowo, testy modułu *Transformations* (który jest zdefiniowany w pliku `src/Transformations.hs`) znajdują się w pliku `test/Transformations/Tests.hs`

Wszystkie testy są grupowane razem w pliku `test/Tests.hs` , który (w skróconej wersji) przedstawiamy poniżej:

```
module Main where

import Test.Framework (defaultMain, Test)

import Data.SGF.Types.Tests
import Data.SGF.Parsing.Tests
import Transformations.Tests

main :: IO ()
main = defaultMain tests

tests :: [Test]
tests = [ data_sgf_types_tests, data_sgf_parsing_tests, transformations_tests ]
```

4.3 Uruchamianie testów i analiza wyników

Aby wykonać cały pakiet dostępnych testów należy z poziomu katalogu głównego wydać polecenie `make test` . Spowoduje to kompilacje całego projektu i utworzenie, a następnie uruchomienie, pliku `GoStatTests` .

Następnie na ekran wypisywane będą na bieżąco wyniki wykonywanych testów, na samym końcu zaś pokazane zostanie podsumowanie. W ramce poniżej widzimy fragment przykładowego raportu. Końcowa tabela pokazuje, że wszystkie testy zostały wykonane pomyślnie.

```
...
normalizeMoves diagonal symmetry handling: [OK, passed 5000 tests]
normalizeMoves horizont symmetry handling: [OK, passed 5000 tests]
normalizeMoves start:55 handling: [OK, passed 5000 tests]
```

	Properties	Test Cases	Total
Passed	26	14	40
Failed	0	0	0
Total	26	14	40

5 Wykorzystane narzędzia pomocnicze

W niniejszej sekcji wymieniono i pokrótce opisano najważniejsze narzędzia, które pozwoliły ukończyć projekt, a nie są bezpośrednio związane z Haskellem.

5.1 Git i portal github

Git [1] to rozproszony system kontroli wersji, stworzony przez Linusa Torvaldsa.

Github [2] to portal, który pozwala na przechowywanie kodu źródłowego (ogólnie: repozytoriów kodu zarządzanych przez git) i udostępnienie go innym programistom.

Poprzez użycie tych zasobów rozwiązano kwestię składowania projektu i możliwe było swobodne eksperymentowanie: nietrafione zmiany można było wycofać jednym poleceniem.

5.2 Latex

System LaTeXpozwolił na utworzenie dokumentacji.

Literatura

[1] *Git – narzędzie do kontroli wersji*
`http://git-scm.com/`

[2] *Portal github.com*
`https://github.com/`

[3] *Repozytorium projektu GoStat w portalu github*
`https://github.com/wjzz`

[4] *SQLite3 – lekka baza danych*
`http://www.sqlite.org/`

[5] *jQuery – biblioteka dla JavaScriptu*
`http://jquery.com/`

[6] *jQuery UI – biblioteka komponentów dla JavaScriptu*
`http://jqueryui.com/`

[7] *Eidogo – interaktywna plansza*
`http://eidogo.com/source`

[8] *Haskell – strona główna*
`http://www.haskell.org/`

[9] *Glasgow Haskell Compiler*
`http://www.haskell.org/ghc/`

[10] *Hackage – kolekcja pakietów Haskellowych*
`http://hackage.haskell.org/`

[11] *Cabal – narzędzie do tworzenia i instalowania pakietów Haskellowych*
`http://www.haskell.org/cabal/`

[12] *Happstack – serwer HTTP dla Haskella*
`http://happstack.com/index.html`

[13] *Biblioteka xhtml dla Haskella*
`http://hackage.haskell.org/package/xhtml-3000.2.0.1`

[14] *Coq – system dowodzenia twierdzeń oparty na OCamlu*
`http://coq.inria.fr/`

[15] *Agda – system dowodzenia twierdzeń oparty na Haskellu*
`http://wiki.portal.chalmers.se/agda/pmwiki.php`